# Vulkan® 1.1.295 - A Specification (with all registered extensions)

The Khronos® Vulkan Working Group

Version 1.1.295, 2024-08-30 09:13:43Z: from git branch: github-main commit: 8e435deb15124bbf9a7e3340e6cc7975999ed5c5

# Table of Contents


1. Preamble . . . 1
2. Introduction . . . 3
   2.1. Document Conventions . . . 3
3. Fundamentals . . . 6
   3.1. Host and Device Environment . . . 6
   3.2. Execution Model . . . 6
   3.3. Object Model . . . 8
   3.4. Application Binary Interface . . . 12
   3.5. Command Syntax and Duration . . . 13
   3.6. Threading Behavior . . . 16
   3.7. Valid Usage . . . 35
   3.8. `VkResult` Return Codes . . . 42
   3.9. Numeric Representation and Computation . . . 47
   3.10. Fixed-Point Data Conversions . . . 49
   3.11. String Representation . . . 51
   3.12. Common Object Types . . . 51
   3.13. API Name Aliases . . . 101
4. Initialization . . . 102
   4.1. Command Function Pointers . . . 102
   4.2. Instances . . . 105
5. Devices and Queues . . . 122
   5.1. Physical Devices . . . 122
   5.2. Devices . . . 159
   5.3. Queues . . . 182
6. Command Buffers . . . 193
   6.1. Command Buffer Lifecycle . . . 193
   6.2. Command Pools . . . 195
   6.3. Command Buffer Allocation and Management . . . 202
   6.4. Command Buffer Recording . . . 207
   6.5. Command Buffer Submission . . . 222
   6.6. Queue Forward Progress . . . 249
   6.7. Secondary Command Buffer Execution . . . 249
   6.8. Nested Command Buffers . . . 258
   6.9. Command Buffer Device Mask . . . 259
7. Synchronization and Cache Control . . . 262
   7.1. Execution and Memory Dependencies . . . 262
   7.2. Implicit Synchronization Guarantees . . . 298
   7.3. Fences . . . 300

7.4. Semaphores . . . 326
7.5. Events . . . 360
7.6. Pipeline Barriers . . . 387
7.7. Memory Barriers . . . 398
7.8. Wait Idle Operations . . . 457
7.9. Host Write Ordering Guarantees . . . 459
7.10. Synchronization and Multiple Physical Devices . . . 459
7.11. Calibrated Timestamps . . . 460
8. Render Pass . . . 464
8.1. Render Pass Objects . . . 491
8.2. Render Pass Creation . . . 493
8.3. Render Pass Compatibility . . . 562
8.4. Framebuffers . . . 562
8.5. Render Pass Load Operations . . . 575
8.6. Render Pass Store Operations . . . 576
8.7. Render Pass Multisample Resolve Operations . . . 578
8.8. Render Pass Commands . . . 580
8.9. Render Pass Creation Feedback . . . 612
8.10. Common Render Pass Data Races (Informative) . . . 616
9. Shaders . . . 618
9.1. Shader Objects . . . 618
9.2. Shader Modules . . . 641
9.3. Shader Module Identifiers . . . 646
9.4. Binding Shaders . . . 649
9.5. Shader Execution . . . 650
9.6. Shader Memory Access Ordering . . . 651
9.7. Shader Inputs and Outputs . . . 651
9.8. Task Shaders . . . 652
9.9. Mesh Shaders . . . 652
9.10. Cluster Culling Shaders . . . 653
9.11. Vertex Shaders . . . 654
9.12. Tessellation Control Shaders . . . 654
9.13. Tessellation Evaluation Shaders . . . 656
9.14. Geometry Shaders . . . 656
9.15. Fragment Shaders . . . 657
9.16. Compute Shaders . . . 657
9.17. Ray Generation Shaders . . . 657
9.18. Intersection Shaders . . . 657
9.19. Any-Hit Shaders . . . 658
9.20. Closest Hit Shaders . . . 658
9.21. Miss Shaders . . . 658

9.22. Callable Shaders . . . 659
9.23. Interpolation Decorations . . . 659
9.24. Static Use . . . 660
9.25. Scope . . . 660
9.26. Group Operations . . . 665
9.27. Quad Group Operations . . . 666
9.28. Derivative Operations . . . 667
9.29. Helper Invocations . . . 669
9.30. Cooperative Matrices . . . 669
9.31. Validation Cache . . . 676
9.32. CUDA Modules . . . 683
10. Pipelines . . . 691
10.1. Multiple Pipeline Creation . . . 693
10.2. Compute Pipelines . . . 693
10.3. Graphics Pipelines . . . 718
10.4. Ray Tracing Pipelines . . . 793
10.5. Pipeline Destruction . . . 821
10.6. Pipeline Derivatives . . . 822
10.7. Pipeline Cache . . . 823
10.8. Pipeline Binaries . . . 831
10.9. Specialization Constants . . . 844
10.10. Pipeline Libraries . . . 847
10.11. Pipeline Binding . . . 849
10.12. Dynamic State . . . 854
10.13. Pipeline Properties and Shader Information . . . 855
10.14. Pipeline Compiler Control . . . 869
10.15. Pipeline Creation Feedback . . . 870
11. Memory Allocation . . . 874
11.1. Host Memory . . . 874
11.2. Device Memory . . . 881
12. Resource Creation . . . 991
12.1. Buffers . . . 991
12.2. Buffer Views . . . 1009
12.3. Images . . . 1015
12.4. Image Layouts . . . 1071
12.5. Image Views . . . 1077
12.6. Acceleration Structures . . . 1106
12.7. Micromaps . . . 1138
12.8. Resource Memory Association . . . 1146
12.9. Resource Sharing Mode . . . 1185
12.10. Memory Aliasing . . . 1187

12.11. Buffer Collections . . . 1189
13. Samplers . . . 1205
13.1. Sampler $Y'C_BC_R$ Conversion . . . 1220
14. Resource Descriptors . . . 1234
14.1. Descriptor Types . . . 1234
14.2. Descriptor Sets . . . 1240
14.3. Physical Storage Buffer Access . . . 1343
14.4. Descriptor Buffers . . . 1347
15. Shader Interfaces . . . 1385
15.1. Shader Input and Output Interfaces . . . 1385
15.2. Vertex Input Interface . . . 1389
15.3. Fragment Output Interface . . . 1389
15.4. Legacy Dithering . . . 1393
15.5. Fragment Tile Image Interface . . . 1394
15.6. Fragment Input Attachment Interface . . . 1394
15.7. Ray Tracing Pipeline Interface . . . 1399
15.8. Shader Resource Interface . . . 1400
15.9. Built-In Variables . . . 1409
16. Image Operations . . . 1463
16.1. Image Operations Overview . . . 1463
16.2. Conversion Formulas . . . 1468
16.3. Texel Input Operations . . . 1469
16.4. Texel Output Operations . . . 1488
16.5. Normalized Texel Coordinate Operations . . . 1489
16.6. Unnormalized Texel Coordinate Operations . . . 1496
16.7. Integer Texel Coordinate Operations . . . 1498
16.8. Image Sample Operations . . . 1499
16.9. Texel Footprint Evaluation . . . 1505
16.10. Weight Image Sampling . . . 1508
16.11. Block Matching . . . 1513
16.12. Box Filter Sampling . . . 1517
16.13. Image Operation Steps . . . 1520
16.14. Image Query Instructions . . . 1520
17. Fragment Density Map Operations . . . 1522
17.1. Fragment Density Map Operations Overview . . . 1522
17.2. Fetch Density Value . . . 1522
17.3. Fragment Area Conversion . . . 1523
18. Queries . . . 1525
18.1. Query Pools . . . 1525
18.2. Query Operation . . . 1533
18.3. Occlusion Queries . . . 1561

18.4. Pipeline Statistics Queries . . . 1562
18.5. Timestamp Queries . . . 1565
18.6. Performance Queries . . . 1572
18.7. Transform Feedback Queries . . . 1575
18.8. Primitives Generated Queries . . . 1576
18.9. Mesh Shader Queries . . . 1576
18.10. Intel Performance Queries . . . 1577
18.11. Result Status Queries . . . 1591
18.12. Video Encode Feedback Queries . . . 1591
19. Clear Commands . . . 1594
19.1. Clearing Images Outside a Render Pass Instance . . . 1594
19.2. Clearing Images Inside a Render Pass Instance . . . 1600
19.3. Clear Values . . . 1604
19.4. Filling Buffers . . . 1606
19.5. Updating Buffers . . . 1608
20. Copy Commands . . . 1611
20.1. Copying Data Between Buffers . . . 1611
20.2. Copying Data Between Images . . . 1617
20.3. Copying Data Between Buffers and Images . . . 1637
20.4. Indirect Copies . . . 1693
20.5. Image Copies With Scaling . . . 1699
20.6. Resolving Multisample Images . . . 1718
20.7. Buffer Markers . . . 1730
21. Drawing Commands . . . 1736
21.1. Primitive Topologies . . . 1739
21.2. Primitive Order . . . 1750
21.3. Programmable Primitive Shading . . . 1751
21.4. Conditional Rendering . . . 2166
21.5. Programmable Mesh Shading . . . 2170
21.6. Programmable Cluster Culling Shading . . . 2432
22. Fixed-Function Vertex Processing . . . 2519
22.1. Vertex Attributes . . . 2519
22.2. Vertex Input Description . . . 2523
22.3. Vertex Attribute Divisor in Instanced Rendering . . . 2536
22.4. Vertex Input Address Calculation . . . 2538
23. Tessellation . . . 2540
23.1. Tessellator . . . 2540
23.2. Tessellator Patch Discard . . . 2543
23.3. Tessellator Spacing . . . 2544
23.4. Tessellation Primitive Ordering . . . 2544
23.5. Tessellator Vertex Winding Order . . . 2545

23.6. Triangle Tessellation . . . 2545
23.7. Quad Tessellation . . . 2547
23.8. Isoline Tessellation . . . 2549
23.9. Tessellation Point Mode . . . 2549
23.10. Tessellation Pipeline State . . . 2549
24. Geometry Shading . . . 2554
24.1. Geometry Shader Input Primitives . . . 2554
24.2. Geometry Shader Output Primitives . . . 2555
24.3. Multiple Invocations of Geometry Shaders . . . 2555
24.4. Geometry Shader Primitive Ordering . . . 2555
24.5. Geometry Shader Passthrough . . . 2555
25. Mesh Shading . . . 2558
25.1. Task Shader Input . . . 2558
25.2. Task Shader Output . . . 2558
25.3. Mesh Generation . . . 2558
25.4. Mesh Shader Input . . . 2558
25.5. Mesh Shader Output . . . 2559
25.6. Mesh Shader Per-View Outputs . . . 2560
25.7. Mesh Shader Primitive Ordering . . . 2560
26. Cluster Culling Shading . . . 2561
26.1. Cluster Culling Shader Input . . . 2561
26.2. Cluster Culling Shader Output . . . 2561
26.3. Cluster Culling Shader Cluster Ordering . . . 2561
26.4. Cluster Culling Shader Primitive Ordering . . . 2562
27. Fixed-Function Vertex Post-Processing . . . 2563
27.1. Transform Feedback . . . 2563
27.2. Viewport Swizzle . . . 2572
27.3. Flat Shading . . . 2576
27.4. Primitive Clipping . . . 2579
27.5. Clipping Shader Outputs . . . 2586
27.6. Controlling Viewport W Scaling . . . 2587
27.7. Coordinate Transformations . . . 2591
27.8. Render Pass Transform . . . 2591
27.9. Controlling the Viewport . . . 2592
28. Rasterization . . . 2603
28.1. Discarding Primitives Before Rasterization . . . 2608
28.2. Controlling the Vertex Stream Used for Rasterization . . . 2609
28.3. Rasterization Order . . . 2613
28.4. Multisampling . . . 2614
28.5. Custom Sample Locations . . . 2620
28.6. Fragment Shading Rates . . . 2625

28.7. Shading Rate Image . . . 2639
28.8. Sample Shading . . . 2654
28.9. Barycentric Interpolation . . . 2655
28.10. Points . . . 2657
28.11. Line Segments . . . 2658
28.12. Polygons . . . 2671
29. Fragment Operations . . . 2694
29.1. Discard Rectangles Test . . . 2696
29.2. Scissor Test . . . 2702
29.3. Exclusive Scissor Test . . . 2704
29.4. Sample Mask Test . . . 2709
29.5. Fragment Shading . . . 2711
29.6. Multisample Coverage . . . 2715
29.7. Depth and Stencil Operations . . . 2718
29.8. Depth Bounds Test . . . 2722
29.9. Stencil Test . . . 2725
29.10. Depth Test . . . 2734
29.11. Representative Fragment Test . . . 2739
29.12. Sample Counting . . . 2741
29.13. Fragment Coverage to Color . . . 2741
29.14. Coverage Reduction . . . 2745
30. The Framebuffer . . . 2758
30.1. Blending . . . 2758
30.2. Logical Operations . . . 2787
30.3. Color Write Mask . . . 2792
30.4. Color Write Enable . . . 2793
30.5. Framebuffer Query Instructions . . . 2795
31. Dispatching Commands . . . 2798
31.1. Dispatch Command for CUDA PTX Kernels . . . 2834
32. Device-Generated Commands . . . 2840
32.1. Indirect Commands Layout . . . 2840
32.2. Indirect Commands Generation and Execution . . . 2857
33. Sparse Resources . . . 2913
33.1. Sparse Resource Features . . . 2913
33.2. Sparse Buffers and Fully-Resident Images . . . 2914
33.3. Sparse Partially-Resident Buffers . . . 2915
33.4. Sparse Partially-Resident Images . . . 2915
33.5. Sparse Memory Aliasing . . . 2923
33.6. Sparse Resource Implementation Guidelines (Informative) . . . 2924
33.7. Sparse Resource API . . . 2925
34. Window System Integration (WSI) . . . 2953

34.1. WSI Platform . . . 2953
34.2. WSI Surface . . . 2953
34.3. Presenting Directly to Display Devices . . . 2983
34.4. Querying for WSI Support . . . 3014
34.5. Surface Queries . . . 3021
34.6. Full Screen Exclusive Control . . . 3054
34.7. Device Group Queries . . . 3056
34.8. Display Timing Queries . . . 3062
34.9. Present Wait . . . 3068
34.10. WSI Swapchain . . . 3068
34.11. HDR Metadata . . . 3122
34.12. Lag Control . . . 3124
34.13. Present Barrier . . . 3138
35. Deferred Host Operations . . . 3140
35.1. Requesting Deferral . . . 3140
35.2. Deferred Host Operations API . . . 3141
36. Private Data . . . 3147
37. Acceleration Structures . . . 3153
37.1. Acceleration Structures . . . 3153
37.2. Host Acceleration Structure Operations . . . 3220
38. Micromap . . . 3236
38.1. Micromaps . . . 3236
38.2. Host Micromap Operations . . . 3256
39. Ray Traversal . . . 3269
39.1. Ray Intersection Candidate Determination . . . 3269
39.2. Ray Intersection Culling . . . 3272
39.3. Ray Intersection Confirmation . . . 3276
39.4. Ray Closest Hit Determination . . . 3278
39.5. Ray Result Determination . . . 3278
40. Ray Tracing . . . 3279
40.1. Shader Call Instructions . . . 3279
40.2. Ray Tracing Commands . . . 3281
40.3. Shader Binding Table . . . 3332
40.4. Ray Tracing Pipeline Stack . . . 3335
40.5. Ray Tracing Capture Replay . . . 3336
40.6. Ray Tracing Validation . . . 3336
41. Memory Decompression . . . 3337
42. Video Coding . . . 3343
42.1. Video Picture Resources . . . 3343
42.2. Decoded Picture Buffer . . . 3345
42.3. Video Profiles . . . 3347

42.4. Video Capabilities . . . 3357
42.5. Video Sessions . . . 3366
42.6. Video Profile Compatibility . . . 3378
42.7. Video Session Parameters . . . 3379
42.8. Video Coding Scope . . . 3400
42.9. Video Coding Control . . . 3410
42.10. Inline Queries . . . 3413
42.11. Video Decode Operations . . . 3415
42.12. H.264 Decode Operations . . . 3432
42.13. H.265 Decode Operations . . . 3445
42.14. AV1 Decode Operations . . . 3459
42.15. Video Encode Operations . . . 3470
42.16. Video Encode Rate Control . . . 3499
42.17. H.264 Encode Operations . . . 3508
42.18. H.265 Encode Operations . . . 3541
43. Optical Flow . . . 3581
43.1. Optical Flow Queues . . . 3581
43.2. Optical Flow Image Formats . . . 3581
43.3. Optical Flow Session . . . 3584
44. Execution Graphs . . . 3597
44.1. Pipeline Creation . . . 3597
44.2. Initializing Scratch Memory . . . 3605
44.3. Dispatching a Graph . . . 3608
44.4. Shader Enqueue . . . 3640
45. Extending Vulkan . . . 3641
45.1. Instance and Device Functionality . . . 3641
45.2. Core Versions . . . 3641
45.3. Layers . . . 3644
45.4. Extensions . . . 3648
45.5. Extension Dependencies . . . 3652
45.6. Compatibility Guarantees (Informative) . . . 3653
46. Features . . . 3658
46.1. Feature Requirements . . . 3819
47. Limits . . . 3827
47.1. Limit Requirements . . . 3921
47.2. Additional Multisampling Capabilities . . . 3948
48. Formats . . . 3950
48.1. Format Definition . . . 3950
48.2. Format Properties . . . 4002
48.3. Required Format Support . . . 4023
49. Additional Capabilities . . . 4045

49.1. Additional Image Capabilities . . . 4045
49.2. Additional Buffer Capabilities . . . 4072
49.3. Optional Semaphore Capabilities . . . 4075
49.4. Optional Fence Capabilities . . . 4081
49.5. Timestamp Calibration Capabilities . . . 4087
50. Debugging . . . 4089
50.1. Debug Utilities . . . 4092
50.2. Debug Markers . . . 4116
50.3. Debug Report Callbacks . . . 4124
50.4. Device Loss Debugging . . . 4132
50.5. Active Tooling Information . . . 4144
50.6. Frame Boundary . . . 4148
Appendix A: Vulkan Environment for SPIR-V . . . 4151
Versions and Formats . . . 4151
Capabilities . . . 4151
Validation Rules Within a Module . . . 4166
Precision and Operation of SPIR-V Instructions . . . 4198
Signedness of SPIR-V Image Accesses . . . 4205
Image Format and Type Matching . . . 4205
Compatibility Between SPIR-V Image Dimensions and Vulkan ImageView Types . . . 4207
Compatibility Between SPIR-V Image Formats and Vulkan Formats . . . 4208
Ray Query Precision and Operation . . . 4209
Appendix B: Memory Model . . . 4210
Agent . . . 4210
Memory Location . . . 4210
Allocation . . . 4210
Memory Operation . . . 4211
Reference . . . 4211
Program-Order . . . 4211
Shader Call Related . . . 4212
Shader Call Order . . . 4212
Scope . . . 4212
Atomic Operation . . . 4213
Scoped Modification Order . . . 4213
Memory Semantics . . . 4214
Release Sequence . . . 4215
Synchronizes-With . . . 4216
System-Synchronizes-With . . . 4218
Private vs. Non-Private . . . 4218
Inter-Thread-Happens-Before . . . 4218
Happens-Before . . . 4219

Availability and Visibility . . . 4219
Availability, Visibility, and Domain Operations . . . 4221
Availability and Visibility Semantics . . . 4222
Per-Instruction Availability and Visibility Semantics . . . 4222
Location-Ordered . . . 4223
Data Race . . . 4224
Visible-To . . . 4224
Acyclicity . . . 4225
Shader I/O . . . 4225
Deallocation . . . 4226
Descriptions (Informative) . . . 4226
Tessellation Output Ordering . . . 4227
Cooperative Matrix Memory Access . . . 4227
Appendix C: Compressed Image Formats . . . 4228
Block-Compressed Image Formats . . . 4229
ETC Compressed Image Formats . . . 4230
ASTC Compressed Image Formats . . . 4231
PVRTC Compressed Image Formats . . . 4233
Appendix D: Core Revisions (Informative) . . . 4234
Vulkan Version 1.1 . . . 4234
Vulkan Version 1.0 . . . 4245
Appendix E: Layers & Extensions (Informative) . . . 4259
Extension Dependencies . . . 4259
Extension Interactions . . . 4259
List of Current Extensions . . . 4260
List of Provisional Extensions . . . 5062
List of Deprecated Extensions . . . 5072
Appendix F: API Boilerplate . . . 5213
Vulkan Header Files . . . 5213
Window System-Specific Header Control (Informative) . . . 5217
Provisional Extension Header Control (Informative) . . . 5219
Video Std Headers . . . 5219
Appendix G: Invariance . . . 5221
Repeatability . . . 5221
Multi-Pass Algorithms . . . 5221
Invariance Rules . . . 5221
Tessellation Invariance . . . 5223
Appendix H: Lexicon . . . 5225
Glossary . . . 5225
Common Abbreviations . . . 5254
Video-Specific Abbreviations . . . 5255

Prefixes . . . . . 5257
Appendix I: Credits (Informative) . . . . . 5259
Working Group Contributors to Vulkan . . . . . 5259
Other Credits . . . . . 5267

# Chapter 1. Preamble

Copyright 2014-2024 The Khronos Group Inc.

This Specification is protected by copyright laws and contains material proprietary to Khronos. Except as described by these terms, it or any components may not be reproduced, republished, distributed, transmitted, displayed, broadcast or otherwise exploited in any manner without the express prior written permission of Khronos.

Khronos grants a conditional copyright license to use and reproduce the unmodified Specification for any purpose, without fee or royalty, EXCEPT no licenses to any patent, trademark or other intellectual property rights are granted under these terms.

Khronos makes no, and expressly disclaims any, representations or warranties, express or implied, regarding this Specification, including, without limitation: merchantability, fitness for a particular purpose, non-infringement of any intellectual property, correctness, accuracy, completeness, timeliness, and reliability. Under no circumstances will Khronos, or any of its Promoters, Contributors or Members, or their respective partners, officers, directors, employees, agents or representatives be liable for any damages, whether direct, indirect, special or consequential damages for lost revenues, lost profits, or otherwise, arising from or in connection with these materials.

This document contains extensions which are not ratified by Khronos, and as such is not a ratified Specification, though it contains text from (and is a superset of) the ratified Vulkan Specification. The ratified versions of the Vulkan Specification can be found at https://registry.khronos.org/vulkan/specs/1.3/html/vkspec.html (core only) and https://registry.khronos.org/vulkan/specs/1.3-khr-extensions/html/vkspec.html (core with all ratified extensions) .

This Specification contains substantially unmodified functionality from, and is a successor to, Khronos specifications including OpenGL, OpenGL ES and OpenCL.

The Khronos Intellectual Property Rights Policy defines the terms 'Scope', 'Compliant Portion', and 'Necessary Patent Claims'.

Some parts of this Specification are purely informative and so are EXCLUDED the Scope of this Specification. The Document Conventions section of the Introduction defines how these parts of the Specification are identified.

Where this Specification uses technical terminology, defined in the Glossary or otherwise, that refer to enabling technologies that are not expressly set forth in this Specification, those enabling technologies are EXCLUDED from the Scope of this Specification. For clarity, enabling technologies not disclosed with particularity in this Specification (e.g. semiconductor manufacturing technology, hardware architecture, processor architecture or microarchitecture, memory architecture, compiler technology, object oriented technology, basic operating system technology, compression technology, algorithms, and so on) are NOT to be considered expressly set forth; only those application program interfaces and data structures disclosed with particularity are included in the Scope of this Specification.

For purposes of the Khronos Intellectual Property Rights Policy as it relates to the definition of

Necessary Patent Claims, all recommended or optional features, behaviors and functionality set forth in this Specification, if implemented, are considered to be included as Compliant Portions.

Where this Specification identifies specific sections of external references, only those specifically identified sections define normative functionality. The Khronos Intellectual Property Rights Policy excludes external references to materials and associated enabling technology not created by Khronos from the Scope of this Specification, and any licenses that may be required to implement such referenced materials and associated technologies must be obtained separately and may involve royalty payments.

Khronos and Vulkan are registered trademarks, and SPIR-V is a trademark of The Khronos Group Inc. OpenCL is a trademark of Apple Inc., used under license by Khronos. OpenGL is a registered trademark and the OpenGL ES logo is a trademark of Hewlett Packard Enterprise, used under license by Khronos. ASTC is a trademark of ARM Holdings PLC. All other product names, trademarks, and/or company names are used solely for identification and belong to their respective owners.

# Chapter 2. Introduction

This document, referred to as the “Vulkan Specification” or just the “Specification” hereafter, describes the Vulkan Application Programming Interface (API). Vulkan is a C99 API designed for explicit control of low-level graphics and compute functionality.

The canonical version of the Specification is available in the official Vulkan Registry (https://registry.khronos.org/vulkan/). The source files used to generate the Vulkan specification are stored in the Vulkan Documentation Repository (https://github.com/KhronosGroup/Vulkan-Docs).

The source repository additionally has a public issue tracker and allows the submission of pull requests that improve the specification.

## 2.1. Document Conventions

The Vulkan specification is intended for use by both implementors of the API and application developers seeking to make use of the API, forming a contract between these parties. Specification text may address either party; typically the intended audience can be inferred from context, though some sections are defined to address only one of these parties. (For example, Valid Usage sections only address application developers). Any requirements, prohibitions, recommendations or options defined by normative terminology are imposed only on the audience of that text.

**NOTE**

Structure and enumerated types defined in extensions that were promoted to core in a later version of Vulkan are now defined in terms of the equivalent Vulkan core interfaces. This affects the Vulkan Specification, the Vulkan header files, and the corresponding XML Registry.

### 2.1.1. Ratification

*Ratification* of a Vulkan core version or extension is a status conferred by vote of the Khronos Board of Promoters, bringing that core version or extension under the umbrella of the Khronos IP Policy.

All Vulkan core versions and `KHR` extensions (including provisional specifications) are ratified, as are some multi-vendor `EXT` extensions. Ratification status of extensions is described in the Layers & Extensions (Informative) appendix.

**NOTE**

Ratification status is primarily of interest to IHVs developing GPU hardware and Vulkan implementations

For developers, ratification does not necessarily mean that an extension is “better”; has a more stable API; or is more widely supported than alternative ways of achieving that functionality.

Interactions between ratified and non-ratified extensions are not themselves ratified.

### 2.1.2. Informative Language

Some language in the specification is purely informative, intended to give background or suggestions to implementors or developers.

If an entire chapter or section contains only informative language, its title will be suffixed with "(Informative)".

All NOTEs are implicitly informative.

### 2.1.3. Normative Terminology

Within this specification, the key words **must**, **required**, **should**, **recommended**, **may**, and **optional** are to be interpreted as described in RFC 2119 - Key words for use in RFCs to Indicate Requirement Levels (https://www.ietf.org/rfc/rfc2119.txt). The additional key word **optionally** is an alternate form of **optional**, for use where grammatically appropriate.

These key words are highlighted in the specification for clarity. In text addressing application developers, their use expresses requirements that apply to application behavior. In text addressing implementors, their use expresses requirements that apply to implementations.

In text addressing application developers, the additional key words **can** and **cannot** are to be interpreted as describing the capabilities of an application, as follows:

**can**

This word means that the application is able to perform the action described.

**cannot**

This word means that the API and/or the execution environment provide no mechanism through which the application can express or accomplish the action described.

These key words are never used in text addressing implementors.

| | |
|---|---|
| **NOTE** | There is an important distinction between **cannot** and **must not**, as used in this Specification. **Cannot** means something the application literally is unable to express or accomplish through the API, while **must not** means something that the application is capable of expressing through the API, but that the consequences of doing so are undefined and potentially unrecoverable for the implementation (see Valid Usage). |

Unless otherwise noted in the section heading, all sections and appendices in this document are normative.

### 2.1.4. Technical Terminology

The Vulkan Specification makes use of common engineering and graphics terms such as **Pipeline**, **Shader**, and **Host** to identify and describe Vulkan API constructs and their attributes, states, and behaviors. The Glossary defines the basic meanings of these terms in the context of the Specification. The Specification text provides fuller definitions of the terms and may elaborate,

extend, or clarify the Glossary definitions. When a term defined in the Glossary is used in normative language within the Specification, the definitions within the Specification govern and supersede any meanings the terms may have in other technical contexts (i.e. outside the Specification).

### 2.1.5. Normative References

References to external documents are considered normative references if the Specification uses any of the normative terms defined in Normative Terminology to refer to them or their requirements, either as a whole or in part.

The following documents are referenced by normative sections of the specification:

IEEE. August, 2008. *IEEE Standard for Floating-Point Arithmetic*. IEEE Std 754-2008. https://dx.doi.org/10.1109/IEEESTD.2008.4610935 .

Andrew Garrard. *Khronos Data Format Specification, version 1.3*. https://registry.khronos.org/DataFormat/specs/1.3/dataformat.1.3.html .

John Kessenich. *SPIR-V Extended Instructions for GLSL, Version 1.00* (February 10, 2016). https://registry.khronos.org/spir-v/ .

John Kessenich, Boaz Ouriel, and Raun Krisch. *SPIR-V Specification, Version 1.5, Revision 3, Unified* (April 24, 2020). https://registry.khronos.org/spir-v/ .

ITU-T. *H.264 Advanced Video Coding for Generic Audiovisual Services* (August, 2021). https://www.itu.int/rec/T-REC-H.264-202108-I/ .

ITU-T. *H.265 High Efficiency Video Coding* (August, 2021). https://www.itu.int/rec/T-REC-H.265-202108-S/ .

Alliance for Open Media. *AV1 Bitstream & Decoding Process Specification* (January 8, 2019). https://aomediacodec.github.io/av1-spec/av1-spec.pdf .

Jon Leech. *The Khronos Vulkan API Registry* (February 26, 2023). https://registry.khronos.org/vulkan/specs/1.3/registry.html .

Jon Leech and Tobias Hector. *Vulkan Documentation and Extensions: Procedures and Conventions* (February 26, 2023). https://registry.khronos.org/vulkan/specs/1.3/styleguide.html .

*Architecture of the Vulkan Loader Interfaces* (October, 2021). https://github.com/KhronosGroup/Vulkan-Loader/blob/main/docs/LoaderInterfaceArchitecture.md .

# Chapter 3. Fundamentals

This chapter introduces fundamental concepts including the Vulkan architecture and execution model, API syntax, queues, pipeline configurations, numeric representation, state and state queries, and the different types of objects and shaders. It provides a framework for interpreting more specific descriptions of commands and behavior in the remainder of the Specification.

## 3.1. Host and Device Environment

The Vulkan Specification assumes and requires: the following properties of the host environment with respect to Vulkan implementations:

- The host **must** have runtime support for 8, 16, 32 and 64-bit signed and unsigned twos-complement integers, all addressable at the granularity of their size in bytes.
- The host **must** have runtime support for 32- and 64-bit floating-point types satisfying the range and precision constraints in the Floating-Point Computation section.
- The representation and endianness of these types on the host **must** match the representation and endianness of the same types on every physical device supported.

**NOTE** Since a variety of data types and structures in Vulkan **may** be accessible by both host and physical device operations, the implementation **should** be able to access such data efficiently in both paths in order to facilitate writing portable and performant applications.

## 3.2. Execution Model

This section outlines the execution model of a Vulkan system.

Vulkan exposes one or more *devices*, each of which exposes one or more *queues* which **may** process work asynchronously to one another. The set of queues supported by a device is partitioned into *families*. Each family supports one or more types of functionality and **may** contain multiple queues with similar characteristics. Queues within a single family are considered *compatible* with one another, and work produced for a family of queues **can** be executed on any queue within that family. This specification defines the following types of functionality that queues **may** support: graphics, compute, video decode, video encode, protected memory management, sparse memory management, and transfer.

**NOTE** A single device **may** report multiple similar queue families rather than, or as well as, reporting multiple members of one or more of those families. This indicates that while members of those families have similar capabilities, they are *not* directly compatible with one another.

Device memory is explicitly managed by the application. Each device **may** advertise one or more heaps, representing different areas of memory. Memory heaps are either device-local or host-local, but are always visible to the device. Further detail about memory heaps is exposed via memory types available on that heap. Examples of memory areas that **may** be available on an

implementation include:

- *device-local* is memory that is physically connected to the device.
- *device-local, host visible* is device-local memory that is visible to the host.
- *host-local, host visible* is memory that is local to the host and visible to the device and host.

On other architectures, there **may** only be a single heap that **can** be used for any purpose.

### 3.2.1. Queue Operation

Vulkan queues provide an interface to the execution engines of a device. Commands for these execution engines are recorded into command buffers ahead of execution time, and then submitted to a queue for execution. Once submitted to a queue, command buffers will begin and complete execution without further application intervention, though the order of this execution is dependent on a number of implicit and explicit ordering constraints.

Work is submitted to queues using *queue submission commands* that typically take the form `vkQueue*` (e.g. vkQueueSubmit , vkQueueBindSparse ), and **can** take a list of semaphores upon which to wait before work begins and a list of semaphores to signal once work has completed. The work itself, as well as signaling and waiting on the semaphores are all *queue operations*. Queue submission commands return control to the application once queue operations have been submitted - they do not wait for completion.

There are no implicit ordering constraints between queue operations on different queues, or between queues and the host, so these **may** operate in any order with respect to each other. Explicit ordering constraints between different queues or with the host **can** be expressed with semaphores and fences.

Command buffer submissions to a single queue respect submission order and other implicit ordering guarantees, but otherwise **may** overlap or execute out of order. Other types of batches and queue submissions against a single queue (e.g. sparse memory binding) have no implicit ordering constraints with any other queue submission or batch. Additional explicit ordering constraints between queue submissions and individual batches can be expressed with semaphores and fences.

Before a fence or semaphore is signaled, it is guaranteed that any previously submitted queue operations have completed execution, and that memory writes from those queue operations are available to future queue operations. Waiting on a signaled semaphore or fence guarantees that previous writes that are available are also visible to subsequent commands.

Command buffer boundaries, both between primary command buffers of the same or different batches or submissions as well as between primary and secondary command buffers, do not introduce any additional ordering constraints. In other words, submitting the set of command buffers (which **can** include executing secondary command buffers) between any semaphore or fence operations execute the recorded commands as if they had all been recorded into a single primary command buffer, except that the current state is reset on each boundary. Explicit ordering constraints **can** be expressed with explicit synchronization primitives.

There are a few implicit ordering guarantees between commands within a command buffer, but only covering a subset of execution. Additional explicit ordering constraints can be expressed with

the various explicit synchronization primitives.

| | |
|---|---|
| **NOTE** | Implementations have significant freedom to overlap execution of work submitted to a queue, and this is common due to deep pipelining and parallelism in Vulkan devices. |

Commands recorded in command buffers can perform actions, set state that persists across commands, synchronize other commands, or indirectly launch other commands, with some commands fulfilling several of these roles. The "Command Properties" section for each such command lists which of these roles the command takes:

**Action**

*Action commands* perform operations that can update values in memory. E.g. draw commands, dispatch commands.

**State**

*State setting commands* update the current state of a command buffer, affecting the operation of future action commands.

**Synchronization**

*Synchronization commands* impose ordering constraints on action commands, by introducing explicit execution and memory dependencies.

**Indirection**

*Indirection commands* execute other commands which were not directly recorded in the same command buffer.

| | |
|---|---|
| **NOTE** | In the absence of explicit synchronization or implicit ordering guarantees, action commands may overlap execution or execute out of order, potentially leading to data races. However, such reordering does not affect the current state observed by any action command. Each action command uses the state in effect at the point where the command occurs in the command buffer, regardless of when it is executed. |

## 3.3. Object Model

The devices, queues, and other entities in Vulkan are represented by Vulkan objects. At the API level, all objects are referred to by handles. There are two classes of handles, dispatchable and non-dispatchable. *Dispatchable* handle types are a pointer to an opaque type. This pointer **may** be used by layers as part of intercepting API commands, and thus each API command takes a dispatchable type as its first parameter. Each object of a dispatchable type **must** have a unique handle value during its lifetime.

*Non-dispatchable* handle types are a 64-bit integer type whose meaning is implementation-dependent. If the `privateData` feature is enabled for a VkDevice, each object of a non-dispatchable type created on that device **must** have a handle value that is unique among objects created on that device, for the duration of the object's lifetime. Otherwise, non-dispatchable handles **may** encode

object information directly in the handle rather than acting as a reference to an underlying object, and thus **may** not have unique handle values. If handle values are not unique, then destroying one such handle **must** not cause identical handles of other types to become invalid, and **must** not cause identical handles of the same type to become invalid if that handle value has been created more times than it has been destroyed.

All objects created or allocated from a `VkDevice` (i.e. with a `VkDevice` as the first parameter) are private to that device, and **must** not be used on other devices.

### 3.3.1. Object Lifetime

Objects are created or allocated by `vkCreate*` and `vkAllocate*` commands, respectively. Once an object is created or allocated, its "structure" is considered to be immutable, though the contents of certain object types is still free to change. Objects are destroyed or freed by `vkDestroy*` and `vkFree*` commands, respectively.

Objects that are allocated (rather than created) take resources from an existing pool object or memory heap, and when freed return resources to that pool or heap. While object creation and destruction are generally expected to be low-frequency occurrences during runtime, allocating and freeing objects **can** occur at high frequency. Pool objects help accommodate improved performance of the allocations and frees.

It is an application's responsibility to track the lifetime of Vulkan objects, and not to destroy them while they are still in use.

The ownership of application-owned memory is immediately acquired by any Vulkan command it is passed into. Ownership of such memory **must** be released back to the application at the end of the duration of the command, unless that command was deferred, so that the application **can** alter or free this memory as soon as all the commands that acquired it have returned. If the command was deferred, ownership of such memory is released back to the application when the deferred operation is complete.

The following object types are consumed when they are passed into a Vulkan command and not further accessed by the objects they are used to create. They **must** not be destroyed in the duration of any API command they are passed into:

- `VkShaderModule`
- `VkPipelineCache`
- `VkValidationCacheEXT`

A `VkRenderPass` or `VkPipelineLayout` object passed as a parameter to create another object is not further accessed by that object after the duration of the command it is passed into. A `VkRenderPass` used in a command buffer follows the rules described below.

`VkDescriptorSetLayout` objects **may** be accessed by commands that operate on descriptor sets allocated using that layout, and those descriptor sets **must** not be updated with vkUpdateDescriptorSets after the descriptor set layout has been destroyed. Otherwise, a VkDescriptorSetLayout object is no longer referenced by an API command it is passed into once host execution of that command completes.

The application **must** not destroy any other type of Vulkan object until all uses of that object by the device (such as via command buffer execution) have completed.

The following Vulkan objects **must** not be destroyed while any command buffers using the object are in the pending state:

- `VkEvent`
- `VkQueryPool`
- `VkBuffer`
- `VkBufferView`
- `VkImage`
- `VkImageView`
- `VkPipeline`
- `VkSampler`
- `VkSamplerYcbcrConversion`
- `VkDescriptorPool`
- `VkFramebuffer`
- `VkRenderPass`
- `VkCommandBuffer`
- `VkCommandPool`
- `VkDeviceMemory`
- `VkDescriptorSet`
- `VkIndirectCommandsLayoutNV`
- `VkAccelerationStructureNV`
- `VkAccelerationStructureKHR`
- `VkVideoSessionKHR`
- `VkVideoSessionParametersKHR`

Destroying these objects will move any command buffers that are in the recording or executable state, and are using those objects, to the invalid state.

The following Vulkan objects **must** not be destroyed while any queue is executing commands that use the object:

- `VkFence`
- `VkSemaphore`
- `VkCommandBuffer`
- `VkCommandPool`

In general, objects **can** be destroyed or freed in any order, even if the object being freed is involved

in the use of another object (e.g. use of a resource in a view, use of a view in a descriptor set, use of a pipeline library in another pipeline, use of a referenced pipeline for additional graphics shader groups in another pipeline, use of a bottom level acceleration structure in an instance referenced by a top level acceleration structure, use of an object in a command buffer, binding of a memory allocation to a resource), as long as any object that uses the freed object is not further used in any way except to be destroyed or to be reset in such a way that it no longer uses the other object (such as resetting a command buffer). If the object has been reset, then it **can** be used as if it never used the freed object. An exception to this is when there is a parent/child relationship between objects. In this case, the application **must** not destroy a parent object before its children, except when the parent is explicitly defined to free its children when it is destroyed (e.g. for pool objects, as defined below).

`VkCommandPool` objects are parents of `VkCommandBuffer` objects. `VkDescriptorPool` objects are parents of `VkDescriptorSet` objects. `VkDevice` objects are parents of many object types (all that take a `VkDevice` as a parameter to their creation).

The following Vulkan objects have specific restrictions for when they **can** be destroyed:

- `VkQueue` objects **cannot** be explicitly destroyed. Instead, they are implicitly destroyed when the `VkDevice` object they are retrieved from is destroyed.
- Destroying a pool object implicitly frees all objects allocated from that pool. Specifically, destroying `VkCommandPool` frees all `VkCommandBuffer` objects that were allocated from it, and destroying `VkDescriptorPool` frees all `VkDescriptorSet` objects that were allocated from it.
- `VkDevice` objects **can** be destroyed when all `VkQueue` objects retrieved from them are idle, and all objects created from them have been destroyed.
  - This includes the following objects:
    - `VkFence`
    - `VkSemaphore`
    - `VkEvent`
    - `VkQueryPool`
    - `VkBuffer`
    - `VkBufferView`
    - `VkImage`
    - `VkImageView`
    - `VkShaderModule`
    - `VkPipelineCache`
    - `VkPipeline`
    - `VkPipelineLayout`
    - `VkSampler`
    - `VkSamplerYcbcrConversion`
    - `VkDescriptorSetLayout`

    - VkDescriptorPool
    - VkFramebuffer
    - VkRenderPass
    - VkCommandPool
    - VkCommandBuffer
    - VkDeviceMemory
    - VkValidationCacheEXT
    - VkAccelerationStructureNV
    - VkAccelerationStructureKHR
    - VkVideoSessionKHR
    - VkVideoSessionParametersKHR

- VkPhysicalDevice objects **cannot** be explicitly destroyed. Instead, they are implicitly destroyed when the VkInstance object they are retrieved from is destroyed.
- VkInstance objects **can** be destroyed once all VkDevice objects created from any of its VkPhysicalDevice objects have been destroyed.

### 3.3.2. External Object Handles

As defined above, the scope of object handles created or allocated from a VkDevice is limited to that logical device. Objects which are not in scope are said to be external. To bring an external object into scope, an external handle **must** be exported from the object in the source scope and imported into the destination scope.

| | |
|---|---|
| **NOTE** | The scope of external handles and their associated resources **may** vary according to their type, but they **can** generally be shared across process and API boundaries. |

## 3.4. Application Binary Interface

The mechanism by which Vulkan is made available to applications is platform- or implementation-defined. On many platforms the C interface described in this Specification is provided by a shared library. Since shared libraries can be changed independently of the applications that use them, they present particular compatibility challenges, and this Specification places some requirements on them.

Shared library implementations **must** use the default Application Binary Interface (ABI) of the standard C compiler for the platform, or provide customized API headers that cause application code to use the implementation's non-default ABI. An ABI in this context means the size, alignment, and layout of C data types; the procedure calling convention; and the naming convention for shared library symbols corresponding to C functions. Customizing the calling convention for a platform is usually accomplished by defining calling convention macros appropriately in vk_platform.h.

On platforms where Vulkan is provided as a shared library, library symbols beginning with "vk" and followed by a digit or uppercase letter are reserved for use by the implementation.

Applications which use Vulkan **must** not provide definitions of these symbols. This allows the Vulkan shared library to be updated with additional symbols for new API versions or extensions without causing symbol conflicts with existing applications.

Shared library implementations **should** provide library symbols for commands in the highest version of this Specification they support, and for Window System Integration extensions relevant to the platform. They **may** also provide library symbols for commands defined by additional extensions.

**NOTE**

These requirements and recommendations are intended to allow implementors to take advantage of platform-specific conventions for SDKs, ABIs, library versioning mechanisms, etc. while still minimizing the code changes necessary to port applications or libraries between platforms. Platform vendors, or providers of the *de facto* standard Vulkan shared library for a platform, are encouraged to document what symbols the shared library provides and how it will be versioned when new symbols are added.

Applications **should** only rely on shared library symbols for commands in the minimum core version required by the application. vkGetInstanceProcAddr and vkGetDeviceProcAddr **should** be used to obtain function pointers for commands in core versions beyond the application's minimum required version.

## 3.5. Command Syntax and Duration

The Specification describes Vulkan commands as functions or procedures using C99 syntax. Language bindings for other languages such as C++ and JavaScript **may** allow for stricter parameter passing, or object-oriented interfaces.

Vulkan uses the standard C types for the base type of scalar parameters (e.g. types from `<stdint.h>`), with exceptions described below, or elsewhere in the text when appropriate:

VkBool32 represents boolean `True` and `False` values, since C does not have a sufficiently portable built-in boolean type:

```
// Provided by VK_VERSION_1_0
typedef uint32_t VkBool32;
```

`VK_TRUE` represents a boolean **True** (unsigned integer 1) value, and `VK_FALSE` a boolean **False** (unsigned integer 0) value.

All values returned from a Vulkan implementation in a VkBool32 will be either `VK_TRUE` or `VK_FALSE`.

Applications **must** not pass any other values than `VK_TRUE` or `VK_FALSE` into a Vulkan implementation where a VkBool32 is expected.

`VK_TRUE` is a constant representing a VkBool32 **True** value.

```
#define VK_TRUE                         1U
```

VK_FALSE is a constant representing a VkBool32 **False** value.

```
#define VK_FALSE                        0U
```

VkDeviceSize represents device memory size and offset values:

```
// Provided by VK_VERSION_1_0
typedef uint64_t VkDeviceSize;
```

VkDeviceAddress represents device buffer address values:

```
// Provided by VK_VERSION_1_0
typedef uint64_t VkDeviceAddress;
```

Commands that create Vulkan objects are of the form vkCreate* and take Vk*CreateInfo structures with the parameters needed to create the object. These Vulkan objects are destroyed with commands of the form vkDestroy*.

The last in-parameter to each command that creates or destroys a Vulkan object is pAllocator. The pAllocator parameter **can** be set to a non-NULL value such that allocations for the given object are delegated to an application provided callback; refer to the Memory Allocation chapter for further details.

Commands that allocate Vulkan objects owned by pool objects are of the form vkAllocate*, and take Vk*AllocateInfo structures. These Vulkan objects are freed with commands of the form vkFree*. These objects do not take allocators; if host memory is needed, they will use the allocator that was specified when their parent pool was created.

Commands are recorded into a command buffer by calling API commands of the form vkCmd*. Each such command **may** have different restrictions on where it **can** be used: in a primary and/or secondary command buffer, inside and/or outside a render pass, and in one or more of the supported queue types. These restrictions are documented together with the definition of each such command.

The *duration* of a Vulkan command refers to the interval between calling the command and its return to the caller.

### 3.5.1. Lifetime of Retrieved Results

Information is retrieved from the implementation with commands of the form vkGet* and vkEnumerate*.

Unless otherwise specified for an individual command, the results are *invariant*; that is, they will

remain unchanged when retrieved again by calling the same command with the same parameters, so long as those parameters themselves all remain valid.

### 3.5.2. Array Results

Some query commands of the form `vkGet*` and `vkEnumerate*` enable retrieving multiple results in the form of a return array. Such commands typically have two pointer arguments as follows:

- An element count pointer pointing to an integer variable, conventionally named as `p*Count` where `*` is the capitalized singular form of the name of the retrieved values.
- A pointer to an array where the result array is retrieved, conventionally named as `p*` where `*` is the capitalized plural form of the name of the retrieved values.

If such commands are called with the array pointer set to `NULL`, then the number of retrievable elements is returned in the variable pointed to by the element count pointer. Otherwise, the element count pointer **must** point to a variable set by the application to the number of elements in the return array, and on return the variable is overwritten with the number of elements actually written to the return array. If the input element count is less than the number of retrievable array elements, the query will write only as many elements to the return array as specified by the element count variable set by the application, and the command will return `VK_INCOMPLETE` instead of `VK_SUCCESS`, to indicate that not all retrievable array elements were returned.

**NOTE**

In practice, this means that applications will typically call such query commands twice:

- First, with the array pointer set to `NULL`, to retrieve the number of retrievable elements.
- Second, with the array pointer pointing to an application allocated storage for at least as many elements as indicated by the variable pointed to by the element count pointer, to retrieve at most as many of the retrievable elements.

Query commands that return one or more structures, regardless of whether they return a single or an array of structures with or without a `pNext` chain, **may** also contain arrays within those structures. Such return arrays are typically defined in the form of two members as follows:

- An integer value specifying the element count, conventionally named as `*Count` where `*` is the singular form of the name of the retrieved values.
- A pointer to an array where the result array is retrieved, conventionally named as `p*` where `*` is the capitalized plural form of the name of the retrieved values.

Analogously to query commands that return multiple results, if the command is called with the array pointer member of the output structure in question set to `NULL`, then the number of retrievable elements is returned in the element count member of that output structure. Otherwise, the element count **must** specify the number of elements in the return array, and on return the element count member is overwritten with the number of elements actually written to the return array. If the input element count is less than the number of retrievable array elements, the query will write only as many elements to the return array as specified by the input element count, and the command will return `VK_INCOMPLETE` instead of `VK_SUCCESS`, if the query command has a VkResult

return type, to indicate that not all retrievable array elements were returned.

**NOTE**

Applications need to separately track the value they provided as the input element count member for such arrays and compare those with the returned element counts in order to determine whether the actually returned element count is smaller than the size of the return array. Another side effect of this is that it is impossible for the application to determine if the number of retrievable elements has increased beyond the provided input element count so using return arrays in output structures **should** be limited to *invariant* array results. In practice, this means that applications will typically call such query commands multiple times:

- First, with the array pointer member(s) set to `NULL`, to retrieve the number(s) of retrievable elements.
- Second, with the array pointer(s) pointing to an application allocated storage for at least as many elements as indicated by the element count member(s), to retrieve at most as many of the retrievable elements.
- Then the process **may** need to be repeated for all other newly introduced return arrays in any nested output structures indirectly specified through the previously retrieved result arrays.

Regardless of the type of query command, any array pointer member of an output structure **must** either be `NULL`, or point to an application-allocated array. Query commands **must** not return a pointer to implementation allocated storage in any output structure.

## 3.6. Threading Behavior

Vulkan is intended to provide scalable performance when used on multiple host threads. All commands support being called concurrently from multiple threads, but certain parameters, or components of parameters are defined to be *externally synchronized*. This means that the caller **must** guarantee that no more than one thread is using such a parameter at a given time.

More precisely, Vulkan commands use simple stores to update the state of Vulkan objects. A parameter declared as externally synchronized **may** have its contents updated at any time during the host execution of the command. If two commands operate on the same object and at least one of the commands declares the object to be externally synchronized, then the caller **must** guarantee not only that the commands do not execute simultaneously, but also that the two commands are separated by an appropriate memory barrier (if needed).

**NOTE**

Memory barriers are particularly relevant for hosts based on the ARM CPU architecture, which is more weakly ordered than many developers are accustomed to from x86/x64 programming. Fortunately, most higher-level synchronization primitives (like the pthread library) perform memory barriers as a part of mutual exclusion, so mutexing Vulkan objects via these primitives will have the desired effect.

Similarly the application **must** avoid any potential data hazard of application-owned memory that has its ownership temporarily acquired by a Vulkan command. While the ownership of application-

owned memory remains acquired by a command the implementation **may** read the memory at any point, and it **may** write non-`const` qualified memory at any point. Parameters referring to non-`const` qualified application-owned memory are not marked explicitly as *externally synchronized* in the Specification.

If an application is using deferred host operations in a command, and that operation is successfully deferred, object parameters and application-owned memory passed to that command **may** be accessed at any time until the deferred operation is complete.

Many object types are *immutable*, meaning the objects **cannot** change once they have been created. These types of objects never need external synchronization, except that they **must** not be destroyed while they are in use on another thread. In certain special cases mutable object parameters are internally synchronized, making external synchronization unnecessary. Any command parameters that are not labeled as externally synchronized are either not mutated by the command or are internally synchronized. Additionally, certain objects related to a command's parameters (e.g. command pools and descriptor pools) **may** be affected by a command, and **must** also be externally synchronized. These implicit parameters are documented as described below.

Parameters of commands that are externally synchronized are listed below.

### Externally Synchronized Parameters

- The `instance` parameter in vkDestroyInstance
- The `device` parameter in vkDestroyDevice
- The `queue` parameter in vkQueueSubmit
- The `fence` parameter in vkQueueSubmit
- The `queue` parameter in vkQueueWaitIdle
- The `memory` parameter in vkFreeMemory
- The `memory` parameter in vkMapMemory
- The `memory` parameter in vkUnmapMemory
- The `buffer` parameter in vkBindBufferMemory
- The `image` parameter in vkBindImageMemory
- The `queue` parameter in vkQueueBindSparse
- The `fence` parameter in vkQueueBindSparse
- The `fence` parameter in vkDestroyFence
- The `semaphore` parameter in vkDestroySemaphore
- The `event` parameter in vkDestroyEvent
- The `event` parameter in vkSetEvent
- The `event` parameter in vkResetEvent
- The `queryPool` parameter in vkDestroyQueryPool
- The `buffer` parameter in vkDestroyBuffer

- The `bufferView` parameter in vkDestroyBufferView
- The `image` parameter in vkDestroyImage
- The `imageView` parameter in vkDestroyImageView
- The `shaderModule` parameter in vkDestroyShaderModule
- The `pipelineCache` parameter in vkDestroyPipelineCache
- The `dstCache` parameter in vkMergePipelineCaches
- The `pipeline` parameter in vkDestroyPipeline
- The `pipelineLayout` parameter in vkDestroyPipelineLayout
- The `sampler` parameter in vkDestroySampler
- The `descriptorSetLayout` parameter in vkDestroyDescriptorSetLayout
- The `descriptorPool` parameter in vkDestroyDescriptorPool
- The `descriptorPool` parameter in vkResetDescriptorPool
- The `descriptorPool` member of the `pAllocateInfo` parameter in vkAllocateDescriptorSets
- The `descriptorPool` parameter in vkFreeDescriptorSets
- The `framebuffer` parameter in vkDestroyFramebuffer
- The `renderPass` parameter in vkDestroyRenderPass
- The `commandPool` parameter in vkDestroyCommandPool
- The `commandPool` parameter in vkResetCommandPool
- The `commandPool` member of the `pAllocateInfo` parameter in vkAllocateCommandBuffers
- The `commandPool` parameter in vkFreeCommandBuffers
- The `commandBuffer` parameter in vkBeginCommandBuffer
- The `commandBuffer` parameter in vkEndCommandBuffer
- The `commandBuffer` parameter in vkResetCommandBuffer
- The `commandBuffer` parameter in vkCmdBindPipeline
- The `commandBuffer` parameter in vkCmdSetViewport
- The `commandBuffer` parameter in vkCmdSetScissor
- The `commandBuffer` parameter in vkCmdSetLineWidth
- The `commandBuffer` parameter in vkCmdSetDepthBias
- The `commandBuffer` parameter in vkCmdSetBlendConstants
- The `commandBuffer` parameter in vkCmdSetDepthBounds
- The `commandBuffer` parameter in vkCmdSetStencilCompareMask
- The `commandBuffer` parameter in vkCmdSetStencilWriteMask
- The `commandBuffer` parameter in vkCmdSetStencilReference
- The `commandBuffer` parameter in vkCmdBindDescriptorSets
- The `commandBuffer` parameter in vkCmdBindIndexBuffer

- The `commandBuffer` parameter in vkCmdBindVertexBuffers
- The `commandBuffer` parameter in vkCmdDraw
- The `commandBuffer` parameter in vkCmdDrawIndexed
- The `commandBuffer` parameter in vkCmdDrawIndirect
- The `commandBuffer` parameter in vkCmdDrawIndexedIndirect
- The `commandBuffer` parameter in vkCmdDispatch
- The `commandBuffer` parameter in vkCmdDispatchIndirect
- The `commandBuffer` parameter in vkCmdCopyBuffer
- The `commandBuffer` parameter in vkCmdCopyImage
- The `commandBuffer` parameter in vkCmdBlitImage
- The `commandBuffer` parameter in vkCmdCopyBufferToImage
- The `commandBuffer` parameter in vkCmdCopyImageToBuffer
- The `commandBuffer` parameter in vkCmdUpdateBuffer
- The `commandBuffer` parameter in vkCmdFillBuffer
- The `commandBuffer` parameter in vkCmdClearColorImage
- The `commandBuffer` parameter in vkCmdClearDepthStencilImage
- The `commandBuffer` parameter in vkCmdClearAttachments
- The `commandBuffer` parameter in vkCmdResolveImage
- The `commandBuffer` parameter in vkCmdSetEvent
- The `commandBuffer` parameter in vkCmdResetEvent
- The `commandBuffer` parameter in vkCmdWaitEvents
- The `commandBuffer` parameter in vkCmdPipelineBarrier
- The `commandBuffer` parameter in vkCmdBeginQuery
- The `commandBuffer` parameter in vkCmdEndQuery
- The `commandBuffer` parameter in vkCmdResetQueryPool
- The `commandBuffer` parameter in vkCmdWriteTimestamp
- The `commandBuffer` parameter in vkCmdCopyQueryPoolResults
- The `commandBuffer` parameter in vkCmdPushConstants
- The `commandBuffer` parameter in vkCmdBeginRenderPass
- The `commandBuffer` parameter in vkCmdNextSubpass
- The `commandBuffer` parameter in vkCmdEndRenderPass
- The `commandBuffer` parameter in vkCmdExecuteCommands
- The `commandBuffer` parameter in vkCmdSetDeviceMask
- The `commandBuffer` parameter in vkCmdDispatchBase
- The `commandPool` parameter in vkTrimCommandPool

- The `ycbcrConversion` parameter in vkDestroySamplerYcbcrConversion
- The `descriptorUpdateTemplate` parameter in vkDestroyDescriptorUpdateTemplate
- The `surface` parameter in vkDestroySurfaceKHR
- The `surface` member of the `pCreateInfo` parameter in vkCreateSwapchainKHR
- The `oldSwapchain` member of the `pCreateInfo` parameter in vkCreateSwapchainKHR
- The `swapchain` parameter in vkDestroySwapchainKHR
- The `swapchain` parameter in vkAcquireNextImageKHR
- The `semaphore` parameter in vkAcquireNextImageKHR
- The `fence` parameter in vkAcquireNextImageKHR
- The `queue` parameter in vkQueuePresentKHR
- The `surface` parameter in vkGetDeviceGroupSurfacePresentModesKHR
- The `surface` parameter in vkGetPhysicalDevicePresentRectanglesKHR
- The `display` parameter in vkCreateDisplayModeKHR
- The `mode` parameter in vkGetDisplayPlaneCapabilitiesKHR
- The `videoSession` parameter in vkDestroyVideoSessionKHR
- The `videoSession` parameter in vkBindVideoSessionMemoryKHR
- The `videoSessionParameters` parameter in vkDestroyVideoSessionParametersKHR
- The `commandBuffer` parameter in vkCmdBeginVideoCodingKHR
- The `commandBuffer` parameter in vkCmdEndVideoCodingKHR
- The `commandBuffer` parameter in vkCmdControlVideoCodingKHR
- The `commandBuffer` parameter in vkCmdDecodeVideoKHR
- The `commandBuffer` parameter in vkCmdBeginRenderingKHR
- The `commandBuffer` parameter in vkCmdEndRenderingKHR
- The `commandBuffer` parameter in vkCmdSetDeviceMaskKHR
- The `commandBuffer` parameter in vkCmdDispatchBaseKHR
- The `commandPool` parameter in vkTrimCommandPoolKHR
- The `commandBuffer` parameter in vkCmdPushDescriptorSetKHR
- The `commandBuffer` parameter in vkCmdPushDescriptorSetWithTemplateKHR
- The `descriptorUpdateTemplate` parameter in vkDestroyDescriptorUpdateTemplateKHR
- The `commandBuffer` parameter in vkCmdBeginRenderPass2KHR
- The `commandBuffer` parameter in vkCmdNextSubpass2KHR
- The `commandBuffer` parameter in vkCmdEndRenderPass2KHR
- The `swapchain` parameter in vkGetSwapchainStatusKHR
- The `ycbcrConversion` parameter in vkDestroySamplerYcbcrConversionKHR
- The `commandBuffer` parameter in vkCmdDrawIndirectCountKHR

- The `commandBuffer` parameter in vkCmdDrawIndexedIndirectCountKHR
- The `commandBuffer` parameter in vkCmdSetFragmentShadingRateKHR
- The `commandBuffer` parameter in vkCmdSetRenderingAttachmentLocationsKHR
- The `commandBuffer` parameter in vkCmdSetRenderingInputAttachmentIndicesKHR
- The `swapchain` parameter in vkWaitForPresentKHR
- The `operation` parameter in vkDestroyDeferredOperationKHR
- The `commandBuffer` parameter in vkCmdEncodeVideoKHR
- The `commandBuffer` parameter in vkCmdSetEvent2KHR
- The `commandBuffer` parameter in vkCmdResetEvent2KHR
- The `commandBuffer` parameter in vkCmdWaitEvents2KHR
- The `commandBuffer` parameter in vkCmdPipelineBarrier2KHR
- The `commandBuffer` parameter in vkCmdWriteTimestamp2KHR
- The `queue` parameter in vkQueueSubmit2KHR
- The `fence` parameter in vkQueueSubmit2KHR
- The `commandBuffer` parameter in vkCmdWriteBufferMarker2AMD
- The `commandBuffer` parameter in vkCmdCopyBuffer2KHR
- The `commandBuffer` parameter in vkCmdCopyImage2KHR
- The `commandBuffer` parameter in vkCmdCopyBufferToImage2KHR
- The `commandBuffer` parameter in vkCmdCopyImageToBuffer2KHR
- The `commandBuffer` parameter in vkCmdBlitImage2KHR
- The `commandBuffer` parameter in vkCmdResolveImage2KHR
- The `commandBuffer` parameter in vkCmdTraceRaysIndirect2KHR
- The `commandBuffer` parameter in vkCmdBindIndexBuffer2KHR
- The `pipelineBinary` parameter in vkDestroyPipelineBinaryKHR
- The `pipeline` member of the `pInfo` parameter in vkReleaseCapturedPipelineDataKHR
- The `commandBuffer` parameter in vkCmdSetLineStippleKHR
- The `commandBuffer` parameter in vkCmdBindDescriptorSets2KHR
- The `commandBuffer` parameter in vkCmdPushConstants2KHR
- The `commandBuffer` parameter in vkCmdPushDescriptorSet2KHR
- The `commandBuffer` parameter in vkCmdPushDescriptorSetWithTemplate2KHR
- The `commandBuffer` parameter in vkCmdSetDescriptorBufferOffsets2EXT
- The `commandBuffer` parameter in vkCmdBindDescriptorBufferEmbeddedSamplers2EXT
- The `callback` parameter in vkDestroyDebugReportCallbackEXT
- The `object` member of the `pTagInfo` parameter in vkDebugMarkerSetObjectTagEXT
- The `object` member of the `pNameInfo` parameter in vkDebugMarkerSetObjectNameEXT

- The `commandBuffer` parameter in vkCmdDebugMarkerBeginEXT
- The `commandBuffer` parameter in vkCmdDebugMarkerEndEXT
- The `commandBuffer` parameter in vkCmdDebugMarkerInsertEXT
- The `commandBuffer` parameter in vkCmdBindTransformFeedbackBuffersEXT
- The `commandBuffer` parameter in vkCmdBeginTransformFeedbackEXT
- The `commandBuffer` parameter in vkCmdEndTransformFeedbackEXT
- The `commandBuffer` parameter in vkCmdBeginQueryIndexedEXT
- The `commandBuffer` parameter in vkCmdEndQueryIndexedEXT
- The `commandBuffer` parameter in vkCmdDrawIndirectByteCountEXT
- The `commandBuffer` parameter in vkCmdDrawIndirectCountAMD
- The `commandBuffer` parameter in vkCmdDrawIndexedIndirectCountAMD
- The `commandBuffer` parameter in vkCmdBeginConditionalRenderingEXT
- The `commandBuffer` parameter in vkCmdEndConditionalRenderingEXT
- The `commandBuffer` parameter in vkCmdSetViewportWScalingNV
- The `swapchain` parameter in vkGetRefreshCycleDurationGOOGLE
- The `swapchain` parameter in vkGetPastPresentationTimingGOOGLE
- The `commandBuffer` parameter in vkCmdSetDiscardRectangleEXT
- The `commandBuffer` parameter in vkCmdSetDiscardRectangleEnableEXT
- The `commandBuffer` parameter in vkCmdSetDiscardRectangleModeEXT
- The `objectHandle` member of the `pNameInfo` parameter in vkSetDebugUtilsObjectNameEXT
- The `objectHandle` member of the `pTagInfo` parameter in vkSetDebugUtilsObjectTagEXT
- The `commandBuffer` parameter in vkCmdBeginDebugUtilsLabelEXT
- The `commandBuffer` parameter in vkCmdEndDebugUtilsLabelEXT
- The `commandBuffer` parameter in vkCmdInsertDebugUtilsLabelEXT
- The `messenger` parameter in vkDestroyDebugUtilsMessengerEXT
- The `commandBuffer` parameter in vkCmdSetSampleLocationsEXT
- The `validationCache` parameter in vkDestroyValidationCacheEXT
- The `dstCache` parameter in vkMergeValidationCachesEXT
- The `commandBuffer` parameter in vkCmdBindShadingRateImageNV
- The `commandBuffer` parameter in vkCmdSetViewportShadingRatePaletteNV
- The `commandBuffer` parameter in vkCmdSetCoarseSampleOrderNV
- The `accelerationStructure` parameter in vkDestroyAccelerationStructureNV
- The `commandBuffer` parameter in vkCmdBuildAccelerationStructureNV
- The `commandBuffer` parameter in vkCmdCopyAccelerationStructureNV
- The `commandBuffer` parameter in vkCmdTraceRaysNV

- The `commandBuffer` parameter in vkCmdWriteAccelerationStructuresPropertiesNV
- The `commandBuffer` parameter in vkCmdWriteBufferMarkerAMD
- The `commandBuffer` parameter in vkCmdDrawMeshTasksNV
- The `commandBuffer` parameter in vkCmdDrawMeshTasksIndirectNV
- The `commandBuffer` parameter in vkCmdDrawMeshTasksIndirectCountNV
- The `commandBuffer` parameter in vkCmdSetExclusiveScissorEnableNV
- The `commandBuffer` parameter in vkCmdSetExclusiveScissorNV
- The `commandBuffer` parameter in vkCmdSetCheckpointNV
- The `commandBuffer` parameter in vkCmdSetPerformanceMarkerINTEL
- The `commandBuffer` parameter in vkCmdSetPerformanceStreamMarkerINTEL
- The `commandBuffer` parameter in vkCmdSetPerformanceOverrideINTEL
- The `configuration` parameter in vkReleasePerformanceConfigurationINTEL
- The `commandBuffer` parameter in vkCmdSetLineStippleEXT
- The `commandBuffer` parameter in vkCmdSetCullModeEXT
- The `commandBuffer` parameter in vkCmdSetFrontFaceEXT
- The `commandBuffer` parameter in vkCmdSetPrimitiveTopologyEXT
- The `commandBuffer` parameter in vkCmdSetViewportWithCountEXT
- The `commandBuffer` parameter in vkCmdSetScissorWithCountEXT
- The `commandBuffer` parameter in vkCmdBindVertexBuffers2EXT
- The `commandBuffer` parameter in vkCmdSetDepthTestEnableEXT
- The `commandBuffer` parameter in vkCmdSetDepthWriteEnableEXT
- The `commandBuffer` parameter in vkCmdSetDepthCompareOpEXT
- The `commandBuffer` parameter in vkCmdSetDepthBoundsTestEnableEXT
- The `commandBuffer` parameter in vkCmdSetStencilTestEnableEXT
- The `commandBuffer` parameter in vkCmdSetStencilOpEXT
- The `commandBuffer` parameter in vkCmdPreprocessGeneratedCommandsNV
- The `commandBuffer` parameter in vkCmdExecuteGeneratedCommandsNV
- The `commandBuffer` parameter in vkCmdBindPipelineShaderGroupNV
- The `indirectCommandsLayout` parameter in vkDestroyIndirectCommandsLayoutNV
- The `commandBuffer` parameter in vkCmdSetDepthBias2EXT
- The `privateDataSlot` parameter in vkDestroyPrivateDataSlotEXT
- The `commandBuffer` parameter in vkCmdBindDescriptorBuffersEXT
- The `commandBuffer` parameter in vkCmdSetDescriptorBufferOffsetsEXT
- The `commandBuffer` parameter in vkCmdBindDescriptorBufferEmbeddedSamplersEXT
- The `commandBuffer` parameter in vkCmdSetFragmentShadingRateEnumNV

- The `commandBuffer` parameter in vkCmdSetVertexInputEXT
- The `commandBuffer` parameter in vkCmdSubpassShadingHUAWEI
- The `commandBuffer` parameter in vkCmdBindInvocationMaskHUAWEI
- The `commandBuffer` parameter in vkCmdSetPatchControlPointsEXT
- The `commandBuffer` parameter in vkCmdSetRasterizerDiscardEnableEXT
- The `commandBuffer` parameter in vkCmdSetDepthBiasEnableEXT
- The `commandBuffer` parameter in vkCmdSetLogicOpEXT
- The `commandBuffer` parameter in vkCmdSetPrimitiveRestartEnableEXT
- The `commandBuffer` parameter in vkCmdSetColorWriteEnableEXT
- The `commandBuffer` parameter in vkCmdDrawMultiEXT
- The `commandBuffer` parameter in vkCmdDrawMultiIndexedEXT
- The `micromap` parameter in vkDestroyMicromapEXT
- The `commandBuffer` parameter in vkCmdBuildMicromapsEXT
- The `commandBuffer` parameter in vkCmdCopyMicromapEXT
- The `commandBuffer` parameter in vkCmdCopyMicromapToMemoryEXT
- The `commandBuffer` parameter in vkCmdCopyMemoryToMicromapEXT
- The `commandBuffer` parameter in vkCmdWriteMicromapsPropertiesEXT
- The `commandBuffer` parameter in vkCmdDrawClusterHUAWEI
- The `commandBuffer` parameter in vkCmdDrawClusterIndirectHUAWEI
- The `commandBuffer` parameter in vkCmdCopyMemoryIndirectNV
- The `commandBuffer` parameter in vkCmdCopyMemoryToImageIndirectNV
- The `commandBuffer` parameter in vkCmdDecompressMemoryNV
- The `commandBuffer` parameter in vkCmdDecompressMemoryIndirectCountNV
- The `commandBuffer` parameter in vkCmdUpdatePipelineIndirectBufferNV
- The `commandBuffer` parameter in vkCmdSetDepthClampEnableEXT
- The `commandBuffer` parameter in vkCmdSetPolygonModeEXT
- The `commandBuffer` parameter in vkCmdSetRasterizationSamplesEXT
- The `commandBuffer` parameter in vkCmdSetSampleMaskEXT
- The `commandBuffer` parameter in vkCmdSetAlphaToCoverageEnableEXT
- The `commandBuffer` parameter in vkCmdSetAlphaToOneEnableEXT
- The `commandBuffer` parameter in vkCmdSetLogicOpEnableEXT
- The `commandBuffer` parameter in vkCmdSetColorBlendEnableEXT
- The `commandBuffer` parameter in vkCmdSetColorBlendEquationEXT
- The `commandBuffer` parameter in vkCmdSetColorWriteMaskEXT
- The `commandBuffer` parameter in vkCmdSetTessellationDomainOriginEXT

- The `commandBuffer` parameter in vkCmdSetRasterizationStreamEXT
- The `commandBuffer` parameter in vkCmdSetConservativeRasterizationModeEXT
- The `commandBuffer` parameter in vkCmdSetExtraPrimitiveOverestimationSizeEXT
- The `commandBuffer` parameter in vkCmdSetDepthClipEnableEXT
- The `commandBuffer` parameter in vkCmdSetSampleLocationsEnableEXT
- The `commandBuffer` parameter in vkCmdSetColorBlendAdvancedEXT
- The `commandBuffer` parameter in vkCmdSetProvokingVertexModeEXT
- The `commandBuffer` parameter in vkCmdSetLineRasterizationModeEXT
- The `commandBuffer` parameter in vkCmdSetLineStippleEnableEXT
- The `commandBuffer` parameter in vkCmdSetDepthClipNegativeOneToOneEXT
- The `commandBuffer` parameter in vkCmdSetViewportWScalingEnableNV
- The `commandBuffer` parameter in vkCmdSetViewportSwizzleNV
- The `commandBuffer` parameter in vkCmdSetCoverageToColorEnableNV
- The `commandBuffer` parameter in vkCmdSetCoverageToColorLocationNV
- The `commandBuffer` parameter in vkCmdSetCoverageModulationModeNV
- The `commandBuffer` parameter in vkCmdSetCoverageModulationTableEnableNV
- The `commandBuffer` parameter in vkCmdSetCoverageModulationTableNV
- The `commandBuffer` parameter in vkCmdSetShadingRateImageEnableNV
- The `commandBuffer` parameter in vkCmdSetRepresentativeFragmentTestEnableNV
- The `commandBuffer` parameter in vkCmdSetCoverageReductionModeNV
- The `shader` parameter in vkDestroyShaderEXT
- The `commandBuffer` parameter in vkCmdBindShadersEXT
- The `commandBuffer` parameter in vkCmdSetAttachmentFeedbackLoopEnableEXT
- The `accelerationStructure` parameter in vkDestroyAccelerationStructureKHR
- The `commandBuffer` parameter in vkCmdBuildAccelerationStructuresKHR
- The `commandBuffer` parameter in vkCmdBuildAccelerationStructuresIndirectKHR
- The `commandBuffer` parameter in vkCmdCopyAccelerationStructureKHR
- The `commandBuffer` parameter in vkCmdCopyAccelerationStructureToMemoryKHR
- The `commandBuffer` parameter in vkCmdCopyMemoryToAccelerationStructureKHR
- The `commandBuffer` parameter in vkCmdWriteAccelerationStructuresPropertiesKHR
- The `commandBuffer` parameter in vkCmdTraceRaysKHR
- The `commandBuffer` parameter in vkCmdTraceRaysIndirectKHR
- The `commandBuffer` parameter in vkCmdSetRayTracingPipelineStackSizeKHR
- The `commandBuffer` parameter in vkCmdDrawMeshTasksEXT
- The `commandBuffer` parameter in vkCmdDrawMeshTasksIndirectEXT

- The `commandBuffer` parameter in vkCmdDrawMeshTasksIndirectCountEXT

For VkPipelineCache objects created with `flags` containing `VK_PIPELINE_CACHE_CREATE_EXTERNALLY_SYNCHRONIZED_BIT`, the above table is extended with the `pipelineCache` parameter to `vkCreate*Pipelines` being externally synchronized.

There are also a few instances where a command **can** take in an application-allocated list whose contents are externally synchronized parameters. In these cases, the caller **must** guarantee that at most one thread is using a given element within the list at a given time. These parameters are listed below.

### Externally Synchronized Parameter Lists

- Each element of the `pFences` parameter in vkResetFences
- Each element of the `pDescriptorSets` parameter in vkFreeDescriptorSets
- Each element of the `pCommandBuffers` parameter in vkFreeCommandBuffers
- Each element of the `pWaitSemaphores` member of the `pPresentInfo` parameter in vkQueuePresentKHR
- Each element of the `pSwapchains` member of the `pPresentInfo` parameter in vkQueuePresentKHR
- The `surface` member of each element of the `pCreateInfos` parameter in vkCreateSharedSwapchainsKHR
- The `oldSwapchain` member of each element of the `pCreateInfos` parameter in vkCreateSharedSwapchainsKHR

In addition, there are some implicit parameters that need to be externally synchronized. For example, when a `commandBuffer` parameter needs to be externally synchronized, it implies that the `commandPool` from which that command buffer was allocated also needs to be externally synchronized. The implicit parameters and their associated object are listed below.

### Implicit Externally Synchronized Parameters

- All `VkPhysicalDevice` objects enumerated from `instance` in vkDestroyInstance
- All `VkQueue` objects created from `device` in vkDestroyDevice
- All `VkQueue` objects created from `device` in vkDeviceWaitIdle
- Any `VkDescriptorSet` objects allocated from `descriptorPool` in vkResetDescriptorPool
- The `VkCommandPool` that `commandBuffer` was allocated from in vkBeginCommandBuffer
- The `VkCommandPool` that `commandBuffer` was allocated from in vkEndCommandBuffer
- The `VkCommandPool` that `commandBuffer` was allocated from in vkResetCommandBuffer
- The `VkCommandPool` that `commandBuffer` was allocated from, in vkCmdBindPipeline
- The `VkCommandPool` that `commandBuffer` was allocated from, in vkCmdSetViewport

- The `VkCommandPool` that `commandBuffer` was allocated from, in vkCmdSetScissor
- The `VkCommandPool` that `commandBuffer` was allocated from, in vkCmdSetLineWidth
- The `VkCommandPool` that `commandBuffer` was allocated from, in vkCmdSetDepthBias
- The `VkCommandPool` that `commandBuffer` was allocated from, in vkCmdSetBlendConstants
- The `VkCommandPool` that `commandBuffer` was allocated from, in vkCmdSetDepthBounds
- The `VkCommandPool` that `commandBuffer` was allocated from, in vkCmdSetStencilCompareMask
- The `VkCommandPool` that `commandBuffer` was allocated from, in vkCmdSetStencilWriteMask
- The `VkCommandPool` that `commandBuffer` was allocated from, in vkCmdSetStencilReference
- The `VkCommandPool` that `commandBuffer` was allocated from, in vkCmdBindDescriptorSets
- The `VkCommandPool` that `commandBuffer` was allocated from, in vkCmdBindIndexBuffer
- The `VkCommandPool` that `commandBuffer` was allocated from, in vkCmdBindVertexBuffers
- The `VkCommandPool` that `commandBuffer` was allocated from, in vkCmdDraw
- The `VkCommandPool` that `commandBuffer` was allocated from, in vkCmdDrawIndexed
- The `VkCommandPool` that `commandBuffer` was allocated from, in vkCmdDrawIndirect
- The `VkCommandPool` that `commandBuffer` was allocated from, in vkCmdDrawIndexedIndirect
- The `VkCommandPool` that `commandBuffer` was allocated from, in vkCmdDispatch
- The `VkCommandPool` that `commandBuffer` was allocated from, in vkCmdDispatchIndirect
- The `VkCommandPool` that `commandBuffer` was allocated from, in vkCmdCopyBuffer
- The `VkCommandPool` that `commandBuffer` was allocated from, in vkCmdCopyImage
- The `VkCommandPool` that `commandBuffer` was allocated from, in vkCmdBlitImage
- The `VkCommandPool` that `commandBuffer` was allocated from, in vkCmdCopyBufferToImage
- The `VkCommandPool` that `commandBuffer` was allocated from, in vkCmdCopyImageToBuffer
- The `VkCommandPool` that `commandBuffer` was allocated from, in vkCmdUpdateBuffer
- The `VkCommandPool` that `commandBuffer` was allocated from, in vkCmdFillBuffer
- The `VkCommandPool` that `commandBuffer` was allocated from, in vkCmdClearColorImage
- The `VkCommandPool` that `commandBuffer` was allocated from, in vkCmdClearDepthStencilImage
- The `VkCommandPool` that `commandBuffer` was allocated from, in vkCmdClearAttachments
- The `VkCommandPool` that `commandBuffer` was allocated from, in vkCmdResolveImage
- The `VkCommandPool` that `commandBuffer` was allocated from, in vkCmdSetEvent
- The `VkCommandPool` that `commandBuffer` was allocated from, in vkCmdResetEvent
- The `VkCommandPool` that `commandBuffer` was allocated from, in vkCmdWaitEvents
- The `VkCommandPool` that `commandBuffer` was allocated from, in vkCmdPipelineBarrier
- The `VkCommandPool` that `commandBuffer` was allocated from, in vkCmdBeginQuery

- The `VkCommandPool` that `commandBuffer` was allocated from, in vkCmdEndQuery
- The `VkCommandPool` that `commandBuffer` was allocated from, in vkCmdResetQueryPool
- The `VkCommandPool` that `commandBuffer` was allocated from, in vkCmdWriteTimestamp
- The `VkCommandPool` that `commandBuffer` was allocated from, in vkCmdCopyQueryPoolResults
- The `VkCommandPool` that `commandBuffer` was allocated from, in vkCmdPushConstants
- The `VkCommandPool` that `commandBuffer` was allocated from, in vkCmdBeginRenderPass
- The `VkCommandPool` that `commandBuffer` was allocated from, in vkCmdNextSubpass
- The `VkCommandPool` that `commandBuffer` was allocated from, in vkCmdEndRenderPass
- The `VkCommandPool` that `commandBuffer` was allocated from, in vkCmdExecuteCommands
- The `VkCommandPool` that `commandBuffer` was allocated from, in vkCmdSetDeviceMask
- The `VkCommandPool` that `commandBuffer` was allocated from, in vkCmdDispatchBase
- The `VkCommandPool` that `commandBuffer` was allocated from, in vkCmdBeginVideoCodingKHR
- The `VkCommandPool` that `commandBuffer` was allocated from, in vkCmdEndVideoCodingKHR
- The `VkCommandPool` that `commandBuffer` was allocated from, in vkCmdControlVideoCodingKHR
- The `VkCommandPool` that `commandBuffer` was allocated from, in vkCmdDecodeVideoKHR
- The `VkCommandPool` that `commandBuffer` was allocated from, in vkCmdBeginRenderingKHR
- The `VkCommandPool` that `commandBuffer` was allocated from, in vkCmdEndRenderingKHR
- The `VkCommandPool` that `commandBuffer` was allocated from, in vkCmdSetDeviceMaskKHR
- The `VkCommandPool` that `commandBuffer` was allocated from, in vkCmdDispatchBaseKHR
- The `VkCommandPool` that `commandBuffer` was allocated from, in vkCmdPushDescriptorSetKHR
- The `VkCommandPool` that `commandBuffer` was allocated from, in vkCmdPushDescriptorSetWithTemplateKHR
- The `VkCommandPool` that `commandBuffer` was allocated from, in vkCmdBeginRenderPass2KHR
- The `VkCommandPool` that `commandBuffer` was allocated from, in vkCmdNextSubpass2KHR
- The `VkCommandPool` that `commandBuffer` was allocated from, in vkCmdEndRenderPass2KHR
- The `VkCommandPool` that `commandBuffer` was allocated from, in vkCmdDrawIndirectCountKHR
- The `VkCommandPool` that `commandBuffer` was allocated from, in vkCmdDrawIndexedIndirectCountKHR
- The `VkCommandPool` that `commandBuffer` was allocated from, in vkCmdSetFragmentShadingRateKHR
- The `VkCommandPool` that `commandBuffer` was allocated from, in vkCmdSetRenderingAttachmentLocationsKHR
- The `VkCommandPool` that `commandBuffer` was allocated from, in vkCmdSetRenderingInputAttachmentIndicesKHR
- The `VkCommandPool` that `commandBuffer` was allocated from, in vkCmdEncodeVideoKHR

- The VkCommandPool that commandBuffer was allocated from, in vkCmdSetEvent2KHR
- The VkCommandPool that commandBuffer was allocated from, in vkCmdResetEvent2KHR
- The VkCommandPool that commandBuffer was allocated from, in vkCmdWaitEvents2KHR
- The VkCommandPool that commandBuffer was allocated from, in vkCmdPipelineBarrier2KHR
- The VkCommandPool that commandBuffer was allocated from, in vkCmdWriteTimestamp2KHR
- The VkCommandPool that commandBuffer was allocated from, in vkCmdWriteBufferMarker2AMD
- The VkCommandPool that commandBuffer was allocated from, in vkCmdCopyBuffer2KHR
- The VkCommandPool that commandBuffer was allocated from, in vkCmdCopyImage2KHR
- The VkCommandPool that commandBuffer was allocated from, in vkCmdCopyBufferToImage2KHR
- The VkCommandPool that commandBuffer was allocated from, in vkCmdCopyImageToBuffer2KHR
- The VkCommandPool that commandBuffer was allocated from, in vkCmdBlitImage2KHR
- The VkCommandPool that commandBuffer was allocated from, in vkCmdResolveImage2KHR
- The VkCommandPool that commandBuffer was allocated from, in vkCmdTraceRaysIndirect2KHR
- The VkCommandPool that commandBuffer was allocated from, in vkCmdBindIndexBuffer2KHR
- The VkCommandPool that commandBuffer was allocated from, in vkCmdSetLineStippleKHR
- The VkCommandPool that commandBuffer was allocated from, in vkCmdBindDescriptorSets2KHR
- The VkCommandPool that commandBuffer was allocated from, in vkCmdPushConstants2KHR
- The VkCommandPool that commandBuffer was allocated from, in vkCmdPushDescriptorSet2KHR
- The VkCommandPool that commandBuffer was allocated from, in vkCmdPushDescriptorSetWithTemplate2KHR
- The VkCommandPool that commandBuffer was allocated from, in vkCmdSetDescriptorBufferOffsets2EXT
- The VkCommandPool that commandBuffer was allocated from, in vkCmdBindDescriptorBufferEmbeddedSamplers2EXT
- The VkCommandPool that commandBuffer was allocated from, in vkCmdDebugMarkerBeginEXT
- The VkCommandPool that commandBuffer was allocated from, in vkCmdDebugMarkerEndEXT
- The VkCommandPool that commandBuffer was allocated from, in vkCmdDebugMarkerInsertEXT
- The VkCommandPool that commandBuffer was allocated from, in vkCmdBindTransformFeedbackBuffersEXT
- The VkCommandPool that commandBuffer was allocated from, in

vkCmdBeginTransformFeedbackEXT

- The VkCommandPool that commandBuffer was allocated from, in vkCmdEndTransformFeedbackEXT
- The VkCommandPool that commandBuffer was allocated from, in vkCmdBeginQueryIndexedEXT
- The VkCommandPool that commandBuffer was allocated from, in vkCmdEndQueryIndexedEXT
- The VkCommandPool that commandBuffer was allocated from, in vkCmdDrawIndirectByteCountEXT
- The VkCommandPool that commandBuffer was allocated from, in vkCmdCuLaunchKernelNVX
- The VkCommandPool that commandBuffer was allocated from, in vkCmdDrawIndirectCountAMD
- The VkCommandPool that commandBuffer was allocated from, in vkCmdDrawIndexedIndirectCountAMD
- The VkCommandPool that commandBuffer was allocated from, in vkCmdBeginConditionalRenderingEXT
- The VkCommandPool that commandBuffer was allocated from, in vkCmdEndConditionalRenderingEXT
- The VkCommandPool that commandBuffer was allocated from, in vkCmdSetViewportWScalingNV
- The VkCommandPool that commandBuffer was allocated from, in vkCmdSetDiscardRectangleEXT
- The VkCommandPool that commandBuffer was allocated from, in vkCmdSetDiscardRectangleEnableEXT
- The VkCommandPool that commandBuffer was allocated from, in vkCmdSetDiscardRectangleModeEXT
- The VkCommandPool that commandBuffer was allocated from, in vkCmdBeginDebugUtilsLabelEXT
- The VkCommandPool that commandBuffer was allocated from, in vkCmdEndDebugUtilsLabelEXT
- The VkCommandPool that commandBuffer was allocated from, in vkCmdInsertDebugUtilsLabelEXT
- The VkCommandPool that commandBuffer was allocated from, in vkCmdInitializeGraphScratchMemoryAMDX
- The VkCommandPool that commandBuffer was allocated from, in vkCmdDispatchGraphAMDX
- The VkCommandPool that commandBuffer was allocated from, in vkCmdDispatchGraphIndirectAMDX
- The VkCommandPool that commandBuffer was allocated from, in vkCmdDispatchGraphIndirectCountAMDX
- The VkCommandPool that commandBuffer was allocated from, in vkCmdSetSampleLocationsEXT

- The VkCommandPool that commandBuffer was allocated from, in vkCmdBindShadingRateImageNV
- The VkCommandPool that commandBuffer was allocated from, in vkCmdSetViewportShadingRatePaletteNV
- The VkCommandPool that commandBuffer was allocated from, in vkCmdSetCoarseSampleOrderNV
- The VkCommandPool that commandBuffer was allocated from, in vkCmdBuildAccelerationStructureNV
- The VkCommandPool that commandBuffer was allocated from, in vkCmdCopyAccelerationStructureNV
- The VkCommandPool that commandBuffer was allocated from, in vkCmdTraceRaysNV
- The VkCommandPool that commandBuffer was allocated from, in vkCmdWriteAccelerationStructuresPropertiesNV
- The VkCommandPool that commandBuffer was allocated from, in vkCmdWriteBufferMarkerAMD
- The VkCommandPool that commandBuffer was allocated from, in vkCmdDrawMeshTasksNV
- The VkCommandPool that commandBuffer was allocated from, in vkCmdDrawMeshTasksIndirectNV
- The VkCommandPool that commandBuffer was allocated from, in vkCmdDrawMeshTasksIndirectCountNV
- The VkCommandPool that commandBuffer was allocated from, in vkCmdSetExclusiveScissorEnableNV
- The VkCommandPool that commandBuffer was allocated from, in vkCmdSetExclusiveScissorNV
- The VkCommandPool that commandBuffer was allocated from, in vkCmdSetCheckpointNV
- The VkCommandPool that commandBuffer was allocated from, in vkCmdSetPerformanceMarkerINTEL
- The VkCommandPool that commandBuffer was allocated from, in vkCmdSetPerformanceStreamMarkerINTEL
- The VkCommandPool that commandBuffer was allocated from, in vkCmdSetPerformanceOverrideINTEL
- The VkCommandPool that commandBuffer was allocated from, in vkCmdSetLineStippleEXT
- The VkCommandPool that commandBuffer was allocated from, in vkCmdSetCullModeEXT
- The VkCommandPool that commandBuffer was allocated from, in vkCmdSetFrontFaceEXT
- The VkCommandPool that commandBuffer was allocated from, in vkCmdSetPrimitiveTopologyEXT
- The VkCommandPool that commandBuffer was allocated from, in vkCmdSetViewportWithCountEXT
- The VkCommandPool that commandBuffer was allocated from, in vkCmdSetScissorWithCountEXT

- The VkCommandPool that commandBuffer was allocated from, in vkCmdBindVertexBuffers2EXT
- The VkCommandPool that commandBuffer was allocated from, in vkCmdSetDepthTestEnableEXT
- The VkCommandPool that commandBuffer was allocated from, in vkCmdSetDepthWriteEnableEXT
- The VkCommandPool that commandBuffer was allocated from, in vkCmdSetDepthCompareOpEXT
- The VkCommandPool that commandBuffer was allocated from, in vkCmdSetDepthBoundsTestEnableEXT
- The VkCommandPool that commandBuffer was allocated from, in vkCmdSetStencilTestEnableEXT
- The VkCommandPool that commandBuffer was allocated from, in vkCmdSetStencilOpEXT
- The VkCommandPool that commandBuffer was allocated from, in vkCmdPreprocessGeneratedCommandsNV
- The VkCommandPool that commandBuffer was allocated from, in vkCmdExecuteGeneratedCommandsNV
- The VkCommandPool that commandBuffer was allocated from, in vkCmdBindPipelineShaderGroupNV
- The VkCommandPool that commandBuffer was allocated from, in vkCmdSetDepthBias2EXT
- The VkCommandPool that commandBuffer was allocated from, in vkCmdCudaLaunchKernelNV
- The VkCommandPool that commandBuffer was allocated from, in vkCmdBindDescriptorBuffersEXT
- The VkCommandPool that commandBuffer was allocated from, in vkCmdSetDescriptorBufferOffsetsEXT
- The VkCommandPool that commandBuffer was allocated from, in vkCmdBindDescriptorBufferEmbeddedSamplersEXT
- The VkCommandPool that commandBuffer was allocated from, in vkCmdSetFragmentShadingRateEnumNV
- The VkCommandPool that commandBuffer was allocated from, in vkCmdSetVertexInputEXT
- The VkCommandPool that commandBuffer was allocated from, in vkCmdSubpassShadingHUAWEI
- The VkCommandPool that commandBuffer was allocated from, in vkCmdBindInvocationMaskHUAWEI
- The VkCommandPool that commandBuffer was allocated from, in vkCmdSetPatchControlPointsEXT
- The VkCommandPool that commandBuffer was allocated from, in vkCmdSetRasterizerDiscardEnableEXT
- The VkCommandPool that commandBuffer was allocated from, in vkCmdSetDepthBiasEnableEXT

- The VkCommandPool that commandBuffer was allocated from, in vkCmdSetLogicOpEXT
- The VkCommandPool that commandBuffer was allocated from, in vkCmdSetPrimitiveRestartEnableEXT
- The VkCommandPool that commandBuffer was allocated from, in vkCmdSetColorWriteEnableEXT
- The VkCommandPool that commandBuffer was allocated from, in vkCmdDrawMultiEXT
- The VkCommandPool that commandBuffer was allocated from, in vkCmdDrawMultiIndexedEXT
- The VkCommandPool that commandBuffer was allocated from, in vkCmdBuildMicromapsEXT
- The VkCommandPool that commandBuffer was allocated from, in vkCmdCopyMicromapEXT
- The VkCommandPool that commandBuffer was allocated from, in vkCmdCopyMicromapToMemoryEXT
- The VkCommandPool that commandBuffer was allocated from, in vkCmdCopyMemoryToMicromapEXT
- The VkCommandPool that commandBuffer was allocated from, in vkCmdWriteMicromapsPropertiesEXT
- The VkCommandPool that commandBuffer was allocated from, in vkCmdDrawClusterHUAWEI
- The VkCommandPool that commandBuffer was allocated from, in vkCmdDrawClusterIndirectHUAWEI
- The VkCommandPool that commandBuffer was allocated from, in vkCmdCopyMemoryIndirectNV
- The VkCommandPool that commandBuffer was allocated from, in vkCmdCopyMemoryToImageIndirectNV
- The VkCommandPool that commandBuffer was allocated from, in vkCmdDecompressMemoryNV
- The VkCommandPool that commandBuffer was allocated from, in vkCmdDecompressMemoryIndirectCountNV
- The VkCommandPool that commandBuffer was allocated from, in vkCmdUpdatePipelineIndirectBufferNV
- The VkCommandPool that commandBuffer was allocated from, in vkCmdSetDepthClampEnableEXT
- The VkCommandPool that commandBuffer was allocated from, in vkCmdSetPolygonModeEXT
- The VkCommandPool that commandBuffer was allocated from, in vkCmdSetRasterizationSamplesEXT
- The VkCommandPool that commandBuffer was allocated from, in vkCmdSetSampleMaskEXT
- The VkCommandPool that commandBuffer was allocated from, in vkCmdSetAlphaToCoverageEnableEXT
- The VkCommandPool that commandBuffer was allocated from, in vkCmdSetAlphaToOneEnableEXT
- The VkCommandPool that commandBuffer was allocated from, in vkCmdSetLogicOpEnableEXT

- The VkCommandPool that commandBuffer was allocated from, in vkCmdSetColorBlendEnableEXT
- The VkCommandPool that commandBuffer was allocated from, in vkCmdSetColorBlendEquationEXT
- The VkCommandPool that commandBuffer was allocated from, in vkCmdSetColorWriteMaskEXT
- The VkCommandPool that commandBuffer was allocated from, in vkCmdSetTessellationDomainOriginEXT
- The VkCommandPool that commandBuffer was allocated from, in vkCmdSetRasterizationStreamEXT
- The VkCommandPool that commandBuffer was allocated from, in vkCmdSetConservativeRasterizationModeEXT
- The VkCommandPool that commandBuffer was allocated from, in vkCmdSetExtraPrimitiveOverestimationSizeEXT
- The VkCommandPool that commandBuffer was allocated from, in vkCmdSetDepthClipEnableEXT
- The VkCommandPool that commandBuffer was allocated from, in vkCmdSetSampleLocationsEnableEXT
- The VkCommandPool that commandBuffer was allocated from, in vkCmdSetColorBlendAdvancedEXT
- The VkCommandPool that commandBuffer was allocated from, in vkCmdSetProvokingVertexModeEXT
- The VkCommandPool that commandBuffer was allocated from, in vkCmdSetLineRasterizationModeEXT
- The VkCommandPool that commandBuffer was allocated from, in vkCmdSetLineStippleEnableEXT
- The VkCommandPool that commandBuffer was allocated from, in vkCmdSetDepthClipNegativeOneToOneEXT
- The VkCommandPool that commandBuffer was allocated from, in vkCmdSetViewportWScalingEnableNV
- The VkCommandPool that commandBuffer was allocated from, in vkCmdSetViewportSwizzleNV
- The VkCommandPool that commandBuffer was allocated from, in vkCmdSetCoverageToColorEnableNV
- The VkCommandPool that commandBuffer was allocated from, in vkCmdSetCoverageToColorLocationNV
- The VkCommandPool that commandBuffer was allocated from, in vkCmdSetCoverageModulationModeNV
- The VkCommandPool that commandBuffer was allocated from, in vkCmdSetCoverageModulationTableEnableNV
- The VkCommandPool that commandBuffer was allocated from, in

vkCmdSetCoverageModulationTableNV

- The VkCommandPool that commandBuffer was allocated from, in vkCmdSetShadingRateImageEnableNV
- The VkCommandPool that commandBuffer was allocated from, in vkCmdSetRepresentativeFragmentTestEnableNV
- The VkCommandPool that commandBuffer was allocated from, in vkCmdSetCoverageReductionModeNV
- The VkCommandPool that commandBuffer was allocated from, in vkCmdOpticalFlowExecuteNV
- The VkCommandPool that commandBuffer was allocated from, in vkCmdBindShadersEXT
- The VkCommandPool that commandBuffer was allocated from, in vkCmdSetAttachmentFeedbackLoopEnableEXT
- The VkCommandPool that commandBuffer was allocated from, in vkCmdBuildAccelerationStructuresKHR
- The VkCommandPool that commandBuffer was allocated from, in vkCmdBuildAccelerationStructuresIndirectKHR
- The VkCommandPool that commandBuffer was allocated from, in vkCmdCopyAccelerationStructureKHR
- The VkCommandPool that commandBuffer was allocated from, in vkCmdCopyAccelerationStructureToMemoryKHR
- The VkCommandPool that commandBuffer was allocated from, in vkCmdCopyMemoryToAccelerationStructureKHR
- The VkCommandPool that commandBuffer was allocated from, in vkCmdWriteAccelerationStructuresPropertiesKHR
- The VkCommandPool that commandBuffer was allocated from, in vkCmdTraceRaysKHR
- The VkCommandPool that commandBuffer was allocated from, in vkCmdTraceRaysIndirectKHR
- The VkCommandPool that commandBuffer was allocated from, in vkCmdSetRayTracingPipelineStackSizeKHR
- The VkCommandPool that commandBuffer was allocated from, in vkCmdDrawMeshTasksEXT
- The VkCommandPool that commandBuffer was allocated from, in vkCmdDrawMeshTasksIndirectEXT
- The VkCommandPool that commandBuffer was allocated from, in vkCmdDrawMeshTasksIndirectCountEXT

## 3.7. Valid Usage

Valid usage defines a set of conditions which **must** be met in order to achieve well-defined runtime behavior in an application. These conditions depend only on Vulkan state, and the parameters or objects whose usage is constrained by the condition.

The core layer assumes applications are using the API correctly. Except as documented elsewhere in

the Specification, the behavior of the core layer to an application using the API incorrectly is undefined, and **may** include program termination. However, implementations **must** ensure that incorrect usage by an application does not affect the integrity of the operating system, the Vulkan implementation, or other applications in the system using Vulkan. In particular, any guarantees made by an operating system about whether memory from one process **can** be visible to another process or not **must** not be violated by a Vulkan implementation for **any memory allocation**. Vulkan implementations are not **required** to make additional security or integrity guarantees beyond those provided by the OS unless explicitly directed by the application's use of a particular feature or extension.

**NOTE**

For instance, if an operating system guarantees that data in all its memory allocations are set to zero when newly allocated, the Vulkan implementation **must** make the same guarantees for any allocations it controls (e.g. VkDeviceMemory).

Similarly, if an operating system guarantees that use-after-free of host allocations will not result in values written by another process becoming visible, the same guarantees **must** be made by the Vulkan implementation for device memory.

If the `protectedMemory` feature is supported, the implementation provides additional guarantees when invalid usage occurs to prevent values in protected memory from being accessed or inferred outside of protected operations, as described in Protected Memory Access Rules.

Some valid usage conditions have dependencies on runtime limits or feature availability. It is possible to validate these conditions against Vulkan's minimum supported values for these limits and features, or some subset of other known values.

Valid usage conditions do not cover conditions where well-defined behavior (including returning an error code) exists.

Valid usage conditions **should** apply to the command or structure where complete information about the condition would be known during execution of an application. This is such that a validation layer or linter **can** be written directly against these statements at the point they are specified.

**NOTE**

This does lead to some non-obvious places for valid usage statements. For instance, the valid values for a structure might depend on a separate value in the calling command. In this case, the structure itself will not reference this valid usage as it is impossible to determine validity from the structure that it is invalid - instead this valid usage would be attached to the calling command.

Another example is draw state - the state setters are independent, and can cause a legitimately invalid state configuration between draw calls; so the valid usage statements are attached to the place where all state needs to be valid - at the drawing command.

Valid usage conditions are described in a block labeled "Valid Usage" following each command or structure they apply to.

### 3.7.1. Usage Validation

Vulkan is a layered API. The lowest layer is the core Vulkan layer, as defined by this Specification. The application **can** use additional layers above the core for debugging, validation, and other purposes.

One of the core principles of Vulkan is that building and submitting command buffers **should** be highly efficient. Thus error checking and validation of state in the core layer is minimal, although more rigorous validation **can** be enabled through the use of layers.

Validation of correct API usage is left to validation layers. Applications **should** be developed with validation layers enabled, to help catch and eliminate errors. Once validated, released applications **should** not enable validation layers by default.

### 3.7.2. Implicit Valid Usage

Some valid usage conditions apply to all commands and structures in the API, unless explicitly denoted otherwise for a specific command or structure. These conditions are considered *implicit*, and are described in a block labeled "Valid Usage (Implicit)" following each command or structure they apply to. Implicit valid usage conditions are described in detail below.

#### Valid Usage for Object Handles

Any input parameter to a command that is an object handle **must** be a valid object handle, unless otherwise specified. An object handle is valid if:

- It has been created or allocated by a previous, successful call to the API. Such calls are noted in the Specification.
- It has not been deleted or freed by a previous call to the API. Such calls are noted in the Specification.
- Any objects used by that object, either as part of creation or execution, **must** also be valid.

The reserved values VK_NULL_HANDLE and `NULL` **can** be used in place of valid non-dispatchable handles and dispatchable handles, respectively, when *explicitly called out in the Specification*. Any command that creates an object successfully **must** not return these values. It is valid to pass these values to `vkDestroy*` or `vkFree*` commands, which will silently ignore these values.

#### Valid Usage for Pointers

Any parameter that is a pointer **must** be a *valid pointer* only if it is explicitly called out by a Valid Usage statement.

A pointer is "valid" if it points at memory containing values of the number and type(s) expected by the command, and all fundamental types accessed through the pointer (e.g. as elements of an array or as members of a structure) satisfy the alignment requirements of the host processor.

#### Valid Usage for Strings

Any parameter that is a pointer to `char` **must** be a finite sequence of values terminated by a null

character, or if *explicitly called out in the Specification*, **can** be `NULL`.

Strings specified as UTF-8 encoded **must** not contain invalid UTF-8 sequences. See String Representation for additional information about strings.

#### Valid Usage for Enumerated Types

Any parameter of an enumerated type **must** be a valid enumerant for that type. Use of an enumerant is valid if the following conditions are true:

- The enumerant is defined as part of the enumerated type.
- The enumerant is not a value suffixed with `_MAX_ENUM`.
  - This value exists only to ensure that C `enum` types are 32 bits in size and **must** not be used by applications.
- If the enumerant is used in a function that has a VkInstance as its first parameter and either:
  - it was added by a core version that is supported (as reported by vkEnumerateInstanceVersion) and the value of VkApplicationInfo::`apiVersion` is greater than or equal to the version that added it; or
  - it was added by an instance extension that was enabled for the instance.
- If the enumerant is used in a function that has a VkPhysicalDevice object as its first parameter and either:
  - it was added by a core version that is supported by that device (as reported by VkPhysicalDeviceProperties::`apiVersion`);
  - it was added by an instance extension that was enabled for the instance; or
  - it was added by a device extension that is supported by that device.
- If the enumerant is used in a function that has any other dispatchable object as its first parameter and either:
  - it was added by a core version that is supported for the device (as reported by VkPhysicalDeviceProperties::`apiVersion`); or
  - it was added by a device extension that was enabled for the device.

Additionally, if `maintenance5` is supported, any integer value representable in the range valid for the defined type is valid when used in a function that has a VkPhysicalDevice object as its first parameter. Physical device queries will either return results indicating lack of support, or ignore unsupported values when used as a bit flag in a `Vk*Flags*` parameter.

Any enumerated type returned from a query command or otherwise output from Vulkan to the application **must** not have a reserved value. Reserved values are values not defined by any extension for that enumerated type.

| | |
|---|---|
| **NOTE** | In some special cases, an enumerant is only meaningful if a feature defined by an extension is also enabled, as well as the extension itself. The global “valid enumerant” rule described here does not address such cases. |

| | |
|---|---|
| **NOTE** | This language is intended to accommodate cases such as “hidden” extensions known only to driver internals, or layers enabling extensions without knowledge of the application, without allowing return of values not defined by any extension. |

| | |
|---|---|
| **NOTE** | Application developers are encouraged to be careful when using `switch` statements with Vulkan API enums. This is because new extensions can add new values to existing enums. Using a `default:` statement within a `switch` may avoid future compilation issues. |

**Valid Usage for Flags**

A collection of flags is represented by a bitmask using the type VkFlags:

```
// Provided by VK_VERSION_1_0
typedef uint32_t VkFlags;
```

Bitmasks are passed to many commands and structures to compactly represent options, but VkFlags is not used directly in the API. Instead, a `Vk*Flags` type which is an alias of VkFlags, and whose name matches the corresponding `Vk*FlagBits` that are valid for that type, is used.

Any `Vk*Flags` member or parameter used in the API as an input **must** be a valid combination of bit flags. A valid combination is either zero or the bitwise OR of valid bit flags.

An individual bit flag is valid for a `Vk*Flags` type if it would be a valid enumerant when used with the equivalent `Vk*FlagBits` type, where the bits type is obtained by taking the flag type and replacing the trailing `Flags` with `FlagBits`. For example, a flag value of type VkColorComponentFlags **must** contain only bit flags defined by VkColorComponentFlagBits.

Any `Vk*Flags` member or parameter returned from a query command or otherwise output from Vulkan to the application **may** contain bit flags undefined in its corresponding `Vk*FlagBits` type. An application **cannot** rely on the state of these unspecified bits.

Only the low-order 31 bits (bit positions zero through 30) are available for use as flag bits.

| | |
|---|---|
| **NOTE** | This restriction is due to poorly defined behavior by C compilers given a C enumerant value of `0x80000000`. In some cases adding this enumerant value may increase the size of the underlying `Vk*FlagBits` type, breaking the ABI. |

A collection of 64-bit flags is represented by a bitmask using the type VkFlags64:

```
// Provided by VK_KHR_synchronization2
typedef uint64_t VkFlags64;
```

When the 31 bits available in VkFlags are insufficient, the VkFlags64 type can be passed to commands and structures to represent up to 64 options. VkFlags64 is not used directly in the API. Instead, a `Vk*Flags2` type which is an alias of VkFlags64, and whose name matches the

corresponding `Vk*FlagBits2` that are valid for that type, is used.

Any `Vk*Flags2` member or parameter used in the API as an input **must** be a valid combination of bit flags. A valid combination is either zero or the bitwise OR of valid bit flags.

An individual bit flag is valid for a `Vk*Flags2` type if it would be a valid enumerant when used with the equivalent `Vk*FlagBits2` type, where the bits type is obtained by taking the flag type and replacing the trailing `Flags2` with `FlagBits2`. For example, a flag value of type VkAccessFlags2KHR **must** contain only bit flags defined by VkAccessFlagBits2KHR.

Any `Vk*Flags2` member or parameter returned from a query command or otherwise output from Vulkan to the application **may** contain bit flags undefined in its corresponding `Vk*FlagBits2` type. An application **cannot** rely on the state of these unspecified bits.

**NOTE**

Both the `Vk*FlagBits2` type, and the individual bits defined for that type, are defined as `uint64_t` integers in the C API. This is in contrast to the 32-bit types, where the `Vk*FlagBits` type is defined as a C `enum` and the individual bits as enumerants belonging to that `enum`. As a result, there is less compile time type checking possible for the 64-bit types. This is unavoidable since there is no sufficiently portable way to define a 64-bit `enum` type in C99.

#### Valid Usage for Structure Types

Any parameter that is a structure containing a `sType` member **must** have a value of `sType` which is a valid VkStructureType value matching the type of the structure.

#### Valid Usage for Structure Pointer Chains

Any parameter that is a structure containing a `void* pNext` member **must** have a value of `pNext` that is either `NULL`, or is a pointer to a valid *extending structure*, containing `sType` and `pNext` members as described in the Vulkan Documentation and Extensions document in the section "Extending Structures". The set of structures connected by `pNext` pointers is referred to as a `pNext` *chain*.

Each structure included in the `pNext` chain **must** be defined at runtime by either:

- a core version which is supported
- an extension which is enabled
- a supported device extension in the case of physical-device-level functionality added by the device extension

Each type of extending structure **must** not appear more than once in a `pNext` chain, including any aliases. This general rule may be explicitly overridden for specific structures.

Any component of the implementation (the loader, any enabled layers, and drivers) **must** skip over, without processing (other than reading the `sType` and `pNext` members) any extending structures in the chain not defined by core versions or extensions supported by that component.

As a convenience to implementations and layers needing to iterate through a structure pointer chain, the Vulkan API provides two *base structures*. These structures allow for some type safety, and

can be used by Vulkan API functions that operate on generic inputs and outputs.

The VkBaseInStructure structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkBaseInStructure {
    VkStructureType                     sType;
    const struct VkBaseInStructure*     pNext;
} VkBaseInStructure;
```

- sType is the structure type of the structure being iterated through.
- pNext is NULL or a pointer to the next structure in a structure chain.

VkBaseInStructure can be used to facilitate iterating through a read-only structure pointer chain.

The VkBaseOutStructure structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkBaseOutStructure {
    VkStructureType                sType;
    struct VkBaseOutStructure*     pNext;
} VkBaseOutStructure;
```

- sType is the structure type of the structure being iterated through.
- pNext is NULL or a pointer to the next structure in a structure chain.

VkBaseOutStructure can be used to facilitate iterating through a structure pointer chain that returns data back to the application.

#### Valid Usage for Nested Structures

The above conditions also apply recursively to members of structures provided as input to a command, either as a direct argument to the command, or themselves a member of another structure.

Specifics on valid usage of each command are covered in their individual sections.

#### Valid Usage for Extensions

Instance-level functionality or behavior added by an instance extension to the API **must** not be used unless that extension is supported by the instance as determined by vkEnumerateInstanceExtensionProperties, and that extension is enabled in VkInstanceCreateInfo.

Physical-device-level functionality or behavior added by an instance extension to the API **must** not be used unless that extension is supported by the instance as determined by vkEnumerateInstanceExtensionProperties, and that extension is enabled in VkInstanceCreateInfo.

Physical-device-level functionality or behavior added by a device extension to the API **must** not be

used unless the conditions described in Extending Physical Device From Device Extensions are met.

Device-level functionality added by a device extension that is dispatched from a VkDevice, or from a child object of a VkDevice **must** not be used unless that extension is supported by the device as determined by vkEnumerateDeviceExtensionProperties, and that extension is enabled in VkDeviceCreateInfo.

#### Valid Usage for Newer Core Versions

Instance-level functionality or behavior added by a new core version of the API **must** not be used unless it is supported by the instance as determined by vkEnumerateInstanceVersion and the specified version of VkApplicationInfo::`apiVersion`.

Physical-device-level functionality or behavior added by a new core version of the API **must** not be used unless it is supported by the physical device as determined by VkPhysicalDeviceProperties::`apiVersion` and the specified version of VkApplicationInfo::`apiVersion`.

Device-level functionality or behavior added by a new core version of the API **must** not be used unless it is supported by the device as determined by VkPhysicalDeviceProperties::`apiVersion` and the specified version of VkApplicationInfo::`apiVersion`.

## 3.8. `VkResult` Return Codes

While the core Vulkan API is not designed to capture incorrect usage, some circumstances still require return codes. Commands in Vulkan return their status via return codes that are in one of two categories:

- Successful completion codes are returned when a command needs to communicate success or status information. All successful completion codes are non-negative values.
- Runtime error codes are returned when a command needs to communicate a failure that could only be detected at runtime. All runtime error codes are negative values.

All return codes in Vulkan are reported via VkResult return values. The possible codes are:

```
// Provided by VK_VERSION_1_0
typedef enum VkResult {
    VK_SUCCESS = 0,
    VK_NOT_READY = 1,
    VK_TIMEOUT = 2,
    VK_EVENT_SET = 3,
    VK_EVENT_RESET = 4,
    VK_INCOMPLETE = 5,
    VK_ERROR_OUT_OF_HOST_MEMORY = -1,
    VK_ERROR_OUT_OF_DEVICE_MEMORY = -2,
    VK_ERROR_INITIALIZATION_FAILED = -3,
    VK_ERROR_DEVICE_LOST = -4,
    VK_ERROR_MEMORY_MAP_FAILED = -5,
    VK_ERROR_LAYER_NOT_PRESENT = -6,
    VK_ERROR_EXTENSION_NOT_PRESENT = -7,
```

```
    VK_ERROR_FEATURE_NOT_PRESENT = -8,
    VK_ERROR_INCOMPATIBLE_DRIVER = -9,
    VK_ERROR_TOO_MANY_OBJECTS = -10,
    VK_ERROR_FORMAT_NOT_SUPPORTED = -11,
    VK_ERROR_FRAGMENTED_POOL = -12,
    VK_ERROR_UNKNOWN = -13,
  // Provided by VK_VERSION_1_1
    VK_ERROR_OUT_OF_POOL_MEMORY = -1000069000,
  // Provided by VK_VERSION_1_1
    VK_ERROR_INVALID_EXTERNAL_HANDLE = -1000072003,
    VK_ERROR_FRAGMENTATION = -1000161000,
    VK_ERROR_INVALID_OPAQUE_CAPTURE_ADDRESS = -1000257000,
    VK_PIPELINE_COMPILE_REQUIRED = 1000297000,
  // Provided by VK_KHR_surface
    VK_ERROR_SURFACE_LOST_KHR = -1000000000,
  // Provided by VK_KHR_surface
    VK_ERROR_NATIVE_WINDOW_IN_USE_KHR = -1000000001,
  // Provided by VK_KHR_swapchain
    VK_SUBOPTIMAL_KHR = 1000001003,
  // Provided by VK_KHR_swapchain
    VK_ERROR_OUT_OF_DATE_KHR = -1000001004,
  // Provided by VK_KHR_display_swapchain
    VK_ERROR_INCOMPATIBLE_DISPLAY_KHR = -1000003001,
  // Provided by VK_EXT_debug_report
    VK_ERROR_VALIDATION_FAILED_EXT = -1000011001,
  // Provided by VK_NV_glsl_shader
    VK_ERROR_INVALID_SHADER_NV = -1000012000,
  // Provided by VK_KHR_video_queue
    VK_ERROR_IMAGE_USAGE_NOT_SUPPORTED_KHR = -1000023000,
  // Provided by VK_KHR_video_queue
    VK_ERROR_VIDEO_PICTURE_LAYOUT_NOT_SUPPORTED_KHR = -1000023001,
  // Provided by VK_KHR_video_queue
    VK_ERROR_VIDEO_PROFILE_OPERATION_NOT_SUPPORTED_KHR = -1000023002,
  // Provided by VK_KHR_video_queue
    VK_ERROR_VIDEO_PROFILE_FORMAT_NOT_SUPPORTED_KHR = -1000023003,
  // Provided by VK_KHR_video_queue
    VK_ERROR_VIDEO_PROFILE_CODEC_NOT_SUPPORTED_KHR = -1000023004,
  // Provided by VK_KHR_video_queue
    VK_ERROR_VIDEO_STD_VERSION_NOT_SUPPORTED_KHR = -1000023005,
  // Provided by VK_EXT_image_drm_format_modifier
    VK_ERROR_INVALID_DRM_FORMAT_MODIFIER_PLANE_LAYOUT_EXT = -1000158000,
  // Provided by VK_KHR_global_priority
    VK_ERROR_NOT_PERMITTED_KHR = -1000174001,
  // Provided by VK_EXT_full_screen_exclusive
    VK_ERROR_FULL_SCREEN_EXCLUSIVE_MODE_LOST_EXT = -1000255000,
  // Provided by VK_KHR_deferred_host_operations
    VK_THREAD_IDLE_KHR = 1000268000,
  // Provided by VK_KHR_deferred_host_operations
    VK_THREAD_DONE_KHR = 1000268001,
  // Provided by VK_KHR_deferred_host_operations
    VK_OPERATION_DEFERRED_KHR = 1000268002,
```

```
  // Provided by VK_KHR_deferred_host_operations
    VK_OPERATION_NOT_DEFERRED_KHR = 1000268003,
  // Provided by VK_KHR_video_encode_queue
    VK_ERROR_INVALID_VIDEO_STD_PARAMETERS_KHR = -1000299000,
  // Provided by VK_EXT_image_compression_control
    VK_ERROR_COMPRESSION_EXHAUSTED_EXT = -1000338000,
  // Provided by VK_EXT_shader_object
    VK_INCOMPATIBLE_SHADER_BINARY_EXT = 1000482000,
  // Provided by VK_KHR_pipeline_binary
    VK_PIPELINE_BINARY_MISSING_KHR = 1000483000,
  // Provided by VK_KHR_pipeline_binary
    VK_ERROR_NOT_ENOUGH_SPACE_KHR = -1000483000,
  // Provided by VK_KHR_maintenance1
    VK_ERROR_OUT_OF_POOL_MEMORY_KHR = VK_ERROR_OUT_OF_POOL_MEMORY,
  // Provided by VK_KHR_external_memory
    VK_ERROR_INVALID_EXTERNAL_HANDLE_KHR = VK_ERROR_INVALID_EXTERNAL_HANDLE,
  // Provided by VK_EXT_descriptor_indexing
    VK_ERROR_FRAGMENTATION_EXT = VK_ERROR_FRAGMENTATION,
  // Provided by VK_EXT_global_priority
    VK_ERROR_NOT_PERMITTED_EXT = VK_ERROR_NOT_PERMITTED_KHR,
  // Provided by VK_EXT_buffer_device_address
    VK_ERROR_INVALID_DEVICE_ADDRESS_EXT = VK_ERROR_INVALID_OPAQUE_CAPTURE_ADDRESS,
  // Provided by VK_KHR_buffer_device_address
    VK_ERROR_INVALID_OPAQUE_CAPTURE_ADDRESS_KHR =
VK_ERROR_INVALID_OPAQUE_CAPTURE_ADDRESS,
  // Provided by VK_EXT_pipeline_creation_cache_control
    VK_PIPELINE_COMPILE_REQUIRED_EXT = VK_PIPELINE_COMPILE_REQUIRED,
  // Provided by VK_EXT_pipeline_creation_cache_control
    VK_ERROR_PIPELINE_COMPILE_REQUIRED_EXT = VK_PIPELINE_COMPILE_REQUIRED,
  // Provided by VK_EXT_shader_object
  // VK_ERROR_INCOMPATIBLE_SHADER_BINARY_EXT is a deprecated alias
    VK_ERROR_INCOMPATIBLE_SHADER_BINARY_EXT = VK_INCOMPATIBLE_SHADER_BINARY_EXT,
} VkResult;
```

*Success Codes*

- `VK_SUCCESS` Command successfully completed
- `VK_NOT_READY` A fence or query has not yet completed
- `VK_TIMEOUT` A wait operation has not completed in the specified time
- `VK_EVENT_SET` An event is signaled
- `VK_EVENT_RESET` An event is unsignaled
- `VK_INCOMPLETE` A return array was too small for the result
- `VK_SUBOPTIMAL_KHR` A swapchain no longer matches the surface properties exactly, but **can** still be used to present to the surface successfully.
- `VK_THREAD_IDLE_KHR` A deferred operation is not complete but there is currently no work for this thread to do at the time of this call.
- `VK_THREAD_DONE_KHR` A deferred operation is not complete but there is no work remaining to

assign to additional threads.

- VK_OPERATION_DEFERRED_KHR A deferred operation was requested and at least some of the work was deferred.
- VK_OPERATION_NOT_DEFERRED_KHR A deferred operation was requested and no operations were deferred.
- VK_PIPELINE_COMPILE_REQUIRED A requested pipeline creation would have required compilation, but the application requested compilation to not be performed.
- VK_PIPELINE_BINARY_MISSING_KHR The application attempted to create a pipeline binary by querying an internal cache, but the internal cache entry did not exist.
- VK_ERROR_NOT_ENOUGH_SPACE_KHR The application did not provide enough space to return all the required data.
- VK_INCOMPATIBLE_SHADER_BINARY_EXT The provided binary shader code is not compatible with this device.

| | |
|---|---|
| **NOTE** | In the initial version of the VK_EXT_shader_object extension, this return code was named VK_ERROR_INCOMPATIBLE_SHADER_BINARY_EXT and improperly described as an error code. The name has been changed, but the old name is retained as an alias for compatibility with old code. |

*Error codes*

- VK_ERROR_OUT_OF_HOST_MEMORY A host memory allocation has failed.
- VK_ERROR_OUT_OF_DEVICE_MEMORY A device memory allocation has failed.
- VK_ERROR_INITIALIZATION_FAILED Initialization of an object could not be completed for implementation-specific reasons.
- VK_ERROR_DEVICE_LOST The logical or physical device has been lost. See Lost Device
- VK_ERROR_MEMORY_MAP_FAILED Mapping of a memory object has failed.
- VK_ERROR_LAYER_NOT_PRESENT A requested layer is not present or could not be loaded.
- VK_ERROR_EXTENSION_NOT_PRESENT A requested extension is not supported.
- VK_ERROR_FEATURE_NOT_PRESENT A requested feature is not supported.
- VK_ERROR_INCOMPATIBLE_DRIVER The requested version of Vulkan is not supported by the driver or is otherwise incompatible for implementation-specific reasons.
- VK_ERROR_TOO_MANY_OBJECTS Too many objects of the type have already been created.
- VK_ERROR_FORMAT_NOT_SUPPORTED A requested format is not supported on this device.
- VK_ERROR_FRAGMENTED_POOL A pool allocation has failed due to fragmentation of the pool's memory. This **must** only be returned if no attempt to allocate host or device memory was made to accommodate the new allocation. This **should** be returned in preference to VK_ERROR_OUT_OF_POOL_MEMORY, but only if the implementation is certain that the pool allocation failure was due to fragmentation.
- VK_ERROR_SURFACE_LOST_KHR A surface is no longer available.
- VK_ERROR_NATIVE_WINDOW_IN_USE_KHR The requested window is already in use by Vulkan or

another API in a manner which prevents it from being used again.

- VK_ERROR_OUT_OF_DATE_KHR A surface has changed in such a way that it is no longer compatible with the swapchain, and further presentation requests using the swapchain will fail. Applications **must** query the new surface properties and recreate their swapchain if they wish to continue presenting to the surface.
- VK_ERROR_INCOMPATIBLE_DISPLAY_KHR The display used by a swapchain does not use the same presentable image layout, or is incompatible in a way that prevents sharing an image.
- VK_ERROR_INVALID_SHADER_NV One or more shaders failed to compile or link. More details are reported back to the application via VK_EXT_debug_report if enabled.
- VK_ERROR_OUT_OF_POOL_MEMORY A pool memory allocation has failed. This **must** only be returned if no attempt to allocate host or device memory was made to accommodate the new allocation. If the failure was definitely due to fragmentation of the pool, VK_ERROR_FRAGMENTED_POOL **should** be returned instead.
- VK_ERROR_INVALID_EXTERNAL_HANDLE An external handle is not a valid handle of the specified type.
- VK_ERROR_FRAGMENTATION A descriptor pool creation has failed due to fragmentation.
- VK_ERROR_INVALID_DEVICE_ADDRESS_EXT A buffer creation failed because the requested address is not available.
- VK_ERROR_INVALID_OPAQUE_CAPTURE_ADDRESS A buffer creation or memory allocation failed because the requested address is not available. A shader group handle assignment failed because the requested shader group handle information is no longer valid.
- VK_ERROR_FULL_SCREEN_EXCLUSIVE_MODE_LOST_EXT An operation on a swapchain created with VK_FULL_SCREEN_EXCLUSIVE_APPLICATION_CONTROLLED_EXT failed as it did not have exclusive full-screen access. This **may** occur due to implementation-dependent reasons, outside of the application's control.
- VK_ERROR_VALIDATION_FAILED_EXT A command failed because invalid usage was detected by the implementation or a validation-layer.
- VK_ERROR_COMPRESSION_EXHAUSTED_EXT An image creation failed because internal resources required for compression are exhausted. This **must** only be returned when fixed-rate compression is requested.
- VK_ERROR_IMAGE_USAGE_NOT_SUPPORTED_KHR The requested VkImageUsageFlags are not supported.
- VK_ERROR_VIDEO_PICTURE_LAYOUT_NOT_SUPPORTED_KHR The requested video picture layout is not supported.
- VK_ERROR_VIDEO_PROFILE_OPERATION_NOT_SUPPORTED_KHR A video profile operation specified via VkVideoProfileInfoKHR::videoCodecOperation is not supported.
- VK_ERROR_VIDEO_PROFILE_FORMAT_NOT_SUPPORTED_KHR Format parameters in a requested VkVideoProfileInfoKHR chain are not supported.
- VK_ERROR_VIDEO_PROFILE_CODEC_NOT_SUPPORTED_KHR Codec-specific parameters in a requested VkVideoProfileInfoKHR chain are not supported.
- VK_ERROR_VIDEO_STD_VERSION_NOT_SUPPORTED_KHR The specified video Std header version is not supported.
- VK_ERROR_INVALID_VIDEO_STD_PARAMETERS_KHR The specified Video Std parameters do not adhere to

the syntactic or semantic requirements of the used video compression standard, or values derived from parameters according to the rules defined by the used video compression standard do not adhere to the capabilities of the video compression standard or the implementation.

- `VK_ERROR_NOT_PERMITTED_KHR` The driver implementation has denied a request to acquire a priority above the default priority (`VK_QUEUE_GLOBAL_PRIORITY_MEDIUM_EXT`) because the application does not have sufficient privileges.
- `VK_ERROR_UNKNOWN` An unknown error has occurred; either the application has provided invalid input, or an implementation failure has occurred.

If a command returns a runtime error, unless otherwise specified any output parameters will have undefined contents, except that if the output parameter is a structure with `sType` and `pNext` fields, those fields will be unmodified. Any structures chained from `pNext` will also have undefined contents, except that `sType` and `pNext` will be unmodified.

`VK_ERROR_OUT_OF_*_MEMORY` errors do not modify any currently existing Vulkan objects. Objects that have already been successfully created **can** still be used by the application.

**NOTE** As a general rule, `Free`, `Release`, and `Reset` commands do not return `VK_ERROR_OUT_OF_HOST_MEMORY`, while any other command with a return code **may** return it. Any exceptions from this rule are described for those commands.

`VK_ERROR_UNKNOWN` will be returned by an implementation when an unexpected error occurs that cannot be attributed to valid behavior of the application and implementation. Under these conditions, it **may** be returned from any command returning a VkResult.

**NOTE** `VK_ERROR_UNKNOWN` is not expected to ever be returned if the application behavior is valid, and if the implementation is bug-free. If `VK_ERROR_UNKNOWN` is received, the application should be checked against the latest validation layers to verify correct behavior as much as possible. If no issues are identified it could be an implementation issue, and the implementor should be contacted for support.

Any command returning a VkResult **may** return `VK_ERROR_VALIDATION_FAILED_EXT` if a violation of valid usage is detected, even though commands do not explicitly list this as a possible return code.

Performance-critical commands generally do not have return codes. If a runtime error occurs in such commands, the implementation will defer reporting the error until a specified point. For commands that record into command buffers (`vkCmd*`) runtime errors are reported by `vkEndCommandBuffer`.

## 3.9. Numeric Representation and Computation

Implementations normally perform computations in floating-point, and **must** meet the range and precision requirements defined under “Floating-Point Computation” below.

These requirements only apply to computations performed in Vulkan operations outside of shader execution, such as texture image specification and sampling, and per-fragment operations. Range

and precision requirements during shader execution differ and are specified by the Precision and Operation of SPIR-V Instructions section.

In some cases, the representation and/or precision of operations is implicitly limited by the specified format of vertex or texel data consumed by Vulkan. Specific floating-point formats are described later in this section.

### 3.9.1. Floating-Point Computation

Most floating-point computation is performed in SPIR-V shader modules. The properties of computation within shaders are constrained as defined by the Precision and Operation of SPIR-V Instructions section.

Some floating-point computation is performed outside of shaders, such as viewport and depth range calculations. For these computations, we do not specify how floating-point numbers are to be represented, or the details of how operations on them are performed, but only place minimal requirements on representation and precision as described in the remainder of this section.

We require simply that numbers' floating-point parts contain enough bits and that their exponent fields are large enough so that individual results of floating-point operations are accurate to about 1 part in $10^5$. The maximum representable magnitude for all floating-point values **must** be at least $2^{32}$.

$x \times 0 = 0 \times x = 0$ for any non-infinite and non-NaN x.

$1 \times x = x \times 1 = x$.

$x + 0 = 0 + x = x$.

$0^0 = 1$.

Occasionally, further requirements will be specified. Most single-precision floating-point formats meet these requirements.

The special values Inf and -Inf encode values with magnitudes too large to be represented; the special value NaN encodes "Not A Number" values resulting from undefined arithmetic operations such as 0 / 0. Implementations **may** support Inf and NaN in their floating-point computations. Any computation which does not support either Inf or NaN, for which that value is an input or output will yield an undefined value.

### 3.9.2. Floating-Point Format Conversions

When a value is converted to a defined floating-point representation, finite values falling between two representable finite values are rounded to one or the other. The rounding mode is not defined. Finite values whose magnitude is larger than that of any representable finite value may be rounded either to the closest representable finite value or to the appropriately signed infinity. For unsigned

destination formats any negative values are converted to zero. Positive infinity is converted to positive infinity; negative infinity is converted to negative infinity in signed formats and to zero in unsigned formats; and any NaN is converted to a NaN.

### 3.9.3. 16-Bit Floating-Point Numbers

16-bit floating-point numbers are defined in the “16-bit floating-point numbers” section of the Khronos Data Format Specification.

### 3.9.4. Unsigned 11-Bit Floating-Point Numbers

Unsigned 11-bit floating-point numbers are defined in the “Unsigned 11-bit floating-point numbers” section of the Khronos Data Format Specification.

### 3.9.5. Unsigned 10-Bit Floating-Point Numbers

Unsigned 10-bit floating-point numbers are defined in the “Unsigned 10-bit floating-point numbers” section of the Khronos Data Format Specification.

### 3.9.6. General Requirements

Any representable floating-point value in the appropriate format is legal as input to a Vulkan command that requires floating-point data. The result of providing a value that is not a floating-point number to such a command is unspecified, but **must** not lead to Vulkan interruption or termination. For example, providing a negative zero (where applicable) or a denormalized number to a Vulkan command **must** yield deterministic results, while providing a NaN or Inf yields unspecified results.

Some calculations require division. In such cases (including implied divisions performed by vector normalization), division by zero produces an unspecified result but **must** not lead to Vulkan interruption or termination.

## 3.10. Fixed-Point Data Conversions

When generic vertex attributes and pixel color or depth *components* are represented as integers, they are often (but not always) considered to be *normalized*. Normalized integer values are treated specially when being converted to and from floating-point values, and are usually referred to as *normalized fixed-point*.

In the remainder of this section, b denotes the bit width of the fixed-point integer representation. When the integer is one of the types defined by the API, b is the bit width of that type. When the integer comes from an image containing color or depth component texels, b is the number of bits allocated to that component in its specified image format.

The signed and unsigned fixed-point representations are assumed to be b-bit binary two’s-complement integers and binary unsigned integers, respectively.

### 3.10.1. Conversion From Normalized Fixed-Point to Floating-Point

Unsigned normalized fixed-point integers represent numbers in the range [0,1]. The conversion from an unsigned normalized fixed-point value c to the corresponding floating-point value f is defined as

$$f = \frac{c}{2^b - 1}$$

Signed normalized fixed-point integers represent numbers in the range [-1,1]. The conversion from a signed normalized fixed-point value c to the corresponding floating-point value f is performed using

$$f = \max\left(\frac{c}{2^{b-1} - 1}, -1.0\right)$$

Only the range $[-2^{b-1} + 1, 2^{b-1} - 1]$ is used to represent signed fixed-point values in the range [-1,1]. For example, if b = 8, then the integer value -127 corresponds to -1.0 and the value 127 corresponds to 1.0. This equation is used everywhere that signed normalized fixed-point values are converted to floating-point.

Note that while zero is exactly expressible in this representation, one value (-128 in the example) is outside the representable range, and implementations **must** clamp it to -1.0. Where the value is subject to further processing by the implementation, e.g. during texture filtering, values less than -1.0 **may** be used but the result **must** be clamped before the value is returned to shaders.

### 3.10.2. Conversion From Floating-Point to Normalized Fixed-Point

The conversion from a floating-point value f to the corresponding unsigned normalized fixed-point value c is defined by first clamping f to the range [0,1], then computing

$$c = \text{convertFloatToUint}(f \times (2^b - 1), b)$$

where convertFloatToUint(r,b) returns one of the two unsigned binary integer values with exactly b bits which are closest to the floating-point value r. Implementations **should** round to nearest. If r is equal to an integer, then that integer value **must** be returned. In particular, if f is equal to 0.0 or 1.0, then c **must** be assigned 0 or $2^b - 1$, respectively.

The conversion from a floating-point value f to the corresponding signed normalized fixed-point value c is performed by clamping f to the range [-1,1], then computing

$$c = \text{convertFloatToInt}(f \times (2^{b-1} - 1), b)$$

where convertFloatToInt(r,b) returns one of the two signed two's-complement binary integer values with exactly b bits which are closest to the floating-point value r. Implementations **should** round to nearest. If r is equal to an integer, then that integer value **must** be returned. In particular, if f is equal to -1.0, 0.0, or 1.0, then c **must** be assigned $-(2^{b-1} - 1)$, 0, or $2^{b-1} - 1$, respectively.

This equation is used everywhere that floating-point values are converted to signed normalized

fixed-point.

## 3.11. String Representation

Strings passed into and returned from Vulkan API commands are usually defined to be null-terminated and UTF-8 encoded.

| | |
|---|---|
| **NOTE** | Exceptions to this rule exist only when strings are defined or used by operating system APIs where that OS has a different convention. For example, VkExportMemoryWin32HandleInfoKHR::`name` is a null-terminated UTF-16 encoded string used in conjunction with Windows handles. |

When a UTF-8 string is **returned from** a Vulkan API query, it is returned in a fixed-length buffer of C `char`. For example, a string returned in VkPhysicalDeviceProperties::`deviceName` has maximum length `VK_MAX_PHYSICAL_DEVICE_NAME_SIZE`, and a string returned in VkExtensionProperties::`extensionName` has maximum length `VK_MAX_EXTENSION_NAME_SIZE`. The string, **including** its null terminator, will always fit completely within this buffer. If the string is shorter than the buffer size, the contents of `char` in the buffer following the null terminator are undefined.

When a UTF-8 string is **passed into** a Vulkan API, such as VkDeviceCreateInfo::`ppEnabledExtensionNames`, there is no explicit limit on the length of that string. However, the string **must** contain a valid UTF-8 encoded string and **must** be null-terminated.

## 3.12. Common Object Types

Some types of Vulkan objects are used in many different structures and command parameters, and are described here. These types include *offsets*, *extents*, and *rectangles*.

### 3.12.1. Offsets

Offsets are used to describe a pixel location within an image or framebuffer, as an (x,y) location for two-dimensional images, or an (x,y,z) location for three-dimensional images.

A two-dimensional offset is defined by the structure:

```
// Provided by VK_VERSION_1_0
typedef struct VkOffset2D {
    int32_t    x;
    int32_t    y;
} VkOffset2D;
```

- `x` is the x offset.
- `y` is the y offset.

A three-dimensional offset is defined by the structure:

```
// Provided by VK_VERSION_1_0
typedef struct VkOffset3D {
    int32_t    x;
    int32_t    y;
    int32_t    z;
} VkOffset3D;
```

- `x` is the x offset.
- `y` is the y offset.
- `z` is the z offset.

### 3.12.2. Extents

Extents are used to describe the size of a rectangular region of pixels within an image or framebuffer, as (width,height) for two-dimensional images, or as (width,height,depth) for three-dimensional images.

A two-dimensional extent is defined by the structure:

```
// Provided by VK_VERSION_1_0
typedef struct VkExtent2D {
    uint32_t    width;
    uint32_t    height;
} VkExtent2D;
```

- `width` is the width of the extent.
- `height` is the height of the extent.

A three-dimensional extent is defined by the structure:

```
// Provided by VK_VERSION_1_0
typedef struct VkExtent3D {
    uint32_t    width;
    uint32_t    height;
    uint32_t    depth;
} VkExtent3D;
```

- `width` is the width of the extent.
- `height` is the height of the extent.
- `depth` is the depth of the extent.

### 3.12.3. Rectangles

Rectangles are used to describe a specified rectangular region of pixels within an image or framebuffer. Rectangles include both an offset and an extent of the same dimensionality, as

described above. Two-dimensional rectangles are defined by the structure

```
// Provided by VK_VERSION_1_0
typedef struct VkRect2D {
    VkOffset2D    offset;
    VkExtent2D    extent;
} VkRect2D;
```

- `offset` is a VkOffset2D specifying the rectangle offset.
- `extent` is a VkExtent2D specifying the rectangle extent.

### 3.12.4. Structure Types

Each value corresponds to a particular structure with a `sType` member with a matching name. As a general rule, the name of each VkStructureType value is obtained by taking the name of the structure, stripping the leading `Vk`, prefixing each capital letter with `_`, converting the entire resulting string to upper case, and prefixing it with `VK_STRUCTURE_TYPE_`. For example, structures of type VkImageCreateInfo correspond to a VkStructureType value of `VK_STRUCTURE_TYPE_IMAGE_CREATE_INFO`, and thus a structure of this type **must** have its `sType` member set to this value before it is passed to the API.

The values `VK_STRUCTURE_TYPE_LOADER_INSTANCE_CREATE_INFO` and `VK_STRUCTURE_TYPE_LOADER_DEVICE_CREATE_INFO` are reserved for internal use by the loader, and do not have corresponding Vulkan structures in this Specification.

Structure types supported by the Vulkan API include:

```
// Provided by VK_VERSION_1_0
typedef enum VkStructureType {
    VK_STRUCTURE_TYPE_APPLICATION_INFO = 0,
    VK_STRUCTURE_TYPE_INSTANCE_CREATE_INFO = 1,
    VK_STRUCTURE_TYPE_DEVICE_QUEUE_CREATE_INFO = 2,
    VK_STRUCTURE_TYPE_DEVICE_CREATE_INFO = 3,
    VK_STRUCTURE_TYPE_SUBMIT_INFO = 4,
    VK_STRUCTURE_TYPE_MEMORY_ALLOCATE_INFO = 5,
    VK_STRUCTURE_TYPE_MAPPED_MEMORY_RANGE = 6,
    VK_STRUCTURE_TYPE_BIND_SPARSE_INFO = 7,
    VK_STRUCTURE_TYPE_FENCE_CREATE_INFO = 8,
    VK_STRUCTURE_TYPE_SEMAPHORE_CREATE_INFO = 9,
    VK_STRUCTURE_TYPE_EVENT_CREATE_INFO = 10,
    VK_STRUCTURE_TYPE_QUERY_POOL_CREATE_INFO = 11,
    VK_STRUCTURE_TYPE_BUFFER_CREATE_INFO = 12,
    VK_STRUCTURE_TYPE_BUFFER_VIEW_CREATE_INFO = 13,
    VK_STRUCTURE_TYPE_IMAGE_CREATE_INFO = 14,
    VK_STRUCTURE_TYPE_IMAGE_VIEW_CREATE_INFO = 15,
    VK_STRUCTURE_TYPE_SHADER_MODULE_CREATE_INFO = 16,
    VK_STRUCTURE_TYPE_PIPELINE_CACHE_CREATE_INFO = 17,
    VK_STRUCTURE_TYPE_PIPELINE_SHADER_STAGE_CREATE_INFO = 18,
```

```
    VK_STRUCTURE_TYPE_PIPELINE_VERTEX_INPUT_STATE_CREATE_INFO = 19,
    VK_STRUCTURE_TYPE_PIPELINE_INPUT_ASSEMBLY_STATE_CREATE_INFO = 20,
    VK_STRUCTURE_TYPE_PIPELINE_TESSELLATION_STATE_CREATE_INFO = 21,
    VK_STRUCTURE_TYPE_PIPELINE_VIEWPORT_STATE_CREATE_INFO = 22,
    VK_STRUCTURE_TYPE_PIPELINE_RASTERIZATION_STATE_CREATE_INFO = 23,
    VK_STRUCTURE_TYPE_PIPELINE_MULTISAMPLE_STATE_CREATE_INFO = 24,
    VK_STRUCTURE_TYPE_PIPELINE_DEPTH_STENCIL_STATE_CREATE_INFO = 25,
    VK_STRUCTURE_TYPE_PIPELINE_COLOR_BLEND_STATE_CREATE_INFO = 26,
    VK_STRUCTURE_TYPE_PIPELINE_DYNAMIC_STATE_CREATE_INFO = 27,
    VK_STRUCTURE_TYPE_GRAPHICS_PIPELINE_CREATE_INFO = 28,
    VK_STRUCTURE_TYPE_COMPUTE_PIPELINE_CREATE_INFO = 29,
    VK_STRUCTURE_TYPE_PIPELINE_LAYOUT_CREATE_INFO = 30,
    VK_STRUCTURE_TYPE_SAMPLER_CREATE_INFO = 31,
    VK_STRUCTURE_TYPE_DESCRIPTOR_SET_LAYOUT_CREATE_INFO = 32,
    VK_STRUCTURE_TYPE_DESCRIPTOR_POOL_CREATE_INFO = 33,
    VK_STRUCTURE_TYPE_DESCRIPTOR_SET_ALLOCATE_INFO = 34,
    VK_STRUCTURE_TYPE_WRITE_DESCRIPTOR_SET = 35,
    VK_STRUCTURE_TYPE_COPY_DESCRIPTOR_SET = 36,
    VK_STRUCTURE_TYPE_FRAMEBUFFER_CREATE_INFO = 37,
    VK_STRUCTURE_TYPE_RENDER_PASS_CREATE_INFO = 38,
    VK_STRUCTURE_TYPE_COMMAND_POOL_CREATE_INFO = 39,
    VK_STRUCTURE_TYPE_COMMAND_BUFFER_ALLOCATE_INFO = 40,
    VK_STRUCTURE_TYPE_COMMAND_BUFFER_INHERITANCE_INFO = 41,
    VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO = 42,
    VK_STRUCTURE_TYPE_RENDER_PASS_BEGIN_INFO = 43,
    VK_STRUCTURE_TYPE_BUFFER_MEMORY_BARRIER = 44,
    VK_STRUCTURE_TYPE_IMAGE_MEMORY_BARRIER = 45,
    VK_STRUCTURE_TYPE_MEMORY_BARRIER = 46,
    VK_STRUCTURE_TYPE_LOADER_INSTANCE_CREATE_INFO = 47,
    VK_STRUCTURE_TYPE_LOADER_DEVICE_CREATE_INFO = 48,
  // Provided by VK_VERSION_1_1
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SUBGROUP_PROPERTIES = 1000094000,
  // Provided by VK_VERSION_1_1
    VK_STRUCTURE_TYPE_BIND_BUFFER_MEMORY_INFO = 1000157000,
  // Provided by VK_VERSION_1_1
    VK_STRUCTURE_TYPE_BIND_IMAGE_MEMORY_INFO = 1000157001,
  // Provided by VK_VERSION_1_1
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_16BIT_STORAGE_FEATURES = 1000083000,
  // Provided by VK_VERSION_1_1
    VK_STRUCTURE_TYPE_MEMORY_DEDICATED_REQUIREMENTS = 1000127000,
  // Provided by VK_VERSION_1_1
    VK_STRUCTURE_TYPE_MEMORY_DEDICATED_ALLOCATE_INFO = 1000127001,
  // Provided by VK_VERSION_1_1
    VK_STRUCTURE_TYPE_MEMORY_ALLOCATE_FLAGS_INFO = 1000060000,
  // Provided by VK_VERSION_1_1
    VK_STRUCTURE_TYPE_DEVICE_GROUP_RENDER_PASS_BEGIN_INFO = 1000060003,
  // Provided by VK_VERSION_1_1
    VK_STRUCTURE_TYPE_DEVICE_GROUP_COMMAND_BUFFER_BEGIN_INFO = 1000060004,
  // Provided by VK_VERSION_1_1
    VK_STRUCTURE_TYPE_DEVICE_GROUP_SUBMIT_INFO = 1000060005,
  // Provided by VK_VERSION_1_1
```

```
    VK_STRUCTURE_TYPE_DEVICE_GROUP_BIND_SPARSE_INFO = 1000060006,
  // Provided by VK_VERSION_1_1
    VK_STRUCTURE_TYPE_BIND_BUFFER_MEMORY_DEVICE_GROUP_INFO = 1000060013,
  // Provided by VK_VERSION_1_1
    VK_STRUCTURE_TYPE_BIND_IMAGE_MEMORY_DEVICE_GROUP_INFO = 1000060014,
  // Provided by VK_VERSION_1_1
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_GROUP_PROPERTIES = 1000070000,
  // Provided by VK_VERSION_1_1
    VK_STRUCTURE_TYPE_DEVICE_GROUP_DEVICE_CREATE_INFO = 1000070001,
  // Provided by VK_VERSION_1_1
    VK_STRUCTURE_TYPE_BUFFER_MEMORY_REQUIREMENTS_INFO_2 = 1000146000,
  // Provided by VK_VERSION_1_1
    VK_STRUCTURE_TYPE_IMAGE_MEMORY_REQUIREMENTS_INFO_2 = 1000146001,
  // Provided by VK_VERSION_1_1
    VK_STRUCTURE_TYPE_IMAGE_SPARSE_MEMORY_REQUIREMENTS_INFO_2 = 1000146002,
  // Provided by VK_VERSION_1_1
    VK_STRUCTURE_TYPE_MEMORY_REQUIREMENTS_2 = 1000146003,
  // Provided by VK_VERSION_1_1
    VK_STRUCTURE_TYPE_SPARSE_IMAGE_MEMORY_REQUIREMENTS_2 = 1000146004,
  // Provided by VK_VERSION_1_1
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_FEATURES_2 = 1000059000,
  // Provided by VK_VERSION_1_1
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PROPERTIES_2 = 1000059001,
  // Provided by VK_VERSION_1_1
    VK_STRUCTURE_TYPE_FORMAT_PROPERTIES_2 = 1000059002,
  // Provided by VK_VERSION_1_1
    VK_STRUCTURE_TYPE_IMAGE_FORMAT_PROPERTIES_2 = 1000059003,
  // Provided by VK_VERSION_1_1
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_IMAGE_FORMAT_INFO_2 = 1000059004,
  // Provided by VK_VERSION_1_1
    VK_STRUCTURE_TYPE_QUEUE_FAMILY_PROPERTIES_2 = 1000059005,
  // Provided by VK_VERSION_1_1
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MEMORY_PROPERTIES_2 = 1000059006,
  // Provided by VK_VERSION_1_1
    VK_STRUCTURE_TYPE_SPARSE_IMAGE_FORMAT_PROPERTIES_2 = 1000059007,
  // Provided by VK_VERSION_1_1
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SPARSE_IMAGE_FORMAT_INFO_2 = 1000059008,
  // Provided by VK_VERSION_1_1
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_POINT_CLIPPING_PROPERTIES = 1000117000,
  // Provided by VK_VERSION_1_1
    VK_STRUCTURE_TYPE_RENDER_PASS_INPUT_ATTACHMENT_ASPECT_CREATE_INFO = 1000117001,
  // Provided by VK_VERSION_1_1
    VK_STRUCTURE_TYPE_IMAGE_VIEW_USAGE_CREATE_INFO = 1000117002,
  // Provided by VK_VERSION_1_1
    VK_STRUCTURE_TYPE_PIPELINE_TESSELLATION_DOMAIN_ORIGIN_STATE_CREATE_INFO =
1000117003,
  // Provided by VK_VERSION_1_1
    VK_STRUCTURE_TYPE_RENDER_PASS_MULTIVIEW_CREATE_INFO = 1000053000,
  // Provided by VK_VERSION_1_1
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MULTIVIEW_FEATURES = 1000053001,
  // Provided by VK_VERSION_1_1
```

```
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MULTIVIEW_PROPERTIES = 1000053002,
  // Provided by VK_VERSION_1_1
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_VARIABLE_POINTERS_FEATURES = 1000120000,
  // Provided by VK_VERSION_1_1
    VK_STRUCTURE_TYPE_PROTECTED_SUBMIT_INFO = 1000145000,
  // Provided by VK_VERSION_1_1
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PROTECTED_MEMORY_FEATURES = 1000145001,
  // Provided by VK_VERSION_1_1
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PROTECTED_MEMORY_PROPERTIES = 1000145002,
  // Provided by VK_VERSION_1_1
    VK_STRUCTURE_TYPE_DEVICE_QUEUE_INFO_2 = 1000145003,
  // Provided by VK_VERSION_1_1
    VK_STRUCTURE_TYPE_SAMPLER_YCBCR_CONVERSION_CREATE_INFO = 1000156000,
  // Provided by VK_VERSION_1_1
    VK_STRUCTURE_TYPE_SAMPLER_YCBCR_CONVERSION_INFO = 1000156001,
  // Provided by VK_VERSION_1_1
    VK_STRUCTURE_TYPE_BIND_IMAGE_PLANE_MEMORY_INFO = 1000156002,
  // Provided by VK_VERSION_1_1
    VK_STRUCTURE_TYPE_IMAGE_PLANE_MEMORY_REQUIREMENTS_INFO = 1000156003,
  // Provided by VK_VERSION_1_1
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SAMPLER_YCBCR_CONVERSION_FEATURES = 1000156004,
  // Provided by VK_VERSION_1_1
    VK_STRUCTURE_TYPE_SAMPLER_YCBCR_CONVERSION_IMAGE_FORMAT_PROPERTIES = 1000156005,
  // Provided by VK_VERSION_1_1
    VK_STRUCTURE_TYPE_DESCRIPTOR_UPDATE_TEMPLATE_CREATE_INFO = 1000085000,
  // Provided by VK_VERSION_1_1
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTERNAL_IMAGE_FORMAT_INFO = 1000071000,
  // Provided by VK_VERSION_1_1
    VK_STRUCTURE_TYPE_EXTERNAL_IMAGE_FORMAT_PROPERTIES = 1000071001,
  // Provided by VK_VERSION_1_1
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTERNAL_BUFFER_INFO = 1000071002,
  // Provided by VK_VERSION_1_1
    VK_STRUCTURE_TYPE_EXTERNAL_BUFFER_PROPERTIES = 1000071003,
  // Provided by VK_VERSION_1_1
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_ID_PROPERTIES = 1000071004,
  // Provided by VK_VERSION_1_1
    VK_STRUCTURE_TYPE_EXTERNAL_MEMORY_BUFFER_CREATE_INFO = 1000072000,
  // Provided by VK_VERSION_1_1
    VK_STRUCTURE_TYPE_EXTERNAL_MEMORY_IMAGE_CREATE_INFO = 1000072001,
  // Provided by VK_VERSION_1_1
    VK_STRUCTURE_TYPE_EXPORT_MEMORY_ALLOCATE_INFO = 1000072002,
  // Provided by VK_VERSION_1_1
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTERNAL_FENCE_INFO = 1000112000,
  // Provided by VK_VERSION_1_1
    VK_STRUCTURE_TYPE_EXTERNAL_FENCE_PROPERTIES = 1000112001,
  // Provided by VK_VERSION_1_1
    VK_STRUCTURE_TYPE_EXPORT_FENCE_CREATE_INFO = 1000113000,
  // Provided by VK_VERSION_1_1
    VK_STRUCTURE_TYPE_EXPORT_SEMAPHORE_CREATE_INFO = 1000077000,
  // Provided by VK_VERSION_1_1
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTERNAL_SEMAPHORE_INFO = 1000076000,
```

```
  // Provided by VK_VERSION_1_1
    VK_STRUCTURE_TYPE_EXTERNAL_SEMAPHORE_PROPERTIES = 1000076001,
  // Provided by VK_VERSION_1_1
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MAINTENANCE_3_PROPERTIES = 1000168000,
  // Provided by VK_VERSION_1_1
    VK_STRUCTURE_TYPE_DESCRIPTOR_SET_LAYOUT_SUPPORT = 1000168001,
  // Provided by VK_VERSION_1_1
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_DRAW_PARAMETERS_FEATURES = 1000063000,
    VK_STRUCTURE_TYPE_IMAGE_FORMAT_LIST_CREATE_INFO = 1000147000,
    VK_STRUCTURE_TYPE_ATTACHMENT_DESCRIPTION_2 = 1000109000,
    VK_STRUCTURE_TYPE_ATTACHMENT_REFERENCE_2 = 1000109001,
    VK_STRUCTURE_TYPE_SUBPASS_DESCRIPTION_2 = 1000109002,
    VK_STRUCTURE_TYPE_SUBPASS_DEPENDENCY_2 = 1000109003,
    VK_STRUCTURE_TYPE_RENDER_PASS_CREATE_INFO_2 = 1000109004,
    VK_STRUCTURE_TYPE_SUBPASS_BEGIN_INFO = 1000109005,
    VK_STRUCTURE_TYPE_SUBPASS_END_INFO = 1000109006,
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_8BIT_STORAGE_FEATURES = 1000177000,
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DRIVER_PROPERTIES = 1000196000,
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_ATOMIC_INT64_FEATURES = 1000180000,
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_FLOAT16_INT8_FEATURES = 1000082000,
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_FLOAT_CONTROLS_PROPERTIES = 1000197000,
    VK_STRUCTURE_TYPE_DESCRIPTOR_SET_LAYOUT_BINDING_FLAGS_CREATE_INFO = 1000161000,
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DESCRIPTOR_INDEXING_FEATURES = 1000161001,
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DESCRIPTOR_INDEXING_PROPERTIES = 1000161002,
    VK_STRUCTURE_TYPE_DESCRIPTOR_SET_VARIABLE_DESCRIPTOR_COUNT_ALLOCATE_INFO =
1000161003,
    VK_STRUCTURE_TYPE_DESCRIPTOR_SET_VARIABLE_DESCRIPTOR_COUNT_LAYOUT_SUPPORT =
1000161004,
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DEPTH_STENCIL_RESOLVE_PROPERTIES = 1000199000,
    VK_STRUCTURE_TYPE_SUBPASS_DESCRIPTION_DEPTH_STENCIL_RESOLVE = 1000199001,
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SCALAR_BLOCK_LAYOUT_FEATURES = 1000221000,
    VK_STRUCTURE_TYPE_IMAGE_STENCIL_USAGE_CREATE_INFO = 1000246000,
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SAMPLER_FILTER_MINMAX_PROPERTIES = 1000130000,
    VK_STRUCTURE_TYPE_SAMPLER_REDUCTION_MODE_CREATE_INFO = 1000130001,
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_VULKAN_MEMORY_MODEL_FEATURES = 1000211000,
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_IMAGELESS_FRAMEBUFFER_FEATURES = 1000108000,
    VK_STRUCTURE_TYPE_FRAMEBUFFER_ATTACHMENTS_CREATE_INFO = 1000108001,
    VK_STRUCTURE_TYPE_FRAMEBUFFER_ATTACHMENT_IMAGE_INFO = 1000108002,
    VK_STRUCTURE_TYPE_RENDER_PASS_ATTACHMENT_BEGIN_INFO = 1000108003,
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_UNIFORM_BUFFER_STANDARD_LAYOUT_FEATURES =
1000253000,
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_SUBGROUP_EXTENDED_TYPES_FEATURES =
1000175000,
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SEPARATE_DEPTH_STENCIL_LAYOUTS_FEATURES =
1000241000,
    VK_STRUCTURE_TYPE_ATTACHMENT_REFERENCE_STENCIL_LAYOUT = 1000241001,
    VK_STRUCTURE_TYPE_ATTACHMENT_DESCRIPTION_STENCIL_LAYOUT = 1000241002,
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_HOST_QUERY_RESET_FEATURES = 1000261000,
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_TIMELINE_SEMAPHORE_FEATURES = 1000207000,
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_TIMELINE_SEMAPHORE_PROPERTIES = 1000207001,
    VK_STRUCTURE_TYPE_SEMAPHORE_TYPE_CREATE_INFO = 1000207002,
```

```
    VK_STRUCTURE_TYPE_TIMELINE_SEMAPHORE_SUBMIT_INFO = 1000207003,
    VK_STRUCTURE_TYPE_SEMAPHORE_WAIT_INFO = 1000207004,
    VK_STRUCTURE_TYPE_SEMAPHORE_SIGNAL_INFO = 1000207005,
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_BUFFER_DEVICE_ADDRESS_FEATURES = 1000257000,
    VK_STRUCTURE_TYPE_BUFFER_DEVICE_ADDRESS_INFO = 1000244001,
    VK_STRUCTURE_TYPE_BUFFER_OPAQUE_CAPTURE_ADDRESS_CREATE_INFO = 1000257002,
    VK_STRUCTURE_TYPE_MEMORY_OPAQUE_CAPTURE_ADDRESS_ALLOCATE_INFO = 1000257003,
    VK_STRUCTURE_TYPE_DEVICE_MEMORY_OPAQUE_CAPTURE_ADDRESS_INFO = 1000257004,
    VK_STRUCTURE_TYPE_PIPELINE_CREATION_FEEDBACK_CREATE_INFO = 1000192000,
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_TERMINATE_INVOCATION_FEATURES =
1000215000,
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_TOOL_PROPERTIES = 1000245000,
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_DEMOTE_TO_HELPER_INVOCATION_FEATURES =
1000276000,
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PRIVATE_DATA_FEATURES = 1000295000,
    VK_STRUCTURE_TYPE_DEVICE_PRIVATE_DATA_CREATE_INFO = 1000295001,
    VK_STRUCTURE_TYPE_PRIVATE_DATA_SLOT_CREATE_INFO = 1000295002,
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PIPELINE_CREATION_CACHE_CONTROL_FEATURES =
1000297000,
    VK_STRUCTURE_TYPE_MEMORY_BARRIER_2 = 1000314000,
    VK_STRUCTURE_TYPE_BUFFER_MEMORY_BARRIER_2 = 1000314001,
    VK_STRUCTURE_TYPE_IMAGE_MEMORY_BARRIER_2 = 1000314002,
    VK_STRUCTURE_TYPE_DEPENDENCY_INFO = 1000314003,
    VK_STRUCTURE_TYPE_SUBMIT_INFO_2 = 1000314004,
    VK_STRUCTURE_TYPE_SEMAPHORE_SUBMIT_INFO = 1000314005,
    VK_STRUCTURE_TYPE_COMMAND_BUFFER_SUBMIT_INFO = 1000314006,
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SYNCHRONIZATION_2_FEATURES = 1000314007,
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_ZERO_INITIALIZE_WORKGROUP_MEMORY_FEATURES =
1000325000,
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_IMAGE_ROBUSTNESS_FEATURES = 1000335000,
    VK_STRUCTURE_TYPE_COPY_BUFFER_INFO_2 = 1000337000,
    VK_STRUCTURE_TYPE_COPY_IMAGE_INFO_2 = 1000337001,
    VK_STRUCTURE_TYPE_COPY_BUFFER_TO_IMAGE_INFO_2 = 1000337002,
    VK_STRUCTURE_TYPE_COPY_IMAGE_TO_BUFFER_INFO_2 = 1000337003,
    VK_STRUCTURE_TYPE_BLIT_IMAGE_INFO_2 = 1000337004,
    VK_STRUCTURE_TYPE_RESOLVE_IMAGE_INFO_2 = 1000337005,
    VK_STRUCTURE_TYPE_BUFFER_COPY_2 = 1000337006,
    VK_STRUCTURE_TYPE_IMAGE_COPY_2 = 1000337007,
    VK_STRUCTURE_TYPE_IMAGE_BLIT_2 = 1000337008,
    VK_STRUCTURE_TYPE_BUFFER_IMAGE_COPY_2 = 1000337009,
    VK_STRUCTURE_TYPE_IMAGE_RESOLVE_2 = 1000337010,
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SUBGROUP_SIZE_CONTROL_PROPERTIES = 1000225000,
    VK_STRUCTURE_TYPE_PIPELINE_SHADER_STAGE_REQUIRED_SUBGROUP_SIZE_CREATE_INFO =
1000225001,
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SUBGROUP_SIZE_CONTROL_FEATURES = 1000225002,
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_INLINE_UNIFORM_BLOCK_FEATURES = 1000138000,
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_INLINE_UNIFORM_BLOCK_PROPERTIES = 1000138001,
    VK_STRUCTURE_TYPE_WRITE_DESCRIPTOR_SET_INLINE_UNIFORM_BLOCK = 1000138002,
    VK_STRUCTURE_TYPE_DESCRIPTOR_POOL_INLINE_UNIFORM_BLOCK_CREATE_INFO = 1000138003,
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_TEXTURE_COMPRESSION_ASTC_HDR_FEATURES =
1000066000,
```

```
    VK_STRUCTURE_TYPE_RENDERING_INFO = 1000044000,
    VK_STRUCTURE_TYPE_RENDERING_ATTACHMENT_INFO = 1000044001,
    VK_STRUCTURE_TYPE_PIPELINE_RENDERING_CREATE_INFO = 1000044002,
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DYNAMIC_RENDERING_FEATURES = 1000044003,
    VK_STRUCTURE_TYPE_COMMAND_BUFFER_INHERITANCE_RENDERING_INFO = 1000044004,
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_INTEGER_DOT_PRODUCT_FEATURES =
1000280000,
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_INTEGER_DOT_PRODUCT_PROPERTIES =
1000280001,
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_TEXEL_BUFFER_ALIGNMENT_PROPERTIES = 1000281001,
    VK_STRUCTURE_TYPE_FORMAT_PROPERTIES_3 = 1000360000,
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MAINTENANCE_4_FEATURES = 1000413000,
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MAINTENANCE_4_PROPERTIES = 1000413001,
    VK_STRUCTURE_TYPE_DEVICE_BUFFER_MEMORY_REQUIREMENTS = 1000413002,
    VK_STRUCTURE_TYPE_DEVICE_IMAGE_MEMORY_REQUIREMENTS = 1000413003,
  // Provided by VK_KHR_swapchain
    VK_STRUCTURE_TYPE_SWAPCHAIN_CREATE_INFO_KHR = 1000001000,
  // Provided by VK_KHR_swapchain
    VK_STRUCTURE_TYPE_PRESENT_INFO_KHR = 1000001001,
  // Provided by VK_VERSION_1_1 with VK_KHR_swapchain, VK_KHR_device_group with
VK_KHR_surface
    VK_STRUCTURE_TYPE_DEVICE_GROUP_PRESENT_CAPABILITIES_KHR = 1000060007,
  // Provided by VK_VERSION_1_1 with VK_KHR_swapchain, VK_KHR_device_group with
VK_KHR_swapchain
    VK_STRUCTURE_TYPE_IMAGE_SWAPCHAIN_CREATE_INFO_KHR = 1000060008,
  // Provided by VK_VERSION_1_1 with VK_KHR_swapchain, VK_KHR_device_group with
VK_KHR_swapchain
    VK_STRUCTURE_TYPE_BIND_IMAGE_MEMORY_SWAPCHAIN_INFO_KHR = 1000060009,
  // Provided by VK_VERSION_1_1 with VK_KHR_swapchain, VK_KHR_device_group with
VK_KHR_swapchain
    VK_STRUCTURE_TYPE_ACQUIRE_NEXT_IMAGE_INFO_KHR = 1000060010,
  // Provided by VK_VERSION_1_1 with VK_KHR_swapchain, VK_KHR_device_group with
VK_KHR_swapchain
    VK_STRUCTURE_TYPE_DEVICE_GROUP_PRESENT_INFO_KHR = 1000060011,
  // Provided by VK_VERSION_1_1 with VK_KHR_swapchain, VK_KHR_device_group with
VK_KHR_swapchain
    VK_STRUCTURE_TYPE_DEVICE_GROUP_SWAPCHAIN_CREATE_INFO_KHR = 1000060012,
  // Provided by VK_KHR_display
    VK_STRUCTURE_TYPE_DISPLAY_MODE_CREATE_INFO_KHR = 1000002000,
  // Provided by VK_KHR_display
    VK_STRUCTURE_TYPE_DISPLAY_SURFACE_CREATE_INFO_KHR = 1000002001,
  // Provided by VK_KHR_display_swapchain
    VK_STRUCTURE_TYPE_DISPLAY_PRESENT_INFO_KHR = 1000003000,
  // Provided by VK_KHR_xlib_surface
    VK_STRUCTURE_TYPE_XLIB_SURFACE_CREATE_INFO_KHR = 1000004000,
  // Provided by VK_KHR_xcb_surface
    VK_STRUCTURE_TYPE_XCB_SURFACE_CREATE_INFO_KHR = 1000005000,
  // Provided by VK_KHR_wayland_surface
    VK_STRUCTURE_TYPE_WAYLAND_SURFACE_CREATE_INFO_KHR = 1000006000,
  // Provided by VK_KHR_android_surface
    VK_STRUCTURE_TYPE_ANDROID_SURFACE_CREATE_INFO_KHR = 1000008000,
```

```
  // Provided by VK_KHR_win32_surface
    VK_STRUCTURE_TYPE_WIN32_SURFACE_CREATE_INFO_KHR = 1000009000,
  // Provided by VK_EXT_debug_report
    VK_STRUCTURE_TYPE_DEBUG_REPORT_CALLBACK_CREATE_INFO_EXT = 1000011000,
  // Provided by VK_AMD_rasterization_order
    VK_STRUCTURE_TYPE_PIPELINE_RASTERIZATION_STATE_RASTERIZATION_ORDER_AMD =
1000018000,
  // Provided by VK_EXT_debug_marker
    VK_STRUCTURE_TYPE_DEBUG_MARKER_OBJECT_NAME_INFO_EXT = 1000022000,
  // Provided by VK_EXT_debug_marker
    VK_STRUCTURE_TYPE_DEBUG_MARKER_OBJECT_TAG_INFO_EXT = 1000022001,
  // Provided by VK_EXT_debug_marker
    VK_STRUCTURE_TYPE_DEBUG_MARKER_MARKER_INFO_EXT = 1000022002,
  // Provided by VK_KHR_video_queue
    VK_STRUCTURE_TYPE_VIDEO_PROFILE_INFO_KHR = 1000023000,
  // Provided by VK_KHR_video_queue
    VK_STRUCTURE_TYPE_VIDEO_CAPABILITIES_KHR = 1000023001,
  // Provided by VK_KHR_video_queue
    VK_STRUCTURE_TYPE_VIDEO_PICTURE_RESOURCE_INFO_KHR = 1000023002,
  // Provided by VK_KHR_video_queue
    VK_STRUCTURE_TYPE_VIDEO_SESSION_MEMORY_REQUIREMENTS_KHR = 1000023003,
  // Provided by VK_KHR_video_queue
    VK_STRUCTURE_TYPE_BIND_VIDEO_SESSION_MEMORY_INFO_KHR = 1000023004,
  // Provided by VK_KHR_video_queue
    VK_STRUCTURE_TYPE_VIDEO_SESSION_CREATE_INFO_KHR = 1000023005,
  // Provided by VK_KHR_video_queue
    VK_STRUCTURE_TYPE_VIDEO_SESSION_PARAMETERS_CREATE_INFO_KHR = 1000023006,
  // Provided by VK_KHR_video_queue
    VK_STRUCTURE_TYPE_VIDEO_SESSION_PARAMETERS_UPDATE_INFO_KHR = 1000023007,
  // Provided by VK_KHR_video_queue
    VK_STRUCTURE_TYPE_VIDEO_BEGIN_CODING_INFO_KHR = 1000023008,
  // Provided by VK_KHR_video_queue
    VK_STRUCTURE_TYPE_VIDEO_END_CODING_INFO_KHR = 1000023009,
  // Provided by VK_KHR_video_queue
    VK_STRUCTURE_TYPE_VIDEO_CODING_CONTROL_INFO_KHR = 1000023010,
  // Provided by VK_KHR_video_queue
    VK_STRUCTURE_TYPE_VIDEO_REFERENCE_SLOT_INFO_KHR = 1000023011,
  // Provided by VK_KHR_video_queue
    VK_STRUCTURE_TYPE_QUEUE_FAMILY_VIDEO_PROPERTIES_KHR = 1000023012,
  // Provided by VK_KHR_video_queue
    VK_STRUCTURE_TYPE_VIDEO_PROFILE_LIST_INFO_KHR = 1000023013,
  // Provided by VK_KHR_video_queue
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_VIDEO_FORMAT_INFO_KHR = 1000023014,
  // Provided by VK_KHR_video_queue
    VK_STRUCTURE_TYPE_VIDEO_FORMAT_PROPERTIES_KHR = 1000023015,
  // Provided by VK_KHR_video_queue
    VK_STRUCTURE_TYPE_QUEUE_FAMILY_QUERY_RESULT_STATUS_PROPERTIES_KHR = 1000023016,
  // Provided by VK_KHR_video_decode_queue
    VK_STRUCTURE_TYPE_VIDEO_DECODE_INFO_KHR = 1000024000,
  // Provided by VK_KHR_video_decode_queue
    VK_STRUCTURE_TYPE_VIDEO_DECODE_CAPABILITIES_KHR = 1000024001,
```

```
  // Provided by VK_KHR_video_decode_queue
    VK_STRUCTURE_TYPE_VIDEO_DECODE_USAGE_INFO_KHR = 1000024002,
  // Provided by VK_NV_dedicated_allocation
    VK_STRUCTURE_TYPE_DEDICATED_ALLOCATION_IMAGE_CREATE_INFO_NV = 1000026000,
  // Provided by VK_NV_dedicated_allocation
    VK_STRUCTURE_TYPE_DEDICATED_ALLOCATION_BUFFER_CREATE_INFO_NV = 1000026001,
  // Provided by VK_NV_dedicated_allocation
    VK_STRUCTURE_TYPE_DEDICATED_ALLOCATION_MEMORY_ALLOCATE_INFO_NV = 1000026002,
  // Provided by VK_EXT_transform_feedback
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_TRANSFORM_FEEDBACK_FEATURES_EXT = 1000028000,
  // Provided by VK_EXT_transform_feedback
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_TRANSFORM_FEEDBACK_PROPERTIES_EXT = 1000028001,
  // Provided by VK_EXT_transform_feedback
    VK_STRUCTURE_TYPE_PIPELINE_RASTERIZATION_STATE_STREAM_CREATE_INFO_EXT =
1000028002,
  // Provided by VK_NVX_binary_import
    VK_STRUCTURE_TYPE_CU_MODULE_CREATE_INFO_NVX = 1000029000,
  // Provided by VK_NVX_binary_import
    VK_STRUCTURE_TYPE_CU_FUNCTION_CREATE_INFO_NVX = 1000029001,
  // Provided by VK_NVX_binary_import
    VK_STRUCTURE_TYPE_CU_LAUNCH_INFO_NVX = 1000029002,
  // Provided by VK_NVX_image_view_handle
    VK_STRUCTURE_TYPE_IMAGE_VIEW_HANDLE_INFO_NVX = 1000030000,
  // Provided by VK_NVX_image_view_handle
    VK_STRUCTURE_TYPE_IMAGE_VIEW_ADDRESS_PROPERTIES_NVX = 1000030001,
  // Provided by VK_KHR_video_encode_h264
    VK_STRUCTURE_TYPE_VIDEO_ENCODE_H264_CAPABILITIES_KHR = 1000038000,
  // Provided by VK_KHR_video_encode_h264
    VK_STRUCTURE_TYPE_VIDEO_ENCODE_H264_SESSION_PARAMETERS_CREATE_INFO_KHR =
1000038001,
  // Provided by VK_KHR_video_encode_h264
    VK_STRUCTURE_TYPE_VIDEO_ENCODE_H264_SESSION_PARAMETERS_ADD_INFO_KHR = 1000038002,
  // Provided by VK_KHR_video_encode_h264
    VK_STRUCTURE_TYPE_VIDEO_ENCODE_H264_PICTURE_INFO_KHR = 1000038003,
  // Provided by VK_KHR_video_encode_h264
    VK_STRUCTURE_TYPE_VIDEO_ENCODE_H264_DPB_SLOT_INFO_KHR = 1000038004,
  // Provided by VK_KHR_video_encode_h264
    VK_STRUCTURE_TYPE_VIDEO_ENCODE_H264_NALU_SLICE_INFO_KHR = 1000038005,
  // Provided by VK_KHR_video_encode_h264
    VK_STRUCTURE_TYPE_VIDEO_ENCODE_H264_GOP_REMAINING_FRAME_INFO_KHR = 1000038006,
  // Provided by VK_KHR_video_encode_h264
    VK_STRUCTURE_TYPE_VIDEO_ENCODE_H264_PROFILE_INFO_KHR = 1000038007,
  // Provided by VK_KHR_video_encode_h264
    VK_STRUCTURE_TYPE_VIDEO_ENCODE_H264_RATE_CONTROL_INFO_KHR = 1000038008,
  // Provided by VK_KHR_video_encode_h264
    VK_STRUCTURE_TYPE_VIDEO_ENCODE_H264_RATE_CONTROL_LAYER_INFO_KHR = 1000038009,
  // Provided by VK_KHR_video_encode_h264
    VK_STRUCTURE_TYPE_VIDEO_ENCODE_H264_SESSION_CREATE_INFO_KHR = 1000038010,
  // Provided by VK_KHR_video_encode_h264
    VK_STRUCTURE_TYPE_VIDEO_ENCODE_H264_QUALITY_LEVEL_PROPERTIES_KHR = 1000038011,
  // Provided by VK_KHR_video_encode_h264
```

```
    VK_STRUCTURE_TYPE_VIDEO_ENCODE_H264_SESSION_PARAMETERS_GET_INFO_KHR = 1000038012,
  // Provided by VK_KHR_video_encode_h264
    VK_STRUCTURE_TYPE_VIDEO_ENCODE_H264_SESSION_PARAMETERS_FEEDBACK_INFO_KHR =
1000038013,
  // Provided by VK_KHR_video_encode_h265
    VK_STRUCTURE_TYPE_VIDEO_ENCODE_H265_CAPABILITIES_KHR = 1000039000,
  // Provided by VK_KHR_video_encode_h265
    VK_STRUCTURE_TYPE_VIDEO_ENCODE_H265_SESSION_PARAMETERS_CREATE_INFO_KHR =
1000039001,
  // Provided by VK_KHR_video_encode_h265
    VK_STRUCTURE_TYPE_VIDEO_ENCODE_H265_SESSION_PARAMETERS_ADD_INFO_KHR = 1000039002,
  // Provided by VK_KHR_video_encode_h265
    VK_STRUCTURE_TYPE_VIDEO_ENCODE_H265_PICTURE_INFO_KHR = 1000039003,
  // Provided by VK_KHR_video_encode_h265
    VK_STRUCTURE_TYPE_VIDEO_ENCODE_H265_DPB_SLOT_INFO_KHR = 1000039004,
  // Provided by VK_KHR_video_encode_h265
    VK_STRUCTURE_TYPE_VIDEO_ENCODE_H265_NALU_SLICE_SEGMENT_INFO_KHR = 1000039005,
  // Provided by VK_KHR_video_encode_h265
    VK_STRUCTURE_TYPE_VIDEO_ENCODE_H265_GOP_REMAINING_FRAME_INFO_KHR = 1000039006,
  // Provided by VK_KHR_video_encode_h265
    VK_STRUCTURE_TYPE_VIDEO_ENCODE_H265_PROFILE_INFO_KHR = 1000039007,
  // Provided by VK_KHR_video_encode_h265
    VK_STRUCTURE_TYPE_VIDEO_ENCODE_H265_RATE_CONTROL_INFO_KHR = 1000039009,
  // Provided by VK_KHR_video_encode_h265
    VK_STRUCTURE_TYPE_VIDEO_ENCODE_H265_RATE_CONTROL_LAYER_INFO_KHR = 1000039010,
  // Provided by VK_KHR_video_encode_h265
    VK_STRUCTURE_TYPE_VIDEO_ENCODE_H265_SESSION_CREATE_INFO_KHR = 1000039011,
  // Provided by VK_KHR_video_encode_h265
    VK_STRUCTURE_TYPE_VIDEO_ENCODE_H265_QUALITY_LEVEL_PROPERTIES_KHR = 1000039012,
  // Provided by VK_KHR_video_encode_h265
    VK_STRUCTURE_TYPE_VIDEO_ENCODE_H265_SESSION_PARAMETERS_GET_INFO_KHR = 1000039013,
  // Provided by VK_KHR_video_encode_h265
    VK_STRUCTURE_TYPE_VIDEO_ENCODE_H265_SESSION_PARAMETERS_FEEDBACK_INFO_KHR =
1000039014,
  // Provided by VK_KHR_video_decode_h264
    VK_STRUCTURE_TYPE_VIDEO_DECODE_H264_CAPABILITIES_KHR = 1000040000,
  // Provided by VK_KHR_video_decode_h264
    VK_STRUCTURE_TYPE_VIDEO_DECODE_H264_PICTURE_INFO_KHR = 1000040001,
  // Provided by VK_KHR_video_decode_h264
    VK_STRUCTURE_TYPE_VIDEO_DECODE_H264_PROFILE_INFO_KHR = 1000040003,
  // Provided by VK_KHR_video_decode_h264
    VK_STRUCTURE_TYPE_VIDEO_DECODE_H264_SESSION_PARAMETERS_CREATE_INFO_KHR =
1000040004,
  // Provided by VK_KHR_video_decode_h264
    VK_STRUCTURE_TYPE_VIDEO_DECODE_H264_SESSION_PARAMETERS_ADD_INFO_KHR = 1000040005,
  // Provided by VK_KHR_video_decode_h264
    VK_STRUCTURE_TYPE_VIDEO_DECODE_H264_DPB_SLOT_INFO_KHR = 1000040006,
  // Provided by VK_AMD_texture_gather_bias_lod
    VK_STRUCTURE_TYPE_TEXTURE_LOD_GATHER_FORMAT_PROPERTIES_AMD = 1000041000,
  // Provided by VK_KHR_dynamic_rendering with VK_KHR_fragment_shading_rate
    VK_STRUCTURE_TYPE_RENDERING_FRAGMENT_SHADING_RATE_ATTACHMENT_INFO_KHR =
```

```
1000044006,
  // Provided by VK_KHR_dynamic_rendering with VK_EXT_fragment_density_map
    VK_STRUCTURE_TYPE_RENDERING_FRAGMENT_DENSITY_MAP_ATTACHMENT_INFO_EXT = 1000044007,
  // Provided by VK_KHR_dynamic_rendering with VK_AMD_mixed_attachment_samples
    VK_STRUCTURE_TYPE_ATTACHMENT_SAMPLE_COUNT_INFO_AMD = 1000044008,
  // Provided by VK_KHR_dynamic_rendering with VK_NVX_multiview_per_view_attributes
    VK_STRUCTURE_TYPE_MULTIVIEW_PER_VIEW_ATTRIBUTES_INFO_NVX = 1000044009,
  // Provided by VK_GGP_stream_descriptor_surface
    VK_STRUCTURE_TYPE_STREAM_DESCRIPTOR_SURFACE_CREATE_INFO_GGP = 1000049000,
  // Provided by VK_NV_corner_sampled_image
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_CORNER_SAMPLED_IMAGE_FEATURES_NV = 1000050000,
  // Provided by VK_NV_external_memory
    VK_STRUCTURE_TYPE_EXTERNAL_MEMORY_IMAGE_CREATE_INFO_NV = 1000056000,
  // Provided by VK_NV_external_memory
    VK_STRUCTURE_TYPE_EXPORT_MEMORY_ALLOCATE_INFO_NV = 1000056001,
  // Provided by VK_NV_external_memory_win32
    VK_STRUCTURE_TYPE_IMPORT_MEMORY_WIN32_HANDLE_INFO_NV = 1000057000,
  // Provided by VK_NV_external_memory_win32
    VK_STRUCTURE_TYPE_EXPORT_MEMORY_WIN32_HANDLE_INFO_NV = 1000057001,
  // Provided by VK_NV_win32_keyed_mutex
    VK_STRUCTURE_TYPE_WIN32_KEYED_MUTEX_ACQUIRE_RELEASE_INFO_NV = 1000058000,
  // Provided by VK_EXT_validation_flags
    VK_STRUCTURE_TYPE_VALIDATION_FLAGS_EXT = 1000061000,
  // Provided by VK_NN_vi_surface
    VK_STRUCTURE_TYPE_VI_SURFACE_CREATE_INFO_NN = 1000062000,
  // Provided by VK_EXT_astc_decode_mode
    VK_STRUCTURE_TYPE_IMAGE_VIEW_ASTC_DECODE_MODE_EXT = 1000067000,
  // Provided by VK_EXT_astc_decode_mode
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_ASTC_DECODE_FEATURES_EXT = 1000067001,
  // Provided by VK_EXT_pipeline_robustness
    VK_STRUCTURE_TYPE_PIPELINE_ROBUSTNESS_CREATE_INFO_EXT = 1000068000,
  // Provided by VK_EXT_pipeline_robustness
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PIPELINE_ROBUSTNESS_FEATURES_EXT = 1000068001,
  // Provided by VK_EXT_pipeline_robustness
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PIPELINE_ROBUSTNESS_PROPERTIES_EXT = 1000068002,
  // Provided by VK_KHR_external_memory_win32
    VK_STRUCTURE_TYPE_IMPORT_MEMORY_WIN32_HANDLE_INFO_KHR = 1000073000,
  // Provided by VK_KHR_external_memory_win32
    VK_STRUCTURE_TYPE_EXPORT_MEMORY_WIN32_HANDLE_INFO_KHR = 1000073001,
  // Provided by VK_KHR_external_memory_win32
    VK_STRUCTURE_TYPE_MEMORY_WIN32_HANDLE_PROPERTIES_KHR = 1000073002,
  // Provided by VK_KHR_external_memory_win32
    VK_STRUCTURE_TYPE_MEMORY_GET_WIN32_HANDLE_INFO_KHR = 1000073003,
  // Provided by VK_KHR_external_memory_fd
    VK_STRUCTURE_TYPE_IMPORT_MEMORY_FD_INFO_KHR = 1000074000,
  // Provided by VK_KHR_external_memory_fd
    VK_STRUCTURE_TYPE_MEMORY_FD_PROPERTIES_KHR = 1000074001,
  // Provided by VK_KHR_external_memory_fd
    VK_STRUCTURE_TYPE_MEMORY_GET_FD_INFO_KHR = 1000074002,
  // Provided by VK_KHR_win32_keyed_mutex
    VK_STRUCTURE_TYPE_WIN32_KEYED_MUTEX_ACQUIRE_RELEASE_INFO_KHR = 1000075000,
```

```
  // Provided by VK_KHR_external_semaphore_win32
    VK_STRUCTURE_TYPE_IMPORT_SEMAPHORE_WIN32_HANDLE_INFO_KHR = 1000078000,
  // Provided by VK_KHR_external_semaphore_win32
    VK_STRUCTURE_TYPE_EXPORT_SEMAPHORE_WIN32_HANDLE_INFO_KHR = 1000078001,
  // Provided by VK_KHR_external_semaphore_win32
    VK_STRUCTURE_TYPE_D3D12_FENCE_SUBMIT_INFO_KHR = 1000078002,
  // Provided by VK_KHR_external_semaphore_win32
    VK_STRUCTURE_TYPE_SEMAPHORE_GET_WIN32_HANDLE_INFO_KHR = 1000078003,
  // Provided by VK_KHR_external_semaphore_fd
    VK_STRUCTURE_TYPE_IMPORT_SEMAPHORE_FD_INFO_KHR = 1000079000,
  // Provided by VK_KHR_external_semaphore_fd
    VK_STRUCTURE_TYPE_SEMAPHORE_GET_FD_INFO_KHR = 1000079001,
  // Provided by VK_KHR_push_descriptor
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PUSH_DESCRIPTOR_PROPERTIES_KHR = 1000080000,
  // Provided by VK_EXT_conditional_rendering
    VK_STRUCTURE_TYPE_COMMAND_BUFFER_INHERITANCE_CONDITIONAL_RENDERING_INFO_EXT =
1000081000,
  // Provided by VK_EXT_conditional_rendering
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_CONDITIONAL_RENDERING_FEATURES_EXT = 1000081001,
  // Provided by VK_EXT_conditional_rendering
    VK_STRUCTURE_TYPE_CONDITIONAL_RENDERING_BEGIN_INFO_EXT = 1000081002,
  // Provided by VK_KHR_incremental_present
    VK_STRUCTURE_TYPE_PRESENT_REGIONS_KHR = 1000084000,
  // Provided by VK_NV_clip_space_w_scaling
    VK_STRUCTURE_TYPE_PIPELINE_VIEWPORT_W_SCALING_STATE_CREATE_INFO_NV = 1000087000,
  // Provided by VK_EXT_display_surface_counter
    VK_STRUCTURE_TYPE_SURFACE_CAPABILITIES_2_EXT = 1000090000,
  // Provided by VK_EXT_display_control
    VK_STRUCTURE_TYPE_DISPLAY_POWER_INFO_EXT = 1000091000,
  // Provided by VK_EXT_display_control
    VK_STRUCTURE_TYPE_DEVICE_EVENT_INFO_EXT = 1000091001,
  // Provided by VK_EXT_display_control
    VK_STRUCTURE_TYPE_DISPLAY_EVENT_INFO_EXT = 1000091002,
  // Provided by VK_EXT_display_control
    VK_STRUCTURE_TYPE_SWAPCHAIN_COUNTER_CREATE_INFO_EXT = 1000091003,
  // Provided by VK_GOOGLE_display_timing
    VK_STRUCTURE_TYPE_PRESENT_TIMES_INFO_GOOGLE = 1000092000,
  // Provided by VK_NVX_multiview_per_view_attributes
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MULTIVIEW_PER_VIEW_ATTRIBUTES_PROPERTIES_NVX =
1000097000,
  // Provided by VK_NV_viewport_swizzle
    VK_STRUCTURE_TYPE_PIPELINE_VIEWPORT_SWIZZLE_STATE_CREATE_INFO_NV = 1000098000,
  // Provided by VK_EXT_discard_rectangles
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DISCARD_RECTANGLE_PROPERTIES_EXT = 1000099000,
  // Provided by VK_EXT_discard_rectangles
    VK_STRUCTURE_TYPE_PIPELINE_DISCARD_RECTANGLE_STATE_CREATE_INFO_EXT = 1000099001,
  // Provided by VK_EXT_conservative_rasterization
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_CONSERVATIVE_RASTERIZATION_PROPERTIES_EXT =
1000101000,
  // Provided by VK_EXT_conservative_rasterization
    VK_STRUCTURE_TYPE_PIPELINE_RASTERIZATION_CONSERVATIVE_STATE_CREATE_INFO_EXT =
```

```
1000101001,
  // Provided by VK_EXT_depth_clip_enable
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DEPTH_CLIP_ENABLE_FEATURES_EXT = 1000102000,
  // Provided by VK_EXT_depth_clip_enable
    VK_STRUCTURE_TYPE_PIPELINE_RASTERIZATION_DEPTH_CLIP_STATE_CREATE_INFO_EXT =
1000102001,
  // Provided by VK_EXT_hdr_metadata
    VK_STRUCTURE_TYPE_HDR_METADATA_EXT = 1000105000,
  // Provided by VK_IMG_relaxed_line_rasterization
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_RELAXED_LINE_RASTERIZATION_FEATURES_IMG =
1000110000,
  // Provided by VK_KHR_shared_presentable_image
    VK_STRUCTURE_TYPE_SHARED_PRESENT_SURFACE_CAPABILITIES_KHR = 1000111000,
  // Provided by VK_KHR_external_fence_win32
    VK_STRUCTURE_TYPE_IMPORT_FENCE_WIN32_HANDLE_INFO_KHR = 1000114000,
  // Provided by VK_KHR_external_fence_win32
    VK_STRUCTURE_TYPE_EXPORT_FENCE_WIN32_HANDLE_INFO_KHR = 1000114001,
  // Provided by VK_KHR_external_fence_win32
    VK_STRUCTURE_TYPE_FENCE_GET_WIN32_HANDLE_INFO_KHR = 1000114002,
  // Provided by VK_KHR_external_fence_fd
    VK_STRUCTURE_TYPE_IMPORT_FENCE_FD_INFO_KHR = 1000115000,
  // Provided by VK_KHR_external_fence_fd
    VK_STRUCTURE_TYPE_FENCE_GET_FD_INFO_KHR = 1000115001,
  // Provided by VK_KHR_performance_query
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PERFORMANCE_QUERY_FEATURES_KHR = 1000116000,
  // Provided by VK_KHR_performance_query
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PERFORMANCE_QUERY_PROPERTIES_KHR = 1000116001,
  // Provided by VK_KHR_performance_query
    VK_STRUCTURE_TYPE_QUERY_POOL_PERFORMANCE_CREATE_INFO_KHR = 1000116002,
  // Provided by VK_KHR_performance_query
    VK_STRUCTURE_TYPE_PERFORMANCE_QUERY_SUBMIT_INFO_KHR = 1000116003,
  // Provided by VK_KHR_performance_query
    VK_STRUCTURE_TYPE_ACQUIRE_PROFILING_LOCK_INFO_KHR = 1000116004,
  // Provided by VK_KHR_performance_query
    VK_STRUCTURE_TYPE_PERFORMANCE_COUNTER_KHR = 1000116005,
  // Provided by VK_KHR_performance_query
    VK_STRUCTURE_TYPE_PERFORMANCE_COUNTER_DESCRIPTION_KHR = 1000116006,
  // Provided by VK_KHR_get_surface_capabilities2
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SURFACE_INFO_2_KHR = 1000119000,
  // Provided by VK_KHR_get_surface_capabilities2
    VK_STRUCTURE_TYPE_SURFACE_CAPABILITIES_2_KHR = 1000119001,
  // Provided by VK_KHR_get_surface_capabilities2
    VK_STRUCTURE_TYPE_SURFACE_FORMAT_2_KHR = 1000119002,
  // Provided by VK_KHR_get_display_properties2
    VK_STRUCTURE_TYPE_DISPLAY_PROPERTIES_2_KHR = 1000121000,
  // Provided by VK_KHR_get_display_properties2
    VK_STRUCTURE_TYPE_DISPLAY_PLANE_PROPERTIES_2_KHR = 1000121001,
  // Provided by VK_KHR_get_display_properties2
    VK_STRUCTURE_TYPE_DISPLAY_MODE_PROPERTIES_2_KHR = 1000121002,
  // Provided by VK_KHR_get_display_properties2
    VK_STRUCTURE_TYPE_DISPLAY_PLANE_INFO_2_KHR = 1000121003,
```

```
  // Provided by VK_KHR_get_display_properties2
    VK_STRUCTURE_TYPE_DISPLAY_PLANE_CAPABILITIES_2_KHR = 1000121004,
  // Provided by VK_MVK_ios_surface
    VK_STRUCTURE_TYPE_IOS_SURFACE_CREATE_INFO_MVK = 1000122000,
  // Provided by VK_MVK_macos_surface
    VK_STRUCTURE_TYPE_MACOS_SURFACE_CREATE_INFO_MVK = 1000123000,
  // Provided by VK_EXT_debug_utils
    VK_STRUCTURE_TYPE_DEBUG_UTILS_OBJECT_NAME_INFO_EXT = 1000128000,
  // Provided by VK_EXT_debug_utils
    VK_STRUCTURE_TYPE_DEBUG_UTILS_OBJECT_TAG_INFO_EXT = 1000128001,
  // Provided by VK_EXT_debug_utils
    VK_STRUCTURE_TYPE_DEBUG_UTILS_LABEL_EXT = 1000128002,
  // Provided by VK_EXT_debug_utils
    VK_STRUCTURE_TYPE_DEBUG_UTILS_MESSENGER_CALLBACK_DATA_EXT = 1000128003,
  // Provided by VK_EXT_debug_utils
    VK_STRUCTURE_TYPE_DEBUG_UTILS_MESSENGER_CREATE_INFO_EXT = 1000128004,
  // Provided by VK_ANDROID_external_memory_android_hardware_buffer
    VK_STRUCTURE_TYPE_ANDROID_HARDWARE_BUFFER_USAGE_ANDROID = 1000129000,
  // Provided by VK_ANDROID_external_memory_android_hardware_buffer
    VK_STRUCTURE_TYPE_ANDROID_HARDWARE_BUFFER_PROPERTIES_ANDROID = 1000129001,
  // Provided by VK_ANDROID_external_memory_android_hardware_buffer
    VK_STRUCTURE_TYPE_ANDROID_HARDWARE_BUFFER_FORMAT_PROPERTIES_ANDROID = 1000129002,
  // Provided by VK_ANDROID_external_memory_android_hardware_buffer
    VK_STRUCTURE_TYPE_IMPORT_ANDROID_HARDWARE_BUFFER_INFO_ANDROID = 1000129003,
  // Provided by VK_ANDROID_external_memory_android_hardware_buffer
    VK_STRUCTURE_TYPE_MEMORY_GET_ANDROID_HARDWARE_BUFFER_INFO_ANDROID = 1000129004,
  // Provided by VK_ANDROID_external_memory_android_hardware_buffer
    VK_STRUCTURE_TYPE_EXTERNAL_FORMAT_ANDROID = 1000129005,
  // Provided by VK_ANDROID_external_memory_android_hardware_buffer with
VK_KHR_format_feature_flags2 or VK_VERSION_1_3
    VK_STRUCTURE_TYPE_ANDROID_HARDWARE_BUFFER_FORMAT_PROPERTIES_2_ANDROID =
1000129006,
#ifdef VK_ENABLE_BETA_EXTENSIONS
  // Provided by VK_AMDX_shader_enqueue
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_ENQUEUE_FEATURES_AMDX = 1000134000,
#endif
#ifdef VK_ENABLE_BETA_EXTENSIONS
  // Provided by VK_AMDX_shader_enqueue
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_ENQUEUE_PROPERTIES_AMDX = 1000134001,
#endif
#ifdef VK_ENABLE_BETA_EXTENSIONS
  // Provided by VK_AMDX_shader_enqueue
    VK_STRUCTURE_TYPE_EXECUTION_GRAPH_PIPELINE_SCRATCH_SIZE_AMDX = 1000134002,
#endif
#ifdef VK_ENABLE_BETA_EXTENSIONS
  // Provided by VK_AMDX_shader_enqueue
    VK_STRUCTURE_TYPE_EXECUTION_GRAPH_PIPELINE_CREATE_INFO_AMDX = 1000134003,
#endif
#ifdef VK_ENABLE_BETA_EXTENSIONS
  // Provided by VK_AMDX_shader_enqueue
    VK_STRUCTURE_TYPE_PIPELINE_SHADER_STAGE_NODE_CREATE_INFO_AMDX = 1000134004,
```

```
#endif
  // Provided by VK_EXT_sample_locations
    VK_STRUCTURE_TYPE_SAMPLE_LOCATIONS_INFO_EXT = 1000143000,
  // Provided by VK_EXT_sample_locations
    VK_STRUCTURE_TYPE_RENDER_PASS_SAMPLE_LOCATIONS_BEGIN_INFO_EXT = 1000143001,
  // Provided by VK_EXT_sample_locations
    VK_STRUCTURE_TYPE_PIPELINE_SAMPLE_LOCATIONS_STATE_CREATE_INFO_EXT = 1000143002,
  // Provided by VK_EXT_sample_locations
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SAMPLE_LOCATIONS_PROPERTIES_EXT = 1000143003,
  // Provided by VK_EXT_sample_locations
    VK_STRUCTURE_TYPE_MULTISAMPLE_PROPERTIES_EXT = 1000143004,
  // Provided by VK_EXT_blend_operation_advanced
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_BLEND_OPERATION_ADVANCED_FEATURES_EXT =
1000148000,
  // Provided by VK_EXT_blend_operation_advanced
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_BLEND_OPERATION_ADVANCED_PROPERTIES_EXT =
1000148001,
  // Provided by VK_EXT_blend_operation_advanced
    VK_STRUCTURE_TYPE_PIPELINE_COLOR_BLEND_ADVANCED_STATE_CREATE_INFO_EXT =
1000148002,
  // Provided by VK_NV_fragment_coverage_to_color
    VK_STRUCTURE_TYPE_PIPELINE_COVERAGE_TO_COLOR_STATE_CREATE_INFO_NV = 1000149000,
  // Provided by VK_KHR_acceleration_structure
    VK_STRUCTURE_TYPE_WRITE_DESCRIPTOR_SET_ACCELERATION_STRUCTURE_KHR = 1000150007,
  // Provided by VK_KHR_acceleration_structure
    VK_STRUCTURE_TYPE_ACCELERATION_STRUCTURE_BUILD_GEOMETRY_INFO_KHR = 1000150000,
  // Provided by VK_KHR_acceleration_structure
    VK_STRUCTURE_TYPE_ACCELERATION_STRUCTURE_DEVICE_ADDRESS_INFO_KHR = 1000150002,
  // Provided by VK_KHR_acceleration_structure
    VK_STRUCTURE_TYPE_ACCELERATION_STRUCTURE_GEOMETRY_AABBS_DATA_KHR = 1000150003,
  // Provided by VK_KHR_acceleration_structure
    VK_STRUCTURE_TYPE_ACCELERATION_STRUCTURE_GEOMETRY_INSTANCES_DATA_KHR = 1000150004,
  // Provided by VK_KHR_acceleration_structure
    VK_STRUCTURE_TYPE_ACCELERATION_STRUCTURE_GEOMETRY_TRIANGLES_DATA_KHR = 1000150005,
  // Provided by VK_KHR_acceleration_structure
    VK_STRUCTURE_TYPE_ACCELERATION_STRUCTURE_GEOMETRY_KHR = 1000150006,
  // Provided by VK_KHR_acceleration_structure
    VK_STRUCTURE_TYPE_ACCELERATION_STRUCTURE_VERSION_INFO_KHR = 1000150009,
  // Provided by VK_KHR_acceleration_structure
    VK_STRUCTURE_TYPE_COPY_ACCELERATION_STRUCTURE_INFO_KHR = 1000150010,
  // Provided by VK_KHR_acceleration_structure
    VK_STRUCTURE_TYPE_COPY_ACCELERATION_STRUCTURE_TO_MEMORY_INFO_KHR = 1000150011,
  // Provided by VK_KHR_acceleration_structure
    VK_STRUCTURE_TYPE_COPY_MEMORY_TO_ACCELERATION_STRUCTURE_INFO_KHR = 1000150012,
  // Provided by VK_KHR_acceleration_structure
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_ACCELERATION_STRUCTURE_FEATURES_KHR =
1000150013,
  // Provided by VK_KHR_acceleration_structure
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_ACCELERATION_STRUCTURE_PROPERTIES_KHR =
1000150014,
  // Provided by VK_KHR_acceleration_structure
```

```
    VK_STRUCTURE_TYPE_ACCELERATION_STRUCTURE_CREATE_INFO_KHR = 1000150017,
  // Provided by VK_KHR_acceleration_structure
    VK_STRUCTURE_TYPE_ACCELERATION_STRUCTURE_BUILD_SIZES_INFO_KHR = 1000150020,
  // Provided by VK_KHR_ray_tracing_pipeline
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_RAY_TRACING_PIPELINE_FEATURES_KHR = 1000347000,
  // Provided by VK_KHR_ray_tracing_pipeline
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_RAY_TRACING_PIPELINE_PROPERTIES_KHR =
1000347001,
  // Provided by VK_KHR_ray_tracing_pipeline
    VK_STRUCTURE_TYPE_RAY_TRACING_PIPELINE_CREATE_INFO_KHR = 1000150015,
  // Provided by VK_KHR_ray_tracing_pipeline
    VK_STRUCTURE_TYPE_RAY_TRACING_SHADER_GROUP_CREATE_INFO_KHR = 1000150016,
  // Provided by VK_KHR_ray_tracing_pipeline
    VK_STRUCTURE_TYPE_RAY_TRACING_PIPELINE_INTERFACE_CREATE_INFO_KHR = 1000150018,
  // Provided by VK_KHR_ray_query
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_RAY_QUERY_FEATURES_KHR = 1000348013,
  // Provided by VK_NV_framebuffer_mixed_samples
    VK_STRUCTURE_TYPE_PIPELINE_COVERAGE_MODULATION_STATE_CREATE_INFO_NV = 1000152000,
  // Provided by VK_NV_shader_sm_builtins
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_SM_BUILTINS_FEATURES_NV = 1000154000,
  // Provided by VK_NV_shader_sm_builtins
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_SM_BUILTINS_PROPERTIES_NV = 1000154001,
  // Provided by VK_EXT_image_drm_format_modifier
    VK_STRUCTURE_TYPE_DRM_FORMAT_MODIFIER_PROPERTIES_LIST_EXT = 1000158000,
  // Provided by VK_EXT_image_drm_format_modifier
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_IMAGE_DRM_FORMAT_MODIFIER_INFO_EXT = 1000158002,
  // Provided by VK_EXT_image_drm_format_modifier
    VK_STRUCTURE_TYPE_IMAGE_DRM_FORMAT_MODIFIER_LIST_CREATE_INFO_EXT = 1000158003,
  // Provided by VK_EXT_image_drm_format_modifier
    VK_STRUCTURE_TYPE_IMAGE_DRM_FORMAT_MODIFIER_EXPLICIT_CREATE_INFO_EXT = 1000158004,
  // Provided by VK_EXT_image_drm_format_modifier
    VK_STRUCTURE_TYPE_IMAGE_DRM_FORMAT_MODIFIER_PROPERTIES_EXT = 1000158005,
  // Provided by VK_EXT_image_drm_format_modifier with VK_KHR_format_feature_flags2 or
VK_VERSION_1_3
    VK_STRUCTURE_TYPE_DRM_FORMAT_MODIFIER_PROPERTIES_LIST_2_EXT = 1000158006,
  // Provided by VK_EXT_validation_cache
    VK_STRUCTURE_TYPE_VALIDATION_CACHE_CREATE_INFO_EXT = 1000160000,
  // Provided by VK_EXT_validation_cache
    VK_STRUCTURE_TYPE_SHADER_MODULE_VALIDATION_CACHE_CREATE_INFO_EXT = 1000160001,
#ifdef VK_ENABLE_BETA_EXTENSIONS
  // Provided by VK_KHR_portability_subset
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PORTABILITY_SUBSET_FEATURES_KHR = 1000163000,
#endif
#ifdef VK_ENABLE_BETA_EXTENSIONS
  // Provided by VK_KHR_portability_subset
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PORTABILITY_SUBSET_PROPERTIES_KHR = 1000163001,
#endif
  // Provided by VK_NV_shading_rate_image
    VK_STRUCTURE_TYPE_PIPELINE_VIEWPORT_SHADING_RATE_IMAGE_STATE_CREATE_INFO_NV =
1000164000,
  // Provided by VK_NV_shading_rate_image
```

```
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADING_RATE_IMAGE_FEATURES_NV = 1000164001,
  // Provided by VK_NV_shading_rate_image
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADING_RATE_IMAGE_PROPERTIES_NV = 1000164002,
  // Provided by VK_NV_shading_rate_image
    VK_STRUCTURE_TYPE_PIPELINE_VIEWPORT_COARSE_SAMPLE_ORDER_STATE_CREATE_INFO_NV =
1000164005,
  // Provided by VK_NV_ray_tracing
    VK_STRUCTURE_TYPE_RAY_TRACING_PIPELINE_CREATE_INFO_NV = 1000165000,
  // Provided by VK_NV_ray_tracing
    VK_STRUCTURE_TYPE_ACCELERATION_STRUCTURE_CREATE_INFO_NV = 1000165001,
  // Provided by VK_NV_ray_tracing
    VK_STRUCTURE_TYPE_GEOMETRY_NV = 1000165003,
  // Provided by VK_NV_ray_tracing
    VK_STRUCTURE_TYPE_GEOMETRY_TRIANGLES_NV = 1000165004,
  // Provided by VK_NV_ray_tracing
    VK_STRUCTURE_TYPE_GEOMETRY_AABB_NV = 1000165005,
  // Provided by VK_NV_ray_tracing
    VK_STRUCTURE_TYPE_BIND_ACCELERATION_STRUCTURE_MEMORY_INFO_NV = 1000165006,
  // Provided by VK_NV_ray_tracing
    VK_STRUCTURE_TYPE_WRITE_DESCRIPTOR_SET_ACCELERATION_STRUCTURE_NV = 1000165007,
  // Provided by VK_NV_ray_tracing
    VK_STRUCTURE_TYPE_ACCELERATION_STRUCTURE_MEMORY_REQUIREMENTS_INFO_NV = 1000165008,
  // Provided by VK_NV_ray_tracing
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_RAY_TRACING_PROPERTIES_NV = 1000165009,
  // Provided by VK_NV_ray_tracing
    VK_STRUCTURE_TYPE_RAY_TRACING_SHADER_GROUP_CREATE_INFO_NV = 1000165011,
  // Provided by VK_NV_ray_tracing
    VK_STRUCTURE_TYPE_ACCELERATION_STRUCTURE_INFO_NV = 1000165012,
  // Provided by VK_NV_representative_fragment_test
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_REPRESENTATIVE_FRAGMENT_TEST_FEATURES_NV =
1000166000,
  // Provided by VK_NV_representative_fragment_test
    VK_STRUCTURE_TYPE_PIPELINE_REPRESENTATIVE_FRAGMENT_TEST_STATE_CREATE_INFO_NV =
1000166001,
  // Provided by VK_EXT_filter_cubic
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_IMAGE_VIEW_IMAGE_FORMAT_INFO_EXT = 1000170000,
  // Provided by VK_EXT_filter_cubic
    VK_STRUCTURE_TYPE_FILTER_CUBIC_IMAGE_VIEW_IMAGE_FORMAT_PROPERTIES_EXT =
1000170001,
  // Provided by VK_EXT_external_memory_host
    VK_STRUCTURE_TYPE_IMPORT_MEMORY_HOST_POINTER_INFO_EXT = 1000178000,
  // Provided by VK_EXT_external_memory_host
    VK_STRUCTURE_TYPE_MEMORY_HOST_POINTER_PROPERTIES_EXT = 1000178001,
  // Provided by VK_EXT_external_memory_host
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTERNAL_MEMORY_HOST_PROPERTIES_EXT =
1000178002,
  // Provided by VK_KHR_shader_clock
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_CLOCK_FEATURES_KHR = 1000181000,
  // Provided by VK_AMD_pipeline_compiler_control
    VK_STRUCTURE_TYPE_PIPELINE_COMPILER_CONTROL_CREATE_INFO_AMD = 1000183000,
  // Provided by VK_AMD_shader_core_properties
```

```
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_CORE_PROPERTIES_AMD = 1000185000,
  // Provided by VK_KHR_video_decode_h265
    VK_STRUCTURE_TYPE_VIDEO_DECODE_H265_CAPABILITIES_KHR = 1000187000,
  // Provided by VK_KHR_video_decode_h265
    VK_STRUCTURE_TYPE_VIDEO_DECODE_H265_SESSION_PARAMETERS_CREATE_INFO_KHR =
1000187001,
  // Provided by VK_KHR_video_decode_h265
    VK_STRUCTURE_TYPE_VIDEO_DECODE_H265_SESSION_PARAMETERS_ADD_INFO_KHR = 1000187002,
  // Provided by VK_KHR_video_decode_h265
    VK_STRUCTURE_TYPE_VIDEO_DECODE_H265_PROFILE_INFO_KHR = 1000187003,
  // Provided by VK_KHR_video_decode_h265
    VK_STRUCTURE_TYPE_VIDEO_DECODE_H265_PICTURE_INFO_KHR = 1000187004,
  // Provided by VK_KHR_video_decode_h265
    VK_STRUCTURE_TYPE_VIDEO_DECODE_H265_DPB_SLOT_INFO_KHR = 1000187005,
  // Provided by VK_KHR_global_priority
    VK_STRUCTURE_TYPE_DEVICE_QUEUE_GLOBAL_PRIORITY_CREATE_INFO_KHR = 1000174000,
  // Provided by VK_KHR_global_priority
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_GLOBAL_PRIORITY_QUERY_FEATURES_KHR = 1000388000,
  // Provided by VK_KHR_global_priority
    VK_STRUCTURE_TYPE_QUEUE_FAMILY_GLOBAL_PRIORITY_PROPERTIES_KHR = 1000388001,
  // Provided by VK_AMD_memory_overallocation_behavior
    VK_STRUCTURE_TYPE_DEVICE_MEMORY_OVERALLOCATION_CREATE_INFO_AMD = 1000189000,
  // Provided by VK_EXT_vertex_attribute_divisor
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_VERTEX_ATTRIBUTE_DIVISOR_PROPERTIES_EXT =
1000190000,
  // Provided by VK_GGP_frame_token
    VK_STRUCTURE_TYPE_PRESENT_FRAME_TOKEN_GGP = 1000191000,
  // Provided by VK_NV_mesh_shader
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MESH_SHADER_FEATURES_NV = 1000202000,
  // Provided by VK_NV_mesh_shader
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MESH_SHADER_PROPERTIES_NV = 1000202001,
  // Provided by VK_NV_shader_image_footprint
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_IMAGE_FOOTPRINT_FEATURES_NV = 1000204000,
  // Provided by VK_NV_scissor_exclusive
    VK_STRUCTURE_TYPE_PIPELINE_VIEWPORT_EXCLUSIVE_SCISSOR_STATE_CREATE_INFO_NV =
1000205000,
  // Provided by VK_NV_scissor_exclusive
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXCLUSIVE_SCISSOR_FEATURES_NV = 1000205002,
  // Provided by VK_NV_device_diagnostic_checkpoints
    VK_STRUCTURE_TYPE_CHECKPOINT_DATA_NV = 1000206000,
  // Provided by VK_NV_device_diagnostic_checkpoints
    VK_STRUCTURE_TYPE_QUEUE_FAMILY_CHECKPOINT_PROPERTIES_NV = 1000206001,
  // Provided by VK_INTEL_shader_integer_functions2
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_INTEGER_FUNCTIONS_2_FEATURES_INTEL =
1000209000,
  // Provided by VK_INTEL_performance_query
    VK_STRUCTURE_TYPE_QUERY_POOL_PERFORMANCE_QUERY_CREATE_INFO_INTEL = 1000210000,
  // Provided by VK_INTEL_performance_query
    VK_STRUCTURE_TYPE_INITIALIZE_PERFORMANCE_API_INFO_INTEL = 1000210001,
  // Provided by VK_INTEL_performance_query
    VK_STRUCTURE_TYPE_PERFORMANCE_MARKER_INFO_INTEL = 1000210002,
```

```
  // Provided by VK_INTEL_performance_query
    VK_STRUCTURE_TYPE_PERFORMANCE_STREAM_MARKER_INFO_INTEL = 1000210003,
  // Provided by VK_INTEL_performance_query
    VK_STRUCTURE_TYPE_PERFORMANCE_OVERRIDE_INFO_INTEL = 1000210004,
  // Provided by VK_INTEL_performance_query
    VK_STRUCTURE_TYPE_PERFORMANCE_CONFIGURATION_ACQUIRE_INFO_INTEL = 1000210005,
  // Provided by VK_EXT_pci_bus_info
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PCI_BUS_INFO_PROPERTIES_EXT = 1000212000,
  // Provided by VK_AMD_display_native_hdr
    VK_STRUCTURE_TYPE_DISPLAY_NATIVE_HDR_SURFACE_CAPABILITIES_AMD = 1000213000,
  // Provided by VK_AMD_display_native_hdr
    VK_STRUCTURE_TYPE_SWAPCHAIN_DISPLAY_NATIVE_HDR_CREATE_INFO_AMD = 1000213001,
  // Provided by VK_FUCHSIA_imagepipe_surface
    VK_STRUCTURE_TYPE_IMAGEPIPE_SURFACE_CREATE_INFO_FUCHSIA = 1000214000,
  // Provided by VK_EXT_metal_surface
    VK_STRUCTURE_TYPE_METAL_SURFACE_CREATE_INFO_EXT = 1000217000,
  // Provided by VK_EXT_fragment_density_map
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_FRAGMENT_DENSITY_MAP_FEATURES_EXT = 1000218000,
  // Provided by VK_EXT_fragment_density_map
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_FRAGMENT_DENSITY_MAP_PROPERTIES_EXT =
1000218001,
  // Provided by VK_EXT_fragment_density_map
    VK_STRUCTURE_TYPE_RENDER_PASS_FRAGMENT_DENSITY_MAP_CREATE_INFO_EXT = 1000218002,
  // Provided by VK_KHR_fragment_shading_rate
    VK_STRUCTURE_TYPE_FRAGMENT_SHADING_RATE_ATTACHMENT_INFO_KHR = 1000226000,
  // Provided by VK_KHR_fragment_shading_rate
    VK_STRUCTURE_TYPE_PIPELINE_FRAGMENT_SHADING_RATE_STATE_CREATE_INFO_KHR =
1000226001,
  // Provided by VK_KHR_fragment_shading_rate
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_FRAGMENT_SHADING_RATE_PROPERTIES_KHR =
1000226002,
  // Provided by VK_KHR_fragment_shading_rate
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_FRAGMENT_SHADING_RATE_FEATURES_KHR = 1000226003,
  // Provided by VK_KHR_fragment_shading_rate
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_FRAGMENT_SHADING_RATE_KHR = 1000226004,
  // Provided by VK_AMD_shader_core_properties2
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_CORE_PROPERTIES_2_AMD = 1000227000,
  // Provided by VK_AMD_device_coherent_memory
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_COHERENT_MEMORY_FEATURES_AMD = 1000229000,
  // Provided by VK_KHR_dynamic_rendering_local_read
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DYNAMIC_RENDERING_LOCAL_READ_FEATURES_KHR =
1000232000,
  // Provided by VK_KHR_dynamic_rendering_local_read
    VK_STRUCTURE_TYPE_RENDERING_ATTACHMENT_LOCATION_INFO_KHR = 1000232001,
  // Provided by VK_KHR_dynamic_rendering_local_read
    VK_STRUCTURE_TYPE_RENDERING_INPUT_ATTACHMENT_INDEX_INFO_KHR = 1000232002,
  // Provided by VK_EXT_shader_image_atomic_int64
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_IMAGE_ATOMIC_INT64_FEATURES_EXT =
1000234000,
  // Provided by VK_KHR_shader_quad_control
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_QUAD_CONTROL_FEATURES_KHR = 1000235000,
```

```
  // Provided by VK_EXT_memory_budget
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MEMORY_BUDGET_PROPERTIES_EXT = 1000237000,
  // Provided by VK_EXT_memory_priority
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MEMORY_PRIORITY_FEATURES_EXT = 1000238000,
  // Provided by VK_EXT_memory_priority
    VK_STRUCTURE_TYPE_MEMORY_PRIORITY_ALLOCATE_INFO_EXT = 1000238001,
  // Provided by VK_KHR_surface_protected_capabilities
    VK_STRUCTURE_TYPE_SURFACE_PROTECTED_CAPABILITIES_KHR = 1000239000,
  // Provided by VK_NV_dedicated_allocation_image_aliasing
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DEDICATED_ALLOCATION_IMAGE_ALIASING_FEATURES_NV
= 1000240000,
  // Provided by VK_EXT_buffer_device_address
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_BUFFER_DEVICE_ADDRESS_FEATURES_EXT = 1000244000,
  // Provided by VK_EXT_buffer_device_address
    VK_STRUCTURE_TYPE_BUFFER_DEVICE_ADDRESS_CREATE_INFO_EXT = 1000244002,
  // Provided by VK_EXT_validation_features
    VK_STRUCTURE_TYPE_VALIDATION_FEATURES_EXT = 1000247000,
  // Provided by VK_KHR_present_wait
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PRESENT_WAIT_FEATURES_KHR = 1000248000,
  // Provided by VK_NV_cooperative_matrix
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_COOPERATIVE_MATRIX_FEATURES_NV = 1000249000,
  // Provided by VK_NV_cooperative_matrix
    VK_STRUCTURE_TYPE_COOPERATIVE_MATRIX_PROPERTIES_NV = 1000249001,
  // Provided by VK_NV_cooperative_matrix
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_COOPERATIVE_MATRIX_PROPERTIES_NV = 1000249002,
  // Provided by VK_NV_coverage_reduction_mode
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_COVERAGE_REDUCTION_MODE_FEATURES_NV =
1000250000,
  // Provided by VK_NV_coverage_reduction_mode
    VK_STRUCTURE_TYPE_PIPELINE_COVERAGE_REDUCTION_STATE_CREATE_INFO_NV = 1000250001,
  // Provided by VK_NV_coverage_reduction_mode
    VK_STRUCTURE_TYPE_FRAMEBUFFER_MIXED_SAMPLES_COMBINATION_NV = 1000250002,
  // Provided by VK_EXT_fragment_shader_interlock
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_FRAGMENT_SHADER_INTERLOCK_FEATURES_EXT =
1000251000,
  // Provided by VK_EXT_ycbcr_image_arrays
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_YCBCR_IMAGE_ARRAYS_FEATURES_EXT = 1000252000,
  // Provided by VK_EXT_provoking_vertex
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PROVOKING_VERTEX_FEATURES_EXT = 1000254000,
  // Provided by VK_EXT_provoking_vertex
    VK_STRUCTURE_TYPE_PIPELINE_RASTERIZATION_PROVOKING_VERTEX_STATE_CREATE_INFO_EXT =
1000254001,
  // Provided by VK_EXT_provoking_vertex
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PROVOKING_VERTEX_PROPERTIES_EXT = 1000254002,
  // Provided by VK_EXT_full_screen_exclusive
    VK_STRUCTURE_TYPE_SURFACE_FULL_SCREEN_EXCLUSIVE_INFO_EXT = 1000255000,
  // Provided by VK_EXT_full_screen_exclusive
    VK_STRUCTURE_TYPE_SURFACE_CAPABILITIES_FULL_SCREEN_EXCLUSIVE_EXT = 1000255002,
  // Provided by VK_KHR_win32_surface with VK_EXT_full_screen_exclusive
    VK_STRUCTURE_TYPE_SURFACE_FULL_SCREEN_EXCLUSIVE_WIN32_INFO_EXT = 1000255001,
  // Provided by VK_EXT_headless_surface
```

```
    VK_STRUCTURE_TYPE_HEADLESS_SURFACE_CREATE_INFO_EXT = 1000256000,
  // Provided by VK_EXT_shader_atomic_float
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_ATOMIC_FLOAT_FEATURES_EXT = 1000260000,
  // Provided by VK_EXT_extended_dynamic_state
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTENDED_DYNAMIC_STATE_FEATURES_EXT =
1000267000,
  // Provided by VK_KHR_pipeline_executable_properties
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PIPELINE_EXECUTABLE_PROPERTIES_FEATURES_KHR =
1000269000,
  // Provided by VK_KHR_pipeline_executable_properties
    VK_STRUCTURE_TYPE_PIPELINE_INFO_KHR = 1000269001,
  // Provided by VK_KHR_pipeline_executable_properties
    VK_STRUCTURE_TYPE_PIPELINE_EXECUTABLE_PROPERTIES_KHR = 1000269002,
  // Provided by VK_KHR_pipeline_executable_properties
    VK_STRUCTURE_TYPE_PIPELINE_EXECUTABLE_INFO_KHR = 1000269003,
  // Provided by VK_KHR_pipeline_executable_properties
    VK_STRUCTURE_TYPE_PIPELINE_EXECUTABLE_STATISTIC_KHR = 1000269004,
  // Provided by VK_KHR_pipeline_executable_properties
    VK_STRUCTURE_TYPE_PIPELINE_EXECUTABLE_INTERNAL_REPRESENTATION_KHR = 1000269005,
  // Provided by VK_EXT_host_image_copy
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_HOST_IMAGE_COPY_FEATURES_EXT = 1000270000,
  // Provided by VK_EXT_host_image_copy
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_HOST_IMAGE_COPY_PROPERTIES_EXT = 1000270001,
  // Provided by VK_EXT_host_image_copy
    VK_STRUCTURE_TYPE_MEMORY_TO_IMAGE_COPY_EXT = 1000270002,
  // Provided by VK_EXT_host_image_copy
    VK_STRUCTURE_TYPE_IMAGE_TO_MEMORY_COPY_EXT = 1000270003,
  // Provided by VK_EXT_host_image_copy
    VK_STRUCTURE_TYPE_COPY_IMAGE_TO_MEMORY_INFO_EXT = 1000270004,
  // Provided by VK_EXT_host_image_copy
    VK_STRUCTURE_TYPE_COPY_MEMORY_TO_IMAGE_INFO_EXT = 1000270005,
  // Provided by VK_EXT_host_image_copy
    VK_STRUCTURE_TYPE_HOST_IMAGE_LAYOUT_TRANSITION_INFO_EXT = 1000270006,
  // Provided by VK_EXT_host_image_copy
    VK_STRUCTURE_TYPE_COPY_IMAGE_TO_IMAGE_INFO_EXT = 1000270007,
  // Provided by VK_EXT_host_image_copy
    VK_STRUCTURE_TYPE_SUBRESOURCE_HOST_MEMCPY_SIZE_EXT = 1000270008,
  // Provided by VK_EXT_host_image_copy
    VK_STRUCTURE_TYPE_HOST_IMAGE_COPY_DEVICE_PERFORMANCE_QUERY_EXT = 1000270009,
  // Provided by VK_KHR_map_memory2
    VK_STRUCTURE_TYPE_MEMORY_MAP_INFO_KHR = 1000271000,
  // Provided by VK_KHR_map_memory2
    VK_STRUCTURE_TYPE_MEMORY_UNMAP_INFO_KHR = 1000271001,
  // Provided by VK_EXT_map_memory_placed
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MAP_MEMORY_PLACED_FEATURES_EXT = 1000272000,
  // Provided by VK_EXT_map_memory_placed
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MAP_MEMORY_PLACED_PROPERTIES_EXT = 1000272001,
  // Provided by VK_EXT_map_memory_placed
    VK_STRUCTURE_TYPE_MEMORY_MAP_PLACED_INFO_EXT = 1000272002,
  // Provided by VK_EXT_shader_atomic_float2
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_ATOMIC_FLOAT_2_FEATURES_EXT = 1000273000,
```

```
  // Provided by VK_EXT_surface_maintenance1
    VK_STRUCTURE_TYPE_SURFACE_PRESENT_MODE_EXT = 1000274000,
  // Provided by VK_EXT_surface_maintenance1
    VK_STRUCTURE_TYPE_SURFACE_PRESENT_SCALING_CAPABILITIES_EXT = 1000274001,
  // Provided by VK_EXT_surface_maintenance1
    VK_STRUCTURE_TYPE_SURFACE_PRESENT_MODE_COMPATIBILITY_EXT = 1000274002,
  // Provided by VK_EXT_swapchain_maintenance1
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SWAPCHAIN_MAINTENANCE_1_FEATURES_EXT =
1000275000,
  // Provided by VK_EXT_swapchain_maintenance1
    VK_STRUCTURE_TYPE_SWAPCHAIN_PRESENT_FENCE_INFO_EXT = 1000275001,
  // Provided by VK_EXT_swapchain_maintenance1
    VK_STRUCTURE_TYPE_SWAPCHAIN_PRESENT_MODES_CREATE_INFO_EXT = 1000275002,
  // Provided by VK_EXT_swapchain_maintenance1
    VK_STRUCTURE_TYPE_SWAPCHAIN_PRESENT_MODE_INFO_EXT = 1000275003,
  // Provided by VK_EXT_swapchain_maintenance1
    VK_STRUCTURE_TYPE_SWAPCHAIN_PRESENT_SCALING_CREATE_INFO_EXT = 1000275004,
  // Provided by VK_EXT_swapchain_maintenance1
    VK_STRUCTURE_TYPE_RELEASE_SWAPCHAIN_IMAGES_INFO_EXT = 1000275005,
  // Provided by VK_NV_device_generated_commands
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DEVICE_GENERATED_COMMANDS_PROPERTIES_NV =
1000277000,
  // Provided by VK_NV_device_generated_commands
    VK_STRUCTURE_TYPE_GRAPHICS_SHADER_GROUP_CREATE_INFO_NV = 1000277001,
  // Provided by VK_NV_device_generated_commands
    VK_STRUCTURE_TYPE_GRAPHICS_PIPELINE_SHADER_GROUPS_CREATE_INFO_NV = 1000277002,
  // Provided by VK_NV_device_generated_commands
    VK_STRUCTURE_TYPE_INDIRECT_COMMANDS_LAYOUT_TOKEN_NV = 1000277003,
  // Provided by VK_NV_device_generated_commands
    VK_STRUCTURE_TYPE_INDIRECT_COMMANDS_LAYOUT_CREATE_INFO_NV = 1000277004,
  // Provided by VK_NV_device_generated_commands
    VK_STRUCTURE_TYPE_GENERATED_COMMANDS_INFO_NV = 1000277005,
  // Provided by VK_NV_device_generated_commands
    VK_STRUCTURE_TYPE_GENERATED_COMMANDS_MEMORY_REQUIREMENTS_INFO_NV = 1000277006,
  // Provided by VK_NV_device_generated_commands
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DEVICE_GENERATED_COMMANDS_FEATURES_NV =
1000277007,
  // Provided by VK_NV_inherited_viewport_scissor
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_INHERITED_VIEWPORT_SCISSOR_FEATURES_NV =
1000278000,
  // Provided by VK_NV_inherited_viewport_scissor
    VK_STRUCTURE_TYPE_COMMAND_BUFFER_INHERITANCE_VIEWPORT_SCISSOR_INFO_NV =
1000278001,
  // Provided by VK_EXT_texel_buffer_alignment
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_TEXEL_BUFFER_ALIGNMENT_FEATURES_EXT =
1000281000,
  // Provided by VK_QCOM_render_pass_transform
    VK_STRUCTURE_TYPE_COMMAND_BUFFER_INHERITANCE_RENDER_PASS_TRANSFORM_INFO_QCOM =
1000282000,
  // Provided by VK_QCOM_render_pass_transform
    VK_STRUCTURE_TYPE_RENDER_PASS_TRANSFORM_BEGIN_INFO_QCOM = 1000282001,
```

```
  // Provided by VK_EXT_depth_bias_control
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DEPTH_BIAS_CONTROL_FEATURES_EXT = 1000283000,
  // Provided by VK_EXT_depth_bias_control
    VK_STRUCTURE_TYPE_DEPTH_BIAS_INFO_EXT = 1000283001,
  // Provided by VK_EXT_depth_bias_control
    VK_STRUCTURE_TYPE_DEPTH_BIAS_REPRESENTATION_INFO_EXT = 1000283002,
  // Provided by VK_EXT_device_memory_report
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DEVICE_MEMORY_REPORT_FEATURES_EXT = 1000284000,
  // Provided by VK_EXT_device_memory_report
    VK_STRUCTURE_TYPE_DEVICE_DEVICE_MEMORY_REPORT_CREATE_INFO_EXT = 1000284001,
  // Provided by VK_EXT_device_memory_report
    VK_STRUCTURE_TYPE_DEVICE_MEMORY_REPORT_CALLBACK_DATA_EXT = 1000284002,
  // Provided by VK_EXT_robustness2
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_ROBUSTNESS_2_FEATURES_EXT = 1000286000,
  // Provided by VK_EXT_robustness2
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_ROBUSTNESS_2_PROPERTIES_EXT = 1000286001,
  // Provided by VK_EXT_custom_border_color
    VK_STRUCTURE_TYPE_SAMPLER_CUSTOM_BORDER_COLOR_CREATE_INFO_EXT = 1000287000,
  // Provided by VK_EXT_custom_border_color
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_CUSTOM_BORDER_COLOR_PROPERTIES_EXT = 1000287001,
  // Provided by VK_EXT_custom_border_color
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_CUSTOM_BORDER_COLOR_FEATURES_EXT = 1000287002,
  // Provided by VK_KHR_pipeline_library
    VK_STRUCTURE_TYPE_PIPELINE_LIBRARY_CREATE_INFO_KHR = 1000290000,
  // Provided by VK_NV_present_barrier
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PRESENT_BARRIER_FEATURES_NV = 1000292000,
  // Provided by VK_NV_present_barrier
    VK_STRUCTURE_TYPE_SURFACE_CAPABILITIES_PRESENT_BARRIER_NV = 1000292001,
  // Provided by VK_NV_present_barrier
    VK_STRUCTURE_TYPE_SWAPCHAIN_PRESENT_BARRIER_CREATE_INFO_NV = 1000292002,
  // Provided by VK_KHR_present_id
    VK_STRUCTURE_TYPE_PRESENT_ID_KHR = 1000294000,
  // Provided by VK_KHR_present_id
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PRESENT_ID_FEATURES_KHR = 1000294001,
  // Provided by VK_KHR_video_encode_queue
    VK_STRUCTURE_TYPE_VIDEO_ENCODE_INFO_KHR = 1000299000,
  // Provided by VK_KHR_video_encode_queue
    VK_STRUCTURE_TYPE_VIDEO_ENCODE_RATE_CONTROL_INFO_KHR = 1000299001,
  // Provided by VK_KHR_video_encode_queue
    VK_STRUCTURE_TYPE_VIDEO_ENCODE_RATE_CONTROL_LAYER_INFO_KHR = 1000299002,
  // Provided by VK_KHR_video_encode_queue
    VK_STRUCTURE_TYPE_VIDEO_ENCODE_CAPABILITIES_KHR = 1000299003,
  // Provided by VK_KHR_video_encode_queue
    VK_STRUCTURE_TYPE_VIDEO_ENCODE_USAGE_INFO_KHR = 1000299004,
  // Provided by VK_KHR_video_encode_queue
    VK_STRUCTURE_TYPE_QUERY_POOL_VIDEO_ENCODE_FEEDBACK_CREATE_INFO_KHR = 1000299005,
  // Provided by VK_KHR_video_encode_queue
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_VIDEO_ENCODE_QUALITY_LEVEL_INFO_KHR =
1000299006,
  // Provided by VK_KHR_video_encode_queue
    VK_STRUCTURE_TYPE_VIDEO_ENCODE_QUALITY_LEVEL_PROPERTIES_KHR = 1000299007,
```

```
// Provided by VK_KHR_video_encode_queue
  VK_STRUCTURE_TYPE_VIDEO_ENCODE_QUALITY_LEVEL_INFO_KHR = 1000299008,
// Provided by VK_KHR_video_encode_queue
  VK_STRUCTURE_TYPE_VIDEO_ENCODE_SESSION_PARAMETERS_GET_INFO_KHR = 1000299009,
// Provided by VK_KHR_video_encode_queue
  VK_STRUCTURE_TYPE_VIDEO_ENCODE_SESSION_PARAMETERS_FEEDBACK_INFO_KHR = 1000299010,
// Provided by VK_NV_device_diagnostics_config
  VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DIAGNOSTICS_CONFIG_FEATURES_NV = 1000300000,
// Provided by VK_NV_device_diagnostics_config
  VK_STRUCTURE_TYPE_DEVICE_DIAGNOSTICS_CONFIG_CREATE_INFO_NV = 1000300001,
// Provided by VK_NV_cuda_kernel_launch
  VK_STRUCTURE_TYPE_CUDA_MODULE_CREATE_INFO_NV = 1000307000,
// Provided by VK_NV_cuda_kernel_launch
  VK_STRUCTURE_TYPE_CUDA_FUNCTION_CREATE_INFO_NV = 1000307001,
// Provided by VK_NV_cuda_kernel_launch
  VK_STRUCTURE_TYPE_CUDA_LAUNCH_INFO_NV = 1000307002,
// Provided by VK_NV_cuda_kernel_launch
  VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_CUDA_KERNEL_LAUNCH_FEATURES_NV = 1000307003,
// Provided by VK_NV_cuda_kernel_launch
  VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_CUDA_KERNEL_LAUNCH_PROPERTIES_NV = 1000307004,
// Provided by VK_NV_low_latency
  VK_STRUCTURE_TYPE_QUERY_LOW_LATENCY_SUPPORT_NV = 1000310000,
// Provided by VK_EXT_metal_objects
  VK_STRUCTURE_TYPE_EXPORT_METAL_OBJECT_CREATE_INFO_EXT = 1000311000,
// Provided by VK_EXT_metal_objects
  VK_STRUCTURE_TYPE_EXPORT_METAL_OBJECTS_INFO_EXT = 1000311001,
// Provided by VK_EXT_metal_objects
  VK_STRUCTURE_TYPE_EXPORT_METAL_DEVICE_INFO_EXT = 1000311002,
// Provided by VK_EXT_metal_objects
  VK_STRUCTURE_TYPE_EXPORT_METAL_COMMAND_QUEUE_INFO_EXT = 1000311003,
// Provided by VK_EXT_metal_objects
  VK_STRUCTURE_TYPE_EXPORT_METAL_BUFFER_INFO_EXT = 1000311004,
// Provided by VK_EXT_metal_objects
  VK_STRUCTURE_TYPE_IMPORT_METAL_BUFFER_INFO_EXT = 1000311005,
// Provided by VK_EXT_metal_objects
  VK_STRUCTURE_TYPE_EXPORT_METAL_TEXTURE_INFO_EXT = 1000311006,
// Provided by VK_EXT_metal_objects
  VK_STRUCTURE_TYPE_IMPORT_METAL_TEXTURE_INFO_EXT = 1000311007,
// Provided by VK_EXT_metal_objects
  VK_STRUCTURE_TYPE_EXPORT_METAL_IO_SURFACE_INFO_EXT = 1000311008,
// Provided by VK_EXT_metal_objects
  VK_STRUCTURE_TYPE_IMPORT_METAL_IO_SURFACE_INFO_EXT = 1000311009,
// Provided by VK_EXT_metal_objects
  VK_STRUCTURE_TYPE_EXPORT_METAL_SHARED_EVENT_INFO_EXT = 1000311010,
// Provided by VK_EXT_metal_objects
  VK_STRUCTURE_TYPE_IMPORT_METAL_SHARED_EVENT_INFO_EXT = 1000311011,
// Provided by VK_KHR_synchronization2 with VK_NV_device_diagnostic_checkpoints
  VK_STRUCTURE_TYPE_QUEUE_FAMILY_CHECKPOINT_PROPERTIES_2_NV = 1000314008,
// Provided by VK_KHR_synchronization2 with VK_NV_device_diagnostic_checkpoints
  VK_STRUCTURE_TYPE_CHECKPOINT_DATA_2_NV = 1000314009,
// Provided by VK_EXT_descriptor_buffer
```

```
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DESCRIPTOR_BUFFER_PROPERTIES_EXT = 1000316000,
  // Provided by VK_EXT_descriptor_buffer
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DESCRIPTOR_BUFFER_DENSITY_MAP_PROPERTIES_EXT =
1000316001,
  // Provided by VK_EXT_descriptor_buffer
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DESCRIPTOR_BUFFER_FEATURES_EXT = 1000316002,
  // Provided by VK_EXT_descriptor_buffer
    VK_STRUCTURE_TYPE_DESCRIPTOR_ADDRESS_INFO_EXT = 1000316003,
  // Provided by VK_EXT_descriptor_buffer
    VK_STRUCTURE_TYPE_DESCRIPTOR_GET_INFO_EXT = 1000316004,
  // Provided by VK_EXT_descriptor_buffer
    VK_STRUCTURE_TYPE_BUFFER_CAPTURE_DESCRIPTOR_DATA_INFO_EXT = 1000316005,
  // Provided by VK_EXT_descriptor_buffer
    VK_STRUCTURE_TYPE_IMAGE_CAPTURE_DESCRIPTOR_DATA_INFO_EXT = 1000316006,
  // Provided by VK_EXT_descriptor_buffer
    VK_STRUCTURE_TYPE_IMAGE_VIEW_CAPTURE_DESCRIPTOR_DATA_INFO_EXT = 1000316007,
  // Provided by VK_EXT_descriptor_buffer
    VK_STRUCTURE_TYPE_SAMPLER_CAPTURE_DESCRIPTOR_DATA_INFO_EXT = 1000316008,
  // Provided by VK_EXT_descriptor_buffer
    VK_STRUCTURE_TYPE_OPAQUE_CAPTURE_DESCRIPTOR_DATA_CREATE_INFO_EXT = 1000316010,
  // Provided by VK_EXT_descriptor_buffer
    VK_STRUCTURE_TYPE_DESCRIPTOR_BUFFER_BINDING_INFO_EXT = 1000316011,
  // Provided by VK_EXT_descriptor_buffer
    VK_STRUCTURE_TYPE_DESCRIPTOR_BUFFER_BINDING_PUSH_DESCRIPTOR_BUFFER_HANDLE_EXT =
1000316012,
  // Provided by VK_EXT_descriptor_buffer with VK_KHR_acceleration_structure or
VK_NV_ray_tracing
    VK_STRUCTURE_TYPE_ACCELERATION_STRUCTURE_CAPTURE_DESCRIPTOR_DATA_INFO_EXT =
1000316009,
  // Provided by VK_EXT_graphics_pipeline_library
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_GRAPHICS_PIPELINE_LIBRARY_FEATURES_EXT =
1000320000,
  // Provided by VK_EXT_graphics_pipeline_library
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_GRAPHICS_PIPELINE_LIBRARY_PROPERTIES_EXT =
1000320001,
  // Provided by VK_EXT_graphics_pipeline_library
    VK_STRUCTURE_TYPE_GRAPHICS_PIPELINE_LIBRARY_CREATE_INFO_EXT = 1000320002,
  // Provided by VK_AMD_shader_early_and_late_fragment_tests

VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_EARLY_AND_LATE_FRAGMENT_TESTS_FEATURES_AMD =
1000321000,
  // Provided by VK_KHR_fragment_shader_barycentric
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_FRAGMENT_SHADER_BARYCENTRIC_FEATURES_KHR =
1000203000,
  // Provided by VK_KHR_fragment_shader_barycentric
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_FRAGMENT_SHADER_BARYCENTRIC_PROPERTIES_KHR =
1000322000,
  // Provided by VK_KHR_shader_subgroup_uniform_control_flow

VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_SUBGROUP_UNIFORM_CONTROL_FLOW_FEATURES_KHR =
1000323000,
```

```
  // Provided by VK_NV_fragment_shading_rate_enums
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_FRAGMENT_SHADING_RATE_ENUMS_PROPERTIES_NV =
1000326000,
  // Provided by VK_NV_fragment_shading_rate_enums
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_FRAGMENT_SHADING_RATE_ENUMS_FEATURES_NV =
1000326001,
  // Provided by VK_NV_fragment_shading_rate_enums
    VK_STRUCTURE_TYPE_PIPELINE_FRAGMENT_SHADING_RATE_ENUM_STATE_CREATE_INFO_NV =
1000326002,
  // Provided by VK_NV_ray_tracing_motion_blur
    VK_STRUCTURE_TYPE_ACCELERATION_STRUCTURE_GEOMETRY_MOTION_TRIANGLES_DATA_NV =
1000327000,
  // Provided by VK_NV_ray_tracing_motion_blur
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_RAY_TRACING_MOTION_BLUR_FEATURES_NV =
1000327001,
  // Provided by VK_NV_ray_tracing_motion_blur
    VK_STRUCTURE_TYPE_ACCELERATION_STRUCTURE_MOTION_INFO_NV = 1000327002,
  // Provided by VK_EXT_mesh_shader
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MESH_SHADER_FEATURES_EXT = 1000328000,
  // Provided by VK_EXT_mesh_shader
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MESH_SHADER_PROPERTIES_EXT = 1000328001,
  // Provided by VK_EXT_ycbcr_2plane_444_formats
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_YCBCR_2_PLANE_444_FORMATS_FEATURES_EXT =
1000330000,
  // Provided by VK_EXT_fragment_density_map2
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_FRAGMENT_DENSITY_MAP_2_FEATURES_EXT =
1000332000,
  // Provided by VK_EXT_fragment_density_map2
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_FRAGMENT_DENSITY_MAP_2_PROPERTIES_EXT =
1000332001,
  // Provided by VK_QCOM_rotated_copy_commands
    VK_STRUCTURE_TYPE_COPY_COMMAND_TRANSFORM_INFO_QCOM = 1000333000,
  // Provided by VK_KHR_workgroup_memory_explicit_layout
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_WORKGROUP_MEMORY_EXPLICIT_LAYOUT_FEATURES_KHR =
1000336000,
  // Provided by VK_EXT_image_compression_control
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_IMAGE_COMPRESSION_CONTROL_FEATURES_EXT =
1000338000,
  // Provided by VK_EXT_image_compression_control
    VK_STRUCTURE_TYPE_IMAGE_COMPRESSION_CONTROL_EXT = 1000338001,
  // Provided by VK_EXT_image_compression_control
    VK_STRUCTURE_TYPE_IMAGE_COMPRESSION_PROPERTIES_EXT = 1000338004,
  // Provided by VK_EXT_attachment_feedback_loop_layout
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_ATTACHMENT_FEEDBACK_LOOP_LAYOUT_FEATURES_EXT =
1000339000,
  // Provided by VK_EXT_4444_formats
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_4444_FORMATS_FEATURES_EXT = 1000340000,
  // Provided by VK_EXT_device_fault
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_FAULT_FEATURES_EXT = 1000341000,
  // Provided by VK_EXT_device_fault
    VK_STRUCTURE_TYPE_DEVICE_FAULT_COUNTS_EXT = 1000341001,
```

```
  // Provided by VK_EXT_device_fault
    VK_STRUCTURE_TYPE_DEVICE_FAULT_INFO_EXT = 1000341002,
  // Provided by VK_EXT_rgba10x6_formats
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_RGBA10X6_FORMATS_FEATURES_EXT = 1000344000,
  // Provided by VK_EXT_directfb_surface
    VK_STRUCTURE_TYPE_DIRECTFB_SURFACE_CREATE_INFO_EXT = 1000346000,
  // Provided by VK_EXT_vertex_input_dynamic_state
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_VERTEX_INPUT_DYNAMIC_STATE_FEATURES_EXT =
1000352000,
  // Provided by VK_EXT_shader_object, VK_EXT_vertex_input_dynamic_state
    VK_STRUCTURE_TYPE_VERTEX_INPUT_BINDING_DESCRIPTION_2_EXT = 1000352001,
  // Provided by VK_EXT_shader_object, VK_EXT_vertex_input_dynamic_state
    VK_STRUCTURE_TYPE_VERTEX_INPUT_ATTRIBUTE_DESCRIPTION_2_EXT = 1000352002,
  // Provided by VK_EXT_physical_device_drm
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DRM_PROPERTIES_EXT = 1000353000,
  // Provided by VK_EXT_device_address_binding_report
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_ADDRESS_BINDING_REPORT_FEATURES_EXT =
1000354000,
  // Provided by VK_EXT_device_address_binding_report
    VK_STRUCTURE_TYPE_DEVICE_ADDRESS_BINDING_CALLBACK_DATA_EXT = 1000354001,
  // Provided by VK_EXT_depth_clip_control
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DEPTH_CLIP_CONTROL_FEATURES_EXT = 1000355000,
  // Provided by VK_EXT_depth_clip_control
    VK_STRUCTURE_TYPE_PIPELINE_VIEWPORT_DEPTH_CLIP_CONTROL_CREATE_INFO_EXT =
1000355001,
  // Provided by VK_EXT_primitive_topology_list_restart
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PRIMITIVE_TOPOLOGY_LIST_RESTART_FEATURES_EXT =
1000356000,
  // Provided by VK_FUCHSIA_external_memory
    VK_STRUCTURE_TYPE_IMPORT_MEMORY_ZIRCON_HANDLE_INFO_FUCHSIA = 1000364000,
  // Provided by VK_FUCHSIA_external_memory
    VK_STRUCTURE_TYPE_MEMORY_ZIRCON_HANDLE_PROPERTIES_FUCHSIA = 1000364001,
  // Provided by VK_FUCHSIA_external_memory
    VK_STRUCTURE_TYPE_MEMORY_GET_ZIRCON_HANDLE_INFO_FUCHSIA = 1000364002,
  // Provided by VK_FUCHSIA_external_semaphore
    VK_STRUCTURE_TYPE_IMPORT_SEMAPHORE_ZIRCON_HANDLE_INFO_FUCHSIA = 1000365000,
  // Provided by VK_FUCHSIA_external_semaphore
    VK_STRUCTURE_TYPE_SEMAPHORE_GET_ZIRCON_HANDLE_INFO_FUCHSIA = 1000365001,
  // Provided by VK_FUCHSIA_buffer_collection
    VK_STRUCTURE_TYPE_BUFFER_COLLECTION_CREATE_INFO_FUCHSIA = 1000366000,
  // Provided by VK_FUCHSIA_buffer_collection
    VK_STRUCTURE_TYPE_IMPORT_MEMORY_BUFFER_COLLECTION_FUCHSIA = 1000366001,
  // Provided by VK_FUCHSIA_buffer_collection
    VK_STRUCTURE_TYPE_BUFFER_COLLECTION_IMAGE_CREATE_INFO_FUCHSIA = 1000366002,
  // Provided by VK_FUCHSIA_buffer_collection
    VK_STRUCTURE_TYPE_BUFFER_COLLECTION_PROPERTIES_FUCHSIA = 1000366003,
  // Provided by VK_FUCHSIA_buffer_collection
    VK_STRUCTURE_TYPE_BUFFER_CONSTRAINTS_INFO_FUCHSIA = 1000366004,
  // Provided by VK_FUCHSIA_buffer_collection
    VK_STRUCTURE_TYPE_BUFFER_COLLECTION_BUFFER_CREATE_INFO_FUCHSIA = 1000366005,
  // Provided by VK_FUCHSIA_buffer_collection
```

```
    VK_STRUCTURE_TYPE_IMAGE_CONSTRAINTS_INFO_FUCHSIA = 1000366006,
  // Provided by VK_FUCHSIA_buffer_collection
    VK_STRUCTURE_TYPE_IMAGE_FORMAT_CONSTRAINTS_INFO_FUCHSIA = 1000366007,
  // Provided by VK_FUCHSIA_buffer_collection
    VK_STRUCTURE_TYPE_SYSMEM_COLOR_SPACE_FUCHSIA = 1000366008,
  // Provided by VK_FUCHSIA_buffer_collection
    VK_STRUCTURE_TYPE_BUFFER_COLLECTION_CONSTRAINTS_INFO_FUCHSIA = 1000366009,
  // Provided by VK_HUAWEI_subpass_shading
    VK_STRUCTURE_TYPE_SUBPASS_SHADING_PIPELINE_CREATE_INFO_HUAWEI = 1000369000,
  // Provided by VK_HUAWEI_subpass_shading
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SUBPASS_SHADING_FEATURES_HUAWEI = 1000369001,
  // Provided by VK_HUAWEI_subpass_shading
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SUBPASS_SHADING_PROPERTIES_HUAWEI = 1000369002,
  // Provided by VK_HUAWEI_invocation_mask
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_INVOCATION_MASK_FEATURES_HUAWEI = 1000370000,
  // Provided by VK_NV_external_memory_rdma
    VK_STRUCTURE_TYPE_MEMORY_GET_REMOTE_ADDRESS_INFO_NV = 1000371000,
  // Provided by VK_NV_external_memory_rdma
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTERNAL_MEMORY_RDMA_FEATURES_NV = 1000371001,
  // Provided by VK_EXT_pipeline_properties
    VK_STRUCTURE_TYPE_PIPELINE_PROPERTIES_IDENTIFIER_EXT = 1000372000,
  // Provided by VK_EXT_pipeline_properties
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PIPELINE_PROPERTIES_FEATURES_EXT = 1000372001,
  // Provided by VK_EXT_frame_boundary
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_FRAME_BOUNDARY_FEATURES_EXT = 1000375000,
  // Provided by VK_EXT_frame_boundary
    VK_STRUCTURE_TYPE_FRAME_BOUNDARY_EXT = 1000375001,
  // Provided by VK_EXT_multisampled_render_to_single_sampled

VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MULTISAMPLED_RENDER_TO_SINGLE_SAMPLED_FEATURES_EXT =
1000376000,
  // Provided by VK_EXT_multisampled_render_to_single_sampled
    VK_STRUCTURE_TYPE_SUBPASS_RESOLVE_PERFORMANCE_QUERY_EXT = 1000376001,
  // Provided by VK_EXT_multisampled_render_to_single_sampled
    VK_STRUCTURE_TYPE_MULTISAMPLED_RENDER_TO_SINGLE_SAMPLED_INFO_EXT = 1000376002,
  // Provided by VK_EXT_extended_dynamic_state2
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTENDED_DYNAMIC_STATE_2_FEATURES_EXT =
1000377000,
  // Provided by VK_QNX_screen_surface
    VK_STRUCTURE_TYPE_SCREEN_SURFACE_CREATE_INFO_QNX = 1000378000,
  // Provided by VK_EXT_color_write_enable
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_COLOR_WRITE_ENABLE_FEATURES_EXT = 1000381000,
  // Provided by VK_EXT_color_write_enable
    VK_STRUCTURE_TYPE_PIPELINE_COLOR_WRITE_CREATE_INFO_EXT = 1000381001,
  // Provided by VK_EXT_primitives_generated_query
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PRIMITIVES_GENERATED_QUERY_FEATURES_EXT =
1000382000,
  // Provided by VK_KHR_ray_tracing_maintenance1
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_RAY_TRACING_MAINTENANCE_1_FEATURES_KHR =
1000386000,
  // Provided by VK_EXT_image_view_min_lod
```

```
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_IMAGE_VIEW_MIN_LOD_FEATURES_EXT = 1000391000,
  // Provided by VK_EXT_image_view_min_lod
    VK_STRUCTURE_TYPE_IMAGE_VIEW_MIN_LOD_CREATE_INFO_EXT = 1000391001,
  // Provided by VK_EXT_multi_draw
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MULTI_DRAW_FEATURES_EXT = 1000392000,
  // Provided by VK_EXT_multi_draw
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MULTI_DRAW_PROPERTIES_EXT = 1000392001,
  // Provided by VK_EXT_image_2d_view_of_3d
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_IMAGE_2D_VIEW_OF_3D_FEATURES_EXT = 1000393000,
  // Provided by VK_EXT_shader_tile_image
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_TILE_IMAGE_FEATURES_EXT = 1000395000,
  // Provided by VK_EXT_shader_tile_image
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_TILE_IMAGE_PROPERTIES_EXT = 1000395001,
  // Provided by VK_EXT_opacity_micromap
    VK_STRUCTURE_TYPE_MICROMAP_BUILD_INFO_EXT = 1000396000,
  // Provided by VK_EXT_opacity_micromap
    VK_STRUCTURE_TYPE_MICROMAP_VERSION_INFO_EXT = 1000396001,
  // Provided by VK_EXT_opacity_micromap
    VK_STRUCTURE_TYPE_COPY_MICROMAP_INFO_EXT = 1000396002,
  // Provided by VK_EXT_opacity_micromap
    VK_STRUCTURE_TYPE_COPY_MICROMAP_TO_MEMORY_INFO_EXT = 1000396003,
  // Provided by VK_EXT_opacity_micromap
    VK_STRUCTURE_TYPE_COPY_MEMORY_TO_MICROMAP_INFO_EXT = 1000396004,
  // Provided by VK_EXT_opacity_micromap
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_OPACITY_MICROMAP_FEATURES_EXT = 1000396005,
  // Provided by VK_EXT_opacity_micromap
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_OPACITY_MICROMAP_PROPERTIES_EXT = 1000396006,
  // Provided by VK_EXT_opacity_micromap
    VK_STRUCTURE_TYPE_MICROMAP_CREATE_INFO_EXT = 1000396007,
  // Provided by VK_EXT_opacity_micromap
    VK_STRUCTURE_TYPE_MICROMAP_BUILD_SIZES_INFO_EXT = 1000396008,
  // Provided by VK_EXT_opacity_micromap
    VK_STRUCTURE_TYPE_ACCELERATION_STRUCTURE_TRIANGLES_OPACITY_MICROMAP_EXT =
1000396009,
#ifdef VK_ENABLE_BETA_EXTENSIONS
  // Provided by VK_NV_displacement_micromap
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DISPLACEMENT_MICROMAP_FEATURES_NV = 1000397000,
#endif
#ifdef VK_ENABLE_BETA_EXTENSIONS
  // Provided by VK_NV_displacement_micromap
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DISPLACEMENT_MICROMAP_PROPERTIES_NV =
1000397001,
#endif
#ifdef VK_ENABLE_BETA_EXTENSIONS
  // Provided by VK_NV_displacement_micromap
    VK_STRUCTURE_TYPE_ACCELERATION_STRUCTURE_TRIANGLES_DISPLACEMENT_MICROMAP_NV =
1000397002,
#endif
  // Provided by VK_HUAWEI_cluster_culling_shader
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_CLUSTER_CULLING_SHADER_FEATURES_HUAWEI =
1000404000,
```

```
  // Provided by VK_HUAWEI_cluster_culling_shader
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_CLUSTER_CULLING_SHADER_PROPERTIES_HUAWEI =
1000404001,
  // Provided by VK_HUAWEI_cluster_culling_shader
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_CLUSTER_CULLING_SHADER_VRS_FEATURES_HUAWEI =
1000404002,
  // Provided by VK_EXT_border_color_swizzle
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_BORDER_COLOR_SWIZZLE_FEATURES_EXT = 1000411000,
  // Provided by VK_EXT_border_color_swizzle
    VK_STRUCTURE_TYPE_SAMPLER_BORDER_COLOR_COMPONENT_MAPPING_CREATE_INFO_EXT =
1000411001,
  // Provided by VK_EXT_pageable_device_local_memory
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PAGEABLE_DEVICE_LOCAL_MEMORY_FEATURES_EXT =
1000412000,
  // Provided by VK_ARM_shader_core_properties
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_CORE_PROPERTIES_ARM = 1000415000,
  // Provided by VK_KHR_shader_subgroup_rotate
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_SUBGROUP_ROTATE_FEATURES_KHR =
1000416000,
  // Provided by VK_ARM_scheduling_controls
    VK_STRUCTURE_TYPE_DEVICE_QUEUE_SHADER_CORE_CONTROL_CREATE_INFO_ARM = 1000417000,
  // Provided by VK_ARM_scheduling_controls
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SCHEDULING_CONTROLS_FEATURES_ARM = 1000417001,
  // Provided by VK_ARM_scheduling_controls
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SCHEDULING_CONTROLS_PROPERTIES_ARM = 1000417002,
  // Provided by VK_EXT_image_sliced_view_of_3d
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_IMAGE_SLICED_VIEW_OF_3D_FEATURES_EXT =
1000418000,
  // Provided by VK_EXT_image_sliced_view_of_3d
    VK_STRUCTURE_TYPE_IMAGE_VIEW_SLICED_CREATE_INFO_EXT = 1000418001,
  // Provided by VK_VALVE_descriptor_set_host_mapping
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DESCRIPTOR_SET_HOST_MAPPING_FEATURES_VALVE =
1000420000,
  // Provided by VK_VALVE_descriptor_set_host_mapping
    VK_STRUCTURE_TYPE_DESCRIPTOR_SET_BINDING_REFERENCE_VALVE = 1000420001,
  // Provided by VK_VALVE_descriptor_set_host_mapping
    VK_STRUCTURE_TYPE_DESCRIPTOR_SET_LAYOUT_HOST_MAPPING_INFO_VALVE = 1000420002,
  // Provided by VK_EXT_depth_clamp_zero_one
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DEPTH_CLAMP_ZERO_ONE_FEATURES_EXT = 1000421000,
  // Provided by VK_EXT_non_seamless_cube_map
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_NON_SEAMLESS_CUBE_MAP_FEATURES_EXT = 1000422000,
  // Provided by VK_ARM_render_pass_striped
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_RENDER_PASS_STRIPED_FEATURES_ARM = 1000424000,
  // Provided by VK_ARM_render_pass_striped
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_RENDER_PASS_STRIPED_PROPERTIES_ARM = 1000424001,
  // Provided by VK_ARM_render_pass_striped
    VK_STRUCTURE_TYPE_RENDER_PASS_STRIPE_BEGIN_INFO_ARM = 1000424002,
  // Provided by VK_ARM_render_pass_striped
    VK_STRUCTURE_TYPE_RENDER_PASS_STRIPE_INFO_ARM = 1000424003,
  // Provided by VK_ARM_render_pass_striped
    VK_STRUCTURE_TYPE_RENDER_PASS_STRIPE_SUBMIT_INFO_ARM = 1000424004,
```

```
  // Provided by VK_QCOM_fragment_density_map_offset
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_FRAGMENT_DENSITY_MAP_OFFSET_FEATURES_QCOM =
1000425000,
  // Provided by VK_QCOM_fragment_density_map_offset
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_FRAGMENT_DENSITY_MAP_OFFSET_PROPERTIES_QCOM =
1000425001,
  // Provided by VK_QCOM_fragment_density_map_offset
    VK_STRUCTURE_TYPE_SUBPASS_FRAGMENT_DENSITY_MAP_OFFSET_END_INFO_QCOM = 1000425002,
  // Provided by VK_NV_copy_memory_indirect
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_COPY_MEMORY_INDIRECT_FEATURES_NV = 1000426000,
  // Provided by VK_NV_copy_memory_indirect
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_COPY_MEMORY_INDIRECT_PROPERTIES_NV = 1000426001,
  // Provided by VK_NV_memory_decompression
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MEMORY_DECOMPRESSION_FEATURES_NV = 1000427000,
  // Provided by VK_NV_memory_decompression
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MEMORY_DECOMPRESSION_PROPERTIES_NV = 1000427001,
  // Provided by VK_NV_device_generated_commands_compute
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DEVICE_GENERATED_COMMANDS_COMPUTE_FEATURES_NV =
1000428000,
  // Provided by VK_NV_device_generated_commands_compute
    VK_STRUCTURE_TYPE_COMPUTE_PIPELINE_INDIRECT_BUFFER_INFO_NV = 1000428001,
  // Provided by VK_NV_device_generated_commands_compute
    VK_STRUCTURE_TYPE_PIPELINE_INDIRECT_DEVICE_ADDRESS_INFO_NV = 1000428002,
  // Provided by VK_NV_linear_color_attachment
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_LINEAR_COLOR_ATTACHMENT_FEATURES_NV =
1000430000,
  // Provided by VK_KHR_shader_maximal_reconvergence
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_MAXIMAL_RECONVERGENCE_FEATURES_KHR =
1000434000,
  // Provided by VK_EXT_image_compression_control_swapchain
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_IMAGE_COMPRESSION_CONTROL_SWAPCHAIN_FEATURES_EXT
= 1000437000,
  // Provided by VK_QCOM_image_processing
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_IMAGE_PROCESSING_FEATURES_QCOM = 1000440000,
  // Provided by VK_QCOM_image_processing
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_IMAGE_PROCESSING_PROPERTIES_QCOM = 1000440001,
  // Provided by VK_QCOM_image_processing
    VK_STRUCTURE_TYPE_IMAGE_VIEW_SAMPLE_WEIGHT_CREATE_INFO_QCOM = 1000440002,
  // Provided by VK_EXT_nested_command_buffer
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_NESTED_COMMAND_BUFFER_FEATURES_EXT = 1000451000,
  // Provided by VK_EXT_nested_command_buffer
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_NESTED_COMMAND_BUFFER_PROPERTIES_EXT =
1000451001,
  // Provided by VK_EXT_external_memory_acquire_unmodified
    VK_STRUCTURE_TYPE_EXTERNAL_MEMORY_ACQUIRE_UNMODIFIED_EXT = 1000453000,
  // Provided by VK_EXT_extended_dynamic_state3
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTENDED_DYNAMIC_STATE_3_FEATURES_EXT =
1000455000,
  // Provided by VK_EXT_extended_dynamic_state3
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTENDED_DYNAMIC_STATE_3_PROPERTIES_EXT =
1000455001,
```

```
  // Provided by VK_EXT_subpass_merge_feedback
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SUBPASS_MERGE_FEEDBACK_FEATURES_EXT =
1000458000,
  // Provided by VK_EXT_subpass_merge_feedback
    VK_STRUCTURE_TYPE_RENDER_PASS_CREATION_CONTROL_EXT = 1000458001,
  // Provided by VK_EXT_subpass_merge_feedback
    VK_STRUCTURE_TYPE_RENDER_PASS_CREATION_FEEDBACK_CREATE_INFO_EXT = 1000458002,
  // Provided by VK_EXT_subpass_merge_feedback
    VK_STRUCTURE_TYPE_RENDER_PASS_SUBPASS_FEEDBACK_CREATE_INFO_EXT = 1000458003,
  // Provided by VK_LUNARG_direct_driver_loading
    VK_STRUCTURE_TYPE_DIRECT_DRIVER_LOADING_INFO_LUNARG = 1000459000,
  // Provided by VK_LUNARG_direct_driver_loading
    VK_STRUCTURE_TYPE_DIRECT_DRIVER_LOADING_LIST_LUNARG = 1000459001,
  // Provided by VK_EXT_shader_module_identifier
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_MODULE_IDENTIFIER_FEATURES_EXT =
1000462000,
  // Provided by VK_EXT_shader_module_identifier
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_MODULE_IDENTIFIER_PROPERTIES_EXT =
1000462001,
  // Provided by VK_EXT_shader_module_identifier
    VK_STRUCTURE_TYPE_PIPELINE_SHADER_STAGE_MODULE_IDENTIFIER_CREATE_INFO_EXT =
1000462002,
  // Provided by VK_EXT_shader_module_identifier
    VK_STRUCTURE_TYPE_SHADER_MODULE_IDENTIFIER_EXT = 1000462003,
  // Provided by VK_EXT_rasterization_order_attachment_access

VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_RASTERIZATION_ORDER_ATTACHMENT_ACCESS_FEATURES_EXT =
1000342000,
  // Provided by VK_NV_optical_flow
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_OPTICAL_FLOW_FEATURES_NV = 1000464000,
  // Provided by VK_NV_optical_flow
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_OPTICAL_FLOW_PROPERTIES_NV = 1000464001,
  // Provided by VK_NV_optical_flow
    VK_STRUCTURE_TYPE_OPTICAL_FLOW_IMAGE_FORMAT_INFO_NV = 1000464002,
  // Provided by VK_NV_optical_flow
    VK_STRUCTURE_TYPE_OPTICAL_FLOW_IMAGE_FORMAT_PROPERTIES_NV = 1000464003,
  // Provided by VK_NV_optical_flow
    VK_STRUCTURE_TYPE_OPTICAL_FLOW_SESSION_CREATE_INFO_NV = 1000464004,
  // Provided by VK_NV_optical_flow
    VK_STRUCTURE_TYPE_OPTICAL_FLOW_EXECUTE_INFO_NV = 1000464005,
  // Provided by VK_NV_optical_flow
    VK_STRUCTURE_TYPE_OPTICAL_FLOW_SESSION_CREATE_PRIVATE_DATA_INFO_NV = 1000464010,
  // Provided by VK_EXT_legacy_dithering
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_LEGACY_DITHERING_FEATURES_EXT = 1000465000,
  // Provided by VK_EXT_pipeline_protected_access
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PIPELINE_PROTECTED_ACCESS_FEATURES_EXT =
1000466000,
  // Provided by VK_ANDROID_external_format_resolve
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTERNAL_FORMAT_RESOLVE_FEATURES_ANDROID =
1000468000,
  // Provided by VK_ANDROID_external_format_resolve
```

```
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTERNAL_FORMAT_RESOLVE_PROPERTIES_ANDROID =
1000468001,
  // Provided by VK_ANDROID_external_format_resolve
    VK_STRUCTURE_TYPE_ANDROID_HARDWARE_BUFFER_FORMAT_RESOLVE_PROPERTIES_ANDROID =
1000468002,
  // Provided by VK_KHR_maintenance5
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MAINTENANCE_5_FEATURES_KHR = 1000470000,
  // Provided by VK_KHR_maintenance5
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MAINTENANCE_5_PROPERTIES_KHR = 1000470001,
  // Provided by VK_KHR_maintenance5
    VK_STRUCTURE_TYPE_RENDERING_AREA_INFO_KHR = 1000470003,
  // Provided by VK_KHR_maintenance5
    VK_STRUCTURE_TYPE_DEVICE_IMAGE_SUBRESOURCE_INFO_KHR = 1000470004,
  // Provided by VK_KHR_maintenance5
    VK_STRUCTURE_TYPE_SUBRESOURCE_LAYOUT_2_KHR = 1000338002,
  // Provided by VK_KHR_maintenance5
    VK_STRUCTURE_TYPE_IMAGE_SUBRESOURCE_2_KHR = 1000338003,
  // Provided by VK_KHR_maintenance5
    VK_STRUCTURE_TYPE_PIPELINE_CREATE_FLAGS_2_CREATE_INFO_KHR = 1000470005,
  // Provided by VK_KHR_maintenance5
    VK_STRUCTURE_TYPE_BUFFER_USAGE_FLAGS_2_CREATE_INFO_KHR = 1000470006,
  // Provided by VK_AMD_anti_lag
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_ANTI_LAG_FEATURES_AMD = 1000476000,
  // Provided by VK_AMD_anti_lag
    VK_STRUCTURE_TYPE_ANTI_LAG_DATA_AMD = 1000476001,
  // Provided by VK_AMD_anti_lag
    VK_STRUCTURE_TYPE_ANTI_LAG_PRESENTATION_INFO_AMD = 1000476002,
  // Provided by VK_KHR_ray_tracing_position_fetch
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_RAY_TRACING_POSITION_FETCH_FEATURES_KHR =
1000481000,
  // Provided by VK_EXT_shader_object
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_OBJECT_FEATURES_EXT = 1000482000,
  // Provided by VK_EXT_shader_object
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_OBJECT_PROPERTIES_EXT = 1000482001,
  // Provided by VK_EXT_shader_object
    VK_STRUCTURE_TYPE_SHADER_CREATE_INFO_EXT = 1000482002,
  // Provided by VK_KHR_pipeline_binary
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PIPELINE_BINARY_FEATURES_KHR = 1000483000,
  // Provided by VK_KHR_pipeline_binary
    VK_STRUCTURE_TYPE_PIPELINE_BINARY_CREATE_INFO_KHR = 1000483001,
  // Provided by VK_KHR_pipeline_binary
    VK_STRUCTURE_TYPE_PIPELINE_BINARY_INFO_KHR = 1000483002,
  // Provided by VK_KHR_pipeline_binary
    VK_STRUCTURE_TYPE_PIPELINE_BINARY_KEY_KHR = 1000483003,
  // Provided by VK_KHR_pipeline_binary
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PIPELINE_BINARY_PROPERTIES_KHR = 1000483004,
  // Provided by VK_KHR_pipeline_binary
    VK_STRUCTURE_TYPE_RELEASE_CAPTURED_PIPELINE_DATA_INFO_KHR = 1000483005,
  // Provided by VK_KHR_pipeline_binary
    VK_STRUCTURE_TYPE_PIPELINE_BINARY_DATA_INFO_KHR = 1000483006,
  // Provided by VK_KHR_pipeline_binary
```

```
    VK_STRUCTURE_TYPE_PIPELINE_CREATE_INFO_KHR = 1000483007,
  // Provided by VK_KHR_pipeline_binary
    VK_STRUCTURE_TYPE_DEVICE_PIPELINE_BINARY_INTERNAL_CACHE_CONTROL_KHR = 1000483008,
  // Provided by VK_KHR_pipeline_binary
    VK_STRUCTURE_TYPE_PIPELINE_BINARY_HANDLES_INFO_KHR = 1000483009,
  // Provided by VK_QCOM_tile_properties
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_TILE_PROPERTIES_FEATURES_QCOM = 1000484000,
  // Provided by VK_QCOM_tile_properties
    VK_STRUCTURE_TYPE_TILE_PROPERTIES_QCOM = 1000484001,
  // Provided by VK_SEC_amigo_profiling
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_AMIGO_PROFILING_FEATURES_SEC = 1000485000,
  // Provided by VK_SEC_amigo_profiling
    VK_STRUCTURE_TYPE_AMIGO_PROFILING_SUBMIT_INFO_SEC = 1000485001,
  // Provided by VK_QCOM_multiview_per_view_viewports
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MULTIVIEW_PER_VIEW_VIEWPORTS_FEATURES_QCOM =
1000488000,
  // Provided by VK_NV_ray_tracing_invocation_reorder
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_RAY_TRACING_INVOCATION_REORDER_FEATURES_NV =
1000490000,
  // Provided by VK_NV_ray_tracing_invocation_reorder
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_RAY_TRACING_INVOCATION_REORDER_PROPERTIES_NV =
1000490001,
  // Provided by VK_NV_extended_sparse_address_space
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTENDED_SPARSE_ADDRESS_SPACE_FEATURES_NV =
1000492000,
  // Provided by VK_NV_extended_sparse_address_space
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTENDED_SPARSE_ADDRESS_SPACE_PROPERTIES_NV =
1000492001,
  // Provided by VK_EXT_mutable_descriptor_type
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MUTABLE_DESCRIPTOR_TYPE_FEATURES_EXT =
1000351000,
  // Provided by VK_EXT_mutable_descriptor_type
    VK_STRUCTURE_TYPE_MUTABLE_DESCRIPTOR_TYPE_CREATE_INFO_EXT = 1000351002,
  // Provided by VK_EXT_legacy_vertex_attributes
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_LEGACY_VERTEX_ATTRIBUTES_FEATURES_EXT =
1000495000,
  // Provided by VK_EXT_legacy_vertex_attributes
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_LEGACY_VERTEX_ATTRIBUTES_PROPERTIES_EXT =
1000495001,
  // Provided by VK_EXT_layer_settings
    VK_STRUCTURE_TYPE_LAYER_SETTINGS_CREATE_INFO_EXT = 1000496000,
  // Provided by VK_ARM_shader_core_builtins
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_CORE_BUILTINS_FEATURES_ARM = 1000497000,
  // Provided by VK_ARM_shader_core_builtins
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_CORE_BUILTINS_PROPERTIES_ARM =
1000497001,
  // Provided by VK_EXT_pipeline_library_group_handles
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PIPELINE_LIBRARY_GROUP_HANDLES_FEATURES_EXT =
1000498000,
  // Provided by VK_EXT_dynamic_rendering_unused_attachments
```

```
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DYNAMIC_RENDERING_UNUSED_ATTACHMENTS_FEATURES_EXT =
1000499000,
  // Provided by VK_NV_low_latency2
    VK_STRUCTURE_TYPE_LATENCY_SLEEP_MODE_INFO_NV = 1000505000,
  // Provided by VK_NV_low_latency2
    VK_STRUCTURE_TYPE_LATENCY_SLEEP_INFO_NV = 1000505001,
  // Provided by VK_NV_low_latency2
    VK_STRUCTURE_TYPE_SET_LATENCY_MARKER_INFO_NV = 1000505002,
  // Provided by VK_NV_low_latency2
    VK_STRUCTURE_TYPE_GET_LATENCY_MARKER_INFO_NV = 1000505003,
  // Provided by VK_NV_low_latency2
    VK_STRUCTURE_TYPE_LATENCY_TIMINGS_FRAME_REPORT_NV = 1000505004,
  // Provided by VK_NV_low_latency2
    VK_STRUCTURE_TYPE_LATENCY_SUBMISSION_PRESENT_ID_NV = 1000505005,
  // Provided by VK_NV_low_latency2
    VK_STRUCTURE_TYPE_OUT_OF_BAND_QUEUE_TYPE_INFO_NV = 1000505006,
  // Provided by VK_NV_low_latency2
    VK_STRUCTURE_TYPE_SWAPCHAIN_LATENCY_CREATE_INFO_NV = 1000505007,
  // Provided by VK_NV_low_latency2
    VK_STRUCTURE_TYPE_LATENCY_SURFACE_CAPABILITIES_NV = 1000505008,
  // Provided by VK_KHR_cooperative_matrix
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_COOPERATIVE_MATRIX_FEATURES_KHR = 1000506000,
  // Provided by VK_KHR_cooperative_matrix
    VK_STRUCTURE_TYPE_COOPERATIVE_MATRIX_PROPERTIES_KHR = 1000506001,
  // Provided by VK_KHR_cooperative_matrix
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_COOPERATIVE_MATRIX_PROPERTIES_KHR = 1000506002,
  // Provided by VK_QCOM_multiview_per_view_render_areas
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MULTIVIEW_PER_VIEW_RENDER_AREAS_FEATURES_QCOM =
1000510000,
  // Provided by VK_QCOM_multiview_per_view_render_areas
    VK_STRUCTURE_TYPE_MULTIVIEW_PER_VIEW_RENDER_AREAS_RENDER_PASS_BEGIN_INFO_QCOM =
1000510001,
  // Provided by VK_KHR_compute_shader_derivatives
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_COMPUTE_SHADER_DERIVATIVES_FEATURES_KHR =
1000201000,
  // Provided by VK_KHR_compute_shader_derivatives
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_COMPUTE_SHADER_DERIVATIVES_PROPERTIES_KHR =
1000511000,
  // Provided by VK_KHR_video_decode_av1
    VK_STRUCTURE_TYPE_VIDEO_DECODE_AV1_CAPABILITIES_KHR = 1000512000,
  // Provided by VK_KHR_video_decode_av1
    VK_STRUCTURE_TYPE_VIDEO_DECODE_AV1_PICTURE_INFO_KHR = 1000512001,
  // Provided by VK_KHR_video_decode_av1
    VK_STRUCTURE_TYPE_VIDEO_DECODE_AV1_PROFILE_INFO_KHR = 1000512003,
  // Provided by VK_KHR_video_decode_av1
    VK_STRUCTURE_TYPE_VIDEO_DECODE_AV1_SESSION_PARAMETERS_CREATE_INFO_KHR =
1000512004,
  // Provided by VK_KHR_video_decode_av1
    VK_STRUCTURE_TYPE_VIDEO_DECODE_AV1_DPB_SLOT_INFO_KHR = 1000512005,
  // Provided by VK_KHR_video_maintenance1
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_VIDEO_MAINTENANCE_1_FEATURES_KHR = 1000515000,
```

```
  // Provided by VK_KHR_video_maintenance1
    VK_STRUCTURE_TYPE_VIDEO_INLINE_QUERY_INFO_KHR = 1000515001,
  // Provided by VK_NV_per_stage_descriptor_set
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PER_STAGE_DESCRIPTOR_SET_FEATURES_NV =
1000516000,
  // Provided by VK_QCOM_image_processing2
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_IMAGE_PROCESSING_2_FEATURES_QCOM = 1000518000,
  // Provided by VK_QCOM_image_processing2
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_IMAGE_PROCESSING_2_PROPERTIES_QCOM = 1000518001,
  // Provided by VK_QCOM_image_processing2
    VK_STRUCTURE_TYPE_SAMPLER_BLOCK_MATCH_WINDOW_CREATE_INFO_QCOM = 1000518002,
  // Provided by VK_QCOM_filter_cubic_weights
    VK_STRUCTURE_TYPE_SAMPLER_CUBIC_WEIGHTS_CREATE_INFO_QCOM = 1000519000,
  // Provided by VK_QCOM_filter_cubic_weights
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_CUBIC_WEIGHTS_FEATURES_QCOM = 1000519001,
  // Provided by VK_QCOM_filter_cubic_weights
    VK_STRUCTURE_TYPE_BLIT_IMAGE_CUBIC_WEIGHTS_INFO_QCOM = 1000519002,
  // Provided by VK_QCOM_ycbcr_degamma
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_YCBCR_DEGAMMA_FEATURES_QCOM = 1000520000,
  // Provided by VK_QCOM_ycbcr_degamma
    VK_STRUCTURE_TYPE_SAMPLER_YCBCR_CONVERSION_YCBCR_DEGAMMA_CREATE_INFO_QCOM =
1000520001,
  // Provided by VK_QCOM_filter_cubic_clamp
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_CUBIC_CLAMP_FEATURES_QCOM = 1000521000,
  // Provided by VK_EXT_attachment_feedback_loop_dynamic_state

VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_ATTACHMENT_FEEDBACK_LOOP_DYNAMIC_STATE_FEATURES_EXT
= 1000524000,
  // Provided by VK_KHR_vertex_attribute_divisor
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_VERTEX_ATTRIBUTE_DIVISOR_PROPERTIES_KHR =
1000525000,
  // Provided by VK_KHR_vertex_attribute_divisor
    VK_STRUCTURE_TYPE_PIPELINE_VERTEX_INPUT_DIVISOR_STATE_CREATE_INFO_KHR =
1000190001,
  // Provided by VK_KHR_vertex_attribute_divisor
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_VERTEX_ATTRIBUTE_DIVISOR_FEATURES_KHR =
1000190002,
  // Provided by VK_KHR_shader_float_controls2
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_FLOAT_CONTROLS_2_FEATURES_KHR =
1000528000,
  // Provided by VK_QNX_external_memory_screen_buffer
    VK_STRUCTURE_TYPE_SCREEN_BUFFER_PROPERTIES_QNX = 1000529000,
  // Provided by VK_QNX_external_memory_screen_buffer
    VK_STRUCTURE_TYPE_SCREEN_BUFFER_FORMAT_PROPERTIES_QNX = 1000529001,
  // Provided by VK_QNX_external_memory_screen_buffer
    VK_STRUCTURE_TYPE_IMPORT_SCREEN_BUFFER_INFO_QNX = 1000529002,
  // Provided by VK_QNX_external_memory_screen_buffer
    VK_STRUCTURE_TYPE_EXTERNAL_FORMAT_QNX = 1000529003,
  // Provided by VK_QNX_external_memory_screen_buffer
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTERNAL_MEMORY_SCREEN_BUFFER_FEATURES_QNX =
1000529004,
```

```
  // Provided by VK_MSFT_layered_driver
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_LAYERED_DRIVER_PROPERTIES_MSFT = 1000530000,
  // Provided by VK_KHR_index_type_uint8
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_INDEX_TYPE_UINT8_FEATURES_KHR = 1000265000,
  // Provided by VK_KHR_line_rasterization
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_LINE_RASTERIZATION_FEATURES_KHR = 1000259000,
  // Provided by VK_KHR_line_rasterization
    VK_STRUCTURE_TYPE_PIPELINE_RASTERIZATION_LINE_STATE_CREATE_INFO_KHR = 1000259001,
  // Provided by VK_KHR_line_rasterization
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_LINE_RASTERIZATION_PROPERTIES_KHR = 1000259002,
  // Provided by VK_KHR_calibrated_timestamps
    VK_STRUCTURE_TYPE_CALIBRATED_TIMESTAMP_INFO_KHR = 1000184000,
  // Provided by VK_KHR_shader_expect_assume
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_EXPECT_ASSUME_FEATURES_KHR = 1000544000,
  // Provided by VK_KHR_maintenance6
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MAINTENANCE_6_FEATURES_KHR = 1000545000,
  // Provided by VK_KHR_maintenance6
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MAINTENANCE_6_PROPERTIES_KHR = 1000545001,
  // Provided by VK_KHR_maintenance6
    VK_STRUCTURE_TYPE_BIND_MEMORY_STATUS_KHR = 1000545002,
  // Provided by VK_KHR_maintenance6
    VK_STRUCTURE_TYPE_BIND_DESCRIPTOR_SETS_INFO_KHR = 1000545003,
  // Provided by VK_KHR_maintenance6
    VK_STRUCTURE_TYPE_PUSH_CONSTANTS_INFO_KHR = 1000545004,
  // Provided by VK_KHR_maintenance6 with VK_KHR_push_descriptor
    VK_STRUCTURE_TYPE_PUSH_DESCRIPTOR_SET_INFO_KHR = 1000545005,
  // Provided by VK_KHR_maintenance6 with VK_KHR_push_descriptor
    VK_STRUCTURE_TYPE_PUSH_DESCRIPTOR_SET_WITH_TEMPLATE_INFO_KHR = 1000545006,
  // Provided by VK_KHR_maintenance6 with VK_EXT_descriptor_buffer
    VK_STRUCTURE_TYPE_SET_DESCRIPTOR_BUFFER_OFFSETS_INFO_EXT = 1000545007,
  // Provided by VK_KHR_maintenance6 with VK_EXT_descriptor_buffer
    VK_STRUCTURE_TYPE_BIND_DESCRIPTOR_BUFFER_EMBEDDED_SAMPLERS_INFO_EXT = 1000545008,
  // Provided by VK_NV_descriptor_pool_overallocation
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DESCRIPTOR_POOL_OVERALLOCATION_FEATURES_NV =
1000546000,
  // Provided by VK_NV_raw_access_chains
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_RAW_ACCESS_CHAINS_FEATURES_NV = 1000555000,
  // Provided by VK_KHR_shader_relaxed_extended_instruction
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_RELAXED_EXTENDED_INSTRUCTION_FEATURES_KHR
= 1000558000,
  // Provided by VK_NV_command_buffer_inheritance
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_COMMAND_BUFFER_INHERITANCE_FEATURES_NV =
1000559000,
  // Provided by VK_KHR_maintenance7
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MAINTENANCE_7_FEATURES_KHR = 1000562000,
  // Provided by VK_KHR_maintenance7
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MAINTENANCE_7_PROPERTIES_KHR = 1000562001,
  // Provided by VK_KHR_maintenance7
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_LAYERED_API_PROPERTIES_LIST_KHR = 1000562002,
  // Provided by VK_KHR_maintenance7
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_LAYERED_API_PROPERTIES_KHR = 1000562003,
```

```
  // Provided by VK_KHR_maintenance7
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_LAYERED_API_VULKAN_PROPERTIES_KHR = 1000562004,
  // Provided by VK_NV_shader_atomic_float16_vector
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_ATOMIC_FLOAT16_VECTOR_FEATURES_NV =
1000563000,
  // Provided by VK_EXT_shader_replicated_composites
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_REPLICATED_COMPOSITES_FEATURES_EXT =
1000564000,
  // Provided by VK_NV_ray_tracing_validation
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_RAY_TRACING_VALIDATION_FEATURES_NV = 1000568000,
  // Provided by VK_MESA_image_alignment_control
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_IMAGE_ALIGNMENT_CONTROL_FEATURES_MESA =
1000575000,
  // Provided by VK_MESA_image_alignment_control
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_IMAGE_ALIGNMENT_CONTROL_PROPERTIES_MESA =
1000575001,
  // Provided by VK_MESA_image_alignment_control
    VK_STRUCTURE_TYPE_IMAGE_ALIGNMENT_CONTROL_CREATE_INFO_MESA = 1000575002,
  // Provided by VK_VERSION_1_1
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_VARIABLE_POINTER_FEATURES =
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_VARIABLE_POINTERS_FEATURES,
  // Provided by VK_VERSION_1_1
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_DRAW_PARAMETER_FEATURES =
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_DRAW_PARAMETERS_FEATURES,
  // Provided by VK_EXT_debug_report
  // VK_STRUCTURE_TYPE_DEBUG_REPORT_CREATE_INFO_EXT is a deprecated alias
    VK_STRUCTURE_TYPE_DEBUG_REPORT_CREATE_INFO_EXT =
VK_STRUCTURE_TYPE_DEBUG_REPORT_CALLBACK_CREATE_INFO_EXT,
  // Provided by VK_KHR_dynamic_rendering
    VK_STRUCTURE_TYPE_RENDERING_INFO_KHR = VK_STRUCTURE_TYPE_RENDERING_INFO,
  // Provided by VK_KHR_dynamic_rendering
    VK_STRUCTURE_TYPE_RENDERING_ATTACHMENT_INFO_KHR =
VK_STRUCTURE_TYPE_RENDERING_ATTACHMENT_INFO,
  // Provided by VK_KHR_dynamic_rendering
    VK_STRUCTURE_TYPE_PIPELINE_RENDERING_CREATE_INFO_KHR =
VK_STRUCTURE_TYPE_PIPELINE_RENDERING_CREATE_INFO,
  // Provided by VK_KHR_dynamic_rendering
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DYNAMIC_RENDERING_FEATURES_KHR =
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DYNAMIC_RENDERING_FEATURES,
  // Provided by VK_KHR_dynamic_rendering
    VK_STRUCTURE_TYPE_COMMAND_BUFFER_INHERITANCE_RENDERING_INFO_KHR =
VK_STRUCTURE_TYPE_COMMAND_BUFFER_INHERITANCE_RENDERING_INFO,
  // Provided by VK_KHR_dynamic_rendering with VK_NV_framebuffer_mixed_samples
    VK_STRUCTURE_TYPE_ATTACHMENT_SAMPLE_COUNT_INFO_NV =
VK_STRUCTURE_TYPE_ATTACHMENT_SAMPLE_COUNT_INFO_AMD,
  // Provided by VK_KHR_multiview
    VK_STRUCTURE_TYPE_RENDER_PASS_MULTIVIEW_CREATE_INFO_KHR =
VK_STRUCTURE_TYPE_RENDER_PASS_MULTIVIEW_CREATE_INFO,
  // Provided by VK_KHR_multiview
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MULTIVIEW_FEATURES_KHR =
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MULTIVIEW_FEATURES,
```

```
  // Provided by VK_KHR_multiview
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MULTIVIEW_PROPERTIES_KHR =
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MULTIVIEW_PROPERTIES,
  // Provided by VK_KHR_get_physical_device_properties2
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_FEATURES_2_KHR =
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_FEATURES_2,
  // Provided by VK_KHR_get_physical_device_properties2
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PROPERTIES_2_KHR =
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PROPERTIES_2,
  // Provided by VK_KHR_get_physical_device_properties2
    VK_STRUCTURE_TYPE_FORMAT_PROPERTIES_2_KHR = VK_STRUCTURE_TYPE_FORMAT_PROPERTIES_2,
  // Provided by VK_KHR_get_physical_device_properties2
    VK_STRUCTURE_TYPE_IMAGE_FORMAT_PROPERTIES_2_KHR =
VK_STRUCTURE_TYPE_IMAGE_FORMAT_PROPERTIES_2,
  // Provided by VK_KHR_get_physical_device_properties2
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_IMAGE_FORMAT_INFO_2_KHR =
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_IMAGE_FORMAT_INFO_2,
  // Provided by VK_KHR_get_physical_device_properties2
    VK_STRUCTURE_TYPE_QUEUE_FAMILY_PROPERTIES_2_KHR =
VK_STRUCTURE_TYPE_QUEUE_FAMILY_PROPERTIES_2,
  // Provided by VK_KHR_get_physical_device_properties2
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MEMORY_PROPERTIES_2_KHR =
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MEMORY_PROPERTIES_2,
  // Provided by VK_KHR_get_physical_device_properties2
    VK_STRUCTURE_TYPE_SPARSE_IMAGE_FORMAT_PROPERTIES_2_KHR =
VK_STRUCTURE_TYPE_SPARSE_IMAGE_FORMAT_PROPERTIES_2,
  // Provided by VK_KHR_get_physical_device_properties2
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SPARSE_IMAGE_FORMAT_INFO_2_KHR =
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SPARSE_IMAGE_FORMAT_INFO_2,
  // Provided by VK_KHR_device_group
    VK_STRUCTURE_TYPE_MEMORY_ALLOCATE_FLAGS_INFO_KHR =
VK_STRUCTURE_TYPE_MEMORY_ALLOCATE_FLAGS_INFO,
  // Provided by VK_KHR_device_group
    VK_STRUCTURE_TYPE_DEVICE_GROUP_RENDER_PASS_BEGIN_INFO_KHR =
VK_STRUCTURE_TYPE_DEVICE_GROUP_RENDER_PASS_BEGIN_INFO,
  // Provided by VK_KHR_device_group
    VK_STRUCTURE_TYPE_DEVICE_GROUP_COMMAND_BUFFER_BEGIN_INFO_KHR =
VK_STRUCTURE_TYPE_DEVICE_GROUP_COMMAND_BUFFER_BEGIN_INFO,
  // Provided by VK_KHR_device_group
    VK_STRUCTURE_TYPE_DEVICE_GROUP_SUBMIT_INFO_KHR =
VK_STRUCTURE_TYPE_DEVICE_GROUP_SUBMIT_INFO,
  // Provided by VK_KHR_device_group
    VK_STRUCTURE_TYPE_DEVICE_GROUP_BIND_SPARSE_INFO_KHR =
VK_STRUCTURE_TYPE_DEVICE_GROUP_BIND_SPARSE_INFO,
  // Provided by VK_KHR_bind_memory2 with VK_KHR_device_group
    VK_STRUCTURE_TYPE_BIND_BUFFER_MEMORY_DEVICE_GROUP_INFO_KHR =
VK_STRUCTURE_TYPE_BIND_BUFFER_MEMORY_DEVICE_GROUP_INFO,
  // Provided by VK_KHR_bind_memory2 with VK_KHR_device_group
    VK_STRUCTURE_TYPE_BIND_IMAGE_MEMORY_DEVICE_GROUP_INFO_KHR =
VK_STRUCTURE_TYPE_BIND_IMAGE_MEMORY_DEVICE_GROUP_INFO,
  // Provided by VK_EXT_texture_compression_astc_hdr
```

```
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_TEXTURE_COMPRESSION_ASTC_HDR_FEATURES_EXT =
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_TEXTURE_COMPRESSION_ASTC_HDR_FEATURES,
  // Provided by VK_KHR_device_group_creation
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_GROUP_PROPERTIES_KHR =
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_GROUP_PROPERTIES,
  // Provided by VK_KHR_device_group_creation
    VK_STRUCTURE_TYPE_DEVICE_GROUP_DEVICE_CREATE_INFO_KHR =
VK_STRUCTURE_TYPE_DEVICE_GROUP_DEVICE_CREATE_INFO,
  // Provided by VK_KHR_external_memory_capabilities
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTERNAL_IMAGE_FORMAT_INFO_KHR =
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTERNAL_IMAGE_FORMAT_INFO,
  // Provided by VK_KHR_external_memory_capabilities
    VK_STRUCTURE_TYPE_EXTERNAL_IMAGE_FORMAT_PROPERTIES_KHR =
VK_STRUCTURE_TYPE_EXTERNAL_IMAGE_FORMAT_PROPERTIES,
  // Provided by VK_KHR_external_memory_capabilities
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTERNAL_BUFFER_INFO_KHR =
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTERNAL_BUFFER_INFO,
  // Provided by VK_KHR_external_memory_capabilities
    VK_STRUCTURE_TYPE_EXTERNAL_BUFFER_PROPERTIES_KHR =
VK_STRUCTURE_TYPE_EXTERNAL_BUFFER_PROPERTIES,
  // Provided by VK_KHR_external_fence_capabilities,
VK_KHR_external_memory_capabilities, VK_KHR_external_semaphore_capabilities
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_ID_PROPERTIES_KHR =
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_ID_PROPERTIES,
  // Provided by VK_KHR_external_memory
    VK_STRUCTURE_TYPE_EXTERNAL_MEMORY_BUFFER_CREATE_INFO_KHR =
VK_STRUCTURE_TYPE_EXTERNAL_MEMORY_BUFFER_CREATE_INFO,
  // Provided by VK_KHR_external_memory
    VK_STRUCTURE_TYPE_EXTERNAL_MEMORY_IMAGE_CREATE_INFO_KHR =
VK_STRUCTURE_TYPE_EXTERNAL_MEMORY_IMAGE_CREATE_INFO,
  // Provided by VK_KHR_external_memory
    VK_STRUCTURE_TYPE_EXPORT_MEMORY_ALLOCATE_INFO_KHR =
VK_STRUCTURE_TYPE_EXPORT_MEMORY_ALLOCATE_INFO,
  // Provided by VK_KHR_external_semaphore_capabilities
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTERNAL_SEMAPHORE_INFO_KHR =
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTERNAL_SEMAPHORE_INFO,
  // Provided by VK_KHR_external_semaphore_capabilities
    VK_STRUCTURE_TYPE_EXTERNAL_SEMAPHORE_PROPERTIES_KHR =
VK_STRUCTURE_TYPE_EXTERNAL_SEMAPHORE_PROPERTIES,
  // Provided by VK_KHR_external_semaphore
    VK_STRUCTURE_TYPE_EXPORT_SEMAPHORE_CREATE_INFO_KHR =
VK_STRUCTURE_TYPE_EXPORT_SEMAPHORE_CREATE_INFO,
  // Provided by VK_KHR_shader_float16_int8
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_FLOAT16_INT8_FEATURES_KHR =
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_FLOAT16_INT8_FEATURES,
  // Provided by VK_KHR_shader_float16_int8
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_FLOAT16_INT8_FEATURES_KHR =
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_FLOAT16_INT8_FEATURES,
  // Provided by VK_KHR_16bit_storage
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_16BIT_STORAGE_FEATURES_KHR =
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_16BIT_STORAGE_FEATURES,
```

```
  // Provided by VK_KHR_descriptor_update_template
    VK_STRUCTURE_TYPE_DESCRIPTOR_UPDATE_TEMPLATE_CREATE_INFO_KHR =
VK_STRUCTURE_TYPE_DESCRIPTOR_UPDATE_TEMPLATE_CREATE_INFO,
  // Provided by VK_EXT_display_surface_counter
  // VK_STRUCTURE_TYPE_SURFACE_CAPABILITIES2_EXT is a deprecated alias
    VK_STRUCTURE_TYPE_SURFACE_CAPABILITIES2_EXT =
VK_STRUCTURE_TYPE_SURFACE_CAPABILITIES_2_EXT,
  // Provided by VK_KHR_imageless_framebuffer
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_IMAGELESS_FRAMEBUFFER_FEATURES_KHR =
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_IMAGELESS_FRAMEBUFFER_FEATURES,
  // Provided by VK_KHR_imageless_framebuffer
    VK_STRUCTURE_TYPE_FRAMEBUFFER_ATTACHMENTS_CREATE_INFO_KHR =
VK_STRUCTURE_TYPE_FRAMEBUFFER_ATTACHMENTS_CREATE_INFO,
  // Provided by VK_KHR_imageless_framebuffer
    VK_STRUCTURE_TYPE_FRAMEBUFFER_ATTACHMENT_IMAGE_INFO_KHR =
VK_STRUCTURE_TYPE_FRAMEBUFFER_ATTACHMENT_IMAGE_INFO,
  // Provided by VK_KHR_imageless_framebuffer
    VK_STRUCTURE_TYPE_RENDER_PASS_ATTACHMENT_BEGIN_INFO_KHR =
VK_STRUCTURE_TYPE_RENDER_PASS_ATTACHMENT_BEGIN_INFO,
  // Provided by VK_KHR_create_renderpass2
    VK_STRUCTURE_TYPE_ATTACHMENT_DESCRIPTION_2_KHR =
VK_STRUCTURE_TYPE_ATTACHMENT_DESCRIPTION_2,
  // Provided by VK_KHR_create_renderpass2
    VK_STRUCTURE_TYPE_ATTACHMENT_REFERENCE_2_KHR =
VK_STRUCTURE_TYPE_ATTACHMENT_REFERENCE_2,
  // Provided by VK_KHR_create_renderpass2
    VK_STRUCTURE_TYPE_SUBPASS_DESCRIPTION_2_KHR =
VK_STRUCTURE_TYPE_SUBPASS_DESCRIPTION_2,
  // Provided by VK_KHR_create_renderpass2
    VK_STRUCTURE_TYPE_SUBPASS_DEPENDENCY_2_KHR =
VK_STRUCTURE_TYPE_SUBPASS_DEPENDENCY_2,
  // Provided by VK_KHR_create_renderpass2
    VK_STRUCTURE_TYPE_RENDER_PASS_CREATE_INFO_2_KHR =
VK_STRUCTURE_TYPE_RENDER_PASS_CREATE_INFO_2,
  // Provided by VK_KHR_create_renderpass2
    VK_STRUCTURE_TYPE_SUBPASS_BEGIN_INFO_KHR = VK_STRUCTURE_TYPE_SUBPASS_BEGIN_INFO,
  // Provided by VK_KHR_create_renderpass2
    VK_STRUCTURE_TYPE_SUBPASS_END_INFO_KHR = VK_STRUCTURE_TYPE_SUBPASS_END_INFO,
  // Provided by VK_KHR_external_fence_capabilities
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTERNAL_FENCE_INFO_KHR =
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTERNAL_FENCE_INFO,
  // Provided by VK_KHR_external_fence_capabilities
    VK_STRUCTURE_TYPE_EXTERNAL_FENCE_PROPERTIES_KHR =
VK_STRUCTURE_TYPE_EXTERNAL_FENCE_PROPERTIES,
  // Provided by VK_KHR_external_fence
    VK_STRUCTURE_TYPE_EXPORT_FENCE_CREATE_INFO_KHR =
VK_STRUCTURE_TYPE_EXPORT_FENCE_CREATE_INFO,
  // Provided by VK_KHR_maintenance2
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_POINT_CLIPPING_PROPERTIES_KHR =
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_POINT_CLIPPING_PROPERTIES,
  // Provided by VK_KHR_maintenance2
```

```
    VK_STRUCTURE_TYPE_RENDER_PASS_INPUT_ATTACHMENT_ASPECT_CREATE_INFO_KHR =
VK_STRUCTURE_TYPE_RENDER_PASS_INPUT_ATTACHMENT_ASPECT_CREATE_INFO,
  // Provided by VK_KHR_maintenance2
    VK_STRUCTURE_TYPE_IMAGE_VIEW_USAGE_CREATE_INFO_KHR =
VK_STRUCTURE_TYPE_IMAGE_VIEW_USAGE_CREATE_INFO,
  // Provided by VK_KHR_maintenance2
    VK_STRUCTURE_TYPE_PIPELINE_TESSELLATION_DOMAIN_ORIGIN_STATE_CREATE_INFO_KHR =
VK_STRUCTURE_TYPE_PIPELINE_TESSELLATION_DOMAIN_ORIGIN_STATE_CREATE_INFO,
  // Provided by VK_KHR_variable_pointers
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_VARIABLE_POINTERS_FEATURES_KHR =
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_VARIABLE_POINTERS_FEATURES,
  // Provided by VK_KHR_variable_pointers
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_VARIABLE_POINTER_FEATURES_KHR =
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_VARIABLE_POINTERS_FEATURES_KHR,
  // Provided by VK_KHR_dedicated_allocation
    VK_STRUCTURE_TYPE_MEMORY_DEDICATED_REQUIREMENTS_KHR =
VK_STRUCTURE_TYPE_MEMORY_DEDICATED_REQUIREMENTS,
  // Provided by VK_KHR_dedicated_allocation
    VK_STRUCTURE_TYPE_MEMORY_DEDICATED_ALLOCATE_INFO_KHR =
VK_STRUCTURE_TYPE_MEMORY_DEDICATED_ALLOCATE_INFO,
  // Provided by VK_EXT_sampler_filter_minmax
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SAMPLER_FILTER_MINMAX_PROPERTIES_EXT =
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SAMPLER_FILTER_MINMAX_PROPERTIES,
  // Provided by VK_EXT_sampler_filter_minmax
    VK_STRUCTURE_TYPE_SAMPLER_REDUCTION_MODE_CREATE_INFO_EXT =
VK_STRUCTURE_TYPE_SAMPLER_REDUCTION_MODE_CREATE_INFO,
  // Provided by VK_EXT_inline_uniform_block
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_INLINE_UNIFORM_BLOCK_FEATURES_EXT =
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_INLINE_UNIFORM_BLOCK_FEATURES,
  // Provided by VK_EXT_inline_uniform_block
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_INLINE_UNIFORM_BLOCK_PROPERTIES_EXT =
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_INLINE_UNIFORM_BLOCK_PROPERTIES,
  // Provided by VK_EXT_inline_uniform_block
    VK_STRUCTURE_TYPE_WRITE_DESCRIPTOR_SET_INLINE_UNIFORM_BLOCK_EXT =
VK_STRUCTURE_TYPE_WRITE_DESCRIPTOR_SET_INLINE_UNIFORM_BLOCK,
  // Provided by VK_EXT_inline_uniform_block
    VK_STRUCTURE_TYPE_DESCRIPTOR_POOL_INLINE_UNIFORM_BLOCK_CREATE_INFO_EXT =
VK_STRUCTURE_TYPE_DESCRIPTOR_POOL_INLINE_UNIFORM_BLOCK_CREATE_INFO,
  // Provided by VK_KHR_get_memory_requirements2
    VK_STRUCTURE_TYPE_BUFFER_MEMORY_REQUIREMENTS_INFO_2_KHR =
VK_STRUCTURE_TYPE_BUFFER_MEMORY_REQUIREMENTS_INFO_2,
  // Provided by VK_KHR_get_memory_requirements2
    VK_STRUCTURE_TYPE_IMAGE_MEMORY_REQUIREMENTS_INFO_2_KHR =
VK_STRUCTURE_TYPE_IMAGE_MEMORY_REQUIREMENTS_INFO_2,
  // Provided by VK_KHR_get_memory_requirements2
    VK_STRUCTURE_TYPE_IMAGE_SPARSE_MEMORY_REQUIREMENTS_INFO_2_KHR =
VK_STRUCTURE_TYPE_IMAGE_SPARSE_MEMORY_REQUIREMENTS_INFO_2,
  // Provided by VK_KHR_get_memory_requirements2
    VK_STRUCTURE_TYPE_MEMORY_REQUIREMENTS_2_KHR =
VK_STRUCTURE_TYPE_MEMORY_REQUIREMENTS_2,
  // Provided by VK_KHR_get_memory_requirements2
```

```
    VK_STRUCTURE_TYPE_SPARSE_IMAGE_MEMORY_REQUIREMENTS_2_KHR =
VK_STRUCTURE_TYPE_SPARSE_IMAGE_MEMORY_REQUIREMENTS_2,
  // Provided by VK_KHR_image_format_list
    VK_STRUCTURE_TYPE_IMAGE_FORMAT_LIST_CREATE_INFO_KHR =
VK_STRUCTURE_TYPE_IMAGE_FORMAT_LIST_CREATE_INFO,
  // Provided by VK_KHR_sampler_ycbcr_conversion
    VK_STRUCTURE_TYPE_SAMPLER_YCBCR_CONVERSION_CREATE_INFO_KHR =
VK_STRUCTURE_TYPE_SAMPLER_YCBCR_CONVERSION_CREATE_INFO,
  // Provided by VK_KHR_sampler_ycbcr_conversion
    VK_STRUCTURE_TYPE_SAMPLER_YCBCR_CONVERSION_INFO_KHR =
VK_STRUCTURE_TYPE_SAMPLER_YCBCR_CONVERSION_INFO,
  // Provided by VK_KHR_sampler_ycbcr_conversion
    VK_STRUCTURE_TYPE_BIND_IMAGE_PLANE_MEMORY_INFO_KHR =
VK_STRUCTURE_TYPE_BIND_IMAGE_PLANE_MEMORY_INFO,
  // Provided by VK_KHR_sampler_ycbcr_conversion
    VK_STRUCTURE_TYPE_IMAGE_PLANE_MEMORY_REQUIREMENTS_INFO_KHR =
VK_STRUCTURE_TYPE_IMAGE_PLANE_MEMORY_REQUIREMENTS_INFO,
  // Provided by VK_KHR_sampler_ycbcr_conversion
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SAMPLER_YCBCR_CONVERSION_FEATURES_KHR =
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SAMPLER_YCBCR_CONVERSION_FEATURES,
  // Provided by VK_KHR_sampler_ycbcr_conversion
    VK_STRUCTURE_TYPE_SAMPLER_YCBCR_CONVERSION_IMAGE_FORMAT_PROPERTIES_KHR =
VK_STRUCTURE_TYPE_SAMPLER_YCBCR_CONVERSION_IMAGE_FORMAT_PROPERTIES,
  // Provided by VK_KHR_bind_memory2
    VK_STRUCTURE_TYPE_BIND_BUFFER_MEMORY_INFO_KHR =
VK_STRUCTURE_TYPE_BIND_BUFFER_MEMORY_INFO,
  // Provided by VK_KHR_bind_memory2
    VK_STRUCTURE_TYPE_BIND_IMAGE_MEMORY_INFO_KHR =
VK_STRUCTURE_TYPE_BIND_IMAGE_MEMORY_INFO,
  // Provided by VK_EXT_descriptor_indexing
    VK_STRUCTURE_TYPE_DESCRIPTOR_SET_LAYOUT_BINDING_FLAGS_CREATE_INFO_EXT =
VK_STRUCTURE_TYPE_DESCRIPTOR_SET_LAYOUT_BINDING_FLAGS_CREATE_INFO,
  // Provided by VK_EXT_descriptor_indexing
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DESCRIPTOR_INDEXING_FEATURES_EXT =
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DESCRIPTOR_INDEXING_FEATURES,
  // Provided by VK_EXT_descriptor_indexing
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DESCRIPTOR_INDEXING_PROPERTIES_EXT =
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DESCRIPTOR_INDEXING_PROPERTIES,
  // Provided by VK_EXT_descriptor_indexing
    VK_STRUCTURE_TYPE_DESCRIPTOR_SET_VARIABLE_DESCRIPTOR_COUNT_ALLOCATE_INFO_EXT =
VK_STRUCTURE_TYPE_DESCRIPTOR_SET_VARIABLE_DESCRIPTOR_COUNT_ALLOCATE_INFO,
  // Provided by VK_EXT_descriptor_indexing
    VK_STRUCTURE_TYPE_DESCRIPTOR_SET_VARIABLE_DESCRIPTOR_COUNT_LAYOUT_SUPPORT_EXT =
VK_STRUCTURE_TYPE_DESCRIPTOR_SET_VARIABLE_DESCRIPTOR_COUNT_LAYOUT_SUPPORT,
  // Provided by VK_KHR_maintenance3
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MAINTENANCE_3_PROPERTIES_KHR =
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MAINTENANCE_3_PROPERTIES,
  // Provided by VK_KHR_maintenance3
    VK_STRUCTURE_TYPE_DESCRIPTOR_SET_LAYOUT_SUPPORT_KHR =
VK_STRUCTURE_TYPE_DESCRIPTOR_SET_LAYOUT_SUPPORT,
  // Provided by VK_EXT_global_priority
```

```
    VK_STRUCTURE_TYPE_DEVICE_QUEUE_GLOBAL_PRIORITY_CREATE_INFO_EXT =
VK_STRUCTURE_TYPE_DEVICE_QUEUE_GLOBAL_PRIORITY_CREATE_INFO_KHR,
  // Provided by VK_KHR_shader_subgroup_extended_types
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_SUBGROUP_EXTENDED_TYPES_FEATURES_KHR =
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_SUBGROUP_EXTENDED_TYPES_FEATURES,
  // Provided by VK_KHR_8bit_storage
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_8BIT_STORAGE_FEATURES_KHR =
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_8BIT_STORAGE_FEATURES,
  // Provided by VK_KHR_shader_atomic_int64
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_ATOMIC_INT64_FEATURES_KHR =
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_ATOMIC_INT64_FEATURES,
  // Provided by VK_EXT_calibrated_timestamps
    VK_STRUCTURE_TYPE_CALIBRATED_TIMESTAMP_INFO_EXT =
VK_STRUCTURE_TYPE_CALIBRATED_TIMESTAMP_INFO_KHR,
  // Provided by VK_EXT_vertex_attribute_divisor
    VK_STRUCTURE_TYPE_PIPELINE_VERTEX_INPUT_DIVISOR_STATE_CREATE_INFO_EXT =
VK_STRUCTURE_TYPE_PIPELINE_VERTEX_INPUT_DIVISOR_STATE_CREATE_INFO_KHR,
  // Provided by VK_EXT_vertex_attribute_divisor
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_VERTEX_ATTRIBUTE_DIVISOR_FEATURES_EXT =
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_VERTEX_ATTRIBUTE_DIVISOR_FEATURES_KHR,
  // Provided by VK_EXT_pipeline_creation_feedback
    VK_STRUCTURE_TYPE_PIPELINE_CREATION_FEEDBACK_CREATE_INFO_EXT =
VK_STRUCTURE_TYPE_PIPELINE_CREATION_FEEDBACK_CREATE_INFO,
  // Provided by VK_KHR_driver_properties
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DRIVER_PROPERTIES_KHR =
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DRIVER_PROPERTIES,
  // Provided by VK_KHR_shader_float_controls
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_FLOAT_CONTROLS_PROPERTIES_KHR =
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_FLOAT_CONTROLS_PROPERTIES,
  // Provided by VK_KHR_depth_stencil_resolve
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DEPTH_STENCIL_RESOLVE_PROPERTIES_KHR =
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DEPTH_STENCIL_RESOLVE_PROPERTIES,
  // Provided by VK_KHR_depth_stencil_resolve
    VK_STRUCTURE_TYPE_SUBPASS_DESCRIPTION_DEPTH_STENCIL_RESOLVE_KHR =
VK_STRUCTURE_TYPE_SUBPASS_DESCRIPTION_DEPTH_STENCIL_RESOLVE,
  // Provided by VK_NV_compute_shader_derivatives
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_COMPUTE_SHADER_DERIVATIVES_FEATURES_NV =
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_COMPUTE_SHADER_DERIVATIVES_FEATURES_KHR,
  // Provided by VK_NV_fragment_shader_barycentric
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_FRAGMENT_SHADER_BARYCENTRIC_FEATURES_NV =
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_FRAGMENT_SHADER_BARYCENTRIC_FEATURES_KHR,
  // Provided by VK_KHR_timeline_semaphore
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_TIMELINE_SEMAPHORE_FEATURES_KHR =
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_TIMELINE_SEMAPHORE_FEATURES,
  // Provided by VK_KHR_timeline_semaphore
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_TIMELINE_SEMAPHORE_PROPERTIES_KHR =
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_TIMELINE_SEMAPHORE_PROPERTIES,
  // Provided by VK_KHR_timeline_semaphore
    VK_STRUCTURE_TYPE_SEMAPHORE_TYPE_CREATE_INFO_KHR =
VK_STRUCTURE_TYPE_SEMAPHORE_TYPE_CREATE_INFO,
  // Provided by VK_KHR_timeline_semaphore
```

```
    VK_STRUCTURE_TYPE_TIMELINE_SEMAPHORE_SUBMIT_INFO_KHR =
VK_STRUCTURE_TYPE_TIMELINE_SEMAPHORE_SUBMIT_INFO,
  // Provided by VK_KHR_timeline_semaphore
    VK_STRUCTURE_TYPE_SEMAPHORE_WAIT_INFO_KHR = VK_STRUCTURE_TYPE_SEMAPHORE_WAIT_INFO,
  // Provided by VK_KHR_timeline_semaphore
    VK_STRUCTURE_TYPE_SEMAPHORE_SIGNAL_INFO_KHR =
VK_STRUCTURE_TYPE_SEMAPHORE_SIGNAL_INFO,
  // Provided by VK_INTEL_performance_query
  // VK_STRUCTURE_TYPE_QUERY_POOL_CREATE_INFO_INTEL is a deprecated alias
    VK_STRUCTURE_TYPE_QUERY_POOL_CREATE_INFO_INTEL =
VK_STRUCTURE_TYPE_QUERY_POOL_PERFORMANCE_QUERY_CREATE_INFO_INTEL,
  // Provided by VK_KHR_vulkan_memory_model
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_VULKAN_MEMORY_MODEL_FEATURES_KHR =
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_VULKAN_MEMORY_MODEL_FEATURES,
  // Provided by VK_KHR_shader_terminate_invocation
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_TERMINATE_INVOCATION_FEATURES_KHR =
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_TERMINATE_INVOCATION_FEATURES,
  // Provided by VK_EXT_scalar_block_layout
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SCALAR_BLOCK_LAYOUT_FEATURES_EXT =
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SCALAR_BLOCK_LAYOUT_FEATURES,
  // Provided by VK_EXT_subgroup_size_control
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SUBGROUP_SIZE_CONTROL_PROPERTIES_EXT =
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SUBGROUP_SIZE_CONTROL_PROPERTIES,
  // Provided by VK_EXT_subgroup_size_control
    VK_STRUCTURE_TYPE_PIPELINE_SHADER_STAGE_REQUIRED_SUBGROUP_SIZE_CREATE_INFO_EXT =
VK_STRUCTURE_TYPE_PIPELINE_SHADER_STAGE_REQUIRED_SUBGROUP_SIZE_CREATE_INFO,
  // Provided by VK_EXT_subgroup_size_control
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SUBGROUP_SIZE_CONTROL_FEATURES_EXT =
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SUBGROUP_SIZE_CONTROL_FEATURES,
  // Provided by VK_KHR_separate_depth_stencil_layouts
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SEPARATE_DEPTH_STENCIL_LAYOUTS_FEATURES_KHR =
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SEPARATE_DEPTH_STENCIL_LAYOUTS_FEATURES,
  // Provided by VK_KHR_separate_depth_stencil_layouts
    VK_STRUCTURE_TYPE_ATTACHMENT_REFERENCE_STENCIL_LAYOUT_KHR =
VK_STRUCTURE_TYPE_ATTACHMENT_REFERENCE_STENCIL_LAYOUT,
  // Provided by VK_KHR_separate_depth_stencil_layouts
    VK_STRUCTURE_TYPE_ATTACHMENT_DESCRIPTION_STENCIL_LAYOUT_KHR =
VK_STRUCTURE_TYPE_ATTACHMENT_DESCRIPTION_STENCIL_LAYOUT,
  // Provided by VK_EXT_buffer_device_address
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_BUFFER_ADDRESS_FEATURES_EXT =
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_BUFFER_DEVICE_ADDRESS_FEATURES_EXT,
  // Provided by VK_EXT_buffer_device_address
    VK_STRUCTURE_TYPE_BUFFER_DEVICE_ADDRESS_INFO_EXT =
VK_STRUCTURE_TYPE_BUFFER_DEVICE_ADDRESS_INFO,
  // Provided by VK_EXT_tooling_info
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_TOOL_PROPERTIES_EXT =
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_TOOL_PROPERTIES,
  // Provided by VK_EXT_separate_stencil_usage
    VK_STRUCTURE_TYPE_IMAGE_STENCIL_USAGE_CREATE_INFO_EXT =
VK_STRUCTURE_TYPE_IMAGE_STENCIL_USAGE_CREATE_INFO,
  // Provided by VK_KHR_uniform_buffer_standard_layout
```

```
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_UNIFORM_BUFFER_STANDARD_LAYOUT_FEATURES_KHR =
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_UNIFORM_BUFFER_STANDARD_LAYOUT_FEATURES,
  // Provided by VK_KHR_buffer_device_address
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_BUFFER_DEVICE_ADDRESS_FEATURES_KHR =
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_BUFFER_DEVICE_ADDRESS_FEATURES,
  // Provided by VK_KHR_buffer_device_address
    VK_STRUCTURE_TYPE_BUFFER_DEVICE_ADDRESS_INFO_KHR =
VK_STRUCTURE_TYPE_BUFFER_DEVICE_ADDRESS_INFO,
  // Provided by VK_KHR_buffer_device_address
    VK_STRUCTURE_TYPE_BUFFER_OPAQUE_CAPTURE_ADDRESS_CREATE_INFO_KHR =
VK_STRUCTURE_TYPE_BUFFER_OPAQUE_CAPTURE_ADDRESS_CREATE_INFO,
  // Provided by VK_KHR_buffer_device_address
    VK_STRUCTURE_TYPE_MEMORY_OPAQUE_CAPTURE_ADDRESS_ALLOCATE_INFO_KHR =
VK_STRUCTURE_TYPE_MEMORY_OPAQUE_CAPTURE_ADDRESS_ALLOCATE_INFO,
  // Provided by VK_KHR_buffer_device_address
    VK_STRUCTURE_TYPE_DEVICE_MEMORY_OPAQUE_CAPTURE_ADDRESS_INFO_KHR =
VK_STRUCTURE_TYPE_DEVICE_MEMORY_OPAQUE_CAPTURE_ADDRESS_INFO,
  // Provided by VK_EXT_line_rasterization
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_LINE_RASTERIZATION_FEATURES_EXT =
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_LINE_RASTERIZATION_FEATURES_KHR,
  // Provided by VK_EXT_line_rasterization
    VK_STRUCTURE_TYPE_PIPELINE_RASTERIZATION_LINE_STATE_CREATE_INFO_EXT =
VK_STRUCTURE_TYPE_PIPELINE_RASTERIZATION_LINE_STATE_CREATE_INFO_KHR,
  // Provided by VK_EXT_line_rasterization
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_LINE_RASTERIZATION_PROPERTIES_EXT =
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_LINE_RASTERIZATION_PROPERTIES_KHR,
  // Provided by VK_EXT_host_query_reset
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_HOST_QUERY_RESET_FEATURES_EXT =
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_HOST_QUERY_RESET_FEATURES,
  // Provided by VK_EXT_index_type_uint8
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_INDEX_TYPE_UINT8_FEATURES_EXT =
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_INDEX_TYPE_UINT8_FEATURES_KHR,
  // Provided by VK_EXT_shader_demote_to_helper_invocation
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_DEMOTE_TO_HELPER_INVOCATION_FEATURES_EXT
= VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_DEMOTE_TO_HELPER_INVOCATION_FEATURES,
  // Provided by VK_KHR_shader_integer_dot_product
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_INTEGER_DOT_PRODUCT_FEATURES_KHR =
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_INTEGER_DOT_PRODUCT_FEATURES,
  // Provided by VK_KHR_shader_integer_dot_product
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_INTEGER_DOT_PRODUCT_PROPERTIES_KHR =
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_INTEGER_DOT_PRODUCT_PROPERTIES,
  // Provided by VK_EXT_texel_buffer_alignment
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_TEXEL_BUFFER_ALIGNMENT_PROPERTIES_EXT =
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_TEXEL_BUFFER_ALIGNMENT_PROPERTIES,
  // Provided by VK_EXT_private_data
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PRIVATE_DATA_FEATURES_EXT =
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PRIVATE_DATA_FEATURES,
  // Provided by VK_EXT_private_data
    VK_STRUCTURE_TYPE_DEVICE_PRIVATE_DATA_CREATE_INFO_EXT =
VK_STRUCTURE_TYPE_DEVICE_PRIVATE_DATA_CREATE_INFO,
  // Provided by VK_EXT_private_data
```

```
    VK_STRUCTURE_TYPE_PRIVATE_DATA_SLOT_CREATE_INFO_EXT =
VK_STRUCTURE_TYPE_PRIVATE_DATA_SLOT_CREATE_INFO,
  // Provided by VK_EXT_pipeline_creation_cache_control
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PIPELINE_CREATION_CACHE_CONTROL_FEATURES_EXT =
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PIPELINE_CREATION_CACHE_CONTROL_FEATURES,
  // Provided by VK_KHR_synchronization2
    VK_STRUCTURE_TYPE_MEMORY_BARRIER_2_KHR = VK_STRUCTURE_TYPE_MEMORY_BARRIER_2,
  // Provided by VK_KHR_synchronization2
    VK_STRUCTURE_TYPE_BUFFER_MEMORY_BARRIER_2_KHR =
VK_STRUCTURE_TYPE_BUFFER_MEMORY_BARRIER_2,
  // Provided by VK_KHR_synchronization2
    VK_STRUCTURE_TYPE_IMAGE_MEMORY_BARRIER_2_KHR =
VK_STRUCTURE_TYPE_IMAGE_MEMORY_BARRIER_2,
  // Provided by VK_KHR_synchronization2
    VK_STRUCTURE_TYPE_DEPENDENCY_INFO_KHR = VK_STRUCTURE_TYPE_DEPENDENCY_INFO,
  // Provided by VK_KHR_synchronization2
    VK_STRUCTURE_TYPE_SUBMIT_INFO_2_KHR = VK_STRUCTURE_TYPE_SUBMIT_INFO_2,
  // Provided by VK_KHR_synchronization2
    VK_STRUCTURE_TYPE_SEMAPHORE_SUBMIT_INFO_KHR =
VK_STRUCTURE_TYPE_SEMAPHORE_SUBMIT_INFO,
  // Provided by VK_KHR_synchronization2
    VK_STRUCTURE_TYPE_COMMAND_BUFFER_SUBMIT_INFO_KHR =
VK_STRUCTURE_TYPE_COMMAND_BUFFER_SUBMIT_INFO,
  // Provided by VK_KHR_synchronization2
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SYNCHRONIZATION_2_FEATURES_KHR =
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SYNCHRONIZATION_2_FEATURES,
  // Provided by VK_KHR_zero_initialize_workgroup_memory
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_ZERO_INITIALIZE_WORKGROUP_MEMORY_FEATURES_KHR =
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_ZERO_INITIALIZE_WORKGROUP_MEMORY_FEATURES,
  // Provided by VK_EXT_image_robustness
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_IMAGE_ROBUSTNESS_FEATURES_EXT =
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_IMAGE_ROBUSTNESS_FEATURES,
  // Provided by VK_KHR_copy_commands2
    VK_STRUCTURE_TYPE_COPY_BUFFER_INFO_2_KHR = VK_STRUCTURE_TYPE_COPY_BUFFER_INFO_2,
  // Provided by VK_KHR_copy_commands2
    VK_STRUCTURE_TYPE_COPY_IMAGE_INFO_2_KHR = VK_STRUCTURE_TYPE_COPY_IMAGE_INFO_2,
  // Provided by VK_KHR_copy_commands2
    VK_STRUCTURE_TYPE_COPY_BUFFER_TO_IMAGE_INFO_2_KHR =
VK_STRUCTURE_TYPE_COPY_BUFFER_TO_IMAGE_INFO_2,
  // Provided by VK_KHR_copy_commands2
    VK_STRUCTURE_TYPE_COPY_IMAGE_TO_BUFFER_INFO_2_KHR =
VK_STRUCTURE_TYPE_COPY_IMAGE_TO_BUFFER_INFO_2,
  // Provided by VK_KHR_copy_commands2
    VK_STRUCTURE_TYPE_BLIT_IMAGE_INFO_2_KHR = VK_STRUCTURE_TYPE_BLIT_IMAGE_INFO_2,
  // Provided by VK_KHR_copy_commands2
    VK_STRUCTURE_TYPE_RESOLVE_IMAGE_INFO_2_KHR =
VK_STRUCTURE_TYPE_RESOLVE_IMAGE_INFO_2,
  // Provided by VK_KHR_copy_commands2
    VK_STRUCTURE_TYPE_BUFFER_COPY_2_KHR = VK_STRUCTURE_TYPE_BUFFER_COPY_2,
  // Provided by VK_KHR_copy_commands2
    VK_STRUCTURE_TYPE_IMAGE_COPY_2_KHR = VK_STRUCTURE_TYPE_IMAGE_COPY_2,
```

```
  // Provided by VK_KHR_copy_commands2
    VK_STRUCTURE_TYPE_IMAGE_BLIT_2_KHR = VK_STRUCTURE_TYPE_IMAGE_BLIT_2,
  // Provided by VK_KHR_copy_commands2
    VK_STRUCTURE_TYPE_BUFFER_IMAGE_COPY_2_KHR = VK_STRUCTURE_TYPE_BUFFER_IMAGE_COPY_2,
  // Provided by VK_KHR_copy_commands2
    VK_STRUCTURE_TYPE_IMAGE_RESOLVE_2_KHR = VK_STRUCTURE_TYPE_IMAGE_RESOLVE_2,
  // Provided by VK_EXT_image_compression_control
    VK_STRUCTURE_TYPE_SUBRESOURCE_LAYOUT_2_EXT =
VK_STRUCTURE_TYPE_SUBRESOURCE_LAYOUT_2_KHR,
  // Provided by VK_EXT_image_compression_control
    VK_STRUCTURE_TYPE_IMAGE_SUBRESOURCE_2_EXT =
VK_STRUCTURE_TYPE_IMAGE_SUBRESOURCE_2_KHR,
  // Provided by VK_ARM_rasterization_order_attachment_access

VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_RASTERIZATION_ORDER_ATTACHMENT_ACCESS_FEATURES_ARM =
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_RASTERIZATION_ORDER_ATTACHMENT_ACCESS_FEATURES_EXT,
  // Provided by VK_VALVE_mutable_descriptor_type
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MUTABLE_DESCRIPTOR_TYPE_FEATURES_VALVE =
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MUTABLE_DESCRIPTOR_TYPE_FEATURES_EXT,
  // Provided by VK_VALVE_mutable_descriptor_type
    VK_STRUCTURE_TYPE_MUTABLE_DESCRIPTOR_TYPE_CREATE_INFO_VALVE =
VK_STRUCTURE_TYPE_MUTABLE_DESCRIPTOR_TYPE_CREATE_INFO_EXT,
  // Provided by VK_KHR_format_feature_flags2
    VK_STRUCTURE_TYPE_FORMAT_PROPERTIES_3_KHR = VK_STRUCTURE_TYPE_FORMAT_PROPERTIES_3,
  // Provided by VK_EXT_pipeline_properties
    VK_STRUCTURE_TYPE_PIPELINE_INFO_EXT = VK_STRUCTURE_TYPE_PIPELINE_INFO_KHR,
  // Provided by VK_EXT_global_priority_query
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_GLOBAL_PRIORITY_QUERY_FEATURES_EXT =
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_GLOBAL_PRIORITY_QUERY_FEATURES_KHR,
  // Provided by VK_EXT_global_priority_query
    VK_STRUCTURE_TYPE_QUEUE_FAMILY_GLOBAL_PRIORITY_PROPERTIES_EXT =
VK_STRUCTURE_TYPE_QUEUE_FAMILY_GLOBAL_PRIORITY_PROPERTIES_KHR,
  // Provided by VK_KHR_maintenance4
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MAINTENANCE_4_FEATURES_KHR =
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MAINTENANCE_4_FEATURES,
  // Provided by VK_KHR_maintenance4
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MAINTENANCE_4_PROPERTIES_KHR =
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MAINTENANCE_4_PROPERTIES,
  // Provided by VK_KHR_maintenance4
    VK_STRUCTURE_TYPE_DEVICE_BUFFER_MEMORY_REQUIREMENTS_KHR =
VK_STRUCTURE_TYPE_DEVICE_BUFFER_MEMORY_REQUIREMENTS,
  // Provided by VK_KHR_maintenance4
    VK_STRUCTURE_TYPE_DEVICE_IMAGE_MEMORY_REQUIREMENTS_KHR =
VK_STRUCTURE_TYPE_DEVICE_IMAGE_MEMORY_REQUIREMENTS,
  // Provided by VK_EXT_shader_object
    VK_STRUCTURE_TYPE_SHADER_REQUIRED_SUBGROUP_SIZE_CREATE_INFO_EXT =
VK_STRUCTURE_TYPE_PIPELINE_SHADER_STAGE_REQUIRED_SUBGROUP_SIZE_CREATE_INFO,
} VkStructureType;
```

## 3.13. API Name Aliases

A small number of APIs did not follow the naming conventions when initially defined. For consistency, when we discover an API name that violates the naming conventions, we rename it in the Specification, XML, and header files. For backwards compatibility, the original (incorrect) name is retained as a “typo alias”. The alias is deprecated and should not be used, but will be retained indefinitely.

**NOTE** `VK_STENCIL_FRONT_AND_BACK` is an example of a *typo alias*. It was initially defined as part of VkStencilFaceFlagBits. Once the naming inconsistency was noticed, it was renamed to `VK_STENCIL_FACE_FRONT_AND_BACK`, and the old name was aliased to the correct name.

# Chapter 4. Initialization

Before using Vulkan, an application **must** initialize it by loading the Vulkan commands, and creating a `VkInstance` object.

## 4.1. Command Function Pointers

Vulkan commands are not necessarily exposed by static linking on a platform. Commands to query function pointers for Vulkan commands are described below.

**NOTE** When extensions are promoted or otherwise incorporated into another extension or Vulkan core version, command aliases may be included. Whilst the behavior of each command alias is identical, the behavior of retrieving each alias's function pointer is not. A function pointer for a given alias can only be retrieved if the extension or version that introduced that alias is supported and enabled, irrespective of whether any other alias is available.

Function pointers for all Vulkan commands **can** be obtained by calling:

```
// Provided by VK_VERSION_1_0
PFN_vkVoidFunction vkGetInstanceProcAddr(
    VkInstance                                  instance,
    const char*                                 pName);
```

- `instance` is the instance that the function pointer will be compatible with, or `NULL` for commands not dependent on any instance.
- `pName` is the name of the command to obtain.

`vkGetInstanceProcAddr` itself is obtained in a platform- and loader- specific manner. Typically, the loader library will export this command as a function symbol, so applications **can** link against the loader library, or load it dynamically and look up the symbol using platform-specific APIs.

The table below defines the various use cases for `vkGetInstanceProcAddr` and expected return value ("fp" is "function pointer") for each case. A valid returned function pointer ("fp") **must** not be `NULL`.

The returned function pointer is of type PFN_vkVoidFunction, and **must** be cast to the type of the command being queried before use.

*Table 1.* `vkGetInstanceProcAddr` *behavior*

| `instance` | `pName` | **return value** |
|---|---|---|
| *[1] | `NULL` | undefined |
| invalid non-`NULL` instance | *[1] | undefined |
| `NULL` | *global command*[2] | fp |
| instance | vkGetInstanceProcAddr | fp |

| instance | pName | return value |
|---|---|---|
| instance | core *dispatchable command* | fp[3] |
| instance | enabled instance extension dispatchable command for instance | fp[3] |
| instance | available device extension[4] dispatchable command for instance | fp[3] |
| any other case, not covered above | | NULL |

**1**

"*" means any representable value for the parameter (including valid values, invalid values, and NULL).

**2**

The global commands are: vkEnumerateInstanceVersion, vkEnumerateInstanceExtensionProperties, vkEnumerateInstanceLayerProperties, and vkCreateInstance. Dispatchable commands are all other commands which are not global.

**3**

The returned function pointer **must** only be called with a dispatchable object (the first parameter) that is instance or a child of instance, e.g. VkInstance, VkPhysicalDevice, VkDevice, VkQueue, or VkCommandBuffer.

**4**

An “available device extension” is a device extension supported by any physical device enumerated by instance.

### Valid Usage (Implicit)

- VUID-vkGetInstanceProcAddr-instance-parameter
  If instance is not NULL, instance **must** be a valid VkInstance handle
- VUID-vkGetInstanceProcAddr-pName-parameter
  pName **must** be a null-terminated UTF-8 string

In order to support systems with multiple Vulkan implementations, the function pointers returned by vkGetInstanceProcAddr **may** point to dispatch code that calls a different real implementation for different VkDevice objects or their child objects. The overhead of the internal dispatch for VkDevice objects can be avoided by obtaining device-specific function pointers for any commands that use a device or device-child object as their dispatchable object. Such function pointers **can** be obtained by calling:

```
// Provided by VK_VERSION_1_0
PFN_vkVoidFunction vkGetDeviceProcAddr(
    VkDevice                                    device,
    const char*                                 pName);
```

The table below defines the various use cases for vkGetDeviceProcAddr and expected return value ("fp" is "function pointer") for each case. A valid returned function pointer ("fp") **must** not be NULL.

The returned function pointer is of type PFN_vkVoidFunction, and **must** be cast to the type of the command being queried before use. The function pointer **must** only be called with a dispatchable object (the first parameter) that is device or a child of device.

*Table 2.* vkGetDeviceProcAddr *behavior*

| device | pName | **return value** |
|---|---|---|
| NULL | *[1] | undefined |
| invalid device | *[1] | undefined |
| device | NULL | undefined |
| device | requested core version[2] device-level dispatchable command[3] | fp[4] |
| device | enabled extension device-level dispatchable command[3] | fp[4] |
| any other case, not covered above | | NULL |

**1**

"*" means any representable value for the parameter (including valid values, invalid values, and NULL).

**2**

Device-level commands which are part of the core version specified by VkApplicationInfo::apiVersion when creating the instance will always return a valid function pointer. If the maintenance5 feature is enabled, core commands beyond that version which are supported by the implementation will return NULL, otherwise the implementation **may** either return NULL or a function pointer. If a function pointer is returned, it **must** not be called.

**3**

In this function, device-level excludes all physical-device-level commands.

**4**

The returned function pointer **must** only be called with a dispatchable object (the first parameter) that is device or a child of device e.g. VkDevice, VkQueue, or VkCommandBuffer.

**Valid Usage (Implicit)**

- VUID-vkGetDeviceProcAddr-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkGetDeviceProcAddr-pName-parameter
  `pName` **must** be a null-terminated UTF-8 string

The definition of PFN_vkVoidFunction is:

```
// Provided by VK_VERSION_1_0
typedef void (VKAPI_PTR *PFN_vkVoidFunction)(void);
```

This type is returned from command function pointer queries, and **must** be cast to an actual command function pointer before use.

### 4.1.1. Extending Physical Device Core Functionality

New core physical-device-level functionality **can** be used when both VkPhysicalDeviceProperties::`apiVersion` and VkApplicationInfo::`apiVersion` are greater than or equal to the version of Vulkan that added the new functionality. The Vulkan version supported by a physical device **can** be obtained by calling vkGetPhysicalDeviceProperties.

### 4.1.2. Extending Physical Device From Device Extensions

When the `VK_KHR_get_physical_device_properties2` extension is enabled, or when both the instance and the physical-device versions are at least 1.1, physical-device-level functionality of a device extension **can** be used with a physical device if the corresponding extension is enumerated by vkEnumerateDeviceExtensionProperties for that physical device, even before a logical device has been created.

To obtain a function pointer for a physical-device-level command from a device extension, an application **can** use vkGetInstanceProcAddr. This function pointer **may** point to dispatch code, which calls a different real implementation for different `VkPhysicalDevice` objects. Applications **must** not use a VkPhysicalDevice in any command added by an extension or core version that is not supported by that physical device.

Device extensions **may** define structures that **can** be added to the `pNext` chain of physical-device-level commands.

## 4.2. Instances

There is no global state in Vulkan and all per-application state is stored in a `VkInstance` object. Creating a `VkInstance` object initializes the Vulkan library and allows the application to pass information about itself to the implementation.

Instances are represented by `VkInstance` handles:

```
// Provided by VK_VERSION_1_0
VK_DEFINE_HANDLE(VkInstance)
```

To query the version of instance-level functionality supported by the implementation, call:

```
// Provided by VK_VERSION_1_1
VkResult vkEnumerateInstanceVersion(
    uint32_t*                                   pApiVersion);
```

- `pApiVersion` is a pointer to a `uint32_t`, which is the version of Vulkan supported by instance-level functionality, encoded as described in Version Numbers.

| | |
|---|---|
| **NOTE** | The intended behavior of vkEnumerateInstanceVersion is that an implementation **should** not need to perform memory allocations and **should** unconditionally return `VK_SUCCESS`. The loader, and any enabled layers, **may** return `VK_ERROR_OUT_OF_HOST_MEMORY` in the case of a failed memory allocation. |

### Valid Usage (Implicit)

- VUID-vkEnumerateInstanceVersion-pApiVersion-parameter
  `pApiVersion` **must** be a valid pointer to a `uint32_t` value

### Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- `VK_ERROR_OUT_OF_HOST_MEMORY`

To create an instance object, call:

```
// Provided by VK_VERSION_1_0
VkResult vkCreateInstance(
    const VkInstanceCreateInfo*                 pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkInstance*                                 pInstance);
```

- `pCreateInfo` is a pointer to a VkInstanceCreateInfo structure controlling creation of the instance.
- `pAllocator` controls host memory allocation as described in the Memory Allocation chapter.
- `pInstance` points a VkInstance handle in which the resulting instance is returned.

`vkCreateInstance` verifies that the requested layers exist. If not, `vkCreateInstance` will return

VK_ERROR_LAYER_NOT_PRESENT. Next vkCreateInstance verifies that the requested extensions are supported (e.g. in the implementation or in any enabled instance layer) and if any requested extension is not supported, vkCreateInstance **must** return VK_ERROR_EXTENSION_NOT_PRESENT. After verifying and enabling the instance layers and extensions the VkInstance object is created and returned to the application. If a requested extension is only supported by a layer, both the layer and the extension need to be specified at vkCreateInstance time for the creation to succeed.

### Valid Usage

- VUID-vkCreateInstance-ppEnabledExtensionNames-01388
  All required extensions for each extension in the VkInstanceCreateInfo ::ppEnabledExtensionNames list **must** also be present in that list

### Valid Usage (Implicit)

- VUID-vkCreateInstance-pCreateInfo-parameter
  pCreateInfo **must** be a valid pointer to a valid VkInstanceCreateInfo structure
- VUID-vkCreateInstance-pAllocator-parameter
  If pAllocator is not NULL, pAllocator **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkCreateInstance-pInstance-parameter
  pInstance **must** be a valid pointer to a VkInstance handle

### Return Codes

**Success**

- VK_SUCCESS

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY
- VK_ERROR_INITIALIZATION_FAILED
- VK_ERROR_LAYER_NOT_PRESENT
- VK_ERROR_EXTENSION_NOT_PRESENT
- VK_ERROR_INCOMPATIBLE_DRIVER

The VkInstanceCreateInfo structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkInstanceCreateInfo {
    VkStructureType             sType;
    const void*                 pNext;
    VkInstanceCreateFlags       flags;
    const VkApplicationInfo*    pApplicationInfo;
    uint32_t                    enabledLayerCount;
    const char* const*          ppEnabledLayerNames;
    uint32_t                    enabledExtensionCount;
    const char* const*          ppEnabledExtensionNames;
} VkInstanceCreateInfo;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is a bitmask of VkInstanceCreateFlagBits indicating the behavior of the instance.
- `pApplicationInfo` is `NULL` or a pointer to a `VkApplicationInfo` structure. If not `NULL`, this information helps implementations recognize behavior inherent to classes of applications. VkApplicationInfo is defined in detail below.
- `enabledLayerCount` is the number of global layers to enable.
- `ppEnabledLayerNames` is a pointer to an array of `enabledLayerCount` null-terminated UTF-8 strings containing the names of layers to enable for the created instance. The layers are loaded in the order they are listed in this array, with the first array element being the closest to the application, and the last array element being the closest to the driver. See the Layers section for further details.
- `enabledExtensionCount` is the number of global extensions to enable.
- `ppEnabledExtensionNames` is a pointer to an array of `enabledExtensionCount` null-terminated UTF-8 strings containing the names of extensions to enable.

To capture events that occur while creating or destroying an instance, an application **can** link a VkDebugReportCallbackCreateInfoEXT structure or a VkDebugUtilsMessengerCreateInfoEXT structure to the `pNext` element of the `VkInstanceCreateInfo` structure given to `vkCreateInstance`. This callback is only valid for the duration of the vkCreateInstance and the vkDestroyInstance call. Use vkCreateDebugReportCallbackEXT or vkCreateDebugUtilsMessengerEXT to create persistent callback objects.

An application can add additional drivers by including the VkDirectDriverLoadingListLUNARG struct to the `pNext` element of the `VkInstanceCreateInfo` structure given to `vkCreateInstance`.

**NOTE** VkDirectDriverLoadingListLUNARG allows applications to ship drivers with themselves. Only drivers that are designed to work with it should be used, such as drivers that implement Vulkan in software or that implement Vulkan by translating it to a different API. Any driver that requires installation should not be used, such as hardware drivers.

## Valid Usage

- VUID-VkInstanceCreateInfo-pNext-04925
  If the `pNext` chain of `VkInstanceCreateInfo` includes a `VkDebugReportCallbackCreateInfoEXT` structure, the list of enabled extensions in `ppEnabledExtensionNames` **must** contain `VK_EXT_debug_report`
- VUID-VkInstanceCreateInfo-pNext-04926
  If the `pNext` chain of `VkInstanceCreateInfo` includes a `VkDebugUtilsMessengerCreateInfoEXT` structure, the list of enabled extensions in `ppEnabledExtensionNames` **must** contain `VK_EXT_debug_utils`
- VUID-VkInstanceCreateInfo-pNext-06779
  If the `pNext` chain includes a VkExportMetalObjectCreateInfoEXT structure, its `exportObjectType` member **must** be either `VK_EXPORT_METAL_OBJECT_TYPE_METAL_DEVICE_BIT_EXT` or `VK_EXPORT_METAL_OBJECT_TYPE_METAL_COMMAND_QUEUE_BIT_EXT`
- VUID-VkInstanceCreateInfo-flags-06559
  If `flags` has the `VK_INSTANCE_CREATE_ENUMERATE_PORTABILITY_BIT_KHR` bit set, the list of enabled extensions in `ppEnabledExtensionNames` **must** contain `VK_KHR_portability_enumeration`
- VUID-VkInstanceCreateInfo-pNext-09400
  If the `pNext` chain of `VkInstanceCreateInfo` includes a VkDirectDriverLoadingListLUNARG structure, the list of enabled extensions in `ppEnabledExtensionNames` **must** contain VK_LUNARG_direct_driver_loading

## Valid Usage (Implicit)

- VUID-VkInstanceCreateInfo-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_INSTANCE_CREATE_INFO`
- VUID-VkInstanceCreateInfo-pNext-pNext
  Each `pNext` member of any structure (including this one) in the `pNext` chain **must** be either `NULL` or a pointer to a valid instance of VkDebugReportCallbackCreateInfoEXT, VkDebugUtilsMessengerCreateInfoEXT, VkDirectDriverLoadingListLUNARG, VkExportMetalObjectCreateInfoEXT, VkLayerSettingsCreateInfoEXT, VkValidationFeaturesEXT, or VkValidationFlagsEXT
- VUID-VkInstanceCreateInfo-sType-unique
  The `sType` value of each struct in the `pNext` chain **must** be unique, with the exception of structures of type VkDebugUtilsMessengerCreateInfoEXT, VkExportMetalObjectCreateInfoEXT, or VkLayerSettingsCreateInfoEXT
- VUID-VkInstanceCreateInfo-flags-parameter
  `flags` **must** be a valid combination of VkInstanceCreateFlagBits values
- VUID-VkInstanceCreateInfo-pApplicationInfo-parameter
  If `pApplicationInfo` is not `NULL`, `pApplicationInfo` **must** be a valid pointer to a valid VkApplicationInfo structure

- VUID-VkInstanceCreateInfo-ppEnabledLayerNames-parameter
  If `enabledLayerCount` is not `0`, `ppEnabledLayerNames` **must** be a valid pointer to an array of `enabledLayerCount` null-terminated UTF-8 strings
- VUID-VkInstanceCreateInfo-ppEnabledExtensionNames-parameter
  If `enabledExtensionCount` is not `0`, `ppEnabledExtensionNames` **must** be a valid pointer to an array of `enabledExtensionCount` null-terminated UTF-8 strings

```
// Provided by VK_VERSION_1_0
typedef enum VkInstanceCreateFlagBits {
  // Provided by VK_KHR_portability_enumeration
    VK_INSTANCE_CREATE_ENUMERATE_PORTABILITY_BIT_KHR = 0x00000001,
} VkInstanceCreateFlagBits;
```

- `VK_INSTANCE_CREATE_ENUMERATE_PORTABILITY_BIT_KHR` specifies that the instance will enumerate available Vulkan Portability-compliant physical devices and groups in addition to the Vulkan physical devices and groups that are enumerated by default.

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkInstanceCreateFlags;
```

`VkInstanceCreateFlags` is a bitmask type for setting a mask of zero or more VkInstanceCreateFlagBits.

When creating a Vulkan instance for which you wish to disable validation checks, add a VkValidationFlagsEXT structure to the `pNext` chain of the VkInstanceCreateInfo structure, specifying the checks to be disabled.

```
// Provided by VK_EXT_validation_flags
typedef struct VkValidationFlagsEXT {
    VkStructureType                sType;
    const void*                    pNext;
    uint32_t                       disabledValidationCheckCount;
    const VkValidationCheckEXT*    pDisabledValidationChecks;
} VkValidationFlagsEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `disabledValidationCheckCount` is the number of checks to disable.
- `pDisabledValidationChecks` is a pointer to an array of VkValidationCheckEXT values specifying the validation checks to be disabled.

### Valid Usage (Implicit)

- VUID-VkValidationFlagsEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_VALIDATION_FLAGS_EXT`
- VUID-VkValidationFlagsEXT-pDisabledValidationChecks-parameter
  `pDisabledValidationChecks` **must** be a valid pointer to an array of `disabledValidationCheckCount` valid VkValidationCheckEXT values
- VUID-VkValidationFlagsEXT-disabledValidationCheckCount-arraylength
  `disabledValidationCheckCount` **must** be greater than `0`

Possible values of elements of the VkValidationFlagsEXT::`pDisabledValidationChecks` array, specifying validation checks to be disabled, are:

```
// Provided by VK_EXT_validation_flags
typedef enum VkValidationCheckEXT {
    VK_VALIDATION_CHECK_ALL_EXT = 0,
    VK_VALIDATION_CHECK_SHADERS_EXT = 1,
} VkValidationCheckEXT;
```

- `VK_VALIDATION_CHECK_ALL_EXT` specifies that all validation checks are disabled.
- `VK_VALIDATION_CHECK_SHADERS_EXT` specifies that shader validation is disabled.

When creating a Vulkan instance for which you wish to enable or disable specific validation features, add a VkValidationFeaturesEXT structure to the `pNext` chain of the VkInstanceCreateInfo structure, specifying the features to be enabled or disabled.

```
// Provided by VK_EXT_validation_features
typedef struct VkValidationFeaturesEXT {
    VkStructureType                         sType;
    const void*                             pNext;
    uint32_t                                enabledValidationFeatureCount;
    const VkValidationFeatureEnableEXT*     pEnabledValidationFeatures;
    uint32_t                                disabledValidationFeatureCount;
    const VkValidationFeatureDisableEXT*    pDisabledValidationFeatures;
} VkValidationFeaturesEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `enabledValidationFeatureCount` is the number of features to enable.
- `pEnabledValidationFeatures` is a pointer to an array of VkValidationFeatureEnableEXT values specifying the validation features to be enabled.
- `disabledValidationFeatureCount` is the number of features to disable.
- `pDisabledValidationFeatures` is a pointer to an array of VkValidationFeatureDisableEXT values

specifying the validation features to be disabled.

### Valid Usage

- VUID-VkValidationFeaturesEXT-pEnabledValidationFeatures-02967
  If the `pEnabledValidationFeatures` array contains `VK_VALIDATION_FEATURE_ENABLE_GPU_ASSISTED_RESERVE_BINDING_SLOT_EXT`, then it **must** also contain `VK_VALIDATION_FEATURE_ENABLE_GPU_ASSISTED_EXT` or `VK_VALIDATION_FEATURE_ENABLE_DEBUG_PRINTF_EXT`
- VUID-VkValidationFeaturesEXT-pEnabledValidationFeatures-02968
  If the `pEnabledValidationFeatures` array contains `VK_VALIDATION_FEATURE_ENABLE_DEBUG_PRINTF_EXT`, then it **must** not contain `VK_VALIDATION_FEATURE_ENABLE_GPU_ASSISTED_EXT`

### Valid Usage (Implicit)

- VUID-VkValidationFeaturesEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_VALIDATION_FEATURES_EXT`
- VUID-VkValidationFeaturesEXT-pEnabledValidationFeatures-parameter
  If `enabledValidationFeatureCount` is not `0`, `pEnabledValidationFeatures` **must** be a valid pointer to an array of `enabledValidationFeatureCount` valid VkValidationFeatureEnableEXT values
- VUID-VkValidationFeaturesEXT-pDisabledValidationFeatures-parameter
  If `disabledValidationFeatureCount` is not `0`, `pDisabledValidationFeatures` **must** be a valid pointer to an array of `disabledValidationFeatureCount` valid VkValidationFeatureDisableEXT values

Possible values of elements of the VkValidationFeaturesEXT::`pEnabledValidationFeatures` array, specifying validation features to be enabled, are:

```
// Provided by VK_EXT_validation_features
typedef enum VkValidationFeatureEnableEXT {
    VK_VALIDATION_FEATURE_ENABLE_GPU_ASSISTED_EXT = 0,
    VK_VALIDATION_FEATURE_ENABLE_GPU_ASSISTED_RESERVE_BINDING_SLOT_EXT = 1,
    VK_VALIDATION_FEATURE_ENABLE_BEST_PRACTICES_EXT = 2,
    VK_VALIDATION_FEATURE_ENABLE_DEBUG_PRINTF_EXT = 3,
    VK_VALIDATION_FEATURE_ENABLE_SYNCHRONIZATION_VALIDATION_EXT = 4,
} VkValidationFeatureEnableEXT;
```

- `VK_VALIDATION_FEATURE_ENABLE_GPU_ASSISTED_EXT` specifies that GPU-assisted validation is enabled. Activating this feature instruments shader programs to generate additional diagnostic data. This feature is disabled by default.
- `VK_VALIDATION_FEATURE_ENABLE_GPU_ASSISTED_RESERVE_BINDING_SLOT_EXT` specifies that the validation layers reserve a descriptor set binding slot for their own use. The layer reports a

value for VkPhysicalDeviceLimits::maxBoundDescriptorSets that is one less than the value reported by the device. If the device supports the binding of only one descriptor set, the validation layer does not perform GPU-assisted validation. This feature is disabled by default.

- VK_VALIDATION_FEATURE_ENABLE_BEST_PRACTICES_EXT specifies that Vulkan best-practices validation is enabled. Activating this feature enables the output of warnings related to common misuse of the API, but which are not explicitly prohibited by the specification. This feature is disabled by default.
- VK_VALIDATION_FEATURE_ENABLE_DEBUG_PRINTF_EXT specifies that the layers will process debugPrintfEXT operations in shaders and send the resulting output to the debug callback. This feature is disabled by default.
- VK_VALIDATION_FEATURE_ENABLE_SYNCHRONIZATION_VALIDATION_EXT specifies that Vulkan synchronization validation is enabled. This feature reports resource access conflicts due to missing or incorrect synchronization operations between actions (Draw, Copy, Dispatch, Blit) reading or writing the same regions of memory. This feature is disabled by default.

Possible values of elements of the VkValidationFeaturesEXT::pDisabledValidationFeatures array, specifying validation features to be disabled, are:

```
// Provided by VK_EXT_validation_features
typedef enum VkValidationFeatureDisableEXT {
    VK_VALIDATION_FEATURE_DISABLE_ALL_EXT = 0,
    VK_VALIDATION_FEATURE_DISABLE_SHADERS_EXT = 1,
    VK_VALIDATION_FEATURE_DISABLE_THREAD_SAFETY_EXT = 2,
    VK_VALIDATION_FEATURE_DISABLE_API_PARAMETERS_EXT = 3,
    VK_VALIDATION_FEATURE_DISABLE_OBJECT_LIFETIMES_EXT = 4,
    VK_VALIDATION_FEATURE_DISABLE_CORE_CHECKS_EXT = 5,
    VK_VALIDATION_FEATURE_DISABLE_UNIQUE_HANDLES_EXT = 6,
    VK_VALIDATION_FEATURE_DISABLE_SHADER_VALIDATION_CACHE_EXT = 7,
} VkValidationFeatureDisableEXT;
```

- VK_VALIDATION_FEATURE_DISABLE_ALL_EXT specifies that all validation checks are disabled.
- VK_VALIDATION_FEATURE_DISABLE_SHADERS_EXT specifies that shader validation, both runtime and standalone, is disabled. This validation occurs inside VkShaderCreateInfoEXT and VkShaderModuleCreateInfo. This feature is enabled by default.
- VK_VALIDATION_FEATURE_DISABLE_THREAD_SAFETY_EXT specifies that thread safety validation is disabled. This feature is enabled by default.
- VK_VALIDATION_FEATURE_DISABLE_API_PARAMETERS_EXT specifies that stateless parameter validation is disabled. This feature is enabled by default.
- VK_VALIDATION_FEATURE_DISABLE_OBJECT_LIFETIMES_EXT specifies that object lifetime validation is disabled. This feature is enabled by default.
- VK_VALIDATION_FEATURE_DISABLE_CORE_CHECKS_EXT specifies that core validation checks are disabled. This feature is enabled by default. If this feature is disabled, VK_VALIDATION_FEATURE_DISABLE_SHADERS_EXT is implied.
- VK_VALIDATION_FEATURE_DISABLE_UNIQUE_HANDLES_EXT specifies that protection against duplicate

non-dispatchable object handles is disabled. This feature is enabled by default.

- VK_VALIDATION_FEATURE_DISABLE_SHADER_VALIDATION_CACHE_EXT specifies that there will be no caching of shader validation results and every shader will be validated on every application execution. Shader validation caching is enabled by default.

| | |
|---|---|
| **NOTE** | Disabling checks such as parameter validation and object lifetime validation prevents the reporting of error conditions that can cause other validation checks to behave incorrectly or crash. Some validation checks assume that their inputs are already valid and do not always revalidate them. |

| | |
|---|---|
| **NOTE** | The VK_EXT_validation_features extension subsumes all the functionality provided in the VK_EXT_validation_flags extension. |

To create a Vulkan instance with a specific configuration of layer settings, add VkLayerSettingsCreateInfoEXT structures to the pNext chain of the VkInstanceCreateInfo structure, specifying the settings to be configured.

```
// Provided by VK_EXT_layer_settings
typedef struct VkLayerSettingsCreateInfoEXT {
    VkStructureType             sType;
    const void*                 pNext;
    uint32_t                    settingCount;
    const VkLayerSettingEXT*    pSettings;
} VkLayerSettingsCreateInfoEXT;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- settingCount is the number of settings to configure.
- pSettings is a pointer to an array of settingCount VkLayerSettingEXT values specifying the settings to be configured.

### Valid Usage (Implicit)

- VUID-VkLayerSettingsCreateInfoEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_LAYER_SETTINGS_CREATE_INFO_EXT
- VUID-VkLayerSettingsCreateInfoEXT-pSettings-parameter
  If settingCount is not 0, pSettings **must** be a valid pointer to an array of settingCount valid VkLayerSettingEXT structures

The values of elements of the VkLayerSettingsCreateInfoEXT::pSettings array, specifying layer settings to be configured, are:

```
// Provided by VK_EXT_layer_settings
typedef struct VkLayerSettingEXT {
    const char*                 pLayerName;
    const char*                 pSettingName;
    VkLayerSettingTypeEXT       type;
    uint32_t                    valueCount;
    const void*                 pValues;
} VkLayerSettingEXT;
```

- `pLayerName` is a pointer to a null-terminated UTF-8 string naming the layer to configure the setting from.
- `pSettingName` is a pointer to a null-terminated UTF-8 string naming the setting to configure. Values of `pSettingName` that are unknown to the layer are ignored.
- `type` is a VkLayerSettingTypeEXT value specifying the type of the `pValues` values.
- `valueCount` is the number of values used to configure the layer setting.
- `pValues` is a pointer to an array of `valueCount` values of the type indicated by `type` to configure the layer setting.

When multiple VkLayerSettingsCreateInfoEXT structures are chained and the same `pSettingName` is referenced for the same `pLayerName`, the value of the first reference of the layer setting is used.

### Valid Usage

- VUID-VkLayerSettingEXT-valueCount-10070
  If `valueCount` is not `0`, `pValues` **must** be a valid pointer to an array of `valueCount` values of the type indicated by `type`

### Valid Usage (Implicit)

- VUID-VkLayerSettingEXT-pLayerName-parameter
  `pLayerName` **must** be a null-terminated UTF-8 string
- VUID-VkLayerSettingEXT-pSettingName-parameter
  `pSettingName` **must** be a null-terminated UTF-8 string
- VUID-VkLayerSettingEXT-type-parameter
  `type` **must** be a valid VkLayerSettingTypeEXT value

Possible values of VkLayerSettingEXT::`type`, specifying the type of the data returned in VkLayerSettingEXT::`pValues`, are:

```
// Provided by VK_EXT_layer_settings
typedef enum VkLayerSettingTypeEXT {
    VK_LAYER_SETTING_TYPE_BOOL32_EXT = 0,
    VK_LAYER_SETTING_TYPE_INT32_EXT = 1,
    VK_LAYER_SETTING_TYPE_INT64_EXT = 2,
    VK_LAYER_SETTING_TYPE_UINT32_EXT = 3,
    VK_LAYER_SETTING_TYPE_UINT64_EXT = 4,
    VK_LAYER_SETTING_TYPE_FLOAT32_EXT = 5,
    VK_LAYER_SETTING_TYPE_FLOAT64_EXT = 6,
    VK_LAYER_SETTING_TYPE_STRING_EXT = 7,
} VkLayerSettingTypeEXT;
```

- `VK_LAYER_SETTING_TYPE_BOOL32_EXT` specifies that the layer setting's type is VkBool32.
- `VK_LAYER_SETTING_TYPE_INT32_EXT` specifies that the layer setting's type is signed 32-bit integer.
- `VK_LAYER_SETTING_TYPE_INT64_EXT` specifies that the layer setting's type is signed 64-bit integer.
- `VK_LAYER_SETTING_TYPE_UINT32_EXT` specifies that the layer setting's type is unsigned 32-bit integer.
- `VK_LAYER_SETTING_TYPE_UINT64_EXT` specifies that the layer setting's type is unsigned 64-bit integer.
- `VK_LAYER_SETTING_TYPE_FLOAT32_EXT` specifies that the layer setting's type is 32-bit floating-point.
- `VK_LAYER_SETTING_TYPE_FLOAT64_EXT` specifies that the layer setting's type is 64-bit floating-point.
- `VK_LAYER_SETTING_TYPE_STRING_EXT` specifies that the layer setting's type is a pointer to a null-terminated UTF-8 string.

The `VkDirectDriverLoadingListLUNARG` structure is defined as:

```
// Provided by VK_LUNARG_direct_driver_loading
typedef struct VkDirectDriverLoadingListLUNARG {
    VkStructureType                             sType;
    const void*                                 pNext;
    VkDirectDriverLoadingModeLUNARG             mode;
    uint32_t                                    driverCount;
    const VkDirectDriverLoadingInfoLUNARG*      pDrivers;
} VkDirectDriverLoadingListLUNARG;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `mode` controls the mode in which to load the provided drivers.
- `driverCount` is the number of driver manifest paths.
- `pDrivers` is a pointer to an array of `driverCount` VkDirectDriverLoadingInfoLUNARG structures.

When creating a Vulkan instance for which additional drivers are to be included, add a `VkDirectDriverLoadingListLUNARG` structure to the pNext chain of the VkInstanceCreateInfo

structure, and include in it the list of VkDirectDriverLoadingInfoLUNARG structures which contain the information necessary to load additional drivers.

### Valid Usage (Implicit)

- VUID-VkDirectDriverLoadingListLUNARG-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_DIRECT_DRIVER_LOADING_LIST_LUNARG
- VUID-VkDirectDriverLoadingListLUNARG-mode-parameter
  mode **must** be a valid VkDirectDriverLoadingModeLUNARG value
- VUID-VkDirectDriverLoadingListLUNARG-pDrivers-parameter
  pDrivers **must** be a valid pointer to an array of driverCount valid VkDirectDriverLoadingInfoLUNARG structures
- VUID-VkDirectDriverLoadingListLUNARG-driverCount-arraylength
  driverCount **must** be greater than 0

The VkDirectDriverLoadingInfoLUNARG structure is defined as:

```
// Provided by VK_LUNARG_direct_driver_loading
typedef struct VkDirectDriverLoadingInfoLUNARG {
    VkStructureType                     sType;
    void*                               pNext;
    VkDirectDriverLoadingFlagsLUNARG    flags;
    PFN_vkGetInstanceProcAddrLUNARG     pfnGetInstanceProcAddr;
} VkDirectDriverLoadingInfoLUNARG;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- flags is reserved for future use.
- pfnGetInstanceProcAddr is a PFN_vkGetInstanceProcAddrLUNARG pointer to the driver vkGetInstanceProcAddr function.

### Valid Usage (Implicit)

- VUID-VkDirectDriverLoadingInfoLUNARG-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_DIRECT_DRIVER_LOADING_INFO_LUNARG
- VUID-VkDirectDriverLoadingInfoLUNARG-flags-zerobitmask
  flags **must** be 0

Possible values of VkDirectDriverLoadingListLUNARG::mode, specifying the mode in which drivers are used, are:

```
// Provided by VK_LUNARG_direct_driver_loading
typedef enum VkDirectDriverLoadingModeLUNARG {
    VK_DIRECT_DRIVER_LOADING_MODE_EXCLUSIVE_LUNARG = 0,
    VK_DIRECT_DRIVER_LOADING_MODE_INCLUSIVE_LUNARG = 1,
} VkDirectDriverLoadingModeLUNARG;
```

- `VK_DIRECT_DRIVER_LOADING_MODE_EXCLUSIVE_LUNARG` specifies that the provided drivers are used instead of the system-loaded drivers.
- `VK_DIRECT_DRIVER_LOADING_MODE_INCLUSIVE_LUNARG` specifies that the provided drivers are used in addition to the system-loaded drivers.

```
// Provided by VK_LUNARG_direct_driver_loading
typedef VkFlags VkDirectDriverLoadingFlagsLUNARG;
```

`VkDirectDriverLoadingFlagsLUNARG` is a bitmask type for setting a mask, but is currently reserved for future use.

The type of PFN_vkGetInstanceProcAddrLUNARG is:

```
// Provided by VK_LUNARG_direct_driver_loading
typedef PFN_vkVoidFunction (VKAPI_PTR *PFN_vkGetInstanceProcAddrLUNARG)(
    VkInstance instance, const char* pName);
```

- `instance` is a VkInstance handle.
- `pName` is the name of a Vulkan command.

This type is compatible with the type of a pointer to the vkGetInstanceProcAddr command, but is used only to specify device driver addresses in VkDirectDriverLoadingInfoLUNARG ::`pfnGetInstanceProcAddr`.

| | |
|---|---|
| **NOTE** | This type exists only because of limitations in the XML schema and processing scripts, and its name may change in the future. Ideally we would use the `PFN_vkGetInstanceProcAddr` type generated in the `vulkan_core.h` header. |

The `VkApplicationInfo` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkApplicationInfo {
    VkStructureType    sType;
    const void*        pNext;
    const char*        pApplicationName;
    uint32_t           applicationVersion;
    const char*        pEngineName;
    uint32_t           engineVersion;
    uint32_t           apiVersion;
} VkApplicationInfo;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `pApplicationName` is `NULL` or is a pointer to a null-terminated UTF-8 string containing the name of the application.
- `applicationVersion` is an unsigned integer variable containing the developer-supplied version number of the application.
- `pEngineName` is `NULL` or is a pointer to a null-terminated UTF-8 string containing the name of the engine (if any) used to create the application.
- `engineVersion` is an unsigned integer variable containing the developer-supplied version number of the engine used to create the application.
- `apiVersion` **must** be the highest version of Vulkan that the application is designed to use, encoded as described in Version Numbers. The patch version number specified in `apiVersion` is ignored when creating an instance object. The variant version of the instance **must** match that requested in `apiVersion`.

Vulkan 1.0 implementations were required to return `VK_ERROR_INCOMPATIBLE_DRIVER` if `apiVersion` was larger than 1.0. Implementations that support Vulkan 1.1 or later **must** not return `VK_ERROR_INCOMPATIBLE_DRIVER` for any value of `apiVersion` .

**NOTE** Because Vulkan 1.0 implementations **may** fail with `VK_ERROR_INCOMPATIBLE_DRIVER`, applications **should** determine the version of Vulkan available before calling vkCreateInstance. If the vkGetInstanceProcAddr returns `NULL` for vkEnumerateInstanceVersion, it is a Vulkan 1.0 implementation. Otherwise, the application **can** call vkEnumerateInstanceVersion to determine the version of Vulkan.

As long as the instance supports at least Vulkan 1.1, an application **can** use different versions of Vulkan with an instance than it does with a device or physical device.

**NOTE** The Khronos validation layers will treat `apiVersion` as the highest API version the application targets, and will validate API usage against the minimum of that version and the implementation version (instance or device, depending on context). If an application tries to use functionality from a greater version than this, a validation error will be triggered.

For example, if the instance supports Vulkan 1.1 and three physical devices support Vulkan 1.0, Vulkan 1.1, and Vulkan 1.2, respectively, and if the application sets `apiVersion` to 1.2, the application **can** use the following versions of Vulkan:

- Vulkan 1.0 **can** be used with the instance and with all physical devices.
- Vulkan 1.1 **can** be used with the instance and with the physical devices that support Vulkan 1.1 and Vulkan 1.2.
- Vulkan 1.2 **can** be used with the physical device that supports Vulkan 1.2.

If we modify the above example so that the application sets `apiVersion` to 1.1, then the application **must** not use Vulkan 1.2 functionality on the physical device that supports Vulkan 1.2.

**NOTE** Providing a `NULL` VkInstanceCreateInfo::`pApplicationInfo` or providing an `apiVersion` of 0 is equivalent to providing an `apiVersion` of `VK_MAKE_API_VERSION(0,1,0,0)`.

### Valid Usage

- VUID-VkApplicationInfo-apiVersion-04010
  If `apiVersion` is not `0`, then it **must** be greater than or equal to VK_API_VERSION_1_0

### Valid Usage (Implicit)

- VUID-VkApplicationInfo-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_APPLICATION_INFO`
- VUID-VkApplicationInfo-pNext-pNext
  `pNext` **must** be `NULL`
- VUID-VkApplicationInfo-pApplicationName-parameter
  If `pApplicationName` is not `NULL`, `pApplicationName` **must** be a null-terminated UTF-8 string
- VUID-VkApplicationInfo-pEngineName-parameter
  If `pEngineName` is not `NULL`, `pEngineName` **must** be a null-terminated UTF-8 string

To destroy an instance, call:

```
// Provided by VK_VERSION_1_0
void vkDestroyInstance(
    VkInstance                                  instance,
    const VkAllocationCallbacks*                pAllocator);
```

- `instance` is the handle of the instance to destroy.
- `pAllocator` controls host memory allocation as described in the Memory Allocation chapter.

## Valid Usage

- VUID-vkDestroyInstance-instance-00629
  All child objects created using `instance` **must** have been destroyed prior to destroying `instance`
- VUID-vkDestroyInstance-instance-00630
  If `VkAllocationCallbacks` were provided when `instance` was created, a compatible set of callbacks **must** be provided here
- VUID-vkDestroyInstance-instance-00631
  If no `VkAllocationCallbacks` were provided when `instance` was created, `pAllocator` **must** be `NULL`

## Valid Usage (Implicit)

- VUID-vkDestroyInstance-instance-parameter
  If `instance` is not `NULL`, `instance` **must** be a valid VkInstance handle
- VUID-vkDestroyInstance-pAllocator-parameter
  If `pAllocator` is not `NULL`, `pAllocator` **must** be a valid pointer to a valid VkAllocationCallbacks structure

## Host Synchronization

- Host access to `instance` **must** be externally synchronized
- Host access to all `VkPhysicalDevice` objects enumerated from `instance` **must** be externally synchronized

# Chapter 5. Devices and Queues

Once Vulkan is initialized, devices and queues are the primary objects used to interact with a Vulkan implementation.

Vulkan separates the concept of *physical* and *logical* devices. A physical device usually represents a single complete implementation of Vulkan (excluding instance-level functionality) available to the host, of which there are a finite number. A logical device represents an instance of that implementation with its own state and resources independent of other logical devices.

Physical devices are represented by `VkPhysicalDevice` handles:

```
// Provided by VK_VERSION_1_0
VK_DEFINE_HANDLE(VkPhysicalDevice)
```

## 5.1. Physical Devices

To retrieve a list of physical device objects representing the physical devices installed in the system, call:

```
// Provided by VK_VERSION_1_0
VkResult vkEnumeratePhysicalDevices(
    VkInstance                                  instance,
    uint32_t*                                   pPhysicalDeviceCount,
    VkPhysicalDevice*                           pPhysicalDevices);
```

- `instance` is a handle to a Vulkan instance previously created with vkCreateInstance.
- `pPhysicalDeviceCount` is a pointer to an integer related to the number of physical devices available or queried, as described below.
- `pPhysicalDevices` is either `NULL` or a pointer to an array of `VkPhysicalDevice` handles.

If `pPhysicalDevices` is `NULL`, then the number of physical devices available is returned in `pPhysicalDeviceCount`. Otherwise, `pPhysicalDeviceCount` **must** point to a variable set by the application to the number of elements in the `pPhysicalDevices` array, and on return the variable is overwritten with the number of handles actually written to `pPhysicalDevices`. If `pPhysicalDeviceCount` is less than the number of physical devices available, at most `pPhysicalDeviceCount` structures will be written, and `VK_INCOMPLETE` will be returned instead of `VK_SUCCESS`, to indicate that not all the available physical devices were returned.

**Valid Usage (Implicit)**

- VUID-vkEnumeratePhysicalDevices-instance-parameter
  `instance` **must** be a valid VkInstance handle
- VUID-vkEnumeratePhysicalDevices-pPhysicalDeviceCount-parameter

pPhysicalDeviceCount **must** be a valid pointer to a uint32_t value

- VUID-vkEnumeratePhysicalDevices-pPhysicalDevices-parameter
  If the value referenced by pPhysicalDeviceCount is not 0, and pPhysicalDevices is not NULL, pPhysicalDevices **must** be a valid pointer to an array of pPhysicalDeviceCount VkPhysicalDevice handles

### Return Codes

**Success**

- VK_SUCCESS
- VK_INCOMPLETE

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY
- VK_ERROR_INITIALIZATION_FAILED

To query general properties of physical devices once enumerated, call:

```
// Provided by VK_VERSION_1_0
void vkGetPhysicalDeviceProperties(
    VkPhysicalDevice                            physicalDevice,
    VkPhysicalDeviceProperties*                 pProperties);
```

- physicalDevice is the handle to the physical device whose properties will be queried.
- pProperties is a pointer to a VkPhysicalDeviceProperties structure in which properties are returned.

### Valid Usage (Implicit)

- VUID-vkGetPhysicalDeviceProperties-physicalDevice-parameter
  physicalDevice **must** be a valid VkPhysicalDevice handle
- VUID-vkGetPhysicalDeviceProperties-pProperties-parameter
  pProperties **must** be a valid pointer to a VkPhysicalDeviceProperties structure

The VkPhysicalDeviceProperties structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkPhysicalDeviceProperties {
    uint32_t                            apiVersion;
    uint32_t                            driverVersion;
    uint32_t                            vendorID;
    uint32_t                            deviceID;
    VkPhysicalDeviceType                deviceType;
    char                                deviceName[VK_MAX_PHYSICAL_DEVICE_NAME_SIZE];
    uint8_t                             pipelineCacheUUID[VK_UUID_SIZE];
    VkPhysicalDeviceLimits              limits;
    VkPhysicalDeviceSparseProperties    sparseProperties;
} VkPhysicalDeviceProperties;
```

- `apiVersion` is the version of Vulkan supported by the device, encoded as described in Version Numbers.
- `driverVersion` is the vendor-specified version of the driver.
- `vendorID` is a unique identifier for the *vendor* (see below) of the physical device.
- `deviceID` is a unique identifier for the physical device among devices available from the vendor.
- `deviceType` is a VkPhysicalDeviceType specifying the type of device.
- `deviceName` is an array of `VK_MAX_PHYSICAL_DEVICE_NAME_SIZE` `char` containing a null-terminated UTF-8 string which is the name of the device.
- `pipelineCacheUUID` is an array of `VK_UUID_SIZE` `uint8_t` values representing a universally unique identifier for the device.
- `limits` is the VkPhysicalDeviceLimits structure specifying device-specific limits of the physical device. See Limits for details.
- `sparseProperties` is the VkPhysicalDeviceSparseProperties structure specifying various sparse related properties of the physical device. See Sparse Properties for details.

**NOTE** The value of `apiVersion` **may** be different than the version returned by vkEnumerateInstanceVersion; either higher or lower. In such cases, the application **must** not use functionality that exceeds the version of Vulkan associated with a given object. The `pApiVersion` parameter returned by vkEnumerateInstanceVersion is the version associated with a VkInstance and its children, except for a VkPhysicalDevice and its children. `VkPhysicalDeviceProperties::apiVersion` is the version associated with a VkPhysicalDevice and its children.

**NOTE** The encoding of `driverVersion` is implementation-defined. It **may** not use the same encoding as `apiVersion`. Applications should follow information from the *vendor* on how to extract the version information from `driverVersion`.

The `vendorID` and `deviceID` fields are provided to allow applications to adapt to device characteristics that are not adequately exposed by other Vulkan queries.

**NOTE** These **may** include performance profiles, hardware errata, or other characteristics.

The *vendor* identified by `vendorID` is the entity responsible for the most salient characteristics of the underlying implementation of the VkPhysicalDevice being queried.

**NOTE** For example, in the case of a discrete GPU implementation, this **should** be the GPU chipset vendor. In the case of a hardware accelerator integrated into a system-on-chip (SoC), this **should** be the supplier of the silicon IP used to create the accelerator.

If the vendor has a PCI vendor ID, the low 16 bits of `vendorID` **must** contain that PCI vendor ID, and the remaining bits **must** be set to zero. Otherwise, the value returned **must** be a valid Khronos vendor ID, obtained as described in the Vulkan Documentation and Extensions: Procedures and Conventions document in the section "Registering a Vendor ID with Khronos". Khronos vendor IDs are allocated starting at 0x10000, to distinguish them from the PCI vendor ID namespace. Khronos vendor IDs are symbolically defined in the VkVendorId type.

The vendor is also responsible for the value returned in `deviceID`. If the implementation is driven primarily by a PCI device with a PCI device ID, the low 16 bits of `deviceID` **must** contain that PCI device ID, and the remaining bits **must** be set to zero. Otherwise, the choice of what values to return **may** be dictated by operating system or platform policies - but **should** uniquely identify both the device version and any major configuration options (for example, core count in the case of multicore devices).

**NOTE** The same device ID **should** be used for all physical implementations of that device version and configuration. For example, all uses of a specific silicon IP GPU version and configuration **should** use the same device ID, even if those uses occur in different SoCs.

Khronos vendor IDs which **may** be returned in VkPhysicalDeviceProperties::`vendorID` are:

```
// Provided by VK_VERSION_1_0
typedef enum VkVendorId {
    VK_VENDOR_ID_KHRONOS = 0x10000,
    VK_VENDOR_ID_VIV = 0x10001,
    VK_VENDOR_ID_VSI = 0x10002,
    VK_VENDOR_ID_KAZAN = 0x10003,
    VK_VENDOR_ID_CODEPLAY = 0x10004,
    VK_VENDOR_ID_MESA = 0x10005,
    VK_VENDOR_ID_POCL = 0x10006,
    VK_VENDOR_ID_MOBILEYE = 0x10007,
} VkVendorId;
```

**NOTE** Khronos vendor IDs may be allocated by vendors at any time. Only the latest canonical versions of this Specification, of the corresponding `vk.xml` API Registry, and of the corresponding `vulkan_core.h` header file **must** contain all reserved Khronos vendor IDs.

Only Khronos vendor IDs are given symbolic names at present. PCI vendor IDs

> returned by the implementation can be looked up in the PCI-SIG database.

VK_MAX_PHYSICAL_DEVICE_NAME_SIZE is the length in char values of an array containing a physical device name string, as returned in VkPhysicalDeviceProperties::deviceName.

```
#define VK_MAX_PHYSICAL_DEVICE_NAME_SIZE  256U
```

The physical device types which **may** be returned in VkPhysicalDeviceProperties::deviceType are:

```
// Provided by VK_VERSION_1_0
typedef enum VkPhysicalDeviceType {
    VK_PHYSICAL_DEVICE_TYPE_OTHER = 0,
    VK_PHYSICAL_DEVICE_TYPE_INTEGRATED_GPU = 1,
    VK_PHYSICAL_DEVICE_TYPE_DISCRETE_GPU = 2,
    VK_PHYSICAL_DEVICE_TYPE_VIRTUAL_GPU = 3,
    VK_PHYSICAL_DEVICE_TYPE_CPU = 4,
} VkPhysicalDeviceType;
```

- VK_PHYSICAL_DEVICE_TYPE_OTHER - the device does not match any other available types.
- VK_PHYSICAL_DEVICE_TYPE_INTEGRATED_GPU - the device is typically one embedded in or tightly coupled with the host.
- VK_PHYSICAL_DEVICE_TYPE_DISCRETE_GPU - the device is typically a separate processor connected to the host via an interlink.
- VK_PHYSICAL_DEVICE_TYPE_VIRTUAL_GPU - the device is typically a virtual node in a virtualization environment.
- VK_PHYSICAL_DEVICE_TYPE_CPU - the device is typically running on the same processors as the host.

The physical device type is advertised for informational purposes only, and does not directly affect the operation of the system. However, the device type **may** correlate with other advertised properties or capabilities of the system, such as how many memory heaps there are.

To query general properties of physical devices once enumerated, call:

```
// Provided by VK_VERSION_1_1
void vkGetPhysicalDeviceProperties2(
    VkPhysicalDevice                            physicalDevice,
    VkPhysicalDeviceProperties2*                pProperties);
```

or the equivalent command

```
// Provided by VK_KHR_get_physical_device_properties2
void vkGetPhysicalDeviceProperties2KHR(
    VkPhysicalDevice                            physicalDevice,
    VkPhysicalDeviceProperties2*                pProperties);
```

- physicalDevice is the handle to the physical device whose properties will be queried.
- pProperties is a pointer to a VkPhysicalDeviceProperties2 structure in which properties are returned.

Each structure in pProperties and its pNext chain contains members corresponding to implementation-dependent properties, behaviors, or limits. vkGetPhysicalDeviceProperties2 fills in each member to specify the corresponding value for the implementation.

### Valid Usage (Implicit)

- VUID-vkGetPhysicalDeviceProperties2-physicalDevice-parameter
  physicalDevice **must** be a valid VkPhysicalDevice handle
- VUID-vkGetPhysicalDeviceProperties2-pProperties-parameter
  pProperties **must** be a valid pointer to a VkPhysicalDeviceProperties2 structure

The VkPhysicalDeviceProperties2 structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkPhysicalDeviceProperties2 {
    VkStructureType                 sType;
    void*                           pNext;
    VkPhysicalDeviceProperties      properties;
} VkPhysicalDeviceProperties2;
```

or the equivalent

```
// Provided by VK_KHR_get_physical_device_properties2
typedef VkPhysicalDeviceProperties2 VkPhysicalDeviceProperties2KHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- properties is a VkPhysicalDeviceProperties structure describing properties of the physical device. This structure is written with the same values as if it were written by vkGetPhysicalDeviceProperties.

The pNext chain of this structure is used to extend the structure with properties defined by extensions.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceProperties2-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PROPERTIES_2
- VUID-VkPhysicalDeviceProperties2-pNext-pNext
  Each pNext member of any structure (including this one) in the pNext chain **must** be either

NULL or a pointer to a valid instance of VkPhysicalDeviceAccelerationStructurePropertiesKHR, VkPhysicalDeviceBlendOperationAdvancedPropertiesEXT, VkPhysicalDeviceClusterCullingShaderPropertiesHUAWEI, VkPhysicalDeviceComputeShaderDerivativesPropertiesKHR, VkPhysicalDeviceConservativeRasterizationPropertiesEXT, VkPhysicalDeviceCooperativeMatrixPropertiesKHR, VkPhysicalDeviceCooperativeMatrixPropertiesNV, VkPhysicalDeviceCopyMemoryIndirectPropertiesNV, VkPhysicalDeviceCudaKernelLaunchPropertiesNV, VkPhysicalDeviceCustomBorderColorPropertiesEXT, VkPhysicalDeviceDepthStencilResolveProperties, VkPhysicalDeviceDescriptorBufferDensityMapPropertiesEXT, VkPhysicalDeviceDescriptorBufferPropertiesEXT, VkPhysicalDeviceDescriptorIndexingProperties, VkPhysicalDeviceDeviceGeneratedCommandsPropertiesNV, VkPhysicalDeviceDiscardRectanglePropertiesEXT, VkPhysicalDeviceDisplacementMicromapPropertiesNV, VkPhysicalDeviceDriverProperties, VkPhysicalDeviceDrmPropertiesEXT, VkPhysicalDeviceExtendedDynamicState3PropertiesEXT, VkPhysicalDeviceExtendedSparseAddressSpacePropertiesNV, VkPhysicalDeviceExternalFormatResolvePropertiesANDROID, VkPhysicalDeviceExternalMemoryHostPropertiesEXT, VkPhysicalDeviceFloatControlsProperties, VkPhysicalDeviceFragmentDensityMap2PropertiesEXT, VkPhysicalDeviceFragmentDensityMapOffsetPropertiesQCOM, VkPhysicalDeviceFragmentDensityMapPropertiesEXT, VkPhysicalDeviceFragmentShaderBarycentricPropertiesKHR, VkPhysicalDeviceFragmentShadingRateEnumsPropertiesNV, VkPhysicalDeviceFragmentShadingRatePropertiesKHR, VkPhysicalDeviceGraphicsPipelineLibraryPropertiesEXT, VkPhysicalDeviceHostImageCopyPropertiesEXT, VkPhysicalDeviceIDProperties, VkPhysicalDeviceImageAlignmentControlPropertiesMESA, VkPhysicalDeviceImageProcessing2PropertiesQCOM, VkPhysicalDeviceImageProcessingPropertiesQCOM, VkPhysicalDeviceInlineUniformBlockProperties, VkPhysicalDeviceLayeredApiPropertiesListKHR, VkPhysicalDeviceLayeredDriverPropertiesMSFT, VkPhysicalDeviceLegacyVertexAttributesPropertiesEXT, VkPhysicalDeviceLineRasterizationPropertiesKHR, VkPhysicalDeviceMaintenance3Properties, VkPhysicalDeviceMaintenance4Properties, VkPhysicalDeviceMaintenance5PropertiesKHR, VkPhysicalDeviceMaintenance6PropertiesKHR, VkPhysicalDeviceMaintenance7PropertiesKHR, VkPhysicalDeviceMapMemoryPlacedPropertiesEXT, VkPhysicalDeviceMemoryDecompressionPropertiesNV, VkPhysicalDeviceMeshShaderPropertiesEXT, VkPhysicalDeviceMeshShaderPropertiesNV, VkPhysicalDeviceMultiDrawPropertiesEXT,

VkPhysicalDeviceMultiviewPerViewAttributesPropertiesNVX, VkPhysicalDeviceMultiviewProperties, VkPhysicalDeviceNestedCommandBufferPropertiesEXT, VkPhysicalDeviceOpacityMicromapPropertiesEXT, VkPhysicalDeviceOpticalFlowPropertiesNV, VkPhysicalDevicePCIBusInfoPropertiesEXT, VkPhysicalDevicePerformanceQueryPropertiesKHR, VkPhysicalDevicePipelineBinaryPropertiesKHR, VkPhysicalDevicePipelineRobustnessPropertiesEXT, VkPhysicalDevicePointClippingProperties, VkPhysicalDevicePortabilitySubsetPropertiesKHR, VkPhysicalDeviceProtectedMemoryProperties, VkPhysicalDeviceProvokingVertexPropertiesEXT, VkPhysicalDevicePushDescriptorPropertiesKHR, VkPhysicalDeviceRayTracingInvocationReorderPropertiesNV, VkPhysicalDeviceRayTracingPipelinePropertiesKHR, VkPhysicalDeviceRayTracingPropertiesNV, VkPhysicalDeviceRenderPassStripedPropertiesARM, VkPhysicalDeviceRobustness2PropertiesEXT, VkPhysicalDeviceSampleLocationsPropertiesEXT, VkPhysicalDeviceSamplerFilterMinmaxProperties, VkPhysicalDeviceSchedulingControlsPropertiesARM, VkPhysicalDeviceShaderCoreBuiltinsPropertiesARM, VkPhysicalDeviceShaderCoreProperties2AMD, VkPhysicalDeviceShaderCorePropertiesAMD, VkPhysicalDeviceShaderCorePropertiesARM, VkPhysicalDeviceShaderEnqueuePropertiesAMDX, VkPhysicalDeviceShaderIntegerDotProductProperties, VkPhysicalDeviceShaderModuleIdentifierPropertiesEXT, VkPhysicalDeviceShaderObjectPropertiesEXT, VkPhysicalDeviceShaderSMBuiltinsPropertiesNV, VkPhysicalDeviceShaderTileImagePropertiesEXT, VkPhysicalDeviceShadingRateImagePropertiesNV, VkPhysicalDeviceSubgroupProperties, VkPhysicalDeviceSubgroupSizeControlProperties, VkPhysicalDeviceSubpassShadingPropertiesHUAWEI, VkPhysicalDeviceTexelBufferAlignmentProperties, VkPhysicalDeviceTimelineSemaphoreProperties, VkPhysicalDeviceTransformFeedbackPropertiesEXT, VkPhysicalDeviceVertexAttributeDivisorPropertiesEXT, or VkPhysicalDeviceVertexAttributeDivisorPropertiesKHR

- VUID-VkPhysicalDeviceProperties2-sType-unique
  The `sType` value of each struct in the `pNext` chain **must** be unique

The `VkPhysicalDeviceIDProperties` structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkPhysicalDeviceIDProperties {
    VkStructureType    sType;
    void*              pNext;
    uint8_t            deviceUUID[VK_UUID_SIZE];
    uint8_t            driverUUID[VK_UUID_SIZE];
    uint8_t            deviceLUID[VK_LUID_SIZE];
    uint32_t           deviceNodeMask;
    VkBool32           deviceLUIDValid;
} VkPhysicalDeviceIDProperties;
```

or the equivalent

```
// Provided by VK_KHR_external_fence_capabilities,
VK_KHR_external_memory_capabilities, VK_KHR_external_semaphore_capabilities
typedef VkPhysicalDeviceIDProperties VkPhysicalDeviceIDPropertiesKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `deviceUUID` is an array of `VK_UUID_SIZE` `uint8_t` values representing a universally unique identifier for the device.
- `driverUUID` is an array of `VK_UUID_SIZE` `uint8_t` values representing a universally unique identifier for the driver build in use by the device.
- `deviceLUID` is an array of `VK_LUID_SIZE` `uint8_t` values representing a locally unique identifier for the device.
- `deviceNodeMask` is a `uint32_t` bitfield identifying the node within a linked device adapter corresponding to the device.
- `deviceLUIDValid` is a boolean value that will be `VK_TRUE` if `deviceLUID` contains a valid LUID and `deviceNodeMask` contains a valid node mask, and `VK_FALSE` if they do not.

If the `VkPhysicalDeviceIDProperties` structure is included in the `pNext` chain of the VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with each corresponding implementation-dependent property.

`deviceUUID` **must** be immutable for a given device across instances, processes, driver APIs, driver versions, and system reboots.

Applications **can** compare the `driverUUID` value across instance and process boundaries, and **can** make similar queries in external APIs to determine whether they are capable of sharing memory objects and resources using them with the device.

`deviceUUID` and/or `driverUUID` **must** be used to determine whether a particular external object can be shared between driver components, where such a restriction exists as defined in the compatibility table for the particular object type:

- External memory handle types compatibility
- External semaphore handle types compatibility
- External fence handle types compatibility

If `deviceLUIDValid` is `VK_FALSE`, the values of `deviceLUID` and `deviceNodeMask` are undefined. If `deviceLUIDValid` is `VK_TRUE` and Vulkan is running on the Windows operating system, the contents of `deviceLUID` **can** be cast to an `LUID` object and **must** be equal to the locally unique identifier of a `IDXGIAdapter1` object that corresponds to `physicalDevice`. If `deviceLUIDValid` is `VK_TRUE`, `deviceNodeMask` **must** contain exactly one bit. If Vulkan is running on an operating system that supports the Direct3D 12 API and `physicalDevice` corresponds to an individual device in a linked device adapter, `deviceNodeMask` identifies the Direct3D 12 node corresponding to `physicalDevice`. Otherwise, `deviceNodeMask` **must** be `1`.

**NOTE**

Although they have identical descriptions, VkPhysicalDeviceIDProperties::`deviceUUID` may differ from VkPhysicalDeviceProperties2::`pipelineCacheUUID`. The former is intended to identify and correlate devices across API and driver boundaries, while the latter is used to identify a compatible device and driver combination to use when serializing and de-serializing pipeline state.

Implementations **should** return `deviceUUID` values which are likely to be unique even in the presence of multiple Vulkan implementations (such as a GPU driver and a software renderer; two drivers for different GPUs; or the same Vulkan driver running on two logically different devices).

Khronos' conformance testing is unable to guarantee that `deviceUUID` values are actually unique, so implementors **should** make their own best efforts to ensure this. In particular, hard-coded `deviceUUID` values, especially all-`0` bits, **should** never be used.

A combination of values unique to the vendor, the driver, and the hardware environment can be used to provide a `deviceUUID` which is unique to a high degree of certainty. Some possible inputs to such a computation are:

- Information reported by vkGetPhysicalDeviceProperties
- PCI device ID (if defined)
- PCI bus ID, or similar system configuration information.
- Driver binary checksums.

**NOTE**

While VkPhysicalDeviceIDProperties::`deviceUUID` is specified to remain consistent across driver versions and system reboots, it is not intended to be usable as a serializable persistent identifier for a device. It may change when a device is physically added to, removed from, or moved to a different connector in a system while that system is powered down. Further, there is no reasonable way to verify with conformance testing that a given device retains the same UUID in a given system across all driver versions supported in that system. While implementations should make every effort to report consistent device UUIDs across driver versions, applications should avoid relying on the persistence of this value for uses other

than identifying compatible devices for external object sharing purposes.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceIDProperties-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_ID_PROPERTIES`

`VK_UUID_SIZE` is the length in `uint8_t` values of an array containing a universally unique device or driver build identifier, as returned in VkPhysicalDeviceIDProperties::`deviceUUID` and VkPhysicalDeviceIDProperties::`driverUUID`.

```
#define VK_UUID_SIZE                      16U
```

`VK_LUID_SIZE` is the length in `uint8_t` values of an array containing a locally unique device identifier, as returned in VkPhysicalDeviceIDProperties::`deviceLUID`.

```
#define VK_LUID_SIZE                      8U
```

or the equivalent

```
#define VK_LUID_SIZE_KHR                  VK_LUID_SIZE
```

The `VkPhysicalDeviceDriverProperties` structure is defined as:

```
typedef struct VkPhysicalDeviceDriverProperties {
    VkStructureType         sType;
    void*                   pNext;
    VkDriverId              driverID;
    char                    driverName[VK_MAX_DRIVER_NAME_SIZE];
    char                    driverInfo[VK_MAX_DRIVER_INFO_SIZE];
    VkConformanceVersion    conformanceVersion;
} VkPhysicalDeviceDriverProperties;
```

or the equivalent

```
// Provided by VK_KHR_driver_properties
typedef VkPhysicalDeviceDriverProperties VkPhysicalDeviceDriverPropertiesKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `driverID` is a unique identifier for the driver of the physical device.

- driverName is an array of VK_MAX_DRIVER_NAME_SIZE char containing a null-terminated UTF-8 string which is the name of the driver.
- driverInfo is an array of VK_MAX_DRIVER_INFO_SIZE char containing a null-terminated UTF-8 string with additional information about the driver.
- conformanceVersion is the latest version of the Vulkan conformance test that the implementor has successfully tested this driver against prior to release (see VkConformanceVersion).

If the VkPhysicalDeviceDriverProperties structure is included in the pNext chain of the VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with each corresponding implementation-dependent property.

These are properties of the driver corresponding to a physical device.

driverID **must** be immutable for a given driver across instances, processes, driver versions, and system reboots.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceDriverProperties-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DRIVER_PROPERTIES

Khronos driver IDs which **may** be returned in VkPhysicalDeviceDriverProperties::driverID are:

```
typedef enum VkDriverId {
    VK_DRIVER_ID_AMD_PROPRIETARY = 1,
    VK_DRIVER_ID_AMD_OPEN_SOURCE = 2,
    VK_DRIVER_ID_MESA_RADV = 3,
    VK_DRIVER_ID_NVIDIA_PROPRIETARY = 4,
    VK_DRIVER_ID_INTEL_PROPRIETARY_WINDOWS = 5,
    VK_DRIVER_ID_INTEL_OPEN_SOURCE_MESA = 6,
    VK_DRIVER_ID_IMAGINATION_PROPRIETARY = 7,
    VK_DRIVER_ID_QUALCOMM_PROPRIETARY = 8,
    VK_DRIVER_ID_ARM_PROPRIETARY = 9,
    VK_DRIVER_ID_GOOGLE_SWIFTSHADER = 10,
    VK_DRIVER_ID_GGP_PROPRIETARY = 11,
    VK_DRIVER_ID_BROADCOM_PROPRIETARY = 12,
    VK_DRIVER_ID_MESA_LLVMPIPE = 13,
    VK_DRIVER_ID_MOLTENVK = 14,
    VK_DRIVER_ID_COREAVI_PROPRIETARY = 15,
    VK_DRIVER_ID_JUICE_PROPRIETARY = 16,
    VK_DRIVER_ID_VERISILICON_PROPRIETARY = 17,
    VK_DRIVER_ID_MESA_TURNIP = 18,
    VK_DRIVER_ID_MESA_V3DV = 19,
    VK_DRIVER_ID_MESA_PANVK = 20,
    VK_DRIVER_ID_SAMSUNG_PROPRIETARY = 21,
    VK_DRIVER_ID_MESA_VENUS = 22,
    VK_DRIVER_ID_MESA_DOZEN = 23,
    VK_DRIVER_ID_MESA_NVK = 24,
```

```
    VK_DRIVER_ID_IMAGINATION_OPEN_SOURCE_MESA = 25,
    VK_DRIVER_ID_MESA_HONEYKRISP = 26,
    VK_DRIVER_ID_RESERVED_27 = 27,
  // Provided by VK_KHR_driver_properties
    VK_DRIVER_ID_AMD_PROPRIETARY_KHR = VK_DRIVER_ID_AMD_PROPRIETARY,
  // Provided by VK_KHR_driver_properties
    VK_DRIVER_ID_AMD_OPEN_SOURCE_KHR = VK_DRIVER_ID_AMD_OPEN_SOURCE,
  // Provided by VK_KHR_driver_properties
    VK_DRIVER_ID_MESA_RADV_KHR = VK_DRIVER_ID_MESA_RADV,
  // Provided by VK_KHR_driver_properties
    VK_DRIVER_ID_NVIDIA_PROPRIETARY_KHR = VK_DRIVER_ID_NVIDIA_PROPRIETARY,
  // Provided by VK_KHR_driver_properties
    VK_DRIVER_ID_INTEL_PROPRIETARY_WINDOWS_KHR =
VK_DRIVER_ID_INTEL_PROPRIETARY_WINDOWS,
  // Provided by VK_KHR_driver_properties
    VK_DRIVER_ID_INTEL_OPEN_SOURCE_MESA_KHR = VK_DRIVER_ID_INTEL_OPEN_SOURCE_MESA,
  // Provided by VK_KHR_driver_properties
    VK_DRIVER_ID_IMAGINATION_PROPRIETARY_KHR = VK_DRIVER_ID_IMAGINATION_PROPRIETARY,
  // Provided by VK_KHR_driver_properties
    VK_DRIVER_ID_QUALCOMM_PROPRIETARY_KHR = VK_DRIVER_ID_QUALCOMM_PROPRIETARY,
  // Provided by VK_KHR_driver_properties
    VK_DRIVER_ID_ARM_PROPRIETARY_KHR = VK_DRIVER_ID_ARM_PROPRIETARY,
  // Provided by VK_KHR_driver_properties
    VK_DRIVER_ID_GOOGLE_SWIFTSHADER_KHR = VK_DRIVER_ID_GOOGLE_SWIFTSHADER,
  // Provided by VK_KHR_driver_properties
    VK_DRIVER_ID_GGP_PROPRIETARY_KHR = VK_DRIVER_ID_GGP_PROPRIETARY,
  // Provided by VK_KHR_driver_properties
    VK_DRIVER_ID_BROADCOM_PROPRIETARY_KHR = VK_DRIVER_ID_BROADCOM_PROPRIETARY,
} VkDriverId;
```

or the equivalent

```
// Provided by VK_KHR_driver_properties
typedef VkDriverId VkDriverIdKHR;
```

**NOTE**

Khronos driver IDs may be allocated by vendors at any time. There may be multiple driver IDs for the same vendor, representing different drivers (for e.g. different platforms, proprietary or open source, etc.). Only the latest canonical versions of this Specification, of the corresponding `vk.xml` API Registry, and of the corresponding `vulkan_core.h` header file **must** contain all reserved Khronos driver IDs.

Only driver IDs registered with Khronos are given symbolic names. There **may** be unregistered driver IDs returned.

`VK_MAX_DRIVER_NAME_SIZE` is the length in `char` values of an array containing a driver name string, as returned in VkPhysicalDeviceDriverProperties::`driverName`.

```
#define VK_MAX_DRIVER_NAME_SIZE           256U
```

or the equivalent

```
#define VK_MAX_DRIVER_NAME_SIZE_KHR       VK_MAX_DRIVER_NAME_SIZE
```

`VK_MAX_DRIVER_INFO_SIZE` is the length in `char` values of an array containing a driver information string, as returned in VkPhysicalDeviceDriverProperties::`driverInfo`.

```
#define VK_MAX_DRIVER_INFO_SIZE           256U
```

or the equivalent

```
#define VK_MAX_DRIVER_INFO_SIZE_KHR       VK_MAX_DRIVER_INFO_SIZE
```

The conformance test suite version an implementation is compliant with is described with the `VkConformanceVersion` structure:

```
typedef struct VkConformanceVersion {
    uint8_t    major;
    uint8_t    minor;
    uint8_t    subminor;
    uint8_t    patch;
} VkConformanceVersion;
```

or the equivalent

```
// Provided by VK_KHR_driver_properties
typedef VkConformanceVersion VkConformanceVersionKHR;
```

- `major` is the major version number of the conformance test suite.
- `minor` is the minor version number of the conformance test suite.
- `subminor` is the subminor version number of the conformance test suite.
- `patch` is the patch version number of the conformance test suite.

The `VkPhysicalDevicePCIBusInfoPropertiesEXT` structure is defined as:

```
// Provided by VK_EXT_pci_bus_info
typedef struct VkPhysicalDevicePCIBusInfoPropertiesEXT {
    VkStructureType    sType;
    void*              pNext;
    uint32_t           pciDomain;
    uint32_t           pciBus;
    uint32_t           pciDevice;
    uint32_t           pciFunction;
} VkPhysicalDevicePCIBusInfoPropertiesEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `pciDomain` is the PCI bus domain.
- `pciBus` is the PCI bus identifier.
- `pciDevice` is the PCI device identifier.
- `pciFunction` is the PCI device function identifier.

If the `VkPhysicalDevicePCIBusInfoPropertiesEXT` structure is included in the `pNext` chain of the VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with each corresponding implementation-dependent property.

These are properties of the PCI bus information of a physical device.

### Valid Usage (Implicit)

- VUID-VkPhysicalDevicePCIBusInfoPropertiesEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PCI_BUS_INFO_PROPERTIES_EXT`

The `VkPhysicalDeviceDrmPropertiesEXT` structure is defined as:

```
// Provided by VK_EXT_physical_device_drm
typedef struct VkPhysicalDeviceDrmPropertiesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           hasPrimary;
    VkBool32           hasRender;
    int64_t            primaryMajor;
    int64_t            primaryMinor;
    int64_t            renderMajor;
    int64_t            renderMinor;
} VkPhysicalDeviceDrmPropertiesEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.

- `hasPrimary` is a boolean indicating whether the physical device has a DRM primary node.
- `hasRender` is a boolean indicating whether the physical device has a DRM render node.
- `primaryMajor` is the DRM primary node major number, if any.
- `primaryMinor` is the DRM primary node minor number, if any.
- `renderMajor` is the DRM render node major number, if any.
- `renderMinor` is the DRM render node minor number, if any.

If the `VkPhysicalDeviceDrmPropertiesEXT` structure is included in the `pNext` chain of the VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with each corresponding implementation-dependent property.

These are properties of the DRM information of a physical device.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceDrmPropertiesEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DRM_PROPERTIES_EXT`

The `VkPhysicalDeviceShaderIntegerDotProductProperties` structure is defined as:

```
typedef struct VkPhysicalDeviceShaderIntegerDotProductProperties {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           integerDotProduct8BitUnsignedAccelerated;
    VkBool32           integerDotProduct8BitSignedAccelerated;
    VkBool32           integerDotProduct8BitMixedSignednessAccelerated;
    VkBool32           integerDotProduct4x8BitPackedUnsignedAccelerated;
    VkBool32           integerDotProduct4x8BitPackedSignedAccelerated;
    VkBool32           integerDotProduct4x8BitPackedMixedSignednessAccelerated;
    VkBool32           integerDotProduct16BitUnsignedAccelerated;
    VkBool32           integerDotProduct16BitSignedAccelerated;
    VkBool32           integerDotProduct16BitMixedSignednessAccelerated;
    VkBool32           integerDotProduct32BitUnsignedAccelerated;
    VkBool32           integerDotProduct32BitSignedAccelerated;
    VkBool32           integerDotProduct32BitMixedSignednessAccelerated;
    VkBool32           integerDotProduct64BitUnsignedAccelerated;
    VkBool32           integerDotProduct64BitSignedAccelerated;
    VkBool32           integerDotProduct64BitMixedSignednessAccelerated;
    VkBool32           integerDotProductAccumulatingSaturating8BitUnsignedAccelerated;
    VkBool32           integerDotProductAccumulatingSaturating8BitSignedAccelerated;
    VkBool32
integerDotProductAccumulatingSaturating8BitMixedSignednessAccelerated;
    VkBool32
integerDotProductAccumulatingSaturating4x8BitPackedUnsignedAccelerated;
    VkBool32
integerDotProductAccumulatingSaturating4x8BitPackedSignedAccelerated;
    VkBool32
integerDotProductAccumulatingSaturating4x8BitPackedMixedSignednessAccelerated;
    VkBool32
integerDotProductAccumulatingSaturating16BitUnsignedAccelerated;
    VkBool32           integerDotProductAccumulatingSaturating16BitSignedAccelerated;
    VkBool32
integerDotProductAccumulatingSaturating16BitMixedSignednessAccelerated;
    VkBool32
integerDotProductAccumulatingSaturating32BitUnsignedAccelerated;
    VkBool32           integerDotProductAccumulatingSaturating32BitSignedAccelerated;
    VkBool32
integerDotProductAccumulatingSaturating32BitMixedSignednessAccelerated;
    VkBool32
integerDotProductAccumulatingSaturating64BitUnsignedAccelerated;
    VkBool32           integerDotProductAccumulatingSaturating64BitSignedAccelerated;
    VkBool32
integerDotProductAccumulatingSaturating64BitMixedSignednessAccelerated;
} VkPhysicalDeviceShaderIntegerDotProductProperties;
```

or the equivalent

```
// Provided by VK_KHR_shader_integer_dot_product
typedef VkPhysicalDeviceShaderIntegerDotProductProperties
VkPhysicalDeviceShaderIntegerDotProductPropertiesKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- integerDotProduct8BitUnsignedAccelerated is a boolean that will be VK_TRUE if the support for 8-bit unsigned dot product operations using the OpUDotKHR SPIR-V instruction is accelerated as defined below.
- integerDotProduct8BitSignedAccelerated is a boolean that will be VK_TRUE if the support for 8-bit signed dot product operations using the OpSDotKHR SPIR-V instruction is accelerated as defined below.
- integerDotProduct8BitMixedSignednessAccelerated is a boolean that will be VK_TRUE if the support for 8-bit mixed signedness dot product operations using the OpSUDotKHR SPIR-V instruction is accelerated as defined below.
- integerDotProduct4x8BitPackedUnsignedAccelerated is a boolean that will be VK_TRUE if the support for 8-bit unsigned dot product operations from operands packed into 32-bit integers using the OpUDotKHR SPIR-V instruction is accelerated as defined below.
- integerDotProduct4x8BitPackedSignedAccelerated is a boolean that will be VK_TRUE if the support for 8-bit signed dot product operations from operands packed into 32-bit integers using the OpSDotKHR SPIR-V instruction is accelerated as defined below.
- integerDotProduct4x8BitPackedMixedSignednessAccelerated is a boolean that will be VK_TRUE if the support for 8-bit mixed signedness dot product operations from operands packed into 32-bit integers using the OpSUDotKHR SPIR-V instruction is accelerated as defined below.
- integerDotProduct16BitUnsignedAccelerated is a boolean that will be VK_TRUE if the support for 16-bit unsigned dot product operations using the OpUDotKHR SPIR-V instruction is accelerated as defined below.
- integerDotProduct16BitSignedAccelerated is a boolean that will be VK_TRUE if the support for 16-bit signed dot product operations using the OpSDotKHR SPIR-V instruction is accelerated as defined below.
- integerDotProduct16BitMixedSignednessAccelerated is a boolean that will be VK_TRUE if the support for 16-bit mixed signedness dot product operations using the OpSUDotKHR SPIR-V instruction is accelerated as defined below.
- integerDotProduct32BitUnsignedAccelerated is a boolean that will be VK_TRUE if the support for 32-bit unsigned dot product operations using the OpUDotKHR SPIR-V instruction is accelerated as defined below.
- integerDotProduct32BitSignedAccelerated is a boolean that will be VK_TRUE if the support for 32-bit signed dot product operations using the OpSDotKHR SPIR-V instruction is accelerated as defined below.
- integerDotProduct32BitMixedSignednessAccelerated is a boolean that will be VK_TRUE if the support for 32-bit mixed signedness dot product operations using the OpSUDotKHR SPIR-V

instruction is accelerated as defined below.

- `integerDotProduct64BitUnsignedAccelerated` is a boolean that will be `VK_TRUE` if the support for 64-bit unsigned dot product operations using the `OpUDotKHR` SPIR-V instruction is accelerated as defined below.
- `integerDotProduct64BitSignedAccelerated` is a boolean that will be `VK_TRUE` if the support for 64-bit signed dot product operations using the `OpSDotKHR` SPIR-V instruction is accelerated as defined below.
- `integerDotProduct64BitMixedSignednessAccelerated` is a boolean that will be `VK_TRUE` if the support for 64-bit mixed signedness dot product operations using the `OpSUDotKHR` SPIR-V instruction is accelerated as defined below.
- `integerDotProductAccumulatingSaturating8BitUnsignedAccelerated` is a boolean that will be `VK_TRUE` if the support for 8-bit unsigned accumulating saturating dot product operations using the `OpUDotAccSatKHR` SPIR-V instruction is accelerated as defined below.
- `integerDotProductAccumulatingSaturating8BitSignedAccelerated` is a boolean that will be `VK_TRUE` if the support for 8-bit signed accumulating saturating dot product operations using the `OpSDotAccSatKHR` SPIR-V instruction is accelerated as defined below.
- `integerDotProductAccumulatingSaturating8BitMixedSignednessAccelerated` is a boolean that will be `VK_TRUE` if the support for 8-bit mixed signedness accumulating saturating dot product operations using the `OpSUDotAccSatKHR` SPIR-V instruction is accelerated as defined below.
- `integerDotProductAccumulatingSaturating4x8BitPackedUnsignedAccelerated` is a boolean that will be `VK_TRUE` if the support for 8-bit unsigned accumulating saturating dot product operations from operands packed into 32-bit integers using the `OpUDotAccSatKHR` SPIR-V instruction is accelerated as defined below.
- `integerDotProductAccumulatingSaturating4x8BitPackedSignedAccelerated` is a boolean that will be `VK_TRUE` if the support for 8-bit signed accumulating saturating dot product operations from operands packed into 32-bit integers using the `OpSDotAccSatKHR` SPIR-V instruction is accelerated as defined below.
- `integerDotProductAccumulatingSaturating4x8BitPackedMixedSignednessAccelerated` is a boolean that will be `VK_TRUE` if the support for 8-bit mixed signedness accumulating saturating dot product operations from operands packed into 32-bit integers using the `OpSUDotAccSatKHR` SPIR-V instruction is accelerated as defined below.
- `integerDotProductAccumulatingSaturating16BitUnsignedAccelerated` is a boolean that will be `VK_TRUE` if the support for 16-bit unsigned accumulating saturating dot product operations using the `OpUDotAccSatKHR` SPIR-V instruction is accelerated as defined below.
- `integerDotProductAccumulatingSaturating16BitSignedAccelerated` is a boolean that will be `VK_TRUE` if the support for 16-bit signed accumulating saturating dot product operations using the `OpSDotAccSatKHR` SPIR-V instruction is accelerated as defined below.
- `integerDotProductAccumulatingSaturating16BitMixedSignednessAccelerated` is a boolean that will be `VK_TRUE` if the support for 16-bit mixed signedness accumulating saturating dot product operations using the `OpSUDotAccSatKHR` SPIR-V instruction is accelerated as defined below.
- `integerDotProductAccumulatingSaturating32BitUnsignedAccelerated` is a boolean that will be `VK_TRUE` if the support for 32-bit unsigned accumulating saturating dot product operations using the `OpUDotAccSatKHR` SPIR-V instruction is accelerated as defined below.

- integerDotProductAccumulatingSaturating32BitSignedAccelerated is a boolean that will be VK_TRUE if the support for 32-bit signed accumulating saturating dot product operations using the OpSDotAccSatKHR SPIR-V instruction is accelerated as defined below.
- integerDotProductAccumulatingSaturating32BitMixedSignednessAccelerated is a boolean that will be VK_TRUE if the support for 32-bit mixed signedness accumulating saturating dot product operations using the OpSUDotAccSatKHR SPIR-V instruction is accelerated as defined below.
- integerDotProductAccumulatingSaturating64BitUnsignedAccelerated is a boolean that will be VK_TRUE if the support for 64-bit unsigned accumulating saturating dot product operations using the OpUDotAccSatKHR SPIR-V instruction is accelerated as defined below.
- integerDotProductAccumulatingSaturating64BitSignedAccelerated is a boolean that will be VK_TRUE if the support for 64-bit signed accumulating saturating dot product operations using the OpSDotAccSatKHR SPIR-V instruction is accelerated as defined below.
- integerDotProductAccumulatingSaturating64BitMixedSignednessAccelerated is a boolean that will be VK_TRUE if the support for 64-bit mixed signedness accumulating saturating dot product operations using the OpSUDotAccSatKHR SPIR-V instruction is accelerated as defined below.

If the VkPhysicalDeviceShaderIntegerDotProductProperties structure is included in the pNext chain of the VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with each corresponding implementation-dependent property.

These are properties of the integer dot product acceleration information of a physical device.

**NOTE** A dot product operation is deemed accelerated if its implementation provides a performance advantage over application-provided code composed from elementary instructions and/or other dot product instructions, either because the implementation uses optimized machine code sequences whose generation from application-provided code cannot be guaranteed or because it uses hardware features that cannot otherwise be targeted from application-provided code.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceShaderIntegerDotProductProperties-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_INTEGER_DOT_PRODUCT_PROPERTIES

The VkPhysicalDeviceImageProcessingPropertiesQCOM structure is defined as:

```
// Provided by VK_QCOM_image_processing
typedef struct VkPhysicalDeviceImageProcessingPropertiesQCOM {
    VkStructureType    sType;
    void*              pNext;
    uint32_t           maxWeightFilterPhases;
    VkExtent2D         maxWeightFilterDimension;
    VkExtent2D         maxBlockMatchRegion;
    VkExtent2D         maxBoxFilterBlockSize;
} VkPhysicalDeviceImageProcessingPropertiesQCOM;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- maxWeightFilterPhases is the maximum value that **can** be specified for VkImageViewSampleWeightCreateInfoQCOM::numPhases in weight image sampling operations.
- maxWeightFilterDimension is a VkExtent2D describing the largest dimensions (width and height) that **can** be specified for VkImageViewSampleWeightCreateInfoQCOM::filterSize.
- maxBlockMatchRegion is a VkExtent2D describing the largest dimensions (width and height) that **can** be specified for blockSize in block matching operations.
- maxBoxFilterBlockSize is a VkExtent2D describing the maximum dimensions (width and height) that **can** be specified for blocksize in box filter sampling operations.

If the VkPhysicalDeviceImageProcessingPropertiesQCOM structure is included in the pNext chain of the VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with each corresponding implementation-dependent property.

These are properties of the image processing information of a physical device.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceImageProcessingPropertiesQCOM-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_IMAGE_PROCESSING_PROPERTIES_QCOM

The VkPhysicalDeviceShaderTileImagePropertiesEXT structure is defined as:

```
// Provided by VK_EXT_shader_tile_image
typedef struct VkPhysicalDeviceShaderTileImagePropertiesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           shaderTileImageCoherentReadAccelerated;
    VkBool32           shaderTileImageReadSampleFromPixelRateInvocation;
    VkBool32           shaderTileImageReadFromHelperInvocation;
} VkPhysicalDeviceShaderTileImagePropertiesEXT;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- shaderTileImageCoherentReadAccelerated is a boolean that will be VK_TRUE if coherent reads of tile image data is accelerated.
- shaderTileImageReadSampleFromPixelRateInvocation is a boolean that will be VK_TRUE if reading from samples from a pixel rate fragment invocation is supported when VkPipelineMultisampleStateCreateInfo::rasterizationSamples > 1.
- shaderTileImageReadFromHelperInvocation is a boolean that will be VK_TRUE if reads of tile image data from helper fragment invocations result in valid values.

If the VkPhysicalDeviceShaderTileImagePropertiesEXT structure is included in the pNext chain of the

VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with each corresponding implementation-dependent property.

These are properties of the tile image information of a physical device.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceShaderTileImagePropertiesEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_TILE_IMAGE_PROPERTIES_EXT`

The `VkPhysicalDeviceImageProcessing2PropertiesQCOM` structure is defined as:

```
// Provided by VK_QCOM_image_processing2
typedef struct VkPhysicalDeviceImageProcessing2PropertiesQCOM {
    VkStructureType    sType;
    void*              pNext;
    VkExtent2D         maxBlockMatchWindow;
} VkPhysicalDeviceImageProcessing2PropertiesQCOM;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `maxBlockMatchWindow` is a VkExtent2D describing the largest dimensions (`width` and `height`) that **can** be specified for the block match window.

If the `VkPhysicalDeviceImageProcessing2PropertiesQCOM` structure is included in the `pNext` chain of the VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with each corresponding implementation-dependent property.

These are properties of the image processing2 information of a physical device.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceImageProcessing2PropertiesQCOM-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_IMAGE_PROCESSING_2_PROPERTIES_QCOM`

The `VkPhysicalDeviceLayeredDriverPropertiesMSFT` structure is defined as:

```
// Provided by VK_MSFT_layered_driver
typedef struct VkPhysicalDeviceLayeredDriverPropertiesMSFT {
    VkStructureType                    sType;
    void*                              pNext;
    VkLayeredDriverUnderlyingApiMSFT   underlyingAPI;
} VkPhysicalDeviceLayeredDriverPropertiesMSFT;
```

- `sType` is a VkStructureType value identifying this structure.

- pNext is NULL or a pointer to a structure extending this structure.
- underlyingAPI is a VkLayeredDriverUnderlyingApiMSFT value indicating which underlying API is used to implement the layered driver, or VK_LAYERED_DRIVER_UNDERLYING_API_NONE_MSFT if the driver is not layered.

These are properties of the driver layering information of a physical device.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceLayeredDriverPropertiesMSFT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_LAYERED_DRIVER_PROPERTIES_MSFT

Underlying APIs which **may** be returned in VkPhysicalDeviceLayeredDriverPropertiesMSFT::underlyingAPI are:

```
// Provided by VK_MSFT_layered_driver
typedef enum VkLayeredDriverUnderlyingApiMSFT {
    VK_LAYERED_DRIVER_UNDERLYING_API_NONE_MSFT = 0,
    VK_LAYERED_DRIVER_UNDERLYING_API_D3D12_MSFT = 1,
} VkLayeredDriverUnderlyingApiMSFT;
```

The VkPhysicalDeviceSchedulingControlsPropertiesARM structure is defined as:

```
// Provided by VK_ARM_scheduling_controls
typedef struct VkPhysicalDeviceSchedulingControlsPropertiesARM {
    VkStructureType                                 sType;
    void*                                           pNext;
    VkPhysicalDeviceSchedulingControlsFlagsARM      schedulingControlsFlags;
} VkPhysicalDeviceSchedulingControlsPropertiesARM;
```

- schedulingControlsFlags specifies the specific scheduling controls that a physical device supports.

If the VkPhysicalDeviceSchedulingControlsPropertiesARM structure is included in the pNext chain of the VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with each corresponding implementation-dependent property.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceSchedulingControlsPropertiesARM-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SCHEDULING_CONTROLS_PROPERTIES_ARM
- VUID-VkPhysicalDeviceSchedulingControlsPropertiesARM-schedulingControlsFlags-parameter
  schedulingControlsFlags **must** be a valid combination of VkPhysicalDeviceSchedulingControlsFlagBitsARM values

- VUID-VkPhysicalDeviceSchedulingControlsPropertiesARM-schedulingControlsFlags-requiredbitmask
  `schedulingControlsFlags` **must** not be `0`

Bits which **can** be set in VkPhysicalDeviceSchedulingControlsPropertiesARM::`schedulingControlsFlags`, specifying supported scheduling controls, are:

```
// Provided by VK_ARM_scheduling_controls
// Flag bits for VkPhysicalDeviceSchedulingControlsFlagBitsARM
typedef VkFlags64 VkPhysicalDeviceSchedulingControlsFlagBitsARM;
static const VkPhysicalDeviceSchedulingControlsFlagBitsARM
VK_PHYSICAL_DEVICE_SCHEDULING_CONTROLS_SHADER_CORE_COUNT_ARM = 0x00000001ULL;
```

- `VK_PHYSICAL_DEVICE_SCHEDULING_CONTROLS_SHADER_CORE_COUNT_ARM` indicates that a VkDeviceQueueShaderCoreControlCreateInfoARM structure **may** be included in the `pNext` chain of a VkDeviceQueueCreateInfo or VkDeviceCreateInfo structure.

```
// Provided by VK_ARM_scheduling_controls
typedef VkFlags64 VkPhysicalDeviceSchedulingControlsFlagsARM;
```

`VkPhysicalDeviceSchedulingControlsFlagsARM` is a bitmask type for setting a mask of zero or more VkPhysicalDeviceSchedulingControlsFlagBitsARM.

To query properties of queues available on a physical device, call:

```
// Provided by VK_VERSION_1_0
void vkGetPhysicalDeviceQueueFamilyProperties(
    VkPhysicalDevice                            physicalDevice,
    uint32_t*                                   pQueueFamilyPropertyCount,
    VkQueueFamilyProperties*                    pQueueFamilyProperties);
```

- `physicalDevice` is the handle to the physical device whose properties will be queried.
- `pQueueFamilyPropertyCount` is a pointer to an integer related to the number of queue families available or queried, as described below.
- `pQueueFamilyProperties` is either `NULL` or a pointer to an array of VkQueueFamilyProperties structures.

If `pQueueFamilyProperties` is `NULL`, then the number of queue families available is returned in `pQueueFamilyPropertyCount`. Implementations **must** support at least one queue family. Otherwise, `pQueueFamilyPropertyCount` **must** point to a variable set by the application to the number of elements in the `pQueueFamilyProperties` array, and on return the variable is overwritten with the number of structures actually written to `pQueueFamilyProperties`. If `pQueueFamilyPropertyCount` is less than the number of queue families available, at most `pQueueFamilyPropertyCount` structures will be written.

### Valid Usage (Implicit)

- VUID-vkGetPhysicalDeviceQueueFamilyProperties-physicalDevice-parameter
  `physicalDevice` **must** be a valid VkPhysicalDevice handle
- VUID-vkGetPhysicalDeviceQueueFamilyProperties-pQueueFamilyPropertyCount-parameter
  `pQueueFamilyPropertyCount` **must** be a valid pointer to a `uint32_t` value
- VUID-vkGetPhysicalDeviceQueueFamilyProperties-pQueueFamilyProperties-parameter
  If the value referenced by `pQueueFamilyPropertyCount` is not `0`, and `pQueueFamilyProperties` is not `NULL`, `pQueueFamilyProperties` **must** be a valid pointer to an array of `pQueueFamilyPropertyCount` VkQueueFamilyProperties structures

The `VkQueueFamilyProperties` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkQueueFamilyProperties {
    VkQueueFlags    queueFlags;
    uint32_t        queueCount;
    uint32_t        timestampValidBits;
    VkExtent3D      minImageTransferGranularity;
} VkQueueFamilyProperties;
```

- `queueFlags` is a bitmask of VkQueueFlagBits indicating capabilities of the queues in this queue family.
- `queueCount` is the unsigned integer count of queues in this queue family. Each queue family **must** support at least one queue.
- `timestampValidBits` is the unsigned integer count of meaningful bits in the timestamps written via vkCmdWriteTimestamp2KHR or vkCmdWriteTimestamp. The valid range for the count is 36 to 64 bits, or a value of 0, indicating no support for timestamps. Bits outside the valid range are guaranteed to be zeros.
- `minImageTransferGranularity` is the minimum granularity supported for image transfer operations on the queues in this queue family.

The value returned in `minImageTransferGranularity` has a unit of compressed texel blocks for images having a block-compressed format, and a unit of texels otherwise.

Possible values of `minImageTransferGranularity` are:

- (0,0,0) specifies that only whole mip levels **must** be transferred using the image transfer operations on the corresponding queues. In this case, the following restrictions apply to all offset and extent parameters of image transfer operations:
  - The `x`, `y`, and `z` members of a VkOffset3D parameter **must** always be zero.
  - The `width`, `height`, and `depth` members of a VkExtent3D parameter **must** always match the width, height, and depth of the image subresource corresponding to the parameter,

respectively.

- ($A_x$, $A_y$, $A_z$) where $A_x$, $A_y$, and $A_z$ are all integer powers of two. In this case the following restrictions apply to all image transfer operations:
  - `x`, `y`, and `z` of a VkOffset3D parameter **must** be integer multiples of $A_x$, $A_y$, and $A_z$, respectively.
  - `width` of a VkExtent3D parameter **must** be an integer multiple of $A_x$, or else `x` + `width` **must** equal the width of the image subresource corresponding to the parameter.
  - `height` of a VkExtent3D parameter **must** be an integer multiple of $A_y$, or else `y` + `height` **must** equal the height of the image subresource corresponding to the parameter.
  - `depth` of a VkExtent3D parameter **must** be an integer multiple of $A_z$, or else `z` + `depth` **must** equal the depth of the image subresource corresponding to the parameter.
  - If the format of the image corresponding to the parameters is one of the block-compressed formats then for the purposes of the above calculations the granularity **must** be scaled up by the compressed texel block dimensions.

Queues supporting graphics and/or compute operations **must** report (1,1,1) in `minImageTransferGranularity`, meaning that there are no additional restrictions on the granularity of image transfer operations for these queues. Other queues supporting image transfer operations are only **required** to support whole mip level transfers, thus `minImageTransferGranularity` for queues belonging to such queue families **may** be (0,0,0).

The Device Memory section describes memory properties queried from the physical device.

For physical device feature queries see the Features chapter.

Bits which **may** be set in VkQueueFamilyProperties::`queueFlags`, indicating capabilities of queues in a queue family are:

```
// Provided by VK_VERSION_1_0
typedef enum VkQueueFlagBits {
    VK_QUEUE_GRAPHICS_BIT = 0x00000001,
    VK_QUEUE_COMPUTE_BIT = 0x00000002,
    VK_QUEUE_TRANSFER_BIT = 0x00000004,
    VK_QUEUE_SPARSE_BINDING_BIT = 0x00000008,
  // Provided by VK_VERSION_1_1
    VK_QUEUE_PROTECTED_BIT = 0x00000010,
  // Provided by VK_KHR_video_decode_queue
    VK_QUEUE_VIDEO_DECODE_BIT_KHR = 0x00000020,
  // Provided by VK_KHR_video_encode_queue
    VK_QUEUE_VIDEO_ENCODE_BIT_KHR = 0x00000040,
  // Provided by VK_NV_optical_flow
    VK_QUEUE_OPTICAL_FLOW_BIT_NV = 0x00000100,
} VkQueueFlagBits;
```

- `VK_QUEUE_GRAPHICS_BIT` specifies that queues in this queue family support graphics operations.
- `VK_QUEUE_COMPUTE_BIT` specifies that queues in this queue family support compute operations.

- VK_QUEUE_TRANSFER_BIT specifies that queues in this queue family support transfer operations.
- VK_QUEUE_SPARSE_BINDING_BIT specifies that queues in this queue family support sparse memory management operations (see Sparse Resources). If any of the sparse resource features are enabled, then at least one queue family **must** support this bit.
- VK_QUEUE_VIDEO_DECODE_BIT_KHR specifies that queues in this queue family support video decode operations.
- VK_QUEUE_VIDEO_ENCODE_BIT_KHR specifies that queues in this queue family support video encode operations.
- VK_QUEUE_OPTICAL_FLOW_BIT_NV specifies that queues in this queue family support optical flow operations.
- VK_QUEUE_PROTECTED_BIT specifies that queues in this queue family support the VK_DEVICE_QUEUE_CREATE_PROTECTED_BIT bit. (see Protected Memory). If the physical device supports the protectedMemory feature, at least one of its queue families **must** support this bit.

If an implementation exposes any queue family that supports graphics operations, at least one queue family of at least one physical device exposed by the implementation **must** support both graphics and compute operations.

Furthermore, if the protectedMemory physical device feature is supported, then at least one queue family of at least one physical device exposed by the implementation **must** support graphics operations, compute operations, and protected memory operations.

**NOTE** All commands that are allowed on a queue that supports transfer operations are also allowed on a queue that supports either graphics or compute operations. Thus, if the capabilities of a queue family include VK_QUEUE_GRAPHICS_BIT or VK_QUEUE_COMPUTE_BIT, then reporting the VK_QUEUE_TRANSFER_BIT capability separately for that queue family is **optional**.

For further details see Queues.

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkQueueFlags;
```

VkQueueFlags is a bitmask type for setting a mask of zero or more VkQueueFlagBits.

To query properties of queues available on a physical device, call:

```
// Provided by VK_VERSION_1_1
void vkGetPhysicalDeviceQueueFamilyProperties2(
    VkPhysicalDevice                            physicalDevice,
    uint32_t*                                   pQueueFamilyPropertyCount,
    VkQueueFamilyProperties2*                   pQueueFamilyProperties);
```

or the equivalent command

```
// Provided by VK_KHR_get_physical_device_properties2
void vkGetPhysicalDeviceQueueFamilyProperties2KHR(
    VkPhysicalDevice                            physicalDevice,
    uint32_t*                                   pQueueFamilyPropertyCount,
    VkQueueFamilyProperties2*                   pQueueFamilyProperties);
```

- `physicalDevice` is the handle to the physical device whose properties will be queried.
- `pQueueFamilyPropertyCount` is a pointer to an integer related to the number of queue families available or queried, as described in vkGetPhysicalDeviceQueueFamilyProperties.
- `pQueueFamilyProperties` is either `NULL` or a pointer to an array of VkQueueFamilyProperties2 structures.

`vkGetPhysicalDeviceQueueFamilyProperties2` behaves similarly to vkGetPhysicalDeviceQueueFamilyProperties, with the ability to return extended information in a `pNext` chain of output structures.

### Valid Usage (Implicit)

- VUID-vkGetPhysicalDeviceQueueFamilyProperties2-physicalDevice-parameter
  `physicalDevice` **must** be a valid VkPhysicalDevice handle
- VUID-vkGetPhysicalDeviceQueueFamilyProperties2-pQueueFamilyPropertyCount-parameter
  `pQueueFamilyPropertyCount` **must** be a valid pointer to a `uint32_t` value
- VUID-vkGetPhysicalDeviceQueueFamilyProperties2-pQueueFamilyProperties-parameter
  If the value referenced by `pQueueFamilyPropertyCount` is not `0`, and `pQueueFamilyProperties` is not `NULL`, `pQueueFamilyProperties` **must** be a valid pointer to an array of `pQueueFamilyPropertyCount` VkQueueFamilyProperties2 structures

The `VkQueueFamilyProperties2` structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkQueueFamilyProperties2 {
    VkStructureType             sType;
    void*                       pNext;
    VkQueueFamilyProperties     queueFamilyProperties;
} VkQueueFamilyProperties2;
```

or the equivalent

```
// Provided by VK_KHR_get_physical_device_properties2
typedef VkQueueFamilyProperties2 VkQueueFamilyProperties2KHR;
```

- `sType` is a VkStructureType value identifying this structure.

- pNext is NULL or a pointer to a structure extending this structure.
- queueFamilyProperties is a VkQueueFamilyProperties structure which is populated with the same values as in vkGetPhysicalDeviceQueueFamilyProperties.

### Valid Usage (Implicit)

- VUID-VkQueueFamilyProperties2-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_QUEUE_FAMILY_PROPERTIES_2
- VUID-VkQueueFamilyProperties2-pNext-pNext
  Each pNext member of any structure (including this one) in the pNext chain **must** be either NULL or a pointer to a valid instance of VkQueueFamilyCheckpointProperties2NV, VkQueueFamilyCheckpointPropertiesNV, VkQueueFamilyGlobalPriorityPropertiesKHR, VkQueueFamilyQueryResultStatusPropertiesKHR, or VkQueueFamilyVideoPropertiesKHR
- VUID-VkQueueFamilyProperties2-sType-unique
  The sType value of each struct in the pNext chain **must** be unique

The VkQueueFamilyGlobalPriorityPropertiesKHR structure is defined as:

```
// Provided by VK_KHR_global_priority
typedef struct VkQueueFamilyGlobalPriorityPropertiesKHR {
    VkStructureType             sType;
    void*                       pNext;
    uint32_t                    priorityCount;
    VkQueueGlobalPriorityKHR    priorities[VK_MAX_GLOBAL_PRIORITY_SIZE_KHR];
} VkQueueFamilyGlobalPriorityPropertiesKHR;
```

or the equivalent

```
// Provided by VK_EXT_global_priority_query
typedef VkQueueFamilyGlobalPriorityPropertiesKHR
VkQueueFamilyGlobalPriorityPropertiesEXT;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- priorityCount is the number of supported global queue priorities in this queue family, and it **must** be greater than 0.
- priorities is an array of VK_MAX_GLOBAL_PRIORITY_SIZE_KHR VkQueueGlobalPriorityKHR enums representing all supported global queue priorities in this queue family. The first priorityCount elements of the array will be valid.

If the VkQueueFamilyGlobalPriorityPropertiesKHR structure is included in the pNext chain of the VkQueueFamilyProperties2 structure passed to vkGetPhysicalDeviceQueueFamilyProperties2, it is filled in with the list of supported global queue priorities for the indicated family.

The valid elements of priorities **must** not contain any duplicate values.

The valid elements of priorities **must** be a continuous sequence of VkQueueGlobalPriorityKHR enums in the ascending order.

| | |
|---|---|
| **NOTE** | For example, returning priorityCount as 3 with supported priorities as VK_QUEUE_GLOBAL_PRIORITY_LOW_KHR, VK_QUEUE_GLOBAL_PRIORITY_MEDIUM_KHR and VK_QUEUE_GLOBAL_PRIORITY_REALTIME_KHR is not allowed. |

### Valid Usage (Implicit)

- VUID-VkQueueFamilyGlobalPriorityPropertiesKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_QUEUE_FAMILY_GLOBAL_PRIORITY_PROPERTIES_KHR

VK_MAX_GLOBAL_PRIORITY_SIZE_KHR is the length of an array of VkQueueGlobalPriorityKHR enumerants representing supported queue priorities, as returned in VkQueueFamilyGlobalPriorityPropertiesKHR::priorities.

```
#define VK_MAX_GLOBAL_PRIORITY_SIZE_KHR   16U
```

or the equivalent

```
#define VK_MAX_GLOBAL_PRIORITY_SIZE_EXT   VK_MAX_GLOBAL_PRIORITY_SIZE_KHR
```

The VkQueueFamilyCheckpointProperties2NV structure is defined as:

```
// Provided by VK_KHR_synchronization2 with VK_NV_device_diagnostic_checkpoints
typedef struct VkQueueFamilyCheckpointProperties2NV {
    VkStructureType          sType;
    void*                    pNext;
    VkPipelineStageFlags2    checkpointExecutionStageMask;
} VkQueueFamilyCheckpointProperties2NV;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- checkpointExecutionStageMask is a mask indicating which pipeline stages the implementation can execute checkpoint markers in.

Additional queue family information can be queried by setting VkQueueFamilyProperties2::pNext to point to a VkQueueFamilyCheckpointProperties2NV structure.

### Valid Usage (Implicit)

- VUID-VkQueueFamilyCheckpointProperties2NV-sType-sType

sType **must** be VK_STRUCTURE_TYPE_QUEUE_FAMILY_CHECKPOINT_PROPERTIES_2_NV

The VkQueueFamilyCheckpointPropertiesNV structure is defined as:

```
// Provided by VK_NV_device_diagnostic_checkpoints
typedef struct VkQueueFamilyCheckpointPropertiesNV {
    VkStructureType         sType;
    void*                   pNext;
    VkPipelineStageFlags    checkpointExecutionStageMask;
} VkQueueFamilyCheckpointPropertiesNV;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- checkpointExecutionStageMask is a mask indicating which pipeline stages the implementation can execute checkpoint markers in.

Additional queue family information can be queried by setting VkQueueFamilyProperties2::pNext to point to a VkQueueFamilyCheckpointPropertiesNV structure.

### Valid Usage (Implicit)

- VUID-VkQueueFamilyCheckpointPropertiesNV-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_QUEUE_FAMILY_CHECKPOINT_PROPERTIES_NV

The VkQueueFamilyVideoPropertiesKHR structure is defined as:

```
// Provided by VK_KHR_video_queue
typedef struct VkQueueFamilyVideoPropertiesKHR {
    VkStructureType                 sType;
    void*                           pNext;
    VkVideoCodecOperationFlagsKHR   videoCodecOperations;
} VkQueueFamilyVideoPropertiesKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- videoCodecOperations is a bitmask of VkVideoCodecOperationFlagBitsKHR that indicates the set of video codec operations supported by the queue family.

If this structure is included in the pNext chain of the VkQueueFamilyProperties2 structure passed to vkGetPhysicalDeviceQueueFamilyProperties2, then it is filled with the set of video codec operations supported by the specified queue family.

### Valid Usage (Implicit)

- VUID-VkQueueFamilyVideoPropertiesKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_QUEUE_FAMILY_VIDEO_PROPERTIES_KHR`

The VkQueueFamilyQueryResultStatusPropertiesKHR structure is defined as:

```
// Provided by VK_KHR_video_queue
typedef struct VkQueueFamilyQueryResultStatusPropertiesKHR {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           queryResultStatusSupport;
} VkQueueFamilyQueryResultStatusPropertiesKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `queryResultStatusSupport` reports `VK_TRUE` if query type `VK_QUERY_TYPE_RESULT_STATUS_ONLY_KHR` and use of `VK_QUERY_RESULT_WITH_STATUS_BIT_KHR` are supported.

If this structure is included in the `pNext` chain of the VkQueueFamilyProperties2 structure passed to vkGetPhysicalDeviceQueueFamilyProperties2, then it is filled with information about whether result status queries are supported by the specified queue family.

### Valid Usage (Implicit)

- VUID-VkQueueFamilyQueryResultStatusPropertiesKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_QUEUE_FAMILY_QUERY_RESULT_STATUS_PROPERTIES_KHR`

To enumerate the performance query counters available on a queue family of a physical device, call:

```
// Provided by VK_KHR_performance_query
VkResult vkEnumeratePhysicalDeviceQueueFamilyPerformanceQueryCountersKHR(
    VkPhysicalDevice                            physicalDevice,
    uint32_t                                    queueFamilyIndex,
    uint32_t*                                   pCounterCount,
    VkPerformanceCounterKHR*                    pCounters,
    VkPerformanceCounterDescriptionKHR*         pCounterDescriptions);
```

- `physicalDevice` is the handle to the physical device whose queue family performance query counter properties will be queried.
- `queueFamilyIndex` is the index into the queue family of the physical device we want to get properties for.

- pCounterCount is a pointer to an integer related to the number of counters available or queried, as described below.
- pCounters is either NULL or a pointer to an array of VkPerformanceCounterKHR structures.
- pCounterDescriptions is either NULL or a pointer to an array of VkPerformanceCounterDescriptionKHR structures.

If pCounters is NULL and pCounterDescriptions is NULL, then the number of counters available is returned in pCounterCount. Otherwise, pCounterCount **must** point to a variable set by the application to the number of elements in the pCounters, pCounterDescriptions, or both arrays and on return the variable is overwritten with the number of structures actually written out. If pCounterCount is less than the number of counters available, at most pCounterCount structures will be written, and VK_INCOMPLETE will be returned instead of VK_SUCCESS, to indicate that not all the available counters were returned.

## Valid Usage (Implicit)

- VUID-vkEnumeratePhysicalDeviceQueueFamilyPerformanceQueryCountersKHR-physicalDevice-parameter
  physicalDevice **must** be a valid VkPhysicalDevice handle
- VUID-vkEnumeratePhysicalDeviceQueueFamilyPerformanceQueryCountersKHR-pCounterCount-parameter
  pCounterCount **must** be a valid pointer to a uint32_t value
- VUID-vkEnumeratePhysicalDeviceQueueFamilyPerformanceQueryCountersKHR-pCounters-parameter
  If the value referenced by pCounterCount is not 0, and pCounters is not NULL, pCounters **must** be a valid pointer to an array of pCounterCount VkPerformanceCounterKHR structures
- VUID-vkEnumeratePhysicalDeviceQueueFamilyPerformanceQueryCountersKHR-pCounterDescriptions-parameter
  If the value referenced by pCounterCount is not 0, and pCounterDescriptions is not NULL, pCounterDescriptions **must** be a valid pointer to an array of pCounterCount VkPerformanceCounterDescriptionKHR structures

## Return Codes

**Success**

- VK_SUCCESS
- VK_INCOMPLETE

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY
- VK_ERROR_INITIALIZATION_FAILED

The VkPerformanceCounterKHR structure is defined as:

```
// Provided by VK_KHR_performance_query
typedef struct VkPerformanceCounterKHR {
    VkStructureType                    sType;
    void*                              pNext;
    VkPerformanceCounterUnitKHR        unit;
    VkPerformanceCounterScopeKHR       scope;
    VkPerformanceCounterStorageKHR     storage;
    uint8_t                            uuid[VK_UUID_SIZE];
} VkPerformanceCounterKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- unit is a VkPerformanceCounterUnitKHR specifying the unit that the counter data will record.
- scope is a VkPerformanceCounterScopeKHR specifying the scope that the counter belongs to.
- storage is a VkPerformanceCounterStorageKHR specifying the storage type that the counter's data uses.
- uuid is an array of size VK_UUID_SIZE, containing 8-bit values that represent a universally unique identifier for the counter of the physical device.

### Valid Usage (Implicit)

- VUID-VkPerformanceCounterKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PERFORMANCE_COUNTER_KHR
- VUID-VkPerformanceCounterKHR-pNext-pNext
  pNext **must** be NULL

Performance counters have an associated unit. This unit describes how to interpret the performance counter result.

The performance counter unit types which **may** be returned in VkPerformanceCounterKHR::unit are:

```
// Provided by VK_KHR_performance_query
typedef enum VkPerformanceCounterUnitKHR {
    VK_PERFORMANCE_COUNTER_UNIT_GENERIC_KHR = 0,
    VK_PERFORMANCE_COUNTER_UNIT_PERCENTAGE_KHR = 1,
    VK_PERFORMANCE_COUNTER_UNIT_NANOSECONDS_KHR = 2,
    VK_PERFORMANCE_COUNTER_UNIT_BYTES_KHR = 3,
    VK_PERFORMANCE_COUNTER_UNIT_BYTES_PER_SECOND_KHR = 4,
    VK_PERFORMANCE_COUNTER_UNIT_KELVIN_KHR = 5,
    VK_PERFORMANCE_COUNTER_UNIT_WATTS_KHR = 6,
    VK_PERFORMANCE_COUNTER_UNIT_VOLTS_KHR = 7,
    VK_PERFORMANCE_COUNTER_UNIT_AMPS_KHR = 8,
    VK_PERFORMANCE_COUNTER_UNIT_HERTZ_KHR = 9,
    VK_PERFORMANCE_COUNTER_UNIT_CYCLES_KHR = 10,
} VkPerformanceCounterUnitKHR;
```

- `VK_PERFORMANCE_COUNTER_UNIT_GENERIC_KHR` - the performance counter unit is a generic data point.
- `VK_PERFORMANCE_COUNTER_UNIT_PERCENTAGE_KHR` - the performance counter unit is a percentage (%).
- `VK_PERFORMANCE_COUNTER_UNIT_NANOSECONDS_KHR` - the performance counter unit is a value of nanoseconds (ns).
- `VK_PERFORMANCE_COUNTER_UNIT_BYTES_KHR` - the performance counter unit is a value of bytes.
- `VK_PERFORMANCE_COUNTER_UNIT_BYTES_PER_SECOND_KHR` - the performance counter unit is a value of bytes/s.
- `VK_PERFORMANCE_COUNTER_UNIT_KELVIN_KHR` - the performance counter unit is a temperature reported in Kelvin.
- `VK_PERFORMANCE_COUNTER_UNIT_WATTS_KHR` - the performance counter unit is a value of watts (W).
- `VK_PERFORMANCE_COUNTER_UNIT_VOLTS_KHR` - the performance counter unit is a value of volts (V).
- `VK_PERFORMANCE_COUNTER_UNIT_AMPS_KHR` - the performance counter unit is a value of amps (A).
- `VK_PERFORMANCE_COUNTER_UNIT_HERTZ_KHR` - the performance counter unit is a value of hertz (Hz).
- `VK_PERFORMANCE_COUNTER_UNIT_CYCLES_KHR` - the performance counter unit is a value of cycles.

Performance counters have an associated scope. This scope describes the granularity of a performance counter.

The performance counter scope types which **may** be returned in VkPerformanceCounterKHR::`scope` are:

```
// Provided by VK_KHR_performance_query
typedef enum VkPerformanceCounterScopeKHR {
    VK_PERFORMANCE_COUNTER_SCOPE_COMMAND_BUFFER_KHR = 0,
    VK_PERFORMANCE_COUNTER_SCOPE_RENDER_PASS_KHR = 1,
    VK_PERFORMANCE_COUNTER_SCOPE_COMMAND_KHR = 2,
  // VK_QUERY_SCOPE_COMMAND_BUFFER_KHR is a deprecated alias
    VK_QUERY_SCOPE_COMMAND_BUFFER_KHR =
VK_PERFORMANCE_COUNTER_SCOPE_COMMAND_BUFFER_KHR,
  // VK_QUERY_SCOPE_RENDER_PASS_KHR is a deprecated alias
    VK_QUERY_SCOPE_RENDER_PASS_KHR = VK_PERFORMANCE_COUNTER_SCOPE_RENDER_PASS_KHR,
  // VK_QUERY_SCOPE_COMMAND_KHR is a deprecated alias
    VK_QUERY_SCOPE_COMMAND_KHR = VK_PERFORMANCE_COUNTER_SCOPE_COMMAND_KHR,
} VkPerformanceCounterScopeKHR;
```

- `VK_PERFORMANCE_COUNTER_SCOPE_COMMAND_BUFFER_KHR` - the performance counter scope is a single complete command buffer.
- `VK_PERFORMANCE_COUNTER_SCOPE_RENDER_PASS_KHR` - the performance counter scope is zero or more complete render passes. The performance query containing the performance counter **must** begin and end outside a render pass instance.
- `VK_PERFORMANCE_COUNTER_SCOPE_COMMAND_KHR` - the performance counter scope is zero or more commands.

Performance counters have an associated storage. This storage describes the payload of a counter result.

The performance counter storage types which **may** be returned in VkPerformanceCounterKHR ::`storage` are:

```
// Provided by VK_KHR_performance_query
typedef enum VkPerformanceCounterStorageKHR {
    VK_PERFORMANCE_COUNTER_STORAGE_INT32_KHR = 0,
    VK_PERFORMANCE_COUNTER_STORAGE_INT64_KHR = 1,
    VK_PERFORMANCE_COUNTER_STORAGE_UINT32_KHR = 2,
    VK_PERFORMANCE_COUNTER_STORAGE_UINT64_KHR = 3,
    VK_PERFORMANCE_COUNTER_STORAGE_FLOAT32_KHR = 4,
    VK_PERFORMANCE_COUNTER_STORAGE_FLOAT64_KHR = 5,
} VkPerformanceCounterStorageKHR;
```

- `VK_PERFORMANCE_COUNTER_STORAGE_INT32_KHR` - the performance counter storage is a 32-bit signed integer.
- `VK_PERFORMANCE_COUNTER_STORAGE_INT64_KHR` - the performance counter storage is a 64-bit signed integer.
- `VK_PERFORMANCE_COUNTER_STORAGE_UINT32_KHR` - the performance counter storage is a 32-bit unsigned integer.
- `VK_PERFORMANCE_COUNTER_STORAGE_UINT64_KHR` - the performance counter storage is a 64-bit unsigned integer.

- VK_PERFORMANCE_COUNTER_STORAGE_FLOAT32_KHR - the performance counter storage is a 32-bit floating-point.
- VK_PERFORMANCE_COUNTER_STORAGE_FLOAT64_KHR - the performance counter storage is a 64-bit floating-point.

The VkPerformanceCounterDescriptionKHR structure is defined as:

```
// Provided by VK_KHR_performance_query
typedef struct VkPerformanceCounterDescriptionKHR {
    VkStructureType                             sType;
    void*                                       pNext;
    VkPerformanceCounterDescriptionFlagsKHR     flags;
    char                                        name[VK_MAX_DESCRIPTION_SIZE];
    char                                        category[VK_MAX_DESCRIPTION_SIZE];
    char                                        description[VK_MAX_DESCRIPTION_SIZE];
} VkPerformanceCounterDescriptionKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- flags is a bitmask of VkPerformanceCounterDescriptionFlagBitsKHR indicating the usage behavior for the counter.
- name is an array of size VK_MAX_DESCRIPTION_SIZE, containing a null-terminated UTF-8 string specifying the name of the counter.
- category is an array of size VK_MAX_DESCRIPTION_SIZE, containing a null-terminated UTF-8 string specifying the category of the counter.
- description is an array of size VK_MAX_DESCRIPTION_SIZE, containing a null-terminated UTF-8 string specifying the description of the counter.

### Valid Usage (Implicit)

- VUID-VkPerformanceCounterDescriptionKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PERFORMANCE_COUNTER_DESCRIPTION_KHR
- VUID-VkPerformanceCounterDescriptionKHR-pNext-pNext
  pNext **must** be NULL

Bits which **can** be set in VkPerformanceCounterDescriptionKHR::flags, specifying usage behavior of a performance counter, are:

```
// Provided by VK_KHR_performance_query
typedef enum VkPerformanceCounterDescriptionFlagBitsKHR {
    VK_PERFORMANCE_COUNTER_DESCRIPTION_PERFORMANCE_IMPACTING_BIT_KHR = 0x00000001,
    VK_PERFORMANCE_COUNTER_DESCRIPTION_CONCURRENTLY_IMPACTED_BIT_KHR = 0x00000002,
  // VK_PERFORMANCE_COUNTER_DESCRIPTION_PERFORMANCE_IMPACTING_KHR is a deprecated
alias
    VK_PERFORMANCE_COUNTER_DESCRIPTION_PERFORMANCE_IMPACTING_KHR =
VK_PERFORMANCE_COUNTER_DESCRIPTION_PERFORMANCE_IMPACTING_BIT_KHR,
  // VK_PERFORMANCE_COUNTER_DESCRIPTION_CONCURRENTLY_IMPACTED_KHR is a deprecated
alias
    VK_PERFORMANCE_COUNTER_DESCRIPTION_CONCURRENTLY_IMPACTED_KHR =
VK_PERFORMANCE_COUNTER_DESCRIPTION_CONCURRENTLY_IMPACTED_BIT_KHR,
} VkPerformanceCounterDescriptionFlagBitsKHR;
```

- VK_PERFORMANCE_COUNTER_DESCRIPTION_PERFORMANCE_IMPACTING_BIT_KHR specifies that recording the counter **may** have a noticeable performance impact.
- VK_PERFORMANCE_COUNTER_DESCRIPTION_CONCURRENTLY_IMPACTED_BIT_KHR specifies that concurrently recording the counter while other submitted command buffers are running **may** impact the accuracy of the recording.

```
// Provided by VK_KHR_performance_query
typedef VkFlags VkPerformanceCounterDescriptionFlagsKHR;
```

VkPerformanceCounterDescriptionFlagsKHR is a bitmask type for setting a mask of zero or more VkPerformanceCounterDescriptionFlagBitsKHR.

## 5.2. Devices

Device objects represent logical connections to physical devices. Each device exposes a number of *queue families* each having one or more *queues*. All queues in a queue family support the same operations.

As described in Physical Devices, a Vulkan application will first query for all physical devices in a system. Each physical device **can** then be queried for its capabilities, including its queue and queue family properties. Once an acceptable physical device is identified, an application will create a corresponding logical device. The created logical device is then the primary interface to the physical device.

How to enumerate the physical devices in a system and query those physical devices for their queue family properties is described in the Physical Device Enumeration section above.

A single logical device **can** be created from multiple physical devices, if those physical devices belong to the same device group. A *device group* is a set of physical devices that support accessing each other's memory and recording a single command buffer that **can** be executed on all the physical devices. Device groups are enumerated by calling vkEnumeratePhysicalDeviceGroups, and a logical device is created from a subset of the physical devices in a device group by passing the physical devices through VkDeviceGroupDeviceCreateInfo. For two physical devices to be in the

same device group, they **must** support identical extensions, features, and properties.

**NOTE** Physical devices in the same device group **must** be so similar because there are no rules for how different features/properties would interact. They **must** return the same values for nearly every invariant `vkGetPhysicalDevice*` feature, property, capability, etc., but could potentially differ for certain queries based on things like having a different display connected, or a different compositor. The specification does not attempt to enumerate which state is in each category, because such a list would quickly become out of date.

To retrieve a list of the device groups present in the system, call:

```
// Provided by VK_VERSION_1_1
VkResult vkEnumeratePhysicalDeviceGroups(
    VkInstance                                  instance,
    uint32_t*                                   pPhysicalDeviceGroupCount,
    VkPhysicalDeviceGroupProperties*            pPhysicalDeviceGroupProperties);
```

or the equivalent command

```
// Provided by VK_KHR_device_group_creation
VkResult vkEnumeratePhysicalDeviceGroupsKHR(
    VkInstance                                  instance,
    uint32_t*                                   pPhysicalDeviceGroupCount,
    VkPhysicalDeviceGroupProperties*            pPhysicalDeviceGroupProperties);
```

- `instance` is a handle to a Vulkan instance previously created with vkCreateInstance.
- `pPhysicalDeviceGroupCount` is a pointer to an integer related to the number of device groups available or queried, as described below.
- `pPhysicalDeviceGroupProperties` is either `NULL` or a pointer to an array of VkPhysicalDeviceGroupProperties structures.

If `pPhysicalDeviceGroupProperties` is `NULL`, then the number of device groups available is returned in `pPhysicalDeviceGroupCount`. Otherwise, `pPhysicalDeviceGroupCount` **must** point to a variable set by the application to the number of elements in the `pPhysicalDeviceGroupProperties` array, and on return the variable is overwritten with the number of structures actually written to `pPhysicalDeviceGroupProperties`. If `pPhysicalDeviceGroupCount` is less than the number of device groups available, at most `pPhysicalDeviceGroupCount` structures will be written, and `VK_INCOMPLETE` will be returned instead of `VK_SUCCESS`, to indicate that not all the available device groups were returned.

Every physical device **must** be in exactly one device group.

### Valid Usage (Implicit)

- VUID-vkEnumeratePhysicalDeviceGroups-instance-parameter

instance **must** be a valid VkInstance handle

- VUID-vkEnumeratePhysicalDeviceGroups-pPhysicalDeviceGroupCount-parameter
  pPhysicalDeviceGroupCount **must** be a valid pointer to a uint32_t value
- VUID-vkEnumeratePhysicalDeviceGroups-pPhysicalDeviceGroupProperties-parameter
  If the value referenced by pPhysicalDeviceGroupCount is not 0, and pPhysicalDeviceGroupProperties is not NULL, pPhysicalDeviceGroupProperties **must** be a valid pointer to an array of pPhysicalDeviceGroupCount VkPhysicalDeviceGroupProperties structures

## Return Codes

**Success**

- VK_SUCCESS
- VK_INCOMPLETE

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY
- VK_ERROR_INITIALIZATION_FAILED

The VkPhysicalDeviceGroupProperties structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkPhysicalDeviceGroupProperties {
    VkStructureType     sType;
    void*               pNext;
    uint32_t            physicalDeviceCount;
    VkPhysicalDevice    physicalDevices[VK_MAX_DEVICE_GROUP_SIZE];
    VkBool32            subsetAllocation;
} VkPhysicalDeviceGroupProperties;
```

or the equivalent

```
// Provided by VK_KHR_device_group_creation
typedef VkPhysicalDeviceGroupProperties VkPhysicalDeviceGroupPropertiesKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- physicalDeviceCount is the number of physical devices in the group.
- physicalDevices is an array of VK_MAX_DEVICE_GROUP_SIZE VkPhysicalDevice handles representing all physical devices in the group. The first physicalDeviceCount elements of the array will be valid.

- subsetAllocation specifies whether logical devices created from the group support allocating device memory on a subset of devices, via the deviceMask member of the VkMemoryAllocateFlagsInfo. If this is VK_FALSE, then all device memory allocations are made across all physical devices in the group. If physicalDeviceCount is 1, then subsetAllocation **must** be VK_FALSE.

**Valid Usage (Implicit)**

- VUID-VkPhysicalDeviceGroupProperties-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_GROUP_PROPERTIES
- VUID-VkPhysicalDeviceGroupProperties-pNext-pNext
  pNext **must** be NULL

VK_MAX_DEVICE_GROUP_SIZE is the length of an array containing VkPhysicalDevice handle values representing all physical devices in a group, as returned in VkPhysicalDeviceGroupProperties ::physicalDevices.

```
#define VK_MAX_DEVICE_GROUP_SIZE          32U
```

or the equivalent

```
#define VK_MAX_DEVICE_GROUP_SIZE_KHR      VK_MAX_DEVICE_GROUP_SIZE
```

### 5.2.1. Device Creation

Logical devices are represented by VkDevice handles:

```
// Provided by VK_VERSION_1_0
VK_DEFINE_HANDLE(VkDevice)
```

A logical device is created as a *connection* to a physical device. To create a logical device, call:

```
// Provided by VK_VERSION_1_0
VkResult vkCreateDevice(
    VkPhysicalDevice                            physicalDevice,
    const VkDeviceCreateInfo*                   pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkDevice*                                   pDevice);
```

- physicalDevice **must** be one of the device handles returned from a call to vkEnumeratePhysicalDevices (see Physical Device Enumeration).
- pCreateInfo is a pointer to a VkDeviceCreateInfo structure containing information about how to create the device.

- `pAllocator` controls host memory allocation as described in the Memory Allocation chapter.
- `pDevice` is a pointer to a handle in which the created VkDevice is returned.

`vkCreateDevice` verifies that extensions and features requested in the `ppEnabledExtensionNames` and `pEnabledFeatures` members of `pCreateInfo`, respectively, are supported by the implementation. If any requested extension is not supported, `vkCreateDevice` **must** return `VK_ERROR_EXTENSION_NOT_PRESENT`. If any requested feature is not supported, `vkCreateDevice` **must** return `VK_ERROR_FEATURE_NOT_PRESENT`. Support for extensions **can** be checked before creating a device by querying vkEnumerateDeviceExtensionProperties. Support for features **can** similarly be checked by querying vkGetPhysicalDeviceFeatures.

After verifying and enabling the extensions the `VkDevice` object is created and returned to the application.

Multiple logical devices **can** be created from the same physical device. Logical device creation **may** fail due to lack of device-specific resources (in addition to other errors). If that occurs, `vkCreateDevice` will return `VK_ERROR_TOO_MANY_OBJECTS`.

## Valid Usage

- VUID-vkCreateDevice-ppEnabledExtensionNames-01387
  All required device extensions for each extension in the VkDeviceCreateInfo ::`ppEnabledExtensionNames` list **must** also be present in that list

## Valid Usage (Implicit)

- VUID-vkCreateDevice-physicalDevice-parameter
  `physicalDevice` **must** be a valid VkPhysicalDevice handle
- VUID-vkCreateDevice-pCreateInfo-parameter
  `pCreateInfo` **must** be a valid pointer to a valid VkDeviceCreateInfo structure
- VUID-vkCreateDevice-pAllocator-parameter
  If `pAllocator` is not `NULL`, `pAllocator` **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkCreateDevice-pDevice-parameter
  `pDevice` **must** be a valid pointer to a VkDevice handle

## Return Codes

### Success

- `VK_SUCCESS`

### Failure

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`

- VK_ERROR_INITIALIZATION_FAILED
- VK_ERROR_EXTENSION_NOT_PRESENT
- VK_ERROR_FEATURE_NOT_PRESENT
- VK_ERROR_TOO_MANY_OBJECTS
- VK_ERROR_DEVICE_LOST

The VkDeviceCreateInfo structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkDeviceCreateInfo {
    VkStructureType                    sType;
    const void*                        pNext;
    VkDeviceCreateFlags                flags;
    uint32_t                           queueCreateInfoCount;
    const VkDeviceQueueCreateInfo*     pQueueCreateInfos;
    // enabledLayerCount is deprecated and should not be used
    uint32_t                           enabledLayerCount;
    // ppEnabledLayerNames is deprecated and should not be used
    const char* const*                 ppEnabledLayerNames;
    uint32_t                           enabledExtensionCount;
    const char* const*                 ppEnabledExtensionNames;
    const VkPhysicalDeviceFeatures*    pEnabledFeatures;
} VkDeviceCreateInfo;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- flags is reserved for future use.
- queueCreateInfoCount is the unsigned integer size of the pQueueCreateInfos array. Refer to the Queue Creation section below for further details.
- pQueueCreateInfos is a pointer to an array of VkDeviceQueueCreateInfo structures describing the queues that are requested to be created along with the logical device. Refer to the Queue Creation section below for further details.
- enabledLayerCount is deprecated and ignored.
- ppEnabledLayerNames is deprecated and ignored. See Device Layer Deprecation.
- enabledExtensionCount is the number of device extensions to enable.
- ppEnabledExtensionNames is a pointer to an array of enabledExtensionCount null-terminated UTF-8 strings containing the names of extensions to enable for the created device. See the Extensions section for further details.
- pEnabledFeatures is NULL or a pointer to a VkPhysicalDeviceFeatures structure containing boolean indicators of all the features to be enabled. Refer to the Features section for further details.

## Valid Usage

- VUID-VkDeviceCreateInfo-queueFamilyIndex-02802
  The `queueFamilyIndex` member of each element of `pQueueCreateInfos` **must** be unique within `pQueueCreateInfos` , except that two members can share the same `queueFamilyIndex` if one describes protected-capable queues and one describes queues that are not protected-capable
- VUID-VkDeviceCreateInfo-pQueueCreateInfos-06755
  If multiple elements of `pQueueCreateInfos` share the same `queueFamilyIndex`, the sum of their `queueCount` members **must** be less than or equal to the `queueCount` member of the `VkQueueFamilyProperties` structure, as returned by `vkGetPhysicalDeviceQueueFamilyProperties` in the `pQueueFamilyProperties`[queueFamilyIndex]
- VUID-VkDeviceCreateInfo-pQueueCreateInfos-06654
  If multiple elements of `pQueueCreateInfos` share the same `queueFamilyIndex`, then all of such elements **must** have the same global priority level, which **can** be specified explicitly by the including a VkDeviceQueueGlobalPriorityCreateInfoKHR structure in the `pNext` chain, or by the implicit default value
- VUID-VkDeviceCreateInfo-pNext-00373
  If the `pNext` chain includes a VkPhysicalDeviceFeatures2 structure, then `pEnabledFeatures` **must** be `NULL`
- VUID-VkDeviceCreateInfo-ppEnabledExtensionNames-01840
  If VkPhysicalDeviceProperties::`apiVersion` advertises Vulkan 1.1 or later, `ppEnabledExtensionNames` **must** not contain `VK_AMD_negative_viewport_height`
- VUID-VkDeviceCreateInfo-ppEnabledExtensionNames-00374
  `ppEnabledExtensionNames` **must** not contain both `VK_KHR_maintenance1` and `VK_AMD_negative_viewport_height`
- VUID-VkDeviceCreateInfo-ppEnabledExtensionNames-03328
  `ppEnabledExtensionNames` **must** not contain both `VK_KHR_buffer_device_address` and `VK_EXT_buffer_device_address`
- VUID-VkDeviceCreateInfo-pProperties-04451
  If the `VK_KHR_portability_subset` extension is included in `pProperties` of vkEnumerateDeviceExtensionProperties, `ppEnabledExtensionNames` **must** include `"VK_KHR_portability_subset"`
- VUID-VkDeviceCreateInfo-shadingRateImage-04478
  If the `shadingRateImage` feature is enabled, the `pipelineFragmentShadingRate` feature **must** not be enabled
- VUID-VkDeviceCreateInfo-shadingRateImage-04479
  If the `shadingRateImage` feature is enabled, the `primitiveFragmentShadingRate` feature **must** not be enabled
- VUID-VkDeviceCreateInfo-shadingRateImage-04480
  If the `shadingRateImage` feature is enabled, the `attachmentFragmentShadingRate` feature **must** not be enabled

- VUID-VkDeviceCreateInfo-fragmentDensityMap-04481
  If the `fragmentDensityMap` feature is enabled, the `pipelineFragmentShadingRate` feature **must** not be enabled
- VUID-VkDeviceCreateInfo-fragmentDensityMap-04482
  If the `fragmentDensityMap` feature is enabled, the `primitiveFragmentShadingRate` feature **must** not be enabled
- VUID-VkDeviceCreateInfo-fragmentDensityMap-04483
  If the `fragmentDensityMap` feature is enabled, the `attachmentFragmentShadingRate` feature **must** not be enabled
- VUID-VkDeviceCreateInfo-None-04896
  If `sparseImageInt64Atomics` is enabled, `shaderImageInt64Atomics` **must** be enabled
- VUID-VkDeviceCreateInfo-None-04897
  If `sparseImageFloat32Atomics` is enabled, `shaderImageFloat32Atomics` **must** be enabled
- VUID-VkDeviceCreateInfo-None-04898
  If `sparseImageFloat32AtomicAdd` is enabled, `shaderImageFloat32AtomicAdd` **must** be enabled
- VUID-VkDeviceCreateInfo-sparseImageFloat32AtomicMinMax-04975
  If `sparseImageFloat32AtomicMinMax` is enabled, `shaderImageFloat32AtomicMinMax` **must** be enabled
- VUID-VkDeviceCreateInfo-None-08095
  If `descriptorBuffer` is enabled, `ppEnabledExtensionNames` **must** not contain `VK_AMD_shader_fragment_mask`
- VUID-VkDeviceCreateInfo-pNext-09396
  If the `pNext` chain includes a VkDeviceQueueShaderCoreControlCreateInfoARM structure, then it **must** not be included in the `pNext` chain of any of the VkDeviceQueueCreateInfo structures in `pQueueCreateInfos`
- VUID-VkDeviceCreateInfo-pNext-09397
  If the `pNext` chain includes a VkDeviceQueueShaderCoreControlCreateInfoARM structure then VkPhysicalDeviceSchedulingControlsPropertiesARM::`schedulingControlsFlags` **must** contain `VK_PHYSICAL_DEVICE_SCHEDULING_CONTROLS_SHADER_CORE_COUNT_ARM`

## Valid Usage (Implicit)

- VUID-VkDeviceCreateInfo-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_DEVICE_CREATE_INFO`
- VUID-VkDeviceCreateInfo-pNext-pNext
  Each `pNext` member of any structure (including this one) in the `pNext` chain **must** be either `NULL` or a pointer to a valid instance of VkDeviceDeviceMemoryReportCreateInfoEXT, VkDeviceDiagnosticsConfigCreateInfoNV, VkDeviceGroupDeviceCreateInfo, VkDeviceMemoryOverallocationCreateInfoAMD, VkDevicePipelineBinaryInternalCacheControlKHR, VkDevicePrivateDataCreateInfo, VkDeviceQueueShaderCoreControlCreateInfoARM, VkPhysicalDevice16BitStorageFeatures, VkPhysicalDevice4444FormatsFeaturesEXT, VkPhysicalDevice8BitStorageFeatures, VkPhysicalDeviceASTCDecodeFeaturesEXT,

VkPhysicalDeviceAccelerationStructureFeaturesKHR,
VkPhysicalDeviceAddressBindingReportFeaturesEXT,
VkPhysicalDeviceAmigoProfilingFeaturesSEC, VkPhysicalDeviceAntiLagFeaturesAMD,
VkPhysicalDeviceAttachmentFeedbackLoopDynamicStateFeaturesEXT,
VkPhysicalDeviceAttachmentFeedbackLoopLayoutFeaturesEXT,
VkPhysicalDeviceBlendOperationAdvancedFeaturesEXT,
VkPhysicalDeviceBorderColorSwizzleFeaturesEXT,
VkPhysicalDeviceBufferDeviceAddressFeatures,
VkPhysicalDeviceBufferDeviceAddressFeaturesEXT,
VkPhysicalDeviceClusterCullingShaderFeaturesHUAWEI,
VkPhysicalDeviceCoherentMemoryFeaturesAMD,
VkPhysicalDeviceColorWriteEnableFeaturesEXT,
VkPhysicalDeviceCommandBufferInheritanceFeaturesNV,
VkPhysicalDeviceComputeShaderDerivativesFeaturesKHR,
VkPhysicalDeviceConditionalRenderingFeaturesEXT,
VkPhysicalDeviceCooperativeMatrixFeaturesKHR,
VkPhysicalDeviceCooperativeMatrixFeaturesNV,
VkPhysicalDeviceCopyMemoryIndirectFeaturesNV,
VkPhysicalDeviceCornerSampledImageFeaturesNV,
VkPhysicalDeviceCoverageReductionModeFeaturesNV,
VkPhysicalDeviceCubicClampFeaturesQCOM,
VkPhysicalDeviceCubicWeightsFeaturesQCOM,
VkPhysicalDeviceCudaKernelLaunchFeaturesNV,
VkPhysicalDeviceCustomBorderColorFeaturesEXT,
VkPhysicalDeviceDedicatedAllocationImageAliasingFeaturesNV,
VkPhysicalDeviceDepthBiasControlFeaturesEXT,
VkPhysicalDeviceDepthClampZeroOneFeaturesEXT,
VkPhysicalDeviceDepthClipControlFeaturesEXT,
VkPhysicalDeviceDepthClipEnableFeaturesEXT,
VkPhysicalDeviceDescriptorBufferFeaturesEXT,
VkPhysicalDeviceDescriptorIndexingFeatures,
VkPhysicalDeviceDescriptorPoolOverallocationFeaturesNV,
VkPhysicalDeviceDescriptorSetHostMappingFeaturesVALVE,
VkPhysicalDeviceDeviceGeneratedCommandsComputeFeaturesNV,
VkPhysicalDeviceDeviceGeneratedCommandsFeaturesNV,
VkPhysicalDeviceDeviceMemoryReportFeaturesEXT,
VkPhysicalDeviceDiagnosticsConfigFeaturesNV,
VkPhysicalDeviceDisplacementMicromapFeaturesNV,
VkPhysicalDeviceDynamicRenderingFeatures,
VkPhysicalDeviceDynamicRenderingLocalReadFeaturesKHR,
VkPhysicalDeviceDynamicRenderingUnusedAttachmentsFeaturesEXT,
VkPhysicalDeviceExclusiveScissorFeaturesNV,
VkPhysicalDeviceExtendedDynamicState2FeaturesEXT,
VkPhysicalDeviceExtendedDynamicState3FeaturesEXT,
VkPhysicalDeviceExtendedDynamicStateFeaturesEXT,
VkPhysicalDeviceExtendedSparseAddressSpaceFeaturesNV,
VkPhysicalDeviceExternalFormatResolveFeaturesANDROID,
VkPhysicalDeviceExternalMemoryRDMAFeaturesNV,

VkPhysicalDeviceExternalMemoryScreenBufferFeaturesQNX,
VkPhysicalDeviceFaultFeaturesEXT, VkPhysicalDeviceFeatures2,
VkPhysicalDeviceFragmentDensityMap2FeaturesEXT,
VkPhysicalDeviceFragmentDensityMapFeaturesEXT,
VkPhysicalDeviceFragmentDensityMapOffsetFeaturesQCOM,
VkPhysicalDeviceFragmentShaderBarycentricFeaturesKHR,
VkPhysicalDeviceFragmentShaderInterlockFeaturesEXT,
VkPhysicalDeviceFragmentShadingRateEnumsFeaturesNV,
VkPhysicalDeviceFragmentShadingRateFeaturesKHR,
VkPhysicalDeviceFrameBoundaryFeaturesEXT,
VkPhysicalDeviceGlobalPriorityQueryFeaturesKHR,
VkPhysicalDeviceGraphicsPipelineLibraryFeaturesEXT,
VkPhysicalDeviceHostImageCopyFeaturesEXT, VkPhysicalDeviceHostQueryResetFeatures,
VkPhysicalDeviceImage2DViewOf3DFeaturesEXT,
VkPhysicalDeviceImageAlignmentControlFeaturesMESA,
VkPhysicalDeviceImageCompressionControlFeaturesEXT,
VkPhysicalDeviceImageCompressionControlSwapchainFeaturesEXT,
VkPhysicalDeviceImageProcessing2FeaturesQCOM,
VkPhysicalDeviceImageProcessingFeaturesQCOM,
VkPhysicalDeviceImageRobustnessFeatures,
VkPhysicalDeviceImageSlicedViewOf3DFeaturesEXT,
VkPhysicalDeviceImageViewMinLodFeaturesEXT,
VkPhysicalDeviceImagelessFramebufferFeatures,
VkPhysicalDeviceIndexTypeUint8FeaturesKHR,
VkPhysicalDeviceInheritedViewportScissorFeaturesNV,
VkPhysicalDeviceInlineUniformBlockFeatures,
VkPhysicalDeviceInvocationMaskFeaturesHUAWEI,
VkPhysicalDeviceLegacyDitheringFeaturesEXT,
VkPhysicalDeviceLegacyVertexAttributesFeaturesEXT,
VkPhysicalDeviceLineRasterizationFeaturesKHR,
VkPhysicalDeviceLinearColorAttachmentFeaturesNV,
VkPhysicalDeviceMaintenance4Features, VkPhysicalDeviceMaintenance5FeaturesKHR,
VkPhysicalDeviceMaintenance6FeaturesKHR,
VkPhysicalDeviceMaintenance7FeaturesKHR,
VkPhysicalDeviceMapMemoryPlacedFeaturesEXT,
VkPhysicalDeviceMemoryDecompressionFeaturesNV,
VkPhysicalDeviceMemoryPriorityFeaturesEXT,
VkPhysicalDeviceMeshShaderFeaturesEXT, VkPhysicalDeviceMeshShaderFeaturesNV,
VkPhysicalDeviceMultiDrawFeaturesEXT,
VkPhysicalDeviceMultisampledRenderToSingleSampledFeaturesEXT,
VkPhysicalDeviceMultiviewFeatures,
VkPhysicalDeviceMultiviewPerViewRenderAreasFeaturesQCOM,
VkPhysicalDeviceMultiviewPerViewViewportsFeaturesQCOM,
VkPhysicalDeviceMutableDescriptorTypeFeaturesEXT,
VkPhysicalDeviceNestedCommandBufferFeaturesEXT,
VkPhysicalDeviceNonSeamlessCubeMapFeaturesEXT,
VkPhysicalDeviceOpacityMicromapFeaturesEXT,
VkPhysicalDeviceOpticalFlowFeaturesNV,

VkPhysicalDevicePageableDeviceLocalMemoryFeaturesEXT,
VkPhysicalDevicePerStageDescriptorSetFeaturesNV,
VkPhysicalDevicePerformanceQueryFeaturesKHR,
VkPhysicalDevicePipelineBinaryFeaturesKHR,
VkPhysicalDevicePipelineCreationCacheControlFeatures,
VkPhysicalDevicePipelineExecutablePropertiesFeaturesKHR,
VkPhysicalDevicePipelineLibraryGroupHandlesFeaturesEXT,
VkPhysicalDevicePipelinePropertiesFeaturesEXT,
VkPhysicalDevicePipelineProtectedAccessFeaturesEXT,
VkPhysicalDevicePipelineRobustnessFeaturesEXT,
VkPhysicalDevicePortabilitySubsetFeaturesKHR,
VkPhysicalDevicePresentBarrierFeaturesNV, VkPhysicalDevicePresentIdFeaturesKHR,
VkPhysicalDevicePresentWaitFeaturesKHR,
VkPhysicalDevicePrimitiveTopologyListRestartFeaturesEXT,
VkPhysicalDevicePrimitivesGeneratedQueryFeaturesEXT,
VkPhysicalDevicePrivateDataFeatures, VkPhysicalDeviceProtectedMemoryFeatures,
VkPhysicalDeviceProvokingVertexFeaturesEXT,
VkPhysicalDeviceRGBA10X6FormatsFeaturesEXT,
VkPhysicalDeviceRasterizationOrderAttachmentAccessFeaturesEXT,
VkPhysicalDeviceRawAccessChainsFeaturesNV, VkPhysicalDeviceRayQueryFeaturesKHR,
VkPhysicalDeviceRayTracingInvocationReorderFeaturesNV,
VkPhysicalDeviceRayTracingMaintenance1FeaturesKHR,
VkPhysicalDeviceRayTracingMotionBlurFeaturesNV,
VkPhysicalDeviceRayTracingPipelineFeaturesKHR,
VkPhysicalDeviceRayTracingPositionFetchFeaturesKHR,
VkPhysicalDeviceRayTracingValidationFeaturesNV,
VkPhysicalDeviceRelaxedLineRasterizationFeaturesIMG,
VkPhysicalDeviceRenderPassStripedFeaturesARM,
VkPhysicalDeviceRepresentativeFragmentTestFeaturesNV,
VkPhysicalDeviceRobustness2FeaturesEXT,
VkPhysicalDeviceSamplerYcbcrConversionFeatures,
VkPhysicalDeviceScalarBlockLayoutFeatures,
VkPhysicalDeviceSchedulingControlsFeaturesARM,
VkPhysicalDeviceSeparateDepthStencilLayoutsFeatures,
VkPhysicalDeviceShaderAtomicFloat16VectorFeaturesNV,
VkPhysicalDeviceShaderAtomicFloat2FeaturesEXT,
VkPhysicalDeviceShaderAtomicFloatFeaturesEXT,
VkPhysicalDeviceShaderAtomicInt64Features,
VkPhysicalDeviceShaderClockFeaturesKHR,
VkPhysicalDeviceShaderCoreBuiltinsFeaturesARM,
VkPhysicalDeviceShaderDemoteToHelperInvocationFeatures,
VkPhysicalDeviceShaderDrawParametersFeatures,
VkPhysicalDeviceShaderEarlyAndLateFragmentTestsFeaturesAMD,
VkPhysicalDeviceShaderEnqueueFeaturesAMDX,
VkPhysicalDeviceShaderExpectAssumeFeaturesKHR,
VkPhysicalDeviceShaderFloat16Int8Features,
VkPhysicalDeviceShaderFloatControls2FeaturesKHR,
VkPhysicalDeviceShaderImageAtomicInt64FeaturesEXT,

VkPhysicalDeviceShaderImageFootprintFeaturesNV, VkPhysicalDeviceShaderIntegerDotProductFeatures, VkPhysicalDeviceShaderIntegerFunctions2FeaturesINTEL, VkPhysicalDeviceShaderMaximalReconvergenceFeaturesKHR, VkPhysicalDeviceShaderModuleIdentifierFeaturesEXT, VkPhysicalDeviceShaderObjectFeaturesEXT, VkPhysicalDeviceShaderQuadControlFeaturesKHR, VkPhysicalDeviceShaderRelaxedExtendedInstructionFeaturesKHR, VkPhysicalDeviceShaderReplicatedCompositesFeaturesEXT, VkPhysicalDeviceShaderSMBuiltinsFeaturesNV, VkPhysicalDeviceShaderSubgroupExtendedTypesFeatures, VkPhysicalDeviceShaderSubgroupRotateFeaturesKHR, VkPhysicalDeviceShaderSubgroupUniformControlFlowFeaturesKHR, VkPhysicalDeviceShaderTerminateInvocationFeatures, VkPhysicalDeviceShaderTileImageFeaturesEXT, VkPhysicalDeviceShadingRateImageFeaturesNV, VkPhysicalDeviceSubgroupSizeControlFeatures, VkPhysicalDeviceSubpassMergeFeedbackFeaturesEXT, VkPhysicalDeviceSubpassShadingFeaturesHUAWEI, VkPhysicalDeviceSwapchainMaintenance1FeaturesEXT, VkPhysicalDeviceSynchronization2Features, VkPhysicalDeviceTexelBufferAlignmentFeaturesEXT, VkPhysicalDeviceTextureCompressionASTCHDRFeatures, VkPhysicalDeviceTilePropertiesFeaturesQCOM, VkPhysicalDeviceTimelineSemaphoreFeatures, VkPhysicalDeviceTransformFeedbackFeaturesEXT, VkPhysicalDeviceUniformBufferStandardLayoutFeatures, VkPhysicalDeviceVariablePointersFeatures, VkPhysicalDeviceVertexAttributeDivisorFeaturesKHR, VkPhysicalDeviceVertexInputDynamicStateFeaturesEXT, VkPhysicalDeviceVideoMaintenance1FeaturesKHR, VkPhysicalDeviceVulkanMemoryModelFeatures, VkPhysicalDeviceWorkgroupMemoryExplicitLayoutFeaturesKHR, VkPhysicalDeviceYcbcr2Plane444FormatsFeaturesEXT, VkPhysicalDeviceYcbcrDegammaFeaturesQCOM, VkPhysicalDeviceYcbcrImageArraysFeaturesEXT, or VkPhysicalDeviceZeroInitializeWorkgroupMemoryFeatures

- VUID-VkDeviceCreateInfo-sType-unique
  The `sType` value of each struct in the `pNext` chain **must** be unique, with the exception of structures of type VkDeviceDeviceMemoryReportCreateInfoEXT or VkDevicePrivateDataCreateInfo
- VUID-VkDeviceCreateInfo-flags-zerobitmask
  `flags` **must** be `0`
- VUID-VkDeviceCreateInfo-pQueueCreateInfos-parameter
  `pQueueCreateInfos` **must** be a valid pointer to an array of `queueCreateInfoCount` valid VkDeviceQueueCreateInfo structures

- VUID-VkDeviceCreateInfo-ppEnabledLayerNames-parameter
  If `enabledLayerCount` is not `0`, `ppEnabledLayerNames` **must** be a valid pointer to an array of `enabledLayerCount` null-terminated UTF-8 strings
- VUID-VkDeviceCreateInfo-ppEnabledExtensionNames-parameter
  If `enabledExtensionCount` is not `0`, `ppEnabledExtensionNames` **must** be a valid pointer to an array of `enabledExtensionCount` null-terminated UTF-8 strings
- VUID-VkDeviceCreateInfo-pEnabledFeatures-parameter
  If `pEnabledFeatures` is not `NULL`, `pEnabledFeatures` **must** be a valid pointer to a valid VkPhysicalDeviceFeatures structure
- VUID-VkDeviceCreateInfo-queueCreateInfoCount-arraylength
  `queueCreateInfoCount` **must** be greater than `0`

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkDeviceCreateFlags;
```

`VkDeviceCreateFlags` is a bitmask type for setting a mask, but is currently reserved for future use.

A logical device **can** be created that connects to one or more physical devices by adding a `VkDeviceGroupDeviceCreateInfo` structure to the `pNext` chain of VkDeviceCreateInfo. The `VkDeviceGroupDeviceCreateInfo` structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkDeviceGroupDeviceCreateInfo {
    VkStructureType            sType;
    const void*                pNext;
    uint32_t                   physicalDeviceCount;
    const VkPhysicalDevice*    pPhysicalDevices;
} VkDeviceGroupDeviceCreateInfo;
```

or the equivalent

```
// Provided by VK_KHR_device_group_creation
typedef VkDeviceGroupDeviceCreateInfo VkDeviceGroupDeviceCreateInfoKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `physicalDeviceCount` is the number of elements in the `pPhysicalDevices` array.
- `pPhysicalDevices` is a pointer to an array of physical device handles belonging to the same device group.

The elements of the `pPhysicalDevices` array are an ordered list of the physical devices that the logical device represents. These **must** be a subset of a single device group, and need not be in the same order as they were enumerated. The order of the physical devices in the `pPhysicalDevices`

array determines the *device index* of each physical device, with element i being assigned a device index of i. Certain commands and structures refer to one or more physical devices by using device indices or *device masks* formed using device indices.

A logical device created without using `VkDeviceGroupDeviceCreateInfo`, or with `physicalDeviceCount` equal to zero, is equivalent to a `physicalDeviceCount` of one and `pPhysicalDevices` pointing to the `physicalDevice` parameter to vkCreateDevice. In particular, the device index of that physical device is zero.

### Valid Usage

- VUID-VkDeviceGroupDeviceCreateInfo-pPhysicalDevices-00375
  Each element of `pPhysicalDevices` **must** be unique
- VUID-VkDeviceGroupDeviceCreateInfo-pPhysicalDevices-00376
  All elements of `pPhysicalDevices` **must** be in the same device group as enumerated by vkEnumeratePhysicalDeviceGroups
- VUID-VkDeviceGroupDeviceCreateInfo-physicalDeviceCount-00377
  If `physicalDeviceCount` is not `0`, the `physicalDevice` parameter of vkCreateDevice **must** be an element of `pPhysicalDevices`

### Valid Usage (Implicit)

- VUID-VkDeviceGroupDeviceCreateInfo-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_DEVICE_GROUP_DEVICE_CREATE_INFO`
- VUID-VkDeviceGroupDeviceCreateInfo-pPhysicalDevices-parameter
  If `physicalDeviceCount` is not `0`, `pPhysicalDevices` **must** be a valid pointer to an array of `physicalDeviceCount` valid VkPhysicalDevice handles

To specify whether device memory allocation is allowed beyond the size reported by VkPhysicalDeviceMemoryProperties, add a VkDeviceMemoryOverallocationCreateInfoAMD structure to the `pNext` chain of the VkDeviceCreateInfo structure. If this structure is not specified, it is as if the `VK_MEMORY_OVERALLOCATION_BEHAVIOR_DEFAULT_AMD` value is used.

```
// Provided by VK_AMD_memory_overallocation_behavior
typedef struct VkDeviceMemoryOverallocationCreateInfoAMD {
    VkStructureType                     sType;
    const void*                         pNext;
    VkMemoryOverallocationBehaviorAMD   overallocationBehavior;
} VkDeviceMemoryOverallocationCreateInfoAMD;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `overallocationBehavior` is the desired overallocation behavior.

### Valid Usage (Implicit)

- VUID-VkDeviceMemoryOverallocationCreateInfoAMD-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_DEVICE_MEMORY_OVERALLOCATION_CREATE_INFO_AMD`
- VUID-VkDeviceMemoryOverallocationCreateInfoAMD-overallocationBehavior-parameter
  `overallocationBehavior` **must** be a valid VkMemoryOverallocationBehaviorAMD value

Possible values for VkDeviceMemoryOverallocationCreateInfoAMD::`overallocationBehavior` include:

```
// Provided by VK_AMD_memory_overallocation_behavior
typedef enum VkMemoryOverallocationBehaviorAMD {
    VK_MEMORY_OVERALLOCATION_BEHAVIOR_DEFAULT_AMD = 0,
    VK_MEMORY_OVERALLOCATION_BEHAVIOR_ALLOWED_AMD = 1,
    VK_MEMORY_OVERALLOCATION_BEHAVIOR_DISALLOWED_AMD = 2,
} VkMemoryOverallocationBehaviorAMD;
```

- `VK_MEMORY_OVERALLOCATION_BEHAVIOR_DEFAULT_AMD` lets the implementation decide if overallocation is allowed.
- `VK_MEMORY_OVERALLOCATION_BEHAVIOR_ALLOWED_AMD` specifies overallocation is allowed if platform permits.
- `VK_MEMORY_OVERALLOCATION_BEHAVIOR_DISALLOWED_AMD` specifies the application is not allowed to allocate device memory beyond the heap sizes reported by VkPhysicalDeviceMemoryProperties. Allocations that are not explicitly made by the application within the scope of the Vulkan instance are not accounted for.

When using the Nsight™ Aftermath SDK, to configure how device crash dumps are created, add a VkDeviceDiagnosticsConfigCreateInfoNV structure to the `pNext` chain of the VkDeviceCreateInfo structure.

```
// Provided by VK_NV_device_diagnostics_config
typedef struct VkDeviceDiagnosticsConfigCreateInfoNV {
    VkStructureType                     sType;
    const void*                         pNext;
    VkDeviceDiagnosticsConfigFlagsNV    flags;
} VkDeviceDiagnosticsConfigCreateInfoNV;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is a bitmask of VkDeviceDiagnosticsConfigFlagBitsNV specifying additional parameters for configuring diagnostic tools.

### Valid Usage (Implicit)

- VUID-VkDeviceDiagnosticsConfigCreateInfoNV-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_DEVICE_DIAGNOSTICS_CONFIG_CREATE_INFO_NV`
- VUID-VkDeviceDiagnosticsConfigCreateInfoNV-flags-parameter
  `flags` **must** be a valid combination of VkDeviceDiagnosticsConfigFlagBitsNV values

Bits which **can** be set in VkDeviceDiagnosticsConfigCreateInfoNV::`flags` include:

```
// Provided by VK_NV_device_diagnostics_config
typedef enum VkDeviceDiagnosticsConfigFlagBitsNV {
    VK_DEVICE_DIAGNOSTICS_CONFIG_ENABLE_SHADER_DEBUG_INFO_BIT_NV = 0x00000001,
    VK_DEVICE_DIAGNOSTICS_CONFIG_ENABLE_RESOURCE_TRACKING_BIT_NV = 0x00000002,
    VK_DEVICE_DIAGNOSTICS_CONFIG_ENABLE_AUTOMATIC_CHECKPOINTS_BIT_NV = 0x00000004,
    VK_DEVICE_DIAGNOSTICS_CONFIG_ENABLE_SHADER_ERROR_REPORTING_BIT_NV = 0x00000008,
} VkDeviceDiagnosticsConfigFlagBitsNV;
```

- `VK_DEVICE_DIAGNOSTICS_CONFIG_ENABLE_SHADER_DEBUG_INFO_BIT_NV` enables the generation of debug information for shaders.
- `VK_DEVICE_DIAGNOSTICS_CONFIG_ENABLE_RESOURCE_TRACKING_BIT_NV` enables driver side tracking of resources (images, buffers, etc.) used to augment the device fault information.
- `VK_DEVICE_DIAGNOSTICS_CONFIG_ENABLE_AUTOMATIC_CHECKPOINTS_BIT_NV` enables automatic insertion of diagnostic checkpoints for draw calls, dispatches, trace rays, and copies. The CPU call stack at the time of the command will be associated as the marker data for the automatically inserted checkpoints.
- `VK_DEVICE_DIAGNOSTICS_CONFIG_ENABLE_SHADER_ERROR_REPORTING_BIT_NV` enables shader error reporting.

```
// Provided by VK_NV_device_diagnostics_config
typedef VkFlags VkDeviceDiagnosticsConfigFlagsNV;
```

`VkDeviceDiagnosticsConfigFlagsNV` is a bitmask type for setting a mask of zero or more VkDeviceDiagnosticsConfigFlagBitsNV.

To register callbacks for underlying device memory events of type VkDeviceMemoryReportEventTypeEXT, add one or multiple VkDeviceDeviceMemoryReportCreateInfoEXT structures to the `pNext` chain of the VkDeviceCreateInfo structure.

```
// Provided by VK_EXT_device_memory_report
typedef struct VkDeviceDeviceMemoryReportCreateInfoEXT {
    VkStructureType                          sType;
    const void*                              pNext;
    VkDeviceMemoryReportFlagsEXT             flags;
    PFN_vkDeviceMemoryReportCallbackEXT      pfnUserCallback;
    void*                                    pUserData;
} VkDeviceDeviceMemoryReportCreateInfoEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is 0 and reserved for future use.
- `pfnUserCallback` is the application callback function to call.
- `pUserData` is user data to be passed to the callback.

The callback **may** be called from multiple threads simultaneously.

The callback **must** be called only once by the implementation when a VkDeviceMemoryReportEventTypeEXT event occurs.

| | |
|---|---|
| **NOTE** | The callback could be called from a background thread other than the thread calling the Vulkan commands. |

### Valid Usage (Implicit)

- VUID-VkDeviceDeviceMemoryReportCreateInfoEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_DEVICE_DEVICE_MEMORY_REPORT_CREATE_INFO_EXT`
- VUID-VkDeviceDeviceMemoryReportCreateInfoEXT-flags-zerobitmask
  `flags` **must** be `0`
- VUID-VkDeviceDeviceMemoryReportCreateInfoEXT-pfnUserCallback-parameter
  `pfnUserCallback` **must** be a valid PFN_vkDeviceMemoryReportCallbackEXT value
- VUID-VkDeviceDeviceMemoryReportCreateInfoEXT-pUserData-parameter
  `pUserData` **must** be a pointer value

The prototype for the VkDeviceDeviceMemoryReportCreateInfoEXT::`pfnUserCallback` function implemented by the application is:

```
// Provided by VK_EXT_device_memory_report
typedef void (VKAPI_PTR *PFN_vkDeviceMemoryReportCallbackEXT)(
    const VkDeviceMemoryReportCallbackDataEXT*  pCallbackData,
    void*                                       pUserData);
```

- `pCallbackData` contains all the callback related data in the

VkDeviceMemoryReportCallbackDataEXT structure.

- pUserData is the user data provided when the VkDeviceDeviceMemoryReportCreateInfoEXT was created.

The callback **must** not make calls to any Vulkan commands.

The definition of VkDeviceMemoryReportCallbackDataEXT is:

```
// Provided by VK_EXT_device_memory_report
typedef struct VkDeviceMemoryReportCallbackDataEXT {
    VkStructureType                     sType;
    void*                               pNext;
    VkDeviceMemoryReportFlagsEXT        flags;
    VkDeviceMemoryReportEventTypeEXT    type;
    uint64_t                            memoryObjectId;
    VkDeviceSize                        size;
    VkObjectType                        objectType;
    uint64_t                            objectHandle;
    uint32_t                            heapIndex;
} VkDeviceMemoryReportCallbackDataEXT;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- flags is 0 and reserved for future use.
- type is a VkDeviceMemoryReportEventTypeEXT type specifying the type of event reported in this VkDeviceMemoryReportCallbackDataEXT structure.
- memoryObjectId is the unique id for the underlying memory object as described below.
- size is the size of the memory object in bytes. If type is VK_DEVICE_MEMORY_REPORT_EVENT_TYPE_ALLOCATE_EXT, VK_DEVICE_MEMORY_REPORT_EVENT_TYPE_IMPORT_EXT or VK_DEVICE_MEMORY_REPORT_EVENT_TYPE_ALLOCATION_FAILED_EXT, size is a valid VkDeviceSize value. Otherwise, size is undefined.
- objectType is a VkObjectType value specifying the type of the object associated with this device memory report event. If type is VK_DEVICE_MEMORY_REPORT_EVENT_TYPE_ALLOCATE_EXT, VK_DEVICE_MEMORY_REPORT_EVENT_TYPE_FREE_EXT, VK_DEVICE_MEMORY_REPORT_EVENT_TYPE_IMPORT_EXT, VK_DEVICE_MEMORY_REPORT_EVENT_TYPE_UNIMPORT_EXT or VK_DEVICE_MEMORY_REPORT_EVENT_TYPE_ALLOCATION_FAILED_EXT, objectType is a valid VkObjectType enum. Otherwise, objectType is undefined.
- objectHandle is the object this device memory report event is attributed to. If type is VK_DEVICE_MEMORY_REPORT_EVENT_TYPE_ALLOCATE_EXT, VK_DEVICE_MEMORY_REPORT_EVENT_TYPE_FREE_EXT, VK_DEVICE_MEMORY_REPORT_EVENT_TYPE_IMPORT_EXT or VK_DEVICE_MEMORY_REPORT_EVENT_TYPE_UNIMPORT_EXT, objectHandle is a valid Vulkan handle of the type associated with objectType as defined in the VkObjectType and Vulkan Handle Relationship table. Otherwise, objectHandle is undefined.

- heapIndex describes which memory heap this device memory allocation is made from. If type is VK_DEVICE_MEMORY_REPORT_EVENT_TYPE_ALLOCATE_EXT or VK_DEVICE_MEMORY_REPORT_EVENT_TYPE_ALLOCATION_FAILED_EXT, heapIndex corresponds to one of the valid heaps from the VkPhysicalDeviceMemoryProperties structure. Otherwise, heapIndex is undefined.

memoryObjectId is used to avoid double-counting on the same memory object.

If an internally-allocated device memory object or a VkDeviceMemory **cannot** be exported, memoryObjectId **must** be unique in the VkDevice.

If an internally-allocated device memory object or a VkDeviceMemory supports being exported, memoryObjectId **must** be unique system wide.

If an internal device memory object or a VkDeviceMemory is backed by an imported external memory object, memoryObjectId **must** be unique system wide.

### Implementor's Note

If the heap backing an internally-allocated device memory **cannot** be used to back VkDeviceMemory, implementations **can** advertise that heap with no types.

**NOTE**

This structure should only be considered valid during the lifetime of the triggered callback.

For VK_DEVICE_MEMORY_REPORT_EVENT_TYPE_ALLOCATE_EXT and VK_DEVICE_MEMORY_REPORT_EVENT_TYPE_IMPORT_EXT events, objectHandle usually will not yet exist when the application or tool receives the callback. objectHandle will only exist when the create or allocate call that triggered the event returns, and if the allocation or import ends up failing objectHandle will not ever exist.

### Valid Usage (Implicit)

- VUID-VkDeviceMemoryReportCallbackDataEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_DEVICE_MEMORY_REPORT_CALLBACK_DATA_EXT
- VUID-VkDeviceMemoryReportCallbackDataEXT-pNext-pNext
  pNext **must** be NULL

```
// Provided by VK_EXT_device_memory_report
typedef VkFlags VkDeviceMemoryReportFlagsEXT;
```

VkDeviceMemoryReportFlagsEXT is a bitmask type for setting a mask, but is currently reserved for future use.

Possible values of VkDeviceMemoryReportCallbackDataEXT::type, specifying event types which

cause the device driver to call the callback, are:

```
// Provided by VK_EXT_device_memory_report
typedef enum VkDeviceMemoryReportEventTypeEXT {
    VK_DEVICE_MEMORY_REPORT_EVENT_TYPE_ALLOCATE_EXT = 0,
    VK_DEVICE_MEMORY_REPORT_EVENT_TYPE_FREE_EXT = 1,
    VK_DEVICE_MEMORY_REPORT_EVENT_TYPE_IMPORT_EXT = 2,
    VK_DEVICE_MEMORY_REPORT_EVENT_TYPE_UNIMPORT_EXT = 3,
    VK_DEVICE_MEMORY_REPORT_EVENT_TYPE_ALLOCATION_FAILED_EXT = 4,
} VkDeviceMemoryReportEventTypeEXT;
```

- `VK_DEVICE_MEMORY_REPORT_EVENT_TYPE_ALLOCATE_EXT` specifies this event corresponds to the allocation of an internal device memory object or a VkDeviceMemory.
- `VK_DEVICE_MEMORY_REPORT_EVENT_TYPE_FREE_EXT` specifies this event corresponds to the deallocation of an internally-allocated device memory object or a VkDeviceMemory.
- `VK_DEVICE_MEMORY_REPORT_EVENT_TYPE_IMPORT_EXT` specifies this event corresponds to the import of an external memory object.
- `VK_DEVICE_MEMORY_REPORT_EVENT_TYPE_UNIMPORT_EXT` specifies this event is the release of an imported external memory object.
- `VK_DEVICE_MEMORY_REPORT_EVENT_TYPE_ALLOCATION_FAILED_EXT` specifies this event corresponds to the failed allocation of an internal device memory object or a VkDeviceMemory.

To reserve private data storage slots, add a VkDevicePrivateDataCreateInfo structure to the `pNext` chain of the VkDeviceCreateInfo structure. Reserving slots in this manner is not strictly necessary, but doing so **may** improve performance.

```
typedef struct VkDevicePrivateDataCreateInfo {
    VkStructureType    sType;
    const void*        pNext;
    uint32_t           privateDataSlotRequestCount;
} VkDevicePrivateDataCreateInfo;
```

or the equivalent

```
// Provided by VK_EXT_private_data
typedef VkDevicePrivateDataCreateInfo VkDevicePrivateDataCreateInfoEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `privateDataSlotRequestCount` is the amount of slots to reserve.

### Valid Usage (Implicit)

- VUID-VkDevicePrivateDataCreateInfo-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_DEVICE_PRIVATE_DATA_CREATE_INFO`

To disable the implementation's internal pipeline cache, add a VkDevicePipelineBinaryInternalCacheControlKHR structure to the `pNext` chain of the VkDeviceCreateInfo structure.

```
// Provided by VK_KHR_pipeline_binary
typedef struct VkDevicePipelineBinaryInternalCacheControlKHR {
    VkStructureType    sType;
    const void*        pNext;
    VkBool32           disableInternalCache;
} VkDevicePipelineBinaryInternalCacheControlKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `disableInternalCache` specifies whether or not to disable the implementation's internal pipeline cache.

If the `VkDeviceCreateInfo::pNext` chain does not include this structure, then `disableInternalCache` defaults to `VK_FALSE`.

### Valid Usage

- VUID-VkDevicePipelineBinaryInternalCacheControlKHR-disableInternalCache-09602
  If VkPhysicalDevicePipelineBinaryPropertiesKHR::`pipelineBinaryInternalCacheControl` is `VK_FALSE`, `disableInternalCache` **must** be `VK_FALSE`

### Valid Usage (Implicit)

- VUID-VkDevicePipelineBinaryInternalCacheControlKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_DEVICE_PIPELINE_BINARY_INTERNAL_CACHE_CONTROL_KHR`

The number of shader cores used by all the queues of a device **can** be controlled by adding a `VkDeviceQueueShaderCoreControlCreateInfoARM` structure to the `pNext` chain of the VkDeviceCreateInfo structure.

## 5.2.2. Device Use

The following is a high-level list of `VkDevice` uses along with references on where to find more information:

- Creation of queues. See the Queues section below for further details.
- Creation and tracking of various synchronization constructs. See Synchronization and Cache Control for further details.
- Allocating, freeing, and managing memory. See Memory Allocation and Resource Creation for further details.
- Creation and destruction of command buffers and command buffer pools. See Command Buffers for further details.
- Creation, destruction, and management of graphics state. See Pipelines and Resource Descriptors, among others, for further details.

### 5.2.3. Lost Device

A logical device **may** become *lost* for a number of implementation-specific reasons, indicating that pending and future command execution **may** fail and cause resources and backing memory to become undefined.

**NOTE**

Typical reasons for device loss will include things like execution timing out (to prevent denial of service), power management events, platform resource management, implementation errors.

Applications not adhering to valid usage may also result in device loss being reported, however this is not guaranteed. Even if device loss is reported, the system may be in an unrecoverable state, and further usage of the API is still considered invalid.

When this happens, certain commands will return `VK_ERROR_DEVICE_LOST`. After any such event, the logical device is considered *lost*. It is not possible to reset the logical device to a non-lost state, however the lost state is specific to a logical device (`VkDevice`), and the corresponding physical device (`VkPhysicalDevice`) **may** be otherwise unaffected.

In some cases, the physical device **may** also be lost, and attempting to create a new logical device will fail, returning `VK_ERROR_DEVICE_LOST`. This is usually indicative of a problem with the underlying implementation, or its connection to the host. If the physical device has not been lost, and a new logical device is successfully created from that physical device, it **must** be in the non-lost state.

**NOTE**

Whilst logical device loss **may** be recoverable, in the case of physical device loss, it is unlikely that an application will be able to recover unless additional, unaffected physical devices exist on the system. The error is largely informational and intended only to inform the application that a platform issue has occurred, and **should** be investigated further. For example, underlying hardware **may** have developed a fault or become physically disconnected from the rest of the system. In many cases, physical device loss **may** cause other more serious issues such as the operating system crashing; in which case it **may** not be reported via the Vulkan API.

When a device is lost, its child objects are not implicitly destroyed and their handles are still valid. Those objects **must** still be destroyed before their parents or the device **can** be destroyed (see the Object Lifetime section). The host address space corresponding to device memory mapped using

vkMapMemory is still valid, and host memory accesses to these mapped regions are still valid, but the contents are undefined. It is still legal to call any API command on the device and child objects.

Once a device is lost, command execution **may** fail, and certain commands that return a VkResult **may** return VK_ERROR_DEVICE_LOST. These commands can be identified by the inclusion of VK_ERROR_DEVICE_LOST in the Return Codes section for each command. Commands that do not allow runtime errors **must** still operate correctly for valid usage and, if applicable, return valid data.

Commands that wait indefinitely for device execution (namely vkDeviceWaitIdle, vkQueueWaitIdle, vkWaitForFences or vkAcquireNextImageKHR with a maximum timeout, and vkGetQueryPoolResults with the VK_QUERY_RESULT_WAIT_BIT bit set in flags) **must** return in finite time even in the case of a lost device, and return either VK_SUCCESS or VK_ERROR_DEVICE_LOST. For any command that **may** return VK_ERROR_DEVICE_LOST, for the purpose of determining whether a command buffer is in the pending state, or whether resources are considered in-use by the device, a return value of VK_ERROR_DEVICE_LOST is equivalent to VK_SUCCESS.

If a device was created with the maintenance5 feature enabled, and any device command returns VK_ERROR_DEVICE_LOST, then all device commands for which VK_ERROR_DEVICE_LOST is a valid return value and which happen-after it on the same host thread **must** return VK_ERROR_DEVICE_LOST.

Device commands executing on other threads **must** begin returning VK_ERROR_DEVICE_LOST within finite time.

The content of any external memory objects that have been exported from or imported to a lost device become undefined. Objects on other logical devices or in other APIs which are associated with the same underlying memory resource as the external memory objects on the lost device are unaffected other than their content becoming undefined. The layout of subresources of images on other logical devices that are bound to VkDeviceMemory objects associated with the same underlying memory resources as external memory objects on the lost device becomes VK_IMAGE_LAYOUT_UNDEFINED.

The state of VkSemaphore objects on other logical devices created by importing a semaphore payload with temporary permanence which was exported from the lost device is undefined. The state of VkSemaphore objects on other logical devices that permanently share a semaphore payload with a VkSemaphore object on the lost device is undefined, and remains undefined following any subsequent signal operations. Implementations **must** ensure pending and subsequently submitted wait operations on such semaphores behave as defined in Semaphore State Requirements For Wait Operations for external semaphores not in a valid state for a wait operation.

### 5.2.4. Device Destruction

To destroy a device, call:

```
// Provided by VK_VERSION_1_0
void vkDestroyDevice(
    VkDevice                                    device,
    const VkAllocationCallbacks*                pAllocator);
```

- `device` is the logical device to destroy.
- `pAllocator` controls host memory allocation as described in the Memory Allocation chapter.

To ensure that no work is active on the device, vkDeviceWaitIdle **can** be used to gate the destruction of the device. Prior to destroying a device, an application is responsible for destroying/freeing any Vulkan objects that were created using that device as the first parameter of the corresponding `vkCreate*` or `vkAllocate*` command.

| | |
|---|---|
| **NOTE** | The lifetime of each of these objects is bound by the lifetime of the `VkDevice` object. Therefore, to avoid resource leaks, it is critical that an application explicitly free all of these resources prior to calling `vkDestroyDevice`. |

### Valid Usage

- VUID-vkDestroyDevice-device-05137
  All child objects created on `device` **must** have been destroyed prior to destroying `device`
- VUID-vkDestroyDevice-device-00379
  If `VkAllocationCallbacks` were provided when `device` was created, a compatible set of callbacks **must** be provided here
- VUID-vkDestroyDevice-device-00380
  If no `VkAllocationCallbacks` were provided when `device` was created, `pAllocator` **must** be `NULL`

### Valid Usage (Implicit)

- VUID-vkDestroyDevice-device-parameter
  If `device` is not `NULL`, `device` **must** be a valid VkDevice handle
- VUID-vkDestroyDevice-pAllocator-parameter
  If `pAllocator` is not `NULL`, `pAllocator` **must** be a valid pointer to a valid VkAllocationCallbacks structure

### Host Synchronization

- Host access to `device` **must** be externally synchronized
- Host access to all `VkQueue` objects created from `device` **must** be externally synchronized

## 5.3. Queues

### 5.3.1. Queue Family Properties

As discussed in the Physical Device Enumeration section above, the vkGetPhysicalDeviceQueueFamilyProperties command is used to retrieve details about the queue

families and queues supported by a device.

Each index in the pQueueFamilyProperties array returned by vkGetPhysicalDeviceQueueFamilyProperties describes a unique queue family on that physical device. These indices are used when creating queues, and they correspond directly with the queueFamilyIndex that is passed to the vkCreateDevice command via the VkDeviceQueueCreateInfo structure as described in the Queue Creation section below.

Grouping of queue families within a physical device is implementation-dependent.

| | |
|---|---|
| **NOTE** | The general expectation is that a physical device groups all queues of matching capabilities into a single family. However, while implementations **should** do this, it is possible that a physical device **may** return two separate queue families with the same capabilities. |

Once an application has identified a physical device with the queue(s) that it desires to use, it will create those queues in conjunction with a logical device. This is described in the following section.

### 5.3.2. Queue Creation

Creating a logical device also creates the queues associated with that device. The queues to create are described by a set of VkDeviceQueueCreateInfo structures that are passed to vkCreateDevice in pQueueCreateInfos.

Queues are represented by VkQueue handles:

```
// Provided by VK_VERSION_1_0
VK_DEFINE_HANDLE(VkQueue)
```

The VkDeviceQueueCreateInfo structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkDeviceQueueCreateInfo {
    VkStructureType             sType;
    const void*                 pNext;
    VkDeviceQueueCreateFlags    flags;
    uint32_t                    queueFamilyIndex;
    uint32_t                    queueCount;
    const float*                pQueuePriorities;
} VkDeviceQueueCreateInfo;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- flags is a bitmask indicating behavior of the queues.
- queueFamilyIndex is an unsigned integer indicating the index of the queue family in which to create the queues on this device. This index corresponds to the index of an element of the

pQueueFamilyProperties array that was returned by vkGetPhysicalDeviceQueueFamilyProperties.

- queueCount is an unsigned integer specifying the number of queues to create in the queue family indicated by queueFamilyIndex, and with the behavior specified by flags.
- pQueuePriorities is a pointer to an array of queueCount normalized floating-point values, specifying priorities of work that will be submitted to each created queue. See Queue Priority for more information.

## Valid Usage

- VUID-VkDeviceQueueCreateInfo-queueFamilyIndex-00381
  queueFamilyIndex **must** be less than pQueueFamilyPropertyCount returned by vkGetPhysicalDeviceQueueFamilyProperties
- VUID-VkDeviceQueueCreateInfo-queueCount-00382
  queueCount **must** be less than or equal to the queueCount member of the VkQueueFamilyProperties structure, as returned by vkGetPhysicalDeviceQueueFamilyProperties in the pQueueFamilyProperties[queueFamilyIndex]
- VUID-VkDeviceQueueCreateInfo-pQueuePriorities-00383
  Each element of pQueuePriorities **must** be between 0.0 and 1.0 inclusive
- VUID-VkDeviceQueueCreateInfo-flags-02861
  If the protectedMemory feature is not enabled, the VK_DEVICE_QUEUE_CREATE_PROTECTED_BIT bit of flags **must** not be set
- VUID-VkDeviceQueueCreateInfo-flags-06449
  If flags includes VK_DEVICE_QUEUE_CREATE_PROTECTED_BIT, queueFamilyIndex **must** be the index of a queue family that includes the VK_QUEUE_PROTECTED_BIT capability
- VUID-VkDeviceQueueCreateInfo-pNext-09398
  If the pNext chain includes a VkDeviceQueueShaderCoreControlCreateInfoARM structure then VkPhysicalDeviceSchedulingControlsPropertiesARM::schedulingControlsFlags **must** contain VK_PHYSICAL_DEVICE_SCHEDULING_CONTROLS_SHADER_CORE_COUNT_ARM

## Valid Usage (Implicit)

- VUID-VkDeviceQueueCreateInfo-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_DEVICE_QUEUE_CREATE_INFO
- VUID-VkDeviceQueueCreateInfo-pNext-pNext
  Each pNext member of any structure (including this one) in the pNext chain **must** be either NULL or a pointer to a valid instance of VkDeviceQueueGlobalPriorityCreateInfoKHR or VkDeviceQueueShaderCoreControlCreateInfoARM
- VUID-VkDeviceQueueCreateInfo-sType-unique
  The sType value of each struct in the pNext chain **must** be unique
- VUID-VkDeviceQueueCreateInfo-flags-parameter
  flags **must** be a valid combination of VkDeviceQueueCreateFlagBits values

- VUID-VkDeviceQueueCreateInfo-pQueuePriorities-parameter
  `pQueuePriorities` **must** be a valid pointer to an array of `queueCount` `float` values
- VUID-VkDeviceQueueCreateInfo-queueCount-arraylength
  `queueCount` **must** be greater than `0`

Bits which **can** be set in VkDeviceQueueCreateInfo::`flags`, specifying usage behavior of a queue, are:

```
// Provided by VK_VERSION_1_1
typedef enum VkDeviceQueueCreateFlagBits {
  // Provided by VK_VERSION_1_1
    VK_DEVICE_QUEUE_CREATE_PROTECTED_BIT = 0x00000001,
} VkDeviceQueueCreateFlagBits;
```

- `VK_DEVICE_QUEUE_CREATE_PROTECTED_BIT` specifies that the device queue is a protected-capable queue.

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkDeviceQueueCreateFlags;
```

`VkDeviceQueueCreateFlags` is a bitmask type for setting a mask of zero or more VkDeviceQueueCreateFlagBits.

Queues **can** be created with a system-wide priority by adding a `VkDeviceQueueGlobalPriorityCreateInfoKHR` structure to the `pNext` chain of VkDeviceQueueCreateInfo.

The `VkDeviceQueueGlobalPriorityCreateInfoKHR` structure is defined as:

```
// Provided by VK_KHR_global_priority
typedef struct VkDeviceQueueGlobalPriorityCreateInfoKHR {
    VkStructureType             sType;
    const void*                 pNext;
    VkQueueGlobalPriorityKHR    globalPriority;
} VkDeviceQueueGlobalPriorityCreateInfoKHR;
```

or the equivalent

```
// Provided by VK_EXT_global_priority
typedef VkDeviceQueueGlobalPriorityCreateInfoKHR
VkDeviceQueueGlobalPriorityCreateInfoEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.

- `globalPriority` is the system-wide priority associated to these queues as specified by VkQueueGlobalPriorityKHR

Queues created without specifying `VkDeviceQueueGlobalPriorityCreateInfoKHR` will default to `VK_QUEUE_GLOBAL_PRIORITY_MEDIUM_KHR`.

### Valid Usage (Implicit)

- VUID-VkDeviceQueueGlobalPriorityCreateInfoKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_DEVICE_QUEUE_GLOBAL_PRIORITY_CREATE_INFO_KHR`
- VUID-VkDeviceQueueGlobalPriorityCreateInfoKHR-globalPriority-parameter
  `globalPriority` **must** be a valid VkQueueGlobalPriorityKHR value

Possible values of VkDeviceQueueGlobalPriorityCreateInfoKHR::`globalPriority`, specifying a system-wide priority level are:

```
// Provided by VK_KHR_global_priority
typedef enum VkQueueGlobalPriorityKHR {
    VK_QUEUE_GLOBAL_PRIORITY_LOW_KHR = 128,
    VK_QUEUE_GLOBAL_PRIORITY_MEDIUM_KHR = 256,
    VK_QUEUE_GLOBAL_PRIORITY_HIGH_KHR = 512,
    VK_QUEUE_GLOBAL_PRIORITY_REALTIME_KHR = 1024,
    VK_QUEUE_GLOBAL_PRIORITY_LOW_EXT = VK_QUEUE_GLOBAL_PRIORITY_LOW_KHR,
    VK_QUEUE_GLOBAL_PRIORITY_MEDIUM_EXT = VK_QUEUE_GLOBAL_PRIORITY_MEDIUM_KHR,
    VK_QUEUE_GLOBAL_PRIORITY_HIGH_EXT = VK_QUEUE_GLOBAL_PRIORITY_HIGH_KHR,
    VK_QUEUE_GLOBAL_PRIORITY_REALTIME_EXT = VK_QUEUE_GLOBAL_PRIORITY_REALTIME_KHR,
} VkQueueGlobalPriorityKHR;
```

or the equivalent

```
// Provided by VK_EXT_global_priority
typedef VkQueueGlobalPriorityKHR VkQueueGlobalPriorityEXT;
```

Priority values are sorted in ascending order. A comparison operation on the enum values can be used to determine the priority order.

- `VK_QUEUE_GLOBAL_PRIORITY_LOW_KHR` is below the system default. Useful for non-interactive tasks.
- `VK_QUEUE_GLOBAL_PRIORITY_MEDIUM_KHR` is the system default priority.
- `VK_QUEUE_GLOBAL_PRIORITY_HIGH_KHR` is above the system default.
- `VK_QUEUE_GLOBAL_PRIORITY_REALTIME_KHR` is the highest priority. Useful for critical tasks.

Queues with higher system priority **may** be allotted more processing time than queues with lower priority. An implementation **may** allow a higher-priority queue to starve a lower-priority queue until the higher-priority queue has no further commands to execute.

Priorities imply no ordering or scheduling constraints.

No specific guarantees are made about higher priority queues receiving more processing time or better quality of service than lower priority queues.

The global priority level of a queue takes precedence over the per-process queue priority (VkDeviceQueueCreateInfo::pQueuePriorities).

Abuse of this feature **may** result in starving the rest of the system of implementation resources. Therefore, the driver implementation **may** deny requests to acquire a priority above the default priority (VK_QUEUE_GLOBAL_PRIORITY_MEDIUM_KHR) if the caller does not have sufficient privileges. In this scenario VK_ERROR_NOT_PERMITTED_KHR is returned.

The driver implementation **may** fail the queue allocation request if resources required to complete the operation have been exhausted (either by the same process or a different process). In this scenario VK_ERROR_INITIALIZATION_FAILED is returned.

If the globalPriorityQuery feature is enabled and the requested global priority is not reported via VkQueueFamilyGlobalPriorityPropertiesKHR, the driver implementation **must** fail the queue creation. In this scenario, VK_ERROR_INITIALIZATION_FAILED is returned.

The number of shader cores used by a queue **can** be controlled by adding a VkDeviceQueueShaderCoreControlCreateInfoARM structure to the pNext chain of VkDeviceQueueCreateInfo structures.

The VkDeviceQueueShaderCoreControlCreateInfoARM structure is defined as:

```
// Provided by VK_ARM_scheduling_controls
typedef struct VkDeviceQueueShaderCoreControlCreateInfoARM {
    VkStructureType    sType;
    void*              pNext;
    uint32_t           shaderCoreCount;
} VkDeviceQueueShaderCoreControlCreateInfoARM;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- shaderCoreCount is the number of shader cores this queue uses.

Queues created without specifying VkDeviceQueueShaderCoreControlCreateInfoARM will default to using all the shader cores available.

### Valid Usage

- VUID-VkDeviceQueueShaderCoreControlCreateInfoARM-shaderCoreCount-09399
  shaderCoreCount **must** be greater than 0 and less than or equal to the total number of shader cores as reported via VkPhysicalDeviceShaderCoreBuiltinsPropertiesARM::shaderCoreCount

### Valid Usage (Implicit)

- VUID-VkDeviceQueueShaderCoreControlCreateInfoARM-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_DEVICE_QUEUE_SHADER_CORE_CONTROL_CREATE_INFO_ARM`

To retrieve a handle to a VkQueue object, call:

```
// Provided by VK_VERSION_1_0
void vkGetDeviceQueue(
    VkDevice                                    device,
    uint32_t                                    queueFamilyIndex,
    uint32_t                                    queueIndex,
    VkQueue*                                    pQueue);
```

- `device` is the logical device that owns the queue.
- `queueFamilyIndex` is the index of the queue family to which the queue belongs.
- `queueIndex` is the index within this queue family of the queue to retrieve.
- `pQueue` is a pointer to a VkQueue object that will be filled with the handle for the requested queue.

`vkGetDeviceQueue` **must** only be used to get queues that were created with the `flags` parameter of VkDeviceQueueCreateInfo set to zero. To get queues that were created with a non-zero `flags` parameter use vkGetDeviceQueue2.

### Valid Usage

- VUID-vkGetDeviceQueue-queueFamilyIndex-00384
  `queueFamilyIndex` **must** be one of the queue family indices specified when `device` was created, via the VkDeviceQueueCreateInfo structure
- VUID-vkGetDeviceQueue-queueIndex-00385
  `queueIndex` **must** be less than the value of VkDeviceQueueCreateInfo::`queueCount` for the queue family indicated by `queueFamilyIndex` when `device` was created
- VUID-vkGetDeviceQueue-flags-01841
  VkDeviceQueueCreateInfo::`flags` **must** have been set to zero when `device` was created

### Valid Usage (Implicit)

- VUID-vkGetDeviceQueue-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkGetDeviceQueue-pQueue-parameter
  `pQueue` **must** be a valid pointer to a VkQueue handle

To retrieve a handle to a VkQueue object with specific VkDeviceQueueCreateFlags creation flags, call:

```
// Provided by VK_VERSION_1_1
void vkGetDeviceQueue2(
    VkDevice                                    device,
    const VkDeviceQueueInfo2*                   pQueueInfo,
    VkQueue*                                    pQueue);
```

- device is the logical device that owns the queue.
- pQueueInfo is a pointer to a VkDeviceQueueInfo2 structure, describing parameters of the device queue to be retrieved.
- pQueue is a pointer to a VkQueue object that will be filled with the handle for the requested queue.

### Valid Usage (Implicit)

- VUID-vkGetDeviceQueue2-device-parameter
  device **must** be a valid VkDevice handle
- VUID-vkGetDeviceQueue2-pQueueInfo-parameter
  pQueueInfo **must** be a valid pointer to a valid VkDeviceQueueInfo2 structure
- VUID-vkGetDeviceQueue2-pQueue-parameter
  pQueue **must** be a valid pointer to a VkQueue handle

The VkDeviceQueueInfo2 structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkDeviceQueueInfo2 {
    VkStructureType             sType;
    const void*                 pNext;
    VkDeviceQueueCreateFlags    flags;
    uint32_t                    queueFamilyIndex;
    uint32_t                    queueIndex;
} VkDeviceQueueInfo2;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure. The pNext chain of VkDeviceQueueInfo2 **can** be used to provide additional device queue parameters to vkGetDeviceQueue2.
- flags is a VkDeviceQueueCreateFlags value indicating the flags used to create the device queue.
- queueFamilyIndex is the index of the queue family to which the queue belongs.
- queueIndex is the index of the queue to retrieve from within the set of queues that share both the queue family and flags specified.

The queue returned by `vkGetDeviceQueue2` **must** have the same `flags` value from this structure as that used at device creation time in a VkDeviceQueueCreateInfo structure.

**NOTE**

Normally, if you create both protected-capable and non-protected-capable queues with the same family, they are treated as separate lists of queues and `queueIndex` is relative to the start of the list of queues specified by both `queueFamilyIndex` and `flags`. However, for historical reasons, some implementations may exhibit different behavior. These divergent implementations instead concatenate the lists of queues and treat `queueIndex` as relative to the start of the first list of queues with the given `queueFamilyIndex`. This only matters in cases where an application has created both protected-capable and non-protected-capable queues from the same queue family.

For such divergent implementations, the maximum value of `queueIndex` is equal to the sum of VkDeviceQueueCreateInfo::`queueCount` minus one, for all VkDeviceQueueCreateInfo structures that share a common `queueFamilyIndex`.

Such implementations will return `NULL` for either the protected or unprotected queues when calling `vkGetDeviceQueue2` with `queueIndex` in the range zero to VkDeviceQueueCreateInfo::`queueCount` minus one. In cases where these implementations returned `NULL`, the corresponding queues are instead located in the extended range described in the preceding two paragraphs.

This behavior will not be observed on any driver that has passed Vulkan conformance test suite version 1.3.3.0, or any subsequent version. This information can be found by querying `VkPhysicalDeviceDriverProperties`::`conformanceVersion`.

## Valid Usage

- VUID-VkDeviceQueueInfo2-queueFamilyIndex-01842
  `queueFamilyIndex` **must** be one of the queue family indices specified when `device` was created, via the VkDeviceQueueCreateInfo structure
- VUID-VkDeviceQueueInfo2-flags-06225
  `flags` **must** be equal to VkDeviceQueueCreateInfo::`flags` for a VkDeviceQueueCreateInfo structure for the queue family indicated by `queueFamilyIndex` when `device` was created
- VUID-VkDeviceQueueInfo2-queueIndex-01843
  `queueIndex` **must** be less than VkDeviceQueueCreateInfo::`queueCount` for the corresponding queue family and flags indicated by `queueFamilyIndex` and `flags` when `device` was created

## Valid Usage (Implicit)

- VUID-VkDeviceQueueInfo2-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_DEVICE_QUEUE_INFO_2`
- VUID-VkDeviceQueueInfo2-pNext-pNext
  `pNext` **must** be `NULL`
- VUID-VkDeviceQueueInfo2-flags-parameter

`flags` **must** be a valid combination of VkDeviceQueueCreateFlagBits values

### 5.3.3. Queue Family Index

The queue family index is used in multiple places in Vulkan in order to tie operations to a specific family of queues.

When retrieving a handle to the queue via `vkGetDeviceQueue`, the queue family index is used to select which queue family to retrieve the `VkQueue` handle from as described in the previous section.

When creating a `VkCommandPool` object (see Command Pools), a queue family index is specified in the VkCommandPoolCreateInfo structure. Command buffers from this pool **can** only be submitted on queues corresponding to this queue family.

When creating `VkImage` (see Images) and `VkBuffer` (see Buffers) resources, a set of queue families is included in the VkImageCreateInfo and VkBufferCreateInfo structures to specify the queue families that **can** access the resource.

When inserting a VkBufferMemoryBarrier or VkImageMemoryBarrier (see Pipeline Barriers), a source and destination queue family index is specified to allow the ownership of a buffer or image to be transferred from one queue family to another. See the Resource Sharing section for details.

### 5.3.4. Queue Priority

Each queue is assigned a priority, as set in the VkDeviceQueueCreateInfo structures when creating the device. The priority of each queue is a normalized floating-point value between 0.0 and 1.0, which is then translated to a discrete priority level by the implementation. Higher values indicate a higher priority, with 0.0 being the lowest priority and 1.0 being the highest.

Within the same device, queues with higher priority **may** be allotted more processing time than queues with lower priority. The implementation makes no guarantees with regards to ordering or scheduling among queues with the same priority, other than the constraints defined by any explicit synchronization primitives. The implementation makes no guarantees with regards to queues across different devices.

An implementation **may** allow a higher-priority queue to starve a lower-priority queue on the same `VkDevice` until the higher-priority queue has no further commands to execute. The relationship of queue priorities **must** not cause queues on one `VkDevice` to starve queues on another `VkDevice`.

No specific guarantees are made about higher priority queues receiving more processing time or better quality of service than lower priority queues.

### 5.3.5. Queue Submission

Work is submitted to a queue via *queue submission* commands such as vkQueueSubmit2KHR or vkQueueSubmit. Queue submission commands define a set of *queue operations* to be executed by the underlying physical device, including synchronization with semaphores and fences.

Submission commands take as parameters a target queue, zero or more *batches* of work, and an

**optional** fence to signal upon completion. Each batch consists of three distinct parts:

1. Zero or more semaphores to wait on before execution of the rest of the batch.
    - If present, these describe a semaphore wait operation.
2. Zero or more work items to execute.
    - If present, these describe a *queue operation* matching the work described.
3. Zero or more semaphores to signal upon completion of the work items.
    - If present, these describe a semaphore signal operation.

If a fence is present in a queue submission, it describes a fence signal operation.

All work described by a queue submission command **must** be submitted to the queue before the command returns.

#### Sparse Memory Binding

In Vulkan it is possible to sparsely bind memory to buffers and images as described in the Sparse Resource chapter. Sparse memory binding is a queue operation. A queue whose flags include the `VK_QUEUE_SPARSE_BINDING_BIT` **must** be able to support the mapping of a virtual address to a physical address on the device. This causes an update to the page table mappings on the device. This update **must** be synchronized on a queue to avoid corrupting page table mappings during execution of graphics commands. By binding the sparse memory resources on queues, all commands that are dependent on the updated bindings are synchronized to only execute after the binding is updated. See the Synchronization and Cache Control chapter for how this synchronization is accomplished.

### 5.3.6. Queue Destruction

Queues are created along with a logical device during `vkCreateDevice`. All queues associated with a logical device are destroyed when `vkDestroyDevice` is called on that device.

# Chapter 6. Command Buffers

Command buffers are objects used to record commands which **can** be subsequently submitted to a device queue for execution. There are two levels of command buffers - *primary command buffers*, which **can** execute secondary command buffers, and which are submitted to queues, and *secondary command buffers*, which **can** be executed by primary command buffers, and which are not directly submitted to queues.

Command buffers are represented by `VkCommandBuffer` handles:

```
// Provided by VK_VERSION_1_0
VK_DEFINE_HANDLE(VkCommandBuffer)
```

Recorded commands include commands to bind pipelines and descriptor sets to the command buffer, commands to modify dynamic state, commands to draw (for graphics rendering), commands to dispatch (for compute), commands to execute secondary command buffers (for primary command buffers only), commands to copy buffers and images, and other commands.

Each command buffer manages state independently of other command buffers. There is no inheritance of state across primary and secondary command buffers, or between secondary command buffers. When a command buffer begins recording, all state in that command buffer is undefined. When secondary command buffer(s) are recorded to execute on a primary command buffer, the secondary command buffer inherits no state from the primary command buffer, and all state of the primary command buffer is undefined after an execute secondary command buffer command is recorded. There is one exception to this rule - if the primary command buffer is inside a render pass instance, then the render pass and subpass state is not disturbed by executing secondary command buffers. For state dependent commands (such as draws and dispatches), any state consumed by those commands **must** not be undefined.

VkCommandBufferInheritanceViewportScissorInfoNV defines an exception allowing limited inheritance of dynamic viewport and scissor state.

Unless otherwise specified, and without explicit synchronization, the various commands submitted to a queue via command buffers **may** execute in arbitrary order relative to each other, and/or concurrently. Also, the memory side effects of those commands **may** not be directly visible to other commands without explicit memory dependencies. This is true within a command buffer, and across command buffers submitted to a given queue. See the synchronization chapter for information on implicit and explicit synchronization between commands.

## 6.1. Command Buffer Lifecycle

Each command buffer is always in one of the following states:

**Initial**

When a command buffer is allocated, it is in the *initial state*. Some commands are able to *reset* a command buffer (or a set of command buffers) back to this state from any of the executable, recording or invalid state. Command buffers in the initial state **can** only be moved to the

recording state, or freed.

**Recording**

vkBeginCommandBuffer changes the state of a command buffer from the initial state to the *recording state*. Once a command buffer is in the recording state, `vkCmd*` commands **can** be used to record to the command buffer.

**Executable**

vkEndCommandBuffer ends the recording of a command buffer, and moves it from the recording state to the *executable state*. Executable command buffers **can** be submitted, reset, or recorded to another command buffer.

**Pending**

Queue submission of a command buffer changes the state of a command buffer from the executable state to the *pending state*. Whilst in the pending state, applications **must** not attempt to modify the command buffer in any way - as the device **may** be processing the commands recorded to it. Once execution of a command buffer completes, the command buffer either reverts back to the *executable state*, or if it was recorded with `VK_COMMAND_BUFFER_USAGE_ONE_TIME_SUBMIT_BIT`, it moves to the *invalid state*. A synchronization command **should** be used to detect when this occurs.

**Invalid**

Some operations, such as modifying or deleting a resource that was used in a command recorded to a command buffer, will transition the state of that command buffer into the *invalid state*. Command buffers in the invalid state **can** only be reset or freed.

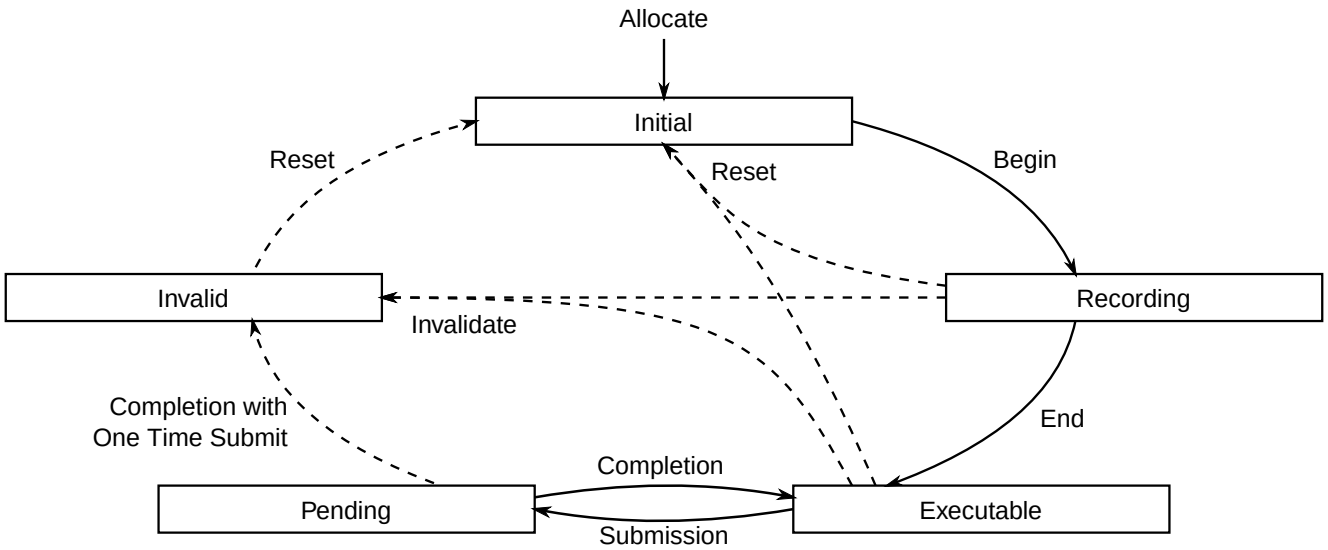



*Figure 1. Lifecycle of a command buffer*

Any given command that operates on a command buffer has its own requirements on what state a command buffer **must** be in, which are detailed in the valid usage constraints for that command.

Resetting a command buffer is an operation that discards any previously recorded commands and puts a command buffer in the *initial state*. Resetting occurs as a result of vkResetCommandBuffer or vkResetCommandPool, or as part of vkBeginCommandBuffer (which additionally puts the command buffer in the *recording state*).

Secondary command buffers **can** be recorded to a primary command buffer via vkCmdExecuteCommands. This partially ties the lifecycle of the two command buffers together - if the primary is submitted to a queue, both the primary and any secondaries recorded to it move to the *pending state*. Once execution of the primary completes, so it does for any secondary recorded within it. After all executions of each command buffer complete, they each move to their appropriate completion state (either to the *executable state* or the *invalid state*, as specified above).

If a secondary moves to the *invalid state* or the *initial state*, then all primary buffers it is recorded in move to the *invalid state*. A primary moving to any other state does not affect the state of a secondary recorded in it.

| | |
|---|---|
| **NOTE** | Resetting or freeing a primary command buffer removes the lifecycle linkage to all secondary command buffers that were recorded into it. |

## 6.2. Command Pools

Command pools are opaque objects that command buffer memory is allocated from, and which allow the implementation to amortize the cost of resource creation across multiple command buffers. Command pools are externally synchronized, meaning that a command pool **must** not be used concurrently in multiple threads. That includes use via recording commands on any command buffers allocated from the pool, as well as operations that allocate, free, and reset command buffers or the pool itself.

Command pools are represented by `VkCommandPool` handles:

```
// Provided by VK_VERSION_1_0
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkCommandPool)
```

To create a command pool, call:

```
// Provided by VK_VERSION_1_0
VkResult vkCreateCommandPool(
    VkDevice                                    device,
    const VkCommandPoolCreateInfo*              pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkCommandPool*                              pCommandPool);
```

- `device` is the logical device that creates the command pool.
- `pCreateInfo` is a pointer to a VkCommandPoolCreateInfo structure specifying the state of the command pool object.
- `pAllocator` controls host memory allocation as described in the Memory Allocation chapter.
- `pCommandPool` is a pointer to a VkCommandPool handle in which the created pool is returned.

### Valid Usage

- VUID-vkCreateCommandPool-queueFamilyIndex-01937
  `pCreateInfo->queueFamilyIndex` **must** be the index of a queue family available in the logical device `device`

### Valid Usage (Implicit)

- VUID-vkCreateCommandPool-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkCreateCommandPool-pCreateInfo-parameter
  `pCreateInfo` **must** be a valid pointer to a valid VkCommandPoolCreateInfo structure
- VUID-vkCreateCommandPool-pAllocator-parameter
  If `pAllocator` is not `NULL`, `pAllocator` **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkCreateCommandPool-pCommandPool-parameter
  `pCommandPool` **must** be a valid pointer to a VkCommandPool handle

### Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`

The `VkCommandPoolCreateInfo` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkCommandPoolCreateInfo {
    VkStructureType             sType;
    const void*                 pNext;
    VkCommandPoolCreateFlags    flags;
    uint32_t                    queueFamilyIndex;
} VkCommandPoolCreateInfo;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is a bitmask of VkCommandPoolCreateFlagBits indicating usage behavior for the pool and command buffers allocated from it.

- queueFamilyIndex designates a queue family as described in section Queue Family Properties. All command buffers allocated from this command pool **must** be submitted on queues from the same queue family.

### Valid Usage

- VUID-VkCommandPoolCreateInfo-flags-02860
  If the protectedMemory feature is not enabled, the VK_COMMAND_POOL_CREATE_PROTECTED_BIT bit of flags **must** not be set

### Valid Usage (Implicit)

- VUID-VkCommandPoolCreateInfo-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_COMMAND_POOL_CREATE_INFO
- VUID-VkCommandPoolCreateInfo-pNext-pNext
  pNext **must** be NULL
- VUID-VkCommandPoolCreateInfo-flags-parameter
  flags **must** be a valid combination of VkCommandPoolCreateFlagBits values

Bits which **can** be set in VkCommandPoolCreateInfo::flags, specifying usage behavior for a command pool, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkCommandPoolCreateFlagBits {
    VK_COMMAND_POOL_CREATE_TRANSIENT_BIT = 0x00000001,
    VK_COMMAND_POOL_CREATE_RESET_COMMAND_BUFFER_BIT = 0x00000002,
  // Provided by VK_VERSION_1_1
    VK_COMMAND_POOL_CREATE_PROTECTED_BIT = 0x00000004,
} VkCommandPoolCreateFlagBits;
```

- VK_COMMAND_POOL_CREATE_TRANSIENT_BIT specifies that command buffers allocated from the pool will be short-lived, meaning that they will be reset or freed in a relatively short timeframe. This flag **may** be used by the implementation to control memory allocation behavior within the pool.
- VK_COMMAND_POOL_CREATE_RESET_COMMAND_BUFFER_BIT allows any command buffer allocated from a pool to be individually reset to the initial state; either by calling vkResetCommandBuffer, or via the implicit reset when calling vkBeginCommandBuffer. If this flag is not set on a pool, then vkResetCommandBuffer **must** not be called for any command buffer allocated from that pool.
- VK_COMMAND_POOL_CREATE_PROTECTED_BIT specifies that command buffers allocated from the pool are protected command buffers.

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkCommandPoolCreateFlags;
```

VkCommandPoolCreateFlags is a bitmask type for setting a mask of zero or more VkCommandPoolCreateFlagBits.

To trim a command pool, call:

```
// Provided by VK_VERSION_1_1
void vkTrimCommandPool(
    VkDevice                                    device,
    VkCommandPool                               commandPool,
    VkCommandPoolTrimFlags                      flags);
```

or the equivalent command

```
// Provided by VK_KHR_maintenance1
void vkTrimCommandPoolKHR(
    VkDevice                                    device,
    VkCommandPool                               commandPool,
    VkCommandPoolTrimFlags                      flags);
```

- device is the logical device that owns the command pool.
- commandPool is the command pool to trim.
- flags is reserved for future use.

Trimming a command pool recycles unused memory from the command pool back to the system. Command buffers allocated from the pool are not affected by the command.

**NOTE**

This command provides applications with some control over the internal memory allocations used by command pools.

Unused memory normally arises from command buffers that have been recorded and later reset, such that they are no longer using the memory. On reset, a command buffer can return memory to its command pool, but the only way to release memory from a command pool to the system requires calling vkResetCommandPool, which cannot be executed while any command buffers from that pool are still in use. Subsequent recording operations into command buffers will reuse this memory but since total memory requirements fluctuate over time, unused memory can accumulate.

In this situation, trimming a command pool **may** be useful to return unused memory back to the system, returning the total outstanding memory allocated by the pool back to a more “average” value.

Implementations utilize many internal allocation strategies that make it impossible to guarantee that all unused memory is released back to the system. For instance, an implementation of a command pool **may** involve allocating memory in bulk from the system and sub-allocating from that memory. In such an implementation any live command buffer that holds a reference to a bulk allocation would prevent

that allocation from being freed, even if only a small proportion of the bulk allocation is in use.

In most cases trimming will result in a reduction in allocated but unused memory, but it does not guarantee the “ideal” behavior.

Trimming **may** be an expensive operation, and **should** not be called frequently. Trimming **should** be treated as a way to relieve memory pressure after application-known points when there exists enough unused memory that the cost of trimming is “worth” it.

### Valid Usage (Implicit)

- VUID-vkTrimCommandPool-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkTrimCommandPool-commandPool-parameter
  `commandPool` **must** be a valid VkCommandPool handle
- VUID-vkTrimCommandPool-flags-zerobitmask
  `flags` **must** be `0`
- VUID-vkTrimCommandPool-commandPool-parent
  `commandPool` **must** have been created, allocated, or retrieved from `device`

### Host Synchronization

- Host access to `commandPool` **must** be externally synchronized

```
// Provided by VK_VERSION_1_1
typedef VkFlags VkCommandPoolTrimFlags;
```

or the equivalent

```
// Provided by VK_KHR_maintenance1
typedef VkCommandPoolTrimFlags VkCommandPoolTrimFlagsKHR;
```

`VkCommandPoolTrimFlags` is a bitmask type for setting a mask, but is currently reserved for future use.

To reset a command pool, call:

```
// Provided by VK_VERSION_1_0
VkResult vkResetCommandPool(
    VkDevice                                    device,
    VkCommandPool                               commandPool,
    VkCommandPoolResetFlags                     flags);
```

- `device` is the logical device that owns the command pool.
- `commandPool` is the command pool to reset.
- `flags` is a bitmask of VkCommandPoolResetFlagBits controlling the reset operation.

Resetting a command pool recycles all of the resources from all of the command buffers allocated from the command pool back to the command pool. All command buffers that have been allocated from the command pool are put in the initial state.

Any primary command buffer allocated from another VkCommandPool that is in the recording or executable state and has a secondary command buffer allocated from `commandPool` recorded into it, becomes invalid.

### Valid Usage

- VUID-vkResetCommandPool-commandPool-00040
  All `VkCommandBuffer` objects allocated from `commandPool` **must** not be in the pending state

### Valid Usage (Implicit)

- VUID-vkResetCommandPool-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkResetCommandPool-commandPool-parameter
  `commandPool` **must** be a valid VkCommandPool handle
- VUID-vkResetCommandPool-flags-parameter
  `flags` **must** be a valid combination of VkCommandPoolResetFlagBits values
- VUID-vkResetCommandPool-commandPool-parent
  `commandPool` **must** have been created, allocated, or retrieved from `device`

### Host Synchronization

- Host access to `commandPool` **must** be externally synchronized

### Return Codes

**Success**

- VK_SUCCESS

**Failure**

- VK_ERROR_OUT_OF_DEVICE_MEMORY

Bits which **can** be set in vkResetCommandPool::flags, controlling the reset operation, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkCommandPoolResetFlagBits {
    VK_COMMAND_POOL_RESET_RELEASE_RESOURCES_BIT = 0x00000001,
} VkCommandPoolResetFlagBits;
```

- VK_COMMAND_POOL_RESET_RELEASE_RESOURCES_BIT specifies that resetting a command pool recycles all of the resources from the command pool back to the system.

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkCommandPoolResetFlags;
```

VkCommandPoolResetFlags is a bitmask type for setting a mask of zero or more VkCommandPoolResetFlagBits.

To destroy a command pool, call:

```
// Provided by VK_VERSION_1_0
void vkDestroyCommandPool(
    VkDevice                                    device,
    VkCommandPool                               commandPool,
    const VkAllocationCallbacks*                pAllocator);
```

- device is the logical device that destroys the command pool.
- commandPool is the handle of the command pool to destroy.
- pAllocator controls host memory allocation as described in the Memory Allocation chapter.

When a pool is destroyed, all command buffers allocated from the pool are freed.

Any primary command buffer allocated from another VkCommandPool that is in the recording or executable state and has a secondary command buffer allocated from commandPool recorded into it, becomes invalid.

### Valid Usage

- VUID-vkDestroyCommandPool-commandPool-00041
  All `VkCommandBuffer` objects allocated from `commandPool` **must** not be in the pending state
- VUID-vkDestroyCommandPool-commandPool-00042
  If `VkAllocationCallbacks` were provided when `commandPool` was created, a compatible set of callbacks **must** be provided here
- VUID-vkDestroyCommandPool-commandPool-00043
  If no `VkAllocationCallbacks` were provided when `commandPool` was created, `pAllocator` **must** be `NULL`

### Valid Usage (Implicit)

- VUID-vkDestroyCommandPool-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkDestroyCommandPool-commandPool-parameter
  If `commandPool` is not VK_NULL_HANDLE, `commandPool` **must** be a valid VkCommandPool handle
- VUID-vkDestroyCommandPool-pAllocator-parameter
  If `pAllocator` is not `NULL`, `pAllocator` **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkDestroyCommandPool-commandPool-parent
  If `commandPool` is a valid handle, it **must** have been created, allocated, or retrieved from `device`

### Host Synchronization

- Host access to `commandPool` **must** be externally synchronized

## 6.3. Command Buffer Allocation and Management

To allocate command buffers, call:

```
// Provided by VK_VERSION_1_0
VkResult vkAllocateCommandBuffers(
    VkDevice                                    device,
    const VkCommandBufferAllocateInfo*          pAllocateInfo,
    VkCommandBuffer*                            pCommandBuffers);
```

- `device` is the logical device that owns the command pool.
- `pAllocateInfo` is a pointer to a VkCommandBufferAllocateInfo structure describing parameters

of the allocation.

- `pCommandBuffers` is a pointer to an array of VkCommandBuffer handles in which the resulting command buffer objects are returned. The array **must** be at least the length specified by the `commandBufferCount` member of `pAllocateInfo`. Each allocated command buffer begins in the initial state.

`vkAllocateCommandBuffers` **can** be used to allocate multiple command buffers. If the allocation of any of those command buffers fails, the implementation **must** free all successfully allocated command buffer objects from this command, set all entries of the `pCommandBuffers` array to `NULL` and return the error.

| | |
|---|---|
| **NOTE** | Filling `pCommandBuffers` with `NULL` values on failure is an exception to the default error behavior that output parameters will have undefined contents. |

When command buffers are first allocated, they are in the initial state.

## Valid Usage (Implicit)

- VUID-vkAllocateCommandBuffers-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkAllocateCommandBuffers-pAllocateInfo-parameter
  `pAllocateInfo` **must** be a valid pointer to a valid VkCommandBufferAllocateInfo structure
- VUID-vkAllocateCommandBuffers-pCommandBuffers-parameter
  `pCommandBuffers` **must** be a valid pointer to an array of `pAllocateInfo->commandBufferCount` VkCommandBuffer handles
- VUID-vkAllocateCommandBuffers-pAllocateInfo::commandBufferCount-arraylength
  `pAllocateInfo->commandBufferCount` **must** be greater than `0`

## Host Synchronization

- Host access to `pAllocateInfo->commandPool` **must** be externally synchronized

## Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`

The `VkCommandBufferAllocateInfo` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkCommandBufferAllocateInfo {
    VkStructureType         sType;
    const void*             pNext;
    VkCommandPool           commandPool;
    VkCommandBufferLevel    level;
    uint32_t                commandBufferCount;
} VkCommandBufferAllocateInfo;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `commandPool` is the command pool from which the command buffers are allocated.
- `level` is a VkCommandBufferLevel value specifying the command buffer level.
- `commandBufferCount` is the number of command buffers to allocate from the pool.

### Valid Usage (Implicit)

- VUID-VkCommandBufferAllocateInfo-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_COMMAND_BUFFER_ALLOCATE_INFO`
- VUID-VkCommandBufferAllocateInfo-pNext-pNext
  `pNext` **must** be `NULL`
- VUID-VkCommandBufferAllocateInfo-commandPool-parameter
  `commandPool` **must** be a valid VkCommandPool handle
- VUID-VkCommandBufferAllocateInfo-level-parameter
  `level` **must** be a valid VkCommandBufferLevel value

Possible values of VkCommandBufferAllocateInfo::`level`, specifying the command buffer level, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkCommandBufferLevel {
    VK_COMMAND_BUFFER_LEVEL_PRIMARY = 0,
    VK_COMMAND_BUFFER_LEVEL_SECONDARY = 1,
} VkCommandBufferLevel;
```

- `VK_COMMAND_BUFFER_LEVEL_PRIMARY` specifies a primary command buffer.
- `VK_COMMAND_BUFFER_LEVEL_SECONDARY` specifies a secondary command buffer.

To reset a command buffer, call:

```
// Provided by VK_VERSION_1_0
VkResult vkResetCommandBuffer(
    VkCommandBuffer                             commandBuffer,
    VkCommandBufferResetFlags                   flags);
```

- `commandBuffer` is the command buffer to reset. The command buffer **can** be in any state other than pending, and is moved into the initial state.
- `flags` is a bitmask of VkCommandBufferResetFlagBits controlling the reset operation.

Any primary command buffer that is in the recording or executable state and has `commandBuffer` recorded into it, becomes invalid.

### Valid Usage

- VUID-vkResetCommandBuffer-commandBuffer-00045
  `commandBuffer` **must** not be in the pending state
- VUID-vkResetCommandBuffer-commandBuffer-00046
  `commandBuffer` **must** have been allocated from a pool that was created with the `VK_COMMAND_POOL_CREATE_RESET_COMMAND_BUFFER_BIT`

### Valid Usage (Implicit)

- VUID-vkResetCommandBuffer-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkResetCommandBuffer-flags-parameter
  `flags` **must** be a valid combination of VkCommandBufferResetFlagBits values

### Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

### Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- `VK_ERROR_OUT_OF_DEVICE_MEMORY`

Bits which **can** be set in vkResetCommandBuffer::`flags`, controlling the reset operation, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkCommandBufferResetFlagBits {
    VK_COMMAND_BUFFER_RESET_RELEASE_RESOURCES_BIT = 0x00000001,
} VkCommandBufferResetFlagBits;
```

- `VK_COMMAND_BUFFER_RESET_RELEASE_RESOURCES_BIT` specifies that most or all memory resources currently owned by the command buffer **should** be returned to the parent command pool. If this flag is not set, then the command buffer **may** hold onto memory resources and reuse them when recording commands. `commandBuffer` is moved to the initial state.

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkCommandBufferResetFlags;
```

`VkCommandBufferResetFlags` is a bitmask type for setting a mask of zero or more VkCommandBufferResetFlagBits.

To free command buffers, call:

```
// Provided by VK_VERSION_1_0
void vkFreeCommandBuffers(
    VkDevice                                    device,
    VkCommandPool                               commandPool,
    uint32_t                                    commandBufferCount,
    const VkCommandBuffer*                      pCommandBuffers);
```

- `device` is the logical device that owns the command pool.
- `commandPool` is the command pool from which the command buffers were allocated.
- `commandBufferCount` is the length of the `pCommandBuffers` array.
- `pCommandBuffers` is a pointer to an array of handles of command buffers to free.

Any primary command buffer that is in the recording or executable state and has any element of `pCommandBuffers` recorded into it, becomes invalid.

### Valid Usage

- VUID-vkFreeCommandBuffers-pCommandBuffers-00047
  All elements of `pCommandBuffers` **must** not be in the pending state
- VUID-vkFreeCommandBuffers-pCommandBuffers-00048
  `pCommandBuffers` **must** be a valid pointer to an array of `commandBufferCount` `VkCommandBuffer` handles, each element of which **must** either be a valid handle or `NULL`

### Valid Usage (Implicit)

- VUID-vkFreeCommandBuffers-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkFreeCommandBuffers-commandPool-parameter
  `commandPool` **must** be a valid VkCommandPool handle
- VUID-vkFreeCommandBuffers-commandBufferCount-arraylength
  `commandBufferCount` **must** be greater than `0`
- VUID-vkFreeCommandBuffers-commandPool-parent
  `commandPool` **must** have been created, allocated, or retrieved from `device`
- VUID-vkFreeCommandBuffers-pCommandBuffers-parent
  Each element of `pCommandBuffers` that is a valid handle **must** have been created, allocated, or retrieved from `commandPool`

### Host Synchronization

- Host access to `commandPool` **must** be externally synchronized
- Host access to each member of `pCommandBuffers` **must** be externally synchronized

## 6.4. Command Buffer Recording

To begin recording a command buffer, call:

```
// Provided by VK_VERSION_1_0
VkResult vkBeginCommandBuffer(
    VkCommandBuffer                             commandBuffer,
    const VkCommandBufferBeginInfo*             pBeginInfo);
```

- `commandBuffer` is the handle of the command buffer which is to be put in the recording state.
- `pBeginInfo` is a pointer to a VkCommandBufferBeginInfo structure defining additional information about how the command buffer begins recording.

### Valid Usage

- VUID-vkBeginCommandBuffer-commandBuffer-00049
  `commandBuffer` **must** not be in the recording or pending state
- VUID-vkBeginCommandBuffer-commandBuffer-00050
  If `commandBuffer` was allocated from a VkCommandPool which did not have the `VK_COMMAND_POOL_CREATE_RESET_COMMAND_BUFFER_BIT` flag set, `commandBuffer` **must** be in the initial state
- VUID-vkBeginCommandBuffer-commandBuffer-00051

If `commandBuffer` is a secondary command buffer, the `pInheritanceInfo` member of `pBeginInfo` **must** be a valid `VkCommandBufferInheritanceInfo` structure

- VUID-vkBeginCommandBuffer-commandBuffer-00052
  If `commandBuffer` is a secondary command buffer and either the `occlusionQueryEnable` member of the `pInheritanceInfo` member of `pBeginInfo` is `VK_FALSE`, or the `occlusionQueryPrecise` feature is not enabled, then `pBeginInfo->pInheritanceInfo->queryFlags` **must** not contain `VK_QUERY_CONTROL_PRECISE_BIT`
- VUID-vkBeginCommandBuffer-commandBuffer-02840
  If `commandBuffer` is a primary command buffer, then `pBeginInfo->flags` **must** not set both the `VK_COMMAND_BUFFER_USAGE_ONE_TIME_SUBMIT_BIT` and the `VK_COMMAND_BUFFER_USAGE_SIMULTANEOUS_USE_BIT` flags

## Valid Usage (Implicit)

- VUID-vkBeginCommandBuffer-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkBeginCommandBuffer-pBeginInfo-parameter
  `pBeginInfo` **must** be a valid pointer to a valid VkCommandBufferBeginInfo structure

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`

The `VkCommandBufferBeginInfo` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkCommandBufferBeginInfo {
    VkStructureType                         sType;
    const void*                             pNext;
    VkCommandBufferUsageFlags               flags;
    const VkCommandBufferInheritanceInfo*   pInheritanceInfo;
} VkCommandBufferBeginInfo;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is a bitmask of VkCommandBufferUsageFlagBits specifying usage behavior for the command buffer.
- `pInheritanceInfo` is a pointer to a VkCommandBufferInheritanceInfo structure, used if `commandBuffer` is a secondary command buffer. If this is a primary command buffer, then this value is ignored.

## Valid Usage

- VUID-VkCommandBufferBeginInfo-flags-09123
  If `flags` contains `VK_COMMAND_BUFFER_USAGE_RENDER_PASS_CONTINUE_BIT`, the VkCommandPool that `commandBuffer` was allocated from **must** support graphics operations
- VUID-VkCommandBufferBeginInfo-flags-00055
  If `flags` contains `VK_COMMAND_BUFFER_USAGE_RENDER_PASS_CONTINUE_BIT`, the `framebuffer` member of `pInheritanceInfo` **must** be either VK_NULL_HANDLE, or a valid `VkFramebuffer` that is compatible with the `renderPass` member of `pInheritanceInfo`
- VUID-VkCommandBufferBeginInfo-flags-09240
  If `flags` contains `VK_COMMAND_BUFFER_USAGE_RENDER_PASS_CONTINUE_BIT` and the `dynamicRendering` feature is not enabled, the `renderPass` member of `pInheritanceInfo` **must** not be VK_NULL_HANDLE
- VUID-VkCommandBufferBeginInfo-flags-06002
  If `flags` contains `VK_COMMAND_BUFFER_USAGE_RENDER_PASS_CONTINUE_BIT` and the `renderPass` member of `pInheritanceInfo` is VK_NULL_HANDLE, the `pNext` chain of `pInheritanceInfo` **must** include a VkCommandBufferInheritanceRenderingInfo structure
- VUID-VkCommandBufferBeginInfo-flags-06003
  If `flags` contains `VK_COMMAND_BUFFER_USAGE_RENDER_PASS_CONTINUE_BIT`, the `renderPass` member of `pInheritanceInfo` is VK_NULL_HANDLE, and the `pNext` chain of `pInheritanceInfo` includes a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, the `colorAttachmentCount` member of that structure **must** be equal to the value of VkCommandBufferInheritanceRenderingInfo::`colorAttachmentCount`
- VUID-VkCommandBufferBeginInfo-flags-06000
  If `flags` contains `VK_COMMAND_BUFFER_USAGE_RENDER_PASS_CONTINUE_BIT` and the `renderPass` member of `pInheritanceInfo` is not VK_NULL_HANDLE, the `renderPass` member of `pInheritanceInfo` **must** be a valid `VkRenderPass`

- VUID-VkCommandBufferBeginInfo-flags-06001
  If `flags` contains `VK_COMMAND_BUFFER_USAGE_RENDER_PASS_CONTINUE_BIT` and the `renderPass` member of `pInheritanceInfo` is not VK_NULL_HANDLE, the `subpass` member of `pInheritanceInfo` **must** be a valid subpass index within the `renderPass` member of `pInheritanceInfo`

### Valid Usage (Implicit)

- VUID-VkCommandBufferBeginInfo-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO`
- VUID-VkCommandBufferBeginInfo-pNext-pNext
  `pNext` **must** be `NULL` or a pointer to a valid instance of VkDeviceGroupCommandBufferBeginInfo
- VUID-VkCommandBufferBeginInfo-sType-unique
  The `sType` value of each struct in the `pNext` chain **must** be unique
- VUID-VkCommandBufferBeginInfo-flags-parameter
  `flags` **must** be a valid combination of VkCommandBufferUsageFlagBits values

Bits which **can** be set in VkCommandBufferBeginInfo::`flags`, specifying usage behavior for a command buffer, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkCommandBufferUsageFlagBits {
    VK_COMMAND_BUFFER_USAGE_ONE_TIME_SUBMIT_BIT = 0x00000001,
    VK_COMMAND_BUFFER_USAGE_RENDER_PASS_CONTINUE_BIT = 0x00000002,
    VK_COMMAND_BUFFER_USAGE_SIMULTANEOUS_USE_BIT = 0x00000004,
} VkCommandBufferUsageFlagBits;
```

- `VK_COMMAND_BUFFER_USAGE_ONE_TIME_SUBMIT_BIT` specifies that each recording of the command buffer will only be submitted once, and the command buffer will be reset and recorded again between each submission.
- `VK_COMMAND_BUFFER_USAGE_RENDER_PASS_CONTINUE_BIT` specifies that a secondary command buffer is considered to be entirely inside a render pass. If this is a primary command buffer, then this bit is ignored.
- `VK_COMMAND_BUFFER_USAGE_SIMULTANEOUS_USE_BIT` specifies that a command buffer **can** be resubmitted to any queue of the same queue family while it is in the *pending state*, and recorded into multiple primary command buffers.

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkCommandBufferUsageFlags;
```

`VkCommandBufferUsageFlags` is a bitmask type for setting a mask of zero or more VkCommandBufferUsageFlagBits.

If the command buffer is a secondary command buffer, then the VkCommandBufferInheritanceInfo structure defines any state that will be inherited from the primary command buffer:

```
// Provided by VK_VERSION_1_0
typedef struct VkCommandBufferInheritanceInfo {
    VkStructureType                 sType;
    const void*                     pNext;
    VkRenderPass                    renderPass;
    uint32_t                        subpass;
    VkFramebuffer                   framebuffer;
    VkBool32                        occlusionQueryEnable;
    VkQueryControlFlags             queryFlags;
    VkQueryPipelineStatisticFlags   pipelineStatistics;
} VkCommandBufferInheritanceInfo;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- renderPass is a VkRenderPass object defining which render passes the VkCommandBuffer will be compatible with and **can** be executed within.
- subpass is the index of the subpass within the render pass instance that the VkCommandBuffer will be executed within.
- framebuffer **can** refer to the VkFramebuffer object that the VkCommandBuffer will be rendering to if it is executed within a render pass instance. It **can** be VK_NULL_HANDLE if the framebuffer is not known.

| | |
|---|---|
| **NOTE** | Specifying the exact framebuffer that the secondary command buffer will be executed with **may** result in better performance at command buffer execution time. |

- occlusionQueryEnable specifies whether the command buffer **can** be executed while an occlusion query is active in the primary command buffer. If this is VK_TRUE, then this command buffer **can** be executed whether the primary command buffer has an occlusion query active or not. If this is VK_FALSE, then the primary command buffer **must** not have an occlusion query active.
- queryFlags specifies the query flags that **can** be used by an active occlusion query in the primary command buffer when this secondary command buffer is executed. If this value includes the VK_QUERY_CONTROL_PRECISE_BIT bit, then the active query **can** return boolean results or actual sample counts. If this bit is not set, then the active query **must** not use the VK_QUERY_CONTROL_PRECISE_BIT bit.
- pipelineStatistics is a bitmask of VkQueryPipelineStatisticFlagBits specifying the set of pipeline statistics that **can** be counted by an active query in the primary command buffer when this secondary command buffer is executed. If this value includes a given bit, then this command buffer **can** be executed whether the primary command buffer has a pipeline statistics query active that includes this bit or not. If this value excludes a given bit, then the active pipeline statistics query **must** not be from a query pool that counts that statistic.

If the VkCommandBuffer will not be executed within a render pass instance, or if the render pass instance was begun with vkCmdBeginRenderingKHR, `renderPass`, `subpass`, and `framebuffer` are ignored.

### Valid Usage

- VUID-VkCommandBufferInheritanceInfo-occlusionQueryEnable-00056
  If the `inheritedQueries` feature is not enabled, `occlusionQueryEnable` **must** be `VK_FALSE`
- VUID-VkCommandBufferInheritanceInfo-queryFlags-00057
  If the `inheritedQueries` feature is enabled, `queryFlags` **must** be a valid combination of VkQueryControlFlagBits values
- VUID-VkCommandBufferInheritanceInfo-queryFlags-02788
  If the `inheritedQueries` feature is not enabled, `queryFlags` **must** be `0`
- VUID-VkCommandBufferInheritanceInfo-pipelineStatistics-02789
  If the `pipelineStatisticsQuery` feature is enabled, `pipelineStatistics` **must** be a valid combination of VkQueryPipelineStatisticFlagBits values
- VUID-VkCommandBufferInheritanceInfo-pipelineStatistics-00058
  If the `pipelineStatisticsQuery` feature is not enabled, `pipelineStatistics` **must** be `0`

### Valid Usage (Implicit)

- VUID-VkCommandBufferInheritanceInfo-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_COMMAND_BUFFER_INHERITANCE_INFO`
- VUID-VkCommandBufferInheritanceInfo-pNext-pNext
  Each `pNext` member of any structure (including this one) in the `pNext` chain **must** be either `NULL` or a pointer to a valid instance of VkAttachmentSampleCountInfoAMD, VkCommandBufferInheritanceConditionalRenderingInfoEXT, VkCommandBufferInheritanceRenderPassTransformInfoQCOM, VkCommandBufferInheritanceRenderingInfo, VkCommandBufferInheritanceViewportScissorInfoNV, VkExternalFormatANDROID, VkMultiviewPerViewAttributesInfoNVX, VkRenderingAttachmentLocationInfoKHR, or VkRenderingInputAttachmentIndexInfoKHR
- VUID-VkCommandBufferInheritanceInfo-sType-unique
  The `sType` value of each struct in the `pNext` chain **must** be unique
- VUID-VkCommandBufferInheritanceInfo-commonparent
  Both of `framebuffer`, and `renderPass` that are valid handles of non-ignored parameters **must** have been created, allocated, or retrieved from the same VkDevice

| | |
|---|---|
| **NOTE** | On some implementations, not using the `VK_COMMAND_BUFFER_USAGE_SIMULTANEOUS_USE_BIT` bit enables command buffers to be patched in-place if needed, rather than creating a copy of the command buffer. |

If a command buffer is in the invalid, or executable state, and the command buffer was allocated

from a command pool with the VK_COMMAND_POOL_CREATE_RESET_COMMAND_BUFFER_BIT flag set, then vkBeginCommandBuffer implicitly resets the command buffer, behaving as if vkResetCommandBuffer had been called with VK_COMMAND_BUFFER_RESET_RELEASE_RESOURCES_BIT not set. After the implicit reset, commandBuffer is moved to the recording state.

If the commandBufferInheritance feature is enabled, all graphics and compute state including bound pipeline state, bound shader objects, bound vertex and index buffers, bound descriptor sets and push constants, and all previously set dynamic state is inherited by the secondary command buffer from the primary or secondary command buffer that executes it. Furthermore, all of the state set by this secondary command buffer is inherited back to the primary or secondard command buffer that executes it. If the commandBufferInheritance feature is not enabled there is a limited amount of inheritance of state into the secondary command buffer as specified below.

If the pNext chain of VkCommandBufferInheritanceInfo includes a VkCommandBufferInheritanceConditionalRenderingInfoEXT structure, then that structure controls whether a command buffer **can** be executed while conditional rendering is active in the primary command buffer.

The VkCommandBufferInheritanceConditionalRenderingInfoEXT structure is defined as:

```
// Provided by VK_EXT_conditional_rendering
typedef struct VkCommandBufferInheritanceConditionalRenderingInfoEXT {
    VkStructureType    sType;
    const void*        pNext;
    VkBool32           conditionalRenderingEnable;
} VkCommandBufferInheritanceConditionalRenderingInfoEXT;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- conditionalRenderingEnable specifies whether the command buffer **can** be executed while conditional rendering is active in the primary command buffer. If this is VK_TRUE, then this command buffer **can** be executed whether the primary command buffer has active conditional rendering or not. If this is VK_FALSE, then the primary command buffer **must** not have conditional rendering active.

If this structure is not present, the behavior is as if conditionalRenderingEnable is VK_FALSE.

### Valid Usage

- VUID-VkCommandBufferInheritanceConditionalRenderingInfoEXT-conditionalRenderingEnable-01977
  If the inheritedConditionalRendering feature is not enabled, conditionalRenderingEnable **must** be VK_FALSE

### Valid Usage (Implicit)

- VUID-VkCommandBufferInheritanceConditionalRenderingInfoEXT-sType-sType `sType` **must** be `VK_STRUCTURE_TYPE_COMMAND_BUFFER_INHERITANCE_CONDITIONAL_RENDERING_INFO_EXT`

To begin recording a secondary command buffer compatible with execution inside a render pass using render pass transform, add the VkCommandBufferInheritanceRenderPassTransformInfoQCOM to the `pNext` chain of VkCommandBufferInheritanceInfo structure passed to the vkBeginCommandBuffer command specifying the parameters for transformed rasterization.

The `VkCommandBufferInheritanceRenderPassTransformInfoQCOM` structure is defined as:

```
// Provided by VK_QCOM_render_pass_transform
typedef struct VkCommandBufferInheritanceRenderPassTransformInfoQCOM {
    VkStructureType                 sType;
    void*                           pNext;
    VkSurfaceTransformFlagBitsKHR   transform;
    VkRect2D                        renderArea;
} VkCommandBufferInheritanceRenderPassTransformInfoQCOM;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `transform` is a VkSurfaceTransformFlagBitsKHR value describing the transform to be applied to the render pass.
- `renderArea` is the render area that is affected by the command buffer.

When the secondary is recorded to execute within a render pass instance using vkCmdExecuteCommands, the render pass transform parameters of the secondary command buffer **must** be consistent with the render pass transform parameters specified for the render pass instance. In particular, the `transform` and `renderArea` for command buffer **must** be identical to the `transform` and `renderArea` of the render pass instance.

### Valid Usage

- VUID-VkCommandBufferInheritanceRenderPassTransformInfoQCOM-transform-02864 `transform` **must** be `VK_SURFACE_TRANSFORM_IDENTITY_BIT_KHR`, `VK_SURFACE_TRANSFORM_ROTATE_90_BIT_KHR`, `VK_SURFACE_TRANSFORM_ROTATE_180_BIT_KHR`, or `VK_SURFACE_TRANSFORM_ROTATE_270_BIT_KHR`

### Valid Usage (Implicit)

- VUID-VkCommandBufferInheritanceRenderPassTransformInfoQCOM-sType-sType

sType **must** be VK_STRUCTURE_TYPE_COMMAND_BUFFER_INHERITANCE_RENDER_PASS_TRANSFORM_INFO_QCOM

The VkCommandBufferInheritanceViewportScissorInfoNV structure is defined as:

```
// Provided by VK_NV_inherited_viewport_scissor
typedef struct VkCommandBufferInheritanceViewportScissorInfoNV {
    VkStructureType      sType;
    const void*          pNext;
    VkBool32             viewportScissor2D;
    uint32_t             viewportDepthCount;
    const VkViewport*    pViewportDepths;
} VkCommandBufferInheritanceViewportScissorInfoNV;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- viewportScissor2D specifies whether the listed dynamic state is inherited.
- viewportDepthCount specifies the maximum number of viewports to inherit. When viewportScissor2D is VK_FALSE, the behavior is as if this value is zero.
- pViewportDepths is a pointer to a VkViewport structure specifying the expected depth range for each inherited viewport.

If the pNext chain of VkCommandBufferInheritanceInfo includes a VkCommandBufferInheritanceViewportScissorInfoNV structure, then that structure controls whether a command buffer **can** inherit the following state from other command buffers:

- VK_DYNAMIC_STATE_SCISSOR
- VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT
- VK_DYNAMIC_STATE_DISCARD_RECTANGLE_EXT
- VK_DYNAMIC_STATE_DISCARD_RECTANGLE_ENABLE_EXT
- VK_DYNAMIC_STATE_DISCARD_RECTANGLE_MODE_EXT

as well as the following state, with restrictions on inherited depth values and viewport count:

- VK_DYNAMIC_STATE_VIEWPORT
- VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT

If viewportScissor2D is VK_FALSE, then the command buffer does not inherit the listed dynamic state, and **should** set this state itself. If this structure is not present, the behavior is as if viewportScissor2D is VK_FALSE.

If viewportScissor2D is VK_TRUE, then the listed dynamic state is inherited, and the command buffer **must** not set this state, except that the viewport and scissor count **may** be set by binding a graphics pipeline that does not specify this state as dynamic.

| | |
|---|---|
| **NOTE** | Due to this restriction, applications **should** ensure either all or none of the graphics pipelines bound in this secondary command buffer use dynamic viewport/scissor counts. |

When the command buffer is executed as part of a the execution of a vkCmdExecuteCommands command, the inherited state (if enabled) is determined by the following procedure, performed separately for each dynamic state, and separately for each value for dynamic state that consists of multiple values (e.g. multiple viewports).

- With i being the index of the executed command buffer in the `pCommandBuffers` array of vkCmdExecuteCommands, if i > 0 and any secondary command buffer from index 0 to i-1 modifies the state, the inherited state is provisionally set to the final value set by the last such secondary command buffer. Binding a graphics pipeline defining the state statically is equivalent to setting the state to an undefined value.
- Otherwise, the tentatative inherited state is that of the primary command buffer at the point the vkCmdExecuteCommands command was recorded; if the state is undefined, then so is the provisional inherited state.
- If the provisional inherited state is an undefined value, then the state is not inherited.
- If the provisional inherited state is a viewport, with n being its viewport index, then if n ≥ `viewportDepthCount`, or if either VkViewport::`minDepth` or VkViewport::`maxDepth` are not equal to the respective values of the $n^{th}$ element of `pViewportDepths`, then the state is not inherited.
- If the provisional inherited state passes both checks, then it becomes the actual inherited state.

| | |
|---|---|
| **NOTE** | There is no support for inheriting dynamic state from a secondary command buffer executed as part of a different `vkCmdExecuteCommands` command. |

## Valid Usage

- VUID-VkCommandBufferInheritanceViewportScissorInfoNV-viewportScissor2D-04782
  If the `inheritedViewportScissor2D` feature is not enabled, `viewportScissor2D` **must** be `VK_FALSE`
- VUID-VkCommandBufferInheritanceViewportScissorInfoNV-viewportScissor2D-04783
  If the `multiViewport` feature is not enabled and `viewportScissor2D` is `VK_TRUE`, then `viewportDepthCount` **must** be `1`
- VUID-VkCommandBufferInheritanceViewportScissorInfoNV-viewportScissor2D-04784
  If `viewportScissor2D` is `VK_TRUE`, then `viewportDepthCount` **must** be greater than `0`
- VUID-VkCommandBufferInheritanceViewportScissorInfoNV-viewportScissor2D-04785
  If `viewportScissor2D` is `VK_TRUE`, then `pViewportDepths` **must** be a valid pointer to an array of `viewportDepthCount` valid `VkViewport` structures, except any requirements on `x`, `y`, `width`, and `height` do not apply
- VUID-VkCommandBufferInheritanceViewportScissorInfoNV-viewportScissor2D-04786
  If `viewportScissor2D` is `VK_TRUE`, then the command buffer **must** be recorded with the `VK_COMMAND_BUFFER_USAGE_RENDER_PASS_CONTINUE_BIT`

### Valid Usage (Implicit)

- VUID-VkCommandBufferInheritanceViewportScissorInfoNV-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_COMMAND_BUFFER_INHERITANCE_VIEWPORT_SCISSOR_INFO_NV`

The `VkCommandBufferInheritanceRenderingInfo` structure is defined as:

```
typedef struct VkCommandBufferInheritanceRenderingInfo {
    VkStructureType          sType;
    const void*              pNext;
    VkRenderingFlags         flags;
    uint32_t                 viewMask;
    uint32_t                 colorAttachmentCount;
    const VkFormat*          pColorAttachmentFormats;
    VkFormat                 depthAttachmentFormat;
    VkFormat                 stencilAttachmentFormat;
    VkSampleCountFlagBits    rasterizationSamples;
} VkCommandBufferInheritanceRenderingInfo;
```

or the equivalent

```
// Provided by VK_KHR_dynamic_rendering
typedef VkCommandBufferInheritanceRenderingInfo
VkCommandBufferInheritanceRenderingInfoKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure
- `flags` is a bitmask of VkRenderingFlagBits used by the render pass instance.
- `viewMask` is the view mask used for rendering.
- `colorAttachmentCount` is the number of color attachments specified in the render pass instance.
- `pColorAttachmentFormats` is a pointer to an array of VkFormat values defining the format of color attachments.
- `depthAttachmentFormat` is a VkFormat value defining the format of the depth attachment.
- `stencilAttachmentFormat` is a VkFormat value defining the format of the stencil attachment.
- `rasterizationSamples` is a VkSampleCountFlagBits specifying the number of samples used in rasterization.

If the `pNext` chain of VkCommandBufferInheritanceInfo includes a `VkCommandBufferInheritanceRenderingInfo` structure, then that structure controls parameters of dynamic render pass instances that the VkCommandBuffer **can** be executed within. If VkCommandBufferInheritanceInfo::`renderPass` is not VK_NULL_HANDLE, or `VK_COMMAND_BUFFER_USAGE_RENDER_PASS_CONTINUE_BIT` is not specified in VkCommandBufferBeginInfo::`flags`, parameters of this structure are ignored.

If colorAttachmentCount is 0 and the variableMultisampleRate feature is enabled, rasterizationSamples is ignored.

If depthAttachmentFormat, stencilAttachmentFormat, or any element of pColorAttachmentFormats is VK_FORMAT_UNDEFINED, it indicates that the corresponding attachment is unused within the render pass and writes to those attachments are discarded.

## Valid Usage

- VUID-VkCommandBufferInheritanceRenderingInfo-colorAttachmentCount-06004
  If colorAttachmentCount is not 0, rasterizationSamples **must** be a valid VkSampleCountFlagBits value
- VUID-VkCommandBufferInheritanceRenderingInfo-variableMultisampleRate-06005
  If the variableMultisampleRate feature is not enabled, rasterizationSamples **must** be a valid VkSampleCountFlagBits value
- VUID-VkCommandBufferInheritanceRenderingInfo-depthAttachmentFormat-06540
  If depthAttachmentFormat is not VK_FORMAT_UNDEFINED, it **must** be a format that includes a depth component
- VUID-VkCommandBufferInheritanceRenderingInfo-depthAttachmentFormat-06007
  If depthAttachmentFormat is not VK_FORMAT_UNDEFINED, it **must** be a format with potential format features that include VK_FORMAT_FEATURE_DEPTH_STENCIL_ATTACHMENT_BIT
- VUID-VkCommandBufferInheritanceRenderingInfo-pColorAttachmentFormats-06492
  If any element of pColorAttachmentFormats is not VK_FORMAT_UNDEFINED, it **must** be a format with potential format features that include VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BIT , or VK_FORMAT_FEATURE_2_LINEAR_COLOR_ATTACHMENT_BIT_NV if the linearColorAttachment feature is enabled
- VUID-VkCommandBufferInheritanceRenderingInfo-stencilAttachmentFormat-06541
  If stencilAttachmentFormat is not VK_FORMAT_UNDEFINED, it **must** be a format that includes a stencil aspect
- VUID-VkCommandBufferInheritanceRenderingInfo-stencilAttachmentFormat-06199
  If stencilAttachmentFormat is not VK_FORMAT_UNDEFINED, it **must** be a format with potential format features that include VK_FORMAT_FEATURE_DEPTH_STENCIL_ATTACHMENT_BIT
- VUID-VkCommandBufferInheritanceRenderingInfo-depthAttachmentFormat-06200
  If depthAttachmentFormat is not VK_FORMAT_UNDEFINED and stencilAttachmentFormat is not VK_FORMAT_UNDEFINED, depthAttachmentFormat **must** equal stencilAttachmentFormat
- VUID-VkCommandBufferInheritanceRenderingInfo-multiview-06008
  If the multiview feature is not enabled, viewMask **must** be 0
- VUID-VkCommandBufferInheritanceRenderingInfo-viewMask-06009
  The index of the most significant bit in viewMask **must** be less than maxMultiviewViewCount

## Valid Usage (Implicit)

- VUID-VkCommandBufferInheritanceRenderingInfo-sType-sType

sType **must** be VK_STRUCTURE_TYPE_COMMAND_BUFFER_INHERITANCE_RENDERING_INFO

- VUID-VkCommandBufferInheritanceRenderingInfo-flags-parameter
  flags **must** be a valid combination of VkRenderingFlagBits values
- VUID-VkCommandBufferInheritanceRenderingInfo-pColorAttachmentFormats-parameter
  If colorAttachmentCount is not 0, pColorAttachmentFormats **must** be a valid pointer to an array of colorAttachmentCount valid VkFormat values
- VUID-VkCommandBufferInheritanceRenderingInfo-depthAttachmentFormat-parameter
  depthAttachmentFormat **must** be a valid VkFormat value
- VUID-VkCommandBufferInheritanceRenderingInfo-stencilAttachmentFormat-parameter
  stencilAttachmentFormat **must** be a valid VkFormat value
- VUID-VkCommandBufferInheritanceRenderingInfo-rasterizationSamples-parameter
  If rasterizationSamples is not 0, rasterizationSamples **must** be a valid VkSampleCountFlagBits value

The VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure is defined as:

```
// Provided by VK_KHR_dynamic_rendering with VK_AMD_mixed_attachment_samples
typedef struct VkAttachmentSampleCountInfoAMD {
    VkStructureType                 sType;
    const void*                     pNext;
    uint32_t                        colorAttachmentCount;
    const VkSampleCountFlagBits*    pColorAttachmentSamples;
    VkSampleCountFlagBits           depthStencilAttachmentSamples;
} VkAttachmentSampleCountInfoAMD;
```

or the equivalent

```
// Provided by VK_KHR_dynamic_rendering with VK_NV_framebuffer_mixed_samples
typedef VkAttachmentSampleCountInfoAMD VkAttachmentSampleCountInfoNV;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure
- colorAttachmentCount is the number of color attachments specified in a render pass instance.
- pColorAttachmentSamples is a pointer to an array of VkSampleCountFlagBits values defining the sample count of color attachments.
- depthStencilAttachmentSamples is a VkSampleCountFlagBits value defining the sample count of a depth/stencil attachment.

If VkCommandBufferInheritanceInfo::renderPass is VK_NULL_HANDLE, VK_COMMAND_BUFFER_USAGE_RENDER_PASS_CONTINUE_BIT is specified in VkCommandBufferBeginInfo::flags, and the pNext chain of VkCommandBufferInheritanceInfo includes VkAttachmentSampleCountInfoAMD, then this structure defines the sample counts of each attachment within the render pass instance. If VkAttachmentSampleCountInfoAMD is not included, the value of

VkCommandBufferInheritanceRenderingInfo::rasterizationSamples is used as the sample count for each attachment. If VkCommandBufferInheritanceInfo::renderPass is not VK_NULL_HANDLE, or VK_COMMAND_BUFFER_USAGE_RENDER_PASS_CONTINUE_BIT is not specified in VkCommandBufferBeginInfo::flags, parameters of this structure are ignored.

VkAttachmentSampleCountInfoAMD **can** also be included in the pNext chain of VkGraphicsPipelineCreateInfo. When a graphics pipeline is created without a VkRenderPass, if this structure is included in the pNext chain of VkGraphicsPipelineCreateInfo, it specifies the sample count of attachments used for rendering. If this structure is not specified, and the pipeline does not include a VkRenderPass, the value of VkPipelineMultisampleStateCreateInfo::rasterizationSamples is used as the sample count for each attachment. If a graphics pipeline is created with a valid VkRenderPass, parameters of this structure are ignored.

### Valid Usage (Implicit)

- VUID-VkAttachmentSampleCountInfoAMD-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_ATTACHMENT_SAMPLE_COUNT_INFO_AMD

Once recording starts, an application records a sequence of commands (vkCmd*) to set state in the command buffer, draw, dispatch, and other commands.

Several commands can also be recorded indirectly from VkBuffer content, see Device-Generated Commands.

To complete recording of a command buffer, call:

```
// Provided by VK_VERSION_1_0
VkResult vkEndCommandBuffer(
    VkCommandBuffer                             commandBuffer);
```

- commandBuffer is the command buffer to complete recording.

The command buffer **must** have been in the recording state, and, if successful, is moved to the executable state.

If there was an error during recording, the application will be notified by an unsuccessful return code returned by vkEndCommandBuffer, and the command buffer will be moved to the invalid state.

In case the application recorded one or more video encode operations into the command buffer, implementations **may** return the VK_ERROR_INVALID_VIDEO_STD_PARAMETERS_KHR error if any of the specified Video Std parameters do not adhere to the syntactic or semantic requirements of the used video compression standard, or if values derived from parameters according to the rules defined by the used video compression standard do not adhere to the capabilities of the video compression standard or the implementation.

**NOTE** Applications **should** not rely on the VK_ERROR_INVALID_VIDEO_STD_PARAMETERS_KHR error being returned by any command as a means to verify Video Std parameters,

as implementations are not required to report the error in any specific set of cases.

## Valid Usage

- VUID-vkEndCommandBuffer-commandBuffer-00059
  `commandBuffer` **must** be in the recording state
- VUID-vkEndCommandBuffer-commandBuffer-00060
  If `commandBuffer` is a primary command buffer, there **must** not be an active render pass instance
- VUID-vkEndCommandBuffer-commandBuffer-00061
  All queries made active during the recording of `commandBuffer` **must** have been made inactive
- VUID-vkEndCommandBuffer-None-01978
  Conditional rendering **must** not be active
- VUID-vkEndCommandBuffer-None-06991
  There **must** be no video session object bound
- VUID-vkEndCommandBuffer-commandBuffer-01815
  If `commandBuffer` is a secondary command buffer, there **must** not be an outstanding vkCmdBeginDebugUtilsLabelEXT command recorded to `commandBuffer` that has not previously been ended by a call to vkCmdEndDebugUtilsLabelEXT
- VUID-vkEndCommandBuffer-commandBuffer-00062
  If `commandBuffer` is a secondary command buffer, there **must** not be an outstanding vkCmdDebugMarkerBeginEXT command recorded to `commandBuffer` that has not previously been ended by a call to vkCmdDebugMarkerEndEXT

## Valid Usage (Implicit)

- VUID-vkEndCommandBuffer-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY
- VK_ERROR_INVALID_VIDEO_STD_PARAMETERS_KHR

When a command buffer is in the executable state, it **can** be submitted to a queue for execution.

## 6.5. Command Buffer Submission

| | |
|---|---|
| **NOTE** | Submission can be a high overhead operation, and applications **should** attempt to batch work together into as few calls to vkQueueSubmit or vkQueueSubmit2KHR as possible. |

To submit command buffers to a queue, call:

```
// Provided by VK_KHR_synchronization2
VkResult vkQueueSubmit2KHR(
    VkQueue                                     queue,
    uint32_t                                    submitCount,
    const VkSubmitInfo2*                        pSubmits,
    VkFence                                     fence);
```

- queue is the queue that the command buffers will be submitted to.
- submitCount is the number of elements in the pSubmits array.
- pSubmits is a pointer to an array of VkSubmitInfo2 structures, each specifying a command buffer submission batch.
- fence is an **optional** handle to a fence to be signaled once all submitted command buffers have completed execution. If fence is not VK_NULL_HANDLE, it defines a fence signal operation.

vkQueueSubmit2KHR is a queue submission command, with each batch defined by an element of pSubmits.

Semaphore operations submitted with vkQueueSubmit2KHR have additional ordering constraints compared to other submission commands, with dependencies involving previous and subsequent queue operations. Information about these additional constraints can be found in the semaphore section of the synchronization chapter.

If any command buffer submitted to this queue is in the executable state, it is moved to the pending state. Once execution of all submissions of a command buffer complete, it moves from the pending state, back to the executable state. If a command buffer was recorded with the VK_COMMAND_BUFFER_USAGE_ONE_TIME_SUBMIT_BIT flag, it instead moves back to the invalid state.

If vkQueueSubmit2KHR fails, it **may** return VK_ERROR_OUT_OF_HOST_MEMORY or VK_ERROR_OUT_OF_DEVICE_MEMORY. If it does, the implementation **must** ensure that the state and contents of any resources or synchronization primitives referenced by the submitted command

buffers and any semaphores referenced by `pSubmits` is unaffected by the call or its failure. If `vkQueueSubmit2KHR` fails in such a way that the implementation is unable to make that guarantee, the implementation **must** return `VK_ERROR_DEVICE_LOST`. See Lost Device.

### Valid Usage

- VUID-vkQueueSubmit2-fence-04894
  If `fence` is not VK_NULL_HANDLE, `fence` **must** be unsignaled
- VUID-vkQueueSubmit2-fence-04895
  If `fence` is not VK_NULL_HANDLE, `fence` **must** not be associated with any other queue command that has not yet completed execution on that queue
- VUID-vkQueueSubmit2-synchronization2-03866
  The `synchronization2` feature **must** be enabled
- VUID-vkQueueSubmit2-commandBuffer-03867
  If a command recorded into the `commandBuffer` member of any element of the `pCommandBufferInfos` member of any element of `pSubmits` referenced a VkEvent, that event **must** not be referenced by a command that has been submitted to another queue and is still in the *pending state*
- VUID-vkQueueSubmit2-semaphore-03868
  The `semaphore` member of any binary semaphore element of the `pSignalSemaphoreInfos` member of any element of `pSubmits` **must** be unsignaled when the semaphore signal operation it defines is executed on the device
- VUID-vkQueueSubmit2-stageMask-03869
  The `stageMask` member of any element of the `pSignalSemaphoreInfos` member of any element of `pSubmits` **must** only include pipeline stages that are supported by the queue family which `queue` belongs to
- VUID-vkQueueSubmit2-stageMask-03870
  The `stageMask` member of any element of the `pWaitSemaphoreInfos` member of any element of `pSubmits` **must** only include pipeline stages that are supported by the queue family which `queue` belongs to
- VUID-vkQueueSubmit2-semaphore-03871
  When a semaphore wait operation for a binary semaphore is executed, as defined by the `semaphore` member of any element of the `pWaitSemaphoreInfos` member of any element of `pSubmits`, there **must** be no other queues waiting on the same semaphore
- VUID-vkQueueSubmit2-semaphore-03873
  The `semaphore` member of any element of the `pWaitSemaphoreInfos` member of any element of `pSubmits` that was created with a VkSemaphoreTypeKHR of `VK_SEMAPHORE_TYPE_BINARY_KHR` **must** reference a semaphore signal operation that has been submitted for execution and any semaphore signal operations on which it depends **must** have also been submitted for execution
- VUID-vkQueueSubmit2-commandBuffer-03874
  The `commandBuffer` member of any element of the `pCommandBufferInfos` member of any element of `pSubmits` **must** be in the pending or executable state

- VUID-vkQueueSubmit2-commandBuffer-03875
  If a command recorded into the `commandBuffer` member of any element of the `pCommandBufferInfos` member of any element of `pSubmits` was not recorded with the `VK_COMMAND_BUFFER_USAGE_SIMULTANEOUS_USE_BIT`, it **must** not be in the pending state
- VUID-vkQueueSubmit2-commandBuffer-03876
  Any secondary command buffers recorded into the `commandBuffer` member of any element of the `pCommandBufferInfos` member of any element of `pSubmits` **must** be in the pending or executable state
- VUID-vkQueueSubmit2-commandBuffer-03877
  If any secondary command buffers recorded into the `commandBuffer` member of any element of the `pCommandBufferInfos` member of any element of `pSubmits` was not recorded with the `VK_COMMAND_BUFFER_USAGE_SIMULTANEOUS_USE_BIT`, it **must** not be in the pending state
- VUID-vkQueueSubmit2-commandBuffer-03878
  The `commandBuffer` member of any element of the `pCommandBufferInfos` member of any element of `pSubmits` **must** have been allocated from a `VkCommandPool` that was created for the same queue family `queue` belongs to
- VUID-vkQueueSubmit2-commandBuffer-03879
  If a command recorded into the `commandBuffer` member of any element of the `pCommandBufferInfos` member of any element of `pSubmits` includes a Queue Family Ownership Transfer Acquire Operation, there **must** exist a previously submitted Queue Family Ownership Transfer Release Operation on a queue in the queue family identified by the acquire operation, with parameters matching the acquire operation as defined in the definition of such acquire operations, and which happens before the acquire operation
- VUID-vkQueueSubmit2-commandBuffer-03880
  If a command recorded into the `commandBuffer` member of any element of the `pCommandBufferInfos` member of any element of `pSubmits` was a vkCmdBeginQuery whose `queryPool` was created with a `queryType` of `VK_QUERY_TYPE_PERFORMANCE_QUERY_KHR`, the profiling lock **must** have been held continuously on the `VkDevice` that `queue` was retrieved from, throughout recording of those command buffers
- VUID-vkQueueSubmit2-queue-06447
  If `queue` was not created with `VK_DEVICE_QUEUE_CREATE_PROTECTED_BIT`, the `flags` member of any element of `pSubmits` **must** not include `VK_SUBMIT_PROTECTED_BIT_KHR`

## Valid Usage (Implicit)

- VUID-vkQueueSubmit2-queue-parameter
  `queue` **must** be a valid VkQueue handle
- VUID-vkQueueSubmit2-pSubmits-parameter
  If `submitCount` is not `0`, `pSubmits` **must** be a valid pointer to an array of `submitCount` valid VkSubmitInfo2 structures
- VUID-vkQueueSubmit2-fence-parameter

If fence is not VK_NULL_HANDLE, fence **must** be a valid VkFence handle

- VUID-vkQueueSubmit2-commonparent
  Both of fence, and queue that are valid handles of non-ignored parameters **must** have been created, allocated, or retrieved from the same VkDevice

## Host Synchronization

- Host access to queue **must** be externally synchronized
- Host access to fence **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| - | - | - | Any | - |

## Return Codes

**Success**

- VK_SUCCESS

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY
- VK_ERROR_DEVICE_LOST

The VkSubmitInfo2 structure is defined as:

```
typedef struct VkSubmitInfo2 {
    VkStructureType                     sType;
    const void*                         pNext;
    VkSubmitFlags                       flags;
    uint32_t                            waitSemaphoreInfoCount;
    const VkSemaphoreSubmitInfo*        pWaitSemaphoreInfos;
    uint32_t                            commandBufferInfoCount;
    const VkCommandBufferSubmitInfo*    pCommandBufferInfos;
    uint32_t                            signalSemaphoreInfoCount;
    const VkSemaphoreSubmitInfo*        pSignalSemaphoreInfos;
} VkSubmitInfo2;
```

or the equivalent

```
// Provided by VK_KHR_synchronization2
typedef VkSubmitInfo2 VkSubmitInfo2KHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is a bitmask of VkSubmitFlagBits.
- `waitSemaphoreInfoCount` is the number of elements in `pWaitSemaphoreInfos`.
- `pWaitSemaphoreInfos` is a pointer to an array of VkSemaphoreSubmitInfo structures defining semaphore wait operations.
- `commandBufferInfoCount` is the number of elements in `pCommandBufferInfos` and the number of command buffers to execute in the batch.
- `pCommandBufferInfos` is a pointer to an array of VkCommandBufferSubmitInfo structures describing command buffers to execute in the batch.
- `signalSemaphoreInfoCount` is the number of elements in `pSignalSemaphoreInfos`.
- `pSignalSemaphoreInfos` is a pointer to an array of VkSemaphoreSubmitInfo describing semaphore signal operations.

## Valid Usage

- VUID-VkSubmitInfo2-semaphore-03881
  If the same semaphore is used as the `semaphore` member of both an element of `pSignalSemaphoreInfos` and `pWaitSemaphoreInfos`, and that semaphore is a timeline semaphore, the `value` member of the `pSignalSemaphoreInfos` element **must** be greater than the `value` member of the `pWaitSemaphoreInfos` element
- VUID-VkSubmitInfo2-semaphore-03882
  If the `semaphore` member of any element of `pSignalSemaphoreInfos` is a timeline semaphore, the `value` member of that element **must** have a value greater than the current value of the semaphore when the semaphore signal operation is executed
- VUID-VkSubmitInfo2-semaphore-03883
  If the `semaphore` member of any element of `pSignalSemaphoreInfos` is a timeline semaphore, the `value` member of that element **must** have a value which does not differ from the current value of the semaphore or the value of any outstanding semaphore wait or signal operation on that semaphore by more than `maxTimelineSemaphoreValueDifference`
- VUID-VkSubmitInfo2-semaphore-03884
  If the `semaphore` member of any element of `pWaitSemaphoreInfos` is a timeline semaphore, the `value` member of that element **must** have a value which does not differ from the current value of the semaphore or the value of any outstanding semaphore wait or signal operation on that semaphore by more than `maxTimelineSemaphoreValueDifference`
- VUID-VkSubmitInfo2-flags-03886
  If `flags` includes `VK_SUBMIT_PROTECTED_BIT`, all elements of `pCommandBuffers` **must** be protected command buffers

- VUID-VkSubmitInfo2-flags-03887
  If `flags` does not include `VK_SUBMIT_PROTECTED_BIT`, each element of `pCommandBuffers` **must** not be a protected command buffer
- VUID-VkSubmitInfo2KHR-commandBuffer-06192
  If any `commandBuffer` member of an element of `pCommandBufferInfos` contains any resumed render pass instances, they **must** be suspended by a render pass instance earlier in submission order within `pCommandBufferInfos`
- VUID-VkSubmitInfo2KHR-commandBuffer-06010
  If any `commandBuffer` member of an element of `pCommandBufferInfos` contains any suspended render pass instances, they **must** be resumed by a render pass instance later in submission order within `pCommandBufferInfos`
- VUID-VkSubmitInfo2KHR-commandBuffer-06011
  If any `commandBuffer` member of an element of `pCommandBufferInfos` contains any suspended render pass instances, there **must** be no action or synchronization commands between that render pass instance and the render pass instance that resumes it
- VUID-VkSubmitInfo2KHR-commandBuffer-06012
  If any `commandBuffer` member of an element of `pCommandBufferInfos` contains any suspended render pass instances, there **must** be no render pass instances between that render pass instance and the render pass instance that resumes it
- VUID-VkSubmitInfo2KHR-variableSampleLocations-06013
  If the `variableSampleLocations` limit is not supported, and any `commandBuffer` member of an element of `pCommandBufferInfos` contains any suspended render pass instances, where a graphics pipeline has been bound, any pipelines bound in the render pass instance that resumes it, or any subsequent render pass instances that resume from that one and so on, **must** use the same sample locations

## Valid Usage (Implicit)

- VUID-VkSubmitInfo2-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_SUBMIT_INFO_2`
- VUID-VkSubmitInfo2-pNext-pNext
  Each `pNext` member of any structure (including this one) in the `pNext` chain **must** be either `NULL` or a pointer to a valid instance of VkFrameBoundaryEXT, VkLatencySubmissionPresentIdNV, VkPerformanceQuerySubmitInfoKHR, VkWin32KeyedMutexAcquireReleaseInfoKHR, or VkWin32KeyedMutexAcquireReleaseInfoNV
- VUID-VkSubmitInfo2-sType-unique
  The `sType` value of each struct in the `pNext` chain **must** be unique
- VUID-VkSubmitInfo2-flags-parameter
  `flags` **must** be a valid combination of VkSubmitFlagBits values
- VUID-VkSubmitInfo2-pWaitSemaphoreInfos-parameter
  If `waitSemaphoreInfoCount` is not `0`, `pWaitSemaphoreInfos` **must** be a valid pointer to an array of `waitSemaphoreInfoCount` valid VkSemaphoreSubmitInfo structures

- VUID-VkSubmitInfo2-pCommandBufferInfos-parameter
  If `commandBufferInfoCount` is not `0`, `pCommandBufferInfos` **must** be a valid pointer to an array of `commandBufferInfoCount` valid VkCommandBufferSubmitInfo structures
- VUID-VkSubmitInfo2-pSignalSemaphoreInfos-parameter
  If `signalSemaphoreInfoCount` is not `0`, `pSignalSemaphoreInfos` **must** be a valid pointer to an array of `signalSemaphoreInfoCount` valid VkSemaphoreSubmitInfo structures

Bits which **can** be set in VkSubmitInfo2::`flags`, specifying submission behavior, are:

```
typedef enum VkSubmitFlagBits {
    VK_SUBMIT_PROTECTED_BIT = 0x00000001,
    VK_SUBMIT_PROTECTED_BIT_KHR = VK_SUBMIT_PROTECTED_BIT,
} VkSubmitFlagBits;
```

or the equivalent

```
// Provided by VK_KHR_synchronization2
typedef VkSubmitFlagBits VkSubmitFlagBitsKHR;
```

- `VK_SUBMIT_PROTECTED_BIT` specifies that this batch is a protected submission.

```
typedef VkFlags VkSubmitFlags;
```

or the equivalent

```
// Provided by VK_KHR_synchronization2
typedef VkSubmitFlags VkSubmitFlagsKHR;
```

`VkSubmitFlags` is a bitmask type for setting a mask of zero or more VkSubmitFlagBits.

The `VkSemaphoreSubmitInfo` structure is defined as:

```
typedef struct VkSemaphoreSubmitInfo {
    VkStructureType          sType;
    const void*              pNext;
    VkSemaphore              semaphore;
    uint64_t                 value;
    VkPipelineStageFlags2    stageMask;
    uint32_t                 deviceIndex;
} VkSemaphoreSubmitInfo;
```

or the equivalent

```
// Provided by VK_KHR_synchronization2
typedef VkSemaphoreSubmitInfo VkSemaphoreSubmitInfoKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `semaphore` is a VkSemaphore affected by this operation.
- `value` is either the value used to signal `semaphore` or the value waited on by `semaphore`, if `semaphore` is a timeline semaphore. Otherwise it is ignored.
- `stageMask` is a VkPipelineStageFlags2 mask of pipeline stages which limit the first synchronization scope of a semaphore signal operation, or second synchronization scope of a semaphore wait operation as described in the semaphore wait operation and semaphore signal operation sections of the synchronization chapter.
- `deviceIndex` is the index of the device within a device group that executes the semaphore wait or signal operation.

Whether this structure defines a semaphore wait or signal operation is defined by how it is used.

### Valid Usage

- VUID-VkSemaphoreSubmitInfo-stageMask-03929
  If the `geometryShader` feature is not enabled, `stageMask` **must** not contain `VK_PIPELINE_STAGE_2_GEOMETRY_SHADER_BIT`
- VUID-VkSemaphoreSubmitInfo-stageMask-03930
  If the `tessellationShader` feature is not enabled, `stageMask` **must** not contain `VK_PIPELINE_STAGE_2_TESSELLATION_CONTROL_SHADER_BIT` or `VK_PIPELINE_STAGE_2_TESSELLATION_EVALUATION_SHADER_BIT`
- VUID-VkSemaphoreSubmitInfo-stageMask-03931
  If the `conditionalRendering` feature is not enabled, `stageMask` **must** not contain `VK_PIPELINE_STAGE_2_CONDITIONAL_RENDERING_BIT_EXT`
- VUID-VkSemaphoreSubmitInfo-stageMask-03932
  If the `fragmentDensityMap` feature is not enabled, `stageMask` **must** not contain `VK_PIPELINE_STAGE_2_FRAGMENT_DENSITY_PROCESS_BIT_EXT`
- VUID-VkSemaphoreSubmitInfo-stageMask-03933
  If the `transformFeedback` feature is not enabled, `stageMask` **must** not contain `VK_PIPELINE_STAGE_2_TRANSFORM_FEEDBACK_BIT_EXT`
- VUID-VkSemaphoreSubmitInfo-stageMask-03934
  If the `meshShader` feature is not enabled, `stageMask` **must** not contain `VK_PIPELINE_STAGE_2_MESH_SHADER_BIT_EXT`
- VUID-VkSemaphoreSubmitInfo-stageMask-03935
  If the `taskShader` feature is not enabled, `stageMask` **must** not contain `VK_PIPELINE_STAGE_2_TASK_SHADER_BIT_EXT`
- VUID-VkSemaphoreSubmitInfo-stageMask-07316

If neither the shadingRateImage or attachmentFragmentShadingRate are enabled, stageMask **must** not contain VK_PIPELINE_STAGE_2_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR

- VUID-VkSemaphoreSubmitInfo-stageMask-04957
  If the subpassShading feature is not enabled, stageMask **must** not contain VK_PIPELINE_STAGE_2_SUBPASS_SHADER_BIT_HUAWEI
- VUID-VkSemaphoreSubmitInfo-stageMask-04995
  If the invocationMask feature is not enabled, stageMask **must** not contain VK_PIPELINE_STAGE_2_INVOCATION_MASK_BIT_HUAWEI
- VUID-VkSemaphoreSubmitInfo-stageMask-07946
  If neither the VK_NV_ray_tracing extension or rayTracingPipeline feature are enabled, stageMask **must** not contain VK_PIPELINE_STAGE_2_RAY_TRACING_SHADER_BIT_KHR
- VUID-VkSemaphoreSubmitInfo-device-03888
  If the device that semaphore was created on is not a device group, deviceIndex **must** be 0
- VUID-VkSemaphoreSubmitInfo-device-03889
  If the device that semaphore was created on is a device group, deviceIndex **must** be a valid device index

### Valid Usage (Implicit)

- VUID-VkSemaphoreSubmitInfo-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_SEMAPHORE_SUBMIT_INFO
- VUID-VkSemaphoreSubmitInfo-pNext-pNext
  pNext **must** be NULL
- VUID-VkSemaphoreSubmitInfo-semaphore-parameter
  semaphore **must** be a valid VkSemaphore handle
- VUID-VkSemaphoreSubmitInfo-stageMask-parameter
  stageMask **must** be a valid combination of VkPipelineStageFlagBits2 values

The VkCommandBufferSubmitInfo structure is defined as:

```
typedef struct VkCommandBufferSubmitInfo {
    VkStructureType    sType;
    const void*        pNext;
    VkCommandBuffer    commandBuffer;
    uint32_t           deviceMask;
} VkCommandBufferSubmitInfo;
```

or the equivalent

```
// Provided by VK_KHR_synchronization2
typedef VkCommandBufferSubmitInfo VkCommandBufferSubmitInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- commandBuffer is a VkCommandBuffer to be submitted for execution.
- deviceMask is a bitmask indicating which devices in a device group execute the command buffer. A deviceMask of 0 is equivalent to setting all bits corresponding to valid devices in the group to 1.

### Valid Usage

- VUID-VkCommandBufferSubmitInfo-commandBuffer-03890
  commandBuffer **must** not have been allocated with VK_COMMAND_BUFFER_LEVEL_SECONDARY
- VUID-VkCommandBufferSubmitInfo-deviceMask-03891
  If deviceMask is not 0, it **must** be a valid device mask
- VUID-VkCommandBufferSubmitInfo-commandBuffer-09445
  If any render pass instance in commandBuffer was recorded with a VkRenderPassStripeBeginInfoARM structure in its pNext chain and did not specify the VK_RENDERING_RESUMING_BIT flag, a VkRenderPassStripeSubmitInfoARM **must** be included in the pNext chain
- VUID-VkCommandBufferSubmitInfo-pNext-09446
  If a VkRenderPassStripeSubmitInfoARM is included in the pNext chain, the value of VkRenderPassStripeSubmitInfoARM::stripeSemaphoreInfoCount **must** be equal to the sum of the VkRenderPassStripeBeginInfoARM::stripeInfoCount parameters provided to render pass instances recorded in commandBuffer that did not specify the VK_RENDERING_RESUMING_BIT flag

### Valid Usage (Implicit)

- VUID-VkCommandBufferSubmitInfo-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_COMMAND_BUFFER_SUBMIT_INFO
- VUID-VkCommandBufferSubmitInfo-pNext-pNext
  pNext **must** be NULL or a pointer to a valid instance of VkRenderPassStripeSubmitInfoARM
- VUID-VkCommandBufferSubmitInfo-sType-unique
  The sType value of each struct in the pNext chain **must** be unique
- VUID-VkCommandBufferSubmitInfo-commandBuffer-parameter
  commandBuffer **must** be a valid VkCommandBuffer handle

The VkRenderPassStripeSubmitInfoARM structure is defined as:

```
// Provided by VK_ARM_render_pass_striped
typedef struct VkRenderPassStripeSubmitInfoARM {
    VkStructureType                 sType;
    const void*                     pNext;
    uint32_t                        stripeSemaphoreInfoCount;
    const VkSemaphoreSubmitInfo*    pStripeSemaphoreInfos;
} VkRenderPassStripeSubmitInfoARM;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `stripeSemaphoreInfoCount` is the number of semaphores used to signal stripe completion in the render pass instances in the submitted command buffer.
- `pStripeSemaphoreInfos` is a pointer to an array of `stripeSemaphoreInfoCount` VkSemaphoreSubmitInfo structures describing the semaphores used to signal stripe completion.

This structure can be included in the `pNext` chain of VkCommandBufferSubmitInfo to provide a set of semaphores to be signaled for each striped render pass instance.

The elements of `pStripeSemaphoreInfos` are mapped to render pass instances in VkCommandBufferSubmitInfo::`commandBuffer` in submission order and in stripe order within each render pass instance. Each semaphore in `pStripeSemaphoreInfos` is signaled when the implementation has completed execution of the associated stripe. In a render pass instance that has multiview enabled, the stripe includes all views in the view mask. In a render pass instance with `layerCount` greater than 1, the stripe includes all layers.

Render pass instances that specify the `VK_RENDERING_RESUMING_BIT` will not have any elements of `pStripeSemaphoreInfos` mapped to them. Instead, for suspending and resuming render pass instances, this mapping is done for the first suspending render pass instance, and the per-stripe semaphores are only signaled for the last resuming render pass instance.

### Valid Usage

- VUID-VkRenderPassStripeSubmitInfoARM-semaphore-09447
  The `semaphore` member of each element of `pStripeSemaphoreInfos` **must** have been created with a VkSemaphoreType of `VK_SEMAPHORE_TYPE_BINARY`

### Valid Usage (Implicit)

- VUID-VkRenderPassStripeSubmitInfoARM-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_RENDER_PASS_STRIPE_SUBMIT_INFO_ARM`
- VUID-VkRenderPassStripeSubmitInfoARM-pStripeSemaphoreInfos-parameter
  `pStripeSemaphoreInfos` **must** be a valid pointer to an array of `stripeSemaphoreInfoCount` valid VkSemaphoreSubmitInfo structures

- VUID-VkRenderPassStripeSubmitInfoARM-stripeSemaphoreInfoCount-arraylength
  `stripeSemaphoreInfoCount` **must** be greater than `0`

To submit command buffers to a queue, call:

```
// Provided by VK_VERSION_1_0
VkResult vkQueueSubmit(
    VkQueue                                     queue,
    uint32_t                                    submitCount,
    const VkSubmitInfo*                         pSubmits,
    VkFence                                     fence);
```

- `queue` is the queue that the command buffers will be submitted to.
- `submitCount` is the number of elements in the `pSubmits` array.
- `pSubmits` is a pointer to an array of VkSubmitInfo structures, each specifying a command buffer submission batch.
- `fence` is an **optional** handle to a fence to be signaled once all submitted command buffers have completed execution. If `fence` is not VK_NULL_HANDLE, it defines a fence signal operation.

`vkQueueSubmit` is a queue submission command, with each batch defined by an element of `pSubmits`. Batches begin execution in the order they appear in `pSubmits`, but **may** complete out of order.

Fence and semaphore operations submitted with vkQueueSubmit have additional ordering constraints compared to other submission commands, with dependencies involving previous and subsequent queue operations. Information about these additional constraints can be found in the semaphore and fence sections of the synchronization chapter.

Details on the interaction of `pWaitDstStageMask` with synchronization are described in the semaphore wait operation section of the synchronization chapter.

The order that batches appear in `pSubmits` is used to determine submission order, and thus all the implicit ordering guarantees that respect it. Other than these implicit ordering guarantees and any explicit synchronization primitives, these batches **may** overlap or otherwise execute out of order.

If any command buffer submitted to this queue is in the executable state, it is moved to the pending state. Once execution of all submissions of a command buffer complete, it moves from the pending state, back to the executable state. If a command buffer was recorded with the `VK_COMMAND_BUFFER_USAGE_ONE_TIME_SUBMIT_BIT` flag, it instead moves to the invalid state.

If `vkQueueSubmit` fails, it **may** return `VK_ERROR_OUT_OF_HOST_MEMORY` or `VK_ERROR_OUT_OF_DEVICE_MEMORY`. If it does, the implementation **must** ensure that the state and contents of any resources or synchronization primitives referenced by the submitted command buffers and any semaphores referenced by `pSubmits` is unaffected by the call or its failure. If `vkQueueSubmit` fails in such a way that the implementation is unable to make that guarantee, the implementation **must** return `VK_ERROR_DEVICE_LOST`. See Lost Device.

## Valid Usage

- VUID-vkQueueSubmit-fence-00063
  If `fence` is not VK_NULL_HANDLE, `fence` **must** be unsignaled
- VUID-vkQueueSubmit-fence-00064
  If `fence` is not VK_NULL_HANDLE, `fence` **must** not be associated with any other queue command that has not yet completed execution on that queue
- VUID-vkQueueSubmit-pCommandBuffers-00065
  Any calls to vkCmdSetEvent, vkCmdResetEvent or vkCmdWaitEvents that have been recorded into any of the command buffer elements of the `pCommandBuffers` member of any element of `pSubmits`, **must** not reference any VkEvent that is referenced by any of those commands in a command buffer that has been submitted to another queue and is still in the *pending state*
- VUID-vkQueueSubmit-pWaitDstStageMask-00066
  Any stage flag included in any element of the `pWaitDstStageMask` member of any element of `pSubmits` **must** be a pipeline stage supported by one of the capabilities of `queue`, as specified in the table of supported pipeline stages
- VUID-vkQueueSubmit-pSignalSemaphores-00067
  Each binary semaphore element of the `pSignalSemaphores` member of any element of `pSubmits` **must** be unsignaled when the semaphore signal operation it defines is executed on the device
- VUID-vkQueueSubmit-pWaitSemaphores-00068
  When a semaphore wait operation referring to a binary semaphore defined by any element of the `pWaitSemaphores` member of any element of `pSubmits` executes on `queue`, there **must** be no other queues waiting on the same semaphore
- VUID-vkQueueSubmit-pWaitSemaphores-03238
  All elements of the `pWaitSemaphores` member of all elements of `pSubmits` created with a VkSemaphoreType of `VK_SEMAPHORE_TYPE_BINARY` **must** reference a semaphore signal operation that has been submitted for execution and any semaphore signal operations on which it depends **must** have also been submitted for execution
- VUID-vkQueueSubmit-pCommandBuffers-00070
  Each element of the `pCommandBuffers` member of each element of `pSubmits` **must** be in the pending or executable state
- VUID-vkQueueSubmit-pCommandBuffers-00071
  If any element of the `pCommandBuffers` member of any element of `pSubmits` was not recorded with the `VK_COMMAND_BUFFER_USAGE_SIMULTANEOUS_USE_BIT`, it **must** not be in the pending state
- VUID-vkQueueSubmit-pCommandBuffers-00072
  Any secondary command buffers recorded into any element of the `pCommandBuffers` member of any element of `pSubmits` **must** be in the pending or executable state
- VUID-vkQueueSubmit-pCommandBuffers-00073
  If any secondary command buffers recorded into any element of the `pCommandBuffers` member of any element of `pSubmits` was not recorded with the

VK_COMMAND_BUFFER_USAGE_SIMULTANEOUS_USE_BIT, it **must** not be in the pending state

- VUID-vkQueueSubmit-pCommandBuffers-00074
  Each element of the pCommandBuffers member of each element of pSubmits **must** have been allocated from a VkCommandPool that was created for the same queue family queue belongs to
- VUID-vkQueueSubmit-pSubmits-02207
  If any element of pSubmits->pCommandBuffers includes a Queue Family Ownership Transfer Acquire Operation, there **must** exist a previously submitted Queue Family Ownership Transfer Release Operation on a queue in the queue family identified by the acquire operation, with parameters matching the acquire operation as defined in the definition of such acquire operations, and which happens-before the acquire operation
- VUID-vkQueueSubmit-pCommandBuffers-03220
  If a command recorded into any element of pCommandBuffers was a vkCmdBeginQuery whose queryPool was created with a queryType of VK_QUERY_TYPE_PERFORMANCE_QUERY_KHR, the profiling lock **must** have been held continuously on the VkDevice that queue was retrieved from, throughout recording of those command buffers
- VUID-vkQueueSubmit-pSubmits-02808
  Any resource created with VK_SHARING_MODE_EXCLUSIVE that is read by an operation specified by pSubmits **must** not be owned by any queue family other than the one which queue belongs to, at the time it is executed
- VUID-vkQueueSubmit-pSubmits-04626
  Any resource created with VK_SHARING_MODE_CONCURRENT that is accessed by an operation specified by pSubmits **must** have included the queue family of queue at resource creation time
- VUID-vkQueueSubmit-queue-06448
  If queue was not created with VK_DEVICE_QUEUE_CREATE_PROTECTED_BIT, there **must** be no element of pSubmits that includes a VkProtectedSubmitInfo structure in its pNext chain with protectedSubmit equal to VK_TRUE

## Valid Usage (Implicit)

- VUID-vkQueueSubmit-queue-parameter
  queue **must** be a valid VkQueue handle
- VUID-vkQueueSubmit-pSubmits-parameter
  If submitCount is not 0, pSubmits **must** be a valid pointer to an array of submitCount valid VkSubmitInfo structures
- VUID-vkQueueSubmit-fence-parameter
  If fence is not VK_NULL_HANDLE, fence **must** be a valid VkFence handle
- VUID-vkQueueSubmit-commonparent
  Both of fence, and queue that are valid handles of non-ignored parameters **must** have been created, allocated, or retrieved from the same VkDevice

### Host Synchronization

- Host access to `queue` **must** be externally synchronized
- Host access to `fence` **must** be externally synchronized

### Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| - | - | - | Any | - |

### Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_DEVICE_LOST`

The `VkSubmitInfo` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkSubmitInfo {
    VkStructureType                 sType;
    const void*                     pNext;
    uint32_t                        waitSemaphoreCount;
    const VkSemaphore*              pWaitSemaphores;
    const VkPipelineStageFlags*     pWaitDstStageMask;
    uint32_t                        commandBufferCount;
    const VkCommandBuffer*          pCommandBuffers;
    uint32_t                        signalSemaphoreCount;
    const VkSemaphore*              pSignalSemaphores;
} VkSubmitInfo;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `waitSemaphoreCount` is the number of semaphores upon which to wait before executing the command buffers for the batch.
- `pWaitSemaphores` is a pointer to an array of VkSemaphore handles upon which to wait before the

command buffers for this batch begin execution. If semaphores to wait on are provided, they define a semaphore wait operation.

- pWaitDstStageMask is a pointer to an array of pipeline stages at which each corresponding semaphore wait will occur.
- commandBufferCount is the number of command buffers to execute in the batch.
- pCommandBuffers is a pointer to an array of VkCommandBuffer handles to execute in the batch.
- signalSemaphoreCount is the number of semaphores to be signaled once the commands specified in pCommandBuffers have completed execution.
- pSignalSemaphores is a pointer to an array of VkSemaphore handles which will be signaled when the command buffers for this batch have completed execution. If semaphores to be signaled are provided, they define a semaphore signal operation.

The order that command buffers appear in pCommandBuffers is used to determine submission order, and thus all the implicit ordering guarantees that respect it. Other than these implicit ordering guarantees and any explicit synchronization primitives, these command buffers **may** overlap or otherwise execute out of order.

### Valid Usage

- VUID-VkSubmitInfo-pWaitDstStageMask-04090
  If the geometryShader feature is not enabled, pWaitDstStageMask **must** not contain VK_PIPELINE_STAGE_GEOMETRY_SHADER_BIT
- VUID-VkSubmitInfo-pWaitDstStageMask-04091
  If the tessellationShader feature is not enabled, pWaitDstStageMask **must** not contain VK_PIPELINE_STAGE_TESSELLATION_CONTROL_SHADER_BIT or VK_PIPELINE_STAGE_TESSELLATION_EVALUATION_SHADER_BIT
- VUID-VkSubmitInfo-pWaitDstStageMask-04092
  If the conditionalRendering feature is not enabled, pWaitDstStageMask **must** not contain VK_PIPELINE_STAGE_CONDITIONAL_RENDERING_BIT_EXT
- VUID-VkSubmitInfo-pWaitDstStageMask-04093
  If the fragmentDensityMap feature is not enabled, pWaitDstStageMask **must** not contain VK_PIPELINE_STAGE_FRAGMENT_DENSITY_PROCESS_BIT_EXT
- VUID-VkSubmitInfo-pWaitDstStageMask-04094
  If the transformFeedback feature is not enabled, pWaitDstStageMask **must** not contain VK_PIPELINE_STAGE_TRANSFORM_FEEDBACK_BIT_EXT
- VUID-VkSubmitInfo-pWaitDstStageMask-04095
  If the meshShader feature is not enabled, pWaitDstStageMask **must** not contain VK_PIPELINE_STAGE_MESH_SHADER_BIT_EXT
- VUID-VkSubmitInfo-pWaitDstStageMask-04096
  If the taskShader feature is not enabled, pWaitDstStageMask **must** not contain VK_PIPELINE_STAGE_TASK_SHADER_BIT_EXT
- VUID-VkSubmitInfo-pWaitDstStageMask-07318
  If neither the shadingRateImage or attachmentFragmentShadingRate are enabled,

pWaitDstStageMask **must** not contain VK_PIPELINE_STAGE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR

- VUID-VkSubmitInfo-pWaitDstStageMask-03937
  If the synchronization2 feature is not enabled, pWaitDstStageMask **must** not be 0
- VUID-VkSubmitInfo-pWaitDstStageMask-07949
  If neither the VK_NV_ray_tracing extension or rayTracingPipeline feature are enabled, pWaitDstStageMask **must** not contain VK_PIPELINE_STAGE_RAY_TRACING_SHADER_BIT_KHR
- VUID-VkSubmitInfo-pCommandBuffers-00075
  Each element of pCommandBuffers **must** not have been allocated with VK_COMMAND_BUFFER_LEVEL_SECONDARY
- VUID-VkSubmitInfo-pWaitDstStageMask-00078
  Each element of pWaitDstStageMask **must** not include VK_PIPELINE_STAGE_HOST_BIT
- VUID-VkSubmitInfo-pWaitSemaphores-03239
  If any element of pWaitSemaphores or pSignalSemaphores was created with a VkSemaphoreType of VK_SEMAPHORE_TYPE_TIMELINE, then the pNext chain **must** include a VkTimelineSemaphoreSubmitInfo structure
- VUID-VkSubmitInfo-pNext-03240
  If the pNext chain of this structure includes a VkTimelineSemaphoreSubmitInfo structure and any element of pWaitSemaphores was created with a VkSemaphoreType of VK_SEMAPHORE_TYPE_TIMELINE, then its waitSemaphoreValueCount member **must** equal waitSemaphoreCount
- VUID-VkSubmitInfo-pNext-03241
  If the pNext chain of this structure includes a VkTimelineSemaphoreSubmitInfo structure and any element of pSignalSemaphores was created with a VkSemaphoreType of VK_SEMAPHORE_TYPE_TIMELINE, then its signalSemaphoreValueCount member **must** equal signalSemaphoreCount
- VUID-VkSubmitInfo-pSignalSemaphores-03242
  For each element of pSignalSemaphores created with a VkSemaphoreType of VK_SEMAPHORE_TYPE_TIMELINE the corresponding element of VkTimelineSemaphoreSubmitInfo::pSignalSemaphoreValues **must** have a value greater than the current value of the semaphore when the semaphore signal operation is executed
- VUID-VkSubmitInfo-pWaitSemaphores-03243
  For each element of pWaitSemaphores created with a VkSemaphoreType of VK_SEMAPHORE_TYPE_TIMELINE the corresponding element of VkTimelineSemaphoreSubmitInfo::pWaitSemaphoreValues **must** have a value which does not differ from the current value of the semaphore or the value of any outstanding semaphore wait or signal operation on that semaphore by more than maxTimelineSemaphoreValueDifference
- VUID-VkSubmitInfo-pSignalSemaphores-03244
  For each element of pSignalSemaphores created with a VkSemaphoreType of VK_SEMAPHORE_TYPE_TIMELINE the corresponding element of VkTimelineSemaphoreSubmitInfo::pSignalSemaphoreValues **must** have a value which does not differ from the current value of the semaphore or the value of any outstanding

semaphore wait or signal operation on that semaphore by more than maxTimelineSemaphoreValueDifference

- VUID-VkSubmitInfo-pNext-04120
  If the pNext chain of this structure does not include a VkProtectedSubmitInfo structure with protectedSubmit set to VK_TRUE, then each element of the pCommandBuffers array **must** be an unprotected command buffer
- VUID-VkSubmitInfo-pNext-04148
  If the pNext chain of this structure includes a VkProtectedSubmitInfo structure with protectedSubmit set to VK_TRUE, then each element of the pCommandBuffers array **must** be a protected command buffer
- VUID-VkSubmitInfo-pCommandBuffers-06193
  If pCommandBuffers contains any resumed render pass instances, they **must** be suspended by a render pass instance earlier in submission order within pCommandBuffers
- VUID-VkSubmitInfo-pCommandBuffers-06014
  If pCommandBuffers contains any suspended render pass instances, they **must** be resumed by a render pass instance later in submission order within pCommandBuffers
- VUID-VkSubmitInfo-pCommandBuffers-06015
  If pCommandBuffers contains any suspended render pass instances, there **must** be no action or synchronization commands executed in a primary or secondary command buffer between that render pass instance and the render pass instance that resumes it
- VUID-VkSubmitInfo-pCommandBuffers-06016
  If pCommandBuffers contains any suspended render pass instances, there **must** be no render pass instances between that render pass instance and the render pass instance that resumes it
- VUID-VkSubmitInfo-variableSampleLocations-06017
  If the variableSampleLocations limit is not supported, and any element of pCommandBuffers contains any suspended render pass instances, where a graphics pipeline has been bound, any pipelines bound in the render pass instance that resumes it, or any subsequent render pass instances that resume from that one and so on, **must** use the same sample locations

## Valid Usage (Implicit)

- VUID-VkSubmitInfo-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_SUBMIT_INFO
- VUID-VkSubmitInfo-pNext-pNext
  Each pNext member of any structure (including this one) in the pNext chain **must** be either NULL or a pointer to a valid instance of VkAmigoProfilingSubmitInfoSEC, VkD3D12FenceSubmitInfoKHR, VkDeviceGroupSubmitInfo, VkFrameBoundaryEXT, VkLatencySubmissionPresentIdNV, VkPerformanceQuerySubmitInfoKHR, VkProtectedSubmitInfo, VkTimelineSemaphoreSubmitInfo, VkWin32KeyedMutexAcquireReleaseInfoKHR, or VkWin32KeyedMutexAcquireReleaseInfoNV

- VUID-VkSubmitInfo-sType-unique
  The `sType` value of each struct in the `pNext` chain **must** be unique
- VUID-VkSubmitInfo-pWaitSemaphores-parameter
  If `waitSemaphoreCount` is not `0`, `pWaitSemaphores` **must** be a valid pointer to an array of `waitSemaphoreCount` valid VkSemaphore handles
- VUID-VkSubmitInfo-pWaitDstStageMask-parameter
  If `waitSemaphoreCount` is not `0`, `pWaitDstStageMask` **must** be a valid pointer to an array of `waitSemaphoreCount` valid combinations of VkPipelineStageFlagBits values
- VUID-VkSubmitInfo-pCommandBuffers-parameter
  If `commandBufferCount` is not `0`, `pCommandBuffers` **must** be a valid pointer to an array of `commandBufferCount` valid VkCommandBuffer handles
- VUID-VkSubmitInfo-pSignalSemaphores-parameter
  If `signalSemaphoreCount` is not `0`, `pSignalSemaphores` **must** be a valid pointer to an array of `signalSemaphoreCount` valid VkSemaphore handles
- VUID-VkSubmitInfo-commonparent
  Each of the elements of `pCommandBuffers`, the elements of `pSignalSemaphores`, and the elements of `pWaitSemaphores` that are valid handles of non-ignored parameters **must** have been created, allocated, or retrieved from the same VkDevice

To specify the values to use when waiting for and signaling semaphores created with a VkSemaphoreType of `VK_SEMAPHORE_TYPE_TIMELINE`, add a VkTimelineSemaphoreSubmitInfo structure to the `pNext` chain of the VkSubmitInfo structure when using vkQueueSubmit or the VkBindSparseInfo structure when using vkQueueBindSparse . The `VkTimelineSemaphoreSubmitInfo` structure is defined as:

```
typedef struct VkTimelineSemaphoreSubmitInfo {
    VkStructureType    sType;
    const void*        pNext;
    uint32_t           waitSemaphoreValueCount;
    const uint64_t*    pWaitSemaphoreValues;
    uint32_t           signalSemaphoreValueCount;
    const uint64_t*    pSignalSemaphoreValues;
} VkTimelineSemaphoreSubmitInfo;
```

or the equivalent

```
// Provided by VK_KHR_timeline_semaphore
typedef VkTimelineSemaphoreSubmitInfo VkTimelineSemaphoreSubmitInfoKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `waitSemaphoreValueCount` is the number of semaphore wait values specified in `pWaitSemaphoreValues`.

- pWaitSemaphoreValues is a pointer to an array of waitSemaphoreValueCount values for the corresponding semaphores in VkSubmitInfo::pWaitSemaphores to wait for.
- signalSemaphoreValueCount is the number of semaphore signal values specified in pSignalSemaphoreValues.
- pSignalSemaphoreValues is a pointer to an array signalSemaphoreValueCount values for the corresponding semaphores in VkSubmitInfo::pSignalSemaphores to set when signaled.

If the semaphore in VkSubmitInfo::pWaitSemaphores or VkSubmitInfo::pSignalSemaphores corresponding to an entry in pWaitSemaphoreValues or pSignalSemaphoreValues respectively was not created with a VkSemaphoreType of VK_SEMAPHORE_TYPE_TIMELINE, the implementation **must** ignore the value in the pWaitSemaphoreValues or pSignalSemaphoreValues entry.

### Valid Usage (Implicit)

- VUID-VkTimelineSemaphoreSubmitInfo-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_TIMELINE_SEMAPHORE_SUBMIT_INFO
- VUID-VkTimelineSemaphoreSubmitInfo-pWaitSemaphoreValues-parameter
  If waitSemaphoreValueCount is not 0, and pWaitSemaphoreValues is not NULL, pWaitSemaphoreValues **must** be a valid pointer to an array of waitSemaphoreValueCount uint64_t values
- VUID-VkTimelineSemaphoreSubmitInfo-pSignalSemaphoreValues-parameter
  If signalSemaphoreValueCount is not 0, and pSignalSemaphoreValues is not NULL, pSignalSemaphoreValues **must** be a valid pointer to an array of signalSemaphoreValueCount uint64_t values

To specify the values to use when waiting for and signaling semaphores whose current payload refers to a Direct3D 12 fence, add a VkD3D12FenceSubmitInfoKHR structure to the pNext chain of the VkSubmitInfo structure. The VkD3D12FenceSubmitInfoKHR structure is defined as:

```
// Provided by VK_KHR_external_semaphore_win32
typedef struct VkD3D12FenceSubmitInfoKHR {
    VkStructureType    sType;
    const void*        pNext;
    uint32_t           waitSemaphoreValuesCount;
    const uint64_t*    pWaitSemaphoreValues;
    uint32_t           signalSemaphoreValuesCount;
    const uint64_t*    pSignalSemaphoreValues;
} VkD3D12FenceSubmitInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- waitSemaphoreValuesCount is the number of semaphore wait values specified in pWaitSemaphoreValues.
- pWaitSemaphoreValues is a pointer to an array of waitSemaphoreValuesCount values for the

corresponding semaphores in VkSubmitInfo::pWaitSemaphores to wait for.

- signalSemaphoreValuesCount is the number of semaphore signal values specified in pSignalSemaphoreValues.
- pSignalSemaphoreValues is a pointer to an array of signalSemaphoreValuesCount values for the corresponding semaphores in VkSubmitInfo::pSignalSemaphores to set when signaled.

If the semaphore in VkSubmitInfo::pWaitSemaphores or VkSubmitInfo::pSignalSemaphores corresponding to an entry in pWaitSemaphoreValues or pSignalSemaphoreValues respectively does not currently have a payload referring to a Direct3D 12 fence, the implementation **must** ignore the value in the pWaitSemaphoreValues or pSignalSemaphoreValues entry.

**NOTE**

As the introduction of the external semaphore handle type VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_D3D12_FENCE_BIT predates that of timeline semaphores, support for importing semaphore payloads from external handles of that type into semaphores created (implicitly or explicitly) with a VkSemaphoreType of VK_SEMAPHORE_TYPE_BINARY is preserved for backwards compatibility. However, applications **should** prefer importing such handle types into semaphores created with a VkSemaphoreType of VK_SEMAPHORE_TYPE_TIMELINE, and use the VkTimelineSemaphoreSubmitInfo structure instead of the VkD3D12FenceSubmitInfoKHR structure to specify the values to use when waiting for and signaling such semaphores.

## Valid Usage

- VUID-VkD3D12FenceSubmitInfoKHR-waitSemaphoreValuesCount-00079
  waitSemaphoreValuesCount **must** be the same value as VkSubmitInfo::waitSemaphoreCount, where this structure is in the pNext chain of a VkSubmitInfo structure
- VUID-VkD3D12FenceSubmitInfoKHR-signalSemaphoreValuesCount-00080
  signalSemaphoreValuesCount **must** be the same value as VkSubmitInfo::signalSemaphoreCount, where this structure is in the pNext chain of a VkSubmitInfo structure

## Valid Usage (Implicit)

- VUID-VkD3D12FenceSubmitInfoKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_D3D12_FENCE_SUBMIT_INFO_KHR
- VUID-VkD3D12FenceSubmitInfoKHR-pWaitSemaphoreValues-parameter
  If waitSemaphoreValuesCount is not 0, and pWaitSemaphoreValues is not NULL, pWaitSemaphoreValues **must** be a valid pointer to an array of waitSemaphoreValuesCount uint64_t values
- VUID-VkD3D12FenceSubmitInfoKHR-pSignalSemaphoreValues-parameter
  If signalSemaphoreValuesCount is not 0, and pSignalSemaphoreValues is not NULL, pSignalSemaphoreValues **must** be a valid pointer to an array of signalSemaphoreValuesCount uint64_t values

When submitting work that operates on memory imported from a Direct3D 11 resource to a queue, the keyed mutex mechanism **may** be used in addition to Vulkan semaphores to synchronize the work. Keyed mutexes are a property of a properly created shareable Direct3D 11 resource. They **can** only be used if the imported resource was created with the `D3D11_RESOURCE_MISC_SHARED_KEYEDMUTEX` flag.

To acquire keyed mutexes before submitted work and/or release them after, add a VkWin32KeyedMutexAcquireReleaseInfoKHR structure to the `pNext` chain of the VkSubmitInfo structure.

The `VkWin32KeyedMutexAcquireReleaseInfoKHR` structure is defined as:

```
// Provided by VK_KHR_win32_keyed_mutex
typedef struct VkWin32KeyedMutexAcquireReleaseInfoKHR {
    VkStructureType          sType;
    const void*              pNext;
    uint32_t                 acquireCount;
    const VkDeviceMemory*    pAcquireSyncs;
    const uint64_t*          pAcquireKeys;
    const uint32_t*          pAcquireTimeouts;
    uint32_t                 releaseCount;
    const VkDeviceMemory*    pReleaseSyncs;
    const uint64_t*          pReleaseKeys;
} VkWin32KeyedMutexAcquireReleaseInfoKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `acquireCount` is the number of entries in the `pAcquireSyncs`, `pAcquireKeys`, and `pAcquireTimeouts` arrays.
- `pAcquireSyncs` is a pointer to an array of VkDeviceMemory objects which were imported from Direct3D 11 resources.
- `pAcquireKeys` is a pointer to an array of mutex key values to wait for prior to beginning the submitted work. Entries refer to the keyed mutex associated with the corresponding entries in `pAcquireSyncs`.
- `pAcquireTimeouts` is a pointer to an array of timeout values, in millisecond units, for each acquire specified in `pAcquireKeys`.
- `releaseCount` is the number of entries in the `pReleaseSyncs` and `pReleaseKeys` arrays.
- `pReleaseSyncs` is a pointer to an array of VkDeviceMemory objects which were imported from Direct3D 11 resources.
- `pReleaseKeys` is a pointer to an array of mutex key values to set when the submitted work has completed. Entries refer to the keyed mutex associated with the corresponding entries in `pReleaseSyncs`.

### Valid Usage

- VUID-VkWin32KeyedMutexAcquireReleaseInfoKHR-pAcquireSyncs-00081
  Each member of `pAcquireSyncs` and `pReleaseSyncs` **must** be a device memory object imported by setting VkImportMemoryWin32HandleInfoKHR::`handleType` to `VK_EXTERNAL_MEMORY_HANDLE_TYPE_D3D11_TEXTURE_BIT` or `VK_EXTERNAL_MEMORY_HANDLE_TYPE_D3D11_TEXTURE_KMT_BIT`

### Valid Usage (Implicit)

- VUID-VkWin32KeyedMutexAcquireReleaseInfoKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_WIN32_KEYED_MUTEX_ACQUIRE_RELEASE_INFO_KHR`
- VUID-VkWin32KeyedMutexAcquireReleaseInfoKHR-pAcquireSyncs-parameter
  If `acquireCount` is not `0`, `pAcquireSyncs` **must** be a valid pointer to an array of `acquireCount` valid VkDeviceMemory handles
- VUID-VkWin32KeyedMutexAcquireReleaseInfoKHR-pAcquireKeys-parameter
  If `acquireCount` is not `0`, `pAcquireKeys` **must** be a valid pointer to an array of `acquireCount` `uint64_t` values
- VUID-VkWin32KeyedMutexAcquireReleaseInfoKHR-pAcquireTimeouts-parameter
  If `acquireCount` is not `0`, `pAcquireTimeouts` **must** be a valid pointer to an array of `acquireCount` `uint32_t` values
- VUID-VkWin32KeyedMutexAcquireReleaseInfoKHR-pReleaseSyncs-parameter
  If `releaseCount` is not `0`, `pReleaseSyncs` **must** be a valid pointer to an array of `releaseCount` valid VkDeviceMemory handles
- VUID-VkWin32KeyedMutexAcquireReleaseInfoKHR-pReleaseKeys-parameter
  If `releaseCount` is not `0`, `pReleaseKeys` **must** be a valid pointer to an array of `releaseCount` `uint64_t` values
- VUID-VkWin32KeyedMutexAcquireReleaseInfoKHR-commonparent
  Both of the elements of `pAcquireSyncs`, and the elements of `pReleaseSyncs` that are valid handles of non-ignored parameters **must** have been created, allocated, or retrieved from the same VkDevice

When submitting work that operates on memory imported from a Direct3D 11 resource to a queue, the keyed mutex mechanism **may** be used in addition to Vulkan semaphores to synchronize the work. Keyed mutexes are a property of a properly created shareable Direct3D 11 resource. They **can** only be used if the imported resource was created with the `D3D11_RESOURCE_MISC_SHARED_KEYEDMUTEX` flag.

To acquire keyed mutexes before submitted work and/or release them after, add a VkWin32KeyedMutexAcquireReleaseInfoNV structure to the `pNext` chain of the VkSubmitInfo structure.

The `VkWin32KeyedMutexAcquireReleaseInfoNV` structure is defined as:

```
// Provided by VK_NV_win32_keyed_mutex
typedef struct VkWin32KeyedMutexAcquireReleaseInfoNV {
    VkStructureType          sType;
    const void*              pNext;
    uint32_t                 acquireCount;
    const VkDeviceMemory*    pAcquireSyncs;
    const uint64_t*          pAcquireKeys;
    const uint32_t*          pAcquireTimeoutMilliseconds;
    uint32_t                 releaseCount;
    const VkDeviceMemory*    pReleaseSyncs;
    const uint64_t*          pReleaseKeys;
} VkWin32KeyedMutexAcquireReleaseInfoNV;
```

- `acquireCount` is the number of entries in the `pAcquireSyncs`, `pAcquireKeys`, and `pAcquireTimeoutMilliseconds` arrays.
- `pAcquireSyncs` is a pointer to an array of VkDeviceMemory objects which were imported from Direct3D 11 resources.
- `pAcquireKeys` is a pointer to an array of mutex key values to wait for prior to beginning the submitted work. Entries refer to the keyed mutex associated with the corresponding entries in `pAcquireSyncs`.
- `pAcquireTimeoutMilliseconds` is a pointer to an array of timeout values, in millisecond units, for each acquire specified in `pAcquireKeys`.
- `releaseCount` is the number of entries in the `pReleaseSyncs` and `pReleaseKeys` arrays.
- `pReleaseSyncs` is a pointer to an array of VkDeviceMemory objects which were imported from Direct3D 11 resources.
- `pReleaseKeys` is a pointer to an array of mutex key values to set when the submitted work has completed. Entries refer to the keyed mutex associated with the corresponding entries in `pReleaseSyncs`.

### Valid Usage (Implicit)

- VUID-VkWin32KeyedMutexAcquireReleaseInfoNV-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_WIN32_KEYED_MUTEX_ACQUIRE_RELEASE_INFO_NV`
- VUID-VkWin32KeyedMutexAcquireReleaseInfoNV-pAcquireSyncs-parameter
  If `acquireCount` is not `0`, `pAcquireSyncs` **must** be a valid pointer to an array of `acquireCount` valid VkDeviceMemory handles
- VUID-VkWin32KeyedMutexAcquireReleaseInfoNV-pAcquireKeys-parameter
  If `acquireCount` is not `0`, `pAcquireKeys` **must** be a valid pointer to an array of `acquireCount` `uint64_t` values
- VUID-VkWin32KeyedMutexAcquireReleaseInfoNV-pAcquireTimeoutMilliseconds-parameter
  If `acquireCount` is not `0`, `pAcquireTimeoutMilliseconds` **must** be a valid pointer to an array of `acquireCount` `uint32_t` values

- VUID-VkWin32KeyedMutexAcquireReleaseInfoNV-pReleaseSyncs-parameter
  If `releaseCount` is not `0`, `pReleaseSyncs` **must** be a valid pointer to an array of `releaseCount` valid VkDeviceMemory handles
- VUID-VkWin32KeyedMutexAcquireReleaseInfoNV-pReleaseKeys-parameter
  If `releaseCount` is not `0`, `pReleaseKeys` **must** be a valid pointer to an array of `releaseCount` `uint64_t` values
- VUID-VkWin32KeyedMutexAcquireReleaseInfoNV-commonparent
  Both of the elements of `pAcquireSyncs`, and the elements of `pReleaseSyncs` that are valid handles of non-ignored parameters **must** have been created, allocated, or retrieved from the same VkDevice

If the `pNext` chain of VkSubmitInfo includes a `VkProtectedSubmitInfo` structure, then the structure indicates whether the batch is protected. The `VkProtectedSubmitInfo` structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkProtectedSubmitInfo {
    VkStructureType    sType;
    const void*        pNext;
    VkBool32           protectedSubmit;
} VkProtectedSubmitInfo;
```

- `protectedSubmit` specifies whether the batch is protected. If `protectedSubmit` is `VK_TRUE`, the batch is protected. If `protectedSubmit` is `VK_FALSE`, the batch is unprotected. If the `VkSubmitInfo`::`pNext` chain does not include this structure, the batch is unprotected.

### Valid Usage (Implicit)

- VUID-VkProtectedSubmitInfo-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PROTECTED_SUBMIT_INFO`

If the `pNext` chain of VkSubmitInfo includes a `VkDeviceGroupSubmitInfo` structure, then that structure includes device indices and masks specifying which physical devices execute semaphore operations and command buffers.

The `VkDeviceGroupSubmitInfo` structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkDeviceGroupSubmitInfo {
    VkStructureType    sType;
    const void*        pNext;
    uint32_t           waitSemaphoreCount;
    const uint32_t*    pWaitSemaphoreDeviceIndices;
    uint32_t           commandBufferCount;
    const uint32_t*    pCommandBufferDeviceMasks;
    uint32_t           signalSemaphoreCount;
    const uint32_t*    pSignalSemaphoreDeviceIndices;
} VkDeviceGroupSubmitInfo;
```

or the equivalent

```
// Provided by VK_KHR_device_group
typedef VkDeviceGroupSubmitInfo VkDeviceGroupSubmitInfoKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `waitSemaphoreCount` is the number of elements in the `pWaitSemaphoreDeviceIndices` array.
- `pWaitSemaphoreDeviceIndices` is a pointer to an array of `waitSemaphoreCount` device indices indicating which physical device executes the semaphore wait operation in the corresponding element of VkSubmitInfo::`pWaitSemaphores`.
- `commandBufferCount` is the number of elements in the `pCommandBufferDeviceMasks` array.
- `pCommandBufferDeviceMasks` is a pointer to an array of `commandBufferCount` device masks indicating which physical devices execute the command buffer in the corresponding element of VkSubmitInfo::`pCommandBuffers`. A physical device executes the command buffer if the corresponding bit is set in the mask.
- `signalSemaphoreCount` is the number of elements in the `pSignalSemaphoreDeviceIndices` array.
- `pSignalSemaphoreDeviceIndices` is a pointer to an array of `signalSemaphoreCount` device indices indicating which physical device executes the semaphore signal operation in the corresponding element of VkSubmitInfo::`pSignalSemaphores`.

If this structure is not present, semaphore operations and command buffers execute on device index zero.

### Valid Usage

- VUID-VkDeviceGroupSubmitInfo-waitSemaphoreCount-00082
  `waitSemaphoreCount` **must** equal VkSubmitInfo::`waitSemaphoreCount`
- VUID-VkDeviceGroupSubmitInfo-commandBufferCount-00083
  `commandBufferCount` **must** equal VkSubmitInfo::`commandBufferCount`
- VUID-VkDeviceGroupSubmitInfo-signalSemaphoreCount-00084

signalSemaphoreCount **must** equal VkSubmitInfo::signalSemaphoreCount

- VUID-VkDeviceGroupSubmitInfo-pWaitSemaphoreDeviceIndices-00085
  All elements of pWaitSemaphoreDeviceIndices and pSignalSemaphoreDeviceIndices **must** be valid device indices
- VUID-VkDeviceGroupSubmitInfo-pCommandBufferDeviceMasks-00086
  All elements of pCommandBufferDeviceMasks **must** be valid device masks

### Valid Usage (Implicit)

- VUID-VkDeviceGroupSubmitInfo-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_DEVICE_GROUP_SUBMIT_INFO
- VUID-VkDeviceGroupSubmitInfo-pWaitSemaphoreDeviceIndices-parameter
  If waitSemaphoreCount is not 0, pWaitSemaphoreDeviceIndices **must** be a valid pointer to an array of waitSemaphoreCount uint32_t values
- VUID-VkDeviceGroupSubmitInfo-pCommandBufferDeviceMasks-parameter
  If commandBufferCount is not 0, pCommandBufferDeviceMasks **must** be a valid pointer to an array of commandBufferCount uint32_t values
- VUID-VkDeviceGroupSubmitInfo-pSignalSemaphoreDeviceIndices-parameter
  If signalSemaphoreCount is not 0, pSignalSemaphoreDeviceIndices **must** be a valid pointer to an array of signalSemaphoreCount uint32_t values

If the pNext chain of VkSubmitInfo includes a VkPerformanceQuerySubmitInfoKHR structure, then the structure indicates which counter pass is active for the batch in that submit.

The VkPerformanceQuerySubmitInfoKHR structure is defined as:

```
// Provided by VK_KHR_performance_query
typedef struct VkPerformanceQuerySubmitInfoKHR {
    VkStructureType    sType;
    const void*        pNext;
    uint32_t           counterPassIndex;
} VkPerformanceQuerySubmitInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- counterPassIndex specifies which counter pass index is active.

If the VkSubmitInfo::pNext chain does not include this structure, the batch defaults to use counter pass index 0.

### Valid Usage

- VUID-VkPerformanceQuerySubmitInfoKHR-counterPassIndex-03221

`counterPassIndex` **must** be less than the number of counter passes required by any queries within the batch. The required number of counter passes for a performance query is obtained by calling vkGetPhysicalDeviceQueueFamilyPerformanceQueryPassesKHR

**Valid Usage (Implicit)**

- VUID-VkPerformanceQuerySubmitInfoKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PERFORMANCE_QUERY_SUBMIT_INFO_KHR`

## 6.6. Queue Forward Progress

When using binary semaphores, the application **must** ensure that command buffer submissions will be able to complete without any subsequent operations by the application on any queue. After any call to `vkQueueSubmit` (or other queue operation), for every queued wait on a semaphore created with a VkSemaphoreType of `VK_SEMAPHORE_TYPE_BINARY` there **must** be a prior signal of that semaphore that will not be consumed by a different wait on the semaphore.

When using timeline semaphores, wait-before-signal behavior is well-defined and applications **can** submit work via `vkQueueSubmit` defining a timeline semaphore wait operation before submitting a corresponding semaphore signal operation. For each timeline semaphore wait operation defined by a call to `vkQueueSubmit`, the application **must** ensure that a corresponding semaphore signal operation is executed before forward progress can be made.

If a command buffer submission waits for any events to be signaled, the application **must** ensure that command buffer submissions will be able to complete without any subsequent operations by the application. Events signaled by the host **must** be signaled before the command buffer waits on those events.

**NOTE** The ability for commands to wait on the host to set an events was originally added to allow low-latency updates to resources between host and device. However, to ensure quality of service, implementations would necessarily detect extended stalls in execution and timeout after a short period. As this period is not defined in the Vulkan specification, it is impossible to correctly validate any application with any wait period. Since the original users of this functionality were highly limited and platform-specific, this functionality is now considered defunct and should not be used.

## 6.7. Secondary Command Buffer Execution

Secondary command buffers **must** not be directly submitted to a queue. To record a secondary command buffer to execute as part of a primary command buffer, call:

```
// Provided by VK_VERSION_1_0
void vkCmdExecuteCommands(
    VkCommandBuffer                             commandBuffer,
    uint32_t                                    commandBufferCount,
    const VkCommandBuffer*                      pCommandBuffers);
```

- `commandBuffer` is a handle to a primary command buffer that the secondary command buffers are executed in.
- `commandBufferCount` is the length of the `pCommandBuffers` array.
- `pCommandBuffers` is a pointer to an array of `commandBufferCount` secondary command buffer handles, which are recorded to execute in the primary command buffer in the order they are listed in the array.

If any element of `pCommandBuffers` was not recorded with the `VK_COMMAND_BUFFER_USAGE_SIMULTANEOUS_USE_BIT` flag, and it was recorded into any other primary command buffer which is currently in the executable or recording state, that primary command buffer becomes invalid.

If the `nestedCommandBuffer` feature is enabled it is valid usage for `vkCmdExecuteCommands` to also be recorded to a secondary command buffer.

### Valid Usage

- VUID-vkCmdExecuteCommands-pCommandBuffers-00088
  Each element of `pCommandBuffers` **must** have been allocated with a `level` of `VK_COMMAND_BUFFER_LEVEL_SECONDARY`
- VUID-vkCmdExecuteCommands-pCommandBuffers-00089
  Each element of `pCommandBuffers` **must** be in the pending or executable state
- VUID-vkCmdExecuteCommands-pCommandBuffers-00091
  If any element of `pCommandBuffers` was not recorded with the `VK_COMMAND_BUFFER_USAGE_SIMULTANEOUS_USE_BIT` flag, it **must** not be in the pending state
- VUID-vkCmdExecuteCommands-pCommandBuffers-00092
  If any element of `pCommandBuffers` was not recorded with the `VK_COMMAND_BUFFER_USAGE_SIMULTANEOUS_USE_BIT` flag, it **must** not have already been recorded to `commandBuffer`
- VUID-vkCmdExecuteCommands-pCommandBuffers-00093
  If any element of `pCommandBuffers` was not recorded with the `VK_COMMAND_BUFFER_USAGE_SIMULTANEOUS_USE_BIT` flag, it **must** not appear more than once in `pCommandBuffers`
- VUID-vkCmdExecuteCommands-pCommandBuffers-00094
  Each element of `pCommandBuffers` **must** have been allocated from a `VkCommandPool` that was created for the same queue family as the `VkCommandPool` from which `commandBuffer` was allocated
- VUID-vkCmdExecuteCommands-pCommandBuffers-00096

If vkCmdExecuteCommands is being called within a render pass instance, each element of pCommandBuffers **must** have been recorded with the VK_COMMAND_BUFFER_USAGE_RENDER_PASS_CONTINUE_BIT

- VUID-vkCmdExecuteCommands-pCommandBuffers-00099
  If vkCmdExecuteCommands is being called within a render pass instance, and any element of pCommandBuffers was recorded with VkCommandBufferInheritanceInfo::framebuffer not equal to VK_NULL_HANDLE, that VkFramebuffer **must** match the VkFramebuffer used in the current render pass instance
- VUID-vkCmdExecuteCommands-contents-09680
  If vkCmdExecuteCommands is being called within a render pass instance begun with vkCmdBeginRenderPass, and vkCmdNextSubpass has not been called in the current render pass instance, the contents parameter of vkCmdBeginRenderPass **must** have been set to VK_SUBPASS_CONTENTS_SECONDARY_COMMAND_BUFFERS , or VK_SUBPASS_CONTENTS_INLINE_AND_SECONDARY_COMMAND_BUFFERS_EXT
- VUID-vkCmdExecuteCommands-None-09681
  If vkCmdExecuteCommands is being called within a render pass instance begun with vkCmdBeginRenderPass, and vkCmdNextSubpass has been called in the current render pass instance, the contents parameter of the last call to vkCmdNextSubpass **must** have been set to VK_SUBPASS_CONTENTS_SECONDARY_COMMAND_BUFFERS , or VK_SUBPASS_CONTENTS_INLINE_AND_SECONDARY_COMMAND_BUFFERS_KHR
- VUID-vkCmdExecuteCommands-pCommandBuffers-06019
  If vkCmdExecuteCommands is being called within a render pass instance begun with vkCmdBeginRenderPass, each element of pCommandBuffers **must** have been recorded with VkCommandBufferInheritanceInfo::subpass set to the index of the subpass which the given command buffer will be executed in
- VUID-vkCmdExecuteCommands-pBeginInfo-06020
  If vkCmdExecuteCommands is being called within a render pass instance begun with vkCmdBeginRenderPass, the render passes specified in the pBeginInfo->pInheritanceInfo->renderPass members of the vkBeginCommandBuffer commands used to begin recording each element of pCommandBuffers **must** be compatible with the current render pass
- VUID-vkCmdExecuteCommands-pNext-02865
  If vkCmdExecuteCommands is being called within a render pass instance that included VkRenderPassTransformBeginInfoQCOM in the pNext chain of VkRenderPassBeginInfo, then each element of pCommandBuffers **must** have been recorded with VkCommandBufferInheritanceRenderPassTransformInfoQCOM in the pNext chain of VkCommandBufferBeginInfo
- VUID-vkCmdExecuteCommands-pNext-02866
  If vkCmdExecuteCommands is being called within a render pass instance that included VkRenderPassTransformBeginInfoQCOM in the pNext chain of VkRenderPassBeginInfo, then each element of pCommandBuffers **must** have been recorded with VkCommandBufferInheritanceRenderPassTransformInfoQCOM::transform identical to VkRenderPassTransformBeginInfoQCOM::transform
- VUID-vkCmdExecuteCommands-pNext-02867
  If vkCmdExecuteCommands is being called within a render pass instance that included VkRenderPassTransformBeginInfoQCOM in the pNext chain of VkRenderPassBeginInfo,

then each element of pCommandBuffers **must** have been recorded with VkCommandBufferInheritanceRenderPassTransformInfoQCOM::renderArea identical to VkRenderPassBeginInfo::renderArea

- VUID-vkCmdExecuteCommands-pCommandBuffers-00100
  If vkCmdExecuteCommands is not being called within a render pass instance, each element of pCommandBuffers **must** not have been recorded with the VK_COMMAND_BUFFER_USAGE_RENDER_PASS_CONTINUE_BIT
- VUID-vkCmdExecuteCommands-commandBuffer-00101
  If the inheritedQueries feature is not enabled, commandBuffer **must** not have any queries active
- VUID-vkCmdExecuteCommands-commandBuffer-00102
  If commandBuffer has a VK_QUERY_TYPE_OCCLUSION query active, then each element of pCommandBuffers **must** have been recorded with VkCommandBufferInheritanceInfo::occlusionQueryEnable set to VK_TRUE
- VUID-vkCmdExecuteCommands-commandBuffer-00103
  If commandBuffer has a VK_QUERY_TYPE_OCCLUSION query active, then each element of pCommandBuffers **must** have been recorded with VkCommandBufferInheritanceInfo::queryFlags having all bits set that are set for the query
- VUID-vkCmdExecuteCommands-commandBuffer-00104
  If commandBuffer has a VK_QUERY_TYPE_PIPELINE_STATISTICS query active, then each element of pCommandBuffers **must** have been recorded with VkCommandBufferInheritanceInfo::pipelineStatistics having all bits set that are set in the VkQueryPool the query uses
- VUID-vkCmdExecuteCommands-pCommandBuffers-00105
  Each element of pCommandBuffers **must** not begin any query types that are active in commandBuffer
- VUID-vkCmdExecuteCommands-commandBuffer-07594
  commandBuffer **must** not have any queries other than VK_QUERY_TYPE_OCCLUSION and VK_QUERY_TYPE_PIPELINE_STATISTICS active
- VUID-vkCmdExecuteCommands-commandBuffer-01820
  If commandBuffer is a protected command buffer and protectedNoFault is not supported, each element of pCommandBuffers **must** be a protected command buffer
- VUID-vkCmdExecuteCommands-commandBuffer-01821
  If commandBuffer is an unprotected command buffer and protectedNoFault is not supported, each element of pCommandBuffers **must** be an unprotected command buffer
- VUID-vkCmdExecuteCommands-None-02286
  This command **must** not be recorded when transform feedback is active
- VUID-vkCmdExecuteCommands-commandBuffer-06533
  If vkCmdExecuteCommands is being called within a render pass instance and any recorded command in commandBuffer in the current subpass will write to an image subresource as an attachment, commands recorded in elements of pCommandBuffers **must** not read from the memory backing that image subresource in any other way
- VUID-vkCmdExecuteCommands-commandBuffer-06534
  If vkCmdExecuteCommands is being called within a render pass instance and any recorded

command in commandBuffer in the current subpass will read from an image subresource used as an attachment in any way other than as an attachment, commands recorded in elements of pCommandBuffers **must** not write to that image subresource as an attachment

- VUID-vkCmdExecuteCommands-pCommandBuffers-06535
  If vkCmdExecuteCommands is being called within a render pass instance and any recorded command in a given element of pCommandBuffers will write to an image subresource as an attachment, commands recorded in elements of pCommandBuffers at a higher index **must** not read from the memory backing that image subresource in any other way
- VUID-vkCmdExecuteCommands-pCommandBuffers-06536
  If vkCmdExecuteCommands is being called within a render pass instance and any recorded command in a given element of pCommandBuffers will read from an image subresource used as an attachment in any way other than as an attachment, commands recorded in elements of pCommandBuffers at a higher index **must** not write to that image subresource as an attachment
- VUID-vkCmdExecuteCommands-pCommandBuffers-06021
  If pCommandBuffers contains any suspended render pass instances, there **must** be no action or synchronization commands between that render pass instance and any render pass instance that resumes it
- VUID-vkCmdExecuteCommands-pCommandBuffers-06022
  If pCommandBuffers contains any suspended render pass instances, there **must** be no render pass instances between that render pass instance and any render pass instance that resumes it
- VUID-vkCmdExecuteCommands-variableSampleLocations-06023
  If the variableSampleLocations limit is not supported, and any element of pCommandBuffers contains any suspended render pass instances, where a graphics pipeline has been bound, any pipelines bound in the render pass instance that resumes it, or any subsequent render pass instances that resume from that one and so on, **must** use the same sample locations
- VUID-vkCmdExecuteCommands-flags-06024
  If vkCmdExecuteCommands is being called within a render pass instance begun with vkCmdBeginRenderingKHR, its VkRenderingInfo::flags parameter **must** have included VK_RENDERING_CONTENTS_SECONDARY_COMMAND_BUFFERS_BIT
- VUID-vkCmdExecuteCommands-pBeginInfo-06025
  If vkCmdExecuteCommands is being called within a render pass instance begun with vkCmdBeginRenderingKHR, the render passes specified in the pBeginInfo->pInheritanceInfo->renderPass members of the vkBeginCommandBuffer commands used to begin recording each element of pCommandBuffers **must** be VK_NULL_HANDLE
- VUID-vkCmdExecuteCommands-flags-06026
  If vkCmdExecuteCommands is being called within a render pass instance begun with vkCmdBeginRenderingKHR, the flags member of the VkCommandBufferInheritanceRenderingInfo structure included in the pNext chain of VkCommandBufferBeginInfo::pInheritanceInfo used to begin recording each element of pCommandBuffers **must** be equal to the VkRenderingInfo::flags parameter to vkCmdBeginRenderingKHR, excluding VK_RENDERING_CONTENTS_SECONDARY_COMMAND_BUFFERS_BIT

- VUID-vkCmdExecuteCommands-colorAttachmentCount-06027
  If `vkCmdExecuteCommands` is being called within a render pass instance begun with vkCmdBeginRenderingKHR, the `colorAttachmentCount` member of the VkCommandBufferInheritanceRenderingInfo structure included in the `pNext` chain of VkCommandBufferBeginInfo::`pInheritanceInfo` used to begin recording each element of `pCommandBuffers` **must** be equal to the VkRenderingInfo::`colorAttachmentCount` parameter to vkCmdBeginRenderingKHR
- VUID-vkCmdExecuteCommands-imageView-06028
  If `vkCmdExecuteCommands` is being called within a render pass instance begun with vkCmdBeginRenderingKHR, if the `imageView` member of an element of the VkRenderingInfo::`pColorAttachments` parameter to vkCmdBeginRenderingKHR is not VK_NULL_HANDLE, the corresponding element of the `pColorAttachmentFormats` member of the VkCommandBufferInheritanceRenderingInfo structure included in the `pNext` chain of VkCommandBufferBeginInfo::`pInheritanceInfo` used to begin recording each element of `pCommandBuffers` **must** be equal to the format used to create that image view
- VUID-vkCmdExecuteCommands-imageView-07606
  If `vkCmdExecuteCommands` is being called within a render pass instance begun with vkCmdBeginRenderingKHR, if the `imageView` member of an element of the VkRenderingInfo::`pColorAttachments` parameter to vkCmdBeginRenderingKHR is VK_NULL_HANDLE, the corresponding element of the `pColorAttachmentFormats` member of the VkCommandBufferInheritanceRenderingInfo structure included in the `pNext` chain of VkCommandBufferBeginInfo::`pInheritanceInfo` used to begin recording each element of `pCommandBuffers` **must** be `VK_FORMAT_UNDEFINED`
- VUID-vkCmdExecuteCommands-pDepthAttachment-06029
  If `vkCmdExecuteCommands` is being called within a render pass instance begun with vkCmdBeginRenderingKHR, if the VkRenderingInfo::`pDepthAttachment->imageView` parameter to vkCmdBeginRenderingKHR is not VK_NULL_HANDLE, the value of the `depthAttachmentFormat` member of the VkCommandBufferInheritanceRenderingInfo structure included in the `pNext` chain of VkCommandBufferBeginInfo::`pInheritanceInfo` used to begin recording each element of `pCommandBuffers` **must** be equal to the format used to create that image view
- VUID-vkCmdExecuteCommands-pStencilAttachment-06030
  If `vkCmdExecuteCommands` is being called within a render pass instance begun with vkCmdBeginRenderingKHR, if the VkRenderingInfo::`pStencilAttachment->imageView` parameter to vkCmdBeginRenderingKHR is not VK_NULL_HANDLE, the value of the `stencilAttachmentFormat` member of the VkCommandBufferInheritanceRenderingInfo structure included in the `pNext` chain of VkCommandBufferBeginInfo::`pInheritanceInfo` used to begin recording each element of `pCommandBuffers` **must** be equal to the format used to create that image view
- VUID-vkCmdExecuteCommands-pDepthAttachment-06774
  If `vkCmdExecuteCommands` is being called within a render pass instance begun with vkCmdBeginRenderingKHR and the VkRenderingInfo::`pDepthAttachment->imageView` parameter to vkCmdBeginRenderingKHR was VK_NULL_HANDLE, the value of the `depthAttachmentFormat` member of the VkCommandBufferInheritanceRenderingInfo structure included in the `pNext` chain of VkCommandBufferBeginInfo::`pInheritanceInfo` used to begin recording each element of `pCommandBuffers` **must** be `VK_FORMAT_UNDEFINED`

- VUID-vkCmdExecuteCommands-pStencilAttachment-06775
  If vkCmdExecuteCommands is being called within a render pass instance begun with vkCmdBeginRenderingKHR and the VkRenderingInfo::pStencilAttachment->imageView parameter to vkCmdBeginRenderingKHR was VK_NULL_HANDLE, the value of the stencilAttachmentFormat member of the VkCommandBufferInheritanceRenderingInfo structure included in the pNext chain of VkCommandBufferBeginInfo::pInheritanceInfo used to begin recording each element of pCommandBuffers **must** be VK_FORMAT_UNDEFINED
- VUID-vkCmdExecuteCommands-viewMask-06031
  If vkCmdExecuteCommands is being called within a render pass instance begun with vkCmdBeginRenderingKHR, the viewMask member of the VkCommandBufferInheritanceRenderingInfo structure included in the pNext chain of VkCommandBufferBeginInfo::pInheritanceInfo used to begin recording each element of pCommandBuffers **must** be equal to the VkRenderingInfo::viewMask parameter to vkCmdBeginRenderingKHR
- VUID-vkCmdExecuteCommands-pNext-06032
  If vkCmdExecuteCommands is being called within a render pass instance begun with vkCmdBeginRenderingKHR and the pNext chain of VkCommandBufferInheritanceInfo includes a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, if the imageView member of an element of the VkRenderingInfo::pColorAttachments parameter to vkCmdBeginRenderingKHR is not VK_NULL_HANDLE, the corresponding element of the pColorAttachmentSamples member of the VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure included in the pNext chain of VkCommandBufferBeginInfo::pInheritanceInfo used to begin recording each element of pCommandBuffers **must** be equal to the sample count used to create that image view
- VUID-vkCmdExecuteCommands-pNext-06033
  If vkCmdExecuteCommands is being called within a render pass instance begun with vkCmdBeginRenderingKHR and the pNext chain of VkCommandBufferInheritanceInfo includes a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, if the VkRenderingInfo::pDepthAttachment->imageView parameter to vkCmdBeginRenderingKHR is not VK_NULL_HANDLE, the value of the depthStencilAttachmentSamples member of the VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure included in the pNext chain of VkCommandBufferBeginInfo::pInheritanceInfo used to begin recording each element of pCommandBuffers **must** be equal to the sample count used to create that image view
- VUID-vkCmdExecuteCommands-pNext-06034
  If vkCmdExecuteCommands is being called within a render pass instance begun with vkCmdBeginRenderingKHR and the pNext chain of VkCommandBufferInheritanceInfo includes a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, if the VkRenderingInfo::pStencilAttachment->imageView parameter to vkCmdBeginRenderingKHR is not VK_NULL_HANDLE, the value of the depthStencilAttachmentSamples member of the VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure included in the pNext chain of VkCommandBufferBeginInfo::pInheritanceInfo used to begin recording each element of pCommandBuffers **must** be equal to the sample count used to create that image view
- VUID-vkCmdExecuteCommands-pNext-06035

If vkCmdExecuteCommands is being called within a render pass instance begun with vkCmdBeginRenderingKHR and the pNext chain of VkCommandBufferInheritanceInfo does not include a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, if the imageView member of an element of the VkRenderingInfo::pColorAttachments parameter to vkCmdBeginRenderingKHR is not VK_NULL_HANDLE, the value of VkCommandBufferInheritanceRenderingInfo::rasterizationSamples **must** be equal to the sample count used to create that image view

- VUID-vkCmdExecuteCommands-pNext-06036
  If vkCmdExecuteCommands is being called within a render pass instance begun with vkCmdBeginRenderingKHR and the pNext chain of VkCommandBufferInheritanceInfo does not include a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, if the VkRenderingInfo::pDepthAttachment->imageView parameter to vkCmdBeginRenderingKHR is not VK_NULL_HANDLE, the value of VkCommandBufferInheritanceRenderingInfo::rasterizationSamples **must** be equal to the sample count used to create that image view
- VUID-vkCmdExecuteCommands-pNext-06037
  If vkCmdExecuteCommands is being called within a render pass instance begun with vkCmdBeginRenderingKHR and the pNext chain of VkCommandBufferInheritanceInfo does not include a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, if the VkRenderingInfo::pStencilAttachment->imageView parameter to vkCmdBeginRenderingKHR is not VK_NULL_HANDLE, the value of VkCommandBufferInheritanceRenderingInfo::rasterizationSamples **must** be equal to the sample count used to create that image view
- VUID-vkCmdExecuteCommands-pNext-09299
  If vkCmdExecuteCommands is being called within a render pass instance begun with vkCmdBeginRenderingKHR, with any color attachment using a resolve mode of VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID, the pNext chain of VkCommandBufferInheritanceInfo used to create each element of pCommandBuffers **must** include a VkExternalFormatANDROID structure with an externalFormat matching that used to create the resolve attachment in the render pass
- VUID-vkCmdExecuteCommands-pNext-09300
  If vkCmdExecuteCommands is being called within a render pass instance begun with vkCmdBeginRenderingKHR with any color attachment using a resolve mode of VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID, and the pNext chain of VkCommandBufferInheritanceInfo does not include a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, the value of VkCommandBufferInheritanceRenderingInfo::rasterizationSamples **must** be VK_SAMPLE_COUNT_1_BIT
- VUID-vkCmdExecuteCommands-commandBuffer-09375
  commandBuffer **must** not be a secondary command buffer unless the nestedCommandBuffer feature is enabled
- VUID-vkCmdExecuteCommands-nestedCommandBuffer-09376
  If the nestedCommandBuffer feature is enabled, the command buffer nesting level of each element of pCommandBuffers **must** be less than maxCommandBufferNestingLevel
- VUID-vkCmdExecuteCommands-nestedCommandBufferRendering-09377

If the `nestedCommandBufferRendering` feature is not enabled, and `commandBuffer` is a secondary command buffer, `commandBuffer` **must** not have been recorded with `VK_COMMAND_BUFFER_USAGE_RENDER_PASS_CONTINUE_BIT`

- VUID-vkCmdExecuteCommands-nestedCommandBufferSimultaneousUse-09378
  If the `nestedCommandBufferSimultaneousUse` feature is not enabled, and `commandBuffer` is a secondary command buffer, each element of `pCommandBuffers` **must** not have been recorded with `VK_COMMAND_BUFFER_USAGE_SIMULTANEOUS_USE_BIT`
- VUID-vkCmdExecuteCommands-pCommandBuffers-09504
  If `vkCmdExecuteCommands` is being called within a render pass instance begun with vkCmdBeginRenderingKHR, the color attachment mapping state specified by VkRenderingAttachmentLocationInfoKHR in the inheritance info of each element of `pCommandBuffers` and in the current state of `commandBuffer` **must** match
- VUID-vkCmdExecuteCommands-pCommandBuffers-09505
  If `vkCmdExecuteCommands` is being called within a render pass instance begun with vkCmdBeginRenderingKHR, the input attachment mapping state specified by VkRenderingInputAttachmentIndexInfoKHR in the inheritance info of each element of `pCommandBuffers` and in the current state of `commandBuffer` **must** match

## Valid Usage (Implicit)

- VUID-vkCmdExecuteCommands-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdExecuteCommands-pCommandBuffers-parameter
  `pCommandBuffers` **must** be a valid pointer to an array of `commandBufferCount` valid VkCommandBuffer handles
- VUID-vkCmdExecuteCommands-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdExecuteCommands-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support transfer, graphics, or compute operations
- VUID-vkCmdExecuteCommands-videocoding
  This command **must** only be called outside of a video coding scope
- VUID-vkCmdExecuteCommands-commandBufferCount-arraylength
  `commandBufferCount` **must** be greater than `0`
- VUID-vkCmdExecuteCommands-commonparent
  Both of `commandBuffer`, and the elements of `pCommandBuffers` **must** have been created, allocated, or retrieved from the same VkDevice

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally

synchronized

**Command Properties**

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Transfer<br>Graphics<br>Compute | Indirection |

## 6.8. Nested Command Buffers

In addition to secondary command buffer execution from primary command buffers, an implementation **may** support nested command buffers, which enable secondary command buffers to be executed from other secondary command buffers. If the `nestedCommandBuffer` feature is enabled, the implementation supports nested command buffers.

Nested command buffer execution works the same as primary-to-secondary execution, except that it is subject to some additional implementation-defined limits.

Each secondary command buffer has a command buffer nesting level, which is determined at vkEndCommandBuffer time and evaluated at vkCmdExecuteCommands time. A secondary command buffer that executes no other secondary command buffers has a command buffer nesting level of zero. Otherwise, the command buffer nesting level of a secondary command buffer is equal to the maximum nesting level of all secondary command buffers executed by that command buffer plus one. Some implementations **may** have a limit on the maximum nesting level of secondary command buffers that **can** be recorded. This limit is advertised in `maxCommandBufferNestingLevel`.

If the `nestedCommandBufferRendering` feature is enabled, the implementation supports calling vkCmdExecuteCommands inside secondary command buffers recorded with `VK_COMMAND_BUFFER_USAGE_RENDER_PASS_CONTINUE_BIT`. If the `nestedCommandBufferSimultaneousUse` feature is enabled, the implementation supports calling vkCmdExecuteCommands with secondary command buffers recorded with `VK_COMMAND_BUFFER_USAGE_SIMULTANEOUS_USE_BIT`.

Whenever vkCmdExecuteCommands is recorded inside a secondary command buffer recorded with `VK_COMMAND_BUFFER_USAGE_RENDER_PASS_CONTINUE_BIT`, each member of `pCommandBuffers` **must** have been recorded with a VkCommandBufferBeginInfo with VkCommandBufferInheritanceInfo compatible with the VkCommandBufferInheritanceInfo of the command buffer into which the vkCmdExecuteCommands call is being recorded. The VkCommandBufferInheritanceRenderingInfo structures are compatible when the `VkCommandBufferInheritanceRenderingInfo::renderpass` are compatible, or if they are VK_NULL_HANDLE then the VkCommandBufferInheritanceRenderingInfo members match, and all other members of `VkCommandBufferInheritanceRenderingInfo` match. This requirement applies recursively, down to the most nested command buffer and up to the command buffer where the render pass was originally begun.

## 6.9. Command Buffer Device Mask

Each command buffer has a piece of state storing the current device mask of the command buffer. This mask controls which physical devices within the logical device all subsequent commands will execute on, including state-setting commands, action commands, and synchronization commands.

Scissor, exclusive scissor, and viewport state (excluding the count of each) **can** be set to different values on each physical device (only when set as dynamic state), and each physical device will render using its local copy of the state. Other state is shared between physical devices, such that all physical devices use the most recently set values for the state. However, when recording an action command that uses a piece of state, the most recent command that set that state **must** have included all physical devices that execute the action command in its current device mask.

The command buffer's device mask is orthogonal to the `pCommandBufferDeviceMasks` member of VkDeviceGroupSubmitInfo. Commands only execute on a physical device if the device index is set in both device masks.

If the `pNext` chain of VkCommandBufferBeginInfo includes a `VkDeviceGroupCommandBufferBeginInfo` structure, then that structure includes an initial device mask for the command buffer.

The `VkDeviceGroupCommandBufferBeginInfo` structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkDeviceGroupCommandBufferBeginInfo {
    VkStructureType    sType;
    const void*        pNext;
    uint32_t           deviceMask;
} VkDeviceGroupCommandBufferBeginInfo;
```

or the equivalent

```
// Provided by VK_KHR_device_group
typedef VkDeviceGroupCommandBufferBeginInfo VkDeviceGroupCommandBufferBeginInfoKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `deviceMask` is the initial value of the command buffer's device mask.

The initial device mask also acts as an upper bound on the set of devices that **can** ever be in the device mask in the command buffer.

If this structure is not present, the initial value of a command buffer's device mask is set to include all physical devices in the logical device when the command buffer begins recording.

### Valid Usage

- VUID-VkDeviceGroupCommandBufferBeginInfo-deviceMask-00106
  `deviceMask` **must** be a valid device mask value
- VUID-VkDeviceGroupCommandBufferBeginInfo-deviceMask-00107
  `deviceMask` **must** not be zero

### Valid Usage (Implicit)

- VUID-VkDeviceGroupCommandBufferBeginInfo-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_DEVICE_GROUP_COMMAND_BUFFER_BEGIN_INFO`

To update the current device mask of a command buffer, call:

```
// Provided by VK_VERSION_1_1
void vkCmdSetDeviceMask(
    VkCommandBuffer                             commandBuffer,
    uint32_t                                    deviceMask);
```

or the equivalent command

```
// Provided by VK_KHR_device_group
void vkCmdSetDeviceMaskKHR(
    VkCommandBuffer                             commandBuffer,
    uint32_t                                    deviceMask);
```

- `commandBuffer` is command buffer whose current device mask is modified.
- `deviceMask` is the new value of the current device mask.

`deviceMask` is used to filter out subsequent commands from executing on all physical devices whose bit indices are not set in the mask, except commands beginning a render pass instance, commands transitioning to the next subpass in the render pass instance, and commands ending a render pass instance, which always execute on the set of physical devices whose bit indices are included in the `deviceMask` member of the VkDeviceGroupRenderPassBeginInfo structure passed to the command beginning the corresponding render pass instance.

### Valid Usage

- VUID-vkCmdSetDeviceMask-deviceMask-00108
  `deviceMask` **must** be a valid device mask value
- VUID-vkCmdSetDeviceMask-deviceMask-00109
  `deviceMask` **must** not be zero

- VUID-vkCmdSetDeviceMask-deviceMask-00110
  `deviceMask` **must** not include any set bits that were not in the VkDeviceGroupCommandBufferBeginInfo::`deviceMask` value when the command buffer began recording
- VUID-vkCmdSetDeviceMask-deviceMask-00111
  If `vkCmdSetDeviceMask` is called inside a render pass instance, `deviceMask` **must** not include any set bits that were not in the VkDeviceGroupRenderPassBeginInfo::`deviceMask` value when the render pass instance began recording

## Valid Usage (Implicit)

- VUID-vkCmdSetDeviceMask-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdSetDeviceMask-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdSetDeviceMask-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics, compute, or transfer operations

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Both | Graphics<br>Compute<br>Transfer | State |

# Chapter 7. Synchronization and Cache Control

Synchronization of access to resources is primarily the responsibility of the application in Vulkan. The order of execution of commands with respect to the host and other commands on the device has few implicit guarantees, and needs to be explicitly specified. Memory caches and other optimizations are also explicitly managed, requiring that the flow of data through the system is largely under application control.

Whilst some implicit guarantees exist between commands, five explicit synchronization mechanisms are exposed by Vulkan:

**Fences**

Fences **can** be used to communicate to the host that execution of some task on the device has completed, controlling resource access between host and device.

**Semaphores**

Semaphores **can** be used to control resource access across multiple queues.

**Events**

Events provide a fine-grained synchronization primitive which **can** be signaled either within a command buffer or by the host, and **can** be waited upon within a command buffer or queried on the host. Events **can** be used to control resource access within a single queue.

**Pipeline Barriers**

Pipeline barriers also provide synchronization control within a command buffer, but at a single point, rather than with separate signal and wait operations. Pipeline barriers **can** be used to control resource access within a single queue.

**Render Pass Objects**

Render pass objects provide a synchronization framework for rendering tasks, built upon the concepts in this chapter. Many cases that would otherwise need an application to use other synchronization primitives **can** be expressed more efficiently as part of a render pass. Render pass objects **can** be used to control resource access within a single queue.

## 7.1. Execution and Memory Dependencies

An *operation* is an arbitrary amount of work to be executed on the host, a device, or an external entity such as a presentation engine. Synchronization commands introduce explicit *execution dependencies*, and *memory dependencies* between two sets of operations defined by the command's two *synchronization scopes*.

The synchronization scopes define which other operations a synchronization command is able to create execution dependencies with. Any type of operation that is not in a synchronization command's synchronization scopes will not be included in the resulting dependency. For example, for many synchronization commands, the synchronization scopes **can** be limited to just operations executing in specific pipeline stages, which allows other pipeline stages to be excluded from a

dependency. Other scoping options are possible, depending on the particular command.

An *execution dependency* is a guarantee that for two sets of operations, the first set **must** *happen-before* the second set. If an operation happens-before another operation, then the first operation **must** complete before the second operation is initiated. More precisely:

- Let $\mathbf{Ops}_1$ and $\mathbf{Ops}_2$ be separate sets of operations.
- Let **Sync** be a synchronization command.
- Let $\mathbf{Scope}_{1st}$ and $\mathbf{Scope}_{2nd}$ be the synchronization scopes of **Sync**.
- Let $\mathbf{ScopedOps}_1$ be the intersection of sets $\mathbf{Ops}_1$ and $\mathbf{Scope}_{1st}$.
- Let $\mathbf{ScopedOps}_2$ be the intersection of sets $\mathbf{Ops}_2$ and $\mathbf{Scope}_{2nd}$.
- Submitting $\mathbf{Ops}_1$, **Sync** and $\mathbf{Ops}_2$ for execution, in that order, will result in execution dependency **ExeDep** between $\mathbf{ScopedOps}_1$ and $\mathbf{ScopedOps}_2$.
- Execution dependency **ExeDep** guarantees that $\mathbf{ScopedOps}_1$ happen-before $\mathbf{ScopedOps}_2$.

An *execution dependency chain* is a sequence of execution dependencies that form a happens-before relation between the first dependency's $\mathbf{ScopedOps}_1$ and the final dependency's $\mathbf{ScopedOps}_2$. For each consecutive pair of execution dependencies, a chain exists if the intersection of $\mathbf{Scope}_{2nd}$ in the first dependency and $\mathbf{Scope}_{1st}$ in the second dependency is not an empty set. The formation of a single execution dependency from an execution dependency chain can be described by substituting the following in the description of execution dependencies:

- Let **Sync** be a set of synchronization commands that generate an execution dependency chain.
- Let $\mathbf{Scope}_{1st}$ be the first synchronization scope of the first command in **Sync**.
- Let $\mathbf{Scope}_{2nd}$ be the second synchronization scope of the last command in **Sync**.

Execution dependencies alone are not sufficient to guarantee that values resulting from writes in one set of operations **can** be read from another set of operations.

Three additional types of operations are used to control memory access. *Availability operations* cause the values generated by specified memory write accesses to become *available* to a memory domain for future access. Any available value remains available until a subsequent write to the same memory location occurs (whether it is made available or not) or the memory is freed. *Memory domain operations* cause writes that are available to a source memory domain to become available to a destination memory domain (an example of this is making writes available to the host domain available to the device domain). *Visibility operations* cause values available to a memory domain to become *visible* to specified memory accesses.

Availability, visibility, memory domains, and memory domain operations are formally defined in the Availability and Visibility section of the Memory Model chapter. Which API operations perform each of these operations is defined in Availability, Visibility, and Domain Operations.

A *memory dependency* is an execution dependency which includes availability and visibility operations such that:

- The first set of operations happens-before the availability operation.

- The availability operation happens-before the visibility operation.
- The visibility operation happens-before the second set of operations.

Once written values are made visible to a particular type of memory access, they **can** be read or written by that type of memory access. Most synchronization commands in Vulkan define a memory dependency.

The specific memory accesses that are made available and visible are defined by the *access scopes* of a memory dependency. Any type of access that is in a memory dependency's first access scope and occurs in **ScopedOps**$_1$ is made available. Any type of access that is in a memory dependency's second access scope and occurs in **ScopedOps**$_2$ has any available writes made visible to it. Any type of operation that is not in a synchronization command's access scopes will not be included in the resulting dependency.

A memory dependency enforces availability and visibility of memory accesses and execution order between two sets of operations. Adding to the description of execution dependency chains:

- Let **MemOps**$_1$ be the set of memory accesses performed by **ScopedOps**$_1$.
- Let **MemOps**$_2$ be the set of memory accesses performed by **ScopedOps**$_2$.
- Let **AccessScope**$_{1st}$ be the first access scope of the first command in the **Sync** chain.
- Let **AccessScope**$_{2nd}$ be the second access scope of the last command in the **Sync** chain.
- Let **ScopedMemOps**$_1$ be the intersection of sets **MemOps**$_1$ and **AccessScope**$_{1st}$.
- Let **ScopedMemOps**$_2$ be the intersection of sets **MemOps**$_2$ and **AccessScope**$_{2nd}$.
- Submitting **Ops**$_1$, **Sync**, and **Ops**$_2$ for execution, in that order, will result in a memory dependency **MemDep** between **ScopedOps**$_1$ and **ScopedOps**$_2$.
- Memory dependency **MemDep** guarantees that:
  - Memory writes in **ScopedMemOps**$_1$ are made available.
  - Available memory writes, including those from **ScopedMemOps**$_1$, are made visible to **ScopedMemOps**$_2$.

**NOTE**

Execution and memory dependencies are used to solve data hazards, i.e. to ensure that read and write operations occur in a well-defined order. Write-after-read hazards can be solved with just an execution dependency, but read-after-write and write-after-write hazards need appropriate memory dependencies to be included between them. If an application does not include dependencies to solve these hazards, the results and execution orders of memory accesses are undefined.

### 7.1.1. Image Layout Transitions

Image subresources **can** be transitioned from one layout to another as part of a memory dependency (e.g. by using an image memory barrier). When a layout transition is specified in a memory dependency, it happens-after the availability operations in the memory dependency, and happens-before the visibility operations. Image layout transitions **may** perform read and write accesses on all memory bound to the image subresource range, so applications **must** ensure that all memory writes have been made available before a layout transition is executed. Available memory

is automatically made visible to a layout transition, and writes performed by a layout transition are automatically made available.

Layout transitions always apply to a particular image subresource range, and specify both an old layout and new layout. The old layout **must** either be VK_IMAGE_LAYOUT_UNDEFINED, or match the current layout of the image subresource range. If the old layout matches the current layout of the image subresource range, the transition preserves the contents of that range. If the old layout is VK_IMAGE_LAYOUT_UNDEFINED, the contents of that range **may** be discarded.

**NOTE**

Image layout transitions with VK_IMAGE_LAYOUT_UNDEFINED allow the implementation to discard the image subresource range, which can provide performance or power benefits. Tile-based architectures may be able to avoid flushing tile data to memory, and immediate style renderers may be able to achieve fast metadata clears to reinitialize frame buffer compression state, or similar.

If the contents of an attachment are not needed after a render pass completes, then applications **should** use VK_ATTACHMENT_STORE_OP_DONT_CARE.

As image layout transitions **may** perform read and write accesses on the memory bound to the image, if the image subresource affected by the layout transition is bound to peer memory for any device in the current device mask then the memory heap the bound memory comes from **must** support the VK_PEER_MEMORY_FEATURE_GENERIC_SRC_BIT and VK_PEER_MEMORY_FEATURE_GENERIC_DST_BIT capabilities as returned by vkGetDeviceGroupPeerMemoryFeatures.

**NOTE**

Applications **must** ensure that layout transitions happen-after all operations accessing the image with the old layout, and happen-before any operations that will access the image with the new layout. Layout transitions are potentially read/write operations, so not defining appropriate memory dependencies to guarantee this will result in a data race.

Image layout transitions interact with memory aliasing.

Layout transitions that are performed via image memory barriers execute in their entirety in submission order, relative to other image layout transitions submitted to the same queue, including those performed by render passes. In effect there is an implicit execution dependency from each such layout transition to all layout transitions previously submitted to the same queue.

The image layout of each image subresource of a depth/stencil image created with VK_IMAGE_CREATE_SAMPLE_LOCATIONS_COMPATIBLE_DEPTH_BIT_EXT is dependent on the last sample locations used to render to the image subresource as a depth/stencil attachment, thus when the image member of an image memory barrier is an image created with this flag the application **can** chain a VkSampleLocationsInfoEXT structure to the pNext chain of VkImageMemoryBarrier2 or VkImageMemoryBarrier to specify the sample locations to use during any image layout transition.

If the VkSampleLocationsInfoEXT structure does not match the sample location state last used to render to the image subresource range specified by subresourceRange, or if no VkSampleLocationsInfoEXT structure is present, then the contents of the given image subresource range becomes undefined as if oldLayout would equal VK_IMAGE_LAYOUT_UNDEFINED.

### 7.1.2. Pipeline Stages

The work performed by an action command consists of multiple operations, which are performed as a sequence of logically independent steps known as *pipeline stages*. The exact pipeline stages executed depend on the particular command that is used, and current command buffer state when the command was recorded.

| | |
|---|---|
| **NOTE** | Operations performed by synchronization commands (e.g. availability and visibility operations) are not executed by a defined pipeline stage. However other commands can still synchronize with them by using the synchronization scopes to create a dependency chain. |

Execution of operations across pipeline stages **must** adhere to implicit ordering guarantees, particularly including pipeline stage order. Otherwise, execution across pipeline stages **may** overlap or execute out of order with regards to other stages, unless otherwise enforced by an execution dependency.

Several of the synchronization commands include pipeline stage parameters, restricting the synchronization scopes for that command to just those stages. This allows fine grained control over the exact execution dependencies and accesses performed by action commands. Implementations **should** use these pipeline stages to avoid unnecessary stalls or cache flushing.

Bits which **can** be set in a VkPipelineStageFlags2 mask, specifying stages of execution, are:

```
// Flag bits for VkPipelineStageFlagBits2
typedef VkFlags64 VkPipelineStageFlagBits2;
static const VkPipelineStageFlagBits2 VK_PIPELINE_STAGE_2_NONE = 0ULL;
static const VkPipelineStageFlagBits2 VK_PIPELINE_STAGE_2_NONE_KHR = 0ULL;
static const VkPipelineStageFlagBits2 VK_PIPELINE_STAGE_2_TOP_OF_PIPE_BIT =
0x00000001ULL;
static const VkPipelineStageFlagBits2 VK_PIPELINE_STAGE_2_TOP_OF_PIPE_BIT_KHR =
0x00000001ULL;
static const VkPipelineStageFlagBits2 VK_PIPELINE_STAGE_2_DRAW_INDIRECT_BIT =
0x00000002ULL;
static const VkPipelineStageFlagBits2 VK_PIPELINE_STAGE_2_DRAW_INDIRECT_BIT_KHR =
0x00000002ULL;
static const VkPipelineStageFlagBits2 VK_PIPELINE_STAGE_2_VERTEX_INPUT_BIT =
0x00000004ULL;
static const VkPipelineStageFlagBits2 VK_PIPELINE_STAGE_2_VERTEX_INPUT_BIT_KHR =
0x00000004ULL;
static const VkPipelineStageFlagBits2 VK_PIPELINE_STAGE_2_VERTEX_SHADER_BIT =
0x00000008ULL;
static const VkPipelineStageFlagBits2 VK_PIPELINE_STAGE_2_VERTEX_SHADER_BIT_KHR =
0x00000008ULL;
static const VkPipelineStageFlagBits2
VK_PIPELINE_STAGE_2_TESSELLATION_CONTROL_SHADER_BIT = 0x00000010ULL;
static const VkPipelineStageFlagBits2
VK_PIPELINE_STAGE_2_TESSELLATION_CONTROL_SHADER_BIT_KHR = 0x00000010ULL;
static const VkPipelineStageFlagBits2
VK_PIPELINE_STAGE_2_TESSELLATION_EVALUATION_SHADER_BIT = 0x00000020ULL;
```

```
static const VkPipelineStageFlagBits2
VK_PIPELINE_STAGE_2_TESSELLATION_EVALUATION_SHADER_BIT_KHR = 0x00000020ULL;
static const VkPipelineStageFlagBits2 VK_PIPELINE_STAGE_2_GEOMETRY_SHADER_BIT =
0x00000040ULL;
static const VkPipelineStageFlagBits2 VK_PIPELINE_STAGE_2_GEOMETRY_SHADER_BIT_KHR =
0x00000040ULL;
static const VkPipelineStageFlagBits2 VK_PIPELINE_STAGE_2_FRAGMENT_SHADER_BIT =
0x00000080ULL;
static const VkPipelineStageFlagBits2 VK_PIPELINE_STAGE_2_FRAGMENT_SHADER_BIT_KHR =
0x00000080ULL;
static const VkPipelineStageFlagBits2 VK_PIPELINE_STAGE_2_EARLY_FRAGMENT_TESTS_BIT =
0x00000100ULL;
static const VkPipelineStageFlagBits2 VK_PIPELINE_STAGE_2_EARLY_FRAGMENT_TESTS_BIT_KHR
= 0x00000100ULL;
static const VkPipelineStageFlagBits2 VK_PIPELINE_STAGE_2_LATE_FRAGMENT_TESTS_BIT =
0x00000200ULL;
static const VkPipelineStageFlagBits2 VK_PIPELINE_STAGE_2_LATE_FRAGMENT_TESTS_BIT_KHR
= 0x00000200ULL;
static const VkPipelineStageFlagBits2 VK_PIPELINE_STAGE_2_COLOR_ATTACHMENT_OUTPUT_BIT
= 0x00000400ULL;
static const VkPipelineStageFlagBits2
VK_PIPELINE_STAGE_2_COLOR_ATTACHMENT_OUTPUT_BIT_KHR = 0x00000400ULL;
static const VkPipelineStageFlagBits2 VK_PIPELINE_STAGE_2_COMPUTE_SHADER_BIT =
0x00000800ULL;
static const VkPipelineStageFlagBits2 VK_PIPELINE_STAGE_2_COMPUTE_SHADER_BIT_KHR =
0x00000800ULL;
static const VkPipelineStageFlagBits2 VK_PIPELINE_STAGE_2_ALL_TRANSFER_BIT =
0x00001000ULL;
static const VkPipelineStageFlagBits2 VK_PIPELINE_STAGE_2_ALL_TRANSFER_BIT_KHR =
0x00001000ULL;
static const VkPipelineStageFlagBits2 VK_PIPELINE_STAGE_2_TRANSFER_BIT =
0x00001000ULL;
static const VkPipelineStageFlagBits2 VK_PIPELINE_STAGE_2_TRANSFER_BIT_KHR =
0x00001000ULL;
static const VkPipelineStageFlagBits2 VK_PIPELINE_STAGE_2_BOTTOM_OF_PIPE_BIT =
0x00002000ULL;
static const VkPipelineStageFlagBits2 VK_PIPELINE_STAGE_2_BOTTOM_OF_PIPE_BIT_KHR =
0x00002000ULL;
static const VkPipelineStageFlagBits2 VK_PIPELINE_STAGE_2_HOST_BIT = 0x00004000ULL;
static const VkPipelineStageFlagBits2 VK_PIPELINE_STAGE_2_HOST_BIT_KHR =
0x00004000ULL;
static const VkPipelineStageFlagBits2 VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT =
0x00008000ULL;
static const VkPipelineStageFlagBits2 VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT_KHR =
0x00008000ULL;
static const VkPipelineStageFlagBits2 VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT =
0x00010000ULL;
static const VkPipelineStageFlagBits2 VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT_KHR =
0x00010000ULL;
static const VkPipelineStageFlagBits2 VK_PIPELINE_STAGE_2_COPY_BIT = 0x100000000ULL;
static const VkPipelineStageFlagBits2 VK_PIPELINE_STAGE_2_COPY_BIT_KHR =
```

```
0x100000000ULL;
static const VkPipelineStageFlagBits2 VK_PIPELINE_STAGE_2_RESOLVE_BIT =
0x200000000ULL;
static const VkPipelineStageFlagBits2 VK_PIPELINE_STAGE_2_RESOLVE_BIT_KHR =
0x200000000ULL;
static const VkPipelineStageFlagBits2 VK_PIPELINE_STAGE_2_BLIT_BIT = 0x400000000ULL;
static const VkPipelineStageFlagBits2 VK_PIPELINE_STAGE_2_BLIT_BIT_KHR =
0x400000000ULL;
static const VkPipelineStageFlagBits2 VK_PIPELINE_STAGE_2_CLEAR_BIT = 0x800000000ULL;
static const VkPipelineStageFlagBits2 VK_PIPELINE_STAGE_2_CLEAR_BIT_KHR =
0x800000000ULL;
static const VkPipelineStageFlagBits2 VK_PIPELINE_STAGE_2_INDEX_INPUT_BIT =
0x1000000000ULL;
static const VkPipelineStageFlagBits2 VK_PIPELINE_STAGE_2_INDEX_INPUT_BIT_KHR =
0x1000000000ULL;
static const VkPipelineStageFlagBits2 VK_PIPELINE_STAGE_2_VERTEX_ATTRIBUTE_INPUT_BIT =
0x2000000000ULL;
static const VkPipelineStageFlagBits2
VK_PIPELINE_STAGE_2_VERTEX_ATTRIBUTE_INPUT_BIT_KHR = 0x2000000000ULL;
static const VkPipelineStageFlagBits2
VK_PIPELINE_STAGE_2_PRE_RASTERIZATION_SHADERS_BIT = 0x4000000000ULL;
static const VkPipelineStageFlagBits2
VK_PIPELINE_STAGE_2_PRE_RASTERIZATION_SHADERS_BIT_KHR = 0x4000000000ULL;
// Provided by VK_KHR_video_decode_queue
static const VkPipelineStageFlagBits2 VK_PIPELINE_STAGE_2_VIDEO_DECODE_BIT_KHR =
0x04000000ULL;
// Provided by VK_KHR_video_encode_queue
static const VkPipelineStageFlagBits2 VK_PIPELINE_STAGE_2_VIDEO_ENCODE_BIT_KHR =
0x08000000ULL;
// Provided by VK_KHR_synchronization2 with VK_EXT_transform_feedback
static const VkPipelineStageFlagBits2 VK_PIPELINE_STAGE_2_TRANSFORM_FEEDBACK_BIT_EXT =
0x01000000ULL;
// Provided by VK_KHR_synchronization2 with VK_EXT_conditional_rendering
static const VkPipelineStageFlagBits2
VK_PIPELINE_STAGE_2_CONDITIONAL_RENDERING_BIT_EXT = 0x00040000ULL;
// Provided by VK_KHR_synchronization2 with VK_NV_device_generated_commands
static const VkPipelineStageFlagBits2 VK_PIPELINE_STAGE_2_COMMAND_PREPROCESS_BIT_NV =
0x00020000ULL;
// Provided by VK_KHR_fragment_shading_rate with VK_KHR_synchronization2
static const VkPipelineStageFlagBits2
VK_PIPELINE_STAGE_2_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR = 0x00400000ULL;
// Provided by VK_KHR_synchronization2 with VK_NV_shading_rate_image
static const VkPipelineStageFlagBits2 VK_PIPELINE_STAGE_2_SHADING_RATE_IMAGE_BIT_NV =
0x00400000ULL;
// Provided by VK_KHR_acceleration_structure with VK_KHR_synchronization2
static const VkPipelineStageFlagBits2
VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_BUILD_BIT_KHR = 0x02000000ULL;
// Provided by VK_KHR_ray_tracing_pipeline with VK_KHR_synchronization2
static const VkPipelineStageFlagBits2 VK_PIPELINE_STAGE_2_RAY_TRACING_SHADER_BIT_KHR =
0x00200000ULL;
// Provided by VK_KHR_synchronization2 with VK_NV_ray_tracing
```

```
static const VkPipelineStageFlagBits2 VK_PIPELINE_STAGE_2_RAY_TRACING_SHADER_BIT_NV =
0x00200000ULL;
// Provided by VK_KHR_synchronization2 with VK_NV_ray_tracing
static const VkPipelineStageFlagBits2
VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_BUILD_BIT_NV = 0x02000000ULL;
// Provided by VK_KHR_synchronization2 with VK_EXT_fragment_density_map
static const VkPipelineStageFlagBits2
VK_PIPELINE_STAGE_2_FRAGMENT_DENSITY_PROCESS_BIT_EXT = 0x00800000ULL;
// Provided by VK_KHR_synchronization2 with VK_NV_mesh_shader
static const VkPipelineStageFlagBits2 VK_PIPELINE_STAGE_2_TASK_SHADER_BIT_NV =
0x00080000ULL;
// Provided by VK_KHR_synchronization2 with VK_NV_mesh_shader
static const VkPipelineStageFlagBits2 VK_PIPELINE_STAGE_2_MESH_SHADER_BIT_NV =
0x00100000ULL;
// Provided by VK_KHR_synchronization2 with VK_EXT_mesh_shader
static const VkPipelineStageFlagBits2 VK_PIPELINE_STAGE_2_TASK_SHADER_BIT_EXT =
0x00080000ULL;
// Provided by VK_KHR_synchronization2 with VK_EXT_mesh_shader
static const VkPipelineStageFlagBits2 VK_PIPELINE_STAGE_2_MESH_SHADER_BIT_EXT =
0x00100000ULL;
// Provided by VK_HUAWEI_subpass_shading
static const VkPipelineStageFlagBits2 VK_PIPELINE_STAGE_2_SUBPASS_SHADER_BIT_HUAWEI =
0x8000000000ULL;
// VK_PIPELINE_STAGE_2_SUBPASS_SHADING_BIT_HUAWEI is a deprecated alias
// Provided by VK_HUAWEI_subpass_shading
static const VkPipelineStageFlagBits2 VK_PIPELINE_STAGE_2_SUBPASS_SHADING_BIT_HUAWEI =
0x8000000000ULL;
// Provided by VK_HUAWEI_invocation_mask
static const VkPipelineStageFlagBits2 VK_PIPELINE_STAGE_2_INVOCATION_MASK_BIT_HUAWEI =
0x10000000000ULL;
// Provided by VK_KHR_ray_tracing_maintenance1 with VK_KHR_synchronization2 or
VK_VERSION_1_3
static const VkPipelineStageFlagBits2
VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_COPY_BIT_KHR = 0x10000000ULL;
// Provided by VK_EXT_opacity_micromap
static const VkPipelineStageFlagBits2 VK_PIPELINE_STAGE_2_MICROMAP_BUILD_BIT_EXT =
0x40000000ULL;
// Provided by VK_HUAWEI_cluster_culling_shader
static const VkPipelineStageFlagBits2
VK_PIPELINE_STAGE_2_CLUSTER_CULLING_SHADER_BIT_HUAWEI = 0x20000000000ULL;
// Provided by VK_NV_optical_flow
static const VkPipelineStageFlagBits2 VK_PIPELINE_STAGE_2_OPTICAL_FLOW_BIT_NV =
0x20000000ULL;
```

or the equivalent

```
// Provided by VK_KHR_synchronization2
typedef VkPipelineStageFlagBits2 VkPipelineStageFlagBits2KHR;
```

- VK_PIPELINE_STAGE_2_NONE specifies no stages of execution.
- VK_PIPELINE_STAGE_2_DRAW_INDIRECT_BIT specifies the stage of the pipeline where indirect command parameters are consumed. This stage also includes reading commands written by vkCmdPreprocessGeneratedCommandsNV.
- VK_PIPELINE_STAGE_2_TASK_SHADER_BIT_EXT specifies the task shader stage.
- VK_PIPELINE_STAGE_2_MESH_SHADER_BIT_EXT specifies the mesh shader stage.
- VK_PIPELINE_STAGE_2_INDEX_INPUT_BIT specifies the stage of the pipeline where index buffers are consumed.
- VK_PIPELINE_STAGE_2_VERTEX_ATTRIBUTE_INPUT_BIT specifies the stage of the pipeline where vertex buffers are consumed.
- VK_PIPELINE_STAGE_2_VERTEX_INPUT_BIT is equivalent to the logical OR of:
  - VK_PIPELINE_STAGE_2_INDEX_INPUT_BIT
  - VK_PIPELINE_STAGE_2_VERTEX_ATTRIBUTE_INPUT_BIT
- VK_PIPELINE_STAGE_2_VERTEX_SHADER_BIT specifies the vertex shader stage.
- VK_PIPELINE_STAGE_2_TESSELLATION_CONTROL_SHADER_BIT specifies the tessellation control shader stage.
- VK_PIPELINE_STAGE_2_TESSELLATION_EVALUATION_SHADER_BIT specifies the tessellation evaluation shader stage.
- VK_PIPELINE_STAGE_2_GEOMETRY_SHADER_BIT specifies the geometry shader stage.
- VK_PIPELINE_STAGE_2_PRE_RASTERIZATION_SHADERS_BIT is equivalent to specifying all supported pre-rasterization shader stages:
  - VK_PIPELINE_STAGE_2_VERTEX_SHADER_BIT
  - VK_PIPELINE_STAGE_2_TESSELLATION_CONTROL_SHADER_BIT
  - VK_PIPELINE_STAGE_2_TESSELLATION_EVALUATION_SHADER_BIT
  - VK_PIPELINE_STAGE_2_GEOMETRY_SHADER_BIT
  - VK_PIPELINE_STAGE_2_TASK_SHADER_BIT_EXT
  - VK_PIPELINE_STAGE_2_MESH_SHADER_BIT_EXT
  - VK_PIPELINE_STAGE_2_CLUSTER_CULLING_SHADER_BIT_HUAWEI
- VK_PIPELINE_STAGE_2_FRAGMENT_SHADER_BIT specifies the fragment shader stage.
- VK_PIPELINE_STAGE_2_EARLY_FRAGMENT_TESTS_BIT specifies the stage of the pipeline where early fragment tests (depth and stencil tests before fragment shading) are performed. This stage also includes render pass load operations for framebuffer attachments with a depth/stencil format.
- VK_PIPELINE_STAGE_2_LATE_FRAGMENT_TESTS_BIT specifies the stage of the pipeline where late fragment tests (depth and stencil tests after fragment shading) are performed. This stage also includes render pass store operations for framebuffer attachments with a depth/stencil format.
- VK_PIPELINE_STAGE_2_COLOR_ATTACHMENT_OUTPUT_BIT specifies the stage of the pipeline where final color values are output from the pipeline. This stage includes blending, logic operations, render pass load and store operations for color attachments, render pass multisample resolve

operations, and vkCmdClearAttachments.

- VK_PIPELINE_STAGE_2_COMPUTE_SHADER_BIT specifies the compute shader stage.
- VK_PIPELINE_STAGE_2_HOST_BIT specifies a pseudo-stage indicating execution on the host of reads/writes of device memory. This stage is not invoked by any commands recorded in a command buffer.
- VK_PIPELINE_STAGE_2_COPY_BIT specifies the execution of all copy commands, including vkCmdCopyQueryPoolResults.
- VK_PIPELINE_STAGE_2_BLIT_BIT specifies the execution of vkCmdBlitImage.
- VK_PIPELINE_STAGE_2_RESOLVE_BIT specifies the execution of vkCmdResolveImage.
- VK_PIPELINE_STAGE_2_CLEAR_BIT specifies the execution of clear commands, with the exception of vkCmdClearAttachments.
- VK_PIPELINE_STAGE_2_ALL_TRANSFER_BIT is equivalent to specifying all of:
  - VK_PIPELINE_STAGE_2_COPY_BIT
  - VK_PIPELINE_STAGE_2_BLIT_BIT
  - VK_PIPELINE_STAGE_2_RESOLVE_BIT
  - VK_PIPELINE_STAGE_2_CLEAR_BIT
  - VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_COPY_BIT_KHR
- VK_PIPELINE_STAGE_2_RAY_TRACING_SHADER_BIT_KHR specifies the execution of the ray tracing shader stages.
- VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_BUILD_BIT_KHR specifies the execution of acceleration structure commands or acceleration structure copy commands.
- VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_COPY_BIT_KHR specifies the execution of acceleration structure copy commands.
- VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT specifies the execution of all graphics pipeline stages, and is equivalent to the logical OR of:
  - VK_PIPELINE_STAGE_2_DRAW_INDIRECT_BIT
  - VK_PIPELINE_STAGE_2_TASK_SHADER_BIT_EXT
  - VK_PIPELINE_STAGE_2_MESH_SHADER_BIT_EXT
  - VK_PIPELINE_STAGE_2_VERTEX_INPUT_BIT
  - VK_PIPELINE_STAGE_2_VERTEX_SHADER_BIT
  - VK_PIPELINE_STAGE_2_TESSELLATION_CONTROL_SHADER_BIT
  - VK_PIPELINE_STAGE_2_TESSELLATION_EVALUATION_SHADER_BIT
  - VK_PIPELINE_STAGE_2_GEOMETRY_SHADER_BIT
  - VK_PIPELINE_STAGE_2_FRAGMENT_SHADER_BIT
  - VK_PIPELINE_STAGE_2_EARLY_FRAGMENT_TESTS_BIT
  - VK_PIPELINE_STAGE_2_LATE_FRAGMENT_TESTS_BIT
  - VK_PIPELINE_STAGE_2_COLOR_ATTACHMENT_OUTPUT_BIT

  - VK_PIPELINE_STAGE_2_CONDITIONAL_RENDERING_BIT_EXT
  - VK_PIPELINE_STAGE_2_TRANSFORM_FEEDBACK_BIT_EXT
  - VK_PIPELINE_STAGE_2_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR
  - VK_PIPELINE_STAGE_2_FRAGMENT_DENSITY_PROCESS_BIT_EXT
  - VK_PIPELINE_STAGE_2_SUBPASS_SHADER_BIT_HUAWEI
  - VK_PIPELINE_STAGE_2_INVOCATION_MASK_BIT_HUAWEI
  - VK_PIPELINE_STAGE_2_CLUSTER_CULLING_SHADER_BIT_HUAWEI

- VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT specifies all operations performed by all commands supported on the queue it is used with.
- VK_PIPELINE_STAGE_2_CONDITIONAL_RENDERING_BIT_EXT specifies the stage of the pipeline where the predicate of conditional rendering is consumed.
- VK_PIPELINE_STAGE_2_TRANSFORM_FEEDBACK_BIT_EXT specifies the stage of the pipeline where vertex attribute output values are written to the transform feedback buffers.
- VK_PIPELINE_STAGE_2_COMMAND_PREPROCESS_BIT_NV specifies the stage of the pipeline where device-side generation of commands via vkCmdPreprocessGeneratedCommandsNV is handled.
- VK_PIPELINE_STAGE_2_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR specifies the stage of the pipeline where the fragment shading rate attachment or shading rate image is read to determine the fragment shading rate for portions of a rasterized primitive.
- VK_PIPELINE_STAGE_2_FRAGMENT_DENSITY_PROCESS_BIT_EXT specifies the stage of the pipeline where the fragment density map is read to generate the fragment areas.
- VK_PIPELINE_STAGE_2_INVOCATION_MASK_BIT_HUAWEI specifies the stage of the pipeline where the invocation mask image is read by the implementation to optimize the ray dispatch.
- VK_PIPELINE_STAGE_2_VIDEO_DECODE_BIT_KHR specifies the execution of video decode operations.
- VK_PIPELINE_STAGE_2_VIDEO_ENCODE_BIT_KHR specifies the execution of video encode operations.
- VK_PIPELINE_STAGE_2_OPTICAL_FLOW_BIT_NV specifies the stage of the pipeline where optical flow operation are performed.
- VK_PIPELINE_STAGE_2_SUBPASS_SHADER_BIT_HUAWEI specifies the subpass shading shader stage.
- VK_PIPELINE_STAGE_2_MICROMAP_BUILD_BIT_EXT specifies the execution of micromap commands.
- VK_PIPELINE_STAGE_2_CLUSTER_CULLING_SHADER_BIT_HUAWEI specifies the cluster culling shader stage.
- VK_PIPELINE_STAGE_2_TOP_OF_PIPE_BIT is equivalent to VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT with VkAccessFlags2 set to 0 when specified in the second synchronization scope, but equivalent to VK_PIPELINE_STAGE_2_NONE in the first scope.
- VK_PIPELINE_STAGE_2_BOTTOM_OF_PIPE_BIT is equivalent to VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT with VkAccessFlags2 set to 0 when specified in the first synchronization scope, but equivalent to VK_PIPELINE_STAGE_2_NONE in the second scope.

**NOTE** The TOP and BOTTOM pipeline stages are deprecated, and applications should prefer VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT and VK_PIPELINE_STAGE_2_NONE.

**NOTE** The VkPipelineStageFlags2 bitmask goes beyond the 31 individual bit flags allowable within a C99 enum, which is how VkPipelineStageFlagBits is defined. The first 31 values are common to both, and are interchangeable.

VkPipelineStageFlags2 is a bitmask type for setting a mask of zero or more VkPipelineStageFlagBits2 flags:

```
typedef VkFlags64 VkPipelineStageFlags2;
```

or the equivalent

```
// Provided by VK_KHR_synchronization2
typedef VkPipelineStageFlags2 VkPipelineStageFlags2KHR;
```

Bits which **can** be set in a VkPipelineStageFlags mask, specifying stages of execution, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkPipelineStageFlagBits {
    VK_PIPELINE_STAGE_TOP_OF_PIPE_BIT = 0x00000001,
    VK_PIPELINE_STAGE_DRAW_INDIRECT_BIT = 0x00000002,
    VK_PIPELINE_STAGE_VERTEX_INPUT_BIT = 0x00000004,
    VK_PIPELINE_STAGE_VERTEX_SHADER_BIT = 0x00000008,
    VK_PIPELINE_STAGE_TESSELLATION_CONTROL_SHADER_BIT = 0x00000010,
    VK_PIPELINE_STAGE_TESSELLATION_EVALUATION_SHADER_BIT = 0x00000020,
    VK_PIPELINE_STAGE_GEOMETRY_SHADER_BIT = 0x00000040,
    VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT = 0x00000080,
    VK_PIPELINE_STAGE_EARLY_FRAGMENT_TESTS_BIT = 0x00000100,
    VK_PIPELINE_STAGE_LATE_FRAGMENT_TESTS_BIT = 0x00000200,
    VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT = 0x00000400,
    VK_PIPELINE_STAGE_COMPUTE_SHADER_BIT = 0x00000800,
    VK_PIPELINE_STAGE_TRANSFER_BIT = 0x00001000,
    VK_PIPELINE_STAGE_BOTTOM_OF_PIPE_BIT = 0x00002000,
    VK_PIPELINE_STAGE_HOST_BIT = 0x00004000,
    VK_PIPELINE_STAGE_ALL_GRAPHICS_BIT = 0x00008000,
    VK_PIPELINE_STAGE_ALL_COMMANDS_BIT = 0x00010000,
    VK_PIPELINE_STAGE_NONE = 0,
  // Provided by VK_EXT_transform_feedback
    VK_PIPELINE_STAGE_TRANSFORM_FEEDBACK_BIT_EXT = 0x01000000,
  // Provided by VK_EXT_conditional_rendering
    VK_PIPELINE_STAGE_CONDITIONAL_RENDERING_BIT_EXT = 0x00040000,
  // Provided by VK_KHR_acceleration_structure
    VK_PIPELINE_STAGE_ACCELERATION_STRUCTURE_BUILD_BIT_KHR = 0x02000000,
  // Provided by VK_KHR_ray_tracing_pipeline
    VK_PIPELINE_STAGE_RAY_TRACING_SHADER_BIT_KHR = 0x00200000,
  // Provided by VK_EXT_fragment_density_map
    VK_PIPELINE_STAGE_FRAGMENT_DENSITY_PROCESS_BIT_EXT = 0x00800000,
  // Provided by VK_KHR_fragment_shading_rate
```

```
    VK_PIPELINE_STAGE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR = 0x00400000,
  // Provided by VK_NV_device_generated_commands
    VK_PIPELINE_STAGE_COMMAND_PREPROCESS_BIT_NV = 0x00020000,
  // Provided by VK_EXT_mesh_shader
    VK_PIPELINE_STAGE_TASK_SHADER_BIT_EXT = 0x00080000,
  // Provided by VK_EXT_mesh_shader
    VK_PIPELINE_STAGE_MESH_SHADER_BIT_EXT = 0x00100000,
  // Provided by VK_NV_shading_rate_image
    VK_PIPELINE_STAGE_SHADING_RATE_IMAGE_BIT_NV =
VK_PIPELINE_STAGE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR,
  // Provided by VK_NV_ray_tracing
    VK_PIPELINE_STAGE_RAY_TRACING_SHADER_BIT_NV =
VK_PIPELINE_STAGE_RAY_TRACING_SHADER_BIT_KHR,
  // Provided by VK_NV_ray_tracing
    VK_PIPELINE_STAGE_ACCELERATION_STRUCTURE_BUILD_BIT_NV =
VK_PIPELINE_STAGE_ACCELERATION_STRUCTURE_BUILD_BIT_KHR,
  // Provided by VK_NV_mesh_shader
    VK_PIPELINE_STAGE_TASK_SHADER_BIT_NV = VK_PIPELINE_STAGE_TASK_SHADER_BIT_EXT,
  // Provided by VK_NV_mesh_shader
    VK_PIPELINE_STAGE_MESH_SHADER_BIT_NV = VK_PIPELINE_STAGE_MESH_SHADER_BIT_EXT,
  // Provided by VK_KHR_synchronization2
    VK_PIPELINE_STAGE_NONE_KHR = VK_PIPELINE_STAGE_NONE,
} VkPipelineStageFlagBits;
```

These values all have the same meaning as the equivalently named values for VkPipelineStageFlags2.

- `VK_PIPELINE_STAGE_NONE` specifies no stages of execution.
- `VK_PIPELINE_STAGE_DRAW_INDIRECT_BIT` specifies the stage of the pipeline where `VkDrawIndirect*` / `VkDispatchIndirect*` / `VkTraceRaysIndirect*` data structures are consumed. This stage also includes reading commands written by vkCmdExecuteGeneratedCommandsNV.
- `VK_PIPELINE_STAGE_TASK_SHADER_BIT_EXT` specifies the task shader stage.
- `VK_PIPELINE_STAGE_MESH_SHADER_BIT_EXT` specifies the mesh shader stage.
- `VK_PIPELINE_STAGE_VERTEX_INPUT_BIT` specifies the stage of the pipeline where vertex and index buffers are consumed.
- `VK_PIPELINE_STAGE_VERTEX_SHADER_BIT` specifies the vertex shader stage.
- `VK_PIPELINE_STAGE_TESSELLATION_CONTROL_SHADER_BIT` specifies the tessellation control shader stage.
- `VK_PIPELINE_STAGE_TESSELLATION_EVALUATION_SHADER_BIT` specifies the tessellation evaluation shader stage.
- `VK_PIPELINE_STAGE_GEOMETRY_SHADER_BIT` specifies the geometry shader stage.
- `VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT` specifies the fragment shader stage.
- `VK_PIPELINE_STAGE_EARLY_FRAGMENT_TESTS_BIT` specifies the stage of the pipeline where early fragment tests (depth and stencil tests before fragment shading) are performed. This stage also includes render pass load operations for framebuffer attachments with a depth/stencil format.

- VK_PIPELINE_STAGE_LATE_FRAGMENT_TESTS_BIT specifies the stage of the pipeline where late fragment tests (depth and stencil tests after fragment shading) are performed. This stage also includes render pass store operations for framebuffer attachments with a depth/stencil format.
- VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT specifies the stage of the pipeline after blending where the final color values are output from the pipeline. This stage includes blending, logic operations, render pass load and store operations for color attachments, render pass multisample resolve operations, and vkCmdClearAttachments.
- VK_PIPELINE_STAGE_COMPUTE_SHADER_BIT specifies the execution of a compute shader.
- VK_PIPELINE_STAGE_TRANSFER_BIT specifies the following commands:
  - All copy commands, including vkCmdCopyQueryPoolResults
  - vkCmdBlitImage2KHR and vkCmdBlitImage
  - vkCmdResolveImage2KHR and vkCmdResolveImage
  - All clear commands, with the exception of vkCmdClearAttachments
- VK_PIPELINE_STAGE_HOST_BIT specifies a pseudo-stage indicating execution on the host of reads/writes of device memory. This stage is not invoked by any commands recorded in a command buffer.
- VK_PIPELINE_STAGE_ACCELERATION_STRUCTURE_BUILD_BIT_KHR specifies the execution of vkCmdBuildAccelerationStructureNV, vkCmdCopyAccelerationStructureNV, vkCmdWriteAccelerationStructuresPropertiesNV , vkCmdBuildAccelerationStructuresKHR, vkCmdBuildAccelerationStructuresIndirectKHR, vkCmdCopyAccelerationStructureKHR, vkCmdCopyAccelerationStructureToMemoryKHR, vkCmdCopyMemoryToAccelerationStructureKHR, and vkCmdWriteAccelerationStructuresPropertiesKHR.
- VK_PIPELINE_STAGE_RAY_TRACING_SHADER_BIT_KHR specifies the execution of the ray tracing shader stages, via vkCmdTraceRaysNV , vkCmdTraceRaysKHR, or vkCmdTraceRaysIndirectKHR
- VK_PIPELINE_STAGE_ALL_GRAPHICS_BIT specifies the execution of all graphics pipeline stages, and is equivalent to the logical OR of:
  - VK_PIPELINE_STAGE_DRAW_INDIRECT_BIT
  - VK_PIPELINE_STAGE_TASK_SHADER_BIT_EXT
  - VK_PIPELINE_STAGE_MESH_SHADER_BIT_EXT
  - VK_PIPELINE_STAGE_VERTEX_INPUT_BIT
  - VK_PIPELINE_STAGE_VERTEX_SHADER_BIT
  - VK_PIPELINE_STAGE_TESSELLATION_CONTROL_SHADER_BIT
  - VK_PIPELINE_STAGE_TESSELLATION_EVALUATION_SHADER_BIT
  - VK_PIPELINE_STAGE_GEOMETRY_SHADER_BIT
  - VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT
  - VK_PIPELINE_STAGE_EARLY_FRAGMENT_TESTS_BIT
  - VK_PIPELINE_STAGE_LATE_FRAGMENT_TESTS_BIT
  - VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT

  - VK_PIPELINE_STAGE_CONDITIONAL_RENDERING_BIT_EXT
  - VK_PIPELINE_STAGE_TRANSFORM_FEEDBACK_BIT_EXT
  - VK_PIPELINE_STAGE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR
  - VK_PIPELINE_STAGE_FRAGMENT_DENSITY_PROCESS_BIT_EXT

- VK_PIPELINE_STAGE_ALL_COMMANDS_BIT specifies all operations performed by all commands supported on the queue it is used with.
- VK_PIPELINE_STAGE_CONDITIONAL_RENDERING_BIT_EXT specifies the stage of the pipeline where the predicate of conditional rendering is consumed.
- VK_PIPELINE_STAGE_TRANSFORM_FEEDBACK_BIT_EXT specifies the stage of the pipeline where vertex attribute output values are written to the transform feedback buffers.
- VK_PIPELINE_STAGE_COMMAND_PREPROCESS_BIT_NV specifies the stage of the pipeline where device-side preprocessing for generated commands via vkCmdPreprocessGeneratedCommandsNV is handled.
- VK_PIPELINE_STAGE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR specifies the stage of the pipeline where the fragment shading rate attachment or shading rate image is read to determine the fragment shading rate for portions of a rasterized primitive.
- VK_PIPELINE_STAGE_FRAGMENT_DENSITY_PROCESS_BIT_EXT specifies the stage of the pipeline where the fragment density map is read to generate the fragment areas.
- VK_PIPELINE_STAGE_TOP_OF_PIPE_BIT is equivalent to VK_PIPELINE_STAGE_ALL_COMMANDS_BIT with VkAccessFlags set to 0 when specified in the second synchronization scope, but specifies no stage of execution when specified in the first scope.
- VK_PIPELINE_STAGE_BOTTOM_OF_PIPE_BIT is equivalent to VK_PIPELINE_STAGE_ALL_COMMANDS_BIT with VkAccessFlags set to 0 when specified in the first synchronization scope, but specifies no stage of execution when specified in the second scope.

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkPipelineStageFlags;
```

VkPipelineStageFlags is a bitmask type for setting a mask of zero or more VkPipelineStageFlagBits.

If a synchronization command includes a source stage mask, its first synchronization scope only includes execution of the pipeline stages specified in that mask and any logically earlier stages. Its first access scope only includes memory accesses performed by pipeline stages explicitly specified in the source stage mask.

If a synchronization command includes a destination stage mask, its second synchronization scope only includes execution of the pipeline stages specified in that mask and any logically later stages. Its second access scope only includes memory accesses performed by pipeline stages explicitly specified in the destination stage mask.

| | |
|---|---|
| **NOTE** | Note that access scopes do not interact with the logically earlier or later stages for either scope - only the stages the application specifies are considered part of each access scope. |

Certain pipeline stages are only available on queues that support a particular set of operations. The following table lists, for each pipeline stage flag, which queue capability flag **must** be supported by the queue. When multiple flags are enumerated in the second column of the table, it means that the pipeline stage is supported on the queue if it supports any of the listed capability flags. For further details on queue capabilities see Physical Device Enumeration and Queues.

*Table 3. Supported pipeline stage flags*

| Pipeline stage flag | Required queue capability flag |
|---|---|
| VK_PIPELINE_STAGE_2_NONE | None required |
| VK_PIPELINE_STAGE_2_TOP_OF_PIPE_BIT | None required |
| VK_PIPELINE_STAGE_2_DRAW_INDIRECT_BIT | VK_QUEUE_GRAPHICS_BIT or VK_QUEUE_COMPUTE_BIT |
| VK_PIPELINE_STAGE_2_VERTEX_INPUT_BIT | VK_QUEUE_GRAPHICS_BIT |
| VK_PIPELINE_STAGE_2_VERTEX_SHADER_BIT | VK_QUEUE_GRAPHICS_BIT |
| VK_PIPELINE_STAGE_2_TESSELLATION_CONTROL_SHADER_BIT | VK_QUEUE_GRAPHICS_BIT |
| VK_PIPELINE_STAGE_2_TESSELLATION_EVALUATION_SHADER_BIT | VK_QUEUE_GRAPHICS_BIT |
| VK_PIPELINE_STAGE_2_GEOMETRY_SHADER_BIT | VK_QUEUE_GRAPHICS_BIT |
| VK_PIPELINE_STAGE_2_FRAGMENT_SHADER_BIT | VK_QUEUE_GRAPHICS_BIT |
| VK_PIPELINE_STAGE_2_EARLY_FRAGMENT_TESTS_BIT | VK_QUEUE_GRAPHICS_BIT |
| VK_PIPELINE_STAGE_2_LATE_FRAGMENT_TESTS_BIT | VK_QUEUE_GRAPHICS_BIT |
| VK_PIPELINE_STAGE_2_COLOR_ATTACHMENT_OUTPUT_BIT | VK_QUEUE_GRAPHICS_BIT |
| VK_PIPELINE_STAGE_2_COMPUTE_SHADER_BIT | VK_QUEUE_COMPUTE_BIT |
| VK_PIPELINE_STAGE_2_ALL_TRANSFER_BIT | VK_QUEUE_GRAPHICS_BIT or VK_QUEUE_COMPUTE_BIT or VK_QUEUE_TRANSFER_BIT |
| VK_PIPELINE_STAGE_2_BOTTOM_OF_PIPE_BIT | None required |
| VK_PIPELINE_STAGE_2_HOST_BIT | None required |
| VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT | VK_QUEUE_GRAPHICS_BIT |
| VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT | None required |
| VK_PIPELINE_STAGE_2_COPY_BIT | VK_QUEUE_GRAPHICS_BIT or VK_QUEUE_COMPUTE_BIT or VK_QUEUE_TRANSFER_BIT |
| VK_PIPELINE_STAGE_2_RESOLVE_BIT | VK_QUEUE_GRAPHICS_BIT or VK_QUEUE_COMPUTE_BIT or VK_QUEUE_TRANSFER_BIT |
| VK_PIPELINE_STAGE_2_BLIT_BIT | VK_QUEUE_GRAPHICS_BIT or VK_QUEUE_COMPUTE_BIT or VK_QUEUE_TRANSFER_BIT |
| VK_PIPELINE_STAGE_2_CLEAR_BIT | VK_QUEUE_GRAPHICS_BIT or VK_QUEUE_COMPUTE_BIT or VK_QUEUE_TRANSFER_BIT |

| Pipeline stage flag | Required queue capability flag |
|---|---|
| VK_PIPELINE_STAGE_2_INDEX_INPUT_BIT | VK_QUEUE_GRAPHICS_BIT |
| VK_PIPELINE_STAGE_2_VERTEX_ATTRIBUTE_INPUT_BIT | VK_QUEUE_GRAPHICS_BIT |
| VK_PIPELINE_STAGE_2_PRE_RASTERIZATION_SHADERS_BIT | VK_QUEUE_GRAPHICS_BIT |
| VK_PIPELINE_STAGE_2_VIDEO_DECODE_BIT_KHR | VK_QUEUE_VIDEO_DECODE_BIT_KHR |
| VK_PIPELINE_STAGE_2_VIDEO_ENCODE_BIT_KHR | VK_QUEUE_VIDEO_ENCODE_BIT_KHR |
| VK_PIPELINE_STAGE_2_TRANSFORM_FEEDBACK_BIT_EXT | VK_QUEUE_GRAPHICS_BIT |
| VK_PIPELINE_STAGE_2_CONDITIONAL_RENDERING_BIT_EXT | VK_QUEUE_GRAPHICS_BIT or VK_QUEUE_COMPUTE_BIT |
| VK_PIPELINE_STAGE_2_COMMAND_PREPROCESS_BIT_NV | VK_QUEUE_GRAPHICS_BIT or VK_QUEUE_COMPUTE_BIT |
| VK_PIPELINE_STAGE_2_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR | VK_QUEUE_GRAPHICS_BIT |
| VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_BUILD_BIT_KHR | VK_QUEUE_COMPUTE_BIT |
| VK_PIPELINE_STAGE_2_RAY_TRACING_SHADER_BIT_KHR | VK_QUEUE_COMPUTE_BIT |
| VK_PIPELINE_STAGE_2_FRAGMENT_DENSITY_PROCESS_BIT_EXT | VK_QUEUE_GRAPHICS_BIT |
| VK_PIPELINE_STAGE_2_TASK_SHADER_BIT_EXT | VK_QUEUE_GRAPHICS_BIT |
| VK_PIPELINE_STAGE_2_MESH_SHADER_BIT_EXT | VK_QUEUE_GRAPHICS_BIT |
| VK_PIPELINE_STAGE_2_SUBPASS_SHADER_BIT_HUAWEI | VK_QUEUE_GRAPHICS_BIT |
| VK_PIPELINE_STAGE_2_INVOCATION_MASK_BIT_HUAWEI | VK_QUEUE_GRAPHICS_BIT |
| VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_COPY_BIT_KHR | VK_QUEUE_GRAPHICS_BIT or VK_QUEUE_COMPUTE_BIT or VK_QUEUE_TRANSFER_BIT |
| VK_PIPELINE_STAGE_2_MICROMAP_BUILD_BIT_EXT | VK_QUEUE_COMPUTE_BIT |
| VK_PIPELINE_STAGE_2_CLUSTER_CULLING_SHADER_BIT_HUAWEI | VK_QUEUE_GRAPHICS_BIT |
| VK_PIPELINE_STAGE_2_OPTICAL_FLOW_BIT_NV | VK_QUEUE_OPTICAL_FLOW_BIT_NV |

Pipeline stages that execute as a result of a command logically complete execution in a specific order, such that completion of a logically later pipeline stage **must** not happen-before completion of a logically earlier stage. This means that including any stage in the source stage mask for a particular synchronization command also implies that any logically earlier stages are included in **Scope**$_{1st}$ for that command.

Similarly, initiation of a logically earlier pipeline stage **must** not happen-after initiation of a logically later pipeline stage. Including any given stage in the destination stage mask for a particular synchronization command also implies that any logically later stages are included in **Scope**$_{2nd}$ for that command.

**NOTE** Implementations **may** not support synchronization at every pipeline stage for every synchronization operation. If a pipeline stage that an implementation does not support synchronization for appears in a source stage mask, it **may** substitute any

> logically later stage in its place for the first synchronization scope. If a pipeline stage that an implementation does not support synchronization for appears in a destination stage mask, it **may** substitute any logically earlier stage in its place for the second synchronization scope.
>
> For example, if an implementation is unable to signal an event immediately after vertex shader execution is complete, it **may** instead signal the event after color attachment output has completed.
>
> If an implementation makes such a substitution, it **must** not affect the semantics of execution or memory dependencies or image and buffer memory barriers.

Graphics pipelines are executable on queues supporting `VK_QUEUE_GRAPHICS_BIT`. Stages executed by graphics pipelines **can** only be specified in commands recorded for queues supporting `VK_QUEUE_GRAPHICS_BIT`.

The graphics primitive pipeline executes the following stages, with the logical ordering of the stages matching the order specified here:

- `VK_PIPELINE_STAGE_2_DRAW_INDIRECT_BIT`
- `VK_PIPELINE_STAGE_2_INDEX_INPUT_BIT`
- `VK_PIPELINE_STAGE_2_VERTEX_ATTRIBUTE_INPUT_BIT`
- `VK_PIPELINE_STAGE_2_VERTEX_SHADER_BIT`
- `VK_PIPELINE_STAGE_2_TESSELLATION_CONTROL_SHADER_BIT`
- `VK_PIPELINE_STAGE_2_TESSELLATION_EVALUATION_SHADER_BIT`
- `VK_PIPELINE_STAGE_2_GEOMETRY_SHADER_BIT`
- `VK_PIPELINE_STAGE_2_TRANSFORM_FEEDBACK_BIT_EXT`
- `VK_PIPELINE_STAGE_2_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR`
- `VK_PIPELINE_STAGE_2_EARLY_FRAGMENT_TESTS_BIT`
- `VK_PIPELINE_STAGE_2_FRAGMENT_SHADER_BIT`
- `VK_PIPELINE_STAGE_2_LATE_FRAGMENT_TESTS_BIT`
- `VK_PIPELINE_STAGE_2_COLOR_ATTACHMENT_OUTPUT_BIT`

The graphics mesh pipeline executes the following stages, with the logical ordering of the stages matching the order specified here:

- `VK_PIPELINE_STAGE_2_DRAW_INDIRECT_BIT`
- `VK_PIPELINE_STAGE_2_TASK_SHADER_BIT_EXT`
- `VK_PIPELINE_STAGE_2_MESH_SHADER_BIT_EXT`
- `VK_PIPELINE_STAGE_2_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR`
- `VK_PIPELINE_STAGE_2_EARLY_FRAGMENT_TESTS_BIT`
- `VK_PIPELINE_STAGE_2_FRAGMENT_SHADER_BIT`

- VK_PIPELINE_STAGE_2_LATE_FRAGMENT_TESTS_BIT
- VK_PIPELINE_STAGE_2_COLOR_ATTACHMENT_OUTPUT_BIT

For the compute pipeline, the following stages occur in this order:

- VK_PIPELINE_STAGE_2_DRAW_INDIRECT_BIT
- VK_PIPELINE_STAGE_2_COMPUTE_SHADER_BIT

For the subpass shading pipeline, the following stages occur in this order:

- VK_PIPELINE_STAGE_2_SUBPASS_SHADER_BIT_HUAWEI

For graphics pipeline commands executing in a render pass with a fragment density map attachment, the following pipeline stage where the fragment density map read happens has no particular order relative to the other stages, except that it is logically earlier than VK_PIPELINE_STAGE_EARLY_FRAGMENT_TESTS_BIT:

- VK_PIPELINE_STAGE_FRAGMENT_DENSITY_PROCESS_BIT_EXT
- VK_PIPELINE_STAGE_EARLY_FRAGMENT_TESTS_BIT

The conditional rendering stage is formally part of both the graphics, and the compute pipeline. The pipeline stage where the predicate read happens has unspecified order relative to other stages of these pipelines:

- VK_PIPELINE_STAGE_CONDITIONAL_RENDERING_BIT_EXT

For the transfer pipeline, the following stages occur in this order:

- VK_PIPELINE_STAGE_2_TRANSFER_BIT

For host operations, only one pipeline stage occurs, so no order is guaranteed:

- VK_PIPELINE_STAGE_2_HOST_BIT

For the command preprocessing pipeline, the following stages occur in this order:

- VK_PIPELINE_STAGE_2_COMMAND_PREPROCESS_BIT_NV

For acceleration structure build operations, only one pipeline stage occurs, so no order is guaranteed:

- VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_BUILD_BIT_KHR

For acceleration structure copy operations, only one pipeline stage occurs, so no order is guaranteed:

- VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_COPY_BIT_KHR

For opacity micromap build operations, only one pipeline stage occurs, so no order is guaranteed:

- VK_PIPELINE_STAGE_2_MICROMAP_BUILD_BIT_EXT

For the ray tracing pipeline, the following stages occur in this order:

- VK_PIPELINE_STAGE_2_DRAW_INDIRECT_BIT
- VK_PIPELINE_STAGE_2_RAY_TRACING_SHADER_BIT_KHR

For the video decode pipeline, the following stages occur in this order:

- VK_PIPELINE_STAGE_2_VIDEO_DECODE_BIT_KHR

For the video encode pipeline, the following stages occur in this order:

- VK_PIPELINE_STAGE_2_VIDEO_ENCODE_BIT_KHR

### 7.1.3. Access Types

Memory in Vulkan **can** be accessed from within shader invocations and via some fixed-function stages of the pipeline. The *access type* is a function of the descriptor type used, or how a fixed-function stage accesses memory.

Some synchronization commands take sets of access types as parameters to define the access scopes of a memory dependency. If a synchronization command includes a *source access mask*, its first access scope only includes accesses via the access types specified in that mask. Similarly, if a synchronization command includes a *destination access mask*, its second access scope only includes accesses via the access types specified in that mask.

Bits which **can** be set in the srcAccessMask and dstAccessMask members of VkMemoryBarrier2KHR, VkImageMemoryBarrier2KHR, and VkBufferMemoryBarrier2KHR, specifying access behavior, are:

```
// Flag bits for VkAccessFlagBits2
typedef VkFlags64 VkAccessFlagBits2;
static const VkAccessFlagBits2 VK_ACCESS_2_NONE = 0ULL;
static const VkAccessFlagBits2 VK_ACCESS_2_NONE_KHR = 0ULL;
static const VkAccessFlagBits2 VK_ACCESS_2_INDIRECT_COMMAND_READ_BIT = 0x00000001ULL;
static const VkAccessFlagBits2 VK_ACCESS_2_INDIRECT_COMMAND_READ_BIT_KHR =
0x00000001ULL;
static const VkAccessFlagBits2 VK_ACCESS_2_INDEX_READ_BIT = 0x00000002ULL;
static const VkAccessFlagBits2 VK_ACCESS_2_INDEX_READ_BIT_KHR = 0x00000002ULL;
static const VkAccessFlagBits2 VK_ACCESS_2_VERTEX_ATTRIBUTE_READ_BIT = 0x00000004ULL;
static const VkAccessFlagBits2 VK_ACCESS_2_VERTEX_ATTRIBUTE_READ_BIT_KHR =
0x00000004ULL;
static const VkAccessFlagBits2 VK_ACCESS_2_UNIFORM_READ_BIT = 0x00000008ULL;
static const VkAccessFlagBits2 VK_ACCESS_2_UNIFORM_READ_BIT_KHR = 0x00000008ULL;
static const VkAccessFlagBits2 VK_ACCESS_2_INPUT_ATTACHMENT_READ_BIT = 0x00000010ULL;
static const VkAccessFlagBits2 VK_ACCESS_2_INPUT_ATTACHMENT_READ_BIT_KHR =
0x00000010ULL;
static const VkAccessFlagBits2 VK_ACCESS_2_SHADER_READ_BIT = 0x00000020ULL;
static const VkAccessFlagBits2 VK_ACCESS_2_SHADER_READ_BIT_KHR = 0x00000020ULL;
static const VkAccessFlagBits2 VK_ACCESS_2_SHADER_WRITE_BIT = 0x00000040ULL;
static const VkAccessFlagBits2 VK_ACCESS_2_SHADER_WRITE_BIT_KHR = 0x00000040ULL;
static const VkAccessFlagBits2 VK_ACCESS_2_COLOR_ATTACHMENT_READ_BIT = 0x00000080ULL;
```

```
static const VkAccessFlagBits2 VK_ACCESS_2_COLOR_ATTACHMENT_READ_BIT_KHR =
0x00000080ULL;
static const VkAccessFlagBits2 VK_ACCESS_2_COLOR_ATTACHMENT_WRITE_BIT = 0x00000100ULL;
static const VkAccessFlagBits2 VK_ACCESS_2_COLOR_ATTACHMENT_WRITE_BIT_KHR =
0x00000100ULL;
static const VkAccessFlagBits2 VK_ACCESS_2_DEPTH_STENCIL_ATTACHMENT_READ_BIT =
0x00000200ULL;
static const VkAccessFlagBits2 VK_ACCESS_2_DEPTH_STENCIL_ATTACHMENT_READ_BIT_KHR =
0x00000200ULL;
static const VkAccessFlagBits2 VK_ACCESS_2_DEPTH_STENCIL_ATTACHMENT_WRITE_BIT =
0x00000400ULL;
static const VkAccessFlagBits2 VK_ACCESS_2_DEPTH_STENCIL_ATTACHMENT_WRITE_BIT_KHR =
0x00000400ULL;
static const VkAccessFlagBits2 VK_ACCESS_2_TRANSFER_READ_BIT = 0x00000800ULL;
static const VkAccessFlagBits2 VK_ACCESS_2_TRANSFER_READ_BIT_KHR = 0x00000800ULL;
static const VkAccessFlagBits2 VK_ACCESS_2_TRANSFER_WRITE_BIT = 0x00001000ULL;
static const VkAccessFlagBits2 VK_ACCESS_2_TRANSFER_WRITE_BIT_KHR = 0x00001000ULL;
static const VkAccessFlagBits2 VK_ACCESS_2_HOST_READ_BIT = 0x00002000ULL;
static const VkAccessFlagBits2 VK_ACCESS_2_HOST_READ_BIT_KHR = 0x00002000ULL;
static const VkAccessFlagBits2 VK_ACCESS_2_HOST_WRITE_BIT = 0x00004000ULL;
static const VkAccessFlagBits2 VK_ACCESS_2_HOST_WRITE_BIT_KHR = 0x00004000ULL;
static const VkAccessFlagBits2 VK_ACCESS_2_MEMORY_READ_BIT = 0x00008000ULL;
static const VkAccessFlagBits2 VK_ACCESS_2_MEMORY_READ_BIT_KHR = 0x00008000ULL;
static const VkAccessFlagBits2 VK_ACCESS_2_MEMORY_WRITE_BIT = 0x00010000ULL;
static const VkAccessFlagBits2 VK_ACCESS_2_MEMORY_WRITE_BIT_KHR = 0x00010000ULL;
static const VkAccessFlagBits2 VK_ACCESS_2_SHADER_SAMPLED_READ_BIT = 0x100000000ULL;
static const VkAccessFlagBits2 VK_ACCESS_2_SHADER_SAMPLED_READ_BIT_KHR =
0x100000000ULL;
static const VkAccessFlagBits2 VK_ACCESS_2_SHADER_STORAGE_READ_BIT = 0x200000000ULL;
static const VkAccessFlagBits2 VK_ACCESS_2_SHADER_STORAGE_READ_BIT_KHR =
0x200000000ULL;
static const VkAccessFlagBits2 VK_ACCESS_2_SHADER_STORAGE_WRITE_BIT = 0x400000000ULL;
static const VkAccessFlagBits2 VK_ACCESS_2_SHADER_STORAGE_WRITE_BIT_KHR =
0x400000000ULL;
// Provided by VK_KHR_video_decode_queue
static const VkAccessFlagBits2 VK_ACCESS_2_VIDEO_DECODE_READ_BIT_KHR = 0x800000000ULL;
// Provided by VK_KHR_video_decode_queue
static const VkAccessFlagBits2 VK_ACCESS_2_VIDEO_DECODE_WRITE_BIT_KHR =
0x1000000000ULL;
// Provided by VK_KHR_video_encode_queue
static const VkAccessFlagBits2 VK_ACCESS_2_VIDEO_ENCODE_READ_BIT_KHR =
0x2000000000ULL;
// Provided by VK_KHR_video_encode_queue
static const VkAccessFlagBits2 VK_ACCESS_2_VIDEO_ENCODE_WRITE_BIT_KHR =
0x4000000000ULL;
// Provided by VK_KHR_synchronization2 with VK_EXT_transform_feedback
static const VkAccessFlagBits2 VK_ACCESS_2_TRANSFORM_FEEDBACK_WRITE_BIT_EXT =
0x02000000ULL;
// Provided by VK_KHR_synchronization2 with VK_EXT_transform_feedback
static const VkAccessFlagBits2 VK_ACCESS_2_TRANSFORM_FEEDBACK_COUNTER_READ_BIT_EXT =
0x04000000ULL;
```

```
// Provided by VK_KHR_synchronization2 with VK_EXT_transform_feedback
static const VkAccessFlagBits2 VK_ACCESS_2_TRANSFORM_FEEDBACK_COUNTER_WRITE_BIT_EXT =
0x08000000ULL;
// Provided by VK_KHR_synchronization2 with VK_EXT_conditional_rendering
static const VkAccessFlagBits2 VK_ACCESS_2_CONDITIONAL_RENDERING_READ_BIT_EXT =
0x00100000ULL;
// Provided by VK_KHR_synchronization2 with VK_NV_device_generated_commands
static const VkAccessFlagBits2 VK_ACCESS_2_COMMAND_PREPROCESS_READ_BIT_NV =
0x00020000ULL;
// Provided by VK_KHR_synchronization2 with VK_NV_device_generated_commands
static const VkAccessFlagBits2 VK_ACCESS_2_COMMAND_PREPROCESS_WRITE_BIT_NV =
0x00040000ULL;
// Provided by VK_KHR_fragment_shading_rate with VK_KHR_synchronization2
static const VkAccessFlagBits2
VK_ACCESS_2_FRAGMENT_SHADING_RATE_ATTACHMENT_READ_BIT_KHR = 0x00800000ULL;
// Provided by VK_KHR_synchronization2 with VK_NV_shading_rate_image
static const VkAccessFlagBits2 VK_ACCESS_2_SHADING_RATE_IMAGE_READ_BIT_NV =
0x00800000ULL;
// Provided by VK_KHR_acceleration_structure with VK_KHR_synchronization2
static const VkAccessFlagBits2 VK_ACCESS_2_ACCELERATION_STRUCTURE_READ_BIT_KHR =
0x00200000ULL;
// Provided by VK_KHR_acceleration_structure with VK_KHR_synchronization2
static const VkAccessFlagBits2 VK_ACCESS_2_ACCELERATION_STRUCTURE_WRITE_BIT_KHR =
0x00400000ULL;
// Provided by VK_KHR_synchronization2 with VK_NV_ray_tracing
static const VkAccessFlagBits2 VK_ACCESS_2_ACCELERATION_STRUCTURE_READ_BIT_NV =
0x00200000ULL;
// Provided by VK_KHR_synchronization2 with VK_NV_ray_tracing
static const VkAccessFlagBits2 VK_ACCESS_2_ACCELERATION_STRUCTURE_WRITE_BIT_NV =
0x00400000ULL;
// Provided by VK_KHR_synchronization2 with VK_EXT_fragment_density_map
static const VkAccessFlagBits2 VK_ACCESS_2_FRAGMENT_DENSITY_MAP_READ_BIT_EXT =
0x01000000ULL;
// Provided by VK_KHR_synchronization2 with VK_EXT_blend_operation_advanced
static const VkAccessFlagBits2 VK_ACCESS_2_COLOR_ATTACHMENT_READ_NONCOHERENT_BIT_EXT =
0x00080000ULL;
// Provided by VK_EXT_descriptor_buffer
static const VkAccessFlagBits2 VK_ACCESS_2_DESCRIPTOR_BUFFER_READ_BIT_EXT =
0x20000000000ULL;
// Provided by VK_HUAWEI_invocation_mask
static const VkAccessFlagBits2 VK_ACCESS_2_INVOCATION_MASK_READ_BIT_HUAWEI =
0x8000000000ULL;
// Provided by VK_KHR_ray_tracing_maintenance1 with (VK_KHR_synchronization2 or
VK_VERSION_1_3) and VK_KHR_ray_tracing_pipeline
static const VkAccessFlagBits2 VK_ACCESS_2_SHADER_BINDING_TABLE_READ_BIT_KHR =
0x10000000000ULL;
// Provided by VK_EXT_opacity_micromap
static const VkAccessFlagBits2 VK_ACCESS_2_MICROMAP_READ_BIT_EXT = 0x100000000000ULL;
// Provided by VK_EXT_opacity_micromap
static const VkAccessFlagBits2 VK_ACCESS_2_MICROMAP_WRITE_BIT_EXT = 0x200000000000ULL;
// Provided by VK_NV_optical_flow
```

```
static const VkAccessFlagBits2 VK_ACCESS_2_OPTICAL_FLOW_READ_BIT_NV =
0x40000000000ULL;
// Provided by VK_NV_optical_flow
static const VkAccessFlagBits2 VK_ACCESS_2_OPTICAL_FLOW_WRITE_BIT_NV =
0x80000000000ULL;
```

or the equivalent

```
// Provided by VK_KHR_synchronization2
typedef VkAccessFlagBits2 VkAccessFlagBits2KHR;
```

- VK_ACCESS_2_NONE specifies no accesses.
- VK_ACCESS_2_MEMORY_READ_BIT specifies all read accesses. It is always valid in any access mask, and is treated as equivalent to setting all READ access flags that are valid where it is used.
- VK_ACCESS_2_MEMORY_WRITE_BIT specifies all write accesses. It is always valid in any access mask, and is treated as equivalent to setting all WRITE access flags that are valid where it is used.
- VK_ACCESS_2_INDIRECT_COMMAND_READ_BIT specifies read access to command data read from indirect buffers as part of an indirect build, trace, drawing or dispatch command. Such access occurs in the VK_PIPELINE_STAGE_2_DRAW_INDIRECT_BIT pipeline stage.
- VK_ACCESS_2_INDEX_READ_BIT specifies read access to an index buffer as part of an indexed drawing command, bound by vkCmdBindIndexBuffer2KHR and vkCmdBindIndexBuffer. Such access occurs in the VK_PIPELINE_STAGE_2_INDEX_INPUT_BIT pipeline stage.
- VK_ACCESS_2_VERTEX_ATTRIBUTE_READ_BIT specifies read access to a vertex buffer as part of a drawing command, bound by vkCmdBindVertexBuffers. Such access occurs in the VK_PIPELINE_STAGE_2_VERTEX_ATTRIBUTE_INPUT_BIT pipeline stage.
- VK_ACCESS_2_UNIFORM_READ_BIT specifies read access to a uniform buffer in any shader pipeline stage.
- VK_ACCESS_2_INPUT_ATTACHMENT_READ_BIT specifies read access to an input attachment within a render pass during subpass shading or fragment shading. Such access occurs in the VK_PIPELINE_STAGE_2_SUBPASS_SHADER_BIT_HUAWEI or VK_PIPELINE_STAGE_2_FRAGMENT_SHADER_BIT pipeline stage.
- VK_ACCESS_2_SHADER_SAMPLED_READ_BIT specifies read access to a uniform texel buffer or sampled image in any shader pipeline stage.
- VK_ACCESS_2_SHADER_STORAGE_READ_BIT specifies read access to a storage buffer, physical storage buffer, storage texel buffer, or storage image in any shader pipeline stage.
- VK_ACCESS_2_SHADER_BINDING_TABLE_READ_BIT_KHR specifies read access to a shader binding table in any shader pipeline stage.
- VK_ACCESS_2_SHADER_READ_BIT is equivalent to the logical OR of:
  - VK_ACCESS_2_SHADER_SAMPLED_READ_BIT
  - VK_ACCESS_2_SHADER_STORAGE_READ_BIT
- VK_ACCESS_2_SHADER_STORAGE_WRITE_BIT specifies write access to a storage buffer, physical storage

buffer, storage texel buffer, or storage image in any shader pipeline stage.

- VK_ACCESS_2_SHADER_WRITE_BIT is equivalent to VK_ACCESS_2_SHADER_STORAGE_WRITE_BIT.
- VK_ACCESS_2_COLOR_ATTACHMENT_READ_BIT specifies read access to a color attachment, such as via blending (other than advanced blend operations), logic operations or certain render pass load operations in the VK_PIPELINE_STAGE_2_COLOR_ATTACHMENT_OUTPUT_BIT pipeline stage or via fragment shader tile image reads in the VK_PIPELINE_STAGE_2_FRAGMENT_SHADER_BIT pipeline stage.
- VK_ACCESS_2_COLOR_ATTACHMENT_WRITE_BIT specifies write access to a color attachment during a render pass or via certain render pass load, store, and multisample resolve operations. Such access occurs in the VK_PIPELINE_STAGE_2_COLOR_ATTACHMENT_OUTPUT_BIT pipeline stage.
- VK_ACCESS_2_DEPTH_STENCIL_ATTACHMENT_READ_BIT specifies read access to a depth/stencil attachment, via depth or stencil operations or certain render pass load operations in the VK_PIPELINE_STAGE_2_EARLY_FRAGMENT_TESTS_BIT or VK_PIPELINE_STAGE_2_LATE_FRAGMENT_TESTS_BIT pipeline stages or via fragment shader tile image reads in the VK_PIPELINE_STAGE_2_FRAGMENT_SHADER_BIT pipeline stage.
- VK_ACCESS_2_DEPTH_STENCIL_ATTACHMENT_WRITE_BIT specifies write access to a depth/stencil attachment, via depth or stencil operations or certain render pass load and store operations. Such access occurs in the VK_PIPELINE_STAGE_2_EARLY_FRAGMENT_TESTS_BIT or VK_PIPELINE_STAGE_2_LATE_FRAGMENT_TESTS_BIT pipeline stages.
- VK_ACCESS_2_TRANSFER_READ_BIT specifies read access to an image or buffer in a copy operation. Such access occurs in the VK_PIPELINE_STAGE_2_COPY_BIT, VK_PIPELINE_STAGE_2_BLIT_BIT, or VK_PIPELINE_STAGE_2_RESOLVE_BIT pipeline stages.
- VK_ACCESS_2_TRANSFER_WRITE_BIT specifies write access to an image or buffer in a clear or copy operation. Such access occurs in the VK_PIPELINE_STAGE_2_COPY_BIT, VK_PIPELINE_STAGE_2_BLIT_BIT, VK_PIPELINE_STAGE_2_CLEAR_BIT, or VK_PIPELINE_STAGE_2_RESOLVE_BIT pipeline stages.
- VK_ACCESS_2_HOST_READ_BIT specifies read access by a host operation. Accesses of this type are not performed through a resource, but directly on memory. Such access occurs in the VK_PIPELINE_STAGE_2_HOST_BIT pipeline stage.
- VK_ACCESS_2_HOST_WRITE_BIT specifies write access by a host operation. Accesses of this type are not performed through a resource, but directly on memory. Such access occurs in the VK_PIPELINE_STAGE_2_HOST_BIT pipeline stage.
- VK_ACCESS_2_CONDITIONAL_RENDERING_READ_BIT_EXT specifies read access to a predicate as part of conditional rendering. Such access occurs in the VK_PIPELINE_STAGE_2_CONDITIONAL_RENDERING_BIT_EXT pipeline stage.
- VK_ACCESS_2_TRANSFORM_FEEDBACK_WRITE_BIT_EXT specifies write access to a transform feedback buffer made when transform feedback is active. Such access occurs in the VK_PIPELINE_STAGE_2_TRANSFORM_FEEDBACK_BIT_EXT pipeline stage.
- VK_ACCESS_2_TRANSFORM_FEEDBACK_COUNTER_READ_BIT_EXT specifies read access to a transform feedback counter buffer which is read when vkCmdBeginTransformFeedbackEXT executes. Such access occurs in the VK_PIPELINE_STAGE_2_TRANSFORM_FEEDBACK_BIT_EXT pipeline stage.
- VK_ACCESS_2_TRANSFORM_FEEDBACK_COUNTER_WRITE_BIT_EXT specifies write access to a transform feedback counter buffer which is written when vkCmdEndTransformFeedbackEXT executes. Such access occurs in the VK_PIPELINE_STAGE_2_TRANSFORM_FEEDBACK_BIT_EXT pipeline stage.

- VK_ACCESS_2_COMMAND_PREPROCESS_READ_BIT_NV specifies reads from buffer inputs to vkCmdPreprocessGeneratedCommandsNV. Such access occurs in the VK_PIPELINE_STAGE_2_COMMAND_PREPROCESS_BIT_NV pipeline stage.
- VK_ACCESS_2_COMMAND_PREPROCESS_WRITE_BIT_NV specifies writes to the target command buffer preprocess outputs. Such access occurs in the VK_PIPELINE_STAGE_2_COMMAND_PREPROCESS_BIT_NV pipeline stage.
- VK_ACCESS_2_COLOR_ATTACHMENT_READ_NONCOHERENT_BIT_EXT specifies read access to color attachments, including advanced blend operations. Such access occurs in the VK_PIPELINE_STAGE_2_COLOR_ATTACHMENT_OUTPUT_BIT pipeline stage.
- VK_ACCESS_2_INVOCATION_MASK_READ_BIT_HUAWEI specifies read access to an invocation mask image in the VK_PIPELINE_STAGE_2_INVOCATION_MASK_BIT_HUAWEI pipeline stage.
- VK_ACCESS_2_ACCELERATION_STRUCTURE_READ_BIT_KHR specifies read access to an acceleration structure as part of a trace, build, or copy command, or to an acceleration structure scratch buffer as part of a build command. Such access occurs in the VK_PIPELINE_STAGE_2_RAY_TRACING_SHADER_BIT_KHR pipeline stage or VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_BUILD_BIT_KHR pipeline stage.
- VK_ACCESS_2_ACCELERATION_STRUCTURE_WRITE_BIT_KHR specifies write access to an acceleration structure or acceleration structure scratch buffer as part of a build or copy command. Such access occurs in the VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_BUILD_BIT_KHR pipeline stage.
- VK_ACCESS_2_FRAGMENT_DENSITY_MAP_READ_BIT_EXT specifies read access to a fragment density map attachment during dynamic fragment density map operations. Such access occurs in the VK_PIPELINE_STAGE_2_FRAGMENT_DENSITY_PROCESS_BIT_EXT pipeline stage.
- VK_ACCESS_2_FRAGMENT_SHADING_RATE_ATTACHMENT_READ_BIT_KHR specifies read access to a fragment shading rate attachment during rasterization. Such access occurs in the VK_PIPELINE_STAGE_2_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR pipeline stage.
- VK_ACCESS_2_SHADING_RATE_IMAGE_READ_BIT_NV specifies read access to a shading rate image during rasterization. Such access occurs in the VK_PIPELINE_STAGE_2_SHADING_RATE_IMAGE_BIT_NV pipeline stage. It is equivalent to VK_ACCESS_2_FRAGMENT_SHADING_RATE_ATTACHMENT_READ_BIT_KHR.
- VK_ACCESS_2_VIDEO_DECODE_READ_BIT_KHR specifies read access to an image or buffer resource in a video decode operation. Such access occurs in the VK_PIPELINE_STAGE_2_VIDEO_DECODE_BIT_KHR pipeline stage.
- VK_ACCESS_2_VIDEO_DECODE_WRITE_BIT_KHR specifies write access to an image or buffer resource in a video decode operation. Such access occurs in the VK_PIPELINE_STAGE_2_VIDEO_DECODE_BIT_KHR pipeline stage.
- VK_ACCESS_2_VIDEO_ENCODE_READ_BIT_KHR specifies read access to an image or buffer resource in a video encode operation. Such access occurs in the VK_PIPELINE_STAGE_2_VIDEO_ENCODE_BIT_KHR pipeline stage.
- VK_ACCESS_2_VIDEO_ENCODE_WRITE_BIT_KHR specifies write access to an image or buffer resource in a video encode operation. Such access occurs in the VK_PIPELINE_STAGE_2_VIDEO_ENCODE_BIT_KHR pipeline stage.
- VK_ACCESS_2_DESCRIPTOR_BUFFER_READ_BIT_EXT specifies read access to a descriptor buffer in any shader pipeline stage.

- VK_ACCESS_2_OPTICAL_FLOW_READ_BIT_NV specifies read access to an image or buffer resource as part of a optical flow operation. Such access occurs in the VK_PIPELINE_STAGE_2_OPTICAL_FLOW_BIT_NV pipeline stage.
- VK_ACCESS_2_OPTICAL_FLOW_WRITE_BIT_NV specifies write access to an image or buffer resource as part of a optical flow operation. Such access occurs in the VK_PIPELINE_STAGE_2_OPTICAL_FLOW_BIT_NV pipeline stage.
- VK_ACCESS_2_MICROMAP_WRITE_BIT_EXT specifies write access to a micromap object. Such access occurs in the VK_PIPELINE_STAGE_2_MICROMAP_BUILD_BIT_EXT pipeline stage.
- VK_ACCESS_2_MICROMAP_READ_BIT_EXT specifies read access to a micromap object. Such access occurs in the VK_PIPELINE_STAGE_2_MICROMAP_BUILD_BIT_EXT and VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_BUILD_BIT_KHR pipeline stages.

**NOTE** In situations where an application wishes to select all access types for a given set of pipeline stages, VK_ACCESS_2_MEMORY_READ_BIT or VK_ACCESS_2_MEMORY_WRITE_BIT can be used. This is particularly useful when specifying stages that only have a single access type.

**NOTE** The VkAccessFlags2 bitmask goes beyond the 31 individual bit flags allowable within a C99 enum, which is how VkAccessFlagBits is defined. The first 31 values are common to both, and are interchangeable.

VkAccessFlags2 is a bitmask type for setting a mask of zero or more VkAccessFlagBits2:

```
typedef VkFlags64 VkAccessFlags2;
```

or the equivalent

```
// Provided by VK_KHR_synchronization2
typedef VkAccessFlags2 VkAccessFlags2KHR;
```

Bits which **can** be set in the srcAccessMask and dstAccessMask members of VkSubpassDependency, VkSubpassDependency2, VkMemoryBarrier, VkBufferMemoryBarrier, and VkImageMemoryBarrier, specifying access behavior, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkAccessFlagBits {
    VK_ACCESS_INDIRECT_COMMAND_READ_BIT = 0x00000001,
    VK_ACCESS_INDEX_READ_BIT = 0x00000002,
    VK_ACCESS_VERTEX_ATTRIBUTE_READ_BIT = 0x00000004,
    VK_ACCESS_UNIFORM_READ_BIT = 0x00000008,
    VK_ACCESS_INPUT_ATTACHMENT_READ_BIT = 0x00000010,
    VK_ACCESS_SHADER_READ_BIT = 0x00000020,
    VK_ACCESS_SHADER_WRITE_BIT = 0x00000040,
    VK_ACCESS_COLOR_ATTACHMENT_READ_BIT = 0x00000080,
    VK_ACCESS_COLOR_ATTACHMENT_WRITE_BIT = 0x00000100,
```

```
    VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_READ_BIT = 0x00000200,
    VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_WRITE_BIT = 0x00000400,
    VK_ACCESS_TRANSFER_READ_BIT = 0x00000800,
    VK_ACCESS_TRANSFER_WRITE_BIT = 0x00001000,
    VK_ACCESS_HOST_READ_BIT = 0x00002000,
    VK_ACCESS_HOST_WRITE_BIT = 0x00004000,
    VK_ACCESS_MEMORY_READ_BIT = 0x00008000,
    VK_ACCESS_MEMORY_WRITE_BIT = 0x00010000,
    VK_ACCESS_NONE = 0,
  // Provided by VK_EXT_transform_feedback
    VK_ACCESS_TRANSFORM_FEEDBACK_WRITE_BIT_EXT = 0x02000000,
  // Provided by VK_EXT_transform_feedback
    VK_ACCESS_TRANSFORM_FEEDBACK_COUNTER_READ_BIT_EXT = 0x04000000,
  // Provided by VK_EXT_transform_feedback
    VK_ACCESS_TRANSFORM_FEEDBACK_COUNTER_WRITE_BIT_EXT = 0x08000000,
  // Provided by VK_EXT_conditional_rendering
    VK_ACCESS_CONDITIONAL_RENDERING_READ_BIT_EXT = 0x00100000,
  // Provided by VK_EXT_blend_operation_advanced
    VK_ACCESS_COLOR_ATTACHMENT_READ_NONCOHERENT_BIT_EXT = 0x00080000,
  // Provided by VK_KHR_acceleration_structure
    VK_ACCESS_ACCELERATION_STRUCTURE_READ_BIT_KHR = 0x00200000,
  // Provided by VK_KHR_acceleration_structure
    VK_ACCESS_ACCELERATION_STRUCTURE_WRITE_BIT_KHR = 0x00400000,
  // Provided by VK_EXT_fragment_density_map
    VK_ACCESS_FRAGMENT_DENSITY_MAP_READ_BIT_EXT = 0x01000000,
  // Provided by VK_KHR_fragment_shading_rate
    VK_ACCESS_FRAGMENT_SHADING_RATE_ATTACHMENT_READ_BIT_KHR = 0x00800000,
  // Provided by VK_NV_device_generated_commands
    VK_ACCESS_COMMAND_PREPROCESS_READ_BIT_NV = 0x00020000,
  // Provided by VK_NV_device_generated_commands
    VK_ACCESS_COMMAND_PREPROCESS_WRITE_BIT_NV = 0x00040000,
  // Provided by VK_NV_shading_rate_image
    VK_ACCESS_SHADING_RATE_IMAGE_READ_BIT_NV =
VK_ACCESS_FRAGMENT_SHADING_RATE_ATTACHMENT_READ_BIT_KHR,
  // Provided by VK_NV_ray_tracing
    VK_ACCESS_ACCELERATION_STRUCTURE_READ_BIT_NV =
VK_ACCESS_ACCELERATION_STRUCTURE_READ_BIT_KHR,
  // Provided by VK_NV_ray_tracing
    VK_ACCESS_ACCELERATION_STRUCTURE_WRITE_BIT_NV =
VK_ACCESS_ACCELERATION_STRUCTURE_WRITE_BIT_KHR,
  // Provided by VK_KHR_synchronization2
    VK_ACCESS_NONE_KHR = VK_ACCESS_NONE,
} VkAccessFlagBits;
```

These values all have the same meaning as the equivalently named values for VkAccessFlags2.

- `VK_ACCESS_NONE` specifies no accesses.
- `VK_ACCESS_MEMORY_READ_BIT` specifies all read accesses. It is always valid in any access mask, and is treated as equivalent to setting all `READ` access flags that are valid where it is used.

- VK_ACCESS_MEMORY_WRITE_BIT specifies all write accesses. It is always valid in any access mask, and is treated as equivalent to setting all WRITE access flags that are valid where it is used.
- VK_ACCESS_INDIRECT_COMMAND_READ_BIT specifies read access to indirect command data read as part of an indirect build, trace, drawing or dispatching command. Such access occurs in the VK_PIPELINE_STAGE_DRAW_INDIRECT_BIT pipeline stage.
- VK_ACCESS_INDEX_READ_BIT specifies read access to an index buffer as part of an indexed drawing command, bound by vkCmdBindIndexBuffer2KHR and vkCmdBindIndexBuffer. Such access occurs in the VK_PIPELINE_STAGE_VERTEX_INPUT_BIT pipeline stage.
- VK_ACCESS_VERTEX_ATTRIBUTE_READ_BIT specifies read access to a vertex buffer as part of a drawing command, bound by vkCmdBindVertexBuffers. Such access occurs in the VK_PIPELINE_STAGE_VERTEX_INPUT_BIT pipeline stage.
- VK_ACCESS_UNIFORM_READ_BIT specifies read access to a uniform buffer in any shader pipeline stage.
- VK_ACCESS_INPUT_ATTACHMENT_READ_BIT specifies read access to an input attachment within a render pass during subpass shading or fragment shading. Such access occurs in the VK_PIPELINE_STAGE_2_SUBPASS_SHADER_BIT_HUAWEI or VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT pipeline stage.
- VK_ACCESS_SHADER_READ_BIT specifies read access to a uniform texel buffer, sampled image, storage buffer, physical storage buffer, shader binding table, storage texel buffer, or storage image in any shader pipeline stage.
- VK_ACCESS_SHADER_WRITE_BIT specifies write access to a storage buffer, physical storage buffer, storage texel buffer, or storage image in any shader pipeline stage.
- VK_ACCESS_COLOR_ATTACHMENT_READ_BIT specifies read access to a color attachment, such as via blending (other than advanced blend operations), logic operations or certain render pass load operations in the VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT pipeline stage or via fragment shader tile image reads in the VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT pipeline stage.
- VK_ACCESS_COLOR_ATTACHMENT_WRITE_BIT specifies write access to a color, resolve, or depth/stencil resolve attachment during a render pass or via certain render pass load and store operations. Such access occurs in the VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT pipeline stage.
- VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_READ_BIT specifies read access to a depth/stencil attachment, via depth or stencil operations or certain render pass load operations in the VK_PIPELINE_STAGE_EARLY_FRAGMENT_TESTS_BIT or VK_PIPELINE_STAGE_LATE_FRAGMENT_TESTS_BIT pipeline stages or via fragment shader tile image reads in the VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT pipeline stage.
- VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_WRITE_BIT specifies write access to a depth/stencil attachment, via depth or stencil operations or certain render pass load and store operations. Such access occurs in the VK_PIPELINE_STAGE_EARLY_FRAGMENT_TESTS_BIT or VK_PIPELINE_STAGE_LATE_FRAGMENT_TESTS_BIT pipeline stages.
- VK_ACCESS_TRANSFER_READ_BIT specifies read access to an image or buffer in a copy operation. Such access occurs in the VK_PIPELINE_STAGE_2_ALL_TRANSFER_BIT pipeline stage.
- VK_ACCESS_TRANSFER_WRITE_BIT specifies write access to an image or buffer in a clear or copy operation. Such access occurs in the VK_PIPELINE_STAGE_2_ALL_TRANSFER_BIT pipeline stage.

- VK_ACCESS_HOST_READ_BIT specifies read access by a host operation. Accesses of this type are not performed through a resource, but directly on memory. Such access occurs in the VK_PIPELINE_STAGE_HOST_BIT pipeline stage.
- VK_ACCESS_HOST_WRITE_BIT specifies write access by a host operation. Accesses of this type are not performed through a resource, but directly on memory. Such access occurs in the VK_PIPELINE_STAGE_HOST_BIT pipeline stage.
- VK_ACCESS_CONDITIONAL_RENDERING_READ_BIT_EXT specifies read access to a predicate as part of conditional rendering. Such access occurs in the VK_PIPELINE_STAGE_CONDITIONAL_RENDERING_BIT_EXT pipeline stage.
- VK_ACCESS_TRANSFORM_FEEDBACK_WRITE_BIT_EXT specifies write access to a transform feedback buffer made when transform feedback is active. Such access occurs in the VK_PIPELINE_STAGE_TRANSFORM_FEEDBACK_BIT_EXT pipeline stage.
- VK_ACCESS_TRANSFORM_FEEDBACK_COUNTER_READ_BIT_EXT specifies read access to a transform feedback counter buffer which is read when vkCmdBeginTransformFeedbackEXT executes. Such access occurs in the VK_PIPELINE_STAGE_TRANSFORM_FEEDBACK_BIT_EXT pipeline stage.
- VK_ACCESS_TRANSFORM_FEEDBACK_COUNTER_WRITE_BIT_EXT specifies write access to a transform feedback counter buffer which is written when vkCmdEndTransformFeedbackEXT executes. Such access occurs in the VK_PIPELINE_STAGE_TRANSFORM_FEEDBACK_BIT_EXT pipeline stage.
- VK_ACCESS_COMMAND_PREPROCESS_READ_BIT_NV specifies reads from buffer inputs to vkCmdPreprocessGeneratedCommandsNV. Such access occurs in the VK_PIPELINE_STAGE_COMMAND_PREPROCESS_BIT_NV pipeline stage.
- VK_ACCESS_COMMAND_PREPROCESS_WRITE_BIT_NV specifies writes to the target command buffer preprocess outputs in vkCmdPreprocessGeneratedCommandsNV. Such access occurs in the VK_PIPELINE_STAGE_COMMAND_PREPROCESS_BIT_NV pipeline stage.
- VK_ACCESS_COLOR_ATTACHMENT_READ_NONCOHERENT_BIT_EXT specifies read access to color attachments, including advanced blend operations. Such access occurs in the VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT pipeline stage.
- VK_ACCESS_2_INVOCATION_MASK_READ_BIT_HUAWEI specifies read access to an invocation mask image in the VK_PIPELINE_STAGE_2_INVOCATION_MASK_BIT_HUAWEI pipeline stage.
- VK_ACCESS_ACCELERATION_STRUCTURE_READ_BIT_KHR specifies read access to an acceleration structure as part of a trace, build, or copy command, or to an acceleration structure scratch buffer as part of a build command. Such access occurs in the VK_PIPELINE_STAGE_RAY_TRACING_SHADER_BIT_KHR pipeline stage or VK_PIPELINE_STAGE_ACCELERATION_STRUCTURE_BUILD_BIT_KHR pipeline stage.
- VK_ACCESS_ACCELERATION_STRUCTURE_WRITE_BIT_KHR specifies write access to an acceleration structure or acceleration structure scratch buffer as part of a build or copy command. Such access occurs in the VK_PIPELINE_STAGE_ACCELERATION_STRUCTURE_BUILD_BIT_KHR pipeline stage.
- VK_ACCESS_FRAGMENT_DENSITY_MAP_READ_BIT_EXT specifies read access to a fragment density map attachment during dynamic fragment density map operations Such access occurs in the VK_PIPELINE_STAGE_FRAGMENT_DENSITY_PROCESS_BIT_EXT pipeline stage.
- VK_ACCESS_FRAGMENT_SHADING_RATE_ATTACHMENT_READ_BIT_KHR specifies read access to a fragment shading rate attachment during rasterization. Such access occurs in the VK_PIPELINE_STAGE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR pipeline stage.

- VK_ACCESS_SHADING_RATE_IMAGE_READ_BIT_NV specifies read access to a shading rate image during rasterization. Such access occurs in the VK_PIPELINE_STAGE_SHADING_RATE_IMAGE_BIT_NV pipeline stage. It is equivalent to VK_PIPELINE_STAGE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR.

Certain access types are only performed by a subset of pipeline stages. Any synchronization command that takes both stage masks and access masks uses both to define the access scopes - only the specified access types performed by the specified stages are included in the access scope. An application **must** not specify an access flag in a synchronization command if it does not include a pipeline stage in the corresponding stage mask that is able to perform accesses of that type. The following table lists, for each access flag, which pipeline stages **can** perform that type of access.

*Table 4. Supported access types*

| Access flag | Supported pipeline stages |
|---|---|
| VK_ACCESS_2_NONE | Any |
| VK_ACCESS_2_INDIRECT_COMMAND_READ_BIT | VK_PIPELINE_STAGE_2_DRAW_INDIRECT_BIT, VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_BUILD_BIT_KHR |
| VK_ACCESS_2_INDEX_READ_BIT | VK_PIPELINE_STAGE_2_VERTEX_INPUT_BIT, VK_PIPELINE_STAGE_2_INDEX_INPUT_BIT |
| VK_ACCESS_2_VERTEX_ATTRIBUTE_READ_BIT | VK_PIPELINE_STAGE_2_VERTEX_INPUT_BIT, VK_PIPELINE_STAGE_2_VERTEX_ATTRIBUTE_INPUT_BIT |
| VK_ACCESS_2_UNIFORM_READ_BIT | VK_PIPELINE_STAGE_2_VERTEX_SHADER_BIT, VK_PIPELINE_STAGE_2_TESSELLATION_CONTROL_SHADER_BIT, VK_PIPELINE_STAGE_2_TESSELLATION_EVALUATION_SHADER_BIT, VK_PIPELINE_STAGE_2_GEOMETRY_SHADER_BIT, VK_PIPELINE_STAGE_2_FRAGMENT_SHADER_BIT, VK_PIPELINE_STAGE_2_COMPUTE_SHADER_BIT, VK_PIPELINE_STAGE_2_RAY_TRACING_SHADER_BIT_KHR, VK_PIPELINE_STAGE_2_TASK_SHADER_BIT_EXT, VK_PIPELINE_STAGE_2_MESH_SHADER_BIT_EXT, VK_PIPELINE_STAGE_2_SUBPASS_SHADER_BIT_HUAWEI, VK_PIPELINE_STAGE_2_CLUSTER_CULLING_SHADER_BIT_HUAWEI |
| VK_ACCESS_2_INPUT_ATTACHMENT_READ_BIT | VK_PIPELINE_STAGE_2_FRAGMENT_SHADER_BIT, VK_PIPELINE_STAGE_2_SUBPASS_SHADER_BIT_HUAWEI |

| Access flag | Supported pipeline stages |
| --- | --- |
| VK_ACCESS_2_SHADER_READ_BIT | VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_BUILD_BIT_KHR, VK_PIPELINE_STAGE_2_MICROMAP_BUILD_BIT_EXT, VK_PIPELINE_STAGE_2_VERTEX_SHADER_BIT, VK_PIPELINE_STAGE_2_TESSELLATION_CONTROL_SHADER_BIT, VK_PIPELINE_STAGE_2_TESSELLATION_EVALUATION_SHADER_BIT, VK_PIPELINE_STAGE_2_GEOMETRY_SHADER_BIT, VK_PIPELINE_STAGE_2_FRAGMENT_SHADER_BIT, VK_PIPELINE_STAGE_2_COMPUTE_SHADER_BIT, VK_PIPELINE_STAGE_2_RAY_TRACING_SHADER_BIT_KHR, VK_PIPELINE_STAGE_2_TASK_SHADER_BIT_EXT, VK_PIPELINE_STAGE_2_MESH_SHADER_BIT_EXT, VK_PIPELINE_STAGE_2_SUBPASS_SHADER_BIT_HUAWEI, VK_PIPELINE_STAGE_2_CLUSTER_CULLING_SHADER_BIT_HUAWEI |
| VK_ACCESS_2_SHADER_WRITE_BIT | VK_PIPELINE_STAGE_2_VERTEX_SHADER_BIT, VK_PIPELINE_STAGE_2_TESSELLATION_CONTROL_SHADER_BIT, VK_PIPELINE_STAGE_2_TESSELLATION_EVALUATION_SHADER_BIT, VK_PIPELINE_STAGE_2_GEOMETRY_SHADER_BIT, VK_PIPELINE_STAGE_2_FRAGMENT_SHADER_BIT, VK_PIPELINE_STAGE_2_COMPUTE_SHADER_BIT, VK_PIPELINE_STAGE_2_RAY_TRACING_SHADER_BIT_KHR, VK_PIPELINE_STAGE_2_TASK_SHADER_BIT_EXT, VK_PIPELINE_STAGE_2_MESH_SHADER_BIT_EXT, VK_PIPELINE_STAGE_2_SUBPASS_SHADER_BIT_HUAWEI, VK_PIPELINE_STAGE_2_CLUSTER_CULLING_SHADER_BIT_HUAWEI |
| VK_ACCESS_2_COLOR_ATTACHMENT_READ_BIT | VK_PIPELINE_STAGE_2_FRAGMENT_SHADER_BIT, VK_PIPELINE_STAGE_2_COLOR_ATTACHMENT_OUTPUT_BIT |
| VK_ACCESS_2_COLOR_ATTACHMENT_WRITE_BIT | VK_PIPELINE_STAGE_2_COLOR_ATTACHMENT_OUTPUT_BIT |
| VK_ACCESS_2_DEPTH_STENCIL_ATTACHMENT_READ_BIT | VK_PIPELINE_STAGE_2_FRAGMENT_SHADER_BIT, VK_PIPELINE_STAGE_2_EARLY_FRAGMENT_TESTS_BIT, VK_PIPELINE_STAGE_2_LATE_FRAGMENT_TESTS_BIT |
| VK_ACCESS_2_DEPTH_STENCIL_ATTACHMENT_WRITE_BIT | VK_PIPELINE_STAGE_2_EARLY_FRAGMENT_TESTS_BIT, VK_PIPELINE_STAGE_2_LATE_FRAGMENT_TESTS_BIT |

| Access flag | Supported pipeline stages |
| --- | --- |
| VK_ACCESS_2_TRANSFER_READ_BIT | VK_PIPELINE_STAGE_2_ALL_TRANSFER_BIT, VK_PIPELINE_STAGE_2_COPY_BIT, VK_PIPELINE_STAGE_2_RESOLVE_BIT, VK_PIPELINE_STAGE_2_BLIT_BIT, VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_BUILD_BIT_KHR, VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_COPY_BIT_KHR, VK_PIPELINE_STAGE_2_MICROMAP_BUILD_BIT_EXT |
| VK_ACCESS_2_TRANSFER_WRITE_BIT | VK_PIPELINE_STAGE_2_ALL_TRANSFER_BIT, VK_PIPELINE_STAGE_2_COPY_BIT, VK_PIPELINE_STAGE_2_RESOLVE_BIT, VK_PIPELINE_STAGE_2_BLIT_BIT, VK_PIPELINE_STAGE_2_CLEAR_BIT, VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_BUILD_BIT_KHR, VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_COPY_BIT_KHR, VK_PIPELINE_STAGE_2_MICROMAP_BUILD_BIT_EXT |
| VK_ACCESS_2_HOST_READ_BIT | VK_PIPELINE_STAGE_2_HOST_BIT |
| VK_ACCESS_2_HOST_WRITE_BIT | VK_PIPELINE_STAGE_2_HOST_BIT |
| VK_ACCESS_2_MEMORY_READ_BIT | Any |
| VK_ACCESS_2_MEMORY_WRITE_BIT | Any |
| VK_ACCESS_2_SHADER_SAMPLED_READ_BIT | VK_PIPELINE_STAGE_2_VERTEX_SHADER_BIT, VK_PIPELINE_STAGE_2_TESSELLATION_CONTROL_SHADER_BIT, VK_PIPELINE_STAGE_2_TESSELLATION_EVALUATION_SHADER_BIT, VK_PIPELINE_STAGE_2_GEOMETRY_SHADER_BIT, VK_PIPELINE_STAGE_2_FRAGMENT_SHADER_BIT, VK_PIPELINE_STAGE_2_COMPUTE_SHADER_BIT, VK_PIPELINE_STAGE_2_RAY_TRACING_SHADER_BIT_KHR, VK_PIPELINE_STAGE_2_TASK_SHADER_BIT_EXT, VK_PIPELINE_STAGE_2_MESH_SHADER_BIT_EXT, VK_PIPELINE_STAGE_2_SUBPASS_SHADER_BIT_HUAWEI, VK_PIPELINE_STAGE_2_CLUSTER_CULLING_SHADER_BIT_HUAWEI |

| Access flag | Supported pipeline stages |
| --- | --- |
| VK_ACCESS_2_SHADER_STORAGE_READ_BIT | VK_PIPELINE_STAGE_2_VERTEX_SHADER_BIT, VK_PIPELINE_STAGE_2_TESSELLATION_CONTROL_SHADER_BIT, VK_PIPELINE_STAGE_2_TESSELLATION_EVALUATION_SHADER_BIT, VK_PIPELINE_STAGE_2_GEOMETRY_SHADER_BIT, VK_PIPELINE_STAGE_2_FRAGMENT_SHADER_BIT, VK_PIPELINE_STAGE_2_COMPUTE_SHADER_BIT, VK_PIPELINE_STAGE_2_RAY_TRACING_SHADER_BIT_KHR, VK_PIPELINE_STAGE_2_TASK_SHADER_BIT_EXT, VK_PIPELINE_STAGE_2_MESH_SHADER_BIT_EXT, VK_PIPELINE_STAGE_2_SUBPASS_SHADER_BIT_HUAWEI, VK_PIPELINE_STAGE_2_CLUSTER_CULLING_SHADER_BIT_HUAWEI |
| VK_ACCESS_2_SHADER_STORAGE_WRITE_BIT | VK_PIPELINE_STAGE_2_VERTEX_SHADER_BIT, VK_PIPELINE_STAGE_2_TESSELLATION_CONTROL_SHADER_BIT, VK_PIPELINE_STAGE_2_TESSELLATION_EVALUATION_SHADER_BIT, VK_PIPELINE_STAGE_2_GEOMETRY_SHADER_BIT, VK_PIPELINE_STAGE_2_FRAGMENT_SHADER_BIT, VK_PIPELINE_STAGE_2_COMPUTE_SHADER_BIT, VK_PIPELINE_STAGE_2_RAY_TRACING_SHADER_BIT_KHR, VK_PIPELINE_STAGE_2_TASK_SHADER_BIT_EXT, VK_PIPELINE_STAGE_2_MESH_SHADER_BIT_EXT, VK_PIPELINE_STAGE_2_SUBPASS_SHADER_BIT_HUAWEI, VK_PIPELINE_STAGE_2_CLUSTER_CULLING_SHADER_BIT_HUAWEI |
| VK_ACCESS_2_VIDEO_DECODE_READ_BIT_KHR | VK_PIPELINE_STAGE_2_VIDEO_DECODE_BIT_KHR |
| VK_ACCESS_2_VIDEO_DECODE_WRITE_BIT_KHR | VK_PIPELINE_STAGE_2_VIDEO_DECODE_BIT_KHR |
| VK_ACCESS_2_VIDEO_ENCODE_READ_BIT_KHR | VK_PIPELINE_STAGE_2_VIDEO_ENCODE_BIT_KHR |
| VK_ACCESS_2_VIDEO_ENCODE_WRITE_BIT_KHR | VK_PIPELINE_STAGE_2_VIDEO_ENCODE_BIT_KHR |
| VK_ACCESS_2_TRANSFORM_FEEDBACK_WRITE_BIT_EXT | VK_PIPELINE_STAGE_2_TRANSFORM_FEEDBACK_BIT_EXT |
| VK_ACCESS_2_TRANSFORM_FEEDBACK_COUNTER_READ_BIT_EXT | VK_PIPELINE_STAGE_2_DRAW_INDIRECT_BIT, VK_PIPELINE_STAGE_2_TRANSFORM_FEEDBACK_BIT_EXT |
| VK_ACCESS_2_TRANSFORM_FEEDBACK_COUNTER_WRITE_BIT_EXT | VK_PIPELINE_STAGE_2_TRANSFORM_FEEDBACK_BIT_EXT |
| VK_ACCESS_2_CONDITIONAL_RENDERING_READ_BIT_EXT | VK_PIPELINE_STAGE_2_CONDITIONAL_RENDERING_BIT_EXT |
| VK_ACCESS_2_COMMAND_PREPROCESS_READ_BIT_NV | VK_PIPELINE_STAGE_2_COMMAND_PREPROCESS_BIT_NV |
| VK_ACCESS_2_COMMAND_PREPROCESS_WRITE_BIT_NV | VK_PIPELINE_STAGE_2_COMMAND_PREPROCESS_BIT_NV |
| VK_ACCESS_2_FRAGMENT_SHADING_RATE_ATTACHMENT_READ_BIT_KHR | VK_PIPELINE_STAGE_2_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR |

| Access flag | Supported pipeline stages |
| --- | --- |
| VK_ACCESS_2_ACCELERATION_STRUCTURE_READ_BIT_KHR | VK_PIPELINE_STAGE_2_VERTEX_SHADER_BIT, VK_PIPELINE_STAGE_2_TESSELLATION_CONTROL_SHADER_BIT, VK_PIPELINE_STAGE_2_TESSELLATION_EVALUATION_SHADER_BIT, VK_PIPELINE_STAGE_2_GEOMETRY_SHADER_BIT, VK_PIPELINE_STAGE_2_FRAGMENT_SHADER_BIT, VK_PIPELINE_STAGE_2_COMPUTE_SHADER_BIT, VK_PIPELINE_STAGE_2_RAY_TRACING_SHADER_BIT_KHR, VK_PIPELINE_STAGE_2_TASK_SHADER_BIT_EXT, VK_PIPELINE_STAGE_2_MESH_SHADER_BIT_EXT, VK_PIPELINE_STAGE_2_CLUSTER_CULLING_SHADER_BIT_HUAWEI, VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_BUILD_BIT_KHR, VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_COPY_BIT_KHR, VK_PIPELINE_STAGE_2_SUBPASS_SHADER_BIT_HUAWEI |
| VK_ACCESS_2_ACCELERATION_STRUCTURE_WRITE_BIT_KHR | VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_BUILD_BIT_KHR, VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_COPY_BIT_KHR |
| VK_ACCESS_2_FRAGMENT_DENSITY_MAP_READ_BIT_EXT | VK_PIPELINE_STAGE_2_FRAGMENT_DENSITY_PROCESS_BIT_EXT |
| VK_ACCESS_2_COLOR_ATTACHMENT_READ_NONCOHERENT_BIT_EXT | VK_PIPELINE_STAGE_2_COLOR_ATTACHMENT_OUTPUT_BIT |
| VK_ACCESS_2_DESCRIPTOR_BUFFER_READ_BIT_EXT | VK_PIPELINE_STAGE_2_VERTEX_SHADER_BIT, VK_PIPELINE_STAGE_2_TESSELLATION_CONTROL_SHADER_BIT, VK_PIPELINE_STAGE_2_TESSELLATION_EVALUATION_SHADER_BIT, VK_PIPELINE_STAGE_2_GEOMETRY_SHADER_BIT, VK_PIPELINE_STAGE_2_FRAGMENT_SHADER_BIT, VK_PIPELINE_STAGE_2_COMPUTE_SHADER_BIT, VK_PIPELINE_STAGE_2_RAY_TRACING_SHADER_BIT_KHR, VK_PIPELINE_STAGE_2_TASK_SHADER_BIT_EXT, VK_PIPELINE_STAGE_2_MESH_SHADER_BIT_EXT, VK_PIPELINE_STAGE_2_SUBPASS_SHADER_BIT_HUAWEI, VK_PIPELINE_STAGE_2_CLUSTER_CULLING_SHADER_BIT_HUAWEI |
| VK_ACCESS_2_INVOCATION_MASK_READ_BIT_HUAWEI | VK_PIPELINE_STAGE_2_INVOCATION_MASK_BIT_HUAWEI |
| VK_ACCESS_2_MICROMAP_READ_BIT_EXT | VK_PIPELINE_STAGE_2_MICROMAP_BUILD_BIT_EXT, VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_BUILD_BIT_KHR |
| VK_ACCESS_2_MICROMAP_WRITE_BIT_EXT | VK_PIPELINE_STAGE_2_MICROMAP_BUILD_BIT_EXT |
| VK_ACCESS_2_OPTICAL_FLOW_READ_BIT_NV | VK_PIPELINE_STAGE_2_OPTICAL_FLOW_BIT_NV |
| VK_ACCESS_2_OPTICAL_FLOW_WRITE_BIT_NV | VK_PIPELINE_STAGE_2_OPTICAL_FLOW_BIT_NV |

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkAccessFlags;
```

`VkAccessFlags` is a bitmask type for setting a mask of zero or more VkAccessFlagBits.

If a memory object does not have the `VK_MEMORY_PROPERTY_HOST_COHERENT_BIT` property, then vkFlushMappedMemoryRanges **must** be called in order to guarantee that writes to the memory object from the host are made available to the host domain, where they **can** be further made available to the device domain via a domain operation. Similarly, vkInvalidateMappedMemoryRanges **must** be called to guarantee that writes which are available to the host domain are made visible to host operations.

If the memory object does have the `VK_MEMORY_PROPERTY_HOST_COHERENT_BIT` property flag, writes to the memory object from the host are automatically made available to the host domain. Similarly, writes made available to the host domain are automatically made visible to the host.

**NOTE** Queue submission commands automatically perform a domain operation from host to device for all writes performed before the command executes, so in most cases an explicit memory barrier is not needed for this case. In the few circumstances where a submit does not occur between the host write and the device read access, writes **can** be made available by using an explicit memory barrier.

### 7.1.4. Framebuffer Region Dependencies

Pipeline stages that operate on, or with respect to, the framebuffer are collectively the *framebuffer-space* pipeline stages. These stages are:

- `VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT`
- `VK_PIPELINE_STAGE_EARLY_FRAGMENT_TESTS_BIT`
- `VK_PIPELINE_STAGE_LATE_FRAGMENT_TESTS_BIT`
- `VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT`

For these pipeline stages, an execution or memory dependency from the first set of operations to the second set **can** either be a single *framebuffer-global* dependency, or split into multiple *framebuffer-local* dependencies. A dependency with non-framebuffer-space pipeline stages is neither framebuffer-global nor framebuffer-local.

A *framebuffer region* is a subset of the entire framebuffer, and **can** either be:

- A *sample region*, which is set of sample (x, y, layer, sample) coordinates that is a subset of the entire framebuffer, or
- A *fragment region*, which is a set of fragment (x, y, layer) coordinates that is a subset of the entire framebuffer.

Both synchronization scopes of a framebuffer-local dependency include only the operations performed within corresponding framebuffer regions (as defined below). No ordering guarantees are made between different framebuffer regions for a framebuffer-local dependency.

Both synchronization scopes of a framebuffer-global dependency include operations on all framebuffer-regions.

If the first synchronization scope includes operations on pixels/fragments with N samples and the second synchronization scope includes operations on pixels/fragments with M samples, where N does not equal M, then a framebuffer region containing all samples at a given (x, y, layer) coordinate in the first synchronization scope corresponds to a region containing all samples at the same coordinate in the second synchronization scope. In other words, the framebuffer region is a fragment region and it is a pixel granularity dependency. If N equals M, and if the `VkSubpassDescription::flags` does not specify the `VK_SUBPASS_DESCRIPTION_FRAGMENT_REGION_BIT_QCOM` flag, then a framebuffer region containing a single (x, y, layer, sample) coordinate in the first synchronization scope corresponds to a region containing the same sample at the same coordinate in the second synchronization scope. In other words, the framebuffer region is a sample region and it is a sample granularity dependency.

If the pipeline performing the operation was created with `VK_PIPELINE_COLOR_BLEND_STATE_CREATE_RASTERIZATION_ORDER_ATTACHMENT_ACCESS_BIT_EXT`, `VK_PIPELINE_DEPTH_STENCIL_STATE_CREATE_RASTERIZATION_ORDER_ATTACHMENT_DEPTH_ACCESS_BIT_EXT`, or `VK_PIPELINE_DEPTH_STENCIL_STATE_CREATE_RASTERIZATION_ORDER_ATTACHMENT_STENCIL_ACCESS_BIT_EXT`, the framebuffer region is a fragment region and it is a pixel granularity dependency.

**NOTE** Since fragment shader invocations are not specified to run in any particular groupings, the size of a framebuffer region is implementation-dependent, not known to the application, and **must** be assumed to be no larger than specified above.

**NOTE** Practically, the pixel vs. sample granularity dependency means that if an input attachment has a different number of samples than the pipeline's `rasterizationSamples`, then a fragment **can** access any sample in the input attachment's pixel even if it only uses framebuffer-local dependencies. If the input attachment has the same number of samples, then the fragment **can** only access the covered samples in its input `SampleMask` (i.e. the fragment operations happen-after a framebuffer-local dependency for each sample the fragment covers). To access samples that are not covered, either the `VkSubpassDescription::flags` `VK_SUBPASS_DESCRIPTION_FRAGMENT_REGION_BIT_QCOM` flag is required, or a framebuffer-global dependency is required.

If a synchronization command includes a `dependencyFlags` parameter, and specifies the `VK_DEPENDENCY_BY_REGION_BIT` flag, then it defines framebuffer-local dependencies for the framebuffer-space pipeline stages in that synchronization command, for all framebuffer regions. If no `dependencyFlags` parameter is included, or the `VK_DEPENDENCY_BY_REGION_BIT` flag is not specified, then a framebuffer-global dependency is specified for those stages. The `VK_DEPENDENCY_BY_REGION_BIT` flag does not affect the dependencies between non-framebuffer-space pipeline stages, nor does it affect the dependencies between framebuffer-space and non-framebuffer-space pipeline stages.

**NOTE** Framebuffer-local dependencies are more efficient for most architectures; particularly tile-based architectures - which can keep framebuffer-regions entirely

> in on-chip registers and thus avoid external bandwidth across such a dependency. Including a framebuffer-global dependency in your rendering will usually force all implementations to flush data to memory, or to a higher level cache, breaking any potential locality optimizations.

### 7.1.5. View-Local Dependencies

In a render pass instance that has multiview enabled, dependencies **can** be either view-local or view-global.

A view-local dependency only includes operations from a single source view from the source subpass in the first synchronization scope, and only includes operations from a single destination view from the destination subpass in the second synchronization scope. A view-global dependency includes all views in the view mask of the source and destination subpasses in the corresponding synchronization scopes.

If a synchronization command includes a `dependencyFlags` parameter and specifies the `VK_DEPENDENCY_VIEW_LOCAL_BIT` flag, then it defines view-local dependencies for that synchronization command, for all views. If no `dependencyFlags` parameter is included or the `VK_DEPENDENCY_VIEW_LOCAL_BIT` flag is not specified, then a view-global dependency is specified.

### 7.1.6. Device-Local Dependencies

Dependencies **can** be either device-local or non-device-local. A device-local dependency acts as multiple separate dependencies, one for each physical device that executes the synchronization command, where each dependency only includes operations from that physical device in both synchronization scopes. A non-device-local dependency is a single dependency where both synchronization scopes include operations from all physical devices that participate in the synchronization command. For subpass dependencies, all physical devices in the VkDeviceGroupRenderPassBeginInfo::`deviceMask` participate in the dependency, and for pipeline barriers all physical devices that are set in the command buffer's current device mask participate in the dependency.

If a synchronization command includes a `dependencyFlags` parameter and specifies the `VK_DEPENDENCY_DEVICE_GROUP_BIT` flag, then it defines a non-device-local dependency for that synchronization command. If no `dependencyFlags` parameter is included or the `VK_DEPENDENCY_DEVICE_GROUP_BIT` flag is not specified, then it defines device-local dependencies for that synchronization command, for all participating physical devices.

Semaphore and event dependencies are device-local and only execute on the one physical device that performs the dependency.

## 7.2. Implicit Synchronization Guarantees

A small number of implicit ordering guarantees are provided by Vulkan, ensuring that the order in which commands are submitted is meaningful, and avoiding unnecessary complexity in common operations.

*Submission order* is a fundamental ordering in Vulkan, giving meaning to the order in which action and synchronization commands are recorded and submitted to a single queue. Explicit and implicit ordering guarantees between commands in Vulkan all work on the premise that this ordering is meaningful. This order does not itself define any execution or memory dependencies; synchronization commands and other orderings within the API use this ordering to define their scopes.

Submission order for any given set of commands is based on the order in which they were recorded to command buffers and then submitted. This order is determined as follows:

1. The initial order is determined by the order in which vkQueueSubmit and vkQueueSubmit2KHR commands are executed on the host, for a single queue, from first to last.
2. The order in which VkSubmitInfo structures are specified in the `pSubmits` parameter of vkQueueSubmit, or in which VkSubmitInfo2 structures are specified in the `pSubmits` parameter of vkQueueSubmit2KHR, from lowest index to highest.
3. The order in which command buffers are specified in the `pCommandBuffers` member of VkSubmitInfo or VkSubmitInfo2 from lowest index to highest.
4. The order in which commands outside of a render pass were recorded to a command buffer on the host, from first to last.
5. The order in which commands inside a single subpass were recorded to a command buffer on the host, from first to last.

**NOTE** When using a render pass object with multiple subpasses, commands in different subpasses have no defined submission order relative to each other, regardless of the order in which the subpasses were recorded. Commands within a subpass are still ordered relative to other commands in the same subpass, and those outside of the render pass.

State commands do not execute any operations on the device, instead they set the state of the command buffer when they execute on the host, in the order that they are recorded. Action commands consume the current state of the command buffer when they are recorded, and will execute state changes on the device as required to match the recorded state.

The order of primitives passing through the graphics pipeline and image layout transitions as part of an image memory barrier provide additional guarantees based on submission order.

Execution of pipeline stages within a given command also has a loose ordering, dependent only on a single command.

*Signal operation order* is a fundamental ordering in Vulkan, giving meaning to the order in which semaphore and fence signal operations occur when submitted to a single queue. The signal operation order for queue operations is determined as follows:

1. The initial order is determined by the order in which vkQueueSubmit and vkQueueSubmit2KHR commands are executed on the host, for a single queue, from first to last.
2. The order in which VkSubmitInfo structures are specified in the `pSubmits` parameter of vkQueueSubmit, or in which VkSubmitInfo2 structures are specified in the `pSubmits` parameter

of vkQueueSubmit2KHR, from lowest index to highest.

3. The fence signal operation defined by the `fence` parameter of a vkQueueSubmit or vkQueueSubmit2KHR or vkQueueBindSparse command is ordered after all semaphore signal operations defined by that command.

Semaphore signal operations defined by a single VkSubmitInfo or VkSubmitInfo2 or VkBindSparseInfo structure are unordered with respect to other semaphore signal operations defined within the same structure.

The vkSignalSemaphoreKHR command does not execute on a queue but instead performs the signal operation from the host. The semaphore signal operation defined by executing a vkSignalSemaphoreKHR command happens-after the vkSignalSemaphoreKHR command is invoked and happens-before the command returns.

**NOTE**

When signaling timeline semaphores, it is the responsibility of the application to ensure that they are ordered such that the semaphore value is strictly increasing. Because the first synchronization scope for a semaphore signal operation contains all semaphore signal operations which occur earlier in submission order, all semaphore signal operations contained in any given batch are guaranteed to happen-after all semaphore signal operations contained in any previous batches. However, no ordering guarantee is provided between the semaphore signal operations defined within a single batch. This, combined with the requirement that timeline semaphore values strictly increase, means that it is invalid to signal the same timeline semaphore twice within a single batch.

If an application wishes to ensure that some semaphore signal operation happens-after some other semaphore signal operation, it can submit a separate batch containing only semaphore signal operations, which will happen-after the semaphore signal operations in any earlier batches.

When signaling a semaphore from the host, the only ordering guarantee is that the signal operation happens-after when vkSignalSemaphoreKHR is called and happens-before it returns. Therefore, it is invalid to call `vkSignalSemaphoreKHR` while there are any outstanding signal operations on that semaphore from any queue submissions unless those queue submissions have some dependency which ensures that they happen-after the host signal operation. One example of this would be if the pending signal operation is, itself, waiting on the same semaphore at a lower value and the call to `vkSignalSemaphoreKHR` signals that lower value. Furthermore, if there are two or more processes or threads signaling the same timeline semaphore from the host, the application must ensure that the `vkSignalSemaphoreKHR` with the lower semaphore value returns before `vkSignalSemaphoreKHR` is called with the higher value.

## 7.3. Fences

Fences are a synchronization primitive that **can** be used to insert a dependency from a queue to the host. Fences have two states - signaled and unsignaled. A fence **can** be signaled as part of the execution of a queue submission command. Fences **can** be unsignaled on the host with

vkResetFences. Fences **can** be waited on by the host with the vkWaitForFences command, and the current state **can** be queried with vkGetFenceStatus.

The internal data of a fence **may** include a reference to any resources and pending work associated with signal or unsignal operations performed on that fence object, collectively referred to as the fence's *payload*. Mechanisms to import and export that internal data to and from fences are provided below. These mechanisms indirectly enable applications to share fence state between two or more fences and other synchronization primitives across process and API boundaries.

Fences are represented by `VkFence` handles:

```
// Provided by VK_VERSION_1_0
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkFence)
```

To create a fence, call:

```
// Provided by VK_VERSION_1_0
VkResult vkCreateFence(
    VkDevice                                    device,
    const VkFenceCreateInfo*                    pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkFence*                                    pFence);
```

- `device` is the logical device that creates the fence.
- `pCreateInfo` is a pointer to a VkFenceCreateInfo structure containing information about how the fence is to be created.
- `pAllocator` controls host memory allocation as described in the Memory Allocation chapter.
- `pFence` is a pointer to a handle in which the resulting fence object is returned.

### Valid Usage (Implicit)

- VUID-vkCreateFence-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkCreateFence-pCreateInfo-parameter
  `pCreateInfo` **must** be a valid pointer to a valid VkFenceCreateInfo structure
- VUID-vkCreateFence-pAllocator-parameter
  If `pAllocator` is not `NULL`, `pAllocator` **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkCreateFence-pFence-parameter
  `pFence` **must** be a valid pointer to a VkFence handle

### Return Codes

**Success**

- VK_SUCCESS

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY

The VkFenceCreateInfo structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkFenceCreateInfo {
    VkStructureType       sType;
    const void*           pNext;
    VkFenceCreateFlags    flags;
} VkFenceCreateInfo;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- flags is a bitmask of VkFenceCreateFlagBits specifying the initial state and behavior of the fence.

### Valid Usage (Implicit)

- VUID-VkFenceCreateInfo-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_FENCE_CREATE_INFO
- VUID-VkFenceCreateInfo-pNext-pNext
  Each pNext member of any structure (including this one) in the pNext chain **must** be either NULL or a pointer to a valid instance of VkExportFenceCreateInfo or VkExportFenceWin32HandleInfoKHR
- VUID-VkFenceCreateInfo-sType-unique
  The sType value of each struct in the pNext chain **must** be unique
- VUID-VkFenceCreateInfo-flags-parameter
  flags **must** be a valid combination of VkFenceCreateFlagBits values

```
// Provided by VK_VERSION_1_0
typedef enum VkFenceCreateFlagBits {
    VK_FENCE_CREATE_SIGNALED_BIT = 0x00000001,
} VkFenceCreateFlagBits;
```

- VK_FENCE_CREATE_SIGNALED_BIT specifies that the fence object is created in the signaled state.

Otherwise, it is created in the unsignaled state.

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkFenceCreateFlags;
```

VkFenceCreateFlags is a bitmask type for setting a mask of zero or more VkFenceCreateFlagBits.

To create a fence whose payload **can** be exported to external handles, add a VkExportFenceCreateInfo structure to the pNext chain of the VkFenceCreateInfo structure. The VkExportFenceCreateInfo structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkExportFenceCreateInfo {
    VkStructureType                     sType;
    const void*                         pNext;
    VkExternalFenceHandleTypeFlags      handleTypes;
} VkExportFenceCreateInfo;
```

or the equivalent

```
// Provided by VK_KHR_external_fence
typedef VkExportFenceCreateInfo VkExportFenceCreateInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- handleTypes is a bitmask of VkExternalFenceHandleTypeFlagBits specifying one or more fence handle types the application **can** export from the resulting fence. The application **can** request multiple handle types for the same fence.

### Valid Usage

- VUID-VkExportFenceCreateInfo-handleTypes-01446
  The bits in handleTypes **must** be supported and compatible, as reported by VkExternalFenceProperties

### Valid Usage (Implicit)

- VUID-VkExportFenceCreateInfo-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_EXPORT_FENCE_CREATE_INFO
- VUID-VkExportFenceCreateInfo-handleTypes-parameter
  handleTypes **must** be a valid combination of VkExternalFenceHandleTypeFlagBits values

To specify additional attributes of NT handles exported from a fence, add a

VkExportFenceWin32HandleInfoKHR structure to the pNext chain of the VkFenceCreateInfo structure. The VkExportFenceWin32HandleInfoKHR structure is defined as:

```
// Provided by VK_KHR_external_fence_win32
typedef struct VkExportFenceWin32HandleInfoKHR {
    VkStructureType                 sType;
    const void*                     pNext;
    const SECURITY_ATTRIBUTES*      pAttributes;
    DWORD                           dwAccess;
    LPCWSTR                         name;
} VkExportFenceWin32HandleInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- pAttributes is a pointer to a Windows SECURITY_ATTRIBUTES structure specifying security attributes of the handle.
- dwAccess is a DWORD specifying access rights of the handle.
- name is a null-terminated UTF-16 string to associate with the underlying synchronization primitive referenced by NT handles exported from the created fence.

If VkExportFenceCreateInfo is not included in the same pNext chain, this structure is ignored.

If VkExportFenceCreateInfo is included in the pNext chain of VkFenceCreateInfo with a Windows handleType, but either VkExportFenceWin32HandleInfoKHR is not included in the pNext chain, or it is included but pAttributes is set to NULL, default security descriptor values will be used, and child processes created by the application will not inherit the handle, as described in the MSDN documentation for "Synchronization Object Security and Access Rights"[1]. Further, if the structure is not present, the access rights will be

DXGI_SHARED_RESOURCE_READ | DXGI_SHARED_RESOURCE_WRITE

for handles of the following types:

VK_EXTERNAL_FENCE_HANDLE_TYPE_OPAQUE_WIN32_BIT

**1**

https://docs.microsoft.com/en-us/windows/win32/sync/synchronization-object-security-and-access-rights

### Valid Usage

- VUID-VkExportFenceWin32HandleInfoKHR-handleTypes-01447
  If VkExportFenceCreateInfo::handleTypes does not include VK_EXTERNAL_FENCE_HANDLE_TYPE_OPAQUE_WIN32_BIT, a VkExportFenceWin32HandleInfoKHR structure **must** not be included in the pNext chain of VkFenceCreateInfo

### Valid Usage (Implicit)

- VUID-VkExportFenceWin32HandleInfoKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_EXPORT_FENCE_WIN32_HANDLE_INFO_KHR`
- VUID-VkExportFenceWin32HandleInfoKHR-pAttributes-parameter
  If `pAttributes` is not `NULL`, `pAttributes` **must** be a valid pointer to a valid `SECURITY_ATTRIBUTES` value

To export a Windows handle representing the state of a fence, call:

```
// Provided by VK_KHR_external_fence_win32
VkResult vkGetFenceWin32HandleKHR(
    VkDevice                                    device,
    const VkFenceGetWin32HandleInfoKHR*         pGetWin32HandleInfo,
    HANDLE*                                     pHandle);
```

- `device` is the logical device that created the fence being exported.
- `pGetWin32HandleInfo` is a pointer to a VkFenceGetWin32HandleInfoKHR structure containing parameters of the export operation.
- `pHandle` will return the Windows handle representing the fence state.

For handle types defined as NT handles, the handles returned by `vkGetFenceWin32HandleKHR` are owned by the application. To avoid leaking resources, the application **must** release ownership of them using the `CloseHandle` system call when they are no longer needed.

Exporting a Windows handle from a fence **may** have side effects depending on the transference of the specified handle type, as described in Importing Fence Payloads.

### Valid Usage (Implicit)

- VUID-vkGetFenceWin32HandleKHR-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkGetFenceWin32HandleKHR-pGetWin32HandleInfo-parameter
  `pGetWin32HandleInfo` **must** be a valid pointer to a valid VkFenceGetWin32HandleInfoKHR structure
- VUID-vkGetFenceWin32HandleKHR-pHandle-parameter
  `pHandle` **must** be a valid pointer to a `HANDLE` value

### Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- VK_ERROR_TOO_MANY_OBJECTS
- VK_ERROR_OUT_OF_HOST_MEMORY

The VkFenceGetWin32HandleInfoKHR structure is defined as:

```
// Provided by VK_KHR_external_fence_win32
typedef struct VkFenceGetWin32HandleInfoKHR {
    VkStructureType                     sType;
    const void*                         pNext;
    VkFence                             fence;
    VkExternalFenceHandleTypeFlagBits   handleType;
} VkFenceGetWin32HandleInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- fence is the fence from which state will be exported.
- handleType is a VkExternalFenceHandleTypeFlagBits value specifying the type of handle requested.

The properties of the handle returned depend on the value of handleType. See VkExternalFenceHandleTypeFlagBits for a description of the properties of the defined external fence handle types.

## Valid Usage

- VUID-VkFenceGetWin32HandleInfoKHR-handleType-01448
  handleType **must** have been included in VkExportFenceCreateInfo::handleTypes when the fence's current payload was created
- VUID-VkFenceGetWin32HandleInfoKHR-handleType-01449
  If handleType is defined as an NT handle, vkGetFenceWin32HandleKHR **must** be called no more than once for each valid unique combination of fence and handleType
- VUID-VkFenceGetWin32HandleInfoKHR-fence-01450
  fence **must** not currently have its payload replaced by an imported payload as described below in Importing Fence Payloads unless that imported payload's handle type was included in VkExternalFenceProperties::exportFromImportedHandleTypes for handleType
- VUID-VkFenceGetWin32HandleInfoKHR-handleType-01451
  If handleType refers to a handle type with copy payload transference semantics, fence **must** be signaled, or have an associated fence signal operation pending execution
- VUID-VkFenceGetWin32HandleInfoKHR-handleType-01452
  handleType **must** be defined as an NT handle or a global share handle

### Valid Usage (Implicit)

- VUID-VkFenceGetWin32HandleInfoKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_FENCE_GET_WIN32_HANDLE_INFO_KHR`
- VUID-VkFenceGetWin32HandleInfoKHR-pNext-pNext
  `pNext` **must** be `NULL`
- VUID-VkFenceGetWin32HandleInfoKHR-fence-parameter
  `fence` **must** be a valid VkFence handle
- VUID-VkFenceGetWin32HandleInfoKHR-handleType-parameter
  `handleType` **must** be a valid VkExternalFenceHandleTypeFlagBits value

To export a POSIX file descriptor representing the payload of a fence, call:

```
// Provided by VK_KHR_external_fence_fd
VkResult vkGetFenceFdKHR(
    VkDevice                                    device,
    const VkFenceGetFdInfoKHR*                  pGetFdInfo,
    int*                                        pFd);
```

- `device` is the logical device that created the fence being exported.
- `pGetFdInfo` is a pointer to a VkFenceGetFdInfoKHR structure containing parameters of the export operation.
- `pFd` will return the file descriptor representing the fence payload.

Each call to `vkGetFenceFdKHR` **must** create a new file descriptor and transfer ownership of it to the application. To avoid leaking resources, the application **must** release ownership of the file descriptor when it is no longer needed.

| | |
|---|---|
| **NOTE** | Ownership can be released in many ways. For example, the application can call `close`() on the file descriptor, or transfer ownership back to Vulkan by using the file descriptor to import a fence payload. |

If `pGetFdInfo->handleType` is `VK_EXTERNAL_FENCE_HANDLE_TYPE_SYNC_FD_BIT` and the fence is signaled at the time `vkGetFenceFdKHR` is called, `pFd` **may** return the value `-1` instead of a valid file descriptor.

Where supported by the operating system, the implementation **must** set the file descriptor to be closed automatically when an `execve` system call is made.

Exporting a file descriptor from a fence **may** have side effects depending on the transference of the specified handle type, as described in Importing Fence State.

### Valid Usage (Implicit)

- VUID-vkGetFenceFdKHR-device-parameter

device **must** be a valid VkDevice handle

- VUID-vkGetFenceFdKHR-pGetFdInfo-parameter
  pGetFdInfo **must** be a valid pointer to a valid VkFenceGetFdInfoKHR structure
- VUID-vkGetFenceFdKHR-pFd-parameter
  pFd **must** be a valid pointer to an int value

## Return Codes

**Success**

- VK_SUCCESS

**Failure**

- VK_ERROR_TOO_MANY_OBJECTS
- VK_ERROR_OUT_OF_HOST_MEMORY

The VkFenceGetFdInfoKHR structure is defined as:

```
// Provided by VK_KHR_external_fence_fd
typedef struct VkFenceGetFdInfoKHR {
    VkStructureType                     sType;
    const void*                         pNext;
    VkFence                             fence;
    VkExternalFenceHandleTypeFlagBits   handleType;
} VkFenceGetFdInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- fence is the fence from which state will be exported.
- handleType is a VkExternalFenceHandleTypeFlagBits value specifying the type of handle requested.

The properties of the file descriptor returned depend on the value of handleType. See VkExternalFenceHandleTypeFlagBits for a description of the properties of the defined external fence handle types.

## Valid Usage

- VUID-VkFenceGetFdInfoKHR-handleType-01453
  handleType **must** have been included in VkExportFenceCreateInfo::handleTypes when fence's current payload was created
- VUID-VkFenceGetFdInfoKHR-handleType-01454
  If handleType refers to a handle type with copy payload transference semantics, fence **must** be signaled, or have an associated fence signal operation pending execution

- VUID-VkFenceGetFdInfoKHR-fence-01455
  `fence` **must** not currently have its payload replaced by an imported payload as described below in Importing Fence Payloads unless that imported payload's handle type was included in VkExternalFenceProperties::`exportFromImportedHandleTypes` for `handleType`
- VUID-VkFenceGetFdInfoKHR-handleType-01456
  `handleType` **must** be defined as a POSIX file descriptor handle

### Valid Usage (Implicit)

- VUID-VkFenceGetFdInfoKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_FENCE_GET_FD_INFO_KHR`
- VUID-VkFenceGetFdInfoKHR-pNext-pNext
  `pNext` **must** be `NULL`
- VUID-VkFenceGetFdInfoKHR-fence-parameter
  `fence` **must** be a valid VkFence handle
- VUID-VkFenceGetFdInfoKHR-handleType-parameter
  `handleType` **must** be a valid VkExternalFenceHandleTypeFlagBits value

To destroy a fence, call:

```
// Provided by VK_VERSION_1_0
void vkDestroyFence(
    VkDevice                                    device,
    VkFence                                     fence,
    const VkAllocationCallbacks*                pAllocator);
```

- `device` is the logical device that destroys the fence.
- `fence` is the handle of the fence to destroy.
- `pAllocator` controls host memory allocation as described in the Memory Allocation chapter.

### Valid Usage

- VUID-vkDestroyFence-fence-01120
  All queue submission commands that refer to `fence` **must** have completed execution
- VUID-vkDestroyFence-fence-01121
  If `VkAllocationCallbacks` were provided when `fence` was created, a compatible set of callbacks **must** be provided here
- VUID-vkDestroyFence-fence-01122
  If no `VkAllocationCallbacks` were provided when `fence` was created, `pAllocator` **must** be `NULL`

### Valid Usage (Implicit)

- VUID-vkDestroyFence-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkDestroyFence-fence-parameter
  If `fence` is not VK_NULL_HANDLE, `fence` **must** be a valid VkFence handle
- VUID-vkDestroyFence-pAllocator-parameter
  If `pAllocator` is not `NULL`, `pAllocator` **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkDestroyFence-fence-parent
  If `fence` is a valid handle, it **must** have been created, allocated, or retrieved from `device`

### Host Synchronization

- Host access to `fence` **must** be externally synchronized

To query the status of a fence from the host, call:

```
// Provided by VK_VERSION_1_0
VkResult vkGetFenceStatus(
    VkDevice                                    device,
    VkFence                                     fence);
```

- `device` is the logical device that owns the fence.
- `fence` is the handle of the fence to query.

Upon success, `vkGetFenceStatus` returns the status of the fence object, with the following return codes:

*Table 5. Fence Object Status Codes*

| Status | Meaning |
|---|---|
| `VK_SUCCESS` | The fence specified by `fence` is signaled. |
| `VK_NOT_READY` | The fence specified by `fence` is unsignaled. |
| `VK_ERROR_DEVICE_LOST` | The device has been lost. See Lost Device. |

If a queue submission command is pending execution, then the value returned by this command **may** immediately be out of date.

If the device has been lost (see Lost Device), `vkGetFenceStatus` **may** return any of the above status

codes. If the device has been lost and vkGetFenceStatus is called repeatedly, it will eventually return either VK_SUCCESS or VK_ERROR_DEVICE_LOST.

### Valid Usage (Implicit)

- VUID-vkGetFenceStatus-device-parameter
  device **must** be a valid VkDevice handle
- VUID-vkGetFenceStatus-fence-parameter
  fence **must** be a valid VkFence handle
- VUID-vkGetFenceStatus-fence-parent
  fence **must** have been created, allocated, or retrieved from device

### Return Codes

**Success**

- VK_SUCCESS
- VK_NOT_READY

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY
- VK_ERROR_DEVICE_LOST

To set the state of fences to unsignaled from the host, call:

```
// Provided by VK_VERSION_1_0
VkResult vkResetFences(
    VkDevice                                    device,
    uint32_t                                    fenceCount,
    const VkFence*                              pFences);
```

- device is the logical device that owns the fences.
- fenceCount is the number of fences to reset.
- pFences is a pointer to an array of fence handles to reset.

If any member of pFences currently has its payload imported with temporary permanence, that fence's prior permanent payload is first restored. The remaining operations described therefore operate on the restored payload.

When vkResetFences is executed on the host, it defines a *fence unsignal operation* for each fence, which resets the fence to the unsignaled state.

If any member of pFences is already in the unsignaled state when vkResetFences is executed, then

vkResetFences has no effect on that fence.

### Valid Usage

- VUID-vkResetFences-pFences-01123
  Each element of `pFences` **must** not be currently associated with any queue command that has not yet completed execution on that queue

### Valid Usage (Implicit)

- VUID-vkResetFences-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkResetFences-pFences-parameter
  `pFences` **must** be a valid pointer to an array of `fenceCount` valid VkFence handles
- VUID-vkResetFences-fenceCount-arraylength
  `fenceCount` **must** be greater than `0`
- VUID-vkResetFences-pFences-parent
  Each element of `pFences` **must** have been created, allocated, or retrieved from `device`

### Host Synchronization

- Host access to each member of `pFences` **must** be externally synchronized

### Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- `VK_ERROR_OUT_OF_DEVICE_MEMORY`

When a fence is submitted to a queue as part of a queue submission command, it defines a memory dependency on the batches that were submitted as part of that command, and defines a *fence signal operation* which sets the fence to the signaled state.

The first synchronization scope includes every batch submitted in the same queue submission command. Fence signal operations that are defined by vkQueueSubmit or vkQueueSubmit2KHR additionally include in the first synchronization scope all commands that occur earlier in submission order. Fence signal operations that are defined by vkQueueSubmit or vkQueueSubmit2KHR or vkQueueBindSparse additionally include in the first synchronization scope any semaphore and fence signal operations that occur earlier in signal operation order.

The second synchronization scope only includes the fence signal operation.

The first access scope includes all memory access performed by the device.

The second access scope is empty.

To wait for one or more fences to enter the signaled state on the host, call:

```
// Provided by VK_VERSION_1_0
VkResult vkWaitForFences(
    VkDevice                                    device,
    uint32_t                                    fenceCount,
    const VkFence*                              pFences,
    VkBool32                                    waitAll,
    uint64_t                                    timeout);
```

- `device` is the logical device that owns the fences.
- `fenceCount` is the number of fences to wait on.
- `pFences` is a pointer to an array of `fenceCount` fence handles.
- `waitAll` is the condition that **must** be satisfied to successfully unblock the wait. If `waitAll` is `VK_TRUE`, then the condition is that all fences in `pFences` are signaled. Otherwise, the condition is that at least one fence in `pFences` is signaled.
- `timeout` is the timeout period in units of nanoseconds. `timeout` is adjusted to the closest value allowed by the implementation-dependent timeout accuracy, which **may** be substantially longer than one nanosecond, and **may** be longer than the requested period.

If the condition is satisfied when `vkWaitForFences` is called, then `vkWaitForFences` returns immediately. If the condition is not satisfied at the time `vkWaitForFences` is called, then `vkWaitForFences` will block and wait until the condition is satisfied or the `timeout` has expired, whichever is sooner.

If `timeout` is zero, then `vkWaitForFences` does not wait, but simply returns the current state of the fences. `VK_TIMEOUT` will be returned in this case if the condition is not satisfied, even though no actual wait was performed.

If the condition is satisfied before the `timeout` has expired, `vkWaitForFences` returns `VK_SUCCESS`. Otherwise, `vkWaitForFences` returns `VK_TIMEOUT` after the `timeout` has expired.

If device loss occurs (see Lost Device) before the timeout has expired, `vkWaitForFences` **must** return in finite time with either `VK_SUCCESS` or `VK_ERROR_DEVICE_LOST`.

| | |
|---|---|
| **NOTE** | While we guarantee that `vkWaitForFences` **must** return in finite time, no guarantees are made that it returns immediately upon device loss. However, the application can reasonably expect that the delay will be on the order of seconds and that calling `vkWaitForFences` will not result in a permanently (or seemingly permanently) dead process. |

### Valid Usage (Implicit)

- VUID-vkWaitForFences-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkWaitForFences-pFences-parameter
  `pFences` **must** be a valid pointer to an array of `fenceCount` valid VkFence handles
- VUID-vkWaitForFences-fenceCount-arraylength
  `fenceCount` **must** be greater than `0`
- VUID-vkWaitForFences-pFences-parent
  Each element of `pFences` **must** have been created, allocated, or retrieved from `device`

### Return Codes

**Success**

- `VK_SUCCESS`
- `VK_TIMEOUT`

**Failure**

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_DEVICE_LOST`

An execution dependency is defined by waiting for a fence to become signaled, either via vkWaitForFences or by polling on vkGetFenceStatus.

The first synchronization scope includes only the fence signal operation.

The second synchronization scope includes the host operations of vkWaitForFences or vkGetFenceStatus indicating that the fence has become signaled.

| | |
|---|---|
| **NOTE** | Signaling a fence and waiting on the host does not guarantee that the results of memory accesses will be visible to the host, as the access scope of a memory dependency defined by a fence only includes device access. A memory barrier or other memory dependency **must** be used to guarantee this. See the description of host access types for more information. |

## 7.3.1. Alternate Methods to Signal Fences

Besides submitting a fence to a queue as part of a queue submission command, a fence **may** also be signaled when a particular event occurs on a device or display.

To create a fence that will be signaled when an event occurs on a device, call:

```
// Provided by VK_EXT_display_control
VkResult vkRegisterDeviceEventEXT(
    VkDevice                                    device,
    const VkDeviceEventInfoEXT*                 pDeviceEventInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkFence*                                    pFence);
```

- `device` is a logical device on which the event **may** occur.
- `pDeviceEventInfo` is a pointer to a VkDeviceEventInfoEXT structure describing the event of interest to the application.
- `pAllocator` controls host memory allocation as described in the Memory Allocation chapter.
- `pFence` is a pointer to a handle in which the resulting fence object is returned.

### Valid Usage (Implicit)

- VUID-vkRegisterDeviceEventEXT-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkRegisterDeviceEventEXT-pDeviceEventInfo-parameter
  `pDeviceEventInfo` **must** be a valid pointer to a valid VkDeviceEventInfoEXT structure
- VUID-vkRegisterDeviceEventEXT-pAllocator-parameter
  If `pAllocator` is not `NULL`, `pAllocator` **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkRegisterDeviceEventEXT-pFence-parameter
  `pFence` **must** be a valid pointer to a VkFence handle

### Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- `VK_ERROR_OUT_OF_HOST_MEMORY`

The `VkDeviceEventInfoEXT` structure is defined as:

```
// Provided by VK_EXT_display_control
typedef struct VkDeviceEventInfoEXT {
    VkStructureType         sType;
    const void*             pNext;
    VkDeviceEventTypeEXT    deviceEvent;
} VkDeviceEventInfoEXT;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- device is a VkDeviceEventTypeEXT value specifying when the fence will be signaled.

### Valid Usage (Implicit)

- VUID-VkDeviceEventInfoEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_DEVICE_EVENT_INFO_EXT
- VUID-VkDeviceEventInfoEXT-pNext-pNext
  pNext **must** be NULL
- VUID-VkDeviceEventInfoEXT-deviceEvent-parameter
  deviceEvent **must** be a valid VkDeviceEventTypeEXT value

Possible values of VkDeviceEventInfoEXT::device, specifying when a fence will be signaled, are:

```
// Provided by VK_EXT_display_control
typedef enum VkDeviceEventTypeEXT {
    VK_DEVICE_EVENT_TYPE_DISPLAY_HOTPLUG_EXT = 0,
} VkDeviceEventTypeEXT;
```

- VK_DEVICE_EVENT_TYPE_DISPLAY_HOTPLUG_EXT specifies that the fence is signaled when a display is plugged into or unplugged from the specified device. Applications **can** use this notification to determine when they need to re-enumerate the available displays on a device.

To create a fence that will be signaled when an event occurs on a VkDisplayKHR object, call:

```
// Provided by VK_EXT_display_control
VkResult vkRegisterDisplayEventEXT(
    VkDevice                                    device,
    VkDisplayKHR                                display,
    const VkDisplayEventInfoEXT*                pDisplayEventInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkFence*                                    pFence);
```

- device is a logical device associated with display
- display is the display on which the event **may** occur.
- pDisplayEventInfo is a pointer to a VkDisplayEventInfoEXT structure describing the event of interest to the application.
- pAllocator controls host memory allocation as described in the Memory Allocation chapter.
- pFence is a pointer to a handle in which the resulting fence object is returned.

### Valid Usage (Implicit)

- VUID-vkRegisterDisplayEventEXT-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkRegisterDisplayEventEXT-display-parameter
  `display` **must** be a valid VkDisplayKHR handle
- VUID-vkRegisterDisplayEventEXT-pDisplayEventInfo-parameter
  `pDisplayEventInfo` **must** be a valid pointer to a valid VkDisplayEventInfoEXT structure
- VUID-vkRegisterDisplayEventEXT-pAllocator-parameter
  If `pAllocator` is not `NULL`, `pAllocator` **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkRegisterDisplayEventEXT-pFence-parameter
  `pFence` **must** be a valid pointer to a VkFence handle
- VUID-vkRegisterDisplayEventEXT-commonparent
  Both of `device`, and `display` **must** have been created, allocated, or retrieved from the same VkPhysicalDevice

### Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- `VK_ERROR_OUT_OF_HOST_MEMORY`

The `VkDisplayEventInfoEXT` structure is defined as:

```
// Provided by VK_EXT_display_control
typedef struct VkDisplayEventInfoEXT {
    VkStructureType          sType;
    const void*              pNext;
    VkDisplayEventTypeEXT    displayEvent;
} VkDisplayEventInfoEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `displayEvent` is a VkDisplayEventTypeEXT specifying when the fence will be signaled.

### Valid Usage (Implicit)

- VUID-VkDisplayEventInfoEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_DISPLAY_EVENT_INFO_EXT`

- VUID-VkDisplayEventInfoEXT-pNext-pNext
  `pNext` **must** be `NULL`
- VUID-VkDisplayEventInfoEXT-displayEvent-parameter
  `displayEvent` **must** be a valid VkDisplayEventTypeEXT value

Possible values of VkDisplayEventInfoEXT::`displayEvent`, specifying when a fence will be signaled, are:

```
// Provided by VK_EXT_display_control
typedef enum VkDisplayEventTypeEXT {
    VK_DISPLAY_EVENT_TYPE_FIRST_PIXEL_OUT_EXT = 0,
} VkDisplayEventTypeEXT;
```

- `VK_DISPLAY_EVENT_TYPE_FIRST_PIXEL_OUT_EXT` specifies that the fence is signaled when the first pixel of the next display refresh cycle leaves the display engine for the display.

### 7.3.2. Importing Fence Payloads

Applications **can** import a fence payload into an existing fence using an external fence handle. The effects of the import operation will be either temporary or permanent, as specified by the application. If the import is temporary, the fence will be *restored* to its permanent state the next time that fence is passed to vkResetFences.

| | |
|---|---|
| **NOTE** | Restoring a fence to its prior permanent payload is a distinct operation from resetting a fence payload. See vkResetFences for more detail. |

Performing a subsequent temporary import on a fence before resetting it has no effect on this requirement; the next unsignal of the fence **must** still restore its last permanent state. A permanent payload import behaves as if the target fence was destroyed, and a new fence was created with the same handle but the imported payload. Because importing a fence payload temporarily or permanently detaches the existing payload from a fence, similar usage restrictions to those applied to `vkDestroyFence` are applied to any command that imports a fence payload. Which of these import types is used is referred to as the import operation's *permanence*. Each handle type supports either one or both types of permanence.

The implementation **must** perform the import operation by either referencing or copying the payload referred to by the specified external fence handle, depending on the handle's type. The import method used is referred to as the handle type's *transference*. When using handle types with reference transference, importing a payload to a fence adds the fence to the set of all fences sharing that payload. This set includes the fence from which the payload was exported. Fence signaling, waiting, and resetting operations performed on any fence in the set **must** behave as if the set were a single fence. Importing a payload using handle types with copy transference creates a duplicate copy of the payload at the time of import, but makes no further reference to it. Fence signaling, waiting, and resetting operations performed on the target of copy imports **must** not affect any other fence or payload.

Export operations have the same transference as the specified handle type's import operations.

Additionally, exporting a fence payload to a handle with copy transference has the same side effects on the source fence's payload as executing a fence reset operation. If the fence was using a temporarily imported payload, the fence's prior permanent payload will be restored.

**NOTE** The tables Handle Types Supported by `VkImportFenceWin32HandleInfoKHR` and Handle Types Supported by `VkImportFenceFdInfoKHR` define the permanence and transference of each handle type.

External synchronization allows implementations to modify an object's internal state, i.e. payload, without internal synchronization. However, for fences sharing a payload across processes, satisfying the external synchronization requirements of `VkFence` parameters as if all fences in the set were the same object is sometimes infeasible. Satisfying valid usage constraints on the state of a fence would similarly require impractical coordination or levels of trust between processes. Therefore, these constraints only apply to a specific fence handle, not to its payload. For distinct fence objects which share a payload:

- If multiple commands which queue a signal operation, or which unsignal a fence, are called concurrently, behavior will be as if the commands were called in an arbitrary sequential order.
- If a queue submission command is called with a fence that is sharing a payload, and the payload is already associated with another queue command that has not yet completed execution, either one or both of the commands will cause the fence to become signaled when they complete execution.
- If a fence payload is reset while it is associated with a queue command that has not yet completed execution, the payload will become unsignaled, but **may** become signaled again when the command completes execution.
- In the preceding cases, any of the devices associated with the fences sharing the payload **may** be lost, or any of the queue submission or fence reset commands **may** return `VK_ERROR_INITIALIZATION_FAILED`.

Other than these non-deterministic results, behavior is well defined. In particular:

- The implementation **must** not crash or enter an internally inconsistent state where future valid Vulkan commands might cause undefined results,
- Timeouts on future wait commands on fences sharing the payload **must** be effective.

**NOTE** These rules allow processes to synchronize access to shared memory without trusting each other. However, such processes must still be cautious not to use the shared fence for more than synchronizing access to the shared memory. For example, a process should not use a fence with shared payload to tell when commands it submitted to a queue have completed and objects used by those commands may be destroyed, since the other process can accidentally or maliciously cause the fence to signal before the commands actually complete.

When a fence is using an imported payload, its VkExportFenceCreateInfo::`handleTypes` value is specified when creating the fence from which the payload was exported, rather than specified when creating the fence. Additionally, VkExternalFenceProperties::`exportFromImportedHandleTypes` restricts which handle types **can** be exported from such a fence based on the specific handle type

used to import the current payload. Passing a fence to vkAcquireNextImageKHR is equivalent to temporarily importing a fence payload to that fence.

**NOTE** Because the exportable handle types of an imported fence correspond to its current imported payload, and vkAcquireNextImageKHR behaves the same as a temporary import operation for which the source fence is opaque to the application, applications have no way of determining whether any external handle types **can** be exported from a fence in this state. Therefore, applications **must** not attempt to export handles from fences using a temporarily imported payload from vkAcquireNextImageKHR.

When importing a fence payload, it is the responsibility of the application to ensure the external handles meet all valid usage requirements. However, implementations **must** perform sufficient validation of external handles to ensure that the operation results in a valid fence which will not cause program termination, device loss, queue stalls, host thread stalls, or corruption of other resources when used as allowed according to its import parameters. If the external handle provided does not meet these requirements, the implementation **must** fail the fence payload import operation with the error code `VK_ERROR_INVALID_EXTERNAL_HANDLE`.

To import a fence payload from a Windows handle, call:

```
// Provided by VK_KHR_external_fence_win32
VkResult vkImportFenceWin32HandleKHR(
    VkDevice                                    device,
    const VkImportFenceWin32HandleInfoKHR*      pImportFenceWin32HandleInfo);
```

- `device` is the logical device that created the fence.
- `pImportFenceWin32HandleInfo` is a pointer to a VkImportFenceWin32HandleInfoKHR structure specifying the fence and import parameters.

Importing a fence payload from Windows handles does not transfer ownership of the handle to the Vulkan implementation. For handle types defined as NT handles, the application **must** release ownership using the `CloseHandle` system call when the handle is no longer needed.

Applications **can** import the same fence payload into multiple instances of Vulkan, into the same instance from which it was exported, and multiple times into a given Vulkan instance.

### Valid Usage

- VUID-vkImportFenceWin32HandleKHR-fence-04448
  `fence` **must** not be associated with any queue command that has not yet completed execution on that queue

### Valid Usage (Implicit)

- VUID-vkImportFenceWin32HandleKHR-device-parameter

device **must** be a valid VkDevice handle

- VUID-vkImportFenceWin32HandleKHR-pImportFenceWin32HandleInfo-parameter pImportFenceWin32HandleInfo **must** be a valid pointer to a valid VkImportFenceWin32HandleInfoKHR structure

## Return Codes

**Success**

- VK_SUCCESS

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_INVALID_EXTERNAL_HANDLE

The VkImportFenceWin32HandleInfoKHR structure is defined as:

```
// Provided by VK_KHR_external_fence_win32
typedef struct VkImportFenceWin32HandleInfoKHR {
    VkStructureType                     sType;
    const void*                         pNext;
    VkFence                             fence;
    VkFenceImportFlags                  flags;
    VkExternalFenceHandleTypeFlagBits   handleType;
    HANDLE                              handle;
    LPCWSTR                             name;
} VkImportFenceWin32HandleInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- fence is the fence into which the state will be imported.
- flags is a bitmask of VkFenceImportFlagBits specifying additional parameters for the fence payload import operation.
- handleType is a VkExternalFenceHandleTypeFlagBits value specifying the type of handle.
- handle is NULL or the external handle to import.
- name is NULL or a null-terminated UTF-16 string naming the underlying synchronization primitive to import.

The handle types supported by handleType are:

*Table 6. Handle Types Supported by* VkImportFenceWin32HandleInfoKHR

| Handle Type | Transference | Permanence Supported |
|---|---|---|
| VK_EXTERNAL_FENCE_HANDLE_TYPE_OPAQUE_WIN32_BIT | Reference | Temporary,Permanent |

| Handle Type | Transference | Permanence Supported |
| --- | --- | --- |
| VK_EXTERNAL_FENCE_HANDLE_TYPE_OPAQUE_WIN32_KMT_BIT | Reference | Temporary,Permanent |

## Valid Usage

- VUID-VkImportFenceWin32HandleInfoKHR-handleType-01457
  handleType **must** be a value included in the Handle Types Supported by VkImportFenceWin32HandleInfoKHR table
- VUID-VkImportFenceWin32HandleInfoKHR-handleType-01459
  If handleType is not VK_EXTERNAL_FENCE_HANDLE_TYPE_OPAQUE_WIN32_BIT, name **must** be NULL
- VUID-VkImportFenceWin32HandleInfoKHR-handleType-01460
  If handle is NULL, name **must** name a valid synchronization primitive of the type specified by handleType
- VUID-VkImportFenceWin32HandleInfoKHR-handleType-01461
  If name is NULL, handle **must** be a valid handle of the type specified by handleType
- VUID-VkImportFenceWin32HandleInfoKHR-handle-01462
  If handle is not NULL, name **must** be NULL
- VUID-VkImportFenceWin32HandleInfoKHR-handle-01539
  If handle is not NULL, it **must** obey any requirements listed for handleType in external fence handle types compatibility
- VUID-VkImportFenceWin32HandleInfoKHR-name-01540
  If name is not NULL, it **must** obey any requirements listed for handleType in external fence handle types compatibility

## Valid Usage (Implicit)

- VUID-VkImportFenceWin32HandleInfoKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_IMPORT_FENCE_WIN32_HANDLE_INFO_KHR
- VUID-VkImportFenceWin32HandleInfoKHR-pNext-pNext
  pNext **must** be NULL
- VUID-VkImportFenceWin32HandleInfoKHR-fence-parameter
  fence **must** be a valid VkFence handle
- VUID-VkImportFenceWin32HandleInfoKHR-flags-parameter
  flags **must** be a valid combination of VkFenceImportFlagBits values

## Host Synchronization

- Host access to fence **must** be externally synchronized

To import a fence payload from a POSIX file descriptor, call:

```
// Provided by VK_KHR_external_fence_fd
VkResult vkImportFenceFdKHR(
    VkDevice                                    device,
    const VkImportFenceFdInfoKHR*               pImportFenceFdInfo);
```

- `device` is the logical device that created the fence.
- `pImportFenceFdInfo` is a pointer to a VkImportFenceFdInfoKHR structure specifying the fence and import parameters.

Importing a fence payload from a file descriptor transfers ownership of the file descriptor from the application to the Vulkan implementation. The application **must** not perform any operations on the file descriptor after a successful import.

Applications **can** import the same fence payload into multiple instances of Vulkan, into the same instance from which it was exported, and multiple times into a given Vulkan instance.

### Valid Usage

- VUID-vkImportFenceFdKHR-fence-01463
  `fence` **must** not be associated with any queue command that has not yet completed execution on that queue

### Valid Usage (Implicit)

- VUID-vkImportFenceFdKHR-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkImportFenceFdKHR-pImportFenceFdInfo-parameter
  `pImportFenceFdInfo` **must** be a valid pointer to a valid VkImportFenceFdInfoKHR structure

### Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_INVALID_EXTERNAL_HANDLE`

The `VkImportFenceFdInfoKHR` structure is defined as:

```
// Provided by VK_KHR_external_fence_fd
typedef struct VkImportFenceFdInfoKHR {
    VkStructureType                     sType;
    const void*                         pNext;
    VkFence                             fence;
    VkFenceImportFlags                  flags;
    VkExternalFenceHandleTypeFlagBits   handleType;
    int                                 fd;
} VkImportFenceFdInfoKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `fence` is the fence into which the payload will be imported.
- `flags` is a bitmask of VkFenceImportFlagBits specifying additional parameters for the fence payload import operation.
- `handleType` is a VkExternalFenceHandleTypeFlagBits value specifying the type of `fd`.
- `fd` is the external handle to import.

The handle types supported by `handleType` are:

*Table 7. Handle Types Supported by* `VkImportFenceFdInfoKHR`

| Handle Type | Transference | Permanence Supported |
|---|---|---|
| `VK_EXTERNAL_FENCE_HANDLE_TYPE_OPAQUE_FD_BIT` | Reference | Temporary,Permanent |
| `VK_EXTERNAL_FENCE_HANDLE_TYPE_SYNC_FD_BIT` | Copy | Temporary |

### Valid Usage

- VUID-VkImportFenceFdInfoKHR-handleType-01464
  `handleType` **must** be a value included in the Handle Types Supported by `VkImportFenceFdInfoKHR` table
- VUID-VkImportFenceFdInfoKHR-fd-01541
  `fd` **must** obey any requirements listed for `handleType` in external fence handle types compatibility
- VUID-VkImportFenceFdInfoKHR-handleType-07306
  If `handleType` refers to a handle type with copy payload transference semantics, `flags` **must** contain `VK_FENCE_IMPORT_TEMPORARY_BIT`

If `handleType` is `VK_EXTERNAL_FENCE_HANDLE_TYPE_SYNC_FD_BIT`, the special value `-1` for `fd` is treated like a valid sync file descriptor referring to an object that has already signaled. The import operation will succeed and the `VkFence` will have a temporarily imported payload as if a valid file descriptor had been provided.

| | |
|---|---|
| **NOTE** | This special behavior for importing an invalid sync file descriptor allows easier interoperability with other system APIs which use the convention that an invalid sync file descriptor represents work that has already completed and does not need to be waited for. It is consistent with the option for implementations to return a `-1` file descriptor when exporting a `VK_EXTERNAL_FENCE_HANDLE_TYPE_SYNC_FD_BIT` from a `VkFence` which is signaled. |

### Valid Usage (Implicit)

- VUID-VkImportFenceFdInfoKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_IMPORT_FENCE_FD_INFO_KHR`
- VUID-VkImportFenceFdInfoKHR-pNext-pNext
  `pNext` **must** be `NULL`
- VUID-VkImportFenceFdInfoKHR-fence-parameter
  `fence` **must** be a valid VkFence handle
- VUID-VkImportFenceFdInfoKHR-flags-parameter
  `flags` **must** be a valid combination of VkFenceImportFlagBits values
- VUID-VkImportFenceFdInfoKHR-handleType-parameter
  `handleType` **must** be a valid VkExternalFenceHandleTypeFlagBits value

### Host Synchronization

- Host access to `fence` **must** be externally synchronized

Bits which **can** be set in

- VkImportFenceWin32HandleInfoKHR::`flags`
- VkImportFenceFdInfoKHR::`flags`

specifying additional parameters of a fence import operation are:

```
// Provided by VK_VERSION_1_1
typedef enum VkFenceImportFlagBits {
    VK_FENCE_IMPORT_TEMPORARY_BIT = 0x00000001,
  // Provided by VK_KHR_external_fence
    VK_FENCE_IMPORT_TEMPORARY_BIT_KHR = VK_FENCE_IMPORT_TEMPORARY_BIT,
} VkFenceImportFlagBits;
```

or the equivalent

```
// Provided by VK_KHR_external_fence
typedef VkFenceImportFlagBits VkFenceImportFlagBitsKHR;
```

- VK_FENCE_IMPORT_TEMPORARY_BIT specifies that the fence payload will be imported only temporarily, as described in Importing Fence Payloads, regardless of the permanence of handleType.

```
// Provided by VK_VERSION_1_1
typedef VkFlags VkFenceImportFlags;
```

or the equivalent

```
// Provided by VK_KHR_external_fence
typedef VkFenceImportFlags VkFenceImportFlagsKHR;
```

VkFenceImportFlags is a bitmask type for setting a mask of zero or more VkFenceImportFlagBits.

## 7.4. Semaphores

Semaphores are a synchronization primitive that **can** be used to insert a dependency between queue operations or between a queue operation and the host. Binary semaphores have two states - signaled and unsignaled. Timeline semaphores have a strictly increasing 64-bit unsigned integer payload and are signaled with respect to a particular reference value. A semaphore **can** be signaled after execution of a queue operation is completed, and a queue operation **can** wait for a semaphore to become signaled before it begins execution. A timeline semaphore **can** additionally be signaled from the host with the vkSignalSemaphoreKHR command and waited on from the host with the vkWaitSemaphoresKHR command.

The internal data of a semaphore **may** include a reference to any resources and pending work associated with signal or unsignal operations performed on that semaphore object, collectively referred to as the semaphore's *payload*. Mechanisms to import and export that internal data to and from semaphores are provided below. These mechanisms indirectly enable applications to share semaphore state between two or more semaphores and other synchronization primitives across process and API boundaries.

Semaphores are represented by VkSemaphore handles:

```
// Provided by VK_VERSION_1_0
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkSemaphore)
```

To create a semaphore, call:

```
// Provided by VK_VERSION_1_0
VkResult vkCreateSemaphore(
    VkDevice                                    device,
    const VkSemaphoreCreateInfo*                pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkSemaphore*                                pSemaphore);
```

- `device` is the logical device that creates the semaphore.
- `pCreateInfo` is a pointer to a VkSemaphoreCreateInfo structure containing information about how the semaphore is to be created.
- `pAllocator` controls host memory allocation as described in the Memory Allocation chapter.
- `pSemaphore` is a pointer to a handle in which the resulting semaphore object is returned.

### Valid Usage (Implicit)

- VUID-vkCreateSemaphore-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkCreateSemaphore-pCreateInfo-parameter
  `pCreateInfo` **must** be a valid pointer to a valid VkSemaphoreCreateInfo structure
- VUID-vkCreateSemaphore-pAllocator-parameter
  If `pAllocator` is not `NULL`, `pAllocator` **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkCreateSemaphore-pSemaphore-parameter
  `pSemaphore` **must** be a valid pointer to a VkSemaphore handle

### Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`

The `VkSemaphoreCreateInfo` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkSemaphoreCreateInfo {
    VkStructureType           sType;
    const void*               pNext;
    VkSemaphoreCreateFlags    flags;
} VkSemaphoreCreateInfo;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is reserved for future use.

### Valid Usage

- VUID-VkSemaphoreCreateInfo-pNext-06789
  If the pNext chain includes a VkExportMetalObjectCreateInfoEXT structure, its exportObjectType member **must** be VK_EXPORT_METAL_OBJECT_TYPE_METAL_SHARED_EVENT_BIT_EXT

### Valid Usage (Implicit)

- VUID-VkSemaphoreCreateInfo-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_SEMAPHORE_CREATE_INFO
- VUID-VkSemaphoreCreateInfo-pNext-pNext
  Each pNext member of any structure (including this one) in the pNext chain **must** be either NULL or a pointer to a valid instance of VkExportMetalObjectCreateInfoEXT, VkExportSemaphoreCreateInfo, VkExportSemaphoreWin32HandleInfoKHR, VkImportMetalSharedEventInfoEXT, VkQueryLowLatencySupportNV, or VkSemaphoreTypeCreateInfo
- VUID-VkSemaphoreCreateInfo-sType-unique
  The sType value of each struct in the pNext chain **must** be unique, with the exception of structures of type VkExportMetalObjectCreateInfoEXT
- VUID-VkSemaphoreCreateInfo-flags-zerobitmask
  flags **must** be 0

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkSemaphoreCreateFlags;
```

VkSemaphoreCreateFlags is a bitmask type for setting a mask, but is currently reserved for future use.

The VkSemaphoreTypeCreateInfo structure is defined as:

```
typedef struct VkSemaphoreTypeCreateInfo {
    VkStructureType    sType;
    const void*        pNext;
    VkSemaphoreType    semaphoreType;
    uint64_t           initialValue;
} VkSemaphoreTypeCreateInfo;
```

or the equivalent

```
// Provided by VK_KHR_timeline_semaphore
typedef VkSemaphoreTypeCreateInfo VkSemaphoreTypeCreateInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- semaphoreType is a VkSemaphoreType value specifying the type of the semaphore.
- initialValue is the initial payload value if semaphoreType is VK_SEMAPHORE_TYPE_TIMELINE.

To create a semaphore of a specific type, add a VkSemaphoreTypeCreateInfo structure to the VkSemaphoreCreateInfo::pNext chain.

If no VkSemaphoreTypeCreateInfo structure is included in the pNext chain of VkSemaphoreCreateInfo, then the created semaphore will have a default VkSemaphoreType of VK_SEMAPHORE_TYPE_BINARY.

### Valid Usage

- VUID-VkSemaphoreTypeCreateInfo-timelineSemaphore-03252
  If the timelineSemaphore feature is not enabled, semaphoreType **must** not equal VK_SEMAPHORE_TYPE_TIMELINE
- VUID-VkSemaphoreTypeCreateInfo-semaphoreType-03279
  If semaphoreType is VK_SEMAPHORE_TYPE_BINARY, initialValue **must** be zero

### Valid Usage (Implicit)

- VUID-VkSemaphoreTypeCreateInfo-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_SEMAPHORE_TYPE_CREATE_INFO
- VUID-VkSemaphoreTypeCreateInfo-semaphoreType-parameter
  semaphoreType **must** be a valid VkSemaphoreType value

Possible values of VkSemaphoreTypeCreateInfo::semaphoreType, specifying the type of a semaphore, are:

```
typedef enum VkSemaphoreType {
    VK_SEMAPHORE_TYPE_BINARY = 0,
    VK_SEMAPHORE_TYPE_TIMELINE = 1,
  // Provided by VK_KHR_timeline_semaphore
    VK_SEMAPHORE_TYPE_BINARY_KHR = VK_SEMAPHORE_TYPE_BINARY,
  // Provided by VK_KHR_timeline_semaphore
    VK_SEMAPHORE_TYPE_TIMELINE_KHR = VK_SEMAPHORE_TYPE_TIMELINE,
} VkSemaphoreType;
```

or the equivalent

```
// Provided by VK_KHR_timeline_semaphore
typedef VkSemaphoreType VkSemaphoreTypeKHR;
```

- VK_SEMAPHORE_TYPE_BINARY specifies a *binary semaphore* type that has a boolean payload indicating whether the semaphore is currently signaled or unsignaled. When created, the semaphore is in the unsignaled state.
- VK_SEMAPHORE_TYPE_TIMELINE specifies a *timeline semaphore* type that has a strictly increasing 64-bit unsigned integer payload indicating whether the semaphore is signaled with respect to a particular reference value. When created, the semaphore payload has the value given by the initialValue field of VkSemaphoreTypeCreateInfo.

To create a semaphore whose payload **can** be exported to external handles, add a VkExportSemaphoreCreateInfo structure to the pNext chain of the VkSemaphoreCreateInfo structure. The VkExportSemaphoreCreateInfo structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkExportSemaphoreCreateInfo {
    VkStructureType                       sType;
    const void*                           pNext;
    VkExternalSemaphoreHandleTypeFlags    handleTypes;
} VkExportSemaphoreCreateInfo;
```

or the equivalent

```
// Provided by VK_KHR_external_semaphore
typedef VkExportSemaphoreCreateInfo VkExportSemaphoreCreateInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- handleTypes is a bitmask of VkExternalSemaphoreHandleTypeFlagBits specifying one or more semaphore handle types the application **can** export from the resulting semaphore. The application **can** request multiple handle types for the same semaphore.

### Valid Usage

- VUID-VkExportSemaphoreCreateInfo-handleTypes-01124
  The bits in handleTypes **must** be supported and compatible, as reported by VkExternalSemaphoreProperties

### Valid Usage (Implicit)

- VUID-VkExportSemaphoreCreateInfo-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_EXPORT_SEMAPHORE_CREATE_INFO
- VUID-VkExportSemaphoreCreateInfo-handleTypes-parameter
  handleTypes **must** be a valid combination of VkExternalSemaphoreHandleTypeFlagBits values

To specify additional attributes of NT handles exported from a semaphore, add a VkExportSemaphoreWin32HandleInfoKHR structure to the pNext chain of the VkSemaphoreCreateInfo structure. The VkExportSemaphoreWin32HandleInfoKHR structure is defined as:

```
// Provided by VK_KHR_external_semaphore_win32
typedef struct VkExportSemaphoreWin32HandleInfoKHR {
    VkStructureType                 sType;
    const void*                     pNext;
    const SECURITY_ATTRIBUTES*      pAttributes;
    DWORD                           dwAccess;
    LPCWSTR                         name;
} VkExportSemaphoreWin32HandleInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- pAttributes is a pointer to a Windows SECURITY_ATTRIBUTES structure specifying security attributes of the handle.
- dwAccess is a DWORD specifying access rights of the handle.
- name is a null-terminated UTF-16 string to associate with the underlying synchronization primitive referenced by NT handles exported from the created semaphore.

If VkExportSemaphoreCreateInfo is not included in the same pNext chain, this structure is ignored.

If VkExportSemaphoreCreateInfo is included in the pNext chain of VkSemaphoreCreateInfo with a Windows handleType, but either VkExportSemaphoreWin32HandleInfoKHR is not included in the pNext chain, or it is included but pAttributes is set to NULL, default security descriptor values will be used, and child processes created by the application will not inherit the handle, as described in the MSDN documentation for “Synchronization Object Security and Access Rights”[1]. Further, if the structure is not present, the access rights used depend on the handle type.

For handles of the following types:

VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_OPAQUE_WIN32_BIT

The implementation **must** ensure the access rights allow both signal and wait operations on the semaphore.

For handles of the following types:

VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_D3D12_FENCE_BIT

The access rights **must** be:

GENERIC_ALL

**1**

https://docs.microsoft.com/en-us/windows/win32/sync/synchronization-object-security-and-access-rights

### Valid Usage

- VUID-VkExportSemaphoreWin32HandleInfoKHR-handleTypes-01125
  If VkExportSemaphoreCreateInfo::handleTypes does not include VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_OPAQUE_WIN32_BIT or VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_D3D12_FENCE_BIT, VkExportSemaphoreWin32HandleInfoKHR **must** not be included in the pNext chain of VkSemaphoreCreateInfo

### Valid Usage (Implicit)

- VUID-VkExportSemaphoreWin32HandleInfoKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_EXPORT_SEMAPHORE_WIN32_HANDLE_INFO_KHR
- VUID-VkExportSemaphoreWin32HandleInfoKHR-pAttributes-parameter
  If pAttributes is not NULL, pAttributes **must** be a valid pointer to a valid SECURITY_ATTRIBUTES value

To export a Windows handle representing the payload of a semaphore, call:

```
// Provided by VK_KHR_external_semaphore_win32
VkResult vkGetSemaphoreWin32HandleKHR(
    VkDevice                                    device,
    const VkSemaphoreGetWin32HandleInfoKHR*     pGetWin32HandleInfo,
    HANDLE*                                     pHandle);
```

- device is the logical device that created the semaphore being exported.
- pGetWin32HandleInfo is a pointer to a VkSemaphoreGetWin32HandleInfoKHR structure containing parameters of the export operation.
- pHandle will return the Windows handle representing the semaphore state.

For handle types defined as NT handles, the handles returned by vkGetSemaphoreWin32HandleKHR are owned by the application. To avoid leaking resources, the application **must** release ownership of them using the CloseHandle system call when they are no longer needed.

Exporting a Windows handle from a semaphore **may** have side effects depending on the transference of the specified handle type, as described in Importing Semaphore Payloads.

### Valid Usage (Implicit)

- VUID-vkGetSemaphoreWin32HandleKHR-device-parameter
  device **must** be a valid VkDevice handle
- VUID-vkGetSemaphoreWin32HandleKHR-pGetWin32HandleInfo-parameter
  pGetWin32HandleInfo **must** be a valid pointer to a valid VkSemaphoreGetWin32HandleInfoKHR structure

- VUID-vkGetSemaphoreWin32HandleKHR-pHandle-parameter
  `pHandle` **must** be a valid pointer to a `HANDLE` value

## Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- `VK_ERROR_TOO_MANY_OBJECTS`
- `VK_ERROR_OUT_OF_HOST_MEMORY`

The `VkSemaphoreGetWin32HandleInfoKHR` structure is defined as:

```
// Provided by VK_KHR_external_semaphore_win32
typedef struct VkSemaphoreGetWin32HandleInfoKHR {
    VkStructureType                           sType;
    const void*                               pNext;
    VkSemaphore                               semaphore;
    VkExternalSemaphoreHandleTypeFlagBits     handleType;
} VkSemaphoreGetWin32HandleInfoKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `semaphore` is the semaphore from which state will be exported.
- `handleType` is a VkExternalSemaphoreHandleTypeFlagBits value specifying the type of handle requested.

The properties of the handle returned depend on the value of `handleType`. See VkExternalSemaphoreHandleTypeFlagBits for a description of the properties of the defined external semaphore handle types.

## Valid Usage

- VUID-VkSemaphoreGetWin32HandleInfoKHR-handleType-01126
  `handleType` **must** have been included in VkExportSemaphoreCreateInfo::`handleTypes` when the `semaphore`'s current payload was created
- VUID-VkSemaphoreGetWin32HandleInfoKHR-handleType-01127
  If `handleType` is defined as an NT handle, vkGetSemaphoreWin32HandleKHR **must** be called no more than once for each valid unique combination of `semaphore` and `handleType`
- VUID-VkSemaphoreGetWin32HandleInfoKHR-semaphore-01128
  `semaphore` **must** not currently have its payload replaced by an imported payload as described below in Importing Semaphore Payloads unless that imported payload's handle

type was included in VkExternalSemaphoreProperties::exportFromImportedHandleTypes for handleType

- VUID-VkSemaphoreGetWin32HandleInfoKHR-handleType-01129
  If handleType refers to a handle type with copy payload transference semantics, as defined below in Importing Semaphore Payloads, there **must** be no queue waiting on semaphore
- VUID-VkSemaphoreGetWin32HandleInfoKHR-handleType-01130
  If handleType refers to a handle type with copy payload transference semantics, semaphore **must** be signaled, or have an associated semaphore signal operation pending execution
- VUID-VkSemaphoreGetWin32HandleInfoKHR-handleType-01131
  handleType **must** be defined as an NT handle or a global share handle

### Valid Usage (Implicit)

- VUID-VkSemaphoreGetWin32HandleInfoKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_SEMAPHORE_GET_WIN32_HANDLE_INFO_KHR
- VUID-VkSemaphoreGetWin32HandleInfoKHR-pNext-pNext
  pNext **must** be NULL
- VUID-VkSemaphoreGetWin32HandleInfoKHR-semaphore-parameter
  semaphore **must** be a valid VkSemaphore handle
- VUID-VkSemaphoreGetWin32HandleInfoKHR-handleType-parameter
  handleType **must** be a valid VkExternalSemaphoreHandleTypeFlagBits value

The VkQueryLowLatencySupportNV structure is defined as:

```
// Provided by VK_NV_low_latency
typedef struct VkQueryLowLatencySupportNV {
    VkStructureType    sType;
    const void*        pNext;
    void*              pQueriedLowLatencyData;
} VkQueryLowLatencySupportNV;
```

This structure describes the following feature:

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- pQueriedLowLatencyData is used for NVIDIA Reflex Support.

### Valid Usage (Implicit)

- VUID-VkQueryLowLatencySupportNV-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_QUERY_LOW_LATENCY_SUPPORT_NV
- VUID-VkQueryLowLatencySupportNV-pQueriedLowLatencyData-parameter

pQueriedLowLatencyData **must** be a pointer value

To export a POSIX file descriptor representing the payload of a semaphore, call:

```
// Provided by VK_KHR_external_semaphore_fd
VkResult vkGetSemaphoreFdKHR(
    VkDevice                                    device,
    const VkSemaphoreGetFdInfoKHR*              pGetFdInfo,
    int*                                        pFd);
```

- device is the logical device that created the semaphore being exported.
- pGetFdInfo is a pointer to a VkSemaphoreGetFdInfoKHR structure containing parameters of the export operation.
- pFd will return the file descriptor representing the semaphore payload.

Each call to vkGetSemaphoreFdKHR **must** create a new file descriptor and transfer ownership of it to the application. To avoid leaking resources, the application **must** release ownership of the file descriptor when it is no longer needed.

| | |
|---|---|
| **NOTE** | Ownership can be released in many ways. For example, the application can call close() on the file descriptor, or transfer ownership back to Vulkan by using the file descriptor to import a semaphore payload. |

Where supported by the operating system, the implementation **must** set the file descriptor to be closed automatically when an execve system call is made.

Exporting a file descriptor from a semaphore **may** have side effects depending on the transference of the specified handle type, as described in Importing Semaphore State.

### Valid Usage (Implicit)

- VUID-vkGetSemaphoreFdKHR-device-parameter
  device **must** be a valid VkDevice handle
- VUID-vkGetSemaphoreFdKHR-pGetFdInfo-parameter
  pGetFdInfo **must** be a valid pointer to a valid VkSemaphoreGetFdInfoKHR structure
- VUID-vkGetSemaphoreFdKHR-pFd-parameter
  pFd **must** be a valid pointer to an int value

### Return Codes

**Success**

- VK_SUCCESS

**Failure**

- VK_ERROR_TOO_MANY_OBJECTS
- VK_ERROR_OUT_OF_HOST_MEMORY

The VkSemaphoreGetFdInfoKHR structure is defined as:

```
// Provided by VK_KHR_external_semaphore_fd
typedef struct VkSemaphoreGetFdInfoKHR {
    VkStructureType                           sType;
    const void*                               pNext;
    VkSemaphore                               semaphore;
    VkExternalSemaphoreHandleTypeFlagBits     handleType;
} VkSemaphoreGetFdInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- semaphore is the semaphore from which state will be exported.
- handleType is a VkExternalSemaphoreHandleTypeFlagBits value specifying the type of handle requested.

The properties of the file descriptor returned depend on the value of handleType. See VkExternalSemaphoreHandleTypeFlagBits for a description of the properties of the defined external semaphore handle types.

### Valid Usage

- VUID-VkSemaphoreGetFdInfoKHR-handleType-01132
  handleType **must** have been included in VkExportSemaphoreCreateInfo::handleTypes when semaphore's current payload was created
- VUID-VkSemaphoreGetFdInfoKHR-semaphore-01133
  semaphore **must** not currently have its payload replaced by an imported payload as described below in Importing Semaphore Payloads unless that imported payload's handle type was included in VkExternalSemaphoreProperties::exportFromImportedHandleTypes for handleType
- VUID-VkSemaphoreGetFdInfoKHR-handleType-01134
  If handleType refers to a handle type with copy payload transference semantics, as defined below in Importing Semaphore Payloads, there **must** be no queue waiting on semaphore
- VUID-VkSemaphoreGetFdInfoKHR-handleType-01135
  If handleType refers to a handle type with copy payload transference semantics, semaphore **must** be signaled, or have an associated semaphore signal operation pending execution
- VUID-VkSemaphoreGetFdInfoKHR-handleType-01136
  handleType **must** be defined as a POSIX file descriptor handle
- VUID-VkSemaphoreGetFdInfoKHR-handleType-03253

If handleType refers to a handle type with copy payload transference semantics, semaphore **must** have been created with a VkSemaphoreType of VK_SEMAPHORE_TYPE_BINARY

- VUID-VkSemaphoreGetFdInfoKHR-handleType-03254
  If handleType refers to a handle type with copy payload transference semantics, semaphore **must** have an associated semaphore signal operation that has been submitted for execution and any semaphore signal operations on which it depends **must** have also been submitted for execution

### Valid Usage (Implicit)

- VUID-VkSemaphoreGetFdInfoKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_SEMAPHORE_GET_FD_INFO_KHR
- VUID-VkSemaphoreGetFdInfoKHR-pNext-pNext
  pNext **must** be NULL
- VUID-VkSemaphoreGetFdInfoKHR-semaphore-parameter
  semaphore **must** be a valid VkSemaphore handle
- VUID-VkSemaphoreGetFdInfoKHR-handleType-parameter
  handleType **must** be a valid VkExternalSemaphoreHandleTypeFlagBits value

To export a Zircon event handle representing the payload of a semaphore, call:

```
// Provided by VK_FUCHSIA_external_semaphore
VkResult vkGetSemaphoreZirconHandleFUCHSIA(
    VkDevice                                    device,
    const VkSemaphoreGetZirconHandleInfoFUCHSIA* pGetZirconHandleInfo,
    zx_handle_t*                                pZirconHandle);
```

- device is the logical device that created the semaphore being exported.
- pGetZirconHandleInfo is a pointer to a VkSemaphoreGetZirconHandleInfoFUCHSIA structure containing parameters of the export operation.
- pZirconHandle will return the Zircon event handle representing the semaphore payload.

Each call to vkGetSemaphoreZirconHandleFUCHSIA **must** create a Zircon event handle and transfer ownership of it to the application. To avoid leaking resources, the application **must** release ownership of the Zircon event handle when it is no longer needed.

| | |
|---|---|
| **NOTE** | Ownership can be released in many ways. For example, the application can call zx_handle_close() on the file descriptor, or transfer ownership back to Vulkan by using the file descriptor to import a semaphore payload. |

Exporting a Zircon event handle from a semaphore **may** have side effects depending on the transference of the specified handle type, as described in Importing Semaphore State.

### Valid Usage (Implicit)

- VUID-vkGetSemaphoreZirconHandleFUCHSIA-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkGetSemaphoreZirconHandleFUCHSIA-pGetZirconHandleInfo-parameter
  `pGetZirconHandleInfo` **must** be a valid pointer to a valid VkSemaphoreGetZirconHandleInfoFUCHSIA structure
- VUID-vkGetSemaphoreZirconHandleFUCHSIA-pZirconHandle-parameter
  `pZirconHandle` **must** be a valid pointer to a `zx_handle_t` value

### Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- `VK_ERROR_TOO_MANY_OBJECTS`
- `VK_ERROR_OUT_OF_HOST_MEMORY`

The `VkSemaphoreGetZirconHandleInfoFUCHSIA` structure is defined as:

```
// Provided by VK_FUCHSIA_external_semaphore
typedef struct VkSemaphoreGetZirconHandleInfoFUCHSIA {
    VkStructureType                           sType;
    const void*                               pNext;
    VkSemaphore                               semaphore;
    VkExternalSemaphoreHandleTypeFlagBits     handleType;
} VkSemaphoreGetZirconHandleInfoFUCHSIA;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `semaphore` is the semaphore from which state will be exported.
- `handleType` is a VkExternalSemaphoreHandleTypeFlagBits value specifying the type of handle requested.

The properties of the Zircon event handle returned depend on the value of `handleType`. See VkExternalSemaphoreHandleTypeFlagBits for a description of the properties of the defined external semaphore handle types.

### Valid Usage

- VUID-VkSemaphoreGetZirconHandleInfoFUCHSIA-handleType-04758
  `handleType` **must** have been included in VkExportSemaphoreCreateInfo::`handleTypes` when

semaphore's current payload was created

- VUID-VkSemaphoreGetZirconHandleInfoFUCHSIA-semaphore-04759
  semaphore **must** not currently have its payload replaced by an imported payload as described below in Importing Semaphore Payloads unless that imported payload's handle type was included in VkExternalSemaphoreProperties::exportFromImportedHandleTypes for handleType
- VUID-VkSemaphoreGetZirconHandleInfoFUCHSIA-handleType-04760
  If handleType refers to a handle type with copy payload transference semantics, as defined below in Importing Semaphore Payloads, there **must** be no queue waiting on semaphore
- VUID-VkSemaphoreGetZirconHandleInfoFUCHSIA-handleType-04761
  If handleType refers to a handle type with copy payload transference semantics, semaphore **must** be signaled, or have an associated semaphore signal operation pending execution
- VUID-VkSemaphoreGetZirconHandleInfoFUCHSIA-handleType-04762
  handleType **must** be defined as a Zircon event handle
- VUID-VkSemaphoreGetZirconHandleInfoFUCHSIA-semaphore-04763
  semaphore **must** have been created with a VkSemaphoreType of VK_SEMAPHORE_TYPE_BINARY

### Valid Usage (Implicit)

- VUID-VkSemaphoreGetZirconHandleInfoFUCHSIA-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_SEMAPHORE_GET_ZIRCON_HANDLE_INFO_FUCHSIA
- VUID-VkSemaphoreGetZirconHandleInfoFUCHSIA-pNext-pNext
  pNext **must** be NULL
- VUID-VkSemaphoreGetZirconHandleInfoFUCHSIA-semaphore-parameter
  semaphore **must** be a valid VkSemaphore handle
- VUID-VkSemaphoreGetZirconHandleInfoFUCHSIA-handleType-parameter
  handleType **must** be a valid VkExternalSemaphoreHandleTypeFlagBits value

To destroy a semaphore, call:

```
// Provided by VK_VERSION_1_0
void vkDestroySemaphore(
    VkDevice                                    device,
    VkSemaphore                                 semaphore,
    const VkAllocationCallbacks*                pAllocator);
```

- device is the logical device that destroys the semaphore.
- semaphore is the handle of the semaphore to destroy.
- pAllocator controls host memory allocation as described in the Memory Allocation chapter.

### Valid Usage

- VUID-vkDestroySemaphore-semaphore-05149
  All submitted batches that refer to `semaphore` **must** have completed execution
- VUID-vkDestroySemaphore-semaphore-01138
  If `VkAllocationCallbacks` were provided when `semaphore` was created, a compatible set of callbacks **must** be provided here
- VUID-vkDestroySemaphore-semaphore-01139
  If no `VkAllocationCallbacks` were provided when `semaphore` was created, `pAllocator` **must** be `NULL`

### Valid Usage (Implicit)

- VUID-vkDestroySemaphore-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkDestroySemaphore-semaphore-parameter
  If `semaphore` is not VK_NULL_HANDLE, `semaphore` **must** be a valid VkSemaphore handle
- VUID-vkDestroySemaphore-pAllocator-parameter
  If `pAllocator` is not `NULL`, `pAllocator` **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkDestroySemaphore-semaphore-parent
  If `semaphore` is a valid handle, it **must** have been created, allocated, or retrieved from `device`

### Host Synchronization

- Host access to `semaphore` **must** be externally synchronized

## 7.4.1. Semaphore Signaling

When a batch is submitted to a queue via a queue submission, and it includes semaphores to be signaled, it defines a memory dependency on the batch, and defines *semaphore signal operations* which set the semaphores to the signaled state.

In case of semaphores created with a VkSemaphoreType of `VK_SEMAPHORE_TYPE_TIMELINE` the semaphore is considered signaled with respect to the counter value set to be signaled as specified in VkTimelineSemaphoreSubmitInfo or VkSemaphoreSignalInfo.

The first synchronization scope includes every command submitted in the same batch. In the case of vkQueueSubmit2KHR, the first synchronization scope is limited to the pipeline stage specified by VkSemaphoreSubmitInfo::`stageMask`. Semaphore signal operations that are defined by vkQueueSubmit or vkQueueSubmit2KHR additionally include all commands that occur earlier in submission order. Semaphore signal operations that are defined by vkQueueSubmit or

vkQueueSubmit2KHR or vkQueueBindSparse additionally include in the first synchronization scope any semaphore and fence signal operations that occur earlier in signal operation order.

The second synchronization scope includes only the semaphore signal operation.

The first access scope includes all memory access performed by the device.

The second access scope is empty.

### 7.4.2. Semaphore Waiting

When a batch is submitted to a queue via a queue submission, and it includes semaphores to be waited on, it defines a memory dependency between prior semaphore signal operations and the batch, and defines *semaphore wait operations*.

Such semaphore wait operations set the semaphores created with a VkSemaphoreType of `VK_SEMAPHORE_TYPE_BINARY` to the unsignaled state. In case of semaphores created with a VkSemaphoreType of `VK_SEMAPHORE_TYPE_TIMELINE` a prior semaphore signal operation defines a memory dependency with a semaphore wait operation if the value the semaphore is signaled with is greater than or equal to the value the semaphore is waited with, thus the semaphore will continue to be considered signaled with respect to the counter value waited on as specified in VkTimelineSemaphoreSubmitInfo.

The first synchronization scope includes all semaphore signal operations that operate on semaphores waited on in the same batch, and that happen-before the wait completes.

The second synchronization scope includes every command submitted in the same batch. In the case of vkQueueSubmit, the second synchronization scope is limited to operations on the pipeline stages determined by the destination stage mask specified by the corresponding element of `pWaitDstStageMask`. In the case of vkQueueSubmit2KHR, the second synchronization scope is limited to the pipeline stage specified by VkSemaphoreSubmitInfo::`stageMask`. Also, in the case of either vkQueueSubmit2KHR or vkQueueSubmit, the second synchronization scope additionally includes all commands that occur later in submission order.

The first access scope is empty.

The second access scope includes all memory access performed by the device.

The semaphore wait operation happens-after the first set of operations in the execution dependency, and happens-before the second set of operations in the execution dependency.

**NOTE** Unlike timeline semaphores, fences or events, waiting for a binary semaphore also unsignals that semaphore when the wait completes. Applications **must** ensure that between two such wait operations, the semaphore is signaled again, with execution dependencies used to ensure these occur in order. Binary semaphore waits and signals should thus occur in discrete 1:1 pairs.

**NOTE** A common scenario for using `pWaitDstStageMask` with values other than `VK_PIPELINE_STAGE_ALL_COMMANDS_BIT` is when synchronizing a window system presentation operation against subsequent command buffers which render the next

frame. In this case, a presentation image **must** not be overwritten until the presentation operation completes, but other pipeline stages **can** execute without waiting. A mask of `VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT` prevents subsequent color attachment writes from executing until the semaphore signals. Some implementations **may** be able to execute transfer operations and/or pre-rasterization work before the semaphore is signaled.

If an image layout transition needs to be performed on a presentable image before it is used in a framebuffer, that **can** be performed as the first operation submitted to the queue after acquiring the image, and **should** not prevent other work from overlapping with the presentation operation. For example, a `VkImageMemoryBarrier` could use:

- `srcStageMask` = `VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT`
- `srcAccessMask` = 0
- `dstStageMask` = `VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT`
- `dstAccessMask` = `VK_ACCESS_COLOR_ATTACHMENT_READ_BIT` | `VK_ACCESS_COLOR_ATTACHMENT_WRITE_BIT`.
- `oldLayout` = `VK_IMAGE_LAYOUT_PRESENT_SRC_KHR`
- `newLayout` = `VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL`

Alternatively, `oldLayout` **can** be `VK_IMAGE_LAYOUT_UNDEFINED`, if the image's contents need not be preserved.

This barrier accomplishes a dependency chain between previous presentation operations and subsequent color attachment output operations, with the layout transition performed in between, and does not introduce a dependency between previous work and any pre-rasterization shader stages. More precisely, the semaphore signals after the presentation operation completes, the semaphore wait stalls the `VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT` stage, and there is a dependency from that same stage to itself with the layout transition performed in between.

### 7.4.3. Semaphore State Requirements for Wait Operations

Before waiting on a semaphore, the application **must** ensure the semaphore is in a valid state for a wait operation. Specifically, when a semaphore wait operation is submitted to a queue:

- A binary semaphore **must** be signaled, or have an associated semaphore signal operation that is pending execution.
- Any semaphore signal operations on which the pending binary semaphore signal operation depends **must** also be completed or pending execution.
- There **must** be no other queue waiting on the same binary semaphore when the operation executes.

### 7.4.4. Host Operations on Semaphores

In addition to semaphore signal operations and semaphore wait operations submitted to device queues, timeline semaphores support the following host operations:

- Query the current counter value of the semaphore using the vkGetSemaphoreCounterValueKHR command.
- Wait for a set of semaphores to reach particular counter values using the vkWaitSemaphoresKHR command.
- Signal the semaphore with a particular counter value from the host using the vkSignalSemaphoreKHR command.

To query the current counter value of a semaphore created with a VkSemaphoreType of `VK_SEMAPHORE_TYPE_TIMELINE` from the host, call:

```
// Provided by VK_KHR_timeline_semaphore
VkResult vkGetSemaphoreCounterValueKHR(
    VkDevice                                    device,
    VkSemaphore                                 semaphore,
    uint64_t*                                   pValue);
```

- `device` is the logical device that owns the semaphore.
- `semaphore` is the handle of the semaphore to query.
- `pValue` is a pointer to a 64-bit integer value in which the current counter value of the semaphore is returned.

**NOTE** If a queue submission command is pending execution, then the value returned by this command **may** immediately be out of date.

#### Valid Usage

- VUID-vkGetSemaphoreCounterValue-semaphore-03255
  `semaphore` **must** have been created with a VkSemaphoreType of `VK_SEMAPHORE_TYPE_TIMELINE`

#### Valid Usage (Implicit)

- VUID-vkGetSemaphoreCounterValue-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkGetSemaphoreCounterValue-semaphore-parameter
  `semaphore` **must** be a valid VkSemaphore handle
- VUID-vkGetSemaphoreCounterValue-pValue-parameter
  `pValue` **must** be a valid pointer to a `uint64_t` value

- VUID-vkGetSemaphoreCounterValue-semaphore-parent
  `semaphore` **must** have been created, allocated, or retrieved from `device`

### Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_DEVICE_LOST`

To wait for a set of semaphores created with a VkSemaphoreType of `VK_SEMAPHORE_TYPE_TIMELINE` to reach particular counter values on the host, call:

```
// Provided by VK_KHR_timeline_semaphore
VkResult vkWaitSemaphoresKHR(
    VkDevice                                    device,
    const VkSemaphoreWaitInfo*                  pWaitInfo,
    uint64_t                                    timeout);
```

- `device` is the logical device that owns the semaphores.
- `pWaitInfo` is a pointer to a VkSemaphoreWaitInfo structure containing information about the wait condition.
- `timeout` is the timeout period in units of nanoseconds. `timeout` is adjusted to the closest value allowed by the implementation-dependent timeout accuracy, which **may** be substantially longer than one nanosecond, and **may** be longer than the requested period.

If the condition is satisfied when `vkWaitSemaphoresKHR` is called, then `vkWaitSemaphoresKHR` returns immediately. If the condition is not satisfied at the time `vkWaitSemaphoresKHR` is called, then `vkWaitSemaphoresKHR` will block and wait until the condition is satisfied or the `timeout` has expired, whichever is sooner.

If `timeout` is zero, then `vkWaitSemaphoresKHR` does not wait, but simply returns information about the current state of the semaphores. `VK_TIMEOUT` will be returned in this case if the condition is not satisfied, even though no actual wait was performed.

If the condition is satisfied before the `timeout` has expired, `vkWaitSemaphoresKHR` returns `VK_SUCCESS`. Otherwise, `vkWaitSemaphoresKHR` returns `VK_TIMEOUT` after the `timeout` has expired.

If device loss occurs (see Lost Device) before the timeout has expired, `vkWaitSemaphoresKHR` **must** return in finite time with either `VK_SUCCESS` or `VK_ERROR_DEVICE_LOST`.

### Valid Usage (Implicit)

- VUID-vkWaitSemaphores-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkWaitSemaphores-pWaitInfo-parameter
  `pWaitInfo` **must** be a valid pointer to a valid VkSemaphoreWaitInfo structure

### Return Codes

**Success**

- `VK_SUCCESS`
- `VK_TIMEOUT`

**Failure**

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_DEVICE_LOST`

The `VkSemaphoreWaitInfo` structure is defined as:

```
typedef struct VkSemaphoreWaitInfo {
    VkStructureType         sType;
    const void*             pNext;
    VkSemaphoreWaitFlags    flags;
    uint32_t                semaphoreCount;
    const VkSemaphore*      pSemaphores;
    const uint64_t*         pValues;
} VkSemaphoreWaitInfo;
```

or the equivalent

```
// Provided by VK_KHR_timeline_semaphore
typedef VkSemaphoreWaitInfo VkSemaphoreWaitInfoKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is a bitmask of VkSemaphoreWaitFlagBits specifying additional parameters for the semaphore wait operation.
- `semaphoreCount` is the number of semaphores to wait on.
- `pSemaphores` is a pointer to an array of `semaphoreCount` semaphore handles to wait on.
- `pValues` is a pointer to an array of `semaphoreCount` timeline semaphore values.

### Valid Usage

- VUID-VkSemaphoreWaitInfo-pSemaphores-03256
  All of the elements of `pSemaphores` **must** reference a semaphore that was created with a VkSemaphoreType of `VK_SEMAPHORE_TYPE_TIMELINE`

### Valid Usage (Implicit)

- VUID-VkSemaphoreWaitInfo-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_SEMAPHORE_WAIT_INFO`
- VUID-VkSemaphoreWaitInfo-pNext-pNext
  `pNext` **must** be `NULL`
- VUID-VkSemaphoreWaitInfo-flags-parameter
  `flags` **must** be a valid combination of VkSemaphoreWaitFlagBits values
- VUID-VkSemaphoreWaitInfo-pSemaphores-parameter
  `pSemaphores` **must** be a valid pointer to an array of `semaphoreCount` valid VkSemaphore handles
- VUID-VkSemaphoreWaitInfo-pValues-parameter
  `pValues` **must** be a valid pointer to an array of `semaphoreCount` `uint64_t` values
- VUID-VkSemaphoreWaitInfo-semaphoreCount-arraylength
  `semaphoreCount` **must** be greater than `0`

Bits which **can** be set in VkSemaphoreWaitInfo::`flags`, specifying additional parameters of a semaphore wait operation, are:

```
typedef enum VkSemaphoreWaitFlagBits {
    VK_SEMAPHORE_WAIT_ANY_BIT = 0x00000001,
  // Provided by VK_KHR_timeline_semaphore
    VK_SEMAPHORE_WAIT_ANY_BIT_KHR = VK_SEMAPHORE_WAIT_ANY_BIT,
} VkSemaphoreWaitFlagBits;
```

or the equivalent

```
// Provided by VK_KHR_timeline_semaphore
typedef VkSemaphoreWaitFlagBits VkSemaphoreWaitFlagBitsKHR;
```

- `VK_SEMAPHORE_WAIT_ANY_BIT` specifies that the semaphore wait condition is that at least one of the semaphores in `VkSemaphoreWaitInfo::pSemaphores` has reached the value specified by the corresponding element of `VkSemaphoreWaitInfo::pValues`. If `VK_SEMAPHORE_WAIT_ANY_BIT` is not set, the semaphore wait condition is that all of the semaphores in `VkSemaphoreWaitInfo::pSemaphores` have reached the value specified by the corresponding element of `VkSemaphoreWaitInfo::pValues`.

```
typedef VkFlags VkSemaphoreWaitFlags;
```

or the equivalent

```
// Provided by VK_KHR_timeline_semaphore
typedef VkSemaphoreWaitFlags VkSemaphoreWaitFlagsKHR;
```

`VkSemaphoreWaitFlags` is a bitmask type for setting a mask of zero or more VkSemaphoreWaitFlagBits.

To signal a semaphore created with a VkSemaphoreType of `VK_SEMAPHORE_TYPE_TIMELINE` with a particular counter value, on the host, call:

```
// Provided by VK_KHR_timeline_semaphore
VkResult vkSignalSemaphoreKHR(
    VkDevice                                    device,
    const VkSemaphoreSignalInfo*                pSignalInfo);
```

- `device` is the logical device that owns the semaphore.
- `pSignalInfo` is a pointer to a VkSemaphoreSignalInfo structure containing information about the signal operation.

When `vkSignalSemaphoreKHR` is executed on the host, it defines and immediately executes a *semaphore signal operation* which sets the timeline semaphore to the given value.

The first synchronization scope is defined by the host execution model, but includes execution of `vkSignalSemaphoreKHR` on the host and anything that happened-before it.

The second synchronization scope is empty.

### Valid Usage (Implicit)

- VUID-vkSignalSemaphore-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkSignalSemaphore-pSignalInfo-parameter
  `pSignalInfo` **must** be a valid pointer to a valid VkSemaphoreSignalInfo structure

### Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- `VK_ERROR_OUT_OF_HOST_MEMORY`

- VK_ERROR_OUT_OF_DEVICE_MEMORY

The VkSemaphoreSignalInfo structure is defined as:

```
typedef struct VkSemaphoreSignalInfo {
    VkStructureType    sType;
    const void*        pNext;
    VkSemaphore        semaphore;
    uint64_t           value;
} VkSemaphoreSignalInfo;
```

or the equivalent

```
// Provided by VK_KHR_timeline_semaphore
typedef VkSemaphoreSignalInfo VkSemaphoreSignalInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- semaphore is the handle of the semaphore to signal.
- value is the value to signal.

### Valid Usage

- VUID-VkSemaphoreSignalInfo-semaphore-03257
  semaphore **must** have been created with a VkSemaphoreType of VK_SEMAPHORE_TYPE_TIMELINE
- VUID-VkSemaphoreSignalInfo-value-03258
  value **must** have a value greater than the current value of the semaphore
- VUID-VkSemaphoreSignalInfo-value-03259
  value **must** be less than the value of any pending semaphore signal operations
- VUID-VkSemaphoreSignalInfo-value-03260
  value **must** have a value which does not differ from the current value of the semaphore or the value of any outstanding semaphore wait or signal operation on semaphore by more than maxTimelineSemaphoreValueDifference

### Valid Usage (Implicit)

- VUID-VkSemaphoreSignalInfo-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_SEMAPHORE_SIGNAL_INFO
- VUID-VkSemaphoreSignalInfo-pNext-pNext
  pNext **must** be NULL

- VUID-VkSemaphoreSignalInfo-semaphore-parameter
  `semaphore` **must** be a valid VkSemaphore handle

### 7.4.5. Importing Semaphore Payloads

Applications **can** import a semaphore payload into an existing semaphore using an external semaphore handle. The effects of the import operation will be either temporary or permanent, as specified by the application. If the import is temporary, the implementation **must** restore the semaphore to its prior permanent state after submitting the next semaphore wait operation. Performing a subsequent temporary import on a semaphore before performing a semaphore wait has no effect on this requirement; the next wait submitted on the semaphore **must** still restore its last permanent state. A permanent payload import behaves as if the target semaphore was destroyed, and a new semaphore was created with the same handle but the imported payload. Because importing a semaphore payload temporarily or permanently detaches the existing payload from a semaphore, similar usage restrictions to those applied to `vkDestroySemaphore` are applied to any command that imports a semaphore payload. Which of these import types is used is referred to as the import operation's *permanence*. Each handle type supports either one or both types of permanence.

The implementation **must** perform the import operation by either referencing or copying the payload referred to by the specified external semaphore handle, depending on the handle's type. The import method used is referred to as the handle type's *transference*. When using handle types with reference transference, importing a payload to a semaphore adds the semaphore to the set of all semaphores sharing that payload. This set includes the semaphore from which the payload was exported. Semaphore signaling and waiting operations performed on any semaphore in the set **must** behave as if the set were a single semaphore. Importing a payload using handle types with copy transference creates a duplicate copy of the payload at the time of import, but makes no further reference to it. Semaphore signaling and waiting operations performed on the target of copy imports **must** not affect any other semaphore or payload.

Export operations have the same transference as the specified handle type's import operations. Additionally, exporting a semaphore payload to a handle with copy transference has the same side effects on the source semaphore's payload as executing a semaphore wait operation. If the semaphore was using a temporarily imported payload, the semaphore's prior permanent payload will be restored.

**NOTE**

The permanence and transference of handle types can be found in:

- Handle Types Supported by `VkImportSemaphoreWin32HandleInfoKHR`
- Handle Types Supported by `VkImportSemaphoreFdInfoKHR`
- Handle Types Supported by `VkImportSemaphoreZirconHandleInfoFUCHSIA`

External synchronization allows implementations to modify an object's internal state, i.e. payload, without internal synchronization. However, for semaphores sharing a payload across processes, satisfying the external synchronization requirements of `VkSemaphore` parameters as if all semaphores in the set were the same object is sometimes infeasible. Satisfying the wait operation state requirements would similarly require impractical coordination or levels of trust between

processes. Therefore, these constraints only apply to a specific semaphore handle, not to its payload. For distinct semaphore objects which share a payload, if the semaphores are passed to separate queue submission commands concurrently, behavior will be as if the commands were called in an arbitrary sequential order. If the wait operation state requirements are violated for the shared payload by a queue submission command, or if a signal operation is queued for a shared payload that is already signaled or has a pending signal operation, effects **must** be limited to one or more of the following:

- Returning `VK_ERROR_INITIALIZATION_FAILED` from the command which resulted in the violation.
- Losing the logical device on which the violation occurred immediately or at a future time, resulting in a `VK_ERROR_DEVICE_LOST` error from subsequent commands, including the one causing the violation.
- Continuing execution of the violating command or operation as if the semaphore wait completed successfully after an implementation-dependent timeout. In this case, the state of the payload becomes undefined, and future operations on semaphores sharing the payload will be subject to these same rules. The semaphore **must** be destroyed or have its payload replaced by an import operation to again have a well-defined state.

**NOTE** These rules allow processes to synchronize access to shared memory without trusting each other. However, such processes must still be cautious not to use the shared semaphore for more than synchronizing access to the shared memory. For example, a process should not use a shared semaphore as part of an execution dependency chain that, when complete, leads to objects being destroyed, if it does not trust other processes sharing the semaphore payload.

When a semaphore is using an imported payload, its VkExportSemaphoreCreateInfo::`handleTypes` value is specified when creating the semaphore from which the payload was exported, rather than specified when creating the semaphore. Additionally, VkExternalSemaphoreProperties::`exportFromImportedHandleTypes` restricts which handle types **can** be exported from such a semaphore based on the specific handle type used to import the current payload. Passing a semaphore to vkAcquireNextImageKHR is equivalent to temporarily importing a semaphore payload to that semaphore.

**NOTE** Because the exportable handle types of an imported semaphore correspond to its current imported payload, and vkAcquireNextImageKHR behaves the same as a temporary import operation for which the source semaphore is opaque to the application, applications have no way of determining whether any external handle types **can** be exported from a semaphore in this state. Therefore, applications **must** not attempt to export external handles from semaphores using a temporarily imported payload from vkAcquireNextImageKHR.

When importing a semaphore payload, it is the responsibility of the application to ensure the external handles meet all valid usage requirements. However, implementations **must** perform sufficient validation of external handles to ensure that the operation results in a valid semaphore which will not cause program termination, device loss, queue stalls, or corruption of other resources when used as allowed according to its import parameters, and excepting those side effects allowed for violations of the valid semaphore state for wait operations rules. If the external

handle provided does not meet these requirements, the implementation **must** fail the semaphore payload import operation with the error code VK_ERROR_INVALID_EXTERNAL_HANDLE.

In addition, when importing a semaphore payload that is not compatible with the payload type corresponding to the VkSemaphoreType the semaphore was created with, the implementation **may** fail the semaphore payload import operation with the error code VK_ERROR_INVALID_EXTERNAL_HANDLE.

**NOTE** As the introduction of the external semaphore handle type VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_D3D12_FENCE_BIT predates that of timeline semaphores, support for importing semaphore payloads from external handles of that type into semaphores created (implicitly or explicitly) with a VkSemaphoreType of VK_SEMAPHORE_TYPE_BINARY is preserved for backwards compatibility. However, applications **should** prefer importing such handle types into semaphores created with a VkSemaphoreType of VK_SEMAPHORE_TYPE_TIMELINE.

To import a semaphore payload from a Windows handle, call:

```
// Provided by VK_KHR_external_semaphore_win32
VkResult vkImportSemaphoreWin32HandleKHR(
    VkDevice                                    device,
    const VkImportSemaphoreWin32HandleInfoKHR*  pImportSemaphoreWin32HandleInfo);
```

- device is the logical device that created the semaphore.
- pImportSemaphoreWin32HandleInfo is a pointer to a VkImportSemaphoreWin32HandleInfoKHR structure specifying the semaphore and import parameters.

Importing a semaphore payload from Windows handles does not transfer ownership of the handle to the Vulkan implementation. For handle types defined as NT handles, the application **must** release ownership using the CloseHandle system call when the handle is no longer needed.

Applications **can** import the same semaphore payload into multiple instances of Vulkan, into the same instance from which it was exported, and multiple times into a given Vulkan instance.

### Valid Usage (Implicit)

- VUID-vkImportSemaphoreWin32HandleKHR-device-parameter
  device **must** be a valid VkDevice handle
- VUID-vkImportSemaphoreWin32HandleKHR-pImportSemaphoreWin32HandleInfo-parameter
  pImportSemaphoreWin32HandleInfo **must** be a valid pointer to a valid VkImportSemaphoreWin32HandleInfoKHR structure

### Return Codes

**Success**

- VK_SUCCESS

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_INVALID_EXTERNAL_HANDLE

The VkImportSemaphoreWin32HandleInfoKHR structure is defined as:

```
// Provided by VK_KHR_external_semaphore_win32
typedef struct VkImportSemaphoreWin32HandleInfoKHR {
    VkStructureType                           sType;
    const void*                               pNext;
    VkSemaphore                               semaphore;
    VkSemaphoreImportFlags                    flags;
    VkExternalSemaphoreHandleTypeFlagBits     handleType;
    HANDLE                                    handle;
    LPCWSTR                                   name;
} VkImportSemaphoreWin32HandleInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- semaphore is the semaphore into which the payload will be imported.
- flags is a bitmask of VkSemaphoreImportFlagBits specifying additional parameters for the semaphore payload import operation.
- handleType is a VkExternalSemaphoreHandleTypeFlagBits value specifying the type of handle.
- handle is NULL or the external handle to import.
- name is NULL or a null-terminated UTF-16 string naming the underlying synchronization primitive to import.

The handle types supported by handleType are:

*Table 8. Handle Types Supported by* VkImportSemaphoreWin32HandleInfoKHR

| Handle Type | Transference | Permanence Supported |
|---|---|---|
| VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_OPAQUE_WIN32_BIT | Reference | Temporary,Permanent |
| VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_OPAQUE_WIN32_KMT_BIT | Reference | Temporary,Permanent |

| Handle Type | Transference | Permanence Supported |
| --- | --- | --- |
| VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_D3D12_FENCE_BIT | Reference | Temporary,Permanent |

## Valid Usage

- VUID-VkImportSemaphoreWin32HandleInfoKHR-handleType-01140
  handleType **must** be a value included in the Handle Types Supported by VkImportSemaphoreWin32HandleInfoKHR table
- VUID-VkImportSemaphoreWin32HandleInfoKHR-handleType-01466
  If handleType is not VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_OPAQUE_WIN32_BIT or VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_D3D12_FENCE_BIT, name **must** be NULL
- VUID-VkImportSemaphoreWin32HandleInfoKHR-handleType-01467
  If handle is NULL, name **must** name a valid synchronization primitive of the type specified by handleType
- VUID-VkImportSemaphoreWin32HandleInfoKHR-handleType-01468
  If name is NULL, handle **must** be a valid handle of the type specified by handleType
- VUID-VkImportSemaphoreWin32HandleInfoKHR-handle-01469
  If handle is not NULL, name **must** be NULL
- VUID-VkImportSemaphoreWin32HandleInfoKHR-handle-01542
  If handle is not NULL, it **must** obey any requirements listed for handleType in external semaphore handle types compatibility
- VUID-VkImportSemaphoreWin32HandleInfoKHR-name-01543
  If name is not NULL, it **must** obey any requirements listed for handleType in external semaphore handle types compatibility
- VUID-VkImportSemaphoreWin32HandleInfoKHR-handleType-03261
  If handleType is VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_OPAQUE_WIN32_BIT or VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_OPAQUE_WIN32_KMT_BIT, the VkSemaphoreCreateInfo::flags field **must** match that of the semaphore from which handle or name was exported
- VUID-VkImportSemaphoreWin32HandleInfoKHR-handleType-03262
  If handleType is VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_OPAQUE_WIN32_BIT or VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_OPAQUE_WIN32_KMT_BIT, the VkSemaphoreTypeCreateInfo::semaphoreType field **must** match that of the semaphore from which handle or name was exported
- VUID-VkImportSemaphoreWin32HandleInfoKHR-flags-03322
  If flags contains VK_SEMAPHORE_IMPORT_TEMPORARY_BIT, the VkSemaphoreTypeCreateInfo::semaphoreType field of the semaphore from which handle or name was exported **must** not be VK_SEMAPHORE_TYPE_TIMELINE

## Valid Usage (Implicit)

- VUID-VkImportSemaphoreWin32HandleInfoKHR-sType-sType

sType **must** be VK_STRUCTURE_TYPE_IMPORT_SEMAPHORE_WIN32_HANDLE_INFO_KHR

- VUID-VkImportSemaphoreWin32HandleInfoKHR-pNext-pNext
  pNext **must** be NULL
- VUID-VkImportSemaphoreWin32HandleInfoKHR-semaphore-parameter
  semaphore **must** be a valid VkSemaphore handle
- VUID-VkImportSemaphoreWin32HandleInfoKHR-flags-parameter
  flags **must** be a valid combination of VkSemaphoreImportFlagBits values

### Host Synchronization

- Host access to semaphore **must** be externally synchronized

To import a semaphore payload from a POSIX file descriptor, call:

```
// Provided by VK_KHR_external_semaphore_fd
VkResult vkImportSemaphoreFdKHR(
    VkDevice                                    device,
    const VkImportSemaphoreFdInfoKHR*           pImportSemaphoreFdInfo);
```

- device is the logical device that created the semaphore.
- pImportSemaphoreFdInfo is a pointer to a VkImportSemaphoreFdInfoKHR structure specifying the semaphore and import parameters.

Importing a semaphore payload from a file descriptor transfers ownership of the file descriptor from the application to the Vulkan implementation. The application **must** not perform any operations on the file descriptor after a successful import.

Applications **can** import the same semaphore payload into multiple instances of Vulkan, into the same instance from which it was exported, and multiple times into a given Vulkan instance.

### Valid Usage

- VUID-vkImportSemaphoreFdKHR-semaphore-01142
  semaphore **must** not be associated with any queue command that has not yet completed execution on that queue

### Valid Usage (Implicit)

- VUID-vkImportSemaphoreFdKHR-device-parameter
  device **must** be a valid VkDevice handle
- VUID-vkImportSemaphoreFdKHR-pImportSemaphoreFdInfo-parameter
  pImportSemaphoreFdInfo **must** be a valid pointer to a valid VkImportSemaphoreFdInfoKHR

structure

## Return Codes

**Success**

- VK_SUCCESS

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_INVALID_EXTERNAL_HANDLE

The VkImportSemaphoreFdInfoKHR structure is defined as:

```
// Provided by VK_KHR_external_semaphore_fd
typedef struct VkImportSemaphoreFdInfoKHR {
    VkStructureType                           sType;
    const void*                               pNext;
    VkSemaphore                               semaphore;
    VkSemaphoreImportFlags                    flags;
    VkExternalSemaphoreHandleTypeFlagBits     handleType;
    int                                       fd;
} VkImportSemaphoreFdInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- semaphore is the semaphore into which the payload will be imported.
- flags is a bitmask of VkSemaphoreImportFlagBits specifying additional parameters for the semaphore payload import operation.
- handleType is a VkExternalSemaphoreHandleTypeFlagBits value specifying the type of fd.
- fd is the external handle to import.

The handle types supported by handleType are:

*Table 9. Handle Types Supported by* VkImportSemaphoreFdInfoKHR

| Handle Type | Transference | Permanence Supported |
|---|---|---|
| VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_OPAQUE_FD_BIT | Reference | Temporary,Permanent |
| VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_SYNC_FD_BIT | Copy | Temporary |

## Valid Usage

- VUID-VkImportSemaphoreFdInfoKHR-handleType-01143

handleType **must** be a value included in the Handle Types Supported by VkImportSemaphoreFdInfoKHR table

- VUID-VkImportSemaphoreFdInfoKHR-fd-01544
  fd **must** obey any requirements listed for handleType in external semaphore handle types compatibility
- VUID-VkImportSemaphoreFdInfoKHR-handleType-03263
  If handleType is VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_OPAQUE_FD_BIT, the VkSemaphoreCreateInfo::flags field **must** match that of the semaphore from which fd was exported
- VUID-VkImportSemaphoreFdInfoKHR-handleType-07307
  If handleType refers to a handle type with copy payload transference semantics, flags **must** contain VK_SEMAPHORE_IMPORT_TEMPORARY_BIT
- VUID-VkImportSemaphoreFdInfoKHR-handleType-03264
  If handleType is VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_OPAQUE_FD_BIT, the VkSemaphoreTypeCreateInfo::semaphoreType field **must** match that of the semaphore from which fd was exported
- VUID-VkImportSemaphoreFdInfoKHR-flags-03323
  If flags contains VK_SEMAPHORE_IMPORT_TEMPORARY_BIT, the VkSemaphoreTypeCreateInfo::semaphoreType field of the semaphore from which fd was exported **must** not be VK_SEMAPHORE_TYPE_TIMELINE

If handleType is VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_SYNC_FD_BIT, the special value -1 for fd is treated like a valid sync file descriptor referring to an object that has already signaled. The import operation will succeed and the VkSemaphore will have a temporarily imported payload as if a valid file descriptor had been provided.

**NOTE** This special behavior for importing an invalid sync file descriptor allows easier interoperability with other system APIs which use the convention that an invalid sync file descriptor represents work that has already completed and does not need to be waited for. It is consistent with the option for implementations to return a -1 file descriptor when exporting a VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_SYNC_FD_BIT from a VkSemaphore which is signaled.

### Valid Usage (Implicit)

- VUID-VkImportSemaphoreFdInfoKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_IMPORT_SEMAPHORE_FD_INFO_KHR
- VUID-VkImportSemaphoreFdInfoKHR-pNext-pNext
  pNext **must** be NULL
- VUID-VkImportSemaphoreFdInfoKHR-semaphore-parameter
  semaphore **must** be a valid VkSemaphore handle
- VUID-VkImportSemaphoreFdInfoKHR-flags-parameter
  flags **must** be a valid combination of VkSemaphoreImportFlagBits values

- VUID-VkImportSemaphoreFdInfoKHR-handleType-parameter
  `handleType` **must** be a valid VkExternalSemaphoreHandleTypeFlagBits value

### Host Synchronization

- Host access to `semaphore` **must** be externally synchronized

To import a semaphore payload from a Zircon event handle, call:

```
// Provided by VK_FUCHSIA_external_semaphore
VkResult vkImportSemaphoreZirconHandleFUCHSIA(
    VkDevice                                    device,
    const VkImportSemaphoreZirconHandleInfoFUCHSIA* pImportSemaphoreZirconHandleInfo);
```

- `device` is the logical device that created the semaphore.
- `pImportSemaphoreZirconHandleInfo` is a pointer to a VkImportSemaphoreZirconHandleInfoFUCHSIA structure specifying the semaphore and import parameters.

Importing a semaphore payload from a Zircon event handle transfers ownership of the handle from the application to the Vulkan implementation. The application **must** not perform any operations on the handle after a successful import.

Applications **can** import the same semaphore payload into multiple instances of Vulkan, into the same instance from which it was exported, and multiple times into a given Vulkan instance.

### Valid Usage

- VUID-vkImportSemaphoreZirconHandleFUCHSIA-semaphore-04764
  `semaphore` **must** not be associated with any queue command that has not yet completed execution on that queue

### Valid Usage (Implicit)

- VUID-vkImportSemaphoreZirconHandleFUCHSIA-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkImportSemaphoreZirconHandleFUCHSIA-pImportSemaphoreZirconHandleInfo-parameter
  `pImportSemaphoreZirconHandleInfo` **must** be a valid pointer to a valid VkImportSemaphoreZirconHandleInfoFUCHSIA structure

## Return Codes

**Success**

- VK_SUCCESS

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_INVALID_EXTERNAL_HANDLE

The VkImportSemaphoreZirconHandleInfoFUCHSIA structure is defined as:

```
// Provided by VK_FUCHSIA_external_semaphore
typedef struct VkImportSemaphoreZirconHandleInfoFUCHSIA {
    VkStructureType                           sType;
    const void*                               pNext;
    VkSemaphore                               semaphore;
    VkSemaphoreImportFlags                    flags;
    VkExternalSemaphoreHandleTypeFlagBits     handleType;
    zx_handle_t                               zirconHandle;
} VkImportSemaphoreZirconHandleInfoFUCHSIA;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- semaphore is the semaphore into which the payload will be imported.
- flags is a bitmask of VkSemaphoreImportFlagBits specifying additional parameters for the semaphore payload import operation.
- handleType is a VkExternalSemaphoreHandleTypeFlagBits value specifying the type of zirconHandle.
- zirconHandle is the external handle to import.

The handle types supported by handleType are:

*Table 10. Handle Types Supported by VkImportSemaphoreZirconHandleInfoFUCHSIA*

| Handle Type | Transference | Permanence Supported |
|---|---|---|
| VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_ZIRCON_EVENT_BIT_FUCHSIA | Reference | Temporary,Permanent |

## Valid Usage

- VUID-VkImportSemaphoreZirconHandleInfoFUCHSIA-handleType-04765
  handleType **must** be a value included in the Handle Types Supported by VkImportSemaphoreZirconHandleInfoFUCHSIA table

- VUID-VkImportSemaphoreZirconHandleInfoFUCHSIA-zirconHandle-04766
  `zirconHandle` **must** obey any requirements listed for `handleType` in external semaphore handle types compatibility
- VUID-VkImportSemaphoreZirconHandleInfoFUCHSIA-zirconHandle-04767
  `zirconHandle` **must** have `ZX_RIGHTS_BASIC` and `ZX_RIGHTS_SIGNAL` rights
- VUID-VkImportSemaphoreZirconHandleInfoFUCHSIA-semaphoreType-04768
  The VkSemaphoreTypeCreateInfo::`semaphoreType` field **must** not be `VK_SEMAPHORE_TYPE_TIMELINE`

## Valid Usage (Implicit)

- VUID-VkImportSemaphoreZirconHandleInfoFUCHSIA-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_IMPORT_SEMAPHORE_ZIRCON_HANDLE_INFO_FUCHSIA`
- VUID-VkImportSemaphoreZirconHandleInfoFUCHSIA-pNext-pNext
  `pNext` **must** be `NULL`
- VUID-VkImportSemaphoreZirconHandleInfoFUCHSIA-semaphore-parameter
  `semaphore` **must** be a valid VkSemaphore handle
- VUID-VkImportSemaphoreZirconHandleInfoFUCHSIA-flags-parameter
  `flags` **must** be a valid combination of VkSemaphoreImportFlagBits values
- VUID-VkImportSemaphoreZirconHandleInfoFUCHSIA-handleType-parameter
  `handleType` **must** be a valid VkExternalSemaphoreHandleTypeFlagBits value

## Host Synchronization

- Host access to `semaphore` **must** be externally synchronized

Bits which **can** be set in

- VkImportSemaphoreWin32HandleInfoKHR::`flags`
- VkImportSemaphoreFdInfoKHR::`flags`
- VkImportSemaphoreZirconHandleInfoFUCHSIA::`flags`

specifying additional parameters of a semaphore import operation are:

```
// Provided by VK_VERSION_1_1
typedef enum VkSemaphoreImportFlagBits {
    VK_SEMAPHORE_IMPORT_TEMPORARY_BIT = 0x00000001,
  // Provided by VK_KHR_external_semaphore
    VK_SEMAPHORE_IMPORT_TEMPORARY_BIT_KHR = VK_SEMAPHORE_IMPORT_TEMPORARY_BIT,
} VkSemaphoreImportFlagBits;
```

or the equivalent

```
// Provided by VK_KHR_external_semaphore
typedef VkSemaphoreImportFlagBits VkSemaphoreImportFlagBitsKHR;
```

These bits have the following meanings:

- VK_SEMAPHORE_IMPORT_TEMPORARY_BIT specifies that the semaphore payload will be imported only temporarily, as described in Importing Semaphore Payloads, regardless of the permanence of handleType.

```
// Provided by VK_VERSION_1_1
typedef VkFlags VkSemaphoreImportFlags;
```

or the equivalent

```
// Provided by VK_KHR_external_semaphore
typedef VkSemaphoreImportFlags VkSemaphoreImportFlagsKHR;
```

VkSemaphoreImportFlags is a bitmask type for setting a mask of zero or more VkSemaphoreImportFlagBits.

## 7.5. Events

Events are a synchronization primitive that **can** be used to insert a fine-grained dependency between commands submitted to the same queue, or between the host and a queue. Events **must** not be used to insert a dependency between commands submitted to different queues. Events have two states - signaled and unsignaled. An application **can** signal or unsignal an event either on the host or on the device. A device **can** be made to wait for an event to become signaled before executing further operations. No command exists to wait for an event to become signaled on the host, but the current state of an event **can** be queried.

Events are represented by VkEvent handles:

```
// Provided by VK_VERSION_1_0
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkEvent)
```

To create an event, call:

```
// Provided by VK_VERSION_1_0
VkResult vkCreateEvent(
    VkDevice                                    device,
    const VkEventCreateInfo*                    pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkEvent*                                    pEvent);
```

- device is the logical device that creates the event.
- pCreateInfo is a pointer to a VkEventCreateInfo structure containing information about how the event is to be created.
- pAllocator controls host memory allocation as described in the Memory Allocation chapter.
- pEvent is a pointer to a handle in which the resulting event object is returned.

When created, the event object is in the unsignaled state.

### Valid Usage

- VUID-vkCreateEvent-device-09672
  device **must** support at least one queue family with one of the VK_QUEUE_VIDEO_ENCODE_BIT_KHR, VK_QUEUE_VIDEO_DECODE_BIT_KHR, VK_QUEUE_COMPUTE_BIT, or VK_QUEUE_GRAPHICS_BIT capabilities
- VUID-vkCreateEvent-events-04468
  If the VK_KHR_portability_subset extension is enabled, and VkPhysicalDevicePortabilitySubsetFeaturesKHR::events is VK_FALSE, then the implementation does not support events, and vkCreateEvent **must** not be used

### Valid Usage (Implicit)

- VUID-vkCreateEvent-device-parameter
  device **must** be a valid VkDevice handle
- VUID-vkCreateEvent-pCreateInfo-parameter
  pCreateInfo **must** be a valid pointer to a valid VkEventCreateInfo structure
- VUID-vkCreateEvent-pAllocator-parameter
  If pAllocator is not NULL, pAllocator **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkCreateEvent-pEvent-parameter
  pEvent **must** be a valid pointer to a VkEvent handle

### Return Codes

**Success**

- VK_SUCCESS

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY

The VkEventCreateInfo structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkEventCreateInfo {
    VkStructureType       sType;
    const void*           pNext;
    VkEventCreateFlags    flags;
} VkEventCreateInfo;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is a bitmask of VkEventCreateFlagBits defining additional creation parameters.

## Valid Usage

- VUID-VkEventCreateInfo-pNext-06790
  If the `pNext` chain includes a VkExportMetalObjectCreateInfoEXT structure, its `exportObjectType` member **must** be `VK_EXPORT_METAL_OBJECT_TYPE_METAL_SHARED_EVENT_BIT_EXT`

## Valid Usage (Implicit)

- VUID-VkEventCreateInfo-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_EVENT_CREATE_INFO`
- VUID-VkEventCreateInfo-pNext-pNext
  Each `pNext` member of any structure (including this one) in the `pNext` chain **must** be either `NULL` or a pointer to a valid instance of VkExportMetalObjectCreateInfoEXT or VkImportMetalSharedEventInfoEXT
- VUID-VkEventCreateInfo-sType-unique
  The `sType` value of each struct in the `pNext` chain **must** be unique, with the exception of structures of type VkExportMetalObjectCreateInfoEXT
- VUID-VkEventCreateInfo-flags-parameter
  `flags` **must** be a valid combination of VkEventCreateFlagBits values

```
// Provided by VK_VERSION_1_0
typedef enum VkEventCreateFlagBits {
    VK_EVENT_CREATE_DEVICE_ONLY_BIT = 0x00000001,
  // Provided by VK_KHR_synchronization2
    VK_EVENT_CREATE_DEVICE_ONLY_BIT_KHR = VK_EVENT_CREATE_DEVICE_ONLY_BIT,
} VkEventCreateFlagBits;
```

- `VK_EVENT_CREATE_DEVICE_ONLY_BIT` specifies that host event commands will not be used with this event.

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkEventCreateFlags;
```

VkEventCreateFlags is a bitmask type for setting a mask of VkEventCreateFlagBits.

To destroy an event, call:

```
// Provided by VK_VERSION_1_0
void vkDestroyEvent(
    VkDevice                                    device,
    VkEvent                                     event,
    const VkAllocationCallbacks*                pAllocator);
```

- device is the logical device that destroys the event.
- event is the handle of the event to destroy.
- pAllocator controls host memory allocation as described in the Memory Allocation chapter.

## Valid Usage

- VUID-vkDestroyEvent-event-01145
  All submitted commands that refer to event **must** have completed execution
- VUID-vkDestroyEvent-event-01146
  If VkAllocationCallbacks were provided when event was created, a compatible set of callbacks **must** be provided here
- VUID-vkDestroyEvent-event-01147
  If no VkAllocationCallbacks were provided when event was created, pAllocator **must** be NULL

## Valid Usage (Implicit)

- VUID-vkDestroyEvent-device-parameter
  device **must** be a valid VkDevice handle
- VUID-vkDestroyEvent-event-parameter
  If event is not VK_NULL_HANDLE, event **must** be a valid VkEvent handle
- VUID-vkDestroyEvent-pAllocator-parameter
  If pAllocator is not NULL, pAllocator **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkDestroyEvent-event-parent
  If event is a valid handle, it **must** have been created, allocated, or retrieved from device

### Host Synchronization

- Host access to `event` **must** be externally synchronized

To query the state of an event from the host, call:

```
// Provided by VK_VERSION_1_0
VkResult vkGetEventStatus(
    VkDevice                                    device,
    VkEvent                                     event);
```

- `device` is the logical device that owns the event.
- `event` is the handle of the event to query.

Upon success, `vkGetEventStatus` returns the state of the event object with the following return codes:

*Table 11. Event Object Status Codes*

| Status | Meaning |
|---|---|
| `VK_EVENT_SET` | The event specified by `event` is signaled. |
| `VK_EVENT_RESET` | The event specified by `event` is unsignaled. |

If a `vkCmdSetEvent` or `vkCmdResetEvent` command is in a command buffer that is in the pending state, then the value returned by this command **may** immediately be out of date.

The state of an event **can** be updated by the host. The state of the event is immediately changed, and subsequent calls to `vkGetEventStatus` will return the new state. If an event is already in the requested state, then updating it to the same state has no effect.

### Valid Usage

- VUID-vkGetEventStatus-event-03940
  `event` **must** not have been created with `VK_EVENT_CREATE_DEVICE_ONLY_BIT`

### Valid Usage (Implicit)

- VUID-vkGetEventStatus-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkGetEventStatus-event-parameter
  `event` **must** be a valid VkEvent handle
- VUID-vkGetEventStatus-event-parent
  `event` **must** have been created, allocated, or retrieved from `device`

### Return Codes

**Success**

- VK_EVENT_SET
- VK_EVENT_RESET

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY
- VK_ERROR_DEVICE_LOST

To set the state of an event to signaled from the host, call:

```
// Provided by VK_VERSION_1_0
VkResult vkSetEvent(
    VkDevice                                    device,
    VkEvent                                     event);
```

- device is the logical device that owns the event.
- event is the event to set.

When vkSetEvent is executed on the host, it defines an *event signal operation* which sets the event to the signaled state.

If event is already in the signaled state when vkSetEvent is executed, then vkSetEvent has no effect, and no event signal operation occurs.

| | |
|---|---|
| **NOTE** | If a command buffer is waiting for an event to be signaled from the host, the application must signal the event before submitting the command buffer, as described in the queue forward progress section. |

### Valid Usage

- VUID-vkSetEvent-event-03941
  event **must** not have been created with VK_EVENT_CREATE_DEVICE_ONLY_BIT
- VUID-vkSetEvent-event-09543
  event **must** not be waited on by a command buffer in the pending state

### Valid Usage (Implicit)

- VUID-vkSetEvent-device-parameter
  device **must** be a valid VkDevice handle
- VUID-vkSetEvent-event-parameter

event **must** be a valid VkEvent handle

- VUID-vkSetEvent-event-parent
  event **must** have been created, allocated, or retrieved from device

## Host Synchronization

- Host access to event **must** be externally synchronized

## Return Codes

**Success**

- VK_SUCCESS

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY

To set the state of an event to unsignaled from the host, call:

```
// Provided by VK_VERSION_1_0
VkResult vkResetEvent(
    VkDevice                                    device,
    VkEvent                                     event);
```

- device is the logical device that owns the event.
- event is the event to reset.

When vkResetEvent is executed on the host, it defines an *event unsignal operation* which resets the event to the unsignaled state.

If event is already in the unsignaled state when vkResetEvent is executed, then vkResetEvent has no effect, and no event unsignal operation occurs.

## Valid Usage

- VUID-vkResetEvent-event-03821
  There **must** be an execution dependency between vkResetEvent and the execution of any vkCmdWaitEvents that includes event in its pEvents parameter
- VUID-vkResetEvent-event-03822
  There **must** be an execution dependency between vkResetEvent and the execution of any vkCmdWaitEvents2KHR that includes event in its pEvents parameter
- VUID-vkResetEvent-event-03823
  event **must** not have been created with VK_EVENT_CREATE_DEVICE_ONLY_BIT

### Valid Usage (Implicit)

- VUID-vkResetEvent-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkResetEvent-event-parameter
  `event` **must** be a valid VkEvent handle
- VUID-vkResetEvent-event-parent
  `event` **must** have been created, allocated, or retrieved from `device`

### Host Synchronization

- Host access to `event` **must** be externally synchronized

### Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- `VK_ERROR_OUT_OF_DEVICE_MEMORY`

The state of an event **can** also be updated on the device by commands inserted in command buffers.

To signal an event from a device, call:

```
// Provided by VK_KHR_synchronization2
void vkCmdSetEvent2KHR(
    VkCommandBuffer                             commandBuffer,
    VkEvent                                     event,
    const VkDependencyInfo*                     pDependencyInfo);
```

- `commandBuffer` is the command buffer into which the command is recorded.
- `event` is the event that will be signaled.
- `pDependencyInfo` is a pointer to a VkDependencyInfo structure defining the first scopes of this operation.

When vkCmdSetEvent2KHR is submitted to a queue, it defines the first half of memory dependencies defined by `pDependencyInfo`, as well as an event signal operation which sets the event to the signaled state. A memory dependency is defined between the event signal operation and commands that occur earlier in submission order.

The first synchronization scope and access scope are defined by the union of all the memory

dependencies defined by pDependencyInfo, and are applied to all operations that occur earlier in submission order. Queue family ownership transfers and image layout transitions defined by pDependencyInfo are also included in the first scopes.

The second synchronization scope includes only the event signal operation, and any queue family ownership transfers and image layout transitions defined by pDependencyInfo.

The second access scope includes only queue family ownership transfers and image layout transitions.

Future vkCmdWaitEvents2KHR commands rely on all values of each element in pDependencyInfo matching exactly with those used to signal the corresponding event. vkCmdWaitEvents **must** not be used to wait on the result of a signal operation defined by vkCmdSetEvent2KHR.

| | |
|---|---|
| **NOTE** | The extra information provided by vkCmdSetEvent2KHR compared to vkCmdSetEvent allows implementations to more efficiently schedule the operations required to satisfy the requested dependencies. With vkCmdSetEvent, the full dependency information is not known until vkCmdWaitEvents is recorded, forcing implementations to insert the required operations at that point and not before. |

If event is already in the signaled state when vkCmdSetEvent2KHR is executed on the device, then vkCmdSetEvent2KHR has no effect, no event signal operation occurs, and no dependency is generated.

## Valid Usage

- VUID-vkCmdSetEvent2-synchronization2-03824
  The synchronization2 feature **must** be enabled
- VUID-vkCmdSetEvent2-dependencyFlags-03825
  The dependencyFlags member of pDependencyInfo **must** be 0
- VUID-vkCmdSetEvent2-srcStageMask-09391
  The srcStageMask member of any element of the pMemoryBarriers, pBufferMemoryBarriers, or pImageMemoryBarriers members of pDependencyInfo **must** not include VK_PIPELINE_STAGE_2_HOST_BIT
- VUID-vkCmdSetEvent2-dstStageMask-09392
  The dstStageMask member of any element of the pMemoryBarriers, pBufferMemoryBarriers, or pImageMemoryBarriers members of pDependencyInfo **must** not include VK_PIPELINE_STAGE_2_HOST_BIT
- VUID-vkCmdSetEvent2-commandBuffer-03826
  The current device mask of commandBuffer **must** include exactly one physical device
- VUID-vkCmdSetEvent2-srcStageMask-03827
  The srcStageMask member of any element of the pMemoryBarriers, pBufferMemoryBarriers, or pImageMemoryBarriers members of pDependencyInfo **must** only include pipeline stages valid for the queue family that was used to create the command pool that commandBuffer was allocated from
- VUID-vkCmdSetEvent2-dstStageMask-03828

The `dstStageMask` member of any element of the `pMemoryBarriers`, `pBufferMemoryBarriers`, or `pImageMemoryBarriers` members of `pDependencyInfo` **must** only include pipeline stages valid for the queue family that was used to create the command pool that `commandBuffer` was allocated from

## Valid Usage (Implicit)

- VUID-vkCmdSetEvent2-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdSetEvent2-event-parameter
  `event` **must** be a valid VkEvent handle
- VUID-vkCmdSetEvent2-pDependencyInfo-parameter
  `pDependencyInfo` **must** be a valid pointer to a valid VkDependencyInfo structure
- VUID-vkCmdSetEvent2-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdSetEvent2-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics, compute, decode, or encode operations
- VUID-vkCmdSetEvent2-renderpass
  This command **must** only be called outside of a render pass instance
- VUID-vkCmdSetEvent2-commonparent
  Both of `commandBuffer`, and `event` **must** have been created, allocated, or retrieved from the same VkDevice

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Outside | Both | Graphics<br>Compute<br>Decode<br>Encode | Synchronization |

The `VkDependencyInfo` structure is defined as:

```
typedef struct VkDependencyInfo {
    VkStructureType                 sType;
    const void*                     pNext;
    VkDependencyFlags               dependencyFlags;
    uint32_t                        memoryBarrierCount;
    const VkMemoryBarrier2*         pMemoryBarriers;
    uint32_t                        bufferMemoryBarrierCount;
    const VkBufferMemoryBarrier2*   pBufferMemoryBarriers;
    uint32_t                        imageMemoryBarrierCount;
    const VkImageMemoryBarrier2*    pImageMemoryBarriers;
} VkDependencyInfo;
```

or the equivalent

```
// Provided by VK_KHR_synchronization2
typedef VkDependencyInfo VkDependencyInfoKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `dependencyFlags` is a bitmask of VkDependencyFlagBits specifying how execution and memory dependencies are formed.
- `memoryBarrierCount` is the length of the `pMemoryBarriers` array.
- `pMemoryBarriers` is a pointer to an array of VkMemoryBarrier2 structures defining memory dependencies between any memory accesses.
- `bufferMemoryBarrierCount` is the length of the `pBufferMemoryBarriers` array.
- `pBufferMemoryBarriers` is a pointer to an array of VkBufferMemoryBarrier2 structures defining memory dependencies between buffer ranges.
- `imageMemoryBarrierCount` is the length of the `pImageMemoryBarriers` array.
- `pImageMemoryBarriers` is a pointer to an array of VkImageMemoryBarrier2 structures defining memory dependencies between image subresources.

This structure defines a set of memory dependencies, as well as queue family ownership transfer operations and image layout transitions.

Each member of `pMemoryBarriers`, `pBufferMemoryBarriers`, and `pImageMemoryBarriers` defines a separate memory dependency.

### Valid Usage (Implicit)

- VUID-VkDependencyInfo-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_DEPENDENCY_INFO`
- VUID-VkDependencyInfo-pNext-pNext
  `pNext` **must** be `NULL`

- VUID-VkDependencyInfo-dependencyFlags-parameter
  `dependencyFlags` **must** be a valid combination of VkDependencyFlagBits values
- VUID-VkDependencyInfo-pMemoryBarriers-parameter
  If `memoryBarrierCount` is not `0`, `pMemoryBarriers` **must** be a valid pointer to an array of `memoryBarrierCount` valid VkMemoryBarrier2 structures
- VUID-VkDependencyInfo-pBufferMemoryBarriers-parameter
  If `bufferMemoryBarrierCount` is not `0`, `pBufferMemoryBarriers` **must** be a valid pointer to an array of `bufferMemoryBarrierCount` valid VkBufferMemoryBarrier2 structures
- VUID-VkDependencyInfo-pImageMemoryBarriers-parameter
  If `imageMemoryBarrierCount` is not `0`, `pImageMemoryBarriers` **must** be a valid pointer to an array of `imageMemoryBarrierCount` valid VkImageMemoryBarrier2 structures

To set the state of an event to signaled from a device, call:

```
// Provided by VK_VERSION_1_0
void vkCmdSetEvent(
    VkCommandBuffer                             commandBuffer,
    VkEvent                                     event,
    VkPipelineStageFlags                        stageMask);
```

- `commandBuffer` is the command buffer into which the command is recorded.
- `event` is the event that will be signaled.
- `stageMask` specifies the source stage mask used to determine the first synchronization scope.

`vkCmdSetEvent` behaves identically to vkCmdSetEvent2KHR, except that it does not define an access scope, and **must** only be used with vkCmdWaitEvents, not vkCmdWaitEvents2KHR.

### Valid Usage

- VUID-vkCmdSetEvent-stageMask-04090
  If the `geometryShader` feature is not enabled, `stageMask` **must** not contain `VK_PIPELINE_STAGE_GEOMETRY_SHADER_BIT`
- VUID-vkCmdSetEvent-stageMask-04091
  If the `tessellationShader` feature is not enabled, `stageMask` **must** not contain `VK_PIPELINE_STAGE_TESSELLATION_CONTROL_SHADER_BIT` or `VK_PIPELINE_STAGE_TESSELLATION_EVALUATION_SHADER_BIT`
- VUID-vkCmdSetEvent-stageMask-04092
  If the `conditionalRendering` feature is not enabled, `stageMask` **must** not contain `VK_PIPELINE_STAGE_CONDITIONAL_RENDERING_BIT_EXT`
- VUID-vkCmdSetEvent-stageMask-04093
  If the `fragmentDensityMap` feature is not enabled, `stageMask` **must** not contain `VK_PIPELINE_STAGE_FRAGMENT_DENSITY_PROCESS_BIT_EXT`
- VUID-vkCmdSetEvent-stageMask-04094

If the transformFeedback feature is not enabled, stageMask **must** not contain VK_PIPELINE_STAGE_TRANSFORM_FEEDBACK_BIT_EXT

- VUID-vkCmdSetEvent-stageMask-04095
  If the meshShader feature is not enabled, stageMask **must** not contain VK_PIPELINE_STAGE_MESH_SHADER_BIT_EXT
- VUID-vkCmdSetEvent-stageMask-04096
  If the taskShader feature is not enabled, stageMask **must** not contain VK_PIPELINE_STAGE_TASK_SHADER_BIT_EXT
- VUID-vkCmdSetEvent-stageMask-07318
  If neither the shadingRateImage or attachmentFragmentShadingRate are enabled, stageMask **must** not contain VK_PIPELINE_STAGE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR
- VUID-vkCmdSetEvent-stageMask-03937
  If the synchronization2 feature is not enabled, stageMask **must** not be 0
- VUID-vkCmdSetEvent-stageMask-07949
  If neither the VK_NV_ray_tracing extension or rayTracingPipeline feature are enabled, stageMask **must** not contain VK_PIPELINE_STAGE_RAY_TRACING_SHADER_BIT_KHR
- VUID-vkCmdSetEvent-stageMask-06457
  Any pipeline stage included in stageMask **must** be supported by the capabilities of the queue family specified by the queueFamilyIndex member of the VkCommandPoolCreateInfo structure that was used to create the VkCommandPool that commandBuffer was allocated from, as specified in the table of supported pipeline stages
- VUID-vkCmdSetEvent-stageMask-01149
  stageMask **must** not include VK_PIPELINE_STAGE_HOST_BIT
- VUID-vkCmdSetEvent-commandBuffer-01152
  The current device mask of commandBuffer **must** include exactly one physical device

## Valid Usage (Implicit)

- VUID-vkCmdSetEvent-commandBuffer-parameter
  commandBuffer **must** be a valid VkCommandBuffer handle
- VUID-vkCmdSetEvent-event-parameter
  event **must** be a valid VkEvent handle
- VUID-vkCmdSetEvent-stageMask-parameter
  stageMask **must** be a valid combination of VkPipelineStageFlagBits values
- VUID-vkCmdSetEvent-commandBuffer-recording
  commandBuffer **must** be in the recording state
- VUID-vkCmdSetEvent-commandBuffer-cmdpool
  The VkCommandPool that commandBuffer was allocated from **must** support graphics, compute, decode, or encode operations
- VUID-vkCmdSetEvent-renderpass
  This command **must** only be called outside of a render pass instance

- VUID-vkCmdSetEvent-commonparent
  Both of `commandBuffer`, and `event` **must** have been created, allocated, or retrieved from the same VkDevice

### Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

### Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Outside | Both | Graphics<br>Compute<br>Decode<br>Encode | Synchronization |

To unsignal the event from a device, call:

```
// Provided by VK_KHR_synchronization2
void vkCmdResetEvent2KHR(
    VkCommandBuffer                             commandBuffer,
    VkEvent                                     event,
    VkPipelineStageFlags2                       stageMask);
```

- `commandBuffer` is the command buffer into which the command is recorded.
- `event` is the event that will be unsignaled.
- `stageMask` is a VkPipelineStageFlags2 mask of pipeline stages used to determine the first synchronization scope.

When vkCmdResetEvent2KHR is submitted to a queue, it defines an execution dependency on commands that were submitted before it, and defines an event unsignal operation which resets the event to the unsignaled state.

The first synchronization scope includes all commands that occur earlier in submission order. The synchronization scope is limited to operations by `stageMask` or stages that are logically earlier than `stageMask`.

The second synchronization scope includes only the event unsignal operation.

If `event` is already in the unsignaled state when vkCmdResetEvent2KHR is executed on the device,

then this command has no effect, no event unsignal operation occurs, and no execution dependency is generated.

### Valid Usage

- VUID-vkCmdResetEvent2-stageMask-03929
  If the `geometryShader` feature is not enabled, `stageMask` **must** not contain `VK_PIPELINE_STAGE_2_GEOMETRY_SHADER_BIT`
- VUID-vkCmdResetEvent2-stageMask-03930
  If the `tessellationShader` feature is not enabled, `stageMask` **must** not contain `VK_PIPELINE_STAGE_2_TESSELLATION_CONTROL_SHADER_BIT` or `VK_PIPELINE_STAGE_2_TESSELLATION_EVALUATION_SHADER_BIT`
- VUID-vkCmdResetEvent2-stageMask-03931
  If the `conditionalRendering` feature is not enabled, `stageMask` **must** not contain `VK_PIPELINE_STAGE_2_CONDITIONAL_RENDERING_BIT_EXT`
- VUID-vkCmdResetEvent2-stageMask-03932
  If the `fragmentDensityMap` feature is not enabled, `stageMask` **must** not contain `VK_PIPELINE_STAGE_2_FRAGMENT_DENSITY_PROCESS_BIT_EXT`
- VUID-vkCmdResetEvent2-stageMask-03933
  If the `transformFeedback` feature is not enabled, `stageMask` **must** not contain `VK_PIPELINE_STAGE_2_TRANSFORM_FEEDBACK_BIT_EXT`
- VUID-vkCmdResetEvent2-stageMask-03934
  If the `meshShader` feature is not enabled, `stageMask` **must** not contain `VK_PIPELINE_STAGE_2_MESH_SHADER_BIT_EXT`
- VUID-vkCmdResetEvent2-stageMask-03935
  If the `taskShader` feature is not enabled, `stageMask` **must** not contain `VK_PIPELINE_STAGE_2_TASK_SHADER_BIT_EXT`
- VUID-vkCmdResetEvent2-stageMask-07316
  If neither the `shadingRateImage` or `attachmentFragmentShadingRate` are enabled, `stageMask` **must** not contain `VK_PIPELINE_STAGE_2_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR`
- VUID-vkCmdResetEvent2-stageMask-04957
  If the `subpassShading` feature is not enabled, `stageMask` **must** not contain `VK_PIPELINE_STAGE_2_SUBPASS_SHADER_BIT_HUAWEI`
- VUID-vkCmdResetEvent2-stageMask-04995
  If the `invocationMask` feature is not enabled, `stageMask` **must** not contain `VK_PIPELINE_STAGE_2_INVOCATION_MASK_BIT_HUAWEI`
- VUID-vkCmdResetEvent2-stageMask-07946
  If neither the VK_NV_ray_tracing extension or `rayTracingPipeline` feature are enabled, `stageMask` **must** not contain `VK_PIPELINE_STAGE_2_RAY_TRACING_SHADER_BIT_KHR`
- VUID-vkCmdResetEvent2-synchronization2-03829
  The `synchronization2` feature **must** be enabled
- VUID-vkCmdResetEvent2-stageMask-03830
  `stageMask` **must** not include `VK_PIPELINE_STAGE_2_HOST_BIT`

- VUID-vkCmdResetEvent2-event-03831
  There **must** be an execution dependency between `vkCmdResetEvent2KHR` and the execution of any vkCmdWaitEvents that includes `event` in its `pEvents` parameter
- VUID-vkCmdResetEvent2-event-03832
  There **must** be an execution dependency between `vkCmdResetEvent2KHR` and the execution of any vkCmdWaitEvents2KHR that includes `event` in its `pEvents` parameter
- VUID-vkCmdResetEvent2-commandBuffer-03833
  `commandBuffer`'s current device mask **must** include exactly one physical device

## Valid Usage (Implicit)

- VUID-vkCmdResetEvent2-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdResetEvent2-event-parameter
  `event` **must** be a valid VkEvent handle
- VUID-vkCmdResetEvent2-stageMask-parameter
  `stageMask` **must** be a valid combination of VkPipelineStageFlagBits2 values
- VUID-vkCmdResetEvent2-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdResetEvent2-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics, compute, decode, or encode operations
- VUID-vkCmdResetEvent2-renderpass
  This command **must** only be called outside of a render pass instance
- VUID-vkCmdResetEvent2-commonparent
  Both of `commandBuffer`, and `event` **must** have been created, allocated, or retrieved from the same VkDevice

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

### Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Outside | Both | Graphics<br>Compute<br>Decode<br>Encode | Synchronization |

To set the state of an event to unsignaled from a device, call:

```
// Provided by VK_VERSION_1_0
void vkCmdResetEvent(
    VkCommandBuffer                             commandBuffer,
    VkEvent                                     event,
    VkPipelineStageFlags                        stageMask);
```

- `commandBuffer` is the command buffer into which the command is recorded.
- `event` is the event that will be unsignaled.
- `stageMask` is a bitmask of VkPipelineStageFlagBits specifying the source stage mask used to determine when the `event` is unsignaled.

`vkCmdResetEvent` behaves identically to vkCmdResetEvent2KHR.

### Valid Usage

- VUID-vkCmdResetEvent-stageMask-04090
  If the `geometryShader` feature is not enabled, `stageMask` **must** not contain `VK_PIPELINE_STAGE_GEOMETRY_SHADER_BIT`
- VUID-vkCmdResetEvent-stageMask-04091
  If the `tessellationShader` feature is not enabled, `stageMask` **must** not contain `VK_PIPELINE_STAGE_TESSELLATION_CONTROL_SHADER_BIT` or `VK_PIPELINE_STAGE_TESSELLATION_EVALUATION_SHADER_BIT`
- VUID-vkCmdResetEvent-stageMask-04092
  If the `conditionalRendering` feature is not enabled, `stageMask` **must** not contain `VK_PIPELINE_STAGE_CONDITIONAL_RENDERING_BIT_EXT`
- VUID-vkCmdResetEvent-stageMask-04093
  If the `fragmentDensityMap` feature is not enabled, `stageMask` **must** not contain `VK_PIPELINE_STAGE_FRAGMENT_DENSITY_PROCESS_BIT_EXT`
- VUID-vkCmdResetEvent-stageMask-04094
  If the `transformFeedback` feature is not enabled, `stageMask` **must** not contain `VK_PIPELINE_STAGE_TRANSFORM_FEEDBACK_BIT_EXT`

- VUID-vkCmdResetEvent-stageMask-04095
  If the `meshShader` feature is not enabled, `stageMask` **must** not contain `VK_PIPELINE_STAGE_MESH_SHADER_BIT_EXT`
- VUID-vkCmdResetEvent-stageMask-04096
  If the `taskShader` feature is not enabled, `stageMask` **must** not contain `VK_PIPELINE_STAGE_TASK_SHADER_BIT_EXT`
- VUID-vkCmdResetEvent-stageMask-07318
  If neither the `shadingRateImage` or `attachmentFragmentShadingRate` are enabled, `stageMask` **must** not contain `VK_PIPELINE_STAGE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR`
- VUID-vkCmdResetEvent-stageMask-03937
  If the `synchronization2` feature is not enabled, `stageMask` **must** not be `0`
- VUID-vkCmdResetEvent-stageMask-07949
  If neither the VK_NV_ray_tracing extension or `rayTracingPipeline` feature are enabled, `stageMask` **must** not contain `VK_PIPELINE_STAGE_RAY_TRACING_SHADER_BIT_KHR`
- VUID-vkCmdResetEvent-stageMask-06458
  Any pipeline stage included in `stageMask` **must** be supported by the capabilities of the queue family specified by the `queueFamilyIndex` member of the VkCommandPoolCreateInfo structure that was used to create the `VkCommandPool` that `commandBuffer` was allocated from, as specified in the table of supported pipeline stages
- VUID-vkCmdResetEvent-stageMask-01153
  `stageMask` **must** not include `VK_PIPELINE_STAGE_HOST_BIT`
- VUID-vkCmdResetEvent-event-03834
  There **must** be an execution dependency between `vkCmdResetEvent` and the execution of any vkCmdWaitEvents that includes `event` in its `pEvents` parameter
- VUID-vkCmdResetEvent-event-03835
  There **must** be an execution dependency between `vkCmdResetEvent` and the execution of any vkCmdWaitEvents2KHR that includes `event` in its `pEvents` parameter
- VUID-vkCmdResetEvent-commandBuffer-01157
  `commandBuffer`'s current device mask **must** include exactly one physical device

## Valid Usage (Implicit)

- VUID-vkCmdResetEvent-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdResetEvent-event-parameter
  `event` **must** be a valid VkEvent handle
- VUID-vkCmdResetEvent-stageMask-parameter
  `stageMask` **must** be a valid combination of VkPipelineStageFlagBits values
- VUID-vkCmdResetEvent-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdResetEvent-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics, compute,

decode, or encode operations

- VUID-vkCmdResetEvent-renderpass
  This command **must** only be called outside of a render pass instance
- VUID-vkCmdResetEvent-commonparent
  Both of `commandBuffer`, and `event` **must** have been created, allocated, or retrieved from the same VkDevice

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Outside | Both | Graphics<br>Compute<br>Decode<br>Encode | Synchronization |

To wait for one or more events to enter the signaled state on a device, call:

```
// Provided by VK_KHR_synchronization2
void vkCmdWaitEvents2KHR(
    VkCommandBuffer                             commandBuffer,
    uint32_t                                    eventCount,
    const VkEvent*                              pEvents,
    const VkDependencyInfo*                     pDependencyInfos);
```

- `commandBuffer` is the command buffer into which the command is recorded.
- `eventCount` is the length of the `pEvents` array.
- `pEvents` is a pointer to an array of `eventCount` events to wait on.
- `pDependencyInfos` is a pointer to an array of `eventCount` VkDependencyInfo structures, defining the second synchronization scope.

When `vkCmdWaitEvents2KHR` is submitted to a queue, it inserts memory dependencies according to the elements of `pDependencyInfos` and each corresponding element of `pEvents`. `vkCmdWaitEvents2KHR` **must** not be used to wait on event signal operations occurring on other queues, or signal operations executed by vkCmdSetEvent.

The first synchronization scope and access scope of each memory dependency defined by any element i of pDependencyInfos are applied to operations that occurred earlier in submission order than the last event signal operation on element i of pEvents.

Signal operations for an event at index i are only included if:

- The event was signaled by a vkCmdSetEvent2KHR command that occurred earlier in submission order with a dependencyInfo parameter exactly equal to the element of pDependencyInfos at index i ; or
- The event was created without VK_EVENT_CREATE_DEVICE_ONLY_BIT, and the first synchronization scope defined by the element of pDependencyInfos at index i only includes host operations (VK_PIPELINE_STAGE_2_HOST_BIT).

The second synchronization scope and access scope of each memory dependency defined by any element i of pDependencyInfos are applied to operations that occurred later in submission order than vkCmdWaitEvents2KHR.

**NOTE** vkCmdWaitEvents2KHR is used with vkCmdSetEvent2KHR to define a memory dependency between two sets of action commands, roughly in the same way as pipeline barriers, but split into two commands such that work between the two **may** execute unhindered.

**NOTE** Applications should be careful to avoid race conditions when using events. There is no direct ordering guarantee between vkCmdSetEvent2KHR and vkCmdResetEvent2KHR, vkCmdResetEvent, or vkCmdSetEvent. Another execution dependency (e.g. a pipeline barrier or semaphore with VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT) is needed to prevent such a race condition.

### Valid Usage

- VUID-vkCmdWaitEvents2-synchronization2-03836
  The synchronization2 feature **must** be enabled
- VUID-vkCmdWaitEvents2-pEvents-03837
  Members of pEvents **must** not have been signaled by vkCmdSetEvent
- VUID-vkCmdWaitEvents2-pEvents-03838
  For any element i of pEvents, if that event is signaled by vkCmdSetEvent2KHR, that command's dependencyInfo parameter **must** be exactly equal to the ith element of pDependencyInfos
- VUID-vkCmdWaitEvents2-pEvents-03839
  For any element i of pEvents, if that event is signaled by vkSetEvent, barriers in the ith element of pDependencyInfos **must** include only host operations in their first synchronization scope
- VUID-vkCmdWaitEvents2-pEvents-03840
  For any element i of pEvents, if barriers in the ith element of pDependencyInfos include only host operations, the ith element of pEvents **must** be signaled before vkCmdWaitEvents2KHR is executed

- VUID-vkCmdWaitEvents2-pEvents-03841
  For any element i of `pEvents`, if barriers in the ith element of `pDependencyInfos` do not include host operations, the ith element of `pEvents` **must** be signaled by a corresponding vkCmdSetEvent2KHR that occurred earlier in submission order
- VUID-vkCmdWaitEvents2-srcStageMask-03842
  The `srcStageMask` member of any element of the `pMemoryBarriers`, `pBufferMemoryBarriers`, or `pImageMemoryBarriers` members of `pDependencyInfos` **must** only include pipeline stages valid for the queue family that was used to create the command pool that `commandBuffer` was allocated from
- VUID-vkCmdWaitEvents2-dstStageMask-03843
  The `dstStageMask` member of any element of the `pMemoryBarriers`, `pBufferMemoryBarriers`, or `pImageMemoryBarriers` members of `pDependencyInfos` **must** only include pipeline stages valid for the queue family that was used to create the command pool that `commandBuffer` was allocated from
- VUID-vkCmdWaitEvents2-dependencyFlags-03844
  If `vkCmdWaitEvents2KHR` is being called inside a render pass instance, the `srcStageMask` member of any element of the `pMemoryBarriers`, `pBufferMemoryBarriers`, or `pImageMemoryBarriers` members of `pDependencyInfos` **must** not include `VK_PIPELINE_STAGE_2_HOST_BIT`
- VUID-vkCmdWaitEvents2-commandBuffer-03846
  `commandBuffer`'s current device mask **must** include exactly one physical device

## Valid Usage (Implicit)

- VUID-vkCmdWaitEvents2-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdWaitEvents2-pEvents-parameter
  `pEvents` **must** be a valid pointer to an array of `eventCount` valid VkEvent handles
- VUID-vkCmdWaitEvents2-pDependencyInfos-parameter
  `pDependencyInfos` **must** be a valid pointer to an array of `eventCount` valid VkDependencyInfo structures
- VUID-vkCmdWaitEvents2-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdWaitEvents2-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics, compute, decode, or encode operations
- VUID-vkCmdWaitEvents2-eventCount-arraylength
  `eventCount` **must** be greater than `0`
- VUID-vkCmdWaitEvents2-commonparent
  Both of `commandBuffer`, and the elements of `pEvents` **must** have been created, allocated, or retrieved from the same VkDevice

### Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

### Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Both | Graphics<br>Compute<br>Decode<br>Encode | Synchronization |

To wait for one or more events to enter the signaled state on a device, call:

```
// Provided by VK_VERSION_1_0
void vkCmdWaitEvents(
    VkCommandBuffer                             commandBuffer,
    uint32_t                                    eventCount,
    const VkEvent*                              pEvents,
    VkPipelineStageFlags                        srcStageMask,
    VkPipelineStageFlags                        dstStageMask,
    uint32_t                                    memoryBarrierCount,
    const VkMemoryBarrier*                      pMemoryBarriers,
    uint32_t                                    bufferMemoryBarrierCount,
    const VkBufferMemoryBarrier*                pBufferMemoryBarriers,
    uint32_t                                    imageMemoryBarrierCount,
    const VkImageMemoryBarrier*                 pImageMemoryBarriers);
```

- `commandBuffer` is the command buffer into which the command is recorded.
- `eventCount` is the length of the `pEvents` array.
- `pEvents` is a pointer to an array of event object handles to wait on.
- `srcStageMask` is a bitmask of VkPipelineStageFlagBits specifying the source stage mask.
- `dstStageMask` is a bitmask of VkPipelineStageFlagBits specifying the destination stage mask.
- `memoryBarrierCount` is the length of the `pMemoryBarriers` array.
- `pMemoryBarriers` is a pointer to an array of VkMemoryBarrier structures.
- `bufferMemoryBarrierCount` is the length of the `pBufferMemoryBarriers` array.
- `pBufferMemoryBarriers` is a pointer to an array of VkBufferMemoryBarrier structures.
- `imageMemoryBarrierCount` is the length of the `pImageMemoryBarriers` array.

- pImageMemoryBarriers is a pointer to an array of VkImageMemoryBarrier structures.

vkCmdWaitEvents is largely similar to vkCmdWaitEvents2KHR, but **can** only wait on signal operations defined by vkCmdSetEvent. As vkCmdSetEvent does not define any access scopes, vkCmdWaitEvents defines the first access scope for each event signal operation in addition to its own access scopes.

**NOTE** Since vkCmdSetEvent does not have any dependency information beyond a stage mask, implementations do not have the same opportunity to perform availability and visibility operations or image layout transitions in advance as they do with vkCmdSetEvent2KHR and vkCmdWaitEvents2KHR.

When vkCmdWaitEvents is submitted to a queue, it defines a memory dependency between prior event signal operations on the same queue or the host, and subsequent commands. vkCmdWaitEvents **must** not be used to wait on event signal operations occurring on other queues.

The first synchronization scope only includes event signal operations that operate on members of pEvents, and the operations that happened-before the event signal operations. Event signal operations performed by vkCmdSetEvent that occur earlier in submission order are included in the first synchronization scope, if the logically latest pipeline stage in their stageMask parameter is logically earlier than or equal to the logically latest pipeline stage in srcStageMask. Event signal operations performed by vkSetEvent are only included in the first synchronization scope if VK_PIPELINE_STAGE_HOST_BIT is included in srcStageMask.

The second synchronization scope includes all commands that occur later in submission order. The second synchronization scope is limited to operations on the pipeline stages determined by the destination stage mask specified by dstStageMask.

The first access scope is limited to accesses in the pipeline stages determined by the source stage mask specified by srcStageMask. Within that, the first access scope only includes the first access scopes defined by elements of the pMemoryBarriers, pBufferMemoryBarriers and pImageMemoryBarriers arrays, which each define a set of memory barriers. If no memory barriers are specified, then the first access scope includes no accesses.

The second access scope is limited to accesses in the pipeline stages determined by the destination stage mask specified by dstStageMask. Within that, the second access scope only includes the second access scopes defined by elements of the pMemoryBarriers, pBufferMemoryBarriers and pImageMemoryBarriers arrays, which each define a set of memory barriers. If no memory barriers are specified, then the second access scope includes no accesses.

### Valid Usage

- VUID-vkCmdWaitEvents-srcStageMask-04090
  If the geometryShader feature is not enabled, srcStageMask **must** not contain VK_PIPELINE_STAGE_GEOMETRY_SHADER_BIT
- VUID-vkCmdWaitEvents-srcStageMask-04091
  If the tessellationShader feature is not enabled, srcStageMask **must** not contain VK_PIPELINE_STAGE_TESSELLATION_CONTROL_SHADER_BIT or VK_PIPELINE_STAGE_TESSELLATION_EVALUATION_SHADER_BIT

- VUID-vkCmdWaitEvents-srcStageMask-04092
  If the conditionalRendering feature is not enabled, srcStageMask **must** not contain VK_PIPELINE_STAGE_CONDITIONAL_RENDERING_BIT_EXT
- VUID-vkCmdWaitEvents-srcStageMask-04093
  If the fragmentDensityMap feature is not enabled, srcStageMask **must** not contain VK_PIPELINE_STAGE_FRAGMENT_DENSITY_PROCESS_BIT_EXT
- VUID-vkCmdWaitEvents-srcStageMask-04094
  If the transformFeedback feature is not enabled, srcStageMask **must** not contain VK_PIPELINE_STAGE_TRANSFORM_FEEDBACK_BIT_EXT
- VUID-vkCmdWaitEvents-srcStageMask-04095
  If the meshShader feature is not enabled, srcStageMask **must** not contain VK_PIPELINE_STAGE_MESH_SHADER_BIT_EXT
- VUID-vkCmdWaitEvents-srcStageMask-04096
  If the taskShader feature is not enabled, srcStageMask **must** not contain VK_PIPELINE_STAGE_TASK_SHADER_BIT_EXT
- VUID-vkCmdWaitEvents-srcStageMask-07318
  If neither the shadingRateImage or attachmentFragmentShadingRate are enabled, srcStageMask **must** not contain VK_PIPELINE_STAGE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR
- VUID-vkCmdWaitEvents-srcStageMask-03937
  If the synchronization2 feature is not enabled, srcStageMask **must** not be 0
- VUID-vkCmdWaitEvents-srcStageMask-07949
  If neither the VK_NV_ray_tracing extension or rayTracingPipeline feature are enabled, srcStageMask **must** not contain VK_PIPELINE_STAGE_RAY_TRACING_SHADER_BIT_KHR
- VUID-vkCmdWaitEvents-srcAccessMask-06257
  If the rayQuery feature is not enabled and a memory barrier srcAccessMask includes VK_ACCESS_ACCELERATION_STRUCTURE_READ_BIT_KHR, srcStageMask **must** not include any of the VK_PIPELINE_STAGE_*_SHADER_BIT stages except VK_PIPELINE_STAGE_RAY_TRACING_SHADER_BIT_KHR
- VUID-vkCmdWaitEvents-dstStageMask-04090
  If the geometryShader feature is not enabled, dstStageMask **must** not contain VK_PIPELINE_STAGE_GEOMETRY_SHADER_BIT
- VUID-vkCmdWaitEvents-dstStageMask-04091
  If the tessellationShader feature is not enabled, dstStageMask **must** not contain VK_PIPELINE_STAGE_TESSELLATION_CONTROL_SHADER_BIT or VK_PIPELINE_STAGE_TESSELLATION_EVALUATION_SHADER_BIT
- VUID-vkCmdWaitEvents-dstStageMask-04092
  If the conditionalRendering feature is not enabled, dstStageMask **must** not contain VK_PIPELINE_STAGE_CONDITIONAL_RENDERING_BIT_EXT
- VUID-vkCmdWaitEvents-dstStageMask-04093
  If the fragmentDensityMap feature is not enabled, dstStageMask **must** not contain VK_PIPELINE_STAGE_FRAGMENT_DENSITY_PROCESS_BIT_EXT

- VUID-vkCmdWaitEvents-dstStageMask-04094
  If the `transformFeedback` feature is not enabled, `dstStageMask` **must** not contain `VK_PIPELINE_STAGE_TRANSFORM_FEEDBACK_BIT_EXT`
- VUID-vkCmdWaitEvents-dstStageMask-04095
  If the `meshShader` feature is not enabled, `dstStageMask` **must** not contain `VK_PIPELINE_STAGE_MESH_SHADER_BIT_EXT`
- VUID-vkCmdWaitEvents-dstStageMask-04096
  If the `taskShader` feature is not enabled, `dstStageMask` **must** not contain `VK_PIPELINE_STAGE_TASK_SHADER_BIT_EXT`
- VUID-vkCmdWaitEvents-dstStageMask-07318
  If neither the `shadingRateImage` or `attachmentFragmentShadingRate` are enabled, `dstStageMask` **must** not contain `VK_PIPELINE_STAGE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR`
- VUID-vkCmdWaitEvents-dstStageMask-03937
  If the `synchronization2` feature is not enabled, `dstStageMask` **must** not be `0`
- VUID-vkCmdWaitEvents-dstStageMask-07949
  If neither the VK_NV_ray_tracing extension or `rayTracingPipeline` feature are enabled, `dstStageMask` **must** not contain `VK_PIPELINE_STAGE_RAY_TRACING_SHADER_BIT_KHR`
- VUID-vkCmdWaitEvents-dstAccessMask-06257
  If the `rayQuery` feature is not enabled and a memory barrier `dstAccessMask` includes `VK_ACCESS_ACCELERATION_STRUCTURE_READ_BIT_KHR`, `dstStageMask` **must** not include any of the `VK_PIPELINE_STAGE_*_SHADER_BIT` stages except `VK_PIPELINE_STAGE_RAY_TRACING_SHADER_BIT_KHR`
- VUID-vkCmdWaitEvents-srcAccessMask-02815
  The `srcAccessMask` member of each element of `pMemoryBarriers` **must** only include access flags that are supported by one or more of the pipeline stages in `srcStageMask`, as specified in the table of supported access types
- VUID-vkCmdWaitEvents-dstAccessMask-02816
  The `dstAccessMask` member of each element of `pMemoryBarriers` **must** only include access flags that are supported by one or more of the pipeline stages in `dstStageMask`, as specified in the table of supported access types
- VUID-vkCmdWaitEvents-pBufferMemoryBarriers-02817
  For any element of `pBufferMemoryBarriers`, if its `srcQueueFamilyIndex` and `dstQueueFamilyIndex` members are equal, or if its `srcQueueFamilyIndex` is the queue family index that was used to create the command pool that `commandBuffer` was allocated from, then its `srcAccessMask` member **must** only contain access flags that are supported by one or more of the pipeline stages in `srcStageMask`, as specified in the table of supported access types
- VUID-vkCmdWaitEvents-pBufferMemoryBarriers-02818
  For any element of `pBufferMemoryBarriers`, if its `srcQueueFamilyIndex` and `dstQueueFamilyIndex` members are equal, or if its `dstQueueFamilyIndex` is the queue family index that was used to create the command pool that `commandBuffer` was allocated from, then its `dstAccessMask` member **must** only contain access flags that are supported by one

or more of the pipeline stages in `dstStageMask`, as specified in the table of supported access types

- VUID-vkCmdWaitEvents-pImageMemoryBarriers-02819
  For any element of `pImageMemoryBarriers`, if its `srcQueueFamilyIndex` and `dstQueueFamilyIndex` members are equal, or if its `srcQueueFamilyIndex` is the queue family index that was used to create the command pool that `commandBuffer` was allocated from, then its `srcAccessMask` member **must** only contain access flags that are supported by one or more of the pipeline stages in `srcStageMask`, as specified in the table of supported access types
- VUID-vkCmdWaitEvents-pImageMemoryBarriers-02820
  For any element of `pImageMemoryBarriers`, if its `srcQueueFamilyIndex` and `dstQueueFamilyIndex` members are equal, or if its `dstQueueFamilyIndex` is the queue family index that was used to create the command pool that `commandBuffer` was allocated from, then its `dstAccessMask` member **must** only contain access flags that are supported by one or more of the pipeline stages in `dstStageMask`, as specified in the table of supported access types
- VUID-vkCmdWaitEvents-srcStageMask-06459
  Any pipeline stage included in `srcStageMask` **must** be supported by the capabilities of the queue family specified by the `queueFamilyIndex` member of the VkCommandPoolCreateInfo structure that was used to create the `VkCommandPool` that `commandBuffer` was allocated from, as specified in the table of supported pipeline stages
- VUID-vkCmdWaitEvents-dstStageMask-06460
  Any pipeline stage included in `dstStageMask` **must** be supported by the capabilities of the queue family specified by the `queueFamilyIndex` member of the VkCommandPoolCreateInfo structure that was used to create the `VkCommandPool` that `commandBuffer` was allocated from, as specified in the table of supported pipeline stages
- VUID-vkCmdWaitEvents-srcStageMask-01158
  `srcStageMask` **must** be the bitwise OR of the `stageMask` parameter used in previous calls to `vkCmdSetEvent` with any of the elements of `pEvents` and `VK_PIPELINE_STAGE_HOST_BIT` if any of the elements of `pEvents` was set using `vkSetEvent`
- VUID-vkCmdWaitEvents-srcStageMask-07308
  If `vkCmdWaitEvents` is being called inside a render pass instance, `srcStageMask` **must** not include `VK_PIPELINE_STAGE_HOST_BIT`
- VUID-vkCmdWaitEvents-srcQueueFamilyIndex-02803
  The `srcQueueFamilyIndex` and `dstQueueFamilyIndex` members of any element of `pBufferMemoryBarriers` or `pImageMemoryBarriers` **must** be equal
- VUID-vkCmdWaitEvents-commandBuffer-01167
  `commandBuffer`'s current device mask **must** include exactly one physical device
- VUID-vkCmdWaitEvents-pEvents-03847
  Elements of `pEvents` **must** not have been signaled by vkCmdSetEvent2KHR

## Valid Usage (Implicit)

- VUID-vkCmdWaitEvents-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdWaitEvents-pEvents-parameter
  `pEvents` **must** be a valid pointer to an array of `eventCount` valid VkEvent handles
- VUID-vkCmdWaitEvents-srcStageMask-parameter
  `srcStageMask` **must** be a valid combination of VkPipelineStageFlagBits values
- VUID-vkCmdWaitEvents-dstStageMask-parameter
  `dstStageMask` **must** be a valid combination of VkPipelineStageFlagBits values
- VUID-vkCmdWaitEvents-pMemoryBarriers-parameter
  If `memoryBarrierCount` is not `0`, `pMemoryBarriers` **must** be a valid pointer to an array of `memoryBarrierCount` valid VkMemoryBarrier structures
- VUID-vkCmdWaitEvents-pBufferMemoryBarriers-parameter
  If `bufferMemoryBarrierCount` is not `0`, `pBufferMemoryBarriers` **must** be a valid pointer to an array of `bufferMemoryBarrierCount` valid VkBufferMemoryBarrier structures
- VUID-vkCmdWaitEvents-pImageMemoryBarriers-parameter
  If `imageMemoryBarrierCount` is not `0`, `pImageMemoryBarriers` **must** be a valid pointer to an array of `imageMemoryBarrierCount` valid VkImageMemoryBarrier structures
- VUID-vkCmdWaitEvents-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdWaitEvents-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics, compute, decode, or encode operations
- VUID-vkCmdWaitEvents-eventCount-arraylength
  `eventCount` **must** be greater than `0`
- VUID-vkCmdWaitEvents-commonparent
  Both of `commandBuffer`, and the elements of `pEvents` **must** have been created, allocated, or retrieved from the same VkDevice

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

### Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Both | Graphics<br>Compute<br>Decode<br>Encode | Synchronization |

## 7.6. Pipeline Barriers

To record a pipeline barrier, call:

```
// Provided by VK_KHR_synchronization2
void vkCmdPipelineBarrier2KHR(
    VkCommandBuffer                             commandBuffer,
    const VkDependencyInfo*                     pDependencyInfo);
```

- `commandBuffer` is the command buffer into which the command is recorded.
- `pDependencyInfo` is a pointer to a VkDependencyInfo structure defining the scopes of this operation.

When vkCmdPipelineBarrier2KHR is submitted to a queue, it defines memory dependencies between commands that were submitted to the same queue before it, and those submitted to the same queue after it.

The first synchronization scope and access scope of each memory dependency defined by `pDependencyInfo` are applied to operations that occurred earlier in submission order.

The second synchronization scope and access scope of each memory dependency defined by `pDependencyInfo` are applied to operations that occurred later in submission order.

If `vkCmdPipelineBarrier2KHR` is recorded within a render pass instance, the synchronization scopes are limited to a subset of operations within the same subpass or render pass instance.

### Valid Usage

- VUID-vkCmdPipelineBarrier2-None-07889
  If `vkCmdPipelineBarrier2KHR` is called within a render pass instance using a VkRenderPass object, the render pass **must** have been created with at least one subpass dependency that expresses a dependency from the current subpass to itself, does not include `VK_DEPENDENCY_BY_REGION_BIT` if this command does not, does not include `VK_DEPENDENCY_VIEW_LOCAL_BIT` if this command does not, and has synchronization scopes and access scopes that are all supersets of the scopes defined in this command

- VUID-vkCmdPipelineBarrier2-bufferMemoryBarrierCount-01178
  If `vkCmdPipelineBarrier2KHR` is called within a render pass instance using a VkRenderPass object, it **must** not include any buffer memory barriers
- VUID-vkCmdPipelineBarrier2-image-04073
  If `vkCmdPipelineBarrier2KHR` is called within a render pass instance using a VkRenderPass object, the `image` member of any image memory barrier included in this command **must** be an attachment used in the current subpass both as an input attachment, and as either a color, color resolve, or depth/stencil attachment
- VUID-vkCmdPipelineBarrier2-image-09373
  If `vkCmdPipelineBarrier2KHR` is called within a render pass instance using a VkRenderPass object, and the `image` member of any image memory barrier is a color resolve attachment, the corresponding color attachment **must** be `VK_ATTACHMENT_UNUSED`
- VUID-vkCmdPipelineBarrier2-image-09374
  If `vkCmdPipelineBarrier2KHR` is called within a render pass instance using a VkRenderPass object, and the `image` member of any image memory barrier is a color resolve attachment, it **must** have been created with a non-zero VkExternalFormatANDROID::`externalFormat` value
- VUID-vkCmdPipelineBarrier2-oldLayout-01181
  If `vkCmdPipelineBarrier2KHR` is called within a render pass instance, the `oldLayout` and `newLayout` members of any image memory barrier included in this command **must** be equal
- VUID-vkCmdPipelineBarrier2-srcQueueFamilyIndex-01182
  If `vkCmdPipelineBarrier2KHR` is called within a render pass instance, the `srcQueueFamilyIndex` and `dstQueueFamilyIndex` members of any memory barrier included in this command **must** be equal
- VUID-vkCmdPipelineBarrier2-None-07890
  If `vkCmdPipelineBarrier2KHR` is called within a render pass instance, and the source stage masks of any memory barriers include framebuffer-space stages, destination stage masks of all memory barriers **must** only include framebuffer-space stages
- VUID-vkCmdPipelineBarrier2-dependencyFlags-07891
  If `vkCmdPipelineBarrier2KHR` is called within a render pass instance, and the source stage masks of any memory barriers include framebuffer-space stages, then `dependencyFlags` **must** include `VK_DEPENDENCY_BY_REGION_BIT`
- VUID-vkCmdPipelineBarrier2-None-07892
  If `vkCmdPipelineBarrier2KHR` is called within a render pass instance, the source and destination stage masks of any memory barriers **must** only include graphics pipeline stages
- VUID-vkCmdPipelineBarrier2-dependencyFlags-01186
  If `vkCmdPipelineBarrier2KHR` is called outside of a render pass instance, the dependency flags **must** not include `VK_DEPENDENCY_VIEW_LOCAL_BIT`
- VUID-vkCmdPipelineBarrier2-None-07893
  If `vkCmdPipelineBarrier2KHR` is called inside a render pass instance, and there is more than one view in the current subpass, dependency flags **must** include `VK_DEPENDENCY_VIEW_LOCAL_BIT`

- VUID-vkCmdPipelineBarrier2-None-09553
  If none of the `shaderTileImageColorReadAccess`, `shaderTileImageStencilReadAccess`, or `shaderTileImageDepthReadAccess` features are enabled, and the `dynamicRenderingLocalRead` feature is not enabled, `vkCmdPipelineBarrier2KHR` **must** not be called within a render pass instance started with vkCmdBeginRenderingKHR
- VUID-vkCmdPipelineBarrier2-None-09554
  If the `dynamicRenderingLocalRead` feature is not enabled, and `vkCmdPipelineBarrier2KHR` is called within a render pass instance started with vkCmdBeginRenderingKHR, there **must** be no buffer or image memory barriers specified by this command
- VUID-vkCmdPipelineBarrier2-None-09586
  If the `dynamicRenderingLocalRead` feature is not enabled, and `vkCmdPipelineBarrier2KHR` is called within a render pass instance started with vkCmdBeginRenderingKHR, memory barriers specified by this command **must** only include `VK_ACCESS_2_COLOR_ATTACHMENT_READ_BIT`, `VK_ACCESS_2_COLOR_ATTACHMENT_WRITE_BIT`, `VK_ACCESS_2_DEPTH_STENCIL_ATTACHMENT_READ_BIT`, or `VK_ACCESS_2_DEPTH_STENCIL_ATTACHMENT_WRITE_BIT` in their access masks
- VUID-vkCmdPipelineBarrier2-image-09555
  If `vkCmdPipelineBarrier2KHR` is called within a render pass instance started with vkCmdBeginRenderingKHR, and the `image` member of any image memory barrier is used as an attachment in the current render pass instance, it **must** be in the `VK_IMAGE_LAYOUT_RENDERING_LOCAL_READ_KHR` or `VK_IMAGE_LAYOUT_GENERAL` layout
- VUID-vkCmdPipelineBarrier2-srcStageMask-09556
  If `vkCmdPipelineBarrier2KHR` is called within a render pass instance started with vkCmdBeginRenderingKHR, this command **must** only specify framebuffer-space stages in `srcStageMask` and `dstStageMask`
- VUID-vkCmdPipelineBarrier2-synchronization2-03848
  The `synchronization2` feature **must** be enabled
- VUID-vkCmdPipelineBarrier2-srcStageMask-09673
  The `srcStageMask` member of any element of the `pMemoryBarriers` member of `pDependencyInfo` **must** only include pipeline stages valid for the queue family that was used to create the command pool that `commandBuffer` was allocated from
- VUID-vkCmdPipelineBarrier2-dstStageMask-09674
  The `dstStageMask` member of any element of the `pMemoryBarriers` member of `pDependencyInfo` **must** only include pipeline stages valid for the queue family that was used to create the command pool that `commandBuffer` was allocated from
- VUID-vkCmdPipelineBarrier2-srcStageMask-09675
  If a buffer or image memory barrier does not specify an acquire operation, the respective `srcStageMask` member of the element of the `pBufferMemoryBarriers` or `pImageMemoryBarriers` members of `pDependencyInfo` **must** only include pipeline stages valid for the queue family that was used to create the command pool that `commandBuffer` was allocated from
- VUID-vkCmdPipelineBarrier2-dstStageMask-09676
  If a buffer or image memory barrier does not specify an release operation, the respective `dstStageMask` member of the element of the `pBufferMemoryBarriers` or `pImageMemoryBarriers` members of `pDependencyInfo` **must** only include pipeline stages valid for the queue family

that was used to create the command pool that `commandBuffer` was allocated from

## Valid Usage (Implicit)

- VUID-vkCmdPipelineBarrier2-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdPipelineBarrier2-pDependencyInfo-parameter
  `pDependencyInfo` **must** be a valid pointer to a valid VkDependencyInfo structure
- VUID-vkCmdPipelineBarrier2-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdPipelineBarrier2-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support transfer, graphics, compute, decode, or encode operations

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Both | Transfer<br>Graphics<br>Compute<br>Decode<br>Encode | Synchronization |

To record a pipeline barrier, call:

```
// Provided by VK_VERSION_1_0
void vkCmdPipelineBarrier(
    VkCommandBuffer                             commandBuffer,
    VkPipelineStageFlags                        srcStageMask,
    VkPipelineStageFlags                        dstStageMask,
    VkDependencyFlags                           dependencyFlags,
    uint32_t                                    memoryBarrierCount,
    const VkMemoryBarrier*                      pMemoryBarriers,
    uint32_t                                    bufferMemoryBarrierCount,
    const VkBufferMemoryBarrier*                pBufferMemoryBarriers,
    uint32_t                                    imageMemoryBarrierCount,
    const VkImageMemoryBarrier*                 pImageMemoryBarriers);
```

- `commandBuffer` is the command buffer into which the command is recorded.
- `srcStageMask` is a bitmask of VkPipelineStageFlagBits specifying the source stages.
- `dstStageMask` is a bitmask of VkPipelineStageFlagBits specifying the destination stages.
- `dependencyFlags` is a bitmask of VkDependencyFlagBits specifying how execution and memory dependencies are formed.
- `memoryBarrierCount` is the length of the `pMemoryBarriers` array.
- `pMemoryBarriers` is a pointer to an array of VkMemoryBarrier structures.
- `bufferMemoryBarrierCount` is the length of the `pBufferMemoryBarriers` array.
- `pBufferMemoryBarriers` is a pointer to an array of VkBufferMemoryBarrier structures.
- `imageMemoryBarrierCount` is the length of the `pImageMemoryBarriers` array.
- `pImageMemoryBarriers` is a pointer to an array of VkImageMemoryBarrier structures.

`vkCmdPipelineBarrier` operates almost identically to vkCmdPipelineBarrier2KHR, except that the scopes and barriers are defined as direct parameters rather than being defined by a VkDependencyInfo.

When vkCmdPipelineBarrier is submitted to a queue, it defines a memory dependency between commands that were submitted to the same queue before it, and those submitted to the same queue after it.

If vkCmdPipelineBarrier was recorded outside a render pass instance, the first synchronization scope includes all commands that occur earlier in submission order. If vkCmdPipelineBarrier was recorded inside a render pass instance, the first synchronization scope includes only commands that occur earlier in submission order within the same subpass. In either case, the first synchronization scope is limited to operations on the pipeline stages determined by the source stage mask specified by `srcStageMask`.

If vkCmdPipelineBarrier was recorded outside a render pass instance, the second synchronization scope includes all commands that occur later in submission order. If vkCmdPipelineBarrier was recorded inside a render pass instance, the second synchronization scope includes only commands that occur later in submission order within the same subpass. In either case, the second synchronization scope is limited to operations on the pipeline stages determined by the destination

stage mask specified by dstStageMask.

The first access scope is limited to accesses in the pipeline stages determined by the source stage mask specified by srcStageMask. Within that, the first access scope only includes the first access scopes defined by elements of the pMemoryBarriers, pBufferMemoryBarriers and pImageMemoryBarriers arrays, which each define a set of memory barriers. If no memory barriers are specified, then the first access scope includes no accesses.

The second access scope is limited to accesses in the pipeline stages determined by the destination stage mask specified by dstStageMask. Within that, the second access scope only includes the second access scopes defined by elements of the pMemoryBarriers, pBufferMemoryBarriers and pImageMemoryBarriers arrays, which each define a set of memory barriers. If no memory barriers are specified, then the second access scope includes no accesses.

If dependencyFlags includes VK_DEPENDENCY_BY_REGION_BIT, then any dependency between framebuffer-space pipeline stages is framebuffer-local - otherwise it is framebuffer-global.

## Valid Usage

- VUID-vkCmdPipelineBarrier-srcStageMask-04090
  If the geometryShader feature is not enabled, srcStageMask **must** not contain VK_PIPELINE_STAGE_GEOMETRY_SHADER_BIT
- VUID-vkCmdPipelineBarrier-srcStageMask-04091
  If the tessellationShader feature is not enabled, srcStageMask **must** not contain VK_PIPELINE_STAGE_TESSELLATION_CONTROL_SHADER_BIT or VK_PIPELINE_STAGE_TESSELLATION_EVALUATION_SHADER_BIT
- VUID-vkCmdPipelineBarrier-srcStageMask-04092
  If the conditionalRendering feature is not enabled, srcStageMask **must** not contain VK_PIPELINE_STAGE_CONDITIONAL_RENDERING_BIT_EXT
- VUID-vkCmdPipelineBarrier-srcStageMask-04093
  If the fragmentDensityMap feature is not enabled, srcStageMask **must** not contain VK_PIPELINE_STAGE_FRAGMENT_DENSITY_PROCESS_BIT_EXT
- VUID-vkCmdPipelineBarrier-srcStageMask-04094
  If the transformFeedback feature is not enabled, srcStageMask **must** not contain VK_PIPELINE_STAGE_TRANSFORM_FEEDBACK_BIT_EXT
- VUID-vkCmdPipelineBarrier-srcStageMask-04095
  If the meshShader feature is not enabled, srcStageMask **must** not contain VK_PIPELINE_STAGE_MESH_SHADER_BIT_EXT
- VUID-vkCmdPipelineBarrier-srcStageMask-04096
  If the taskShader feature is not enabled, srcStageMask **must** not contain VK_PIPELINE_STAGE_TASK_SHADER_BIT_EXT
- VUID-vkCmdPipelineBarrier-srcStageMask-07318
  If neither the shadingRateImage or attachmentFragmentShadingRate are enabled, srcStageMask **must** not contain VK_PIPELINE_STAGE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR

- VUID-vkCmdPipelineBarrier-srcStageMask-03937
  If the synchronization2 feature is not enabled, srcStageMask **must** not be 0
- VUID-vkCmdPipelineBarrier-srcStageMask-07949
  If neither the VK_NV_ray_tracing extension or rayTracingPipeline feature are enabled, srcStageMask **must** not contain VK_PIPELINE_STAGE_RAY_TRACING_SHADER_BIT_KHR
- VUID-vkCmdPipelineBarrier-srcAccessMask-06257
  If the rayQuery feature is not enabled and a memory barrier srcAccessMask includes VK_ACCESS_ACCELERATION_STRUCTURE_READ_BIT_KHR, srcStageMask **must** not include any of the VK_PIPELINE_STAGE_*_SHADER_BIT stages except VK_PIPELINE_STAGE_RAY_TRACING_SHADER_BIT_KHR
- VUID-vkCmdPipelineBarrier-dstStageMask-04090
  If the geometryShader feature is not enabled, dstStageMask **must** not contain VK_PIPELINE_STAGE_GEOMETRY_SHADER_BIT
- VUID-vkCmdPipelineBarrier-dstStageMask-04091
  If the tessellationShader feature is not enabled, dstStageMask **must** not contain VK_PIPELINE_STAGE_TESSELLATION_CONTROL_SHADER_BIT or VK_PIPELINE_STAGE_TESSELLATION_EVALUATION_SHADER_BIT
- VUID-vkCmdPipelineBarrier-dstStageMask-04092
  If the conditionalRendering feature is not enabled, dstStageMask **must** not contain VK_PIPELINE_STAGE_CONDITIONAL_RENDERING_BIT_EXT
- VUID-vkCmdPipelineBarrier-dstStageMask-04093
  If the fragmentDensityMap feature is not enabled, dstStageMask **must** not contain VK_PIPELINE_STAGE_FRAGMENT_DENSITY_PROCESS_BIT_EXT
- VUID-vkCmdPipelineBarrier-dstStageMask-04094
  If the transformFeedback feature is not enabled, dstStageMask **must** not contain VK_PIPELINE_STAGE_TRANSFORM_FEEDBACK_BIT_EXT
- VUID-vkCmdPipelineBarrier-dstStageMask-04095
  If the meshShader feature is not enabled, dstStageMask **must** not contain VK_PIPELINE_STAGE_MESH_SHADER_BIT_EXT
- VUID-vkCmdPipelineBarrier-dstStageMask-04096
  If the taskShader feature is not enabled, dstStageMask **must** not contain VK_PIPELINE_STAGE_TASK_SHADER_BIT_EXT
- VUID-vkCmdPipelineBarrier-dstStageMask-07318
  If neither the shadingRateImage or attachmentFragmentShadingRate are enabled, dstStageMask **must** not contain VK_PIPELINE_STAGE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR
- VUID-vkCmdPipelineBarrier-dstStageMask-03937
  If the synchronization2 feature is not enabled, dstStageMask **must** not be 0
- VUID-vkCmdPipelineBarrier-dstStageMask-07949
  If neither the VK_NV_ray_tracing extension or rayTracingPipeline feature are enabled, dstStageMask **must** not contain VK_PIPELINE_STAGE_RAY_TRACING_SHADER_BIT_KHR

- VUID-vkCmdPipelineBarrier-dstAccessMask-06257
  If the `rayQuery` feature is not enabled and a memory barrier `dstAccessMask` includes `VK_ACCESS_ACCELERATION_STRUCTURE_READ_BIT_KHR`, `dstStageMask` **must** not include any of the `VK_PIPELINE_STAGE_*_SHADER_BIT` stages except `VK_PIPELINE_STAGE_RAY_TRACING_SHADER_BIT_KHR`

- VUID-vkCmdPipelineBarrier-srcAccessMask-02815
  The `srcAccessMask` member of each element of `pMemoryBarriers` **must** only include access flags that are supported by one or more of the pipeline stages in `srcStageMask`, as specified in the table of supported access types

- VUID-vkCmdPipelineBarrier-dstAccessMask-02816
  The `dstAccessMask` member of each element of `pMemoryBarriers` **must** only include access flags that are supported by one or more of the pipeline stages in `dstStageMask`, as specified in the table of supported access types

- VUID-vkCmdPipelineBarrier-pBufferMemoryBarriers-02817
  For any element of `pBufferMemoryBarriers`, if its `srcQueueFamilyIndex` and `dstQueueFamilyIndex` members are equal, or if its `srcQueueFamilyIndex` is the queue family index that was used to create the command pool that `commandBuffer` was allocated from, then its `srcAccessMask` member **must** only contain access flags that are supported by one or more of the pipeline stages in `srcStageMask`, as specified in the table of supported access types

- VUID-vkCmdPipelineBarrier-pBufferMemoryBarriers-02818
  For any element of `pBufferMemoryBarriers`, if its `srcQueueFamilyIndex` and `dstQueueFamilyIndex` members are equal, or if its `dstQueueFamilyIndex` is the queue family index that was used to create the command pool that `commandBuffer` was allocated from, then its `dstAccessMask` member **must** only contain access flags that are supported by one or more of the pipeline stages in `dstStageMask`, as specified in the table of supported access types

- VUID-vkCmdPipelineBarrier-pImageMemoryBarriers-02819
  For any element of `pImageMemoryBarriers`, if its `srcQueueFamilyIndex` and `dstQueueFamilyIndex` members are equal, or if its `srcQueueFamilyIndex` is the queue family index that was used to create the command pool that `commandBuffer` was allocated from, then its `srcAccessMask` member **must** only contain access flags that are supported by one or more of the pipeline stages in `srcStageMask`, as specified in the table of supported access types

- VUID-vkCmdPipelineBarrier-pImageMemoryBarriers-02820
  For any element of `pImageMemoryBarriers`, if its `srcQueueFamilyIndex` and `dstQueueFamilyIndex` members are equal, or if its `dstQueueFamilyIndex` is the queue family index that was used to create the command pool that `commandBuffer` was allocated from, then its `dstAccessMask` member **must** only contain access flags that are supported by one or more of the pipeline stages in `dstStageMask`, as specified in the table of supported access types

- VUID-vkCmdPipelineBarrier-None-07889
  If `vkCmdPipelineBarrier` is called within a render pass instance using a VkRenderPass object, the render pass **must** have been created with at least one subpass dependency that

expresses a dependency from the current subpass to itself, does not include VK_DEPENDENCY_BY_REGION_BIT if this command does not, does not include VK_DEPENDENCY_VIEW_LOCAL_BIT if this command does not, and has synchronization scopes and access scopes that are all supersets of the scopes defined in this command

- VUID-vkCmdPipelineBarrier-bufferMemoryBarrierCount-01178
  If vkCmdPipelineBarrier is called within a render pass instance using a VkRenderPass object, it **must** not include any buffer memory barriers
- VUID-vkCmdPipelineBarrier-image-04073
  If vkCmdPipelineBarrier is called within a render pass instance using a VkRenderPass object, the image member of any image memory barrier included in this command **must** be an attachment used in the current subpass both as an input attachment, and as either a color, color resolve, or depth/stencil attachment
- VUID-vkCmdPipelineBarrier-image-09373
  If vkCmdPipelineBarrier is called within a render pass instance using a VkRenderPass object, and the image member of any image memory barrier is a color resolve attachment, the corresponding color attachment **must** be VK_ATTACHMENT_UNUSED
- VUID-vkCmdPipelineBarrier-image-09374
  If vkCmdPipelineBarrier is called within a render pass instance using a VkRenderPass object, and the image member of any image memory barrier is a color resolve attachment, it **must** have been created with a non-zero VkExternalFormatANDROID::externalFormat value
- VUID-vkCmdPipelineBarrier-oldLayout-01181
  If vkCmdPipelineBarrier is called within a render pass instance, the oldLayout and newLayout members of any image memory barrier included in this command **must** be equal
- VUID-vkCmdPipelineBarrier-srcQueueFamilyIndex-01182
  If vkCmdPipelineBarrier is called within a render pass instance, the srcQueueFamilyIndex and dstQueueFamilyIndex members of any memory barrier included in this command **must** be equal
- VUID-vkCmdPipelineBarrier-None-07890
  If vkCmdPipelineBarrier is called within a render pass instance, and the source stage masks of any memory barriers include framebuffer-space stages, destination stage masks of all memory barriers **must** only include framebuffer-space stages
- VUID-vkCmdPipelineBarrier-dependencyFlags-07891
  If vkCmdPipelineBarrier is called within a render pass instance, and the source stage masks of any memory barriers include framebuffer-space stages, then dependencyFlags **must** include VK_DEPENDENCY_BY_REGION_BIT
- VUID-vkCmdPipelineBarrier-None-07892
  If vkCmdPipelineBarrier is called within a render pass instance, the source and destination stage masks of any memory barriers **must** only include graphics pipeline stages
- VUID-vkCmdPipelineBarrier-dependencyFlags-01186
  If vkCmdPipelineBarrier is called outside of a render pass instance, the dependency flags **must** not include VK_DEPENDENCY_VIEW_LOCAL_BIT

- VUID-vkCmdPipelineBarrier-None-07893
  If vkCmdPipelineBarrier is called inside a render pass instance, and there is more than one view in the current subpass, dependency flags **must** include VK_DEPENDENCY_VIEW_LOCAL_BIT
- VUID-vkCmdPipelineBarrier-None-09553
  If none of the shaderTileImageColorReadAccess, shaderTileImageStencilReadAccess, or shaderTileImageDepthReadAccess features are enabled, and the dynamicRenderingLocalRead feature is not enabled, vkCmdPipelineBarrier **must** not be called within a render pass instance started with vkCmdBeginRenderingKHR
- VUID-vkCmdPipelineBarrier-None-09554
  If the dynamicRenderingLocalRead feature is not enabled, and vkCmdPipelineBarrier is called within a render pass instance started with vkCmdBeginRenderingKHR, there **must** be no buffer or image memory barriers specified by this command
- VUID-vkCmdPipelineBarrier-None-09586
  If the dynamicRenderingLocalRead feature is not enabled, and vkCmdPipelineBarrier is called within a render pass instance started with vkCmdBeginRenderingKHR, memory barriers specified by this command **must** only include VK_ACCESS_2_COLOR_ATTACHMENT_READ_BIT, VK_ACCESS_2_COLOR_ATTACHMENT_WRITE_BIT, VK_ACCESS_2_DEPTH_STENCIL_ATTACHMENT_READ_BIT, or VK_ACCESS_2_DEPTH_STENCIL_ATTACHMENT_WRITE_BIT in their access masks
- VUID-vkCmdPipelineBarrier-image-09555
  If vkCmdPipelineBarrier is called within a render pass instance started with vkCmdBeginRenderingKHR, and the image member of any image memory barrier is used as an attachment in the current render pass instance, it **must** be in the VK_IMAGE_LAYOUT_RENDERING_LOCAL_READ_KHR or VK_IMAGE_LAYOUT_GENERAL layout
- VUID-vkCmdPipelineBarrier-srcStageMask-09556
  If vkCmdPipelineBarrier is called within a render pass instance started with vkCmdBeginRenderingKHR, this command **must** only specify framebuffer-space stages in srcStageMask and dstStageMask
- VUID-vkCmdPipelineBarrier-srcStageMask-06461
  Any pipeline stage included in srcStageMask **must** be supported by the capabilities of the queue family specified by the queueFamilyIndex member of the VkCommandPoolCreateInfo structure that was used to create the VkCommandPool that commandBuffer was allocated from, as specified in the table of supported pipeline stages
- VUID-vkCmdPipelineBarrier-dstStageMask-06462
  Any pipeline stage included in dstStageMask **must** be supported by the capabilities of the queue family specified by the queueFamilyIndex member of the VkCommandPoolCreateInfo structure that was used to create the VkCommandPool that commandBuffer was allocated from, as specified in the table of supported pipeline stages
- VUID-vkCmdPipelineBarrier-srcStageMask-09633
  If either srcStageMask or dstStageMask includes VK_PIPELINE_STAGE_HOST_BIT, for any element of pImageMemoryBarriers, srcQueueFamilyIndex and dstQueueFamilyIndex **must** be equal
- VUID-vkCmdPipelineBarrier-srcStageMask-09634
  If either srcStageMask or dstStageMask includes VK_PIPELINE_STAGE_HOST_BIT, for any element of pBufferMemoryBarriers, srcQueueFamilyIndex and dstQueueFamilyIndex **must** be

equal

## Valid Usage (Implicit)

- VUID-vkCmdPipelineBarrier-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdPipelineBarrier-srcStageMask-parameter
  `srcStageMask` **must** be a valid combination of VkPipelineStageFlagBits values
- VUID-vkCmdPipelineBarrier-dstStageMask-parameter
  `dstStageMask` **must** be a valid combination of VkPipelineStageFlagBits values
- VUID-vkCmdPipelineBarrier-dependencyFlags-parameter
  `dependencyFlags` **must** be a valid combination of VkDependencyFlagBits values
- VUID-vkCmdPipelineBarrier-pMemoryBarriers-parameter
  If `memoryBarrierCount` is not `0`, `pMemoryBarriers` **must** be a valid pointer to an array of `memoryBarrierCount` valid VkMemoryBarrier structures
- VUID-vkCmdPipelineBarrier-pBufferMemoryBarriers-parameter
  If `bufferMemoryBarrierCount` is not `0`, `pBufferMemoryBarriers` **must** be a valid pointer to an array of `bufferMemoryBarrierCount` valid VkBufferMemoryBarrier structures
- VUID-vkCmdPipelineBarrier-pImageMemoryBarriers-parameter
  If `imageMemoryBarrierCount` is not `0`, `pImageMemoryBarriers` **must** be a valid pointer to an array of `imageMemoryBarrierCount` valid VkImageMemoryBarrier structures
- VUID-vkCmdPipelineBarrier-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdPipelineBarrier-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support transfer, graphics, compute, decode, or encode operations

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

**Command Properties**

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Both | Transfer<br>Graphics<br>Compute<br>Decode<br>Encode | Synchronization |

Bits which **can** be set in `vkCmdPipelineBarrier::dependencyFlags`, specifying how execution and memory dependencies are formed, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkDependencyFlagBits {
    VK_DEPENDENCY_BY_REGION_BIT = 0x00000001,
  // Provided by VK_VERSION_1_1
    VK_DEPENDENCY_DEVICE_GROUP_BIT = 0x00000004,
  // Provided by VK_VERSION_1_1
    VK_DEPENDENCY_VIEW_LOCAL_BIT = 0x00000002,
  // Provided by VK_EXT_attachment_feedback_loop_layout
    VK_DEPENDENCY_FEEDBACK_LOOP_BIT_EXT = 0x00000008,
  // Provided by VK_KHR_multiview
    VK_DEPENDENCY_VIEW_LOCAL_BIT_KHR = VK_DEPENDENCY_VIEW_LOCAL_BIT,
  // Provided by VK_KHR_device_group
    VK_DEPENDENCY_DEVICE_GROUP_BIT_KHR = VK_DEPENDENCY_DEVICE_GROUP_BIT,
} VkDependencyFlagBits;
```

- `VK_DEPENDENCY_BY_REGION_BIT` specifies that dependencies will be framebuffer-local.
- `VK_DEPENDENCY_VIEW_LOCAL_BIT` specifies that dependencies will be view-local.
- `VK_DEPENDENCY_DEVICE_GROUP_BIT` specifies that dependencies are non-device-local.
- `VK_DEPENDENCY_FEEDBACK_LOOP_BIT_EXT` specifies that the render pass will write to and read from the same image using the `VK_IMAGE_LAYOUT_ATTACHMENT_FEEDBACK_LOOP_OPTIMAL_EXT` layout.

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkDependencyFlags;
```

`VkDependencyFlags` is a bitmask type for setting a mask of zero or more VkDependencyFlagBits.

## 7.7. Memory Barriers

*Memory barriers* are used to explicitly control access to buffer and image subresource ranges. Memory barriers are used to transfer ownership between queue families, change image layouts,

and define availability and visibility operations. They explicitly define the access types and buffer and image subresource ranges that are included in the access scopes of a memory dependency that is created by a synchronization command that includes them.

### 7.7.1. Global Memory Barriers

Global memory barriers apply to memory accesses involving all memory objects that exist at the time of its execution.

The `VkMemoryBarrier2` structure is defined as:

```
typedef struct VkMemoryBarrier2 {
    VkStructureType          sType;
    const void*              pNext;
    VkPipelineStageFlags2    srcStageMask;
    VkAccessFlags2           srcAccessMask;
    VkPipelineStageFlags2    dstStageMask;
    VkAccessFlags2           dstAccessMask;
} VkMemoryBarrier2;
```

or the equivalent

```
// Provided by VK_KHR_synchronization2
typedef VkMemoryBarrier2 VkMemoryBarrier2KHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `srcStageMask` is a VkPipelineStageFlags2 mask of pipeline stages to be included in the first synchronization scope.
- `srcAccessMask` is a VkAccessFlags2 mask of access flags to be included in the first access scope.
- `dstStageMask` is a VkPipelineStageFlags2 mask of pipeline stages to be included in the second synchronization scope.
- `dstAccessMask` is a VkAccessFlags2 mask of access flags to be included in the second access scope.

This structure defines a memory dependency affecting all device memory.

The first synchronization scope and access scope described by this structure include only operations and memory accesses specified by `srcStageMask` and `srcAccessMask`.

The second synchronization scope and access scope described by this structure include only operations and memory accesses specified by `dstStageMask` and `dstAccessMask`.

**Valid Usage**

- VUID-VkMemoryBarrier2-srcStageMask-03929

If the geometryShader feature is not enabled, srcStageMask **must** not contain VK_PIPELINE_STAGE_2_GEOMETRY_SHADER_BIT

- VUID-VkMemoryBarrier2-srcStageMask-03930
  If the tessellationShader feature is not enabled, srcStageMask **must** not contain VK_PIPELINE_STAGE_2_TESSELLATION_CONTROL_SHADER_BIT or VK_PIPELINE_STAGE_2_TESSELLATION_EVALUATION_SHADER_BIT
- VUID-VkMemoryBarrier2-srcStageMask-03931
  If the conditionalRendering feature is not enabled, srcStageMask **must** not contain VK_PIPELINE_STAGE_2_CONDITIONAL_RENDERING_BIT_EXT
- VUID-VkMemoryBarrier2-srcStageMask-03932
  If the fragmentDensityMap feature is not enabled, srcStageMask **must** not contain VK_PIPELINE_STAGE_2_FRAGMENT_DENSITY_PROCESS_BIT_EXT
- VUID-VkMemoryBarrier2-srcStageMask-03933
  If the transformFeedback feature is not enabled, srcStageMask **must** not contain VK_PIPELINE_STAGE_2_TRANSFORM_FEEDBACK_BIT_EXT
- VUID-VkMemoryBarrier2-srcStageMask-03934
  If the meshShader feature is not enabled, srcStageMask **must** not contain VK_PIPELINE_STAGE_2_MESH_SHADER_BIT_EXT
- VUID-VkMemoryBarrier2-srcStageMask-03935
  If the taskShader feature is not enabled, srcStageMask **must** not contain VK_PIPELINE_STAGE_2_TASK_SHADER_BIT_EXT
- VUID-VkMemoryBarrier2-srcStageMask-07316
  If neither the shadingRateImage or attachmentFragmentShadingRate are enabled, srcStageMask **must** not contain VK_PIPELINE_STAGE_2_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR
- VUID-VkMemoryBarrier2-srcStageMask-04957
  If the subpassShading feature is not enabled, srcStageMask **must** not contain VK_PIPELINE_STAGE_2_SUBPASS_SHADER_BIT_HUAWEI
- VUID-VkMemoryBarrier2-srcStageMask-04995
  If the invocationMask feature is not enabled, srcStageMask **must** not contain VK_PIPELINE_STAGE_2_INVOCATION_MASK_BIT_HUAWEI
- VUID-VkMemoryBarrier2-srcStageMask-07946
  If neither the VK_NV_ray_tracing extension or rayTracingPipeline feature are enabled, srcStageMask **must** not contain VK_PIPELINE_STAGE_2_RAY_TRACING_SHADER_BIT_KHR
- VUID-VkMemoryBarrier2-srcAccessMask-03900
  If srcAccessMask includes VK_ACCESS_2_INDIRECT_COMMAND_READ_BIT, srcStageMask **must** include VK_PIPELINE_STAGE_2_DRAW_INDIRECT_BIT, VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_BUILD_BIT_KHR, VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkMemoryBarrier2-srcAccessMask-03901
  If srcAccessMask includes VK_ACCESS_2_INDEX_READ_BIT, srcStageMask **must** include VK_PIPELINE_STAGE_2_INDEX_INPUT_BIT, VK_PIPELINE_STAGE_2_VERTEX_INPUT_BIT, VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT

- VUID-VkMemoryBarrier2-srcAccessMask-03902
  If srcAccessMask includes VK_ACCESS_2_VERTEX_ATTRIBUTE_READ_BIT, srcStageMask **must** include VK_PIPELINE_STAGE_2_VERTEX_ATTRIBUTE_INPUT_BIT, VK_PIPELINE_STAGE_2_VERTEX_INPUT_BIT, VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkMemoryBarrier2-srcAccessMask-03903
  If srcAccessMask includes VK_ACCESS_2_INPUT_ATTACHMENT_READ_BIT, srcStageMask **must** include VK_PIPELINE_STAGE_2_FRAGMENT_SHADER_BIT, VK_PIPELINE_STAGE_2_SUBPASS_SHADER_BIT_HUAWEI, VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkMemoryBarrier2-srcAccessMask-03904
  If srcAccessMask includes VK_ACCESS_2_UNIFORM_READ_BIT, srcStageMask **must** include VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT, or one of the VK_PIPELINE_STAGE_*_SHADER_BIT stages
- VUID-VkMemoryBarrier2-srcAccessMask-03905
  If srcAccessMask includes VK_ACCESS_2_SHADER_SAMPLED_READ_BIT, srcStageMask **must** include VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT, or one of the VK_PIPELINE_STAGE_*_SHADER_BIT stages
- VUID-VkMemoryBarrier2-srcAccessMask-03906
  If srcAccessMask includes VK_ACCESS_2_SHADER_STORAGE_READ_BIT, srcStageMask **must** include VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT, or one of the VK_PIPELINE_STAGE_*_SHADER_BIT stages
- VUID-VkMemoryBarrier2-srcAccessMask-03907
  If srcAccessMask includes VK_ACCESS_2_SHADER_STORAGE_WRITE_BIT, srcStageMask **must** include VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT, or one of the VK_PIPELINE_STAGE_*_SHADER_BIT stages
- VUID-VkMemoryBarrier2-srcAccessMask-07454
  If srcAccessMask includes VK_ACCESS_2_SHADER_READ_BIT, srcStageMask **must** include VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT, VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_BUILD_BIT_KHR, VK_PIPELINE_STAGE_2_MICROMAP_BUILD_BIT_EXT, or one of the VK_PIPELINE_STAGE_*_SHADER_BIT stages
- VUID-VkMemoryBarrier2-srcAccessMask-03909
  If srcAccessMask includes VK_ACCESS_2_SHADER_WRITE_BIT, srcStageMask **must** include VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT, or one of the VK_PIPELINE_STAGE_*_SHADER_BIT stages
- VUID-VkMemoryBarrier2-srcAccessMask-03910
  If srcAccessMask includes VK_ACCESS_2_COLOR_ATTACHMENT_READ_BIT, srcStageMask **must** include VK_PIPELINE_STAGE_2_COLOR_ATTACHMENT_OUTPUT_BIT VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkMemoryBarrier2-srcAccessMask-03911
  If srcAccessMask includes VK_ACCESS_2_COLOR_ATTACHMENT_WRITE_BIT, srcStageMask **must** include VK_PIPELINE_STAGE_2_COLOR_ATTACHMENT_OUTPUT_BIT VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT

- VUID-VkMemoryBarrier2-srcAccessMask-03912
  If srcAccessMask includes VK_ACCESS_2_DEPTH_STENCIL_ATTACHMENT_READ_BIT, srcStageMask **must** include VK_PIPELINE_STAGE_2_EARLY_FRAGMENT_TESTS_BIT, VK_PIPELINE_STAGE_2_LATE_FRAGMENT_TESTS_BIT, VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkMemoryBarrier2-srcAccessMask-03913
  If srcAccessMask includes VK_ACCESS_2_DEPTH_STENCIL_ATTACHMENT_WRITE_BIT, srcStageMask **must** include VK_PIPELINE_STAGE_2_EARLY_FRAGMENT_TESTS_BIT, VK_PIPELINE_STAGE_2_LATE_FRAGMENT_TESTS_BIT, VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkMemoryBarrier2-srcAccessMask-03914
  If srcAccessMask includes VK_ACCESS_2_TRANSFER_READ_BIT, srcStageMask **must** include VK_PIPELINE_STAGE_2_COPY_BIT, VK_PIPELINE_STAGE_2_BLIT_BIT, VK_PIPELINE_STAGE_2_RESOLVE_BIT, VK_PIPELINE_STAGE_2_ALL_TRANSFER_BIT, VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_BUILD_BIT_KHR, VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_COPY_BIT_KHR, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkMemoryBarrier2-srcAccessMask-03915
  If srcAccessMask includes VK_ACCESS_2_TRANSFER_WRITE_BIT, srcStageMask **must** include VK_PIPELINE_STAGE_2_COPY_BIT, VK_PIPELINE_STAGE_2_BLIT_BIT, VK_PIPELINE_STAGE_2_RESOLVE_BIT, VK_PIPELINE_STAGE_2_CLEAR_BIT, VK_PIPELINE_STAGE_2_ALL_TRANSFER_BIT, VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_BUILD_BIT_KHR, or VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_COPY_BIT_KHR, VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkMemoryBarrier2-srcAccessMask-03916
  If srcAccessMask includes VK_ACCESS_2_HOST_READ_BIT, srcStageMask **must** include VK_PIPELINE_STAGE_2_HOST_BIT
- VUID-VkMemoryBarrier2-srcAccessMask-03917
  If srcAccessMask includes VK_ACCESS_2_HOST_WRITE_BIT, srcStageMask **must** include VK_PIPELINE_STAGE_2_HOST_BIT
- VUID-VkMemoryBarrier2-srcAccessMask-03918
  If srcAccessMask includes VK_ACCESS_2_CONDITIONAL_RENDERING_READ_BIT_EXT, srcStageMask **must** include VK_PIPELINE_STAGE_2_CONDITIONAL_RENDERING_BIT_EXT, VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkMemoryBarrier2-srcAccessMask-03919
  If srcAccessMask includes VK_ACCESS_2_FRAGMENT_DENSITY_MAP_READ_BIT_EXT, srcStageMask **must** include VK_PIPELINE_STAGE_2_FRAGMENT_DENSITY_PROCESS_BIT_EXT, VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkMemoryBarrier2-srcAccessMask-03920
  If srcAccessMask includes VK_ACCESS_2_TRANSFORM_FEEDBACK_WRITE_BIT_EXT, srcStageMask **must** include VK_PIPELINE_STAGE_2_TRANSFORM_FEEDBACK_BIT_EXT, VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkMemoryBarrier2-srcAccessMask-04747

If srcAccessMask includes VK_ACCESS_2_TRANSFORM_FEEDBACK_COUNTER_READ_BIT_EXT, srcStageMask **must** include VK_PIPELINE_STAGE_2_DRAW_INDIRECT_BIT, VK_PIPELINE_STAGE_2_TRANSFORM_FEEDBACK_BIT_EXT, VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT

- VUID-VkMemoryBarrier2-srcAccessMask-03922
  If srcAccessMask includes VK_ACCESS_2_TRANSFORM_FEEDBACK_COUNTER_WRITE_BIT_EXT, srcStageMask **must** include VK_PIPELINE_STAGE_2_TRANSFORM_FEEDBACK_BIT_EXT, VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkMemoryBarrier2-srcAccessMask-03923
  If srcAccessMask includes VK_ACCESS_2_SHADING_RATE_IMAGE_READ_BIT_NV, srcStageMask **must** include VK_PIPELINE_STAGE_2_SHADING_RATE_IMAGE_BIT_NV, VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkMemoryBarrier2-srcAccessMask-04994
  If srcAccessMask includes VK_ACCESS_2_INVOCATION_MASK_READ_BIT_HUAWEI, srcStageMask **must** include VK_PIPELINE_STAGE_2_INVOCATION_MASK_BIT_HUAWEI
- VUID-VkMemoryBarrier2-srcAccessMask-03924
  If srcAccessMask includes VK_ACCESS_2_COMMAND_PREPROCESS_READ_BIT_NV, srcStageMask **must** include VK_PIPELINE_STAGE_2_COMMAND_PREPROCESS_BIT_NV or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkMemoryBarrier2-srcAccessMask-03925
  If srcAccessMask includes VK_ACCESS_2_COMMAND_PREPROCESS_WRITE_BIT_NV, srcStageMask **must** include VK_PIPELINE_STAGE_2_COMMAND_PREPROCESS_BIT_NV or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkMemoryBarrier2-srcAccessMask-03926
  If srcAccessMask includes VK_ACCESS_2_COLOR_ATTACHMENT_READ_NONCOHERENT_BIT_EXT, srcStageMask **must** include VK_PIPELINE_STAGE_2_COLOR_ATTACHMENT_OUTPUT_BIT VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkMemoryBarrier2-srcAccessMask-03927
  If srcAccessMask includes VK_ACCESS_2_ACCELERATION_STRUCTURE_READ_BIT_KHR, srcStageMask **must** include VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_BUILD_BIT_KHR, VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_COPY_BIT_KHR, VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT, or one of the VK_PIPELINE_STAGE_*_SHADER_BIT stages
- VUID-VkMemoryBarrier2-srcAccessMask-03928
  If srcAccessMask includes VK_ACCESS_2_ACCELERATION_STRUCTURE_WRITE_BIT_KHR, srcStageMask **must** include VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_COPY_BIT_KHR, VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_BUILD_BIT_KHR or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkMemoryBarrier2-srcAccessMask-06256
  If the rayQuery feature is not enabled and srcAccessMask includes VK_ACCESS_2_ACCELERATION_STRUCTURE_READ_BIT_KHR, srcStageMask **must** not include any of the VK_PIPELINE_STAGE_*_SHADER_BIT stages except VK_PIPELINE_STAGE_2_RAY_TRACING_SHADER_BIT_KHR
- VUID-VkMemoryBarrier2-srcAccessMask-07272

If srcAccessMask includes VK_ACCESS_2_SHADER_BINDING_TABLE_READ_BIT_KHR, srcStageMask **must** include VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT or VK_PIPELINE_STAGE_2_RAY_TRACING_SHADER_BIT_KHR

- VUID-VkMemoryBarrier2-srcAccessMask-04858
  If srcAccessMask includes VK_ACCESS_2_VIDEO_DECODE_READ_BIT_KHR, srcStageMask **must** include VK_PIPELINE_STAGE_2_VIDEO_DECODE_BIT_KHR
- VUID-VkMemoryBarrier2-srcAccessMask-04859
  If srcAccessMask includes VK_ACCESS_2_VIDEO_DECODE_WRITE_BIT_KHR, srcStageMask **must** include VK_PIPELINE_STAGE_2_VIDEO_DECODE_BIT_KHR
- VUID-VkMemoryBarrier2-srcAccessMask-04860
  If srcAccessMask includes VK_ACCESS_2_VIDEO_ENCODE_READ_BIT_KHR, srcStageMask **must** include VK_PIPELINE_STAGE_2_VIDEO_ENCODE_BIT_KHR
- VUID-VkMemoryBarrier2-srcAccessMask-04861
  If srcAccessMask includes VK_ACCESS_2_VIDEO_ENCODE_WRITE_BIT_KHR, srcStageMask **must** include VK_PIPELINE_STAGE_2_VIDEO_ENCODE_BIT_KHR
- VUID-VkMemoryBarrier2-srcAccessMask-07455
  If srcAccessMask includes VK_ACCESS_2_OPTICAL_FLOW_READ_BIT_NV, srcStageMask **must** include VK_PIPELINE_STAGE_2_OPTICAL_FLOW_BIT_NV
- VUID-VkMemoryBarrier2-srcAccessMask-07456
  If srcAccessMask includes VK_ACCESS_2_OPTICAL_FLOW_WRITE_BIT_NV, srcStageMask **must** include VK_PIPELINE_STAGE_2_OPTICAL_FLOW_BIT_NV
- VUID-VkMemoryBarrier2-srcAccessMask-07457
  If srcAccessMask includes VK_ACCESS_2_MICROMAP_WRITE_BIT_EXT, srcStageMask **must** include VK_PIPELINE_STAGE_2_MICROMAP_BUILD_BIT_EXT
- VUID-VkMemoryBarrier2-srcAccessMask-07458
  If srcAccessMask includes VK_ACCESS_2_MICROMAP_READ_BIT_EXT, srcStageMask **must** include VK_PIPELINE_STAGE_2_MICROMAP_BUILD_BIT_EXT or VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_BUILD_BIT_KHR
- VUID-VkMemoryBarrier2-srcAccessMask-08118
  If srcAccessMask includes VK_ACCESS_2_DESCRIPTOR_BUFFER_READ_BIT_EXT, srcStageMask **must** include VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT, or one of VK_PIPELINE_STAGE_*_SHADER_BIT stages

- VUID-VkMemoryBarrier2-dstStageMask-03929
  If the geometryShader feature is not enabled, dstStageMask **must** not contain VK_PIPELINE_STAGE_2_GEOMETRY_SHADER_BIT
- VUID-VkMemoryBarrier2-dstStageMask-03930
  If the tessellationShader feature is not enabled, dstStageMask **must** not contain VK_PIPELINE_STAGE_2_TESSELLATION_CONTROL_SHADER_BIT or VK_PIPELINE_STAGE_2_TESSELLATION_EVALUATION_SHADER_BIT
- VUID-VkMemoryBarrier2-dstStageMask-03931
  If the conditionalRendering feature is not enabled, dstStageMask **must** not contain VK_PIPELINE_STAGE_2_CONDITIONAL_RENDERING_BIT_EXT

- VUID-VkMemoryBarrier2-dstStageMask-03932
  If the fragmentDensityMap feature is not enabled, dstStageMask **must** not contain VK_PIPELINE_STAGE_2_FRAGMENT_DENSITY_PROCESS_BIT_EXT
- VUID-VkMemoryBarrier2-dstStageMask-03933
  If the transformFeedback feature is not enabled, dstStageMask **must** not contain VK_PIPELINE_STAGE_2_TRANSFORM_FEEDBACK_BIT_EXT
- VUID-VkMemoryBarrier2-dstStageMask-03934
  If the meshShader feature is not enabled, dstStageMask **must** not contain VK_PIPELINE_STAGE_2_MESH_SHADER_BIT_EXT
- VUID-VkMemoryBarrier2-dstStageMask-03935
  If the taskShader feature is not enabled, dstStageMask **must** not contain VK_PIPELINE_STAGE_2_TASK_SHADER_BIT_EXT
- VUID-VkMemoryBarrier2-dstStageMask-07316
  If neither the shadingRateImage or attachmentFragmentShadingRate are enabled, dstStageMask **must** not contain VK_PIPELINE_STAGE_2_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR
- VUID-VkMemoryBarrier2-dstStageMask-04957
  If the subpassShading feature is not enabled, dstStageMask **must** not contain VK_PIPELINE_STAGE_2_SUBPASS_SHADER_BIT_HUAWEI
- VUID-VkMemoryBarrier2-dstStageMask-04995
  If the invocationMask feature is not enabled, dstStageMask **must** not contain VK_PIPELINE_STAGE_2_INVOCATION_MASK_BIT_HUAWEI
- VUID-VkMemoryBarrier2-dstStageMask-07946
  If neither the VK_NV_ray_tracing extension or rayTracingPipeline feature are enabled, dstStageMask **must** not contain VK_PIPELINE_STAGE_2_RAY_TRACING_SHADER_BIT_KHR
- VUID-VkMemoryBarrier2-dstAccessMask-03900
  If dstAccessMask includes VK_ACCESS_2_INDIRECT_COMMAND_READ_BIT, dstStageMask **must** include VK_PIPELINE_STAGE_2_DRAW_INDIRECT_BIT, VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_BUILD_BIT_KHR, VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkMemoryBarrier2-dstAccessMask-03901
  If dstAccessMask includes VK_ACCESS_2_INDEX_READ_BIT, dstStageMask **must** include VK_PIPELINE_STAGE_2_INDEX_INPUT_BIT, VK_PIPELINE_STAGE_2_VERTEX_INPUT_BIT, VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkMemoryBarrier2-dstAccessMask-03902
  If dstAccessMask includes VK_ACCESS_2_VERTEX_ATTRIBUTE_READ_BIT, dstStageMask **must** include VK_PIPELINE_STAGE_2_VERTEX_ATTRIBUTE_INPUT_BIT, VK_PIPELINE_STAGE_2_VERTEX_INPUT_BIT, VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkMemoryBarrier2-dstAccessMask-03903
  If dstAccessMask includes VK_ACCESS_2_INPUT_ATTACHMENT_READ_BIT, dstStageMask **must** include VK_PIPELINE_STAGE_2_FRAGMENT_SHADER_BIT, VK_PIPELINE_STAGE_2_SUBPASS_SHADER_BIT_HUAWEI, VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or

VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT

- VUID-VkMemoryBarrier2-dstAccessMask-03904
  If dstAccessMask includes VK_ACCESS_2_UNIFORM_READ_BIT, dstStageMask **must** include VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT, or one of the VK_PIPELINE_STAGE_*_SHADER_BIT stages
- VUID-VkMemoryBarrier2-dstAccessMask-03905
  If dstAccessMask includes VK_ACCESS_2_SHADER_SAMPLED_READ_BIT, dstStageMask **must** include VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT, or one of the VK_PIPELINE_STAGE_*_SHADER_BIT stages
- VUID-VkMemoryBarrier2-dstAccessMask-03906
  If dstAccessMask includes VK_ACCESS_2_SHADER_STORAGE_READ_BIT, dstStageMask **must** include VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT, or one of the VK_PIPELINE_STAGE_*_SHADER_BIT stages
- VUID-VkMemoryBarrier2-dstAccessMask-03907
  If dstAccessMask includes VK_ACCESS_2_SHADER_STORAGE_WRITE_BIT, dstStageMask **must** include VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT, or one of the VK_PIPELINE_STAGE_*_SHADER_BIT stages
- VUID-VkMemoryBarrier2-dstAccessMask-07454
  If dstAccessMask includes VK_ACCESS_2_SHADER_READ_BIT, dstStageMask **must** include VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT, VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_BUILD_BIT_KHR, VK_PIPELINE_STAGE_2_MICROMAP_BUILD_BIT_EXT, or one of the VK_PIPELINE_STAGE_*_SHADER_BIT stages
- VUID-VkMemoryBarrier2-dstAccessMask-03909
  If dstAccessMask includes VK_ACCESS_2_SHADER_WRITE_BIT, dstStageMask **must** include VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT, or one of the VK_PIPELINE_STAGE_*_SHADER_BIT stages
- VUID-VkMemoryBarrier2-dstAccessMask-03910
  If dstAccessMask includes VK_ACCESS_2_COLOR_ATTACHMENT_READ_BIT, dstStageMask **must** include VK_PIPELINE_STAGE_2_COLOR_ATTACHMENT_OUTPUT_BIT VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkMemoryBarrier2-dstAccessMask-03911
  If dstAccessMask includes VK_ACCESS_2_COLOR_ATTACHMENT_WRITE_BIT, dstStageMask **must** include VK_PIPELINE_STAGE_2_COLOR_ATTACHMENT_OUTPUT_BIT VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkMemoryBarrier2-dstAccessMask-03912
  If dstAccessMask includes VK_ACCESS_2_DEPTH_STENCIL_ATTACHMENT_READ_BIT, dstStageMask **must** include VK_PIPELINE_STAGE_2_EARLY_FRAGMENT_TESTS_BIT, VK_PIPELINE_STAGE_2_LATE_FRAGMENT_TESTS_BIT, VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkMemoryBarrier2-dstAccessMask-03913
  If dstAccessMask includes VK_ACCESS_2_DEPTH_STENCIL_ATTACHMENT_WRITE_BIT, dstStageMask **must** include VK_PIPELINE_STAGE_2_EARLY_FRAGMENT_TESTS_BIT, VK_PIPELINE_STAGE_2_LATE_FRAGMENT_TESTS_BIT, VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or

VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT

- VUID-VkMemoryBarrier2-dstAccessMask-03914
  If dstAccessMask includes VK_ACCESS_2_TRANSFER_READ_BIT, dstStageMask **must** include VK_PIPELINE_STAGE_2_COPY_BIT, VK_PIPELINE_STAGE_2_BLIT_BIT, VK_PIPELINE_STAGE_2_RESOLVE_BIT, VK_PIPELINE_STAGE_2_ALL_TRANSFER_BIT, VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_BUILD_BIT_KHR, VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_COPY_BIT_KHR, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkMemoryBarrier2-dstAccessMask-03915
  If dstAccessMask includes VK_ACCESS_2_TRANSFER_WRITE_BIT, dstStageMask **must** include VK_PIPELINE_STAGE_2_COPY_BIT, VK_PIPELINE_STAGE_2_BLIT_BIT, VK_PIPELINE_STAGE_2_RESOLVE_BIT, VK_PIPELINE_STAGE_2_CLEAR_BIT, VK_PIPELINE_STAGE_2_ALL_TRANSFER_BIT, VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_BUILD_BIT_KHR, or VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_COPY_BIT_KHR, VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkMemoryBarrier2-dstAccessMask-03916
  If dstAccessMask includes VK_ACCESS_2_HOST_READ_BIT, dstStageMask **must** include VK_PIPELINE_STAGE_2_HOST_BIT
- VUID-VkMemoryBarrier2-dstAccessMask-03917
  If dstAccessMask includes VK_ACCESS_2_HOST_WRITE_BIT, dstStageMask **must** include VK_PIPELINE_STAGE_2_HOST_BIT
- VUID-VkMemoryBarrier2-dstAccessMask-03918
  If dstAccessMask includes VK_ACCESS_2_CONDITIONAL_RENDERING_READ_BIT_EXT, dstStageMask **must** include VK_PIPELINE_STAGE_2_CONDITIONAL_RENDERING_BIT_EXT, VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkMemoryBarrier2-dstAccessMask-03919
  If dstAccessMask includes VK_ACCESS_2_FRAGMENT_DENSITY_MAP_READ_BIT_EXT, dstStageMask **must** include VK_PIPELINE_STAGE_2_FRAGMENT_DENSITY_PROCESS_BIT_EXT, VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkMemoryBarrier2-dstAccessMask-03920
  If dstAccessMask includes VK_ACCESS_2_TRANSFORM_FEEDBACK_WRITE_BIT_EXT, dstStageMask **must** include VK_PIPELINE_STAGE_2_TRANSFORM_FEEDBACK_BIT_EXT, VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkMemoryBarrier2-dstAccessMask-04747
  If dstAccessMask includes VK_ACCESS_2_TRANSFORM_FEEDBACK_COUNTER_READ_BIT_EXT, dstStageMask **must** include VK_PIPELINE_STAGE_2_DRAW_INDIRECT_BIT, VK_PIPELINE_STAGE_2_TRANSFORM_FEEDBACK_BIT_EXT, VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkMemoryBarrier2-dstAccessMask-03922
  If dstAccessMask includes VK_ACCESS_2_TRANSFORM_FEEDBACK_COUNTER_WRITE_BIT_EXT, dstStageMask **must** include VK_PIPELINE_STAGE_2_TRANSFORM_FEEDBACK_BIT_EXT, VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkMemoryBarrier2-dstAccessMask-03923

If dstAccessMask includes VK_ACCESS_2_SHADING_RATE_IMAGE_READ_BIT_NV, dstStageMask **must** include VK_PIPELINE_STAGE_2_SHADING_RATE_IMAGE_BIT_NV, VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT

- VUID-VkMemoryBarrier2-dstAccessMask-04994
  If dstAccessMask includes VK_ACCESS_2_INVOCATION_MASK_READ_BIT_HUAWEI, dstStageMask **must** include VK_PIPELINE_STAGE_2_INVOCATION_MASK_BIT_HUAWEI
- VUID-VkMemoryBarrier2-dstAccessMask-03924
  If dstAccessMask includes VK_ACCESS_2_COMMAND_PREPROCESS_READ_BIT_NV, dstStageMask **must** include VK_PIPELINE_STAGE_2_COMMAND_PREPROCESS_BIT_NV or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkMemoryBarrier2-dstAccessMask-03925
  If dstAccessMask includes VK_ACCESS_2_COMMAND_PREPROCESS_WRITE_BIT_NV, dstStageMask **must** include VK_PIPELINE_STAGE_2_COMMAND_PREPROCESS_BIT_NV or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkMemoryBarrier2-dstAccessMask-03926
  If dstAccessMask includes VK_ACCESS_2_COLOR_ATTACHMENT_READ_NONCOHERENT_BIT_EXT, dstStageMask **must** include VK_PIPELINE_STAGE_2_COLOR_ATTACHMENT_OUTPUT_BIT VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkMemoryBarrier2-dstAccessMask-03927
  If dstAccessMask includes VK_ACCESS_2_ACCELERATION_STRUCTURE_READ_BIT_KHR, dstStageMask **must** include VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_BUILD_BIT_KHR, VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_COPY_BIT_KHR, VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT, or one of the VK_PIPELINE_STAGE_*_SHADER_BIT stages
- VUID-VkMemoryBarrier2-dstAccessMask-03928
  If dstAccessMask includes VK_ACCESS_2_ACCELERATION_STRUCTURE_WRITE_BIT_KHR, dstStageMask **must** include VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_COPY_BIT_KHR, VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_BUILD_BIT_KHR or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkMemoryBarrier2-dstAccessMask-06256
  If the rayQuery feature is not enabled and dstAccessMask includes VK_ACCESS_2_ACCELERATION_STRUCTURE_READ_BIT_KHR, dstStageMask **must** not include any of the VK_PIPELINE_STAGE_*_SHADER_BIT stages except VK_PIPELINE_STAGE_2_RAY_TRACING_SHADER_BIT_KHR
- VUID-VkMemoryBarrier2-dstAccessMask-07272
  If dstAccessMask includes VK_ACCESS_2_SHADER_BINDING_TABLE_READ_BIT_KHR, dstStageMask **must** include VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT or VK_PIPELINE_STAGE_2_RAY_TRACING_SHADER_BIT_KHR
- VUID-VkMemoryBarrier2-dstAccessMask-04858
  If dstAccessMask includes VK_ACCESS_2_VIDEO_DECODE_READ_BIT_KHR, dstStageMask **must** include VK_PIPELINE_STAGE_2_VIDEO_DECODE_BIT_KHR
- VUID-VkMemoryBarrier2-dstAccessMask-04859
  If dstAccessMask includes VK_ACCESS_2_VIDEO_DECODE_WRITE_BIT_KHR, dstStageMask **must** include VK_PIPELINE_STAGE_2_VIDEO_DECODE_BIT_KHR

- VUID-VkMemoryBarrier2-dstAccessMask-04860
  If `dstAccessMask` includes `VK_ACCESS_2_VIDEO_ENCODE_READ_BIT_KHR`, `dstStageMask` **must** include `VK_PIPELINE_STAGE_2_VIDEO_ENCODE_BIT_KHR`
- VUID-VkMemoryBarrier2-dstAccessMask-04861
  If `dstAccessMask` includes `VK_ACCESS_2_VIDEO_ENCODE_WRITE_BIT_KHR`, `dstStageMask` **must** include `VK_PIPELINE_STAGE_2_VIDEO_ENCODE_BIT_KHR`
- VUID-VkMemoryBarrier2-dstAccessMask-07455
  If `dstAccessMask` includes `VK_ACCESS_2_OPTICAL_FLOW_READ_BIT_NV`, `dstStageMask` **must** include `VK_PIPELINE_STAGE_2_OPTICAL_FLOW_BIT_NV`
- VUID-VkMemoryBarrier2-dstAccessMask-07456
  If `dstAccessMask` includes `VK_ACCESS_2_OPTICAL_FLOW_WRITE_BIT_NV`, `dstStageMask` **must** include `VK_PIPELINE_STAGE_2_OPTICAL_FLOW_BIT_NV`
- VUID-VkMemoryBarrier2-dstAccessMask-07457
  If `dstAccessMask` includes `VK_ACCESS_2_MICROMAP_WRITE_BIT_EXT`, `dstStageMask` **must** include `VK_PIPELINE_STAGE_2_MICROMAP_BUILD_BIT_EXT`
- VUID-VkMemoryBarrier2-dstAccessMask-07458
  If `dstAccessMask` includes `VK_ACCESS_2_MICROMAP_READ_BIT_EXT`, `dstStageMask` **must** include `VK_PIPELINE_STAGE_2_MICROMAP_BUILD_BIT_EXT` or `VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_BUILD_BIT_KHR`
- VUID-VkMemoryBarrier2-dstAccessMask-08118
  If `dstAccessMask` includes `VK_ACCESS_2_DESCRIPTOR_BUFFER_READ_BIT_EXT`, `dstStageMask` **must** include `VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT`, `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`, or one of `VK_PIPELINE_STAGE_*_SHADER_BIT` stages

### Valid Usage (Implicit)

- VUID-VkMemoryBarrier2-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_MEMORY_BARRIER_2`
- VUID-VkMemoryBarrier2-srcStageMask-parameter
  `srcStageMask` **must** be a valid combination of VkPipelineStageFlagBits2 values
- VUID-VkMemoryBarrier2-srcAccessMask-parameter
  `srcAccessMask` **must** be a valid combination of VkAccessFlagBits2 values
- VUID-VkMemoryBarrier2-dstStageMask-parameter
  `dstStageMask` **must** be a valid combination of VkPipelineStageFlagBits2 values
- VUID-VkMemoryBarrier2-dstAccessMask-parameter
  `dstAccessMask` **must** be a valid combination of VkAccessFlagBits2 values

The `VkMemoryBarrier` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkMemoryBarrier {
    VkStructureType    sType;
    const void*        pNext;
    VkAccessFlags      srcAccessMask;
    VkAccessFlags      dstAccessMask;
} VkMemoryBarrier;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `srcAccessMask` is a bitmask of VkAccessFlagBits specifying a source access mask.
- `dstAccessMask` is a bitmask of VkAccessFlagBits specifying a destination access mask.

The first access scope is limited to access types in the source access mask specified by `srcAccessMask`.

The second access scope is limited to access types in the destination access mask specified by `dstAccessMask`.

**Valid Usage (Implicit)**

- VUID-VkMemoryBarrier-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_MEMORY_BARRIER`
- VUID-VkMemoryBarrier-pNext-pNext
  `pNext` **must** be `NULL`
- VUID-VkMemoryBarrier-srcAccessMask-parameter
  `srcAccessMask` **must** be a valid combination of VkAccessFlagBits values
- VUID-VkMemoryBarrier-dstAccessMask-parameter
  `dstAccessMask` **must** be a valid combination of VkAccessFlagBits values

### 7.7.2. Buffer Memory Barriers

Buffer memory barriers only apply to memory accesses involving a specific buffer range. That is, a memory dependency formed from a buffer memory barrier is scoped to access via the specified buffer range. Buffer memory barriers **can** also be used to define a queue family ownership transfer for the specified buffer range.

The `VkBufferMemoryBarrier2` structure is defined as:

```
typedef struct VkBufferMemoryBarrier2 {
    VkStructureType          sType;
    const void*              pNext;
    VkPipelineStageFlags2    srcStageMask;
    VkAccessFlags2           srcAccessMask;
    VkPipelineStageFlags2    dstStageMask;
    VkAccessFlags2           dstAccessMask;
    uint32_t                 srcQueueFamilyIndex;
    uint32_t                 dstQueueFamilyIndex;
    VkBuffer                 buffer;
    VkDeviceSize             offset;
    VkDeviceSize             size;
} VkBufferMemoryBarrier2;
```

or the equivalent

```
// Provided by VK_KHR_synchronization2
typedef VkBufferMemoryBarrier2 VkBufferMemoryBarrier2KHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `srcStageMask` is a VkPipelineStageFlags2 mask of pipeline stages to be included in the first synchronization scope.
- `srcAccessMask` is a VkAccessFlags2 mask of access flags to be included in the first access scope.
- `dstStageMask` is a VkPipelineStageFlags2 mask of pipeline stages to be included in the second synchronization scope.
- `dstAccessMask` is a VkAccessFlags2 mask of access flags to be included in the second access scope.
- `srcQueueFamilyIndex` is the source queue family for a queue family ownership transfer.
- `dstQueueFamilyIndex` is the destination queue family for a queue family ownership transfer.
- `buffer` is a handle to the buffer whose backing memory is affected by the barrier.
- `offset` is an offset in bytes into the backing memory for `buffer`; this is relative to the base offset as bound to the buffer (see vkBindBufferMemory).
- `size` is a size in bytes of the affected area of backing memory for `buffer`, or `VK_WHOLE_SIZE` to use the range from `offset` to the end of the buffer.

This structure defines a memory dependency limited to a range of a buffer, and **can** define a queue family ownership transfer operation for that range.

The first synchronization scope and access scope described by this structure include only operations and memory accesses specified by `srcStageMask` and `srcAccessMask`.

The second synchronization scope and access scope described by this structure include only operations and memory accesses specified by `dstStageMask` and `dstAccessMask`.

Both access scopes are limited to only memory accesses to `buffer` in the range defined by `offset` and `size`.

If `buffer` was created with `VK_SHARING_MODE_EXCLUSIVE`, and `srcQueueFamilyIndex` is not equal to `dstQueueFamilyIndex`, this memory barrier defines a queue family ownership transfer operation. When executed on a queue in the family identified by `srcQueueFamilyIndex`, this barrier defines a queue family release operation for the specified buffer range, and the second synchronization scope does not apply to this operation. When executed on a queue in the family identified by `dstQueueFamilyIndex`, this barrier defines a queue family acquire operation for the specified buffer range, and the first synchronization scope does not apply to this operation.

A queue family ownership transfer operation is also defined if the values are not equal, and either is one of the special queue family values reserved for external memory ownership transfers, as described in Queue Family Ownership Transfer. A queue family release operation is defined when `dstQueueFamilyIndex` is one of those values, and a queue family acquire operation is defined when `srcQueueFamilyIndex` is one of those values.

### Valid Usage

- VUID-VkBufferMemoryBarrier2-srcStageMask-03929
  If the `geometryShader` feature is not enabled, `srcStageMask` **must** not contain `VK_PIPELINE_STAGE_2_GEOMETRY_SHADER_BIT`
- VUID-VkBufferMemoryBarrier2-srcStageMask-03930
  If the `tessellationShader` feature is not enabled, `srcStageMask` **must** not contain `VK_PIPELINE_STAGE_2_TESSELLATION_CONTROL_SHADER_BIT` or `VK_PIPELINE_STAGE_2_TESSELLATION_EVALUATION_SHADER_BIT`
- VUID-VkBufferMemoryBarrier2-srcStageMask-03931
  If the `conditionalRendering` feature is not enabled, `srcStageMask` **must** not contain `VK_PIPELINE_STAGE_2_CONDITIONAL_RENDERING_BIT_EXT`
- VUID-VkBufferMemoryBarrier2-srcStageMask-03932
  If the `fragmentDensityMap` feature is not enabled, `srcStageMask` **must** not contain `VK_PIPELINE_STAGE_2_FRAGMENT_DENSITY_PROCESS_BIT_EXT`
- VUID-VkBufferMemoryBarrier2-srcStageMask-03933
  If the `transformFeedback` feature is not enabled, `srcStageMask` **must** not contain `VK_PIPELINE_STAGE_2_TRANSFORM_FEEDBACK_BIT_EXT`
- VUID-VkBufferMemoryBarrier2-srcStageMask-03934
  If the `meshShader` feature is not enabled, `srcStageMask` **must** not contain `VK_PIPELINE_STAGE_2_MESH_SHADER_BIT_EXT`
- VUID-VkBufferMemoryBarrier2-srcStageMask-03935
  If the `taskShader` feature is not enabled, `srcStageMask` **must** not contain `VK_PIPELINE_STAGE_2_TASK_SHADER_BIT_EXT`
- VUID-VkBufferMemoryBarrier2-srcStageMask-07316
  If neither the `shadingRateImage` or `attachmentFragmentShadingRate` are enabled, `srcStageMask` **must** not contain `VK_PIPELINE_STAGE_2_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR`

- VUID-VkBufferMemoryBarrier2-srcStageMask-04957
  If the subpassShading feature is not enabled, srcStageMask **must** not contain VK_PIPELINE_STAGE_2_SUBPASS_SHADER_BIT_HUAWEI
- VUID-VkBufferMemoryBarrier2-srcStageMask-04995
  If the invocationMask feature is not enabled, srcStageMask **must** not contain VK_PIPELINE_STAGE_2_INVOCATION_MASK_BIT_HUAWEI
- VUID-VkBufferMemoryBarrier2-srcStageMask-07946
  If neither the VK_NV_ray_tracing extension or rayTracingPipeline feature are enabled, srcStageMask **must** not contain VK_PIPELINE_STAGE_2_RAY_TRACING_SHADER_BIT_KHR
- VUID-VkBufferMemoryBarrier2-srcAccessMask-03900
  If srcAccessMask includes VK_ACCESS_2_INDIRECT_COMMAND_READ_BIT, srcStageMask **must** include VK_PIPELINE_STAGE_2_DRAW_INDIRECT_BIT, VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_BUILD_BIT_KHR, VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkBufferMemoryBarrier2-srcAccessMask-03901
  If srcAccessMask includes VK_ACCESS_2_INDEX_READ_BIT, srcStageMask **must** include VK_PIPELINE_STAGE_2_INDEX_INPUT_BIT, VK_PIPELINE_STAGE_2_VERTEX_INPUT_BIT, VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkBufferMemoryBarrier2-srcAccessMask-03902
  If srcAccessMask includes VK_ACCESS_2_VERTEX_ATTRIBUTE_READ_BIT, srcStageMask **must** include VK_PIPELINE_STAGE_2_VERTEX_ATTRIBUTE_INPUT_BIT, VK_PIPELINE_STAGE_2_VERTEX_INPUT_BIT, VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkBufferMemoryBarrier2-srcAccessMask-03903
  If srcAccessMask includes VK_ACCESS_2_INPUT_ATTACHMENT_READ_BIT, srcStageMask **must** include VK_PIPELINE_STAGE_2_FRAGMENT_SHADER_BIT, VK_PIPELINE_STAGE_2_SUBPASS_SHADER_BIT_HUAWEI, VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkBufferMemoryBarrier2-srcAccessMask-03904
  If srcAccessMask includes VK_ACCESS_2_UNIFORM_READ_BIT, srcStageMask **must** include VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT, or one of the VK_PIPELINE_STAGE_*_SHADER_BIT stages
- VUID-VkBufferMemoryBarrier2-srcAccessMask-03905
  If srcAccessMask includes VK_ACCESS_2_SHADER_SAMPLED_READ_BIT, srcStageMask **must** include VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT, or one of the VK_PIPELINE_STAGE_*_SHADER_BIT stages
- VUID-VkBufferMemoryBarrier2-srcAccessMask-03906
  If srcAccessMask includes VK_ACCESS_2_SHADER_STORAGE_READ_BIT, srcStageMask **must** include VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT, or one of the VK_PIPELINE_STAGE_*_SHADER_BIT stages
- VUID-VkBufferMemoryBarrier2-srcAccessMask-03907
  If srcAccessMask includes VK_ACCESS_2_SHADER_STORAGE_WRITE_BIT, srcStageMask **must** include VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT, or

one of the VK_PIPELINE_STAGE_*_SHADER_BIT stages

- VUID-VkBufferMemoryBarrier2-srcAccessMask-07454
  If srcAccessMask includes VK_ACCESS_2_SHADER_READ_BIT, srcStageMask **must** include VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT, VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_BUILD_BIT_KHR, VK_PIPELINE_STAGE_2_MICROMAP_BUILD_BIT_EXT, or one of the VK_PIPELINE_STAGE_*_SHADER_BIT stages
- VUID-VkBufferMemoryBarrier2-srcAccessMask-03909
  If srcAccessMask includes VK_ACCESS_2_SHADER_WRITE_BIT, srcStageMask **must** include VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT, or one of the VK_PIPELINE_STAGE_*_SHADER_BIT stages
- VUID-VkBufferMemoryBarrier2-srcAccessMask-03910
  If srcAccessMask includes VK_ACCESS_2_COLOR_ATTACHMENT_READ_BIT, srcStageMask **must** include VK_PIPELINE_STAGE_2_COLOR_ATTACHMENT_OUTPUT_BIT VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkBufferMemoryBarrier2-srcAccessMask-03911
  If srcAccessMask includes VK_ACCESS_2_COLOR_ATTACHMENT_WRITE_BIT, srcStageMask **must** include VK_PIPELINE_STAGE_2_COLOR_ATTACHMENT_OUTPUT_BIT VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkBufferMemoryBarrier2-srcAccessMask-03912
  If srcAccessMask includes VK_ACCESS_2_DEPTH_STENCIL_ATTACHMENT_READ_BIT, srcStageMask **must** include VK_PIPELINE_STAGE_2_EARLY_FRAGMENT_TESTS_BIT, VK_PIPELINE_STAGE_2_LATE_FRAGMENT_TESTS_BIT, VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkBufferMemoryBarrier2-srcAccessMask-03913
  If srcAccessMask includes VK_ACCESS_2_DEPTH_STENCIL_ATTACHMENT_WRITE_BIT, srcStageMask **must** include VK_PIPELINE_STAGE_2_EARLY_FRAGMENT_TESTS_BIT, VK_PIPELINE_STAGE_2_LATE_FRAGMENT_TESTS_BIT, VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkBufferMemoryBarrier2-srcAccessMask-03914
  If srcAccessMask includes VK_ACCESS_2_TRANSFER_READ_BIT, srcStageMask **must** include VK_PIPELINE_STAGE_2_COPY_BIT, VK_PIPELINE_STAGE_2_BLIT_BIT, VK_PIPELINE_STAGE_2_RESOLVE_BIT, VK_PIPELINE_STAGE_2_ALL_TRANSFER_BIT, VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_BUILD_BIT_KHR, VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_COPY_BIT_KHR, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkBufferMemoryBarrier2-srcAccessMask-03915
  If srcAccessMask includes VK_ACCESS_2_TRANSFER_WRITE_BIT, srcStageMask **must** include VK_PIPELINE_STAGE_2_COPY_BIT, VK_PIPELINE_STAGE_2_BLIT_BIT, VK_PIPELINE_STAGE_2_RESOLVE_BIT, VK_PIPELINE_STAGE_2_CLEAR_BIT, VK_PIPELINE_STAGE_2_ALL_TRANSFER_BIT, VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_BUILD_BIT_KHR, or VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_COPY_BIT_KHR, VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT

- VUID-VkBufferMemoryBarrier2-srcAccessMask-03916
  If srcAccessMask includes VK_ACCESS_2_HOST_READ_BIT, srcStageMask **must** include VK_PIPELINE_STAGE_2_HOST_BIT
- VUID-VkBufferMemoryBarrier2-srcAccessMask-03917
  If srcAccessMask includes VK_ACCESS_2_HOST_WRITE_BIT, srcStageMask **must** include VK_PIPELINE_STAGE_2_HOST_BIT
- VUID-VkBufferMemoryBarrier2-srcAccessMask-03918
  If srcAccessMask includes VK_ACCESS_2_CONDITIONAL_RENDERING_READ_BIT_EXT, srcStageMask **must** include VK_PIPELINE_STAGE_2_CONDITIONAL_RENDERING_BIT_EXT, VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkBufferMemoryBarrier2-srcAccessMask-03919
  If srcAccessMask includes VK_ACCESS_2_FRAGMENT_DENSITY_MAP_READ_BIT_EXT, srcStageMask **must** include VK_PIPELINE_STAGE_2_FRAGMENT_DENSITY_PROCESS_BIT_EXT, VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkBufferMemoryBarrier2-srcAccessMask-03920
  If srcAccessMask includes VK_ACCESS_2_TRANSFORM_FEEDBACK_WRITE_BIT_EXT, srcStageMask **must** include VK_PIPELINE_STAGE_2_TRANSFORM_FEEDBACK_BIT_EXT, VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkBufferMemoryBarrier2-srcAccessMask-04747
  If srcAccessMask includes VK_ACCESS_2_TRANSFORM_FEEDBACK_COUNTER_READ_BIT_EXT, srcStageMask **must** include VK_PIPELINE_STAGE_2_DRAW_INDIRECT_BIT, VK_PIPELINE_STAGE_2_TRANSFORM_FEEDBACK_BIT_EXT, VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkBufferMemoryBarrier2-srcAccessMask-03922
  If srcAccessMask includes VK_ACCESS_2_TRANSFORM_FEEDBACK_COUNTER_WRITE_BIT_EXT, srcStageMask **must** include VK_PIPELINE_STAGE_2_TRANSFORM_FEEDBACK_BIT_EXT, VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkBufferMemoryBarrier2-srcAccessMask-03923
  If srcAccessMask includes VK_ACCESS_2_SHADING_RATE_IMAGE_READ_BIT_NV, srcStageMask **must** include VK_PIPELINE_STAGE_2_SHADING_RATE_IMAGE_BIT_NV, VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkBufferMemoryBarrier2-srcAccessMask-04994
  If srcAccessMask includes VK_ACCESS_2_INVOCATION_MASK_READ_BIT_HUAWEI, srcStageMask **must** include VK_PIPELINE_STAGE_2_INVOCATION_MASK_BIT_HUAWEI
- VUID-VkBufferMemoryBarrier2-srcAccessMask-03924
  If srcAccessMask includes VK_ACCESS_2_COMMAND_PREPROCESS_READ_BIT_NV, srcStageMask **must** include VK_PIPELINE_STAGE_2_COMMAND_PREPROCESS_BIT_NV or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkBufferMemoryBarrier2-srcAccessMask-03925
  If srcAccessMask includes VK_ACCESS_2_COMMAND_PREPROCESS_WRITE_BIT_NV, srcStageMask **must** include VK_PIPELINE_STAGE_2_COMMAND_PREPROCESS_BIT_NV or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkBufferMemoryBarrier2-srcAccessMask-03926

If srcAccessMask includes VK_ACCESS_2_COLOR_ATTACHMENT_READ_NONCOHERENT_BIT_EXT, srcStageMask **must** include VK_PIPELINE_STAGE_2_COLOR_ATTACHMENT_OUTPUT_BIT VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT

- VUID-VkBufferMemoryBarrier2-srcAccessMask-03927
  If srcAccessMask includes VK_ACCESS_2_ACCELERATION_STRUCTURE_READ_BIT_KHR, srcStageMask **must** include VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_BUILD_BIT_KHR, VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_COPY_BIT_KHR, VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT, or one of the VK_PIPELINE_STAGE_*_SHADER_BIT stages
- VUID-VkBufferMemoryBarrier2-srcAccessMask-03928
  If srcAccessMask includes VK_ACCESS_2_ACCELERATION_STRUCTURE_WRITE_BIT_KHR, srcStageMask **must** include VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_COPY_BIT_KHR, VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_BUILD_BIT_KHR or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkBufferMemoryBarrier2-srcAccessMask-06256
  If the rayQuery feature is not enabled and srcAccessMask includes VK_ACCESS_2_ACCELERATION_STRUCTURE_READ_BIT_KHR, srcStageMask **must** not include any of the VK_PIPELINE_STAGE_*_SHADER_BIT stages except VK_PIPELINE_STAGE_2_RAY_TRACING_SHADER_BIT_KHR
- VUID-VkBufferMemoryBarrier2-srcAccessMask-07272
  If srcAccessMask includes VK_ACCESS_2_SHADER_BINDING_TABLE_READ_BIT_KHR, srcStageMask **must** include VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT or VK_PIPELINE_STAGE_2_RAY_TRACING_SHADER_BIT_KHR
- VUID-VkBufferMemoryBarrier2-srcAccessMask-04858
  If srcAccessMask includes VK_ACCESS_2_VIDEO_DECODE_READ_BIT_KHR, srcStageMask **must** include VK_PIPELINE_STAGE_2_VIDEO_DECODE_BIT_KHR
- VUID-VkBufferMemoryBarrier2-srcAccessMask-04859
  If srcAccessMask includes VK_ACCESS_2_VIDEO_DECODE_WRITE_BIT_KHR, srcStageMask **must** include VK_PIPELINE_STAGE_2_VIDEO_DECODE_BIT_KHR
- VUID-VkBufferMemoryBarrier2-srcAccessMask-04860
  If srcAccessMask includes VK_ACCESS_2_VIDEO_ENCODE_READ_BIT_KHR, srcStageMask **must** include VK_PIPELINE_STAGE_2_VIDEO_ENCODE_BIT_KHR
- VUID-VkBufferMemoryBarrier2-srcAccessMask-04861
  If srcAccessMask includes VK_ACCESS_2_VIDEO_ENCODE_WRITE_BIT_KHR, srcStageMask **must** include VK_PIPELINE_STAGE_2_VIDEO_ENCODE_BIT_KHR
- VUID-VkBufferMemoryBarrier2-srcAccessMask-07455
  If srcAccessMask includes VK_ACCESS_2_OPTICAL_FLOW_READ_BIT_NV, srcStageMask **must** include VK_PIPELINE_STAGE_2_OPTICAL_FLOW_BIT_NV
- VUID-VkBufferMemoryBarrier2-srcAccessMask-07456
  If srcAccessMask includes VK_ACCESS_2_OPTICAL_FLOW_WRITE_BIT_NV, srcStageMask **must** include VK_PIPELINE_STAGE_2_OPTICAL_FLOW_BIT_NV
- VUID-VkBufferMemoryBarrier2-srcAccessMask-07457
  If srcAccessMask includes VK_ACCESS_2_MICROMAP_WRITE_BIT_EXT, srcStageMask **must** include

VK_PIPELINE_STAGE_2_MICROMAP_BUILD_BIT_EXT

- VUID-VkBufferMemoryBarrier2-srcAccessMask-07458
  If srcAccessMask includes VK_ACCESS_2_MICROMAP_READ_BIT_EXT, srcStageMask **must** include VK_PIPELINE_STAGE_2_MICROMAP_BUILD_BIT_EXT or VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_BUILD_BIT_KHR
- VUID-VkBufferMemoryBarrier2-srcAccessMask-08118
  If srcAccessMask includes VK_ACCESS_2_DESCRIPTOR_BUFFER_READ_BIT_EXT, srcStageMask **must** include VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT, or one of VK_PIPELINE_STAGE_*_SHADER_BIT stages
- VUID-VkBufferMemoryBarrier2-dstStageMask-03929
  If the geometryShader feature is not enabled, dstStageMask **must** not contain VK_PIPELINE_STAGE_2_GEOMETRY_SHADER_BIT
- VUID-VkBufferMemoryBarrier2-dstStageMask-03930
  If the tessellationShader feature is not enabled, dstStageMask **must** not contain VK_PIPELINE_STAGE_2_TESSELLATION_CONTROL_SHADER_BIT or VK_PIPELINE_STAGE_2_TESSELLATION_EVALUATION_SHADER_BIT
- VUID-VkBufferMemoryBarrier2-dstStageMask-03931
  If the conditionalRendering feature is not enabled, dstStageMask **must** not contain VK_PIPELINE_STAGE_2_CONDITIONAL_RENDERING_BIT_EXT
- VUID-VkBufferMemoryBarrier2-dstStageMask-03932
  If the fragmentDensityMap feature is not enabled, dstStageMask **must** not contain VK_PIPELINE_STAGE_2_FRAGMENT_DENSITY_PROCESS_BIT_EXT
- VUID-VkBufferMemoryBarrier2-dstStageMask-03933
  If the transformFeedback feature is not enabled, dstStageMask **must** not contain VK_PIPELINE_STAGE_2_TRANSFORM_FEEDBACK_BIT_EXT
- VUID-VkBufferMemoryBarrier2-dstStageMask-03934
  If the meshShader feature is not enabled, dstStageMask **must** not contain VK_PIPELINE_STAGE_2_MESH_SHADER_BIT_EXT
- VUID-VkBufferMemoryBarrier2-dstStageMask-03935
  If the taskShader feature is not enabled, dstStageMask **must** not contain VK_PIPELINE_STAGE_2_TASK_SHADER_BIT_EXT
- VUID-VkBufferMemoryBarrier2-dstStageMask-07316
  If neither the shadingRateImage or attachmentFragmentShadingRate are enabled, dstStageMask **must** not contain VK_PIPELINE_STAGE_2_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR
- VUID-VkBufferMemoryBarrier2-dstStageMask-04957
  If the subpassShading feature is not enabled, dstStageMask **must** not contain VK_PIPELINE_STAGE_2_SUBPASS_SHADER_BIT_HUAWEI
- VUID-VkBufferMemoryBarrier2-dstStageMask-04995
  If the invocationMask feature is not enabled, dstStageMask **must** not contain VK_PIPELINE_STAGE_2_INVOCATION_MASK_BIT_HUAWEI
- VUID-VkBufferMemoryBarrier2-dstStageMask-07946

If neither the VK_NV_ray_tracing extension or rayTracingPipeline feature are enabled, dstStageMask **must** not contain VK_PIPELINE_STAGE_2_RAY_TRACING_SHADER_BIT_KHR

- VUID-VkBufferMemoryBarrier2-dstAccessMask-03900
  If dstAccessMask includes VK_ACCESS_2_INDIRECT_COMMAND_READ_BIT, dstStageMask **must** include VK_PIPELINE_STAGE_2_DRAW_INDIRECT_BIT, VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_BUILD_BIT_KHR, VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkBufferMemoryBarrier2-dstAccessMask-03901
  If dstAccessMask includes VK_ACCESS_2_INDEX_READ_BIT, dstStageMask **must** include VK_PIPELINE_STAGE_2_INDEX_INPUT_BIT, VK_PIPELINE_STAGE_2_VERTEX_INPUT_BIT, VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkBufferMemoryBarrier2-dstAccessMask-03902
  If dstAccessMask includes VK_ACCESS_2_VERTEX_ATTRIBUTE_READ_BIT, dstStageMask **must** include VK_PIPELINE_STAGE_2_VERTEX_ATTRIBUTE_INPUT_BIT, VK_PIPELINE_STAGE_2_VERTEX_INPUT_BIT, VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkBufferMemoryBarrier2-dstAccessMask-03903
  If dstAccessMask includes VK_ACCESS_2_INPUT_ATTACHMENT_READ_BIT, dstStageMask **must** include VK_PIPELINE_STAGE_2_FRAGMENT_SHADER_BIT, VK_PIPELINE_STAGE_2_SUBPASS_SHADER_BIT_HUAWEI, VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkBufferMemoryBarrier2-dstAccessMask-03904
  If dstAccessMask includes VK_ACCESS_2_UNIFORM_READ_BIT, dstStageMask **must** include VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT, or one of the VK_PIPELINE_STAGE_*_SHADER_BIT stages
- VUID-VkBufferMemoryBarrier2-dstAccessMask-03905
  If dstAccessMask includes VK_ACCESS_2_SHADER_SAMPLED_READ_BIT, dstStageMask **must** include VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT, or one of the VK_PIPELINE_STAGE_*_SHADER_BIT stages
- VUID-VkBufferMemoryBarrier2-dstAccessMask-03906
  If dstAccessMask includes VK_ACCESS_2_SHADER_STORAGE_READ_BIT, dstStageMask **must** include VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT, or one of the VK_PIPELINE_STAGE_*_SHADER_BIT stages
- VUID-VkBufferMemoryBarrier2-dstAccessMask-03907
  If dstAccessMask includes VK_ACCESS_2_SHADER_STORAGE_WRITE_BIT, dstStageMask **must** include VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT, or one of the VK_PIPELINE_STAGE_*_SHADER_BIT stages
- VUID-VkBufferMemoryBarrier2-dstAccessMask-07454
  If dstAccessMask includes VK_ACCESS_2_SHADER_READ_BIT, dstStageMask **must** include VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT, VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_BUILD_BIT_KHR, VK_PIPELINE_STAGE_2_MICROMAP_BUILD_BIT_EXT, or one of the VK_PIPELINE_STAGE_*_SHADER_BIT stages

- VUID-VkBufferMemoryBarrier2-dstAccessMask-03909
  If dstAccessMask includes VK_ACCESS_2_SHADER_WRITE_BIT, dstStageMask **must** include VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT, or one of the VK_PIPELINE_STAGE_*_SHADER_BIT stages
- VUID-VkBufferMemoryBarrier2-dstAccessMask-03910
  If dstAccessMask includes VK_ACCESS_2_COLOR_ATTACHMENT_READ_BIT, dstStageMask **must** include VK_PIPELINE_STAGE_2_COLOR_ATTACHMENT_OUTPUT_BIT VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkBufferMemoryBarrier2-dstAccessMask-03911
  If dstAccessMask includes VK_ACCESS_2_COLOR_ATTACHMENT_WRITE_BIT, dstStageMask **must** include VK_PIPELINE_STAGE_2_COLOR_ATTACHMENT_OUTPUT_BIT VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkBufferMemoryBarrier2-dstAccessMask-03912
  If dstAccessMask includes VK_ACCESS_2_DEPTH_STENCIL_ATTACHMENT_READ_BIT, dstStageMask **must** include VK_PIPELINE_STAGE_2_EARLY_FRAGMENT_TESTS_BIT, VK_PIPELINE_STAGE_2_LATE_FRAGMENT_TESTS_BIT, VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkBufferMemoryBarrier2-dstAccessMask-03913
  If dstAccessMask includes VK_ACCESS_2_DEPTH_STENCIL_ATTACHMENT_WRITE_BIT, dstStageMask **must** include VK_PIPELINE_STAGE_2_EARLY_FRAGMENT_TESTS_BIT, VK_PIPELINE_STAGE_2_LATE_FRAGMENT_TESTS_BIT, VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkBufferMemoryBarrier2-dstAccessMask-03914
  If dstAccessMask includes VK_ACCESS_2_TRANSFER_READ_BIT, dstStageMask **must** include VK_PIPELINE_STAGE_2_COPY_BIT, VK_PIPELINE_STAGE_2_BLIT_BIT, VK_PIPELINE_STAGE_2_RESOLVE_BIT, VK_PIPELINE_STAGE_2_ALL_TRANSFER_BIT, VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_BUILD_BIT_KHR, VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_COPY_BIT_KHR, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkBufferMemoryBarrier2-dstAccessMask-03915
  If dstAccessMask includes VK_ACCESS_2_TRANSFER_WRITE_BIT, dstStageMask **must** include VK_PIPELINE_STAGE_2_COPY_BIT, VK_PIPELINE_STAGE_2_BLIT_BIT, VK_PIPELINE_STAGE_2_RESOLVE_BIT, VK_PIPELINE_STAGE_2_CLEAR_BIT, VK_PIPELINE_STAGE_2_ALL_TRANSFER_BIT, VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_BUILD_BIT_KHR, or VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_COPY_BIT_KHR, VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkBufferMemoryBarrier2-dstAccessMask-03916
  If dstAccessMask includes VK_ACCESS_2_HOST_READ_BIT, dstStageMask **must** include VK_PIPELINE_STAGE_2_HOST_BIT
- VUID-VkBufferMemoryBarrier2-dstAccessMask-03917
  If dstAccessMask includes VK_ACCESS_2_HOST_WRITE_BIT, dstStageMask **must** include VK_PIPELINE_STAGE_2_HOST_BIT
- VUID-VkBufferMemoryBarrier2-dstAccessMask-03918

If dstAccessMask includes VK_ACCESS_2_CONDITIONAL_RENDERING_READ_BIT_EXT, dstStageMask **must** include VK_PIPELINE_STAGE_2_CONDITIONAL_RENDERING_BIT_EXT, VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT

- VUID-VkBufferMemoryBarrier2-dstAccessMask-03919
  If dstAccessMask includes VK_ACCESS_2_FRAGMENT_DENSITY_MAP_READ_BIT_EXT, dstStageMask **must** include VK_PIPELINE_STAGE_2_FRAGMENT_DENSITY_PROCESS_BIT_EXT, VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkBufferMemoryBarrier2-dstAccessMask-03920
  If dstAccessMask includes VK_ACCESS_2_TRANSFORM_FEEDBACK_WRITE_BIT_EXT, dstStageMask **must** include VK_PIPELINE_STAGE_2_TRANSFORM_FEEDBACK_BIT_EXT, VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkBufferMemoryBarrier2-dstAccessMask-04747
  If dstAccessMask includes VK_ACCESS_2_TRANSFORM_FEEDBACK_COUNTER_READ_BIT_EXT, dstStageMask **must** include VK_PIPELINE_STAGE_2_DRAW_INDIRECT_BIT, VK_PIPELINE_STAGE_2_TRANSFORM_FEEDBACK_BIT_EXT, VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkBufferMemoryBarrier2-dstAccessMask-03922
  If dstAccessMask includes VK_ACCESS_2_TRANSFORM_FEEDBACK_COUNTER_WRITE_BIT_EXT, dstStageMask **must** include VK_PIPELINE_STAGE_2_TRANSFORM_FEEDBACK_BIT_EXT, VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkBufferMemoryBarrier2-dstAccessMask-03923
  If dstAccessMask includes VK_ACCESS_2_SHADING_RATE_IMAGE_READ_BIT_NV, dstStageMask **must** include VK_PIPELINE_STAGE_2_SHADING_RATE_IMAGE_BIT_NV, VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkBufferMemoryBarrier2-dstAccessMask-04994
  If dstAccessMask includes VK_ACCESS_2_INVOCATION_MASK_READ_BIT_HUAWEI, dstStageMask **must** include VK_PIPELINE_STAGE_2_INVOCATION_MASK_BIT_HUAWEI
- VUID-VkBufferMemoryBarrier2-dstAccessMask-03924
  If dstAccessMask includes VK_ACCESS_2_COMMAND_PREPROCESS_READ_BIT_NV, dstStageMask **must** include VK_PIPELINE_STAGE_2_COMMAND_PREPROCESS_BIT_NV or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkBufferMemoryBarrier2-dstAccessMask-03925
  If dstAccessMask includes VK_ACCESS_2_COMMAND_PREPROCESS_WRITE_BIT_NV, dstStageMask **must** include VK_PIPELINE_STAGE_2_COMMAND_PREPROCESS_BIT_NV or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkBufferMemoryBarrier2-dstAccessMask-03926
  If dstAccessMask includes VK_ACCESS_2_COLOR_ATTACHMENT_READ_NONCOHERENT_BIT_EXT, dstStageMask **must** include VK_PIPELINE_STAGE_2_COLOR_ATTACHMENT_OUTPUT_BIT VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkBufferMemoryBarrier2-dstAccessMask-03927
  If dstAccessMask includes VK_ACCESS_2_ACCELERATION_STRUCTURE_READ_BIT_KHR, dstStageMask **must** include VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_BUILD_BIT_KHR, VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_COPY_BIT_KHR, VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT, or one of the VK_PIPELINE_STAGE_*_SHADER_BIT

stages

- VUID-VkBufferMemoryBarrier2-dstAccessMask-03928
  If `dstAccessMask` includes `VK_ACCESS_2_ACCELERATION_STRUCTURE_WRITE_BIT_KHR`, `dstStageMask` **must** include `VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_COPY_BIT_KHR`, `VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_BUILD_BIT_KHR` or `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`
- VUID-VkBufferMemoryBarrier2-dstAccessMask-06256
  If the `rayQuery` feature is not enabled and `dstAccessMask` includes `VK_ACCESS_2_ACCELERATION_STRUCTURE_READ_BIT_KHR`, `dstStageMask` **must** not include any of the `VK_PIPELINE_STAGE_*_SHADER_BIT` stages except `VK_PIPELINE_STAGE_2_RAY_TRACING_SHADER_BIT_KHR`
- VUID-VkBufferMemoryBarrier2-dstAccessMask-07272
  If `dstAccessMask` includes `VK_ACCESS_2_SHADER_BINDING_TABLE_READ_BIT_KHR`, `dstStageMask` **must** include `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT` or `VK_PIPELINE_STAGE_2_RAY_TRACING_SHADER_BIT_KHR`
- VUID-VkBufferMemoryBarrier2-dstAccessMask-04858
  If `dstAccessMask` includes `VK_ACCESS_2_VIDEO_DECODE_READ_BIT_KHR`, `dstStageMask` **must** include `VK_PIPELINE_STAGE_2_VIDEO_DECODE_BIT_KHR`
- VUID-VkBufferMemoryBarrier2-dstAccessMask-04859
  If `dstAccessMask` includes `VK_ACCESS_2_VIDEO_DECODE_WRITE_BIT_KHR`, `dstStageMask` **must** include `VK_PIPELINE_STAGE_2_VIDEO_DECODE_BIT_KHR`
- VUID-VkBufferMemoryBarrier2-dstAccessMask-04860
  If `dstAccessMask` includes `VK_ACCESS_2_VIDEO_ENCODE_READ_BIT_KHR`, `dstStageMask` **must** include `VK_PIPELINE_STAGE_2_VIDEO_ENCODE_BIT_KHR`
- VUID-VkBufferMemoryBarrier2-dstAccessMask-04861
  If `dstAccessMask` includes `VK_ACCESS_2_VIDEO_ENCODE_WRITE_BIT_KHR`, `dstStageMask` **must** include `VK_PIPELINE_STAGE_2_VIDEO_ENCODE_BIT_KHR`
- VUID-VkBufferMemoryBarrier2-dstAccessMask-07455
  If `dstAccessMask` includes `VK_ACCESS_2_OPTICAL_FLOW_READ_BIT_NV`, `dstStageMask` **must** include `VK_PIPELINE_STAGE_2_OPTICAL_FLOW_BIT_NV`
- VUID-VkBufferMemoryBarrier2-dstAccessMask-07456
  If `dstAccessMask` includes `VK_ACCESS_2_OPTICAL_FLOW_WRITE_BIT_NV`, `dstStageMask` **must** include `VK_PIPELINE_STAGE_2_OPTICAL_FLOW_BIT_NV`
- VUID-VkBufferMemoryBarrier2-dstAccessMask-07457
  If `dstAccessMask` includes `VK_ACCESS_2_MICROMAP_WRITE_BIT_EXT`, `dstStageMask` **must** include `VK_PIPELINE_STAGE_2_MICROMAP_BUILD_BIT_EXT`
- VUID-VkBufferMemoryBarrier2-dstAccessMask-07458
  If `dstAccessMask` includes `VK_ACCESS_2_MICROMAP_READ_BIT_EXT`, `dstStageMask` **must** include `VK_PIPELINE_STAGE_2_MICROMAP_BUILD_BIT_EXT` or `VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_BUILD_BIT_KHR`
- VUID-VkBufferMemoryBarrier2-dstAccessMask-08118
  If `dstAccessMask` includes `VK_ACCESS_2_DESCRIPTOR_BUFFER_READ_BIT_EXT`, `dstStageMask` **must** include `VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT`, `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`, or

one of VK_PIPELINE_STAGE_*_SHADER_BIT stages

- VUID-VkBufferMemoryBarrier2-offset-01187
  offset **must** be less than the size of buffer
- VUID-VkBufferMemoryBarrier2-size-01188
  If size is not equal to VK_WHOLE_SIZE, size **must** be greater than 0
- VUID-VkBufferMemoryBarrier2-size-01189
  If size is not equal to VK_WHOLE_SIZE, size **must** be less than or equal to than the size of buffer minus offset
- VUID-VkBufferMemoryBarrier2-buffer-01931
  If buffer is non-sparse then it **must** be bound completely and contiguously to a single VkDeviceMemory object
- VUID-VkBufferMemoryBarrier2-buffer-09095
  If buffer was created with a sharing mode of VK_SHARING_MODE_EXCLUSIVE, and srcQueueFamilyIndex and dstQueueFamilyIndex are not equal, srcQueueFamilyIndex **must** be VK_QUEUE_FAMILY_EXTERNAL, VK_QUEUE_FAMILY_FOREIGN_EXT, or a valid queue family
- VUID-VkBufferMemoryBarrier2-buffer-09096
  If buffer was created with a sharing mode of VK_SHARING_MODE_EXCLUSIVE, and srcQueueFamilyIndex and dstQueueFamilyIndex are not equal, dstQueueFamilyIndex **must** be VK_QUEUE_FAMILY_EXTERNAL, VK_QUEUE_FAMILY_FOREIGN_EXT, or a valid queue family
- VUID-VkBufferMemoryBarrier2-srcQueueFamilyIndex-04087
  If srcQueueFamilyIndex is not equal to dstQueueFamilyIndex, at least one of srcQueueFamilyIndex or dstQueueFamilyIndex **must** not be VK_QUEUE_FAMILY_EXTERNAL or VK_QUEUE_FAMILY_FOREIGN_EXT
- VUID-VkBufferMemoryBarrier2-None-09097
  If the VK_KHR_external_memory extension is not enabled, and the value of VkApplicationInfo::apiVersion used to create the VkInstance is not greater than or equal to Version 1.1, srcQueueFamilyIndex **must** not be VK_QUEUE_FAMILY_EXTERNAL
- VUID-VkBufferMemoryBarrier2-None-09098
  If the VK_KHR_external_memory extension is not enabled, and the value of VkApplicationInfo::apiVersion used to create the VkInstance is not greater than or equal to Version 1.1, dstQueueFamilyIndex **must** not be VK_QUEUE_FAMILY_EXTERNAL
- VUID-VkBufferMemoryBarrier2-srcQueueFamilyIndex-09099
  If the VK_EXT_queue_family_foreign extension is not enabled srcQueueFamilyIndex **must** not be VK_QUEUE_FAMILY_FOREIGN_EXT
- VUID-VkBufferMemoryBarrier2-dstQueueFamilyIndex-09100
  If the VK_EXT_queue_family_foreign extension is not enabled dstQueueFamilyIndex **must** not be VK_QUEUE_FAMILY_FOREIGN_EXT
- VUID-VkBufferMemoryBarrier2-srcStageMask-03851
  If either srcStageMask or dstStageMask includes VK_PIPELINE_STAGE_2_HOST_BIT, srcQueueFamilyIndex and dstQueueFamilyIndex **must** be equal

### Valid Usage (Implicit)

- VUID-VkBufferMemoryBarrier2-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_BUFFER_MEMORY_BARRIER_2`
- VUID-VkBufferMemoryBarrier2-pNext-pNext
  `pNext` **must** be `NULL` or a pointer to a valid instance of VkExternalMemoryAcquireUnmodifiedEXT
- VUID-VkBufferMemoryBarrier2-sType-unique
  The `sType` value of each struct in the `pNext` chain **must** be unique
- VUID-VkBufferMemoryBarrier2-srcStageMask-parameter
  `srcStageMask` **must** be a valid combination of VkPipelineStageFlagBits2 values
- VUID-VkBufferMemoryBarrier2-srcAccessMask-parameter
  `srcAccessMask` **must** be a valid combination of VkAccessFlagBits2 values
- VUID-VkBufferMemoryBarrier2-dstStageMask-parameter
  `dstStageMask` **must** be a valid combination of VkPipelineStageFlagBits2 values
- VUID-VkBufferMemoryBarrier2-dstAccessMask-parameter
  `dstAccessMask` **must** be a valid combination of VkAccessFlagBits2 values
- VUID-VkBufferMemoryBarrier2-buffer-parameter
  `buffer` **must** be a valid VkBuffer handle

The `VkBufferMemoryBarrier` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkBufferMemoryBarrier {
    VkStructureType    sType;
    const void*        pNext;
    VkAccessFlags      srcAccessMask;
    VkAccessFlags      dstAccessMask;
    uint32_t           srcQueueFamilyIndex;
    uint32_t           dstQueueFamilyIndex;
    VkBuffer           buffer;
    VkDeviceSize       offset;
    VkDeviceSize       size;
} VkBufferMemoryBarrier;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `srcAccessMask` is a bitmask of VkAccessFlagBits specifying a source access mask.
- `dstAccessMask` is a bitmask of VkAccessFlagBits specifying a destination access mask.
- `srcQueueFamilyIndex` is the source queue family for a queue family ownership transfer.
- `dstQueueFamilyIndex` is the destination queue family for a queue family ownership transfer.

- `buffer` is a handle to the buffer whose backing memory is affected by the barrier.
- `offset` is an offset in bytes into the backing memory for `buffer`; this is relative to the base offset as bound to the buffer (see vkBindBufferMemory).
- `size` is a size in bytes of the affected area of backing memory for `buffer`, or `VK_WHOLE_SIZE` to use the range from `offset` to the end of the buffer.

The first access scope is limited to access to memory through the specified buffer range, via access types in the source access mask specified by `srcAccessMask`. If `srcAccessMask` includes `VK_ACCESS_HOST_WRITE_BIT`, a memory domain operation is performed where available memory in the host domain is also made available to the device domain.

The second access scope is limited to access to memory through the specified buffer range, via access types in the destination access mask specified by `dstAccessMask`. If `dstAccessMask` includes `VK_ACCESS_HOST_WRITE_BIT` or `VK_ACCESS_HOST_READ_BIT`, a memory domain operation is performed where available memory in the device domain is also made available to the host domain.

**NOTE** When `VK_MEMORY_PROPERTY_HOST_COHERENT_BIT` is used, available memory in host domain is automatically made visible to host domain, and any host write is automatically made available to host domain.

If `srcQueueFamilyIndex` is not equal to `dstQueueFamilyIndex`, and `srcQueueFamilyIndex` is equal to the current queue family, then the memory barrier defines a queue family release operation for the specified buffer range, and the second synchronization scope of the calling command does not apply to this operation.

If `dstQueueFamilyIndex` is not equal to `srcQueueFamilyIndex`, and `dstQueueFamilyIndex` is equal to the current queue family, then the memory barrier defines a queue family acquire operation for the specified buffer range, and the first synchronization scope of the calling command does not apply to this operation.

### Valid Usage

- VUID-VkBufferMemoryBarrier-offset-01187
  `offset` **must** be less than the size of `buffer`
- VUID-VkBufferMemoryBarrier-size-01188
  If `size` is not equal to `VK_WHOLE_SIZE`, `size` **must** be greater than `0`
- VUID-VkBufferMemoryBarrier-size-01189
  If `size` is not equal to `VK_WHOLE_SIZE`, `size` **must** be less than or equal to than the size of `buffer` minus `offset`
- VUID-VkBufferMemoryBarrier-buffer-01931
  If `buffer` is non-sparse then it **must** be bound completely and contiguously to a single `VkDeviceMemory` object
- VUID-VkBufferMemoryBarrier-buffer-09095
  If `buffer` was created with a sharing mode of `VK_SHARING_MODE_EXCLUSIVE`, and `srcQueueFamilyIndex` and `dstQueueFamilyIndex` are not equal, `srcQueueFamilyIndex` **must** be `VK_QUEUE_FAMILY_EXTERNAL`, `VK_QUEUE_FAMILY_FOREIGN_EXT`, or a valid queue family

- VUID-VkBufferMemoryBarrier-buffer-09096
  If `buffer` was created with a sharing mode of `VK_SHARING_MODE_EXCLUSIVE`, and `srcQueueFamilyIndex` and `dstQueueFamilyIndex` are not equal, `dstQueueFamilyIndex` **must** be `VK_QUEUE_FAMILY_EXTERNAL`, `VK_QUEUE_FAMILY_FOREIGN_EXT`, or a valid queue family
- VUID-VkBufferMemoryBarrier-srcQueueFamilyIndex-04087
  If `srcQueueFamilyIndex` is not equal to `dstQueueFamilyIndex`, at least one of `srcQueueFamilyIndex` or `dstQueueFamilyIndex` **must** not be `VK_QUEUE_FAMILY_EXTERNAL` or `VK_QUEUE_FAMILY_FOREIGN_EXT`
- VUID-VkBufferMemoryBarrier-None-09097
  If the VK_KHR_external_memory extension is not enabled, and the value of VkApplicationInfo::`apiVersion` used to create the VkInstance is not greater than or equal to Version 1.1, `srcQueueFamilyIndex` **must** not be `VK_QUEUE_FAMILY_EXTERNAL`
- VUID-VkBufferMemoryBarrier-None-09098
  If the VK_KHR_external_memory extension is not enabled, and the value of VkApplicationInfo::`apiVersion` used to create the VkInstance is not greater than or equal to Version 1.1, `dstQueueFamilyIndex` **must** not be `VK_QUEUE_FAMILY_EXTERNAL`
- VUID-VkBufferMemoryBarrier-srcQueueFamilyIndex-09099
  If the VK_EXT_queue_family_foreign extension is not enabled `srcQueueFamilyIndex` **must** not be `VK_QUEUE_FAMILY_FOREIGN_EXT`
- VUID-VkBufferMemoryBarrier-dstQueueFamilyIndex-09100
  If the VK_EXT_queue_family_foreign extension is not enabled `dstQueueFamilyIndex` **must** not be `VK_QUEUE_FAMILY_FOREIGN_EXT`
- VUID-VkBufferMemoryBarrier-None-09049
  If the `synchronization2` feature is not enabled, and `buffer` was created with a sharing mode of `VK_SHARING_MODE_CONCURRENT`, at least one of `srcQueueFamilyIndex` and `dstQueueFamilyIndex` **must** be `VK_QUEUE_FAMILY_IGNORED`
- VUID-VkBufferMemoryBarrier-None-09050
  If the `synchronization2` feature is not enabled, and `buffer` was created with a sharing mode of `VK_SHARING_MODE_CONCURRENT`, `srcQueueFamilyIndex` **must** be `VK_QUEUE_FAMILY_IGNORED` or `VK_QUEUE_FAMILY_EXTERNAL`
- VUID-VkBufferMemoryBarrier-None-09051
  If the `synchronization2` feature is not enabled, and `buffer` was created with a sharing mode of `VK_SHARING_MODE_CONCURRENT`, `dstQueueFamilyIndex` **must** be `VK_QUEUE_FAMILY_IGNORED` or `VK_QUEUE_FAMILY_EXTERNAL`

## Valid Usage (Implicit)

- VUID-VkBufferMemoryBarrier-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_BUFFER_MEMORY_BARRIER`
- VUID-VkBufferMemoryBarrier-pNext-pNext
  `pNext` **must** be `NULL` or a pointer to a valid instance of VkExternalMemoryAcquireUnmodifiedEXT
- VUID-VkBufferMemoryBarrier-sType-unique

The sType value of each struct in the pNext chain **must** be unique

- VUID-VkBufferMemoryBarrier-buffer-parameter
  buffer **must** be a valid VkBuffer handle

VK_WHOLE_SIZE is a special value indicating that the entire remaining length of a buffer following a given offset should be used. It **can** be specified for VkBufferMemoryBarrier::size and other structures.

```
#define VK_WHOLE_SIZE                    (~0ULL)
```

### 7.7.3. Image Memory Barriers

Image memory barriers only apply to memory accesses involving a specific image subresource range. That is, a memory dependency formed from an image memory barrier is scoped to access via the specified image subresource range. Image memory barriers **can** also be used to define image layout transitions or a queue family ownership transfer for the specified image subresource range.

The VkImageMemoryBarrier2 structure is defined as:

```
typedef struct VkImageMemoryBarrier2 {
    VkStructureType            sType;
    const void*                pNext;
    VkPipelineStageFlags2      srcStageMask;
    VkAccessFlags2             srcAccessMask;
    VkPipelineStageFlags2      dstStageMask;
    VkAccessFlags2             dstAccessMask;
    VkImageLayout              oldLayout;
    VkImageLayout              newLayout;
    uint32_t                   srcQueueFamilyIndex;
    uint32_t                   dstQueueFamilyIndex;
    VkImage                    image;
    VkImageSubresourceRange    subresourceRange;
} VkImageMemoryBarrier2;
```

or the equivalent

```
// Provided by VK_KHR_synchronization2
typedef VkImageMemoryBarrier2 VkImageMemoryBarrier2KHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- srcStageMask is a VkPipelineStageFlags2 mask of pipeline stages to be included in the first synchronization scope.

- srcAccessMask is a VkAccessFlags2 mask of access flags to be included in the first access scope.
- dstStageMask is a VkPipelineStageFlags2 mask of pipeline stages to be included in the second synchronization scope.
- dstAccessMask is a VkAccessFlags2 mask of access flags to be included in the second access scope.
- oldLayout is the old layout in an image layout transition.
- newLayout is the new layout in an image layout transition.
- srcQueueFamilyIndex is the source queue family for a queue family ownership transfer.
- dstQueueFamilyIndex is the destination queue family for a queue family ownership transfer.
- image is a handle to the image affected by this barrier.
- subresourceRange describes the image subresource range within image that is affected by this barrier.

This structure defines a memory dependency limited to an image subresource range, and **can** define a queue family ownership transfer operation and image layout transition for that subresource range.

The first synchronization scope and access scope described by this structure include only operations and memory accesses specified by srcStageMask and srcAccessMask.

The second synchronization scope and access scope described by this structure include only operations and memory accesses specified by dstStageMask and dstAccessMask.

Both access scopes are limited to only memory accesses to image in the subresource range defined by subresourceRange.

If image was created with VK_SHARING_MODE_EXCLUSIVE, and srcQueueFamilyIndex is not equal to dstQueueFamilyIndex, this memory barrier defines a queue family ownership transfer operation. When executed on a queue in the family identified by srcQueueFamilyIndex, this barrier defines a queue family release operation for the specified image subresource range, and the second synchronization scope does not apply to this operation. When executed on a queue in the family identified by dstQueueFamilyIndex, this barrier defines a queue family acquire operation for the specified image subresource range, and the first synchronization, the first synchronization scope does not apply to this operation.

A queue family ownership transfer operation is also defined if the values are not equal, and either is one of the special queue family values reserved for external memory ownership transfers, as described in Queue Family Ownership Transfer. A queue family release operation is defined when dstQueueFamilyIndex is one of those values, and a queue family acquire operation is defined when srcQueueFamilyIndex is one of those values.

If oldLayout is not equal to newLayout, then the memory barrier defines an image layout transition for the specified image subresource range. If this memory barrier defines a queue family ownership transfer operation, the layout transition is only executed once between the queues.

| | |
|---|---|
| **NOTE** | When the old and new layout are equal, the layout values are ignored - data is preserved no matter what values are specified, or what layout the image is |

currently in.

If image has a multi-planar format and the image is *disjoint*, then including VK_IMAGE_ASPECT_COLOR_BIT in the aspectMask member of subresourceRange is equivalent to including VK_IMAGE_ASPECT_PLANE_0_BIT, VK_IMAGE_ASPECT_PLANE_1_BIT, and (for three-plane formats only) VK_IMAGE_ASPECT_PLANE_2_BIT.

## Valid Usage

- VUID-VkImageMemoryBarrier2-srcStageMask-03929
  If the geometryShader feature is not enabled, srcStageMask **must** not contain VK_PIPELINE_STAGE_2_GEOMETRY_SHADER_BIT
- VUID-VkImageMemoryBarrier2-srcStageMask-03930
  If the tessellationShader feature is not enabled, srcStageMask **must** not contain VK_PIPELINE_STAGE_2_TESSELLATION_CONTROL_SHADER_BIT or VK_PIPELINE_STAGE_2_TESSELLATION_EVALUATION_SHADER_BIT
- VUID-VkImageMemoryBarrier2-srcStageMask-03931
  If the conditionalRendering feature is not enabled, srcStageMask **must** not contain VK_PIPELINE_STAGE_2_CONDITIONAL_RENDERING_BIT_EXT
- VUID-VkImageMemoryBarrier2-srcStageMask-03932
  If the fragmentDensityMap feature is not enabled, srcStageMask **must** not contain VK_PIPELINE_STAGE_2_FRAGMENT_DENSITY_PROCESS_BIT_EXT
- VUID-VkImageMemoryBarrier2-srcStageMask-03933
  If the transformFeedback feature is not enabled, srcStageMask **must** not contain VK_PIPELINE_STAGE_2_TRANSFORM_FEEDBACK_BIT_EXT
- VUID-VkImageMemoryBarrier2-srcStageMask-03934
  If the meshShader feature is not enabled, srcStageMask **must** not contain VK_PIPELINE_STAGE_2_MESH_SHADER_BIT_EXT
- VUID-VkImageMemoryBarrier2-srcStageMask-03935
  If the taskShader feature is not enabled, srcStageMask **must** not contain VK_PIPELINE_STAGE_2_TASK_SHADER_BIT_EXT
- VUID-VkImageMemoryBarrier2-srcStageMask-07316
  If neither the shadingRateImage or attachmentFragmentShadingRate are enabled, srcStageMask **must** not contain VK_PIPELINE_STAGE_2_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR
- VUID-VkImageMemoryBarrier2-srcStageMask-04957
  If the subpassShading feature is not enabled, srcStageMask **must** not contain VK_PIPELINE_STAGE_2_SUBPASS_SHADER_BIT_HUAWEI
- VUID-VkImageMemoryBarrier2-srcStageMask-04995
  If the invocationMask feature is not enabled, srcStageMask **must** not contain VK_PIPELINE_STAGE_2_INVOCATION_MASK_BIT_HUAWEI
- VUID-VkImageMemoryBarrier2-srcStageMask-07946
  If neither the VK_NV_ray_tracing extension or rayTracingPipeline feature are enabled,

srcStageMask **must** not contain VK_PIPELINE_STAGE_2_RAY_TRACING_SHADER_BIT_KHR

- VUID-VkImageMemoryBarrier2-srcAccessMask-03900
  If srcAccessMask includes VK_ACCESS_2_INDIRECT_COMMAND_READ_BIT, srcStageMask **must** include VK_PIPELINE_STAGE_2_DRAW_INDIRECT_BIT, VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_BUILD_BIT_KHR, VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkImageMemoryBarrier2-srcAccessMask-03901
  If srcAccessMask includes VK_ACCESS_2_INDEX_READ_BIT, srcStageMask **must** include VK_PIPELINE_STAGE_2_INDEX_INPUT_BIT, VK_PIPELINE_STAGE_2_VERTEX_INPUT_BIT, VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkImageMemoryBarrier2-srcAccessMask-03902
  If srcAccessMask includes VK_ACCESS_2_VERTEX_ATTRIBUTE_READ_BIT, srcStageMask **must** include VK_PIPELINE_STAGE_2_VERTEX_ATTRIBUTE_INPUT_BIT, VK_PIPELINE_STAGE_2_VERTEX_INPUT_BIT, VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkImageMemoryBarrier2-srcAccessMask-03903
  If srcAccessMask includes VK_ACCESS_2_INPUT_ATTACHMENT_READ_BIT, srcStageMask **must** include VK_PIPELINE_STAGE_2_FRAGMENT_SHADER_BIT, VK_PIPELINE_STAGE_2_SUBPASS_SHADER_BIT_HUAWEI, VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkImageMemoryBarrier2-srcAccessMask-03904
  If srcAccessMask includes VK_ACCESS_2_UNIFORM_READ_BIT, srcStageMask **must** include VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT, or one of the VK_PIPELINE_STAGE_*_SHADER_BIT stages
- VUID-VkImageMemoryBarrier2-srcAccessMask-03905
  If srcAccessMask includes VK_ACCESS_2_SHADER_SAMPLED_READ_BIT, srcStageMask **must** include VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT, or one of the VK_PIPELINE_STAGE_*_SHADER_BIT stages
- VUID-VkImageMemoryBarrier2-srcAccessMask-03906
  If srcAccessMask includes VK_ACCESS_2_SHADER_STORAGE_READ_BIT, srcStageMask **must** include VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT, or one of the VK_PIPELINE_STAGE_*_SHADER_BIT stages
- VUID-VkImageMemoryBarrier2-srcAccessMask-03907
  If srcAccessMask includes VK_ACCESS_2_SHADER_STORAGE_WRITE_BIT, srcStageMask **must** include VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT, or one of the VK_PIPELINE_STAGE_*_SHADER_BIT stages
- VUID-VkImageMemoryBarrier2-srcAccessMask-07454
  If srcAccessMask includes VK_ACCESS_2_SHADER_READ_BIT, srcStageMask **must** include VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT, VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_BUILD_BIT_KHR, VK_PIPELINE_STAGE_2_MICROMAP_BUILD_BIT_EXT, or one of the VK_PIPELINE_STAGE_*_SHADER_BIT stages
- VUID-VkImageMemoryBarrier2-srcAccessMask-03909

If srcAccessMask includes VK_ACCESS_2_SHADER_WRITE_BIT, srcStageMask **must** include VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT, or one of the VK_PIPELINE_STAGE_*_SHADER_BIT stages

- VUID-VkImageMemoryBarrier2-srcAccessMask-03910
  If srcAccessMask includes VK_ACCESS_2_COLOR_ATTACHMENT_READ_BIT, srcStageMask **must** include VK_PIPELINE_STAGE_2_COLOR_ATTACHMENT_OUTPUT_BIT VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkImageMemoryBarrier2-srcAccessMask-03911
  If srcAccessMask includes VK_ACCESS_2_COLOR_ATTACHMENT_WRITE_BIT, srcStageMask **must** include VK_PIPELINE_STAGE_2_COLOR_ATTACHMENT_OUTPUT_BIT VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkImageMemoryBarrier2-srcAccessMask-03912
  If srcAccessMask includes VK_ACCESS_2_DEPTH_STENCIL_ATTACHMENT_READ_BIT, srcStageMask **must** include VK_PIPELINE_STAGE_2_EARLY_FRAGMENT_TESTS_BIT, VK_PIPELINE_STAGE_2_LATE_FRAGMENT_TESTS_BIT, VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkImageMemoryBarrier2-srcAccessMask-03913
  If srcAccessMask includes VK_ACCESS_2_DEPTH_STENCIL_ATTACHMENT_WRITE_BIT, srcStageMask **must** include VK_PIPELINE_STAGE_2_EARLY_FRAGMENT_TESTS_BIT, VK_PIPELINE_STAGE_2_LATE_FRAGMENT_TESTS_BIT, VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkImageMemoryBarrier2-srcAccessMask-03914
  If srcAccessMask includes VK_ACCESS_2_TRANSFER_READ_BIT, srcStageMask **must** include VK_PIPELINE_STAGE_2_COPY_BIT, VK_PIPELINE_STAGE_2_BLIT_BIT, VK_PIPELINE_STAGE_2_RESOLVE_BIT, VK_PIPELINE_STAGE_2_ALL_TRANSFER_BIT, VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_BUILD_BIT_KHR, VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_COPY_BIT_KHR, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkImageMemoryBarrier2-srcAccessMask-03915
  If srcAccessMask includes VK_ACCESS_2_TRANSFER_WRITE_BIT, srcStageMask **must** include VK_PIPELINE_STAGE_2_COPY_BIT, VK_PIPELINE_STAGE_2_BLIT_BIT, VK_PIPELINE_STAGE_2_RESOLVE_BIT, VK_PIPELINE_STAGE_2_CLEAR_BIT, VK_PIPELINE_STAGE_2_ALL_TRANSFER_BIT, VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_BUILD_BIT_KHR, or VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_COPY_BIT_KHR, VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkImageMemoryBarrier2-srcAccessMask-03916
  If srcAccessMask includes VK_ACCESS_2_HOST_READ_BIT, srcStageMask **must** include VK_PIPELINE_STAGE_2_HOST_BIT
- VUID-VkImageMemoryBarrier2-srcAccessMask-03917
  If srcAccessMask includes VK_ACCESS_2_HOST_WRITE_BIT, srcStageMask **must** include VK_PIPELINE_STAGE_2_HOST_BIT
- VUID-VkImageMemoryBarrier2-srcAccessMask-03918
  If srcAccessMask includes VK_ACCESS_2_CONDITIONAL_RENDERING_READ_BIT_EXT, srcStageMask

**must** include VK_PIPELINE_STAGE_2_CONDITIONAL_RENDERING_BIT_EXT, VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT

- VUID-VkImageMemoryBarrier2-srcAccessMask-03919
  If srcAccessMask includes VK_ACCESS_2_FRAGMENT_DENSITY_MAP_READ_BIT_EXT, srcStageMask **must** include VK_PIPELINE_STAGE_2_FRAGMENT_DENSITY_PROCESS_BIT_EXT, VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkImageMemoryBarrier2-srcAccessMask-03920
  If srcAccessMask includes VK_ACCESS_2_TRANSFORM_FEEDBACK_WRITE_BIT_EXT, srcStageMask **must** include VK_PIPELINE_STAGE_2_TRANSFORM_FEEDBACK_BIT_EXT, VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkImageMemoryBarrier2-srcAccessMask-04747
  If srcAccessMask includes VK_ACCESS_2_TRANSFORM_FEEDBACK_COUNTER_READ_BIT_EXT, srcStageMask **must** include VK_PIPELINE_STAGE_2_DRAW_INDIRECT_BIT, VK_PIPELINE_STAGE_2_TRANSFORM_FEEDBACK_BIT_EXT, VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkImageMemoryBarrier2-srcAccessMask-03922
  If srcAccessMask includes VK_ACCESS_2_TRANSFORM_FEEDBACK_COUNTER_WRITE_BIT_EXT, srcStageMask **must** include VK_PIPELINE_STAGE_2_TRANSFORM_FEEDBACK_BIT_EXT, VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkImageMemoryBarrier2-srcAccessMask-03923
  If srcAccessMask includes VK_ACCESS_2_SHADING_RATE_IMAGE_READ_BIT_NV, srcStageMask **must** include VK_PIPELINE_STAGE_2_SHADING_RATE_IMAGE_BIT_NV, VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkImageMemoryBarrier2-srcAccessMask-04994
  If srcAccessMask includes VK_ACCESS_2_INVOCATION_MASK_READ_BIT_HUAWEI, srcStageMask **must** include VK_PIPELINE_STAGE_2_INVOCATION_MASK_BIT_HUAWEI
- VUID-VkImageMemoryBarrier2-srcAccessMask-03924
  If srcAccessMask includes VK_ACCESS_2_COMMAND_PREPROCESS_READ_BIT_NV, srcStageMask **must** include VK_PIPELINE_STAGE_2_COMMAND_PREPROCESS_BIT_NV or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkImageMemoryBarrier2-srcAccessMask-03925
  If srcAccessMask includes VK_ACCESS_2_COMMAND_PREPROCESS_WRITE_BIT_NV, srcStageMask **must** include VK_PIPELINE_STAGE_2_COMMAND_PREPROCESS_BIT_NV or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkImageMemoryBarrier2-srcAccessMask-03926
  If srcAccessMask includes VK_ACCESS_2_COLOR_ATTACHMENT_READ_NONCOHERENT_BIT_EXT, srcStageMask **must** include VK_PIPELINE_STAGE_2_COLOR_ATTACHMENT_OUTPUT_BIT VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkImageMemoryBarrier2-srcAccessMask-03927
  If srcAccessMask includes VK_ACCESS_2_ACCELERATION_STRUCTURE_READ_BIT_KHR, srcStageMask **must** include VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_BUILD_BIT_KHR, VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_COPY_BIT_KHR, VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT, or one of the VK_PIPELINE_STAGE_*_SHADER_BIT stages

- VUID-VkImageMemoryBarrier2-srcAccessMask-03928
  If srcAccessMask includes VK_ACCESS_2_ACCELERATION_STRUCTURE_WRITE_BIT_KHR, srcStageMask **must** include VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_COPY_BIT_KHR, VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_BUILD_BIT_KHR or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkImageMemoryBarrier2-srcAccessMask-06256
  If the rayQuery feature is not enabled and srcAccessMask includes VK_ACCESS_2_ACCELERATION_STRUCTURE_READ_BIT_KHR, srcStageMask **must** not include any of the VK_PIPELINE_STAGE_*_SHADER_BIT stages except VK_PIPELINE_STAGE_2_RAY_TRACING_SHADER_BIT_KHR
- VUID-VkImageMemoryBarrier2-srcAccessMask-07272
  If srcAccessMask includes VK_ACCESS_2_SHADER_BINDING_TABLE_READ_BIT_KHR, srcStageMask **must** include VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT or VK_PIPELINE_STAGE_2_RAY_TRACING_SHADER_BIT_KHR
- VUID-VkImageMemoryBarrier2-srcAccessMask-04858
  If srcAccessMask includes VK_ACCESS_2_VIDEO_DECODE_READ_BIT_KHR, srcStageMask **must** include VK_PIPELINE_STAGE_2_VIDEO_DECODE_BIT_KHR
- VUID-VkImageMemoryBarrier2-srcAccessMask-04859
  If srcAccessMask includes VK_ACCESS_2_VIDEO_DECODE_WRITE_BIT_KHR, srcStageMask **must** include VK_PIPELINE_STAGE_2_VIDEO_DECODE_BIT_KHR
- VUID-VkImageMemoryBarrier2-srcAccessMask-04860
  If srcAccessMask includes VK_ACCESS_2_VIDEO_ENCODE_READ_BIT_KHR, srcStageMask **must** include VK_PIPELINE_STAGE_2_VIDEO_ENCODE_BIT_KHR
- VUID-VkImageMemoryBarrier2-srcAccessMask-04861
  If srcAccessMask includes VK_ACCESS_2_VIDEO_ENCODE_WRITE_BIT_KHR, srcStageMask **must** include VK_PIPELINE_STAGE_2_VIDEO_ENCODE_BIT_KHR
- VUID-VkImageMemoryBarrier2-srcAccessMask-07455
  If srcAccessMask includes VK_ACCESS_2_OPTICAL_FLOW_READ_BIT_NV, srcStageMask **must** include VK_PIPELINE_STAGE_2_OPTICAL_FLOW_BIT_NV
- VUID-VkImageMemoryBarrier2-srcAccessMask-07456
  If srcAccessMask includes VK_ACCESS_2_OPTICAL_FLOW_WRITE_BIT_NV, srcStageMask **must** include VK_PIPELINE_STAGE_2_OPTICAL_FLOW_BIT_NV
- VUID-VkImageMemoryBarrier2-srcAccessMask-07457
  If srcAccessMask includes VK_ACCESS_2_MICROMAP_WRITE_BIT_EXT, srcStageMask **must** include VK_PIPELINE_STAGE_2_MICROMAP_BUILD_BIT_EXT
- VUID-VkImageMemoryBarrier2-srcAccessMask-07458
  If srcAccessMask includes VK_ACCESS_2_MICROMAP_READ_BIT_EXT, srcStageMask **must** include VK_PIPELINE_STAGE_2_MICROMAP_BUILD_BIT_EXT or VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_BUILD_BIT_KHR
- VUID-VkImageMemoryBarrier2-srcAccessMask-08118
  If srcAccessMask includes VK_ACCESS_2_DESCRIPTOR_BUFFER_READ_BIT_EXT, srcStageMask **must** include VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT, or one of VK_PIPELINE_STAGE_*_SHADER_BIT stages

- VUID-VkImageMemoryBarrier2-dstStageMask-03929
  If the geometryShader feature is not enabled, dstStageMask **must** not contain VK_PIPELINE_STAGE_2_GEOMETRY_SHADER_BIT
- VUID-VkImageMemoryBarrier2-dstStageMask-03930
  If the tessellationShader feature is not enabled, dstStageMask **must** not contain VK_PIPELINE_STAGE_2_TESSELLATION_CONTROL_SHADER_BIT or VK_PIPELINE_STAGE_2_TESSELLATION_EVALUATION_SHADER_BIT
- VUID-VkImageMemoryBarrier2-dstStageMask-03931
  If the conditionalRendering feature is not enabled, dstStageMask **must** not contain VK_PIPELINE_STAGE_2_CONDITIONAL_RENDERING_BIT_EXT
- VUID-VkImageMemoryBarrier2-dstStageMask-03932
  If the fragmentDensityMap feature is not enabled, dstStageMask **must** not contain VK_PIPELINE_STAGE_2_FRAGMENT_DENSITY_PROCESS_BIT_EXT
- VUID-VkImageMemoryBarrier2-dstStageMask-03933
  If the transformFeedback feature is not enabled, dstStageMask **must** not contain VK_PIPELINE_STAGE_2_TRANSFORM_FEEDBACK_BIT_EXT
- VUID-VkImageMemoryBarrier2-dstStageMask-03934
  If the meshShader feature is not enabled, dstStageMask **must** not contain VK_PIPELINE_STAGE_2_MESH_SHADER_BIT_EXT
- VUID-VkImageMemoryBarrier2-dstStageMask-03935
  If the taskShader feature is not enabled, dstStageMask **must** not contain VK_PIPELINE_STAGE_2_TASK_SHADER_BIT_EXT
- VUID-VkImageMemoryBarrier2-dstStageMask-07316
  If neither the shadingRateImage or attachmentFragmentShadingRate are enabled, dstStageMask **must** not contain VK_PIPELINE_STAGE_2_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR
- VUID-VkImageMemoryBarrier2-dstStageMask-04957
  If the subpassShading feature is not enabled, dstStageMask **must** not contain VK_PIPELINE_STAGE_2_SUBPASS_SHADER_BIT_HUAWEI
- VUID-VkImageMemoryBarrier2-dstStageMask-04995
  If the invocationMask feature is not enabled, dstStageMask **must** not contain VK_PIPELINE_STAGE_2_INVOCATION_MASK_BIT_HUAWEI
- VUID-VkImageMemoryBarrier2-dstStageMask-07946
  If neither the VK_NV_ray_tracing extension or rayTracingPipeline feature are enabled, dstStageMask **must** not contain VK_PIPELINE_STAGE_2_RAY_TRACING_SHADER_BIT_KHR
- VUID-VkImageMemoryBarrier2-dstAccessMask-03900
  If dstAccessMask includes VK_ACCESS_2_INDIRECT_COMMAND_READ_BIT, dstStageMask **must** include VK_PIPELINE_STAGE_2_DRAW_INDIRECT_BIT, VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_BUILD_BIT_KHR, VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkImageMemoryBarrier2-dstAccessMask-03901
  If dstAccessMask includes VK_ACCESS_2_INDEX_READ_BIT, dstStageMask **must** include VK_PIPELINE_STAGE_2_INDEX_INPUT_BIT, VK_PIPELINE_STAGE_2_VERTEX_INPUT_BIT,

VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT

- VUID-VkImageMemoryBarrier2-dstAccessMask-03902
  If dstAccessMask includes VK_ACCESS_2_VERTEX_ATTRIBUTE_READ_BIT, dstStageMask **must** include VK_PIPELINE_STAGE_2_VERTEX_ATTRIBUTE_INPUT_BIT, VK_PIPELINE_STAGE_2_VERTEX_INPUT_BIT, VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkImageMemoryBarrier2-dstAccessMask-03903
  If dstAccessMask includes VK_ACCESS_2_INPUT_ATTACHMENT_READ_BIT, dstStageMask **must** include VK_PIPELINE_STAGE_2_FRAGMENT_SHADER_BIT, VK_PIPELINE_STAGE_2_SUBPASS_SHADER_BIT_HUAWEI, VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkImageMemoryBarrier2-dstAccessMask-03904
  If dstAccessMask includes VK_ACCESS_2_UNIFORM_READ_BIT, dstStageMask **must** include VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT, or one of the VK_PIPELINE_STAGE_*_SHADER_BIT stages
- VUID-VkImageMemoryBarrier2-dstAccessMask-03905
  If dstAccessMask includes VK_ACCESS_2_SHADER_SAMPLED_READ_BIT, dstStageMask **must** include VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT, or one of the VK_PIPELINE_STAGE_*_SHADER_BIT stages
- VUID-VkImageMemoryBarrier2-dstAccessMask-03906
  If dstAccessMask includes VK_ACCESS_2_SHADER_STORAGE_READ_BIT, dstStageMask **must** include VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT, or one of the VK_PIPELINE_STAGE_*_SHADER_BIT stages
- VUID-VkImageMemoryBarrier2-dstAccessMask-03907
  If dstAccessMask includes VK_ACCESS_2_SHADER_STORAGE_WRITE_BIT, dstStageMask **must** include VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT, or one of the VK_PIPELINE_STAGE_*_SHADER_BIT stages
- VUID-VkImageMemoryBarrier2-dstAccessMask-07454
  If dstAccessMask includes VK_ACCESS_2_SHADER_READ_BIT, dstStageMask **must** include VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT, VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_BUILD_BIT_KHR, VK_PIPELINE_STAGE_2_MICROMAP_BUILD_BIT_EXT, or one of the VK_PIPELINE_STAGE_*_SHADER_BIT stages
- VUID-VkImageMemoryBarrier2-dstAccessMask-03909
  If dstAccessMask includes VK_ACCESS_2_SHADER_WRITE_BIT, dstStageMask **must** include VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT, or one of the VK_PIPELINE_STAGE_*_SHADER_BIT stages
- VUID-VkImageMemoryBarrier2-dstAccessMask-03910
  If dstAccessMask includes VK_ACCESS_2_COLOR_ATTACHMENT_READ_BIT, dstStageMask **must** include VK_PIPELINE_STAGE_2_COLOR_ATTACHMENT_OUTPUT_BIT VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkImageMemoryBarrier2-dstAccessMask-03911
  If dstAccessMask includes VK_ACCESS_2_COLOR_ATTACHMENT_WRITE_BIT, dstStageMask **must** include VK_PIPELINE_STAGE_2_COLOR_ATTACHMENT_OUTPUT_BIT

VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT

- VUID-VkImageMemoryBarrier2-dstAccessMask-03912
  If dstAccessMask includes VK_ACCESS_2_DEPTH_STENCIL_ATTACHMENT_READ_BIT, dstStageMask **must** include VK_PIPELINE_STAGE_2_EARLY_FRAGMENT_TESTS_BIT, VK_PIPELINE_STAGE_2_LATE_FRAGMENT_TESTS_BIT, VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkImageMemoryBarrier2-dstAccessMask-03913
  If dstAccessMask includes VK_ACCESS_2_DEPTH_STENCIL_ATTACHMENT_WRITE_BIT, dstStageMask **must** include VK_PIPELINE_STAGE_2_EARLY_FRAGMENT_TESTS_BIT, VK_PIPELINE_STAGE_2_LATE_FRAGMENT_TESTS_BIT, VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkImageMemoryBarrier2-dstAccessMask-03914
  If dstAccessMask includes VK_ACCESS_2_TRANSFER_READ_BIT, dstStageMask **must** include VK_PIPELINE_STAGE_2_COPY_BIT, VK_PIPELINE_STAGE_2_BLIT_BIT, VK_PIPELINE_STAGE_2_RESOLVE_BIT, VK_PIPELINE_STAGE_2_ALL_TRANSFER_BIT, VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_BUILD_BIT_KHR, VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_COPY_BIT_KHR, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkImageMemoryBarrier2-dstAccessMask-03915
  If dstAccessMask includes VK_ACCESS_2_TRANSFER_WRITE_BIT, dstStageMask **must** include VK_PIPELINE_STAGE_2_COPY_BIT, VK_PIPELINE_STAGE_2_BLIT_BIT, VK_PIPELINE_STAGE_2_RESOLVE_BIT, VK_PIPELINE_STAGE_2_CLEAR_BIT, VK_PIPELINE_STAGE_2_ALL_TRANSFER_BIT, VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_BUILD_BIT_KHR, or VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_COPY_BIT_KHR, VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkImageMemoryBarrier2-dstAccessMask-03916
  If dstAccessMask includes VK_ACCESS_2_HOST_READ_BIT, dstStageMask **must** include VK_PIPELINE_STAGE_2_HOST_BIT
- VUID-VkImageMemoryBarrier2-dstAccessMask-03917
  If dstAccessMask includes VK_ACCESS_2_HOST_WRITE_BIT, dstStageMask **must** include VK_PIPELINE_STAGE_2_HOST_BIT
- VUID-VkImageMemoryBarrier2-dstAccessMask-03918
  If dstAccessMask includes VK_ACCESS_2_CONDITIONAL_RENDERING_READ_BIT_EXT, dstStageMask **must** include VK_PIPELINE_STAGE_2_CONDITIONAL_RENDERING_BIT_EXT, VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkImageMemoryBarrier2-dstAccessMask-03919
  If dstAccessMask includes VK_ACCESS_2_FRAGMENT_DENSITY_MAP_READ_BIT_EXT, dstStageMask **must** include VK_PIPELINE_STAGE_2_FRAGMENT_DENSITY_PROCESS_BIT_EXT, VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkImageMemoryBarrier2-dstAccessMask-03920
  If dstAccessMask includes VK_ACCESS_2_TRANSFORM_FEEDBACK_WRITE_BIT_EXT, dstStageMask **must** include VK_PIPELINE_STAGE_2_TRANSFORM_FEEDBACK_BIT_EXT, VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT

- VUID-VkImageMemoryBarrier2-dstAccessMask-04747
  If dstAccessMask includes VK_ACCESS_2_TRANSFORM_FEEDBACK_COUNTER_READ_BIT_EXT, dstStageMask **must** include VK_PIPELINE_STAGE_2_DRAW_INDIRECT_BIT, VK_PIPELINE_STAGE_2_TRANSFORM_FEEDBACK_BIT_EXT, VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkImageMemoryBarrier2-dstAccessMask-03922
  If dstAccessMask includes VK_ACCESS_2_TRANSFORM_FEEDBACK_COUNTER_WRITE_BIT_EXT, dstStageMask **must** include VK_PIPELINE_STAGE_2_TRANSFORM_FEEDBACK_BIT_EXT, VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkImageMemoryBarrier2-dstAccessMask-03923
  If dstAccessMask includes VK_ACCESS_2_SHADING_RATE_IMAGE_READ_BIT_NV, dstStageMask **must** include VK_PIPELINE_STAGE_2_SHADING_RATE_IMAGE_BIT_NV, VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkImageMemoryBarrier2-dstAccessMask-04994
  If dstAccessMask includes VK_ACCESS_2_INVOCATION_MASK_READ_BIT_HUAWEI, dstStageMask **must** include VK_PIPELINE_STAGE_2_INVOCATION_MASK_BIT_HUAWEI
- VUID-VkImageMemoryBarrier2-dstAccessMask-03924
  If dstAccessMask includes VK_ACCESS_2_COMMAND_PREPROCESS_READ_BIT_NV, dstStageMask **must** include VK_PIPELINE_STAGE_2_COMMAND_PREPROCESS_BIT_NV or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkImageMemoryBarrier2-dstAccessMask-03925
  If dstAccessMask includes VK_ACCESS_2_COMMAND_PREPROCESS_WRITE_BIT_NV, dstStageMask **must** include VK_PIPELINE_STAGE_2_COMMAND_PREPROCESS_BIT_NV or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkImageMemoryBarrier2-dstAccessMask-03926
  If dstAccessMask includes VK_ACCESS_2_COLOR_ATTACHMENT_READ_NONCOHERENT_BIT_EXT, dstStageMask **must** include VK_PIPELINE_STAGE_2_COLOR_ATTACHMENT_OUTPUT_BIT VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkImageMemoryBarrier2-dstAccessMask-03927
  If dstAccessMask includes VK_ACCESS_2_ACCELERATION_STRUCTURE_READ_BIT_KHR, dstStageMask **must** include VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_BUILD_BIT_KHR, VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_COPY_BIT_KHR, VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT, or one of the VK_PIPELINE_STAGE_*_SHADER_BIT stages
- VUID-VkImageMemoryBarrier2-dstAccessMask-03928
  If dstAccessMask includes VK_ACCESS_2_ACCELERATION_STRUCTURE_WRITE_BIT_KHR, dstStageMask **must** include VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_COPY_BIT_KHR, VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_BUILD_BIT_KHR or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT
- VUID-VkImageMemoryBarrier2-dstAccessMask-06256
  If the rayQuery feature is not enabled and dstAccessMask includes VK_ACCESS_2_ACCELERATION_STRUCTURE_READ_BIT_KHR, dstStageMask **must** not include any of the VK_PIPELINE_STAGE_*_SHADER_BIT stages except VK_PIPELINE_STAGE_2_RAY_TRACING_SHADER_BIT_KHR

- VUID-VkImageMemoryBarrier2-dstAccessMask-07272
  If `dstAccessMask` includes `VK_ACCESS_2_SHADER_BINDING_TABLE_READ_BIT_KHR`, `dstStageMask` **must** include `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT` or `VK_PIPELINE_STAGE_2_RAY_TRACING_SHADER_BIT_KHR`
- VUID-VkImageMemoryBarrier2-dstAccessMask-04858
  If `dstAccessMask` includes `VK_ACCESS_2_VIDEO_DECODE_READ_BIT_KHR`, `dstStageMask` **must** include `VK_PIPELINE_STAGE_2_VIDEO_DECODE_BIT_KHR`
- VUID-VkImageMemoryBarrier2-dstAccessMask-04859
  If `dstAccessMask` includes `VK_ACCESS_2_VIDEO_DECODE_WRITE_BIT_KHR`, `dstStageMask` **must** include `VK_PIPELINE_STAGE_2_VIDEO_DECODE_BIT_KHR`
- VUID-VkImageMemoryBarrier2-dstAccessMask-04860
  If `dstAccessMask` includes `VK_ACCESS_2_VIDEO_ENCODE_READ_BIT_KHR`, `dstStageMask` **must** include `VK_PIPELINE_STAGE_2_VIDEO_ENCODE_BIT_KHR`
- VUID-VkImageMemoryBarrier2-dstAccessMask-04861
  If `dstAccessMask` includes `VK_ACCESS_2_VIDEO_ENCODE_WRITE_BIT_KHR`, `dstStageMask` **must** include `VK_PIPELINE_STAGE_2_VIDEO_ENCODE_BIT_KHR`
- VUID-VkImageMemoryBarrier2-dstAccessMask-07455
  If `dstAccessMask` includes `VK_ACCESS_2_OPTICAL_FLOW_READ_BIT_NV`, `dstStageMask` **must** include `VK_PIPELINE_STAGE_2_OPTICAL_FLOW_BIT_NV`
- VUID-VkImageMemoryBarrier2-dstAccessMask-07456
  If `dstAccessMask` includes `VK_ACCESS_2_OPTICAL_FLOW_WRITE_BIT_NV`, `dstStageMask` **must** include `VK_PIPELINE_STAGE_2_OPTICAL_FLOW_BIT_NV`
- VUID-VkImageMemoryBarrier2-dstAccessMask-07457
  If `dstAccessMask` includes `VK_ACCESS_2_MICROMAP_WRITE_BIT_EXT`, `dstStageMask` **must** include `VK_PIPELINE_STAGE_2_MICROMAP_BUILD_BIT_EXT`
- VUID-VkImageMemoryBarrier2-dstAccessMask-07458
  If `dstAccessMask` includes `VK_ACCESS_2_MICROMAP_READ_BIT_EXT`, `dstStageMask` **must** include `VK_PIPELINE_STAGE_2_MICROMAP_BUILD_BIT_EXT` or `VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_BUILD_BIT_KHR`
- VUID-VkImageMemoryBarrier2-dstAccessMask-08118
  If `dstAccessMask` includes `VK_ACCESS_2_DESCRIPTOR_BUFFER_READ_BIT_EXT`, `dstStageMask` **must** include `VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT`, `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`, or one of `VK_PIPELINE_STAGE_*_SHADER_BIT` stages

- VUID-VkImageMemoryBarrier2-oldLayout-01208
  If `srcQueueFamilyIndex` and `dstQueueFamilyIndex` define a queue family ownership transfer or `oldLayout` and `newLayout` define an image layout transition, and `oldLayout` or `newLayout` is `VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL` then `image` **must** have been created with `VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT`
- VUID-VkImageMemoryBarrier2-oldLayout-01209
  If `srcQueueFamilyIndex` and `dstQueueFamilyIndex` define a queue family ownership transfer or `oldLayout` and `newLayout` define an image layout transition, and `oldLayout` or `newLayout` is `VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL` then `image` **must** have been created with `VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT`

- VUID-VkImageMemoryBarrier2-oldLayout-01210
  If srcQueueFamilyIndex and dstQueueFamilyIndex define a queue family ownership transfer or oldLayout and newLayout define an image layout transition, and oldLayout or newLayout is VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL then image **must** have been created with VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT
- VUID-VkImageMemoryBarrier2-oldLayout-01211
  If srcQueueFamilyIndex and dstQueueFamilyIndex define a queue family ownership transfer or oldLayout and newLayout define an image layout transition, and oldLayout or newLayout is VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL then image **must** have been created with VK_IMAGE_USAGE_SAMPLED_BIT or VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT
- VUID-VkImageMemoryBarrier2-oldLayout-01212
  If srcQueueFamilyIndex and dstQueueFamilyIndex define a queue family ownership transfer or oldLayout and newLayout define an image layout transition, and oldLayout or newLayout is VK_IMAGE_LAYOUT_TRANSFER_SRC_OPTIMAL then image **must** have been created with VK_IMAGE_USAGE_TRANSFER_SRC_BIT
- VUID-VkImageMemoryBarrier2-oldLayout-01213
  If srcQueueFamilyIndex and dstQueueFamilyIndex define a queue family ownership transfer or oldLayout and newLayout define an image layout transition, and oldLayout or newLayout is VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL then image **must** have been created with VK_IMAGE_USAGE_TRANSFER_DST_BIT
- VUID-VkImageMemoryBarrier2-oldLayout-01197
  If srcQueueFamilyIndex and dstQueueFamilyIndex define a queue family ownership transfer or oldLayout and newLayout define an image layout transition, oldLayout **must** be VK_IMAGE_LAYOUT_UNDEFINED or the current layout of the image subresources affected by the barrier
- VUID-VkImageMemoryBarrier2-newLayout-01198
  If srcQueueFamilyIndex and dstQueueFamilyIndex define a queue family ownership transfer or oldLayout and newLayout define an image layout transition, newLayout **must** not be VK_IMAGE_LAYOUT_UNDEFINED or VK_IMAGE_LAYOUT_PREINITIALIZED
- VUID-VkImageMemoryBarrier2-oldLayout-01658
  If srcQueueFamilyIndex and dstQueueFamilyIndex define a queue family ownership transfer or oldLayout and newLayout define an image layout transition, and oldLayout or newLayout is VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_STENCIL_ATTACHMENT_OPTIMAL then image **must** have been created with VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT
- VUID-VkImageMemoryBarrier2-oldLayout-01659
  If srcQueueFamilyIndex and dstQueueFamilyIndex define a queue family ownership transfer or oldLayout and newLayout define an image layout transition, and oldLayout or newLayout is VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_STENCIL_READ_ONLY_OPTIMAL then image **must** have been created with VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT
- VUID-VkImageMemoryBarrier2-synchronization2-07793
  If the synchronization2 feature is not enabled, oldLayout **must** not be VK_IMAGE_LAYOUT_ATTACHMENT_OPTIMAL_KHR or VK_IMAGE_LAYOUT_READ_ONLY_OPTIMAL_KHR
- VUID-VkImageMemoryBarrier2-synchronization2-07794
  If the synchronization2 feature is not enabled, newLayout **must** not be

VK_IMAGE_LAYOUT_ATTACHMENT_OPTIMAL_KHR or VK_IMAGE_LAYOUT_READ_ONLY_OPTIMAL_KHR

- VUID-VkImageMemoryBarrier2-srcQueueFamilyIndex-03938
  If srcQueueFamilyIndex and dstQueueFamilyIndex define a queue family ownership transfer or oldLayout and newLayout define an image layout transition, and oldLayout or newLayout is VK_IMAGE_LAYOUT_ATTACHMENT_OPTIMAL, image **must** have been created with VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT or VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT
- VUID-VkImageMemoryBarrier2-srcQueueFamilyIndex-03939
  If srcQueueFamilyIndex and dstQueueFamilyIndex define a queue family ownership transfer or oldLayout and newLayout define an image layout transition, and oldLayout or newLayout is VK_IMAGE_LAYOUT_READ_ONLY_OPTIMAL, image **must** have been created with at least one of VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT, VK_IMAGE_USAGE_SAMPLED_BIT, or VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT
- VUID-VkImageMemoryBarrier2-oldLayout-02088
  If srcQueueFamilyIndex and dstQueueFamilyIndex define a queue family ownership transfer or oldLayout and newLayout define an image layout transition, and oldLayout or newLayout is VK_IMAGE_LAYOUT_FRAGMENT_SHADING_RATE_ATTACHMENT_OPTIMAL_KHR then image **must** have been created with VK_IMAGE_USAGE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR set
- VUID-VkImageMemoryBarrier2-image-09117
  If image was created with a sharing mode of VK_SHARING_MODE_EXCLUSIVE, and srcQueueFamilyIndex and dstQueueFamilyIndex are not equal, srcQueueFamilyIndex **must** be VK_QUEUE_FAMILY_EXTERNAL, VK_QUEUE_FAMILY_FOREIGN_EXT, or a valid queue family
- VUID-VkImageMemoryBarrier2-image-09118
  If image was created with a sharing mode of VK_SHARING_MODE_EXCLUSIVE, and srcQueueFamilyIndex and dstQueueFamilyIndex are not equal, dstQueueFamilyIndex **must** be VK_QUEUE_FAMILY_EXTERNAL, VK_QUEUE_FAMILY_FOREIGN_EXT, or a valid queue family
- VUID-VkImageMemoryBarrier2-srcQueueFamilyIndex-04070
  If srcQueueFamilyIndex is not equal to dstQueueFamilyIndex, at least one of srcQueueFamilyIndex or dstQueueFamilyIndex **must** not be VK_QUEUE_FAMILY_EXTERNAL or VK_QUEUE_FAMILY_FOREIGN_EXT
- VUID-VkImageMemoryBarrier2-None-09119
  If the VK_KHR_external_memory extension is not enabled, and the value of VkApplicationInfo::apiVersion used to create the VkInstance is not greater than or equal to Version 1.1, srcQueueFamilyIndex **must** not be VK_QUEUE_FAMILY_EXTERNAL
- VUID-VkImageMemoryBarrier2-None-09120
  If the VK_KHR_external_memory extension is not enabled, and the value of VkApplicationInfo::apiVersion used to create the VkInstance is not greater than or equal to Version 1.1, dstQueueFamilyIndex **must** not be VK_QUEUE_FAMILY_EXTERNAL
- VUID-VkImageMemoryBarrier2-srcQueueFamilyIndex-09121
  If the VK_EXT_queue_family_foreign extension is not enabled srcQueueFamilyIndex **must** not be VK_QUEUE_FAMILY_FOREIGN_EXT
- VUID-VkImageMemoryBarrier2-dstQueueFamilyIndex-09122
  If the VK_EXT_queue_family_foreign extension is not enabled dstQueueFamilyIndex **must** not be VK_QUEUE_FAMILY_FOREIGN_EXT

- VUID-VkImageMemoryBarrier2-srcQueueFamilyIndex-07120
  If srcQueueFamilyIndex and dstQueueFamilyIndex define a queue family ownership transfer or oldLayout and newLayout define an image layout transition, and oldLayout or newLayout is VK_IMAGE_LAYOUT_VIDEO_DECODE_SRC_KHR then image **must** have been created with VK_IMAGE_USAGE_VIDEO_DECODE_SRC_BIT_KHR
- VUID-VkImageMemoryBarrier2-srcQueueFamilyIndex-07121
  If srcQueueFamilyIndex and dstQueueFamilyIndex define a queue family ownership transfer or oldLayout and newLayout define an image layout transition, and oldLayout or newLayout is VK_IMAGE_LAYOUT_VIDEO_DECODE_DST_KHR then image **must** have been created with VK_IMAGE_USAGE_VIDEO_DECODE_DST_BIT_KHR
- VUID-VkImageMemoryBarrier2-srcQueueFamilyIndex-07122
  If srcQueueFamilyIndex and dstQueueFamilyIndex define a queue family ownership transfer or oldLayout and newLayout define an image layout transition, and oldLayout or newLayout is VK_IMAGE_LAYOUT_VIDEO_DECODE_DPB_KHR then image **must** have been created with VK_IMAGE_USAGE_VIDEO_DECODE_DPB_BIT_KHR
- VUID-VkImageMemoryBarrier2-srcQueueFamilyIndex-07123
  If srcQueueFamilyIndex and dstQueueFamilyIndex define a queue family ownership transfer or oldLayout and newLayout define an image layout transition, and oldLayout or newLayout is VK_IMAGE_LAYOUT_VIDEO_ENCODE_SRC_KHR then image **must** have been created with VK_IMAGE_USAGE_VIDEO_ENCODE_SRC_BIT_KHR
- VUID-VkImageMemoryBarrier2-srcQueueFamilyIndex-07124
  If srcQueueFamilyIndex and dstQueueFamilyIndex define a queue family ownership transfer or oldLayout and newLayout define an image layout transition, and oldLayout or newLayout is VK_IMAGE_LAYOUT_VIDEO_ENCODE_DST_KHR then image **must** have been created with VK_IMAGE_USAGE_VIDEO_ENCODE_DST_BIT_KHR
- VUID-VkImageMemoryBarrier2-srcQueueFamilyIndex-07125
  If srcQueueFamilyIndex and dstQueueFamilyIndex define a queue family ownership transfer or oldLayout and newLayout define an image layout transition, and oldLayout or newLayout is VK_IMAGE_LAYOUT_VIDEO_ENCODE_DPB_KHR then image **must** have been created with VK_IMAGE_USAGE_VIDEO_ENCODE_DPB_BIT_KHR
- VUID-VkImageMemoryBarrier2-srcQueueFamilyIndex-07006
  If srcQueueFamilyIndex and dstQueueFamilyIndex define a queue family ownership transfer or oldLayout and newLayout define an image layout transition, and oldLayout or newLayout is VK_IMAGE_LAYOUT_ATTACHMENT_FEEDBACK_LOOP_OPTIMAL_EXT then image **must** have been created with either the VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT or VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT usage bits, and the VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT or VK_IMAGE_USAGE_SAMPLED_BIT usage bits, and the VK_IMAGE_USAGE_ATTACHMENT_FEEDBACK_LOOP_BIT_EXT usage bit
- VUID-VkImageMemoryBarrier2-attachmentFeedbackLoopLayout-07313
  If the attachmentFeedbackLoopLayout feature is not enabled, newLayout **must** not be VK_IMAGE_LAYOUT_ATTACHMENT_FEEDBACK_LOOP_OPTIMAL_EXT
- VUID-VkImageMemoryBarrier2-srcQueueFamilyIndex-09550
  If srcQueueFamilyIndex and dstQueueFamilyIndex define a queue family ownership transfer or oldLayout and newLayout define an image layout transition, and oldLayout or newLayout is VK_IMAGE_LAYOUT_RENDERING_LOCAL_READ_KHR then image **must** have been created with either

VK_IMAGE_USAGE_STORAGE_BIT, or with both VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT and either of VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT or VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT

- VUID-VkImageMemoryBarrier2-dynamicRenderingLocalRead-09551
  If the dynamicRenderingLocalRead feature is not enabled, oldLayout **must** not be VK_IMAGE_LAYOUT_RENDERING_LOCAL_READ_KHR
- VUID-VkImageMemoryBarrier2-dynamicRenderingLocalRead-09552
  If the dynamicRenderingLocalRead feature is not enabled, newLayout **must** not be VK_IMAGE_LAYOUT_RENDERING_LOCAL_READ_KHR
- VUID-VkImageMemoryBarrier2-subresourceRange-01486
  subresourceRange.baseMipLevel **must** be less than the mipLevels specified in VkImageCreateInfo when image was created
- VUID-VkImageMemoryBarrier2-subresourceRange-01724
  If subresourceRange.levelCount is not VK_REMAINING_MIP_LEVELS, subresourceRange.baseMipLevel + subresourceRange.levelCount **must** be less than or equal to the mipLevels specified in VkImageCreateInfo when image was created
- VUID-VkImageMemoryBarrier2-subresourceRange-01488
  subresourceRange.baseArrayLayer **must** be less than the arrayLayers specified in VkImageCreateInfo when image was created
- VUID-VkImageMemoryBarrier2-subresourceRange-01725
  If subresourceRange.layerCount is not VK_REMAINING_ARRAY_LAYERS, subresourceRange.baseArrayLayer + subresourceRange.layerCount **must** be less than or equal to the arrayLayers specified in VkImageCreateInfo when image was created
- VUID-VkImageMemoryBarrier2-image-01932
  If image is non-sparse then it **must** be bound completely and contiguously to a single VkDeviceMemory object
- VUID-VkImageMemoryBarrier2-image-09241
  If image has a color format that is single-plane, then the aspectMask member of subresourceRange **must** be VK_IMAGE_ASPECT_COLOR_BIT
- VUID-VkImageMemoryBarrier2-image-09242
  If image has a color format and is not *disjoint*, then the aspectMask member of subresourceRange **must** be VK_IMAGE_ASPECT_COLOR_BIT
- VUID-VkImageMemoryBarrier2-image-01672
  If image has a multi-planar format and the image is *disjoint*, then the aspectMask member of subresourceRange **must** include at least one multi-planar aspect mask bit or VK_IMAGE_ASPECT_COLOR_BIT
- VUID-VkImageMemoryBarrier2-image-03320
  If image has a depth/stencil format with both depth and stencil and the separateDepthStencilLayouts feature is not enabled, then the aspectMask member of subresourceRange **must** include both VK_IMAGE_ASPECT_DEPTH_BIT and VK_IMAGE_ASPECT_STENCIL_BIT
- VUID-VkImageMemoryBarrier2-image-03319
  If image has a depth/stencil format with both depth and stencil and the separateDepthStencilLayouts feature is enabled, then the aspectMask member of

subresourceRange **must** include either or both VK_IMAGE_ASPECT_DEPTH_BIT and VK_IMAGE_ASPECT_STENCIL_BIT

- VUID-VkImageMemoryBarrier2-aspectMask-08702
  If the aspectMask member of subresourceRange includes VK_IMAGE_ASPECT_DEPTH_BIT, oldLayout and newLayout **must** not be one of VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL or VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL
- VUID-VkImageMemoryBarrier2-aspectMask-08703
  If the aspectMask member of subresourceRange includes VK_IMAGE_ASPECT_STENCIL_BIT, oldLayout and newLayout **must** not be one of VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL or VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL
- VUID-VkImageMemoryBarrier2-subresourceRange-09601
  subresourceRange.aspectMask **must** be valid for the format the image was created with
- VUID-VkImageMemoryBarrier2-srcStageMask-03854
  If either srcStageMask or dstStageMask includes VK_PIPELINE_STAGE_2_HOST_BIT, srcQueueFamilyIndex and dstQueueFamilyIndex **must** be equal
- VUID-VkImageMemoryBarrier2-srcStageMask-03855
  If srcStageMask includes VK_PIPELINE_STAGE_2_HOST_BIT, and srcQueueFamilyIndex and dstQueueFamilyIndex define a queue family ownership transfer or oldLayout and newLayout define an image layout transition, oldLayout **must** be one of VK_IMAGE_LAYOUT_PREINITIALIZED, VK_IMAGE_LAYOUT_UNDEFINED, or VK_IMAGE_LAYOUT_GENERAL

## Valid Usage (Implicit)

- VUID-VkImageMemoryBarrier2-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_IMAGE_MEMORY_BARRIER_2
- VUID-VkImageMemoryBarrier2-pNext-pNext
  Each pNext member of any structure (including this one) in the pNext chain **must** be either NULL or a pointer to a valid instance of VkExternalMemoryAcquireUnmodifiedEXT or VkSampleLocationsInfoEXT
- VUID-VkImageMemoryBarrier2-sType-unique
  The sType value of each struct in the pNext chain **must** be unique
- VUID-VkImageMemoryBarrier2-srcStageMask-parameter
  srcStageMask **must** be a valid combination of VkPipelineStageFlagBits2 values
- VUID-VkImageMemoryBarrier2-srcAccessMask-parameter
  srcAccessMask **must** be a valid combination of VkAccessFlagBits2 values
- VUID-VkImageMemoryBarrier2-dstStageMask-parameter
  dstStageMask **must** be a valid combination of VkPipelineStageFlagBits2 values
- VUID-VkImageMemoryBarrier2-dstAccessMask-parameter
  dstAccessMask **must** be a valid combination of VkAccessFlagBits2 values
- VUID-VkImageMemoryBarrier2-oldLayout-parameter
  oldLayout **must** be a valid VkImageLayout value
- VUID-VkImageMemoryBarrier2-newLayout-parameter

newLayout **must** be a valid VkImageLayout value

- VUID-VkImageMemoryBarrier2-image-parameter
  image **must** be a valid VkImage handle
- VUID-VkImageMemoryBarrier2-subresourceRange-parameter
  subresourceRange **must** be a valid VkImageSubresourceRange structure

The VkImageMemoryBarrier structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkImageMemoryBarrier {
    VkStructureType            sType;
    const void*                pNext;
    VkAccessFlags              srcAccessMask;
    VkAccessFlags              dstAccessMask;
    VkImageLayout              oldLayout;
    VkImageLayout              newLayout;
    uint32_t                   srcQueueFamilyIndex;
    uint32_t                   dstQueueFamilyIndex;
    VkImage                    image;
    VkImageSubresourceRange    subresourceRange;
} VkImageMemoryBarrier;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- srcAccessMask is a bitmask of VkAccessFlagBits specifying a source access mask.
- dstAccessMask is a bitmask of VkAccessFlagBits specifying a destination access mask.
- oldLayout is the old layout in an image layout transition.
- newLayout is the new layout in an image layout transition.
- srcQueueFamilyIndex is the source queue family for a queue family ownership transfer.
- dstQueueFamilyIndex is the destination queue family for a queue family ownership transfer.
- image is a handle to the image affected by this barrier.
- subresourceRange describes the image subresource range within image that is affected by this barrier.

The first access scope is limited to access to memory through the specified image subresource range, via access types in the source access mask specified by srcAccessMask. If srcAccessMask includes VK_ACCESS_HOST_WRITE_BIT, memory writes performed by that access type are also made visible, as that access type is not performed through a resource.

The second access scope is limited to access to memory through the specified image subresource range, via access types in the destination access mask specified by dstAccessMask. If dstAccessMask includes VK_ACCESS_HOST_WRITE_BIT or VK_ACCESS_HOST_READ_BIT, available memory writes are also made visible to accesses of those types, as those access types are not performed through a resource.

If srcQueueFamilyIndex is not equal to dstQueueFamilyIndex, and srcQueueFamilyIndex is equal to the current queue family, then the memory barrier defines a queue family release operation for the specified image subresource range, and the second synchronization scope of the calling command does not apply to this operation.

If dstQueueFamilyIndex is not equal to srcQueueFamilyIndex, and dstQueueFamilyIndex is equal to the current queue family, then the memory barrier defines a queue family acquire operation for the specified image subresource range, and the first synchronization scope of the calling command does not apply to this operation.

If the synchronization2 feature is not enabled or oldLayout is not equal to newLayout, oldLayout and newLayout define an image layout transition for the specified image subresource range.

| | |
|---|---|
| **NOTE** | If the synchronization2 feature is enabled, when the old and new layout are equal, the layout values are ignored - data is preserved no matter what values are specified, or what layout the image is currently in. |

If image has a multi-planar format and the image is *disjoint*, then including VK_IMAGE_ASPECT_COLOR_BIT in the aspectMask member of subresourceRange is equivalent to including VK_IMAGE_ASPECT_PLANE_0_BIT, VK_IMAGE_ASPECT_PLANE_1_BIT, and (for three-plane formats only) VK_IMAGE_ASPECT_PLANE_2_BIT.

### Valid Usage

- VUID-VkImageMemoryBarrier-oldLayout-01208
  If srcQueueFamilyIndex and dstQueueFamilyIndex define a queue family ownership transfer or oldLayout and newLayout define an image layout transition, and oldLayout or newLayout is VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL then image **must** have been created with VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT
- VUID-VkImageMemoryBarrier-oldLayout-01209
  If srcQueueFamilyIndex and dstQueueFamilyIndex define a queue family ownership transfer or oldLayout and newLayout define an image layout transition, and oldLayout or newLayout is VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL then image **must** have been created with VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT
- VUID-VkImageMemoryBarrier-oldLayout-01210
  If srcQueueFamilyIndex and dstQueueFamilyIndex define a queue family ownership transfer or oldLayout and newLayout define an image layout transition, and oldLayout or newLayout is VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL then image **must** have been created with VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT
- VUID-VkImageMemoryBarrier-oldLayout-01211
  If srcQueueFamilyIndex and dstQueueFamilyIndex define a queue family ownership transfer or oldLayout and newLayout define an image layout transition, and oldLayout or newLayout is VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL then image **must** have been created with VK_IMAGE_USAGE_SAMPLED_BIT or VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT
- VUID-VkImageMemoryBarrier-oldLayout-01212
  If srcQueueFamilyIndex and dstQueueFamilyIndex define a queue family ownership transfer

or oldLayout and newLayout define an image layout transition, and oldLayout or newLayout is VK_IMAGE_LAYOUT_TRANSFER_SRC_OPTIMAL then image **must** have been created with VK_IMAGE_USAGE_TRANSFER_SRC_BIT

- VUID-VkImageMemoryBarrier-oldLayout-01213
  If srcQueueFamilyIndex and dstQueueFamilyIndex define a queue family ownership transfer or oldLayout and newLayout define an image layout transition, and oldLayout or newLayout is VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL then image **must** have been created with VK_IMAGE_USAGE_TRANSFER_DST_BIT
- VUID-VkImageMemoryBarrier-oldLayout-01197
  If srcQueueFamilyIndex and dstQueueFamilyIndex define a queue family ownership transfer or oldLayout and newLayout define an image layout transition, oldLayout **must** be VK_IMAGE_LAYOUT_UNDEFINED or the current layout of the image subresources affected by the barrier
- VUID-VkImageMemoryBarrier-newLayout-01198
  If srcQueueFamilyIndex and dstQueueFamilyIndex define a queue family ownership transfer or oldLayout and newLayout define an image layout transition, newLayout **must** not be VK_IMAGE_LAYOUT_UNDEFINED or VK_IMAGE_LAYOUT_PREINITIALIZED
- VUID-VkImageMemoryBarrier-oldLayout-01658
  If srcQueueFamilyIndex and dstQueueFamilyIndex define a queue family ownership transfer or oldLayout and newLayout define an image layout transition, and oldLayout or newLayout is VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_STENCIL_ATTACHMENT_OPTIMAL then image **must** have been created with VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT
- VUID-VkImageMemoryBarrier-oldLayout-01659
  If srcQueueFamilyIndex and dstQueueFamilyIndex define a queue family ownership transfer or oldLayout and newLayout define an image layout transition, and oldLayout or newLayout is VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_STENCIL_READ_ONLY_OPTIMAL then image **must** have been created with VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT
- VUID-VkImageMemoryBarrier-synchronization2-07793
  If the synchronization2 feature is not enabled, oldLayout **must** not be VK_IMAGE_LAYOUT_ATTACHMENT_OPTIMAL_KHR or VK_IMAGE_LAYOUT_READ_ONLY_OPTIMAL_KHR
- VUID-VkImageMemoryBarrier-synchronization2-07794
  If the synchronization2 feature is not enabled, newLayout **must** not be VK_IMAGE_LAYOUT_ATTACHMENT_OPTIMAL_KHR or VK_IMAGE_LAYOUT_READ_ONLY_OPTIMAL_KHR
- VUID-VkImageMemoryBarrier-srcQueueFamilyIndex-03938
  If srcQueueFamilyIndex and dstQueueFamilyIndex define a queue family ownership transfer or oldLayout and newLayout define an image layout transition, and oldLayout or newLayout is VK_IMAGE_LAYOUT_ATTACHMENT_OPTIMAL, image **must** have been created with VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT or VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT
- VUID-VkImageMemoryBarrier-srcQueueFamilyIndex-03939
  If srcQueueFamilyIndex and dstQueueFamilyIndex define a queue family ownership transfer or oldLayout and newLayout define an image layout transition, and oldLayout or newLayout is VK_IMAGE_LAYOUT_READ_ONLY_OPTIMAL, image **must** have been created with at least one of VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT, VK_IMAGE_USAGE_SAMPLED_BIT, or VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT

- VUID-VkImageMemoryBarrier-oldLayout-02088
  If srcQueueFamilyIndex and dstQueueFamilyIndex define a queue family ownership transfer or oldLayout and newLayout define an image layout transition, and oldLayout or newLayout is VK_IMAGE_LAYOUT_FRAGMENT_SHADING_RATE_ATTACHMENT_OPTIMAL_KHR then image **must** have been created with VK_IMAGE_USAGE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR set
- VUID-VkImageMemoryBarrier-image-09117
  If image was created with a sharing mode of VK_SHARING_MODE_EXCLUSIVE, and srcQueueFamilyIndex and dstQueueFamilyIndex are not equal, srcQueueFamilyIndex **must** be VK_QUEUE_FAMILY_EXTERNAL, VK_QUEUE_FAMILY_FOREIGN_EXT, or a valid queue family
- VUID-VkImageMemoryBarrier-image-09118
  If image was created with a sharing mode of VK_SHARING_MODE_EXCLUSIVE, and srcQueueFamilyIndex and dstQueueFamilyIndex are not equal, dstQueueFamilyIndex **must** be VK_QUEUE_FAMILY_EXTERNAL, VK_QUEUE_FAMILY_FOREIGN_EXT, or a valid queue family
- VUID-VkImageMemoryBarrier-srcQueueFamilyIndex-04070
  If srcQueueFamilyIndex is not equal to dstQueueFamilyIndex, at least one of srcQueueFamilyIndex or dstQueueFamilyIndex **must** not be VK_QUEUE_FAMILY_EXTERNAL or VK_QUEUE_FAMILY_FOREIGN_EXT
- VUID-VkImageMemoryBarrier-None-09119
  If the VK_KHR_external_memory extension is not enabled, and the value of VkApplicationInfo::apiVersion used to create the VkInstance is not greater than or equal to Version 1.1, srcQueueFamilyIndex **must** not be VK_QUEUE_FAMILY_EXTERNAL
- VUID-VkImageMemoryBarrier-None-09120
  If the VK_KHR_external_memory extension is not enabled, and the value of VkApplicationInfo::apiVersion used to create the VkInstance is not greater than or equal to Version 1.1, dstQueueFamilyIndex **must** not be VK_QUEUE_FAMILY_EXTERNAL
- VUID-VkImageMemoryBarrier-srcQueueFamilyIndex-09121
  If the VK_EXT_queue_family_foreign extension is not enabled srcQueueFamilyIndex **must** not be VK_QUEUE_FAMILY_FOREIGN_EXT
- VUID-VkImageMemoryBarrier-dstQueueFamilyIndex-09122
  If the VK_EXT_queue_family_foreign extension is not enabled dstQueueFamilyIndex **must** not be VK_QUEUE_FAMILY_FOREIGN_EXT
- VUID-VkImageMemoryBarrier-srcQueueFamilyIndex-07120
  If srcQueueFamilyIndex and dstQueueFamilyIndex define a queue family ownership transfer or oldLayout and newLayout define an image layout transition, and oldLayout or newLayout is VK_IMAGE_LAYOUT_VIDEO_DECODE_SRC_KHR then image **must** have been created with VK_IMAGE_USAGE_VIDEO_DECODE_SRC_BIT_KHR
- VUID-VkImageMemoryBarrier-srcQueueFamilyIndex-07121
  If srcQueueFamilyIndex and dstQueueFamilyIndex define a queue family ownership transfer or oldLayout and newLayout define an image layout transition, and oldLayout or newLayout is VK_IMAGE_LAYOUT_VIDEO_DECODE_DST_KHR then image **must** have been created with VK_IMAGE_USAGE_VIDEO_DECODE_DST_BIT_KHR
- VUID-VkImageMemoryBarrier-srcQueueFamilyIndex-07122
  If srcQueueFamilyIndex and dstQueueFamilyIndex define a queue family ownership transfer or oldLayout and newLayout define an image layout transition, and oldLayout or newLayout is

VK_IMAGE_LAYOUT_VIDEO_DECODE_DPB_KHR then image **must** have been created with VK_IMAGE_USAGE_VIDEO_DECODE_DPB_BIT_KHR

- VUID-VkImageMemoryBarrier-srcQueueFamilyIndex-07123
  If srcQueueFamilyIndex and dstQueueFamilyIndex define a queue family ownership transfer or oldLayout and newLayout define an image layout transition, and oldLayout or newLayout is VK_IMAGE_LAYOUT_VIDEO_ENCODE_SRC_KHR then image **must** have been created with VK_IMAGE_USAGE_VIDEO_ENCODE_SRC_BIT_KHR
- VUID-VkImageMemoryBarrier-srcQueueFamilyIndex-07124
  If srcQueueFamilyIndex and dstQueueFamilyIndex define a queue family ownership transfer or oldLayout and newLayout define an image layout transition, and oldLayout or newLayout is VK_IMAGE_LAYOUT_VIDEO_ENCODE_DST_KHR then image **must** have been created with VK_IMAGE_USAGE_VIDEO_ENCODE_DST_BIT_KHR
- VUID-VkImageMemoryBarrier-srcQueueFamilyIndex-07125
  If srcQueueFamilyIndex and dstQueueFamilyIndex define a queue family ownership transfer or oldLayout and newLayout define an image layout transition, and oldLayout or newLayout is VK_IMAGE_LAYOUT_VIDEO_ENCODE_DPB_KHR then image **must** have been created with VK_IMAGE_USAGE_VIDEO_ENCODE_DPB_BIT_KHR
- VUID-VkImageMemoryBarrier-srcQueueFamilyIndex-07006
  If srcQueueFamilyIndex and dstQueueFamilyIndex define a queue family ownership transfer or oldLayout and newLayout define an image layout transition, and oldLayout or newLayout is VK_IMAGE_LAYOUT_ATTACHMENT_FEEDBACK_LOOP_OPTIMAL_EXT then image **must** have been created with either the VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT or VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT usage bits, and the VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT or VK_IMAGE_USAGE_SAMPLED_BIT usage bits, and the VK_IMAGE_USAGE_ATTACHMENT_FEEDBACK_LOOP_BIT_EXT usage bit
- VUID-VkImageMemoryBarrier-attachmentFeedbackLoopLayout-07313
  If the attachmentFeedbackLoopLayout feature is not enabled, newLayout **must** not be VK_IMAGE_LAYOUT_ATTACHMENT_FEEDBACK_LOOP_OPTIMAL_EXT
- VUID-VkImageMemoryBarrier-srcQueueFamilyIndex-09550
  If srcQueueFamilyIndex and dstQueueFamilyIndex define a queue family ownership transfer or oldLayout and newLayout define an image layout transition, and oldLayout or newLayout is VK_IMAGE_LAYOUT_RENDERING_LOCAL_READ_KHR then image **must** have been created with either VK_IMAGE_USAGE_STORAGE_BIT, or with both VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT and either of VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT or VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT
- VUID-VkImageMemoryBarrier-dynamicRenderingLocalRead-09551
  If the dynamicRenderingLocalRead feature is not enabled, oldLayout **must** not be VK_IMAGE_LAYOUT_RENDERING_LOCAL_READ_KHR
- VUID-VkImageMemoryBarrier-dynamicRenderingLocalRead-09552
  If the dynamicRenderingLocalRead feature is not enabled, newLayout **must** not be VK_IMAGE_LAYOUT_RENDERING_LOCAL_READ_KHR
- VUID-VkImageMemoryBarrier-subresourceRange-01486
  subresourceRange.baseMipLevel **must** be less than the mipLevels specified in VkImageCreateInfo when image was created

- VUID-VkImageMemoryBarrier-subresourceRange-01724
  If `subresourceRange.levelCount` is not `VK_REMAINING_MIP_LEVELS`, `subresourceRange.baseMipLevel` + `subresourceRange.levelCount` **must** be less than or equal to the `mipLevels` specified in VkImageCreateInfo when `image` was created
- VUID-VkImageMemoryBarrier-subresourceRange-01488
  `subresourceRange.baseArrayLayer` **must** be less than the `arrayLayers` specified in VkImageCreateInfo when `image` was created
- VUID-VkImageMemoryBarrier-subresourceRange-01725
  If `subresourceRange.layerCount` is not `VK_REMAINING_ARRAY_LAYERS`, `subresourceRange.baseArrayLayer` + `subresourceRange.layerCount` **must** be less than or equal to the `arrayLayers` specified in VkImageCreateInfo when `image` was created
- VUID-VkImageMemoryBarrier-image-01932
  If `image` is non-sparse then it **must** be bound completely and contiguously to a single `VkDeviceMemory` object
- VUID-VkImageMemoryBarrier-image-09241
  If `image` has a color format that is single-plane, then the `aspectMask` member of `subresourceRange` **must** be `VK_IMAGE_ASPECT_COLOR_BIT`
- VUID-VkImageMemoryBarrier-image-09242
  If `image` has a color format and is not *disjoint*, then the `aspectMask` member of `subresourceRange` **must** be `VK_IMAGE_ASPECT_COLOR_BIT`
- VUID-VkImageMemoryBarrier-image-01672
  If `image` has a multi-planar format and the image is *disjoint*, then the `aspectMask` member of `subresourceRange` **must** include at least one multi-planar aspect mask bit or `VK_IMAGE_ASPECT_COLOR_BIT`
- VUID-VkImageMemoryBarrier-image-03320
  If `image` has a depth/stencil format with both depth and stencil and the `separateDepthStencilLayouts` feature is not enabled, then the `aspectMask` member of `subresourceRange` **must** include both `VK_IMAGE_ASPECT_DEPTH_BIT` and `VK_IMAGE_ASPECT_STENCIL_BIT`
- VUID-VkImageMemoryBarrier-image-03319
  If `image` has a depth/stencil format with both depth and stencil and the `separateDepthStencilLayouts` feature is enabled, then the `aspectMask` member of `subresourceRange` **must** include either or both `VK_IMAGE_ASPECT_DEPTH_BIT` and `VK_IMAGE_ASPECT_STENCIL_BIT`
- VUID-VkImageMemoryBarrier-aspectMask-08702
  If the `aspectMask` member of `subresourceRange` includes `VK_IMAGE_ASPECT_DEPTH_BIT`, `oldLayout` and `newLayout` **must** not be one of `VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL` or `VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL`
- VUID-VkImageMemoryBarrier-aspectMask-08703
  If the `aspectMask` member of `subresourceRange` includes `VK_IMAGE_ASPECT_STENCIL_BIT`, `oldLayout` and `newLayout` **must** not be one of `VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL` or `VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL`
- VUID-VkImageMemoryBarrier-subresourceRange-09601

subresourceRange.aspectMask **must** be valid for the format the image was created with

- VUID-VkImageMemoryBarrier-None-09052
  If the synchronization2 feature is not enabled, and image was created with a sharing mode of VK_SHARING_MODE_CONCURRENT, at least one of srcQueueFamilyIndex and dstQueueFamilyIndex **must** be VK_QUEUE_FAMILY_IGNORED
- VUID-VkImageMemoryBarrier-None-09053
  If the synchronization2 feature is not enabled, and image was created with a sharing mode of VK_SHARING_MODE_CONCURRENT, srcQueueFamilyIndex **must** be VK_QUEUE_FAMILY_IGNORED or VK_QUEUE_FAMILY_EXTERNAL
- VUID-VkImageMemoryBarrier-None-09054
  If the synchronization2 feature is not enabled, and image was created with a sharing mode of VK_SHARING_MODE_CONCURRENT, dstQueueFamilyIndex **must** be VK_QUEUE_FAMILY_IGNORED or VK_QUEUE_FAMILY_EXTERNAL

### Valid Usage (Implicit)

- VUID-VkImageMemoryBarrier-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_IMAGE_MEMORY_BARRIER
- VUID-VkImageMemoryBarrier-pNext-pNext
  Each pNext member of any structure (including this one) in the pNext chain **must** be either NULL or a pointer to a valid instance of VkExternalMemoryAcquireUnmodifiedEXT or VkSampleLocationsInfoEXT
- VUID-VkImageMemoryBarrier-sType-unique
  The sType value of each struct in the pNext chain **must** be unique
- VUID-VkImageMemoryBarrier-oldLayout-parameter
  oldLayout **must** be a valid VkImageLayout value
- VUID-VkImageMemoryBarrier-newLayout-parameter
  newLayout **must** be a valid VkImageLayout value
- VUID-VkImageMemoryBarrier-image-parameter
  image **must** be a valid VkImage handle
- VUID-VkImageMemoryBarrier-subresourceRange-parameter
  subresourceRange **must** be a valid VkImageSubresourceRange structure

To facilitate usage of images whose memory is initialized on the host, Vulkan allows image layout transitions to be performed by the host as well, albeit supporting limited layouts.

To perform an image layout transition on the host, call:

```
// Provided by VK_EXT_host_image_copy
VkResult vkTransitionImageLayoutEXT(
    VkDevice                                    device,
    uint32_t                                    transitionCount,
    const VkHostImageLayoutTransitionInfoEXT*   pTransitions);
```

- `device` is the device which owns `pTransitions`[i].`image`.
- `transitionCount` is the number of image layout transitions to perform.
- `pTransitions` is a pointer to an array of VkHostImageLayoutTransitionInfoEXT structures specifying the image and subresource ranges within them to transition.

### Valid Usage (Implicit)

- VUID-vkTransitionImageLayoutEXT-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkTransitionImageLayoutEXT-pTransitions-parameter
  `pTransitions` **must** be a valid pointer to an array of `transitionCount` valid VkHostImageLayoutTransitionInfoEXT structures
- VUID-vkTransitionImageLayoutEXT-transitionCount-arraylength
  `transitionCount` **must** be greater than `0`

### Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- `VK_ERROR_INITIALIZATION_FAILED`
- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_MEMORY_MAP_FAILED`

The `VkHostImageLayoutTransitionInfoEXT` structure is defined as:

```
// Provided by VK_EXT_host_image_copy
typedef struct VkHostImageLayoutTransitionInfoEXT {
    VkStructureType            sType;
    const void*                pNext;
    VkImage                    image;
    VkImageLayout              oldLayout;
    VkImageLayout              newLayout;
    VkImageSubresourceRange    subresourceRange;
} VkHostImageLayoutTransitionInfoEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `image` is a handle to the image affected by this layout transition.
- `oldLayout` is the old layout in an image layout transition.
- `newLayout` is the new layout in an image layout transition.
- `subresourceRange` describes the image subresource range within `image` that is affected by this layout transition.

`vkTransitionImageLayoutEXT` does not check whether the device memory associated with an image is currently in use before performing the layout transition. The application **must** guarantee that any previously submitted command that reads from or writes to this subresource has completed before the host performs the layout transition. The memory of `image` is accessed by the host as if coherent.

**NOTE** Image layout transitions performed on the host do not require queue family ownership transfers as the physical layout of the image will not vary between queue families for the layouts supported by this function.

**NOTE** If the device has written to the image memory, it is not automatically made available to the host. Before this command can be called, a memory barrier for this image **must** have been issued on the device with the second synchronization scope including `VK_PIPELINE_STAGE_HOST_BIT` and `VK_ACCESS_HOST_READ_BIT`.

Because queue submissions automatically make host memory visible to the device, there would not be a need for a memory barrier before using the results of this layout transition on the device.

### Valid Usage

- VUID-VkHostImageLayoutTransitionInfoEXT-image-09055
  `image` **must** have been created with `VK_IMAGE_USAGE_HOST_TRANSFER_BIT_EXT`
- VUID-VkHostImageLayoutTransitionInfoEXT-subresourceRange-01486
  `subresourceRange.baseMipLevel` **must** be less than the `mipLevels` specified in VkImageCreateInfo when `image` was created
- VUID-VkHostImageLayoutTransitionInfoEXT-subresourceRange-01724

If `subresourceRange.levelCount` is not `VK_REMAINING_MIP_LEVELS`, `subresourceRange.baseMipLevel` + `subresourceRange.levelCount` **must** be less than or equal to the `mipLevels` specified in VkImageCreateInfo when `image` was created

- VUID-VkHostImageLayoutTransitionInfoEXT-subresourceRange-01488
  `subresourceRange.baseArrayLayer` **must** be less than the `arrayLayers` specified in VkImageCreateInfo when `image` was created
- VUID-VkHostImageLayoutTransitionInfoEXT-subresourceRange-01725
  If `subresourceRange.layerCount` is not `VK_REMAINING_ARRAY_LAYERS`, `subresourceRange.baseArrayLayer` + `subresourceRange.layerCount` **must** be less than or equal to the `arrayLayers` specified in VkImageCreateInfo when `image` was created
- VUID-VkHostImageLayoutTransitionInfoEXT-image-01932
  If `image` is non-sparse then it **must** be bound completely and contiguously to a single `VkDeviceMemory` object
- VUID-VkHostImageLayoutTransitionInfoEXT-image-09241
  If `image` has a color format that is single-plane, then the `aspectMask` member of `subresourceRange` **must** be `VK_IMAGE_ASPECT_COLOR_BIT`
- VUID-VkHostImageLayoutTransitionInfoEXT-image-09242
  If `image` has a color format and is not *disjoint*, then the `aspectMask` member of `subresourceRange` **must** be `VK_IMAGE_ASPECT_COLOR_BIT`
- VUID-VkHostImageLayoutTransitionInfoEXT-image-01672
  If `image` has a multi-planar format and the image is *disjoint*, then the `aspectMask` member of `subresourceRange` **must** include at least one multi-planar aspect mask bit or `VK_IMAGE_ASPECT_COLOR_BIT`
- VUID-VkHostImageLayoutTransitionInfoEXT-image-03320
  If `image` has a depth/stencil format with both depth and stencil and the `separateDepthStencilLayouts` feature is not enabled, then the `aspectMask` member of `subresourceRange` **must** include both `VK_IMAGE_ASPECT_DEPTH_BIT` and `VK_IMAGE_ASPECT_STENCIL_BIT`
- VUID-VkHostImageLayoutTransitionInfoEXT-image-03319
  If `image` has a depth/stencil format with both depth and stencil and the `separateDepthStencilLayouts` feature is enabled, then the `aspectMask` member of `subresourceRange` **must** include either or both `VK_IMAGE_ASPECT_DEPTH_BIT` and `VK_IMAGE_ASPECT_STENCIL_BIT`
- VUID-VkHostImageLayoutTransitionInfoEXT-aspectMask-08702
  If the `aspectMask` member of `subresourceRange` includes `VK_IMAGE_ASPECT_DEPTH_BIT`, `oldLayout` and `newLayout` **must** not be one of `VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL` or `VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL`
- VUID-VkHostImageLayoutTransitionInfoEXT-aspectMask-08703
  If the `aspectMask` member of `subresourceRange` includes `VK_IMAGE_ASPECT_STENCIL_BIT`, `oldLayout` and `newLayout` **must** not be one of `VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL` or `VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL`
- VUID-VkHostImageLayoutTransitionInfoEXT-subresourceRange-09601
  `subresourceRange.aspectMask` **must** be valid for the `format` the `image` was created with

- VUID-VkHostImageLayoutTransitionInfoEXT-oldLayout-09229
  `oldLayout` **must** be either `VK_IMAGE_LAYOUT_UNDEFINED` or the current layout of the image subresources as specified in `subresourceRange`
- VUID-VkHostImageLayoutTransitionInfoEXT-oldLayout-09230
  If `oldLayout` is not `VK_IMAGE_LAYOUT_UNDEFINED` or `VK_IMAGE_LAYOUT_PREINITIALIZED`, it **must** be one of the layouts in VkPhysicalDeviceHostImageCopyPropertiesEXT::`pCopySrcLayouts`
- VUID-VkHostImageLayoutTransitionInfoEXT-newLayout-09057
  `newLayout` **must** be one of the layouts in VkPhysicalDeviceHostImageCopyPropertiesEXT::`pCopyDstLayouts`

### Valid Usage (Implicit)

- VUID-VkHostImageLayoutTransitionInfoEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_HOST_IMAGE_LAYOUT_TRANSITION_INFO_EXT`
- VUID-VkHostImageLayoutTransitionInfoEXT-pNext-pNext
  `pNext` **must** be `NULL`
- VUID-VkHostImageLayoutTransitionInfoEXT-image-parameter
  `image` **must** be a valid VkImage handle
- VUID-VkHostImageLayoutTransitionInfoEXT-oldLayout-parameter
  `oldLayout` **must** be a valid VkImageLayout value
- VUID-VkHostImageLayoutTransitionInfoEXT-newLayout-parameter
  `newLayout` **must** be a valid VkImageLayout value
- VUID-VkHostImageLayoutTransitionInfoEXT-subresourceRange-parameter
  `subresourceRange` **must** be a valid VkImageSubresourceRange structure

## 7.7.4. Queue Family Ownership Transfer

Resources created with a VkSharingMode of `VK_SHARING_MODE_EXCLUSIVE` **must** have their ownership explicitly transferred from one queue family to another in order to access their content in a well-defined manner on a queue in a different queue family.

The special queue family index `VK_QUEUE_FAMILY_IGNORED` indicates that a queue family parameter or member is ignored.

```
#define VK_QUEUE_FAMILY_IGNORED           (~0U)
```

Resources shared with external APIs or instances using external memory **must** also explicitly manage ownership transfers between local and external queues (or equivalent constructs in external APIs) regardless of the VkSharingMode specified when creating them.

The special queue family index `VK_QUEUE_FAMILY_EXTERNAL` represents any queue external to the resource's current Vulkan instance, as long as the queue uses the same underlying device group or physical device, and the same driver version as the resource's VkDevice, as indicated by

VkPhysicalDeviceIDProperties::deviceUUID and VkPhysicalDeviceIDProperties::driverUUID.

```
#define VK_QUEUE_FAMILY_EXTERNAL          (~1U)
```

or the equivalent

```
#define VK_QUEUE_FAMILY_EXTERNAL_KHR      VK_QUEUE_FAMILY_EXTERNAL
```

The special queue family index VK_QUEUE_FAMILY_FOREIGN_EXT represents any queue external to the resource's current Vulkan instance, regardless of the queue's underlying physical device or driver version. This includes, for example, queues for fixed-function image processing devices, media codec devices, and display devices, as well as all queues that use the same underlying device group or physical device, and the same driver version as the resource's VkDevice.

```
#define VK_QUEUE_FAMILY_FOREIGN_EXT       (~2U)
```

If memory dependencies are correctly expressed between uses of such a resource between two queues in different families, but no ownership transfer is defined, the contents of that resource are undefined for any read accesses performed by the second queue family.

| | |
|---|---|
| **NOTE** | If an application does not need the contents of a resource to remain valid when transferring from one queue family to another, then the ownership transfer **should** be skipped. |
| **NOTE** | Applications should expect transfers to/from VK_QUEUE_FAMILY_FOREIGN_EXT to be more expensive than transfers to/from VK_QUEUE_FAMILY_EXTERNAL_KHR. |

A queue family ownership transfer consists of two distinct parts:

1. Release exclusive ownership from the source queue family
2. Acquire exclusive ownership for the destination queue family

An application **must** ensure that these operations occur in the correct order by defining an execution dependency between them, e.g. using a semaphore.

A *release operation* is used to release exclusive ownership of a range of a buffer or image subresource range. A release operation is defined by executing a buffer memory barrier (for a buffer range) or an image memory barrier (for an image subresource range) using a pipeline barrier command, on a queue from the source queue family. The srcQueueFamilyIndex parameter of the barrier **must** be set to the source queue family index, and the dstQueueFamilyIndex parameter to the destination queue family index. dstAccessMask is ignored for such a barrier, such that no visibility operation is executed - the value of this mask does not affect the validity of the barrier. The release operation happens-after the availability operation. dstStageMask is also ignored for such a barrier as defined by buffer memory ownership transfer and image memory ownership transfer.

An *acquire operation* is used to acquire exclusive ownership of a range of a buffer or image subresource range. An acquire operation is defined by executing a buffer memory barrier (for a buffer range) or an image memory barrier (for an image subresource range) using a pipeline barrier command, on a queue from the destination queue family. The buffer range or image subresource range specified in an acquire operation **must** match exactly that of a previous release operation. The `srcQueueFamilyIndex` parameter of the barrier **must** be set to the source queue family index, and the `dstQueueFamilyIndex` parameter to the destination queue family index. `srcAccessMask` is ignored for such a barrier, such that no availability operation is executed - the value of this mask does not affect the validity of the barrier. The acquire operation happens-before the visibility operation. `srcStageMask` is also ignored for such a barrier as defined by buffer memory ownership transfer and image memory ownership transfer. As the first synchronization scope for an acquire operation is empty there is no happens-before dependency. Such a dependency **can** be introduced by using `VK_PIPELINE_STAGE_ALL_COMMANDS_BIT`.

**NOTE**

Whilst it is not invalid to provide destination or source access masks for memory barriers used for release or acquire operations, respectively, they have no practical effect. Access after a release operation has undefined results, and so visibility for those accesses has no practical effect. Similarly, write access before an acquire operation will produce undefined results for future access, so availability of those writes has no practical use. In an earlier version of the specification, these were required to match on both sides - but this was subsequently relaxed. These masks **should** be set to 0.

**NOTE**

Since a release and acquire operation does not synchronize with second and first scopes respectively, the `VK_PIPELINE_STAGE_ALL_COMMANDS_BIT` stage must be used to wait for a release operation to complete. Typically, a release and acquire pair is performed by a VkSemaphore signal and wait in their respective queues. Signaling a semaphore with vkQueueSubmit waits for `VK_PIPELINE_STAGE_ALL_COMMANDS_BIT`. With vkQueueSubmit2KHR, `stageMask` for the signal semaphore must be `VK_PIPELINE_STAGE_ALL_COMMANDS_BIT`. Similarly, for the acquire operation, waiting for a semaphore must use `VK_PIPELINE_STAGE_ALL_COMMANDS_BIT` to make sure the acquire operation is synchronized.

If the transfer is via an image memory barrier, and an image layout transition is desired, then the values of `oldLayout` and `newLayout` in the *release operation*'s memory barrier **must** be equal to values of `oldLayout` and `newLayout` in the *acquire operation*'s memory barrier. Although the image layout transition is submitted twice, it will only be executed once. A layout transition specified in this way happens-after the *release operation* and happens-before the *acquire operation*.

If the values of `srcQueueFamilyIndex` and `dstQueueFamilyIndex` are equal, no ownership transfer is performed, and the barrier operates as if they were both set to `VK_QUEUE_FAMILY_IGNORED`.

Queue family ownership transfers **may** perform read and write accesses on all memory bound to the image subresource or buffer range, so applications **must** ensure that all memory writes have been made available before a queue family ownership transfer is executed. Available memory is automatically made visible to queue family release and acquire operations, and writes performed by those operations are automatically made available.

Once a queue family has acquired ownership of a buffer range or image subresource range of a VK_SHARING_MODE_EXCLUSIVE resource, its contents are undefined to other queue families unless ownership is transferred. The contents of any portion of another resource which aliases memory that is bound to the transferred buffer or image subresource range are undefined after a release or acquire operation.

**NOTE** Because events **cannot** be used directly for inter-queue synchronization, and because vkCmdSetEvent does not have the queue family index or memory barrier parameters needed by a *release operation*, the release and acquire operations of a queue family ownership transfer **can** only be performed using vkCmdPipelineBarrier.

An *acquire operation* **may** have a performance penalty when acquiring ownership of a subresource range from one of the special queue families reserved for external memory ownership transfers described above. The application **can** reduce the performance penalty in some cases by adding a VkExternalMemoryAcquireUnmodifiedEXT structure to the pNext chain of the *acquire operation*'s memory barrier structure.

The VkExternalMemoryAcquireUnmodifiedEXT structure is defined as:

```
// Provided by VK_EXT_external_memory_acquire_unmodified
typedef struct VkExternalMemoryAcquireUnmodifiedEXT {
    VkStructureType    sType;
    const void*        pNext;
    VkBool32           acquireUnmodifiedMemory;
} VkExternalMemoryAcquireUnmodifiedEXT;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- acquireUnmodifiedMemory specifies, if VK_TRUE, that no range of VkDeviceMemory bound to the resource of the memory barrier's subresource range was modified at any time since the resource's most recent release of ownership to the queue family specified by the memory barrier's srcQueueFamilyIndex. If VK_FALSE, it specifies nothing.

If the application releases ownership of the subresource range to one of the special queue families reserved for external memory ownership transfers with a memory barrier structure, and later re-acquires ownership from the same queue family with a memory barrier structure, and if no range of VkDeviceMemory bound to the resource was modified at any time between the *release operation* and the *acquire operation*, then the application **should** add a VkExternalMemoryAcquireUnmodifiedEXT structure to the pNext chain of the *acquire operation*'s memory barrier structure because this **may** reduce the performance penalty.

This struct is ignored if acquireUnmodifiedMemory is VK_FALSE. In particular, VK_FALSE does *not* specify that memory was modified.

This struct is ignored if the memory barrier's srcQueueFamilyIndex is not a special queue family reserved for external memory ownership transfers.

**NOTE**

The method by which the application determines whether memory was modified between the *release operation* and *acquire operation* is outside the scope of Vulkan.

For any Vulkan operation that accesses a resource, the application **must** not assume the implementation accesses the resource's memory as read-only, even for *apparently* read-only operations such as transfer commands and shader reads.

The validity of VkExternalMemoryAcquireUnmodifiedEXT::`acquireUnmodifiedMemory` is independent of memory ranges outside the ranges of VkDeviceMemory bound to the resource. In particular, it is independent of any implementation-private memory associated with the resource.

### Valid Usage

- VUID-VkExternalMemoryAcquireUnmodifiedEXT-acquireUnmodifiedMemory-08922
  If `acquireUnmodifiedMemory` is `VK_TRUE`, and the memory barrier's `srcQueueFamilyIndex` is a special queue family reserved for external memory ownership transfers (as described in Queue Family Ownership Transfer), then each range of VkDeviceMemory bound to the resource **must** have remained unmodified during all time since the resource's most recent release of ownership to the queue family

### Valid Usage (Implicit)

- VUID-VkExternalMemoryAcquireUnmodifiedEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_EXTERNAL_MEMORY_ACQUIRE_UNMODIFIED_EXT`

## 7.8. Wait Idle Operations

To wait on the host for the completion of outstanding queue operations for a given queue, call:

```
// Provided by VK_VERSION_1_0
VkResult vkQueueWaitIdle(
    VkQueue                                     queue);
```

- `queue` is the queue on which to wait.

`vkQueueWaitIdle` is equivalent to having submitted a valid fence to every previously executed queue submission command that accepts a fence, then waiting for all of those fences to signal using vkWaitForFences with an infinite timeout and `waitAll` set to `VK_TRUE`.

### Valid Usage (Implicit)

- VUID-vkQueueWaitIdle-queue-parameter
  `queue` **must** be a valid VkQueue handle

### Host Synchronization

- Host access to `queue` **must** be externally synchronized

### Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| - | - | - | Any | - |

### Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_DEVICE_LOST`

To wait on the host for the completion of outstanding queue operations for all queues on a given logical device, call:

```
// Provided by VK_VERSION_1_0
VkResult vkDeviceWaitIdle(
    VkDevice                                    device);
```

- `device` is the logical device to idle.

`vkDeviceWaitIdle` is equivalent to calling `vkQueueWaitIdle` for all queues owned by `device`.

### Valid Usage (Implicit)

- VUID-vkDeviceWaitIdle-device-parameter
  `device` **must** be a valid VkDevice handle

### Host Synchronization

- Host access to all `VkQueue` objects created from `device` **must** be externally synchronized

**Return Codes**

**Success**

- `VK_SUCCESS`

**Failure**

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_DEVICE_LOST`

## 7.9. Host Write Ordering Guarantees

When batches of command buffers are submitted to a queue via a queue submission command, it defines a memory dependency with prior host operations, and execution of command buffers submitted to the queue.

The first synchronization scope includes execution of vkQueueSubmit on the host and anything that happened-before it, as defined by the host memory model.

**NOTE** Some systems allow writes that do not directly integrate with the host memory model; these have to be synchronized by the application manually. One example of this is non-temporal store instructions on x86; to ensure these happen-before submission, applications should call `_mm_sfence()`.

The second synchronization scope includes all commands submitted in the same queue submission, and all commands that occur later in submission order.

The first access scope includes all host writes to mappable device memory that are available to the host memory domain.

The second access scope includes all memory access performed by the device.

## 7.10. Synchronization and Multiple Physical Devices

If a logical device includes more than one physical device, then fences, semaphores, and events all still have a single instance of the signaled state.

A fence becomes signaled when all physical devices complete the necessary queue operations.

Semaphore wait and signal operations all include a device index that is the sole physical device that performs the operation. These indices are provided in the VkDeviceGroupSubmitInfo and VkDeviceGroupBindSparseInfo structures. Semaphores are not exclusively owned by any physical device. For example, a semaphore can be signaled by one physical device and then waited on by a different physical device.

An event **can** only be waited on by the same physical device that signaled it (or the host).

## 7.11. Calibrated Timestamps

In order to be able to correlate the time a particular operation took place at on timelines of different time domains (e.g. a device operation vs. a host operation), Vulkan allows querying calibrated timestamps from multiple time domains.

To query calibrated timestamps from a set of time domains, call:

```
// Provided by VK_KHR_calibrated_timestamps
VkResult vkGetCalibratedTimestampsKHR(
    VkDevice                                    device,
    uint32_t                                    timestampCount,
    const VkCalibratedTimestampInfoKHR*         pTimestampInfos,
    uint64_t*                                   pTimestamps,
    uint64_t*                                   pMaxDeviation);
```

or the equivalent command

```
// Provided by VK_EXT_calibrated_timestamps
VkResult vkGetCalibratedTimestampsEXT(
    VkDevice                                    device,
    uint32_t                                    timestampCount,
    const VkCalibratedTimestampInfoKHR*         pTimestampInfos,
    uint64_t*                                   pTimestamps,
    uint64_t*                                   pMaxDeviation);
```

- `device` is the logical device used to perform the query.
- `timestampCount` is the number of timestamps to query.
- `pTimestampInfos` is a pointer to an array of `timestampCount` VkCalibratedTimestampInfoKHR structures, describing the time domains the calibrated timestamps should be captured from.
- `pTimestamps` is a pointer to an array of `timestampCount` 64-bit unsigned integer values in which the requested calibrated timestamp values are returned.
- `pMaxDeviation` is a pointer to a 64-bit unsigned integer value in which the strictly positive maximum deviation, in nanoseconds, of the calibrated timestamp values is returned.

**NOTE**

The maximum deviation **may** vary between calls to `vkGetCalibratedTimestampsKHR` even for the same set of time domains due to implementation and platform specific reasons. It is the application's responsibility to assess whether the returned maximum deviation makes the timestamp values suitable for any particular purpose and **can** choose to re-issue the timestamp calibration call pursuing a lower deviation value.

Calibrated timestamp values **can** be extrapolated to estimate future coinciding timestamp values, however, depending on the nature of the time domains and other properties of the platform extrapolating values over a sufficiently long period of time **may** no longer be accurate enough to fit

any particular purpose, so applications are expected to re-calibrate the timestamps on a regular basis.

### Valid Usage

- VUID-vkGetCalibratedTimestampsEXT-timeDomain-09246
  The `timeDomain` value of each VkCalibratedTimestampInfoKHR in `pTimestampInfos` **must** be unique

### Valid Usage (Implicit)

- VUID-vkGetCalibratedTimestampsKHR-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkGetCalibratedTimestampsKHR-pTimestampInfos-parameter
  `pTimestampInfos` **must** be a valid pointer to an array of `timestampCount` valid VkCalibratedTimestampInfoKHR structures
- VUID-vkGetCalibratedTimestampsKHR-pTimestamps-parameter
  `pTimestamps` **must** be a valid pointer to an array of `timestampCount` `uint64_t` values
- VUID-vkGetCalibratedTimestampsKHR-pMaxDeviation-parameter
  `pMaxDeviation` **must** be a valid pointer to a `uint64_t` value
- VUID-vkGetCalibratedTimestampsKHR-timestampCount-arraylength
  `timestampCount` **must** be greater than `0`

### Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`

The `VkCalibratedTimestampInfoKHR` structure is defined as:

```
// Provided by VK_KHR_calibrated_timestamps
typedef struct VkCalibratedTimestampInfoKHR {
    VkStructureType    sType;
    const void*        pNext;
    VkTimeDomainKHR    timeDomain;
} VkCalibratedTimestampInfoKHR;
```

or the equivalent

```
// Provided by VK_EXT_calibrated_timestamps
typedef VkCalibratedTimestampInfoKHR VkCalibratedTimestampInfoEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `timeDomain` is a VkTimeDomainKHR value specifying the time domain from which the calibrated timestamp value should be returned.

### Valid Usage

- VUID-VkCalibratedTimestampInfoEXT-timeDomain-02354
  `timeDomain` **must** be one of the VkTimeDomainKHR values returned by vkGetPhysicalDeviceCalibrateableTimeDomainsKHR

### Valid Usage (Implicit)

- VUID-VkCalibratedTimestampInfoKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_CALIBRATED_TIMESTAMP_INFO_KHR`
- VUID-VkCalibratedTimestampInfoKHR-pNext-pNext
  `pNext` **must** be `NULL`
- VUID-VkCalibratedTimestampInfoKHR-timeDomain-parameter
  `timeDomain` **must** be a valid VkTimeDomainKHR value

The set of supported time domains consists of:

```
// Provided by VK_KHR_calibrated_timestamps
typedef enum VkTimeDomainKHR {
    VK_TIME_DOMAIN_DEVICE_KHR = 0,
    VK_TIME_DOMAIN_CLOCK_MONOTONIC_KHR = 1,
    VK_TIME_DOMAIN_CLOCK_MONOTONIC_RAW_KHR = 2,
    VK_TIME_DOMAIN_QUERY_PERFORMANCE_COUNTER_KHR = 3,
  // Provided by VK_EXT_calibrated_timestamps
    VK_TIME_DOMAIN_DEVICE_EXT = VK_TIME_DOMAIN_DEVICE_KHR,
  // Provided by VK_EXT_calibrated_timestamps
    VK_TIME_DOMAIN_CLOCK_MONOTONIC_EXT = VK_TIME_DOMAIN_CLOCK_MONOTONIC_KHR,
  // Provided by VK_EXT_calibrated_timestamps
    VK_TIME_DOMAIN_CLOCK_MONOTONIC_RAW_EXT = VK_TIME_DOMAIN_CLOCK_MONOTONIC_RAW_KHR,
  // Provided by VK_EXT_calibrated_timestamps
    VK_TIME_DOMAIN_QUERY_PERFORMANCE_COUNTER_EXT =
VK_TIME_DOMAIN_QUERY_PERFORMANCE_COUNTER_KHR,
} VkTimeDomainKHR;
```

or the equivalent

```
// Provided by VK_EXT_calibrated_timestamps
typedef VkTimeDomainKHR VkTimeDomainEXT;
```

- VK_TIME_DOMAIN_DEVICE_KHR specifies the device time domain. Timestamp values in this time domain use the same units and are comparable with device timestamp values captured using vkCmdWriteTimestamp or vkCmdWriteTimestamp2KHR and are defined to be incrementing according to the timestampPeriod of the device.
- VK_TIME_DOMAIN_CLOCK_MONOTONIC_KHR specifies the CLOCK_MONOTONIC time domain available on POSIX platforms. Timestamp values in this time domain are in units of nanoseconds and are comparable with platform timestamp values captured using the POSIX clock_gettime API as computed by this example:

**NOTE** An implementation supporting VK_KHR_calibrated_timestamps or VK_EXT_calibrated_timestamps will use the same time domain for all its VkQueue so that timestamp values reported for VK_TIME_DOMAIN_DEVICE_KHR can be matched to any timestamp captured through vkCmdWriteTimestamp or vkCmdWriteTimestamp2KHR .

```
struct timespec tv;
clock_gettime(CLOCK_MONOTONIC, &tv);
return tv.tv_nsec + tv.tv_sec*1000000000ull;
```

- VK_TIME_DOMAIN_CLOCK_MONOTONIC_RAW_KHR specifies the CLOCK_MONOTONIC_RAW time domain available on POSIX platforms. Timestamp values in this time domain are in units of nanoseconds and are comparable with platform timestamp values captured using the POSIX clock_gettime API as computed by this example:

```
struct timespec tv;
clock_gettime(CLOCK_MONOTONIC_RAW, &tv);
return tv.tv_nsec + tv.tv_sec*1000000000ull;
```

- VK_TIME_DOMAIN_QUERY_PERFORMANCE_COUNTER_KHR specifies the performance counter (QPC) time domain available on Windows. Timestamp values in this time domain are in the same units as those provided by the Windows QueryPerformanceCounter API and are comparable with platform timestamp values captured using that API as computed by this example:

```
LARGE_INTEGER counter;
QueryPerformanceCounter(&counter);
return counter.QuadPart;
```

# Chapter 8. Render Pass

Draw commands **must** be recorded within a *render pass instance*. Each render pass instance defines a set of image resources, referred to as *attachments*, used during rendering.

To begin a render pass instance, call:

```
// Provided by VK_KHR_dynamic_rendering
void vkCmdBeginRenderingKHR(
    VkCommandBuffer                             commandBuffer,
    const VkRenderingInfo*                      pRenderingInfo);
```

- `commandBuffer` is the command buffer in which to record the command.
- `pRenderingInfo` is a pointer to a VkRenderingInfo structure specifying details of the render pass instance to begin.

After beginning a render pass instance, the command buffer is ready to record draw commands.

If `pRenderingInfo->flags` includes `VK_RENDERING_RESUMING_BIT` then this render pass is resumed from a render pass instance that has been suspended earlier in submission order.

### Valid Usage

- VUID-vkCmdBeginRendering-dynamicRendering-06446
  The `dynamicRendering` feature **must** be enabled
- VUID-vkCmdBeginRendering-commandBuffer-06068
  If `commandBuffer` is a secondary command buffer, and the `nestedCommandBuffer` feature is not enabled, `pRenderingInfo->flags` **must** not include `VK_RENDERING_CONTENTS_SECONDARY_COMMAND_BUFFERS_BIT`
- VUID-vkCmdBeginRendering-pRenderingInfo-09588
  If `pRenderingInfo->pDepthAttachment` is not `NULL` and `pRenderingInfo->pDepthAttachment->imageView` is not VK_NULL_HANDLE, `pRenderingInfo->pDepthAttachment->imageView` **must** be in the layout specified by `pRenderingInfo->pDepthAttachment->imageLayout`
- VUID-vkCmdBeginRendering-pRenderingInfo-09589
  If `pRenderingInfo->pDepthAttachment` is not `NULL`, `pRenderingInfo->pDepthAttachment->imageView` is not VK_NULL_HANDLE, `pRenderingInfo->pDepthAttachment->imageResolveMode` is not `VK_RESOLVE_MODE_NONE`, and `pRenderingInfo->pDepthAttachment->resolveImageView` is not VK_NULL_HANDLE, `pRenderingInfo->pDepthAttachment->resolveImageView` **must** be in the layout specified by `pRenderingInfo->pDepthAttachment->resolveImageLayout`
- VUID-vkCmdBeginRendering-pRenderingInfo-09590
  If `pRenderingInfo->pStencilAttachment` is not `NULL` and `pRenderingInfo->pStencilAttachment->imageView` is not VK_NULL_HANDLE, `pRenderingInfo->pStencilAttachment->imageView` **must** be in the layout specified by `pRenderingInfo->pStencilAttachment->imageLayout`

- VUID-vkCmdBeginRendering-pRenderingInfo-09591
  If `pRenderingInfo->pStencilAttachment` is not `NULL`, `pRenderingInfo->pStencilAttachment->imageView` is not VK_NULL_HANDLE, `pRenderingInfo->pStencilAttachment->imageResolveMode` is not `VK_RESOLVE_MODE_NONE`, and `pRenderingInfo->pStencilAttachment->resolveImageView` is not VK_NULL_HANDLE, `pRenderingInfo->pStencilAttachment->resolveImageView` **must** be in the layout specified by `pRenderingInfo->pStencilAttachment->resolveImageLayout`
- VUID-vkCmdBeginRendering-pRenderingInfo-09592
  For any element of `pRenderingInfo->pColorAttachments`, if `imageView` is not VK_NULL_HANDLE, that image view **must** be in the layout specified by `imageLayout`
- VUID-vkCmdBeginRendering-pRenderingInfo-09593
  For any element of `pRenderingInfo->pColorAttachments`, if either `imageResolveMode` is `VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID`, or `imageView` is not VK_NULL_HANDLE and `resolveMode` is not `VK_RESOLVE_MODE_NONE`, and `resolveImageView` is not VK_NULL_HANDLE, `resolveImageView` **must** be in the layout specified by `resolveImageLayout`

## Valid Usage (Implicit)

- VUID-vkCmdBeginRendering-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdBeginRendering-pRenderingInfo-parameter
  `pRenderingInfo` **must** be a valid pointer to a valid VkRenderingInfo structure
- VUID-vkCmdBeginRendering-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdBeginRendering-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- VUID-vkCmdBeginRendering-renderpass
  This command **must** only be called outside of a render pass instance
- VUID-vkCmdBeginRendering-videocoding
  This command **must** only be called outside of a video coding scope

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

### Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Outside | Outside | Graphics | Action<br>State |

The `VkRenderingInfo` structure is defined as:

```
typedef struct VkRenderingInfo {
    VkStructureType                     sType;
    const void*                         pNext;
    VkRenderingFlags                    flags;
    VkRect2D                            renderArea;
    uint32_t                            layerCount;
    uint32_t                            viewMask;
    uint32_t                            colorAttachmentCount;
    const VkRenderingAttachmentInfo*    pColorAttachments;
    const VkRenderingAttachmentInfo*    pDepthAttachment;
    const VkRenderingAttachmentInfo*    pStencilAttachment;
} VkRenderingInfo;
```

or the equivalent

```
// Provided by VK_KHR_dynamic_rendering, VK_QCOM_tile_properties with
VK_KHR_dynamic_rendering or VK_VERSION_1_3
typedef VkRenderingInfo VkRenderingInfoKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is a bitmask of VkRenderingFlagBits.
- `renderArea` is the render area that is affected by the render pass instance.
- `layerCount` is the number of layers rendered to in each attachment when `viewMask` is `0`.
- `viewMask` is the view mask indicating the indices of attachment layers that will be rendered when it is not `0`.
- `colorAttachmentCount` is the number of elements in `pColorAttachments`.
- `pColorAttachments` is a pointer to an array of `colorAttachmentCount` VkRenderingAttachmentInfo structures describing any color attachments used.
- `pDepthAttachment` is a pointer to a VkRenderingAttachmentInfo structure describing a depth attachment.
- `pStencilAttachment` is a pointer to a VkRenderingAttachmentInfo structure describing a stencil

attachment.

If viewMask is not 0, multiview is enabled.

If there is an instance of VkDeviceGroupRenderPassBeginInfo included in the pNext chain and its deviceRenderAreaCount member is not 0, then renderArea is ignored, and the render area is defined per-device by that structure.

If multiview is enabled, and the multiviewPerViewRenderAreas feature is enabled, and there is an instance of VkMultiviewPerViewRenderAreasRenderPassBeginInfoQCOM included in the pNext chain with perViewRenderAreaCount not equal to 0, then the elements of VkMultiviewPerViewRenderAreasRenderPassBeginInfoQCOM::pPerViewRenderAreas override renderArea and define a render area for each view. In this case, renderArea **must** be set to an area at least as large as the union of all the per-view render areas.

Each element of the pColorAttachments array corresponds to an output location in the shader, i.e. if the shader declares an output variable decorated with a Location value of **X**, then it uses the attachment provided in pColorAttachments[**X**]. If the imageView member of any element of pColorAttachments is VK_NULL_HANDLE, and resolveMode is not VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID, writes to the corresponding location by a fragment are discarded.

## Valid Usage

- VUID-VkRenderingInfo-viewMask-06069
  If viewMask is 0, layerCount **must** not be 0
- VUID-VkRenderingInfo-multisampledRenderToSingleSampled-06857
  imageView members of pDepthAttachment, pStencilAttachment, and elements of pColorAttachments that are not VK_NULL_HANDLE **must** have been created with the same sampleCount , if none of the following are enabled:
  - The VK_AMD_mixed_attachment_samples extension
  - The VK_NV_framebuffer_mixed_samples extension
  - The multisampledRenderToSingleSampled feature,
- VUID-VkRenderingInfo-imageView-09429
  imageView members of elements of pColorAttachments that are not VK_NULL_HANDLE **must** have been created with the same sampleCount , if the multisampledRenderToSingleSampled feature is not enabled
- VUID-VkRenderingInfo-None-08994
  If VkDeviceGroupRenderPassBeginInfo::deviceRenderAreaCount is 0, renderArea.extent.width **must** be greater than 0
- VUID-VkRenderingInfo-None-08995
  If VkDeviceGroupRenderPassBeginInfo::deviceRenderAreaCount is 0, renderArea.extent.height **must** be greater than 0
- VUID-VkRenderingInfo-imageView-06858
  If multisampled-render-to-single-sampled is enabled, then all attachments referenced by

imageView members of pDepthAttachment, pStencilAttachment, and elements of pColorAttachments that are not VK_NULL_HANDLE **must** have a sample count that is either VK_SAMPLE_COUNT_1_BIT or equal to VkMultisampledRenderToSingleSampledInfoEXT::rasterizationSamples

- VUID-VkRenderingInfo-imageView-06859
  If multisampled-render-to-single-sampled is enabled, then all attachments referenced by imageView members of pDepthAttachment, pStencilAttachment, and elements of pColorAttachments that are not VK_NULL_HANDLE and have a sample count of VK_SAMPLE_COUNT_1_BIT **must** have been created with VK_IMAGE_CREATE_MULTISAMPLED_RENDER_TO_SINGLE_SAMPLED_BIT_EXT in their VkImageCreateInfo::flags
- VUID-VkRenderingInfo-pNext-06077
  If the pNext chain does not contain VkDeviceGroupRenderPassBeginInfo or its deviceRenderAreaCount member is equal to 0, renderArea.offset.x **must** be greater than or equal to 0
- VUID-VkRenderingInfo-pNext-06078
  If the pNext chain does not contain VkDeviceGroupRenderPassBeginInfo or its deviceRenderAreaCount member is equal to 0, renderArea.offset.y **must** be greater than or equal to 0
- VUID-VkRenderingInfo-pNext-07815
  If the pNext chain does not contain VkDeviceGroupRenderPassBeginInfo or its deviceRenderAreaCount member is equal to 0, the sum of renderArea.extent.width and renderArea.offset.x **must** be less than or equal to maxFramebufferWidth
- VUID-VkRenderingInfo-pNext-07816
  If the pNext chain does not contain VkDeviceGroupRenderPassBeginInfo or its deviceRenderAreaCount member is equal to 0, the sum of renderArea.extent.height and renderArea.offset.y **must** be less than or equal to maxFramebufferHeight
- VUID-VkRenderingInfo-pNext-06079
  If the pNext chain does not contain VkDeviceGroupRenderPassBeginInfo or its deviceRenderAreaCount member is equal to 0, the width of the imageView member of any element of pColorAttachments, pDepthAttachment, or pStencilAttachment that is not VK_NULL_HANDLE **must** be greater than or equal to renderArea.offset.x + renderArea.extent.width
- VUID-VkRenderingInfo-pNext-06080
  If the pNext chain does not contain VkDeviceGroupRenderPassBeginInfo or its deviceRenderAreaCount member is equal to 0, the height of the imageView member of any element of pColorAttachments, pDepthAttachment, or pStencilAttachment that is not VK_NULL_HANDLE **must** be greater than or equal to renderArea.offset.y + renderArea.extent.height
- VUID-VkRenderingInfo-pNext-06083
  If the pNext chain contains VkDeviceGroupRenderPassBeginInfo, the width of the imageView member of any element of pColorAttachments, pDepthAttachment, or pStencilAttachment that is not VK_NULL_HANDLE **must** be greater than or equal to the sum of the offset.x and extent.width members of each element of pDeviceRenderAreas

- VUID-VkRenderingInfo-pNext-06084
  If the `pNext` chain contains VkDeviceGroupRenderPassBeginInfo, the height of the `imageView` member of any element of `pColorAttachments`, `pDepthAttachment`, or `pStencilAttachment` that is not VK_NULL_HANDLE **must** be greater than or equal to the sum of the `offset.y` and `extent.height` members of each element of `pDeviceRenderAreas`
- VUID-VkRenderingInfo-pDepthAttachment-06085
  If neither `pDepthAttachment` or `pStencilAttachment` are `NULL` and the `imageView` member of either structure is not VK_NULL_HANDLE, the `imageView` member of each structure **must** be the same
- VUID-VkRenderingInfo-pDepthAttachment-06086
  If neither `pDepthAttachment` or `pStencilAttachment` are `NULL`, and the `resolveMode` member of each is not `VK_RESOLVE_MODE_NONE`, the `resolveImageView` member of each structure **must** be the same
- VUID-VkRenderingInfo-colorAttachmentCount-06087
  If `colorAttachmentCount` is not `0` and the `imageView` member of an element of `pColorAttachments` is not VK_NULL_HANDLE, that `imageView` **must** have been created with `VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT`
- VUID-VkRenderingInfo-colorAttachmentCount-09476
  If `colorAttachmentCount` is not `0` and there is an element of `pColorAttachments` with either its `resolveMode` member set to `VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID`, or its `imageView` member not VK_NULL_HANDLE, and its `resolveMode` member not set to `VK_RESOLVE_MODE_NONE`, the `resolveImageView` member of that element of `pColorAttachments` **must** have been created with `VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT`
- VUID-VkRenderingInfo-pDepthAttachment-06547
  If `pDepthAttachment` is not `NULL` and `pDepthAttachment->imageView` is not VK_NULL_HANDLE, `pDepthAttachment->imageView` **must** have been created with a format that includes a depth component
- VUID-VkRenderingInfo-pDepthAttachment-06088
  If `pDepthAttachment` is not `NULL` and `pDepthAttachment->imageView` is not VK_NULL_HANDLE, `pDepthAttachment->imageView` **must** have been created with `VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT`
- VUID-VkRenderingInfo-pDepthAttachment-09477
  If `pDepthAttachment` is not `NULL` and `pDepthAttachment->resolveMode` is not `VK_RESOLVE_MODE_NONE`, `pDepthAttachment->resolveImageView` **must** have been created with `VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT`
- VUID-VkRenderingInfo-pStencilAttachment-06548
  If `pStencilAttachment` is not `NULL` and `pStencilAttachment->imageView` is not VK_NULL_HANDLE, `pStencilAttachment->imageView` **must** have been created with a format that includes a stencil aspect
- VUID-VkRenderingInfo-pStencilAttachment-06089
  If `pStencilAttachment` is not `NULL` and `pStencilAttachment->imageView` is not VK_NULL_HANDLE, `pStencilAttachment->imageView` **must** have been created with a stencil usage including `VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT`
- VUID-VkRenderingInfo-pStencilAttachment-09478

If pStencilAttachment is not NULL and pStencilAttachment->resolveMode is not VK_RESOLVE_MODE_NONE, pStencilAttachment->resolveImageView **must** have been created with VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT

- VUID-VkRenderingInfo-colorAttachmentCount-06090
  If colorAttachmentCount is not 0 and the imageView member of an element of pColorAttachments is not VK_NULL_HANDLE, the layout member of that element of pColorAttachments **must** not be VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL or VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL
- VUID-VkRenderingInfo-colorAttachmentCount-06091
  If colorAttachmentCount is not 0 and the imageView member of an element of pColorAttachments is not VK_NULL_HANDLE, if the resolveMode member of that element of pColorAttachments is not VK_RESOLVE_MODE_NONE, its resolveImageLayout member **must** not be VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL or VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL
- VUID-VkRenderingInfo-pDepthAttachment-06092
  If pDepthAttachment is not NULL and pDepthAttachment->imageView is not VK_NULL_HANDLE, pDepthAttachment->layout **must** not be VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL
- VUID-VkRenderingInfo-pDepthAttachment-06093
  If pDepthAttachment is not NULL, pDepthAttachment->imageView is not VK_NULL_HANDLE, and pDepthAttachment->resolveMode is not VK_RESOLVE_MODE_NONE, pDepthAttachment->resolveImageLayout **must** not be VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL
- VUID-VkRenderingInfo-pStencilAttachment-06094
  If pStencilAttachment is not NULL and pStencilAttachment->imageView is not VK_NULL_HANDLE, pStencilAttachment->layout **must** not be VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL
- VUID-VkRenderingInfo-pStencilAttachment-06095
  If pStencilAttachment is not NULL, pStencilAttachment->imageView is not VK_NULL_HANDLE, and pStencilAttachment->resolveMode is not VK_RESOLVE_MODE_NONE, pStencilAttachment->resolveImageLayout **must** not be VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL
- VUID-VkRenderingInfo-colorAttachmentCount-06096
  If colorAttachmentCount is not 0 and the imageView member of an element of pColorAttachments is not VK_NULL_HANDLE, the layout member of that element of pColorAttachments **must** not be VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_STENCIL_ATTACHMENT_OPTIMAL or VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_STENCIL_READ_ONLY_OPTIMAL
- VUID-VkRenderingInfo-colorAttachmentCount-06097
  If colorAttachmentCount is not 0 and the imageView member of an element of pColorAttachments is not VK_NULL_HANDLE, if the resolveMode member of that element of pColorAttachments is not VK_RESOLVE_MODE_NONE, its resolveImageLayout member **must** not be VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_STENCIL_ATTACHMENT_OPTIMAL or VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_STENCIL_READ_ONLY_OPTIMAL
- VUID-VkRenderingInfo-pDepthAttachment-06098
  If pDepthAttachment is not NULL, pDepthAttachment->imageView is not VK_NULL_HANDLE, and pDepthAttachment->resolveMode is not VK_RESOLVE_MODE_NONE, pDepthAttachment-

>resolveImageLayout **must** not be VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_STENCIL_ATTACHMENT_OPTIMAL

- VUID-VkRenderingInfo-pStencilAttachment-06099
  If pStencilAttachment is not NULL, pStencilAttachment->imageView is not VK_NULL_HANDLE, and pStencilAttachment->resolveMode is not VK_RESOLVE_MODE_NONE, pStencilAttachment->resolveImageLayout **must** not be VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_STENCIL_READ_ONLY_OPTIMAL
- VUID-VkRenderingInfo-colorAttachmentCount-06100
  If colorAttachmentCount is not 0 and the imageView member of an element of pColorAttachments is not VK_NULL_HANDLE, the layout member of that element of pColorAttachments **must** not be VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL, VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL, VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL, or VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL
- VUID-VkRenderingInfo-colorAttachmentCount-06101
  If colorAttachmentCount is not 0 and the imageView member of an element of pColorAttachments is not VK_NULL_HANDLE, if the resolveMode member of that element of pColorAttachments is not VK_RESOLVE_MODE_NONE, its resolveImageLayout member **must** not be VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL, VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL, VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL, or VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL
- VUID-VkRenderingInfo-pDepthAttachment-07732
  If pDepthAttachment is not NULL and pDepthAttachment->imageView is not VK_NULL_HANDLE, pDepthAttachment->layout **must** not be VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL or VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL
- VUID-VkRenderingInfo-pDepthAttachment-07733
  If pDepthAttachment is not NULL, pDepthAttachment->imageView is not VK_NULL_HANDLE, and pDepthAttachment->resolveMode is not VK_RESOLVE_MODE_NONE, pDepthAttachment->resolveImageLayout **must** not be VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL or VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL
- VUID-VkRenderingInfo-pStencilAttachment-07734
  If pStencilAttachment is not NULL and pStencilAttachment->imageView is not VK_NULL_HANDLE, pStencilAttachment->layout **must** not be VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL or VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL
- VUID-VkRenderingInfo-pStencilAttachment-07735
  If pStencilAttachment is not NULL, pStencilAttachment->imageView is not VK_NULL_HANDLE, and pStencilAttachment->resolveMode is not VK_RESOLVE_MODE_NONE, pStencilAttachment->resolveImageLayout **must** not be VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL or VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL
- VUID-VkRenderingInfo-pDepthAttachment-06102
  If pDepthAttachment is not NULL and pDepthAttachment->imageView is not VK_NULL_HANDLE, pDepthAttachment->resolveMode **must** be one of the bits set in VkPhysicalDeviceDepthStencilResolveProperties::supportedDepthResolveModes
- VUID-VkRenderingInfo-pStencilAttachment-06103
  If pStencilAttachment is not NULL and pStencilAttachment->imageView is not

VK_NULL_HANDLE, pStencilAttachment->resolveMode **must** be one of the bits set in VkPhysicalDeviceDepthStencilResolveProperties::supportedStencilResolveModes

- VUID-VkRenderingInfo-pDepthAttachment-06104
  If pDepthAttachment or pStencilAttachment are both not NULL, pDepthAttachment->imageView and pStencilAttachment->imageView are both not VK_NULL_HANDLE, and VkPhysicalDeviceDepthStencilResolveProperties::independentResolveNone is VK_FALSE, the resolveMode of both structures **must** be the same value
- VUID-VkRenderingInfo-pDepthAttachment-06105
  If pDepthAttachment or pStencilAttachment are both not NULL, pDepthAttachment->imageView and pStencilAttachment->imageView are both not VK_NULL_HANDLE, VkPhysicalDeviceDepthStencilResolveProperties::independentResolve is VK_FALSE, and the resolveMode of neither structure is VK_RESOLVE_MODE_NONE, the resolveMode of both structures **must** be the same value
- VUID-VkRenderingInfo-colorAttachmentCount-06106
  colorAttachmentCount **must** be less than or equal to VkPhysicalDeviceLimits::maxColorAttachments
- VUID-VkRenderingInfo-imageView-06107
  If the imageView member of a VkRenderingFragmentDensityMapAttachmentInfoEXT structure included in the pNext chain is not VK_NULL_HANDLE, and the fragmentDensityMapNonSubsampledImages feature is not enabled, valid imageView and resolveImageView members of pDepthAttachment, pStencilAttachment, and each element of pColorAttachments **must** be a VkImageView created with VK_IMAGE_CREATE_SUBSAMPLED_BIT_EXT
- VUID-VkRenderingInfo-imageView-06108
  If the imageView member of a VkRenderingFragmentDensityMapAttachmentInfoEXT structure included in the pNext chain is not VK_NULL_HANDLE, and viewMask is not 0, imageView **must** have a layerCount greater than or equal to the index of the most significant bit in viewMask
- VUID-VkRenderingInfo-imageView-06109
  If the imageView member of a VkRenderingFragmentDensityMapAttachmentInfoEXT structure included in the pNext chain is not VK_NULL_HANDLE, and viewMask is 0, imageView **must** have a layerCount equal to 1
- VUID-VkRenderingInfo-pNext-06112
  If the pNext chain does not contain VkDeviceGroupRenderPassBeginInfo or its deviceRenderAreaCount member is equal to 0 and the imageView member of a VkRenderingFragmentDensityMapAttachmentInfoEXT structure included in the pNext chain is not VK_NULL_HANDLE, imageView **must** have a width greater than or equal to $\lceil \frac{renderArea_x + renderArea_{width}}{maxFragmentDensityTexelSize_{width}} \rceil$
- VUID-VkRenderingInfo-pNext-06114
  If the pNext chain does not contain VkDeviceGroupRenderPassBeginInfo or its deviceRenderAreaCount member is equal to 0 and the imageView member of a VkRenderingFragmentDensityMapAttachmentInfoEXT structure included in the pNext chain is not VK_NULL_HANDLE, imageView **must** have a height greater than or equal to $\lceil \frac{renderArea_y + renderArea_{height}}{maxFragmentDensityTexelSize_{height}} \rceil$

- VUID-VkRenderingInfo-pNext-06113
  If the pNext chain contains a VkDeviceGroupRenderPassBeginInfo structure, its deviceRenderAreaCount member is not 0, and the imageView member of a VkRenderingFragmentDensityMapAttachmentInfoEXT structure included in the pNext chain is not VK_NULL_HANDLE, imageView **must** have a width greater than or equal to $\lceil \frac{pDeviceRenderAreas_{x} + pDeviceRenderAreas_{width}}{maxFragmentDensityTexelSize_{width}} \rceil$ for each element of pDeviceRenderAreas
- VUID-VkRenderingInfo-pNext-06115
  If the pNext chain contains a VkDeviceGroupRenderPassBeginInfo structure, its deviceRenderAreaCount member is not 0, and the imageView member of a VkRenderingFragmentDensityMapAttachmentInfoEXT structure included in the pNext chain is not VK_NULL_HANDLE, imageView **must** have a height greater than or equal to $\lceil \frac{pDeviceRenderAreas_{y} + pDeviceRenderAreas_{height}}{maxFragmentDensityTexelSize_{height}} \rceil$ for each element of pDeviceRenderAreas
- VUID-VkRenderingInfo-imageView-06116
  If the imageView member of a VkRenderingFragmentDensityMapAttachmentInfoEXT structure included in the pNext chain is not VK_NULL_HANDLE, it **must** not be equal to the imageView or resolveImageView member of pDepthAttachment, pStencilAttachment, or any element of pColorAttachments
- VUID-VkRenderingInfo-pNext-06119
  If maintenance7 is not enabled or the robustFragmentShadingRateAttachmentAccess limit is VK_FALSE or the imageView member of a VkRenderingFragmentShadingRateAttachmentInfoKHR structure was created with VkImageSubresourceRange::baseMipLevel greater than 0, the pNext chain does not contain VkDeviceGroupRenderPassBeginInfo or its deviceRenderAreaCount member is equal to 0, and the imageView member of a VkRenderingFragmentShadingRateAttachmentInfoKHR structure included in the pNext chain is not VK_NULL_HANDLE, imageView **must** have a width greater than or equal to $\lceil \frac{renderArea_{x} + renderArea_{width}}{shadingRateAttachmentTexelSize_{width}} \rceil$
- VUID-VkRenderingInfo-pNext-06121
  If maintenance7 is not enabled or the robustFragmentShadingRateAttachmentAccess limit is VK_FALSE or the imageView member of a VkRenderingFragmentShadingRateAttachmentInfoKHR structure was created with VkImageSubresourceRange::baseMipLevel greater than 0, the pNext chain does not contain VkDeviceGroupRenderPassBeginInfo or its deviceRenderAreaCount member is equal to 0 and the imageView member of a VkRenderingFragmentShadingRateAttachmentInfoKHR structure included in the pNext chain is not VK_NULL_HANDLE, imageView **must** have a height greater than or equal to $\lceil \frac{renderArea_{y} + renderArea_{height}}{shadingRateAttachmentTexelSize_{height}} \rceil$
- VUID-VkRenderingInfo-pNext-06120
  If maintenance7 is not enabled or the robustFragmentShadingRateAttachmentAccess limit is VK_FALSE or the imageView member of a VkRenderingFragmentShadingRateAttachmentInfoKHR structure was created with VkImageSubresourceRange::baseMipLevel greater than 0, the pNext chain contains a VkDeviceGroupRenderPassBeginInfo structure, its deviceRenderAreaCount member is not 0, and the imageView member of a VkRenderingFragmentShadingRateAttachmentInfoKHR structure included in the pNext chain is not VK_NULL_HANDLE, imageView **must** have a width greater than or equal to $\lceil \frac{pDeviceRenderAreas_{x} + pDeviceRenderAreas_{width}}{shadingRateAttachmentTexelSize_{width}} \rceil$ for each element of pDeviceRenderAreas

- VUID-VkRenderingInfo-pNext-06122
  If `maintenance7` is not enabled or the `robustFragmentShadingRateAttachmentAccess` limit is `VK_FALSE` or the `imageView` member of a VkRenderingFragmentShadingRateAttachmentInfoKHR structure was created with VkImageSubresourceRange::`baseMipLevel` greater than 0, the `pNext` chain contains a VkDeviceGroupRenderPassBeginInfo structure, its `deviceRenderAreaCount` member is not 0, and the `imageView` member of a VkRenderingFragmentShadingRateAttachmentInfoKHR structure included in the `pNext` chain is not VK_NULL_HANDLE, `imageView` **must** have a height greater than or equal to $\lceil \frac{pDeviceRenderAreas_{y} + pDeviceRenderAreas_{height}}{shadingRateAttachmentTexelSize_{height}} \rceil$ for each element of `pDeviceRenderAreas`
- VUID-VkRenderingInfo-layerCount-07817
  `layerCount` **must** be less than or equal to `maxFramebufferLayers`
- VUID-VkRenderingInfo-imageView-06123
  If the `imageView` member of a VkRenderingFragmentShadingRateAttachmentInfoKHR structure included in the `pNext` chain is not VK_NULL_HANDLE, and `viewMask` is `0`, `imageView` **must** have a `layerCount` that is either equal to `1` or greater than or equal to `layerCount`
- VUID-VkRenderingInfo-imageView-06124
  If the `imageView` member of a VkRenderingFragmentShadingRateAttachmentInfoKHR structure included in the `pNext` chain is not VK_NULL_HANDLE, and `viewMask` is not `0`, `imageView` **must** have a `layerCount` that either equal to `1` or greater than or equal to the index of the most significant bit in `viewMask`
- VUID-VkRenderingInfo-imageView-06125
  If the `imageView` member of a VkRenderingFragmentShadingRateAttachmentInfoKHR structure included in the `pNext` chain is not VK_NULL_HANDLE, it **must** not be equal to the `imageView` or `resolveImageView` member of `pDepthAttachment`, `pStencilAttachment`, or any element of `pColorAttachments`
- VUID-VkRenderingInfo-imageView-06126
  If the `imageView` member of a VkRenderingFragmentShadingRateAttachmentInfoKHR structure included in the `pNext` chain is not VK_NULL_HANDLE, it **must** not be equal to the `imageView` member of a VkRenderingFragmentDensityMapAttachmentInfoEXT structure included in the `pNext` chain
- VUID-VkRenderingInfo-multiview-06127
  If the `multiview` feature is not enabled, `viewMask` **must** be `0`
- VUID-VkRenderingInfo-viewMask-06128
  The index of the most significant bit in `viewMask` **must** be less than `maxMultiviewViewCount`
- VUID-VkRenderingInfo-perViewRenderAreaCount-07857
  If the `perViewRenderAreaCount` member of a VkMultiviewPerViewRenderAreasRenderPassBeginInfoQCOM structure included in the `pNext` chain is not `0`, then the `multiviewPerViewRenderAreas` feature **must** be enabled
- VUID-VkRenderingInfo-perViewRenderAreaCount-07858
  If the `perViewRenderAreaCount` member of a VkMultiviewPerViewRenderAreasRenderPassBeginInfoQCOM structure included in the `pNext` chain is not `0`, then `renderArea` **must** specify a render area that includes the union of

all per view render areas

- VUID-VkRenderingInfo-None-09044
  Valid attachments specified by this structure **must** not be bound to memory locations that are bound to any other valid attachments specified by this structure
- VUID-VkRenderingInfo-flags-10012
  If `flags` includes `VK_RENDERING_CONTENTS_INLINE_BIT_KHR` then at least one of the following features **must** be enabled
  - `maintenance7`
  - `nestedCommandBuffer`
- VUID-VkRenderingInfo-pDepthAttachment-09318
  `pDepthAttachment->resolveMode` **must** not be `VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID`
- VUID-VkRenderingInfo-pStencilAttachment-09319
  `pStencilAttachment->resolveMode` **must** not be `VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID`
- VUID-VkRenderingInfo-colorAttachmentCount-09320
  If `colorAttachmentCount` is not `1`, the `resolveMode` member of any element of `pColorAttachments` **must** not be `VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID`
- VUID-VkRenderingInfo-resolveMode-09321
  If the `resolveMode` of any element of `pColorAttachments` is `VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID`, VkRenderingFragmentDensityMapAttachmentInfoEXT::`imageView` **must** be VK_NULL_HANDLE
- VUID-VkRenderingInfo-resolveMode-09322
  If the `resolveMode` of any element of `pColorAttachments` is `VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID`, VkRenderingFragmentShadingRateAttachmentInfoKHR::`imageView` **must** be VK_NULL_HANDLE
- VUID-VkRenderingInfo-pNext-09535
  If the `pNext` chain contains a VkRenderPassStripeBeginInfoARM structure, the union of stripe areas defined by the elements of VkRenderPassStripeInfoARM::`pStripeInfos` **must** cover the `renderArea`
- VUID-VkRenderingInfo-colorAttachmentCount-09479
  If `colorAttachmentCount` is not `0` and the `imageView` member of an element of `pColorAttachments` is not VK_NULL_HANDLE, that `imageView` **must** have been created with the identity swizzle
- VUID-VkRenderingInfo-colorAttachmentCount-09480
  If `colorAttachmentCount` is not `0`, and there is an element of `pColorAttachments` with either its `resolveMode` member set to `VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID`, or its `imageView` member not set to VK_NULL_HANDLE and its `resolveMode` member not set to `VK_RESOLVE_MODE_NONE`, the `resolveImageView` member of that element of `pColorAttachments` **must** have been created with the identity swizzle
- VUID-VkRenderingInfo-pDepthAttachment-09481

If pDepthAttachment is not NULL and pDepthAttachment->imageView is not VK_NULL_HANDLE, pDepthAttachment->imageView **must** have been created with the identity swizzle

- VUID-VkRenderingInfo-pDepthAttachment-09482
  If pDepthAttachment is not NULL, pDepthAttachment->imageView is not VK_NULL_HANDLE, and pDepthAttachment->resolveMode is not VK_RESOLVE_MODE_NONE, pDepthAttachment->resolveImageView **must** have been created with the identity swizzle
- VUID-VkRenderingInfo-pStencilAttachment-09483
  If pStencilAttachment is not NULL and pStencilAttachment->imageView is not VK_NULL_HANDLE, pStencilAttachment->imageView **must** have been created with the identity swizzle
- VUID-VkRenderingInfo-pStencilAttachment-09484
  If pStencilAttachment is not NULL, pStencilAttachment->imageView is not VK_NULL_HANDLE, and pStencilAttachment->resolveMode is not VK_RESOLVE_MODE_NONE, pStencilAttachment->resolveImageView **must** have been created with the identity swizzle
- VUID-VkRenderingInfo-imageView-09485
  If the imageView member of a VkRenderingFragmentShadingRateAttachmentInfoKHR structure included in the pNext chain is not VK_NULL_HANDLE, it **must** have been created with the identity swizzle
- VUID-VkRenderingInfo-imageView-09486
  If the imageView member of a VkRenderingFragmentDensityMapAttachmentInfoEXT structure included in the pNext chain is not VK_NULL_HANDLE, it **must** have been created with the identity swizzle

## Valid Usage (Implicit)

- VUID-VkRenderingInfo-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_RENDERING_INFO
- VUID-VkRenderingInfo-pNext-pNext
  Each pNext member of any structure (including this one) in the pNext chain **must** be either NULL or a pointer to a valid instance of VkDeviceGroupRenderPassBeginInfo, VkMultisampledRenderToSingleSampledInfoEXT, VkMultiviewPerViewAttributesInfoNVX, VkMultiviewPerViewRenderAreasRenderPassBeginInfoQCOM, VkRenderPassStripeBeginInfoARM, VkRenderingFragmentDensityMapAttachmentInfoEXT, or VkRenderingFragmentShadingRateAttachmentInfoKHR
- VUID-VkRenderingInfo-sType-unique
  The sType value of each struct in the pNext chain **must** be unique
- VUID-VkRenderingInfo-flags-parameter
  flags **must** be a valid combination of VkRenderingFlagBits values
- VUID-VkRenderingInfo-pColorAttachments-parameter
  If colorAttachmentCount is not 0, pColorAttachments **must** be a valid pointer to an array of colorAttachmentCount valid VkRenderingAttachmentInfo structures

- VUID-VkRenderingInfo-pDepthAttachment-parameter
  If `pDepthAttachment` is not `NULL`, `pDepthAttachment` **must** be a valid pointer to a valid VkRenderingAttachmentInfo structure
- VUID-VkRenderingInfo-pStencilAttachment-parameter
  If `pStencilAttachment` is not `NULL`, `pStencilAttachment` **must** be a valid pointer to a valid VkRenderingAttachmentInfo structure

Bits which **can** be set in VkRenderingInfo::`flags` describing additional properties of the render pass are:

```
typedef enum VkRenderingFlagBits {
    VK_RENDERING_CONTENTS_SECONDARY_COMMAND_BUFFERS_BIT = 0x00000001,
    VK_RENDERING_SUSPENDING_BIT = 0x00000002,
    VK_RENDERING_RESUMING_BIT = 0x00000004,
  // Provided by VK_EXT_legacy_dithering with (VK_KHR_dynamic_rendering or
VK_VERSION_1_3) and VK_KHR_maintenance5
    VK_RENDERING_ENABLE_LEGACY_DITHERING_BIT_EXT = 0x00000008,
  // Provided by VK_KHR_maintenance7
    VK_RENDERING_CONTENTS_INLINE_BIT_KHR = 0x00000010,
    VK_RENDERING_CONTENTS_SECONDARY_COMMAND_BUFFERS_BIT_KHR =
VK_RENDERING_CONTENTS_SECONDARY_COMMAND_BUFFERS_BIT,
    VK_RENDERING_SUSPENDING_BIT_KHR = VK_RENDERING_SUSPENDING_BIT,
    VK_RENDERING_RESUMING_BIT_KHR = VK_RENDERING_RESUMING_BIT,
  // Provided by VK_EXT_nested_command_buffer
    VK_RENDERING_CONTENTS_INLINE_BIT_EXT = VK_RENDERING_CONTENTS_INLINE_BIT_KHR,
} VkRenderingFlagBits;
```

or the equivalent

```
// Provided by VK_KHR_dynamic_rendering
typedef VkRenderingFlagBits VkRenderingFlagBitsKHR;
```

- `VK_RENDERING_CONTENTS_SECONDARY_COMMAND_BUFFERS_BIT` specifies that draw calls for the render pass instance will be recorded in secondary command buffers. If the `nestedCommandBuffer` feature is enabled, the draw calls **can** come from both inline and vkCmdExecuteCommands.
- `VK_RENDERING_RESUMING_BIT` specifies that the render pass instance is resuming an earlier suspended render pass instance.
- `VK_RENDERING_SUSPENDING_BIT` specifies that the render pass instance will be suspended.
- `VK_RENDERING_ENABLE_LEGACY_DITHERING_BIT_EXT` specifies that Legacy Dithering is enabled for the render pass instance.
- `VK_RENDERING_CONTENTS_INLINE_BIT_KHR` specifies that draw calls for the render pass instance **can** be recorded inline within the current command buffer. This **can** be combined with the `VK_RENDERING_CONTENTS_SECONDARY_COMMAND_BUFFERS_BIT` bit to allow draw calls to be recorded both inline and in secondary command buffers.

The contents of pRenderingInfo **must** match between suspended render pass instances and the render pass instances that resume them, other than the presence or absence of the VK_RENDERING_RESUMING_BIT, VK_RENDERING_SUSPENDING_BIT, and VK_RENDERING_CONTENTS_SECONDARY_COMMAND_BUFFERS_BIT flags. No action or synchronization commands, or other render pass instances, are allowed between suspending and resuming render pass instances.

```
typedef VkFlags VkRenderingFlags;
```

or the equivalent

```
// Provided by VK_KHR_dynamic_rendering
typedef VkRenderingFlags VkRenderingFlagsKHR;
```

VkRenderingFlags is a bitmask type for setting a mask of zero or more VkRenderingFlagBits.

The VkRenderingAttachmentInfo structure is defined as:

```
typedef struct VkRenderingAttachmentInfo {
    VkStructureType          sType;
    const void*              pNext;
    VkImageView              imageView;
    VkImageLayout            imageLayout;
    VkResolveModeFlagBits    resolveMode;
    VkImageView              resolveImageView;
    VkImageLayout            resolveImageLayout;
    VkAttachmentLoadOp       loadOp;
    VkAttachmentStoreOp      storeOp;
    VkClearValue             clearValue;
} VkRenderingAttachmentInfo;
```

or the equivalent

```
// Provided by VK_KHR_dynamic_rendering
typedef VkRenderingAttachmentInfo VkRenderingAttachmentInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- imageView is the image view that will be used for rendering.
- imageLayout is the layout that imageView will be in during rendering.
- resolveMode is a VkResolveModeFlagBits value defining how data written to imageView will be resolved into resolveImageView.
- resolveImageView is an image view used to write resolved data at the end of rendering.

- resolveImageLayout is the layout that resolveImageView will be in during rendering.
- loadOp is a VkAttachmentLoadOp value defining the load operation for the attachment.
- storeOp is a VkAttachmentStoreOp value defining the store operation for the attachment.
- clearValue is a VkClearValue structure defining values used to clear imageView when loadOp is VK_ATTACHMENT_LOAD_OP_CLEAR.

Values in imageView are loaded and stored according to the values of loadOp and storeOp, within the render area for each device specified in VkRenderingInfo. If imageView is VK_NULL_HANDLE, and resolveMode is not VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID, other members of this structure are ignored; writes to this attachment will be discarded, and no load, store, or multisample resolve operations will be performed.

If resolveMode is VK_RESOLVE_MODE_NONE, then resolveImageView is ignored. If resolveMode is not VK_RESOLVE_MODE_NONE, and resolveImageView is not VK_NULL_HANDLE, a render pass multisample resolve operation is defined for the attachment subresource. If resolveMode is VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID, and the nullColorAttachmentWithExternalFormatResolve limit is VK_TRUE, values are only undefined once load operations have completed.

| | |
|---|---|
| **NOTE** | The resolve mode and store operation are independent; it is valid to write both resolved and unresolved values, and equally valid to discard the unresolved values while writing the resolved ones. |

Store and resolve operations are only performed at the end of a render pass instance that does not specify the VK_RENDERING_SUSPENDING_BIT_KHR flag.

Load operations are only performed at the beginning of a render pass instance that does not specify the VK_RENDERING_RESUMING_BIT_KHR flag.

Image contents at the end of a suspended render pass instance remain defined for access by a resuming render pass instance.

If the nullColorAttachmentWithExternalFormatResolve limit is VK_TRUE, and resolveMode is VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID, values in the color attachment will be loaded from the resolve attachment at the start of rendering, and **may** also be reloaded any time after a resolve occurs or the resolve attachment is written to; if this occurs it **must** happen-before any writes to the color attachment are performed which happen-after the resolve that triggers this. If any color component in the external format is subsampled, values will be read from the nearest sample in the image when they are loaded.

### Valid Usage

- VUID-VkRenderingAttachmentInfo-imageView-06129
  If imageView is not VK_NULL_HANDLE and has a non-integer color format, resolveMode **must** be VK_RESOLVE_MODE_NONE or VK_RESOLVE_MODE_AVERAGE_BIT
- VUID-VkRenderingAttachmentInfo-imageView-06130
  If imageView is not VK_NULL_HANDLE and has an integer color format, resolveMode **must**

be VK_RESOLVE_MODE_NONE or VK_RESOLVE_MODE_SAMPLE_ZERO_BIT

- VUID-VkRenderingAttachmentInfo-imageView-06861
  imageView **must** not have a sample count of VK_SAMPLE_COUNT_1_BIT if all of the following hold:
  - imageView is not VK_NULL_HANDLE
  - resolveMode is not VK_RESOLVE_MODE_NONE
  - the pNext chain of VkRenderingInfo does not include a VkMultisampledRenderToSingleSampledInfoEXT structure with the multisampledRenderToSingleSampledEnable field equal to VK_TRUE
- VUID-VkRenderingAttachmentInfo-imageView-06862
  resolveImageView **must** not be VK_NULL_HANDLE if all of the following hold:
  - imageView is not VK_NULL_HANDLE
  - resolveMode is not VK_RESOLVE_MODE_NONE
  - the pNext chain of VkRenderingInfo does not include a VkMultisampledRenderToSingleSampledInfoEXT structure with the multisampledRenderToSingleSampledEnable field equal to VK_TRUE
- VUID-VkRenderingAttachmentInfo-imageView-06863
  If imageView is not VK_NULL_HANDLE, resolveMode is not VK_RESOLVE_MODE_NONE, the pNext chain of VkRenderingInfo includes a VkMultisampledRenderToSingleSampledInfoEXT structure with the multisampledRenderToSingleSampledEnable field equal to VK_TRUE, and imageView has a sample count of VK_SAMPLE_COUNT_1_BIT, resolveImageView **must** be VK_NULL_HANDLE
- VUID-VkRenderingAttachmentInfo-imageView-06864
  If imageView is not VK_NULL_HANDLE, resolveImageView is not VK_NULL_HANDLE, and resolveMode is not VK_RESOLVE_MODE_NONE, resolveImageView **must** have a sample count of VK_SAMPLE_COUNT_1_BIT
- VUID-VkRenderingAttachmentInfo-imageView-06865
  If imageView is not VK_NULL_HANDLE, resolveImageView is not VK_NULL_HANDLE, and resolveMode is not VK_RESOLVE_MODE_NONE, imageView and resolveImageView **must** have the same VkFormat
- VUID-VkRenderingAttachmentInfo-imageView-06135
  If imageView is not VK_NULL_HANDLE, imageLayout **must** not be VK_IMAGE_LAYOUT_UNDEFINED, VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL, VK_IMAGE_LAYOUT_TRANSFER_SRC_OPTIMAL, VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL, or VK_IMAGE_LAYOUT_PREINITIALIZED
- VUID-VkRenderingAttachmentInfo-imageView-06136
  If imageView is not VK_NULL_HANDLE and resolveMode is not VK_RESOLVE_MODE_NONE, resolveImageLayout **must** not be VK_IMAGE_LAYOUT_UNDEFINED, VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL, VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL, VK_IMAGE_LAYOUT_TRANSFER_SRC_OPTIMAL, VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL, or VK_IMAGE_LAYOUT_PREINITIALIZED
- VUID-VkRenderingAttachmentInfo-imageView-06137
  If imageView is not VK_NULL_HANDLE and resolveMode is not VK_RESOLVE_MODE_NONE,

resolveImageLayout **must** not be VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL or VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL

- VUID-VkRenderingAttachmentInfo-imageView-06138
  If imageView is not VK_NULL_HANDLE, imageLayout **must** not be VK_IMAGE_LAYOUT_SHADING_RATE_OPTIMAL_NV
- VUID-VkRenderingAttachmentInfo-imageView-06139
  If imageView is not VK_NULL_HANDLE and resolveMode is not VK_RESOLVE_MODE_NONE, resolveImageLayout **must** not be VK_IMAGE_LAYOUT_SHADING_RATE_OPTIMAL_NV
- VUID-VkRenderingAttachmentInfo-imageView-06140
  If imageView is not VK_NULL_HANDLE, imageLayout **must** not be VK_IMAGE_LAYOUT_FRAGMENT_DENSITY_MAP_OPTIMAL_EXT
- VUID-VkRenderingAttachmentInfo-imageView-06141
  If imageView is not VK_NULL_HANDLE and resolveMode is not VK_RESOLVE_MODE_NONE, resolveImageLayout **must** not be VK_IMAGE_LAYOUT_FRAGMENT_DENSITY_MAP_OPTIMAL_EXT
- VUID-VkRenderingAttachmentInfo-imageView-06142
  If imageView is not VK_NULL_HANDLE and resolveMode is not VK_RESOLVE_MODE_NONE, resolveImageLayout **must** not be VK_IMAGE_LAYOUT_READ_ONLY_OPTIMAL_KHR
- VUID-VkRenderingAttachmentInfo-imageView-06143
  If imageView is not VK_NULL_HANDLE, imageLayout **must** not be VK_IMAGE_LAYOUT_FRAGMENT_SHADING_RATE_ATTACHMENT_OPTIMAL_KHR
- VUID-VkRenderingAttachmentInfo-imageView-06144
  If imageView is not VK_NULL_HANDLE and resolveMode is not VK_RESOLVE_MODE_NONE, resolveImageLayout **must** not be VK_IMAGE_LAYOUT_FRAGMENT_SHADING_RATE_ATTACHMENT_OPTIMAL_KHR
- VUID-VkRenderingAttachmentInfo-imageView-06145
  If imageView is not VK_NULL_HANDLE, imageLayout **must** not be VK_IMAGE_LAYOUT_PRESENT_SRC_KHR
- VUID-VkRenderingAttachmentInfo-imageView-06146
  If imageView is not VK_NULL_HANDLE and resolveMode is not VK_RESOLVE_MODE_NONE, resolveImageLayout **must** not be VK_IMAGE_LAYOUT_PRESENT_SRC_KHR
- VUID-VkRenderingAttachmentInfo-externalFormatResolve-09323
  If externalFormatResolve is not enabled, resolveMode **must** not be VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID
- VUID-VkRenderingAttachmentInfo-resolveMode-09324
  If resolveMode is VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID, resolveImageView **must** be a valid image view
- VUID-VkRenderingAttachmentInfo-nullColorAttachmentWithExternalFormatResolve-09325
  If the nullColorAttachmentWithExternalFormatResolve property is VK_TRUE and resolveMode is VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID, resolveImageView **must** have been created with an image with a samples value of VK_SAMPLE_COUNT_1_BIT
- VUID-VkRenderingAttachmentInfo-resolveMode-09326
  If resolveMode is VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID, resolveImageView

**must** have been created with an external format specified by VkExternalFormatANDROID

- VUID-VkRenderingAttachmentInfo-resolveMode-09327
  If `resolveMode` is `VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID`, `resolveImageView` **must** have been created with a `subresourceRange.layerCount` of `1`
- VUID-VkRenderingAttachmentInfo-resolveMode-09328
  If `resolveMode` is `VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID` and `nullColorAttachmentWithExternalFormatResolve` is `VK_TRUE`, `imageView` **must** be VK_NULL_HANDLE
- VUID-VkRenderingAttachmentInfo-resolveMode-09329
  If `resolveMode` is `VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID` and `nullColorAttachmentWithExternalFormatResolve` is `VK_FALSE`, `imageView` **must** be a valid VkImageView
- VUID-VkRenderingAttachmentInfo-resolveMode-09330
  If `resolveMode` is `VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID` and `nullColorAttachmentWithExternalFormatResolve` is `VK_FALSE`, `imageView` **must** have a format equal to the value of VkAndroidHardwareBufferFormatResolvePropertiesANDROID::`colorAttachmentFormat` as returned by a call to vkGetAndroidHardwareBufferPropertiesANDROID for the Android hardware buffer that was used to create `resolveImageView`

## Valid Usage (Implicit)

- VUID-VkRenderingAttachmentInfo-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_RENDERING_ATTACHMENT_INFO`
- VUID-VkRenderingAttachmentInfo-pNext-pNext
  `pNext` **must** be `NULL`
- VUID-VkRenderingAttachmentInfo-imageView-parameter
  If `imageView` is not VK_NULL_HANDLE, `imageView` **must** be a valid VkImageView handle
- VUID-VkRenderingAttachmentInfo-imageLayout-parameter
  `imageLayout` **must** be a valid VkImageLayout value
- VUID-VkRenderingAttachmentInfo-resolveMode-parameter
  If `resolveMode` is not `0`, `resolveMode` **must** be a valid VkResolveModeFlagBits value
- VUID-VkRenderingAttachmentInfo-resolveImageView-parameter
  If `resolveImageView` is not VK_NULL_HANDLE, `resolveImageView` **must** be a valid VkImageView handle
- VUID-VkRenderingAttachmentInfo-resolveImageLayout-parameter
  `resolveImageLayout` **must** be a valid VkImageLayout value
- VUID-VkRenderingAttachmentInfo-loadOp-parameter
  `loadOp` **must** be a valid VkAttachmentLoadOp value
- VUID-VkRenderingAttachmentInfo-storeOp-parameter
  `storeOp` **must** be a valid VkAttachmentStoreOp value
- VUID-VkRenderingAttachmentInfo-clearValue-parameter

clearValue **must** be a valid VkClearValue union

- VUID-VkRenderingAttachmentInfo-commonparent
  Both of imageView, and resolveImageView that are valid handles of non-ignored parameters **must** have been created, allocated, or retrieved from the same VkDevice

The VkRenderingFragmentShadingRateAttachmentInfoKHR structure is defined as:

```
// Provided by VK_KHR_dynamic_rendering with VK_KHR_fragment_shading_rate
typedef struct VkRenderingFragmentShadingRateAttachmentInfoKHR {
    VkStructureType    sType;
    const void*        pNext;
    VkImageView        imageView;
    VkImageLayout      imageLayout;
    VkExtent2D         shadingRateAttachmentTexelSize;
} VkRenderingFragmentShadingRateAttachmentInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- imageView is the image view that will be used as a fragment shading rate attachment.
- imageLayout is the layout that imageView will be in during rendering.
- shadingRateAttachmentTexelSize specifies the number of pixels corresponding to each texel in imageView.

This structure can be included in the pNext chain of VkRenderingInfo to define a fragment shading rate attachment. If imageView is VK_NULL_HANDLE, or if this structure is not specified, the implementation behaves as if a valid shading rate attachment was specified with all texels specifying a single pixel per fragment.

### Valid Usage

- VUID-VkRenderingFragmentShadingRateAttachmentInfoKHR-imageView-06147
  If imageView is not VK_NULL_HANDLE, layout **must** be VK_IMAGE_LAYOUT_GENERAL or VK_IMAGE_LAYOUT_FRAGMENT_SHADING_RATE_ATTACHMENT_OPTIMAL_KHR
- VUID-VkRenderingFragmentShadingRateAttachmentInfoKHR-imageView-06148
  If imageView is not VK_NULL_HANDLE, it **must** have been created with VK_IMAGE_USAGE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR
- VUID-VkRenderingFragmentShadingRateAttachmentInfoKHR-imageView-06149
  If imageView is not VK_NULL_HANDLE, shadingRateAttachmentTexelSize.width **must** be a power of two value
- VUID-VkRenderingFragmentShadingRateAttachmentInfoKHR-imageView-06150
  If imageView is not VK_NULL_HANDLE, shadingRateAttachmentTexelSize.width **must** be less than or equal to maxFragmentShadingRateAttachmentTexelSize.width
- VUID-VkRenderingFragmentShadingRateAttachmentInfoKHR-imageView-06151

If imageView is not VK_NULL_HANDLE, shadingRateAttachmentTexelSize.width **must** be greater than or equal to minFragmentShadingRateAttachmentTexelSize.width

- VUID-VkRenderingFragmentShadingRateAttachmentInfoKHR-imageView-06152
  If imageView is not VK_NULL_HANDLE, shadingRateAttachmentTexelSize.height **must** be a power of two value
- VUID-VkRenderingFragmentShadingRateAttachmentInfoKHR-imageView-06153
  If imageView is not VK_NULL_HANDLE, shadingRateAttachmentTexelSize.height **must** be less than or equal to maxFragmentShadingRateAttachmentTexelSize.height
- VUID-VkRenderingFragmentShadingRateAttachmentInfoKHR-imageView-06154
  If imageView is not VK_NULL_HANDLE, shadingRateAttachmentTexelSize.height **must** be greater than or equal to minFragmentShadingRateAttachmentTexelSize.height
- VUID-VkRenderingFragmentShadingRateAttachmentInfoKHR-imageView-06155
  If imageView is not VK_NULL_HANDLE, the quotient of shadingRateAttachmentTexelSize.width and shadingRateAttachmentTexelSize.height **must** be less than or equal to maxFragmentShadingRateAttachmentTexelSizeAspectRatio
- VUID-VkRenderingFragmentShadingRateAttachmentInfoKHR-imageView-06156
  If imageView is not VK_NULL_HANDLE, the quotient of shadingRateAttachmentTexelSize.height and shadingRateAttachmentTexelSize.width **must** be less than or equal to maxFragmentShadingRateAttachmentTexelSizeAspectRatio

### Valid Usage (Implicit)

- VUID-VkRenderingFragmentShadingRateAttachmentInfoKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_RENDERING_FRAGMENT_SHADING_RATE_ATTACHMENT_INFO_KHR
- VUID-VkRenderingFragmentShadingRateAttachmentInfoKHR-imageView-parameter
  If imageView is not VK_NULL_HANDLE, imageView **must** be a valid VkImageView handle
- VUID-VkRenderingFragmentShadingRateAttachmentInfoKHR-imageLayout-parameter
  imageLayout **must** be a valid VkImageLayout value

The VkRenderingFragmentDensityMapAttachmentInfoEXT structure is defined as:

```
// Provided by VK_KHR_dynamic_rendering with VK_EXT_fragment_density_map
typedef struct VkRenderingFragmentDensityMapAttachmentInfoEXT {
    VkStructureType    sType;
    const void*        pNext;
    VkImageView        imageView;
    VkImageLayout      imageLayout;
} VkRenderingFragmentDensityMapAttachmentInfoEXT;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- imageView is the image view that will be used as a fragment density map attachment.

- imageLayout is the layout that imageView will be in during rendering.

This structure can be included in the pNext chain of VkRenderingInfo to define a fragment density map. If this structure is not included in the pNext chain, imageView is treated as VK_NULL_HANDLE.

### Valid Usage

- VUID-VkRenderingFragmentDensityMapAttachmentInfoEXT-imageView-06157
  If imageView is not VK_NULL_HANDLE, imageLayout **must** be VK_IMAGE_LAYOUT_GENERAL or VK_IMAGE_LAYOUT_FRAGMENT_DENSITY_MAP_OPTIMAL_EXT
- VUID-VkRenderingFragmentDensityMapAttachmentInfoEXT-imageView-06158
  If imageView is not VK_NULL_HANDLE, it **must** have been created with VK_IMAGE_USAGE_FRAGMENT_DENSITY_MAP_BIT_EXT
- VUID-VkRenderingFragmentDensityMapAttachmentInfoEXT-imageView-06159
  If imageView is not VK_NULL_HANDLE, it **must** not have been created with VK_IMAGE_CREATE_SUBSAMPLED_BIT_EXT
- VUID-VkRenderingFragmentDensityMapAttachmentInfoEXT-apiVersion-07908
  If the multiview feature is not enabled, VkPhysicalDeviceProperties::apiVersion is less than Vulkan 1.1, and imageView is not VK_NULL_HANDLE, it **must** have a layerCount equal to 1

### Valid Usage (Implicit)

- VUID-VkRenderingFragmentDensityMapAttachmentInfoEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_RENDERING_FRAGMENT_DENSITY_MAP_ATTACHMENT_INFO_EXT
- VUID-VkRenderingFragmentDensityMapAttachmentInfoEXT-imageView-parameter
  imageView **must** be a valid VkImageView handle
- VUID-VkRenderingFragmentDensityMapAttachmentInfoEXT-imageLayout-parameter
  imageLayout **must** be a valid VkImageLayout value

To query the render area granularity for a render pass instance, call:

```
// Provided by VK_KHR_maintenance5
void vkGetRenderingAreaGranularityKHR(
    VkDevice                                    device,
    const VkRenderingAreaInfoKHR*               pRenderingAreaInfo,
    VkExtent2D*                                 pGranularity);
```

- device is the logical device that owns the render pass instance.
- pRenderingAreaInfo is a pointer to a VkRenderingAreaInfoKHR structure specifying details of the render pass instance to query the render area granularity for.
- pGranularity is a pointer to a VkExtent2D structure in which the granularity is returned.

The conditions leading to an optimal renderArea are:

- the `offset.x` member in `renderArea` is a multiple of the `width` member of the returned VkExtent2D (the horizontal granularity).
- the `offset.y` member in `renderArea` is a multiple of the `height` member of the returned VkExtent2D (the vertical granularity).
- either the `extent.width` member in `renderArea` is a multiple of the horizontal granularity or `offset.x+extent.width` is equal to the `width` of each attachment used in the render pass instance.
- either the `extent.height` member in `renderArea` is a multiple of the vertical granularity or `offset.y+extent.height` is equal to the `height` of each attachment used in the render pass instance.

### Valid Usage (Implicit)

- VUID-vkGetRenderingAreaGranularityKHR-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkGetRenderingAreaGranularityKHR-pRenderingAreaInfo-parameter
  `pRenderingAreaInfo` **must** be a valid pointer to a valid VkRenderingAreaInfoKHR structure
- VUID-vkGetRenderingAreaGranularityKHR-pGranularity-parameter
  `pGranularity` **must** be a valid pointer to a VkExtent2D structure

The `VkRenderingAreaInfoKHR` structure is defined as:

```
// Provided by VK_KHR_maintenance5
typedef struct VkRenderingAreaInfoKHR {
    VkStructureType    sType;
    const void*        pNext;
    uint32_t           viewMask;
    uint32_t           colorAttachmentCount;
    const VkFormat*    pColorAttachmentFormats;
    VkFormat           depthAttachmentFormat;
    VkFormat           stencilAttachmentFormat;
} VkRenderingAreaInfoKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `viewMask` is the viewMask used for rendering.
- `colorAttachmentCount` is the number of entries in `pColorAttachmentFormats`
- `pColorAttachmentFormats` is a pointer to an array of VkFormat values defining the format of color attachments used in the render pass instance.
- `depthAttachmentFormat` is a VkFormat value defining the format of the depth attachment used in the render pass instance.
- `stencilAttachmentFormat` is a VkFormat value defining the format of the stencil attachment used in the render pass instance.

### Valid Usage (Implicit)

- VUID-VkRenderingAreaInfoKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_RENDERING_AREA_INFO_KHR`
- VUID-VkRenderingAreaInfoKHR-pNext-pNext
  `pNext` **must** be `NULL`

The `VkRenderPassStripeBeginInfoARM` structure is defined as:

```
// Provided by VK_ARM_render_pass_striped
typedef struct VkRenderPassStripeBeginInfoARM {
    VkStructureType                     sType;
    const void*                         pNext;
    uint32_t                            stripeInfoCount;
    const VkRenderPassStripeInfoARM*    pStripeInfos;
} VkRenderPassStripeBeginInfoARM;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `stripeInfoCount` is the number of stripes in this render pass instance
- `pStripeInfos` is a pointer to an array of `stripeInfoCount` VkRenderPassStripeInfoARM structures describing the stripes used by the render pass instance.

This structure can be included in the `pNext` chain of VkRenderPassBeginInfo or VkRenderingInfo to define how the render pass instance is split into stripes.

### Valid Usage

- VUID-VkRenderPassStripeBeginInfoARM-stripeInfoCount-09450
  `stripeInfoCount` **must** be less than or equal to `VkPhysicalDeviceRenderPassStripedPropertiesARM::maxRenderPassStripes`
- VUID-VkRenderPassStripeBeginInfoARM-stripeArea-09451
  The `stripeArea` defined by each element of `pStripeInfos` **must** not overlap the `stripeArea` of any other element

### Valid Usage (Implicit)

- VUID-VkRenderPassStripeBeginInfoARM-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_RENDER_PASS_STRIPE_BEGIN_INFO_ARM`
- VUID-VkRenderPassStripeBeginInfoARM-pStripeInfos-parameter
  `pStripeInfos` **must** be a valid pointer to an array of `stripeInfoCount` valid VkRenderPassStripeInfoARM structures

- VUID-VkRenderPassStripeBeginInfoARM-stripeInfoCount-arraylength
  `stripeInfoCount` **must** be greater than `0`

The `VkRenderPassStripeInfoARM` structure is defined as:

```
// Provided by VK_ARM_render_pass_striped
typedef struct VkRenderPassStripeInfoARM {
    VkStructureType    sType;
    const void*        pNext;
    VkRect2D           stripeArea;
} VkRenderPassStripeInfoARM;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `stripeArea` is the stripe area, and is described in more detail below.

`stripeArea` is the render area that is affected by this stripe of the render pass instance. It **must** be a subregion of the `renderArea` of the render pass instance.

### Valid Usage

- VUID-VkRenderPassStripeInfoARM-stripeArea-09452
  `stripeArea.offset.x` **must** be a multiple of VkPhysicalDeviceRenderPassStripedPropertiesARM::`renderPassStripeGranularity.width`
- VUID-VkRenderPassStripeInfoARM-stripeArea-09453
  `stripeArea.extent.width` **must** be a multiple of VkPhysicalDeviceRenderPassStripedPropertiesARM::`renderPassStripeGranularity.width`, or the sum of `stripeArea.offset.x` and `stripeArea.extent.width` **must** be equal to the `renderArea.extent.width` of the render pass instance
- VUID-VkRenderPassStripeInfoARM-stripeArea-09454
  `stripeArea.offset.y` **must** be a multiple of VkPhysicalDeviceRenderPassStripedPropertiesARM::`renderPassStripeGranularity.height`
- VUID-VkRenderPassStripeInfoARM-stripeArea-09455
  `stripeArea.extent.height` **must** be a multiple of VkPhysicalDeviceRenderPassStripedPropertiesARM::`renderPassStripeGranularity.height`, or the sum of `stripeArea.offset.y` and `stripeArea.extent.height` **must** be equal to the `renderArea.extent.height` of the render pass instance

### Valid Usage (Implicit)

- VUID-VkRenderPassStripeInfoARM-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_RENDER_PASS_STRIPE_INFO_ARM`
- VUID-VkRenderPassStripeInfoARM-pNext-pNext

```
    pNext must be NULL
```

To end a render pass instance, call:

```
// Provided by VK_KHR_dynamic_rendering
void vkCmdEndRenderingKHR(
    VkCommandBuffer                             commandBuffer);
```

- `commandBuffer` is the command buffer in which to record the command.

If the value of `pRenderingInfo->flags` used to begin this render pass instance included `VK_RENDERING_SUSPENDING_BIT`, then this render pass is suspended and will be resumed later in submission order.

### Valid Usage

- VUID-vkCmdEndRendering-None-06161
  The current render pass instance **must** have been begun with vkCmdBeginRenderingKHR
- VUID-vkCmdEndRendering-commandBuffer-06162
  The current render pass instance **must** have been begun in `commandBuffer`
- VUID-vkCmdEndRendering-None-06781
  This command **must** not be recorded when transform feedback is active
- VUID-vkCmdEndRendering-None-06999
  If `vkCmdBeginQuery*` was called within the render pass, the corresponding `vkCmdEndQuery*` **must** have been called subsequently within the same subpass

### Valid Usage (Implicit)

- VUID-vkCmdEndRendering-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdEndRendering-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdEndRendering-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- VUID-vkCmdEndRendering-renderpass
  This command **must** only be called inside of a render pass instance
- VUID-vkCmdEndRendering-videocoding
  This command **must** only be called outside of a video coding scope

### Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

### Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Inside | Outside | Graphics | Action<br>State |

**NOTE** For more complex rendering graphs, it is possible to pre-define a static *render pass* object, which as well as allowing draw commands, allows the definition of framebuffer-local dependencies between multiple subpasses. These objects have a lot of setup cost compared to vkCmdBeginRenderingKHR, but use of subpass dependencies can confer important performance benefits on some devices.

The `VkTilePropertiesQCOM` structure is defined as:

```
// Provided by VK_QCOM_tile_properties
typedef struct VkTilePropertiesQCOM {
    VkStructureType    sType;
    void*              pNext;
    VkExtent3D         tileSize;
    VkExtent2D         apronSize;
    VkOffset2D         origin;
} VkTilePropertiesQCOM;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `tileSize` is the dimensions of a tile, with width and height describing the width and height of a tile in pixels, and depth corresponding to the number of slices the tile spans.
- `apronSize` is the dimension of the apron.
- `origin` is the top-left corner of the first tile in attachment space.

All tiles will be tightly packed around the first tile, with edges being multiples of tile width and/or height from the origin.

**NOTE** Reported value for `apronSize` will be zero and its functionality will be described in a future extension.

### Valid Usage (Implicit)

- VUID-VkTilePropertiesQCOM-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_TILE_PROPERTIES_QCOM`
- VUID-VkTilePropertiesQCOM-pNext-pNext
  `pNext` **must** be `NULL`

To query the tile properties when using dynamic rendering, call:

```
// Provided by VK_QCOM_tile_properties
VkResult vkGetDynamicRenderingTilePropertiesQCOM(
    VkDevice                                    device,
    const VkRenderingInfo*                      pRenderingInfo,
    VkTilePropertiesQCOM*                       pProperties);
```

- `device` is a logical device associated with the render pass.
- `pRenderingInfo` is a pointer to the VkRenderingInfo structure specifying details of the render pass instance in dynamic rendering.
- `pProperties` is a pointer to a VkTilePropertiesQCOM structure in which the properties are returned.

### Valid Usage (Implicit)

- VUID-vkGetDynamicRenderingTilePropertiesQCOM-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkGetDynamicRenderingTilePropertiesQCOM-pRenderingInfo-parameter
  `pRenderingInfo` **must** be a valid pointer to a valid VkRenderingInfo structure
- VUID-vkGetDynamicRenderingTilePropertiesQCOM-pProperties-parameter
  `pProperties` **must** be a valid pointer to a VkTilePropertiesQCOM structure

### Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

None

## 8.1. Render Pass Objects

A render pass object represents a collection of attachments, subpasses, and dependencies between the subpasses, and describes how the attachments are used over the course of the subpasses.

Render passes are represented by VkRenderPass handles:

```
// Provided by VK_VERSION_1_0
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkRenderPass)
```

An *attachment description* describes the properties of an attachment including its format, sample count, and how its contents are treated at the beginning and end of each render pass instance.

A *subpass* represents a phase of rendering that reads and writes a subset of the attachments in a render pass. Rendering commands are recorded into a particular subpass of a render pass instance.

A *subpass description* describes the subset of attachments that is involved in the execution of a subpass. Each subpass **can** read from some attachments as *input attachments*, write to some as *color attachments* or *depth/stencil attachments*, perform *shader resolve operations* to *color_attachments* or *depth/stencil_attachments*, and perform *multisample resolve operations* to *resolve attachments*. A subpass description **can** also include a set of *preserve attachments*, which are attachments that are not read or written by the subpass but whose contents **must** be preserved throughout the subpass.

A subpass *uses an attachment* if the attachment is a color, depth/stencil, resolve, depth/stencil resolve, fragment shading rate, or input attachment for that subpass (as determined by the pColorAttachments, pDepthStencilAttachment, pResolveAttachments, VkSubpassDescriptionDepthStencilResolve::pDepthStencilResolveAttachment, VkFragmentShadingRateAttachmentInfoKHR::pFragmentShadingRateAttachment->attachment, and pInputAttachments members of VkSubpassDescription, respectively). A subpass does not use an attachment if that attachment is preserved by the subpass. The *first use of an attachment* is in the lowest numbered subpass that uses that attachment. Similarly, the *last use of an attachment* is in the highest numbered subpass that uses that attachment.

The subpasses in a render pass all render to the same dimensions, and fragments for pixel (x,y,layer) in one subpass **can** only read attachment contents written by previous subpasses at that same (x,y,layer) location. For multi-pixel fragments, the pixel read from an input attachment is selected from the pixels covered by that fragment in an implementation-dependent manner. However, this selection **must** be made consistently for any fragment with the same shading rate for the lifetime of the VkDevice.

**NOTE**

By describing a complete set of subpasses in advance, render passes provide the implementation an opportunity to optimize the storage and transfer of attachment data between subpasses.

In practice, this means that subpasses with a simple framebuffer-space dependency **may** be merged into a single tiled rendering pass, keeping the attachment data on-chip for the duration of a render pass instance. However, it is also quite common for a render pass to only contain a single subpass.

*Subpass dependencies* describe execution and memory dependencies between subpasses.

A *subpass dependency chain* is a sequence of subpass dependencies in a render pass, where the

source subpass of each subpass dependency (after the first) equals the destination subpass of the previous dependency.

Execution of subpasses **may** overlap or execute out of order with regards to other subpasses, unless otherwise enforced by an execution dependency. Each subpass only respects submission order for commands recorded in the same subpass, and the vkCmdBeginRenderPass and vkCmdEndRenderPass commands that delimit the render pass - commands within other subpasses are not included. This affects most other implicit ordering guarantees.

A render pass describes the structure of subpasses and attachments independent of any specific image views for the attachments. The specific image views that will be used for the attachments, and their dimensions, are specified in `VkFramebuffer` objects. Framebuffers are created with respect to a specific render pass that the framebuffer is compatible with (see Render Pass Compatibility). Collectively, a render pass and a framebuffer define the complete render target state for one or more subpasses as well as the algorithmic dependencies between the subpasses.

The various pipeline stages of the drawing commands for a given subpass **may** execute concurrently and/or out of order, both within and across drawing commands, whilst still respecting pipeline order. However for a given (x,y,layer,sample) sample location, certain per-sample operations are performed in rasterization order.

`VK_ATTACHMENT_UNUSED` is a constant indicating that a render pass attachment is not used.

```
#define VK_ATTACHMENT_UNUSED              (~0U)
```

## 8.2. Render Pass Creation

To create a render pass, call:

```
// Provided by VK_VERSION_1_0
VkResult vkCreateRenderPass(
    VkDevice                                    device,
    const VkRenderPassCreateInfo*               pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkRenderPass*                               pRenderPass);
```

- `device` is the logical device that creates the render pass.
- `pCreateInfo` is a pointer to a VkRenderPassCreateInfo structure describing the parameters of the render pass.
- `pAllocator` controls host memory allocation as described in the Memory Allocation chapter.
- `pRenderPass` is a pointer to a VkRenderPass handle in which the resulting render pass object is returned.

## Valid Usage

- VUID-vkCreateRenderPass-device-10000
  `device` **must** support at least one queue family with the `VK_QUEUE_GRAPHICS_BIT` capability

## Valid Usage (Implicit)

- VUID-vkCreateRenderPass-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkCreateRenderPass-pCreateInfo-parameter
  `pCreateInfo` **must** be a valid pointer to a valid VkRenderPassCreateInfo structure
- VUID-vkCreateRenderPass-pAllocator-parameter
  If `pAllocator` is not `NULL`, `pAllocator` **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkCreateRenderPass-pRenderPass-parameter
  `pRenderPass` **must** be a valid pointer to a VkRenderPass handle

## Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`

The `VkRenderPassCreateInfo` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkRenderPassCreateInfo {
    VkStructureType                    sType;
    const void*                        pNext;
    VkRenderPassCreateFlags            flags;
    uint32_t                           attachmentCount;
    const VkAttachmentDescription*     pAttachments;
    uint32_t                           subpassCount;
    const VkSubpassDescription*        pSubpasses;
    uint32_t                           dependencyCount;
    const VkSubpassDependency*         pDependencies;
} VkRenderPassCreateInfo;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.

- flags is a bitmask of VkRenderPassCreateFlagBits
- attachmentCount is the number of attachments used by this render pass.
- pAttachments is a pointer to an array of attachmentCount VkAttachmentDescription structures describing the attachments used by the render pass.
- subpassCount is the number of subpasses to create.
- pSubpasses is a pointer to an array of subpassCount VkSubpassDescription structures describing each subpass.
- dependencyCount is the number of memory dependencies between pairs of subpasses.
- pDependencies is a pointer to an array of dependencyCount VkSubpassDependency structures describing dependencies between pairs of subpasses.

**NOTE** Care should be taken to avoid a data race here; if any subpasses access attachments with overlapping memory locations, and one of those accesses is a write, a subpass dependency needs to be included between them.

## Valid Usage

- VUID-VkRenderPassCreateInfo-attachment-00834
  If the attachment member of any element of pInputAttachments, pColorAttachments, pResolveAttachments or pDepthStencilAttachment, or any element of pPreserveAttachments in any element of pSubpasses is not VK_ATTACHMENT_UNUSED, then it **must** be less than attachmentCount
- VUID-VkRenderPassCreateInfo-fragmentDensityMapAttachment-06471
  If the pNext chain includes a VkRenderPassFragmentDensityMapCreateInfoEXT structure and the fragmentDensityMapAttachment member is not VK_ATTACHMENT_UNUSED, then attachment **must** be less than attachmentCount
- VUID-VkRenderPassCreateInfo-pAttachments-00836
  For any member of pAttachments with a loadOp equal to VK_ATTACHMENT_LOAD_OP_CLEAR, the first use of that attachment **must** not specify a layout equal to VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL or VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL
- VUID-VkRenderPassCreateInfo-pAttachments-02511
  For any member of pAttachments with a stencilLoadOp equal to VK_ATTACHMENT_LOAD_OP_CLEAR, the first use of that attachment **must** not specify a layout equal to VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL or VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL
- VUID-VkRenderPassCreateInfo-pAttachments-01566
  For any member of pAttachments with a loadOp equal to VK_ATTACHMENT_LOAD_OP_CLEAR, the first use of that attachment **must** not specify a layout equal to VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_STENCIL_ATTACHMENT_OPTIMAL
- VUID-VkRenderPassCreateInfo-pAttachments-01567
  For any member of pAttachments with a stencilLoadOp equal to VK_ATTACHMENT_LOAD_OP_CLEAR, the first use of that attachment **must** not specify a layout

equal to VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_STENCIL_READ_ONLY_OPTIMAL

- VUID-VkRenderPassCreateInfo-pNext-01926
  If the pNext chain includes a VkRenderPassInputAttachmentAspectCreateInfo structure, the subpass member of each element of its pAspectReferences member **must** be less than subpassCount
- VUID-VkRenderPassCreateInfo-pNext-01927
  If the pNext chain includes a VkRenderPassInputAttachmentAspectCreateInfo structure, the inputAttachmentIndex member of each element of its pAspectReferences member **must** be less than the value of inputAttachmentCount in the element of pSubpasses identified by its subpass member
- VUID-VkRenderPassCreateInfo-pNext-01963
  If the pNext chain includes a VkRenderPassInputAttachmentAspectCreateInfo structure, for any element of the pInputAttachments member of any element of pSubpasses where the attachment member is not VK_ATTACHMENT_UNUSED, the aspectMask member of the corresponding element of VkRenderPassInputAttachmentAspectCreateInfo::pAspectReferences **must** only include aspects that are present in images of the format specified by the element of pAttachments at attachment
- VUID-VkRenderPassCreateInfo-pNext-01928
  If the pNext chain includes a VkRenderPassMultiviewCreateInfo structure, and its subpassCount member is not zero, that member **must** be equal to the value of subpassCount
- VUID-VkRenderPassCreateInfo-pNext-01929
  If the pNext chain includes a VkRenderPassMultiviewCreateInfo structure, if its dependencyCount member is not zero, it **must** be equal to dependencyCount
- VUID-VkRenderPassCreateInfo-pNext-01930
  If the pNext chain includes a VkRenderPassMultiviewCreateInfo structure, for each non-zero element of pViewOffsets, the srcSubpass and dstSubpass members of pDependencies at the same index **must** not be equal
- VUID-VkRenderPassCreateInfo-pNext-02512
  If the pNext chain includes a VkRenderPassMultiviewCreateInfo structure, for any element of pDependencies with a dependencyFlags member that does not include VK_DEPENDENCY_VIEW_LOCAL_BIT, the corresponding element of the pViewOffsets member of that VkRenderPassMultiviewCreateInfo instance **must** be 0
- VUID-VkRenderPassCreateInfo-pNext-02513
  If the pNext chain includes a VkRenderPassMultiviewCreateInfo structure, elements of its pViewMasks member **must** either all be 0, or all not be 0
- VUID-VkRenderPassCreateInfo-pNext-02514
  If the pNext chain includes a VkRenderPassMultiviewCreateInfo structure, and each element of its pViewMasks member is 0, the dependencyFlags member of each element of pDependencies **must** not include VK_DEPENDENCY_VIEW_LOCAL_BIT
- VUID-VkRenderPassCreateInfo-pNext-02515
  If the pNext chain includes a VkRenderPassMultiviewCreateInfo structure, and each element of its pViewMasks member is 0, its correlationMaskCount member **must** be 0
- VUID-VkRenderPassCreateInfo-pDependencies-00837

For any element of pDependencies, if the srcSubpass is not VK_SUBPASS_EXTERNAL, all stage flags included in the srcStageMask member of that dependency **must** be a pipeline stage supported by the pipeline identified by the pipelineBindPoint member of the source subpass

- VUID-VkRenderPassCreateInfo-pDependencies-00838
  For any element of pDependencies, if the dstSubpass is not VK_SUBPASS_EXTERNAL, all stage flags included in the dstStageMask member of that dependency **must** be a pipeline stage supported by the pipeline identified by the pipelineBindPoint member of the destination subpass
- VUID-VkRenderPassCreateInfo-pDependencies-06866
  For any element of pDependencies, if its srcSubpass is not VK_SUBPASS_EXTERNAL, it **must** be less than subpassCount
- VUID-VkRenderPassCreateInfo-pDependencies-06867
  For any element of pDependencies, if its dstSubpass is not VK_SUBPASS_EXTERNAL, it **must** be less than subpassCount

## Valid Usage (Implicit)

- VUID-VkRenderPassCreateInfo-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_RENDER_PASS_CREATE_INFO
- VUID-VkRenderPassCreateInfo-pNext-pNext
  Each pNext member of any structure (including this one) in the pNext chain **must** be either NULL or a pointer to a valid instance of VkRenderPassFragmentDensityMapCreateInfoEXT, VkRenderPassInputAttachmentAspectCreateInfo, or VkRenderPassMultiviewCreateInfo
- VUID-VkRenderPassCreateInfo-sType-unique
  The sType value of each struct in the pNext chain **must** be unique
- VUID-VkRenderPassCreateInfo-flags-parameter
  flags **must** be a valid combination of VkRenderPassCreateFlagBits values
- VUID-VkRenderPassCreateInfo-pAttachments-parameter
  If attachmentCount is not 0, pAttachments **must** be a valid pointer to an array of attachmentCount valid VkAttachmentDescription structures
- VUID-VkRenderPassCreateInfo-pSubpasses-parameter
  pSubpasses **must** be a valid pointer to an array of subpassCount valid VkSubpassDescription structures
- VUID-VkRenderPassCreateInfo-pDependencies-parameter
  If dependencyCount is not 0, pDependencies **must** be a valid pointer to an array of dependencyCount valid VkSubpassDependency structures
- VUID-VkRenderPassCreateInfo-subpassCount-arraylength
  subpassCount **must** be greater than 0

Bits which **can** be set in VkRenderPassCreateInfo::flags, describing additional properties of the render pass, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkRenderPassCreateFlagBits {
  // Provided by VK_QCOM_render_pass_transform
    VK_RENDER_PASS_CREATE_TRANSFORM_BIT_QCOM = 0x00000002,
} VkRenderPassCreateFlagBits;
```

- VK_RENDER_PASS_CREATE_TRANSFORM_BIT_QCOM specifies that the created render pass is compatible with render pass transform.

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkRenderPassCreateFlags;
```

VkRenderPassCreateFlags is a bitmask type for setting a mask of zero or more VkRenderPassCreateFlagBits.

If the VkRenderPassCreateInfo::pNext chain includes a VkRenderPassMultiviewCreateInfo structure, then that structure includes an array of view masks, view offsets, and correlation masks for the render pass.

The VkRenderPassMultiviewCreateInfo structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkRenderPassMultiviewCreateInfo {
    VkStructureType    sType;
    const void*        pNext;
    uint32_t           subpassCount;
    const uint32_t*    pViewMasks;
    uint32_t           dependencyCount;
    const int32_t*     pViewOffsets;
    uint32_t           correlationMaskCount;
    const uint32_t*    pCorrelationMasks;
} VkRenderPassMultiviewCreateInfo;
```

or the equivalent

```
// Provided by VK_KHR_multiview
typedef VkRenderPassMultiviewCreateInfo VkRenderPassMultiviewCreateInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- subpassCount is zero or the number of subpasses in the render pass.
- pViewMasks is a pointer to an array of subpassCount view masks, where each mask is a bitfield of view indices describing which views rendering is broadcast to in each subpass, when multiview is enabled. If subpassCount is zero, each view mask is treated as zero.

- dependencyCount is zero or the number of dependencies in the render pass.
- pViewOffsets is a pointer to an array of dependencyCount view offsets, one for each dependency. If dependencyCount is zero, each dependency's view offset is treated as zero. Each view offset controls which views in the source subpass the views in the destination subpass depend on.
- correlationMaskCount is zero or the number of correlation masks.
- pCorrelationMasks is a pointer to an array of correlationMaskCount view masks indicating sets of views that **may** be more efficient to render concurrently.

When a subpass uses a non-zero view mask, *multiview* functionality is considered to be enabled. Multiview is all-or-nothing for a render pass - that is, either all subpasses **must** have a non-zero view mask (though some subpasses **may** have only one view) or all **must** be zero. Multiview causes all drawing and clear commands in the subpass to behave as if they were broadcast to each view, where a view is represented by one layer of the framebuffer attachments. All draws and clears are broadcast to each *view index* whose bit is set in the view mask. The view index is provided in the ViewIndex shader input variable, and color, depth/stencil, and input attachments all read/write the layer of the framebuffer corresponding to the view index.

If the view mask is zero for all subpasses, multiview is considered to be disabled and all drawing commands execute normally, without this additional broadcasting.

Some implementations **may** not support multiview in conjunction with mesh shaders, geometry shaders or tessellation shaders.

When multiview is enabled, the VK_DEPENDENCY_VIEW_LOCAL_BIT bit in a dependency **can** be used to express a view-local dependency, meaning that each view in the destination subpass depends on a single view in the source subpass. Unlike pipeline barriers, a subpass dependency **can** potentially have a different view mask in the source subpass and the destination subpass. If the dependency is view-local, then each view (dstView) in the destination subpass depends on the view dstView + pViewOffsets[dependency] in the source subpass. If there is not such a view in the source subpass, then this dependency does not affect that view in the destination subpass. If the dependency is not view-local, then all views in the destination subpass depend on all views in the source subpass, and the view offset is ignored. A non-zero view offset is not allowed in a self-dependency.

The elements of pCorrelationMasks are a set of masks of views indicating that views in the same mask **may** exhibit spatial coherency between the views, making it more efficient to render them concurrently. Correlation masks **must** not have a functional effect on the results of the multiview rendering.

When multiview is enabled, at the beginning of each subpass all non-render pass state is undefined. In particular, each time vkCmdBeginRenderPass or vkCmdNextSubpass is called the graphics pipeline **must** be bound, any relevant descriptor sets or vertex/index buffers **must** be bound, and any relevant dynamic state or push constants **must** be set before they are used.

A multiview subpass **can** declare that its shaders will write per-view attributes for all views in a single invocation, by setting the VK_SUBPASS_DESCRIPTION_PER_VIEW_ATTRIBUTES_BIT_NVX bit in the subpass description. The only supported per-view attributes are position and viewport mask, and per-view position and viewport masks are written to output array variables decorated with PositionPerViewNV and ViewportMaskPerViewNV, respectively. If VK_NV_viewport_array2 is not supported

and enabled, `ViewportMaskPerViewNV` **must** not be used. Values written to elements of `PositionPerViewNV` and `ViewportMaskPerViewNV` **must** not depend on the `ViewIndex`. The shader **must** also write to an output variable decorated with `Position`, and the value written to `Position` **must** equal the value written to `PositionPerViewNV`[`ViewIndex`]. Similarly, if `ViewportMaskPerViewNV` is written to then the shader **must** also write to an output variable decorated with `ViewportMaskNV`, and the value written to `ViewportMaskNV` **must** equal the value written to `ViewportMaskPerViewNV`[`ViewIndex`]. Implementations will either use values taken from `Position` and `ViewportMaskNV` and invoke the shader once for each view, or will use values taken from `PositionPerViewNV` and `ViewportMaskPerViewNV` and invoke the shader fewer times. The values written to `Position` and `ViewportMaskNV` **must** not depend on the values written to `PositionPerViewNV` and `ViewportMaskPerViewNV`, or vice versa (to allow compilers to eliminate the unused outputs). All attributes that do not have `*PerViewNV` counterparts **must** not depend on `ViewIndex`.

Per-view attributes are all-or-nothing for a subpass. That is, all pipelines compiled against a subpass that includes the `VK_SUBPASS_DESCRIPTION_PER_VIEW_ATTRIBUTES_BIT_NVX` bit **must** write per-view attributes to the `*PerViewNV[]` shader outputs, in addition to the non-per-view (e.g. `Position`) outputs. Pipelines compiled against a subpass that does not include this bit **must** not include the `*PerViewNV[]` outputs in their interfaces.

### Valid Usage

- VUID-VkRenderPassMultiviewCreateInfo-pCorrelationMasks-00841
  Each view index **must** not be set in more than one element of `pCorrelationMasks`
- VUID-VkRenderPassMultiviewCreateInfo-multiview-06555
  If the `multiview` feature is not enabled, each element of `pViewMasks` **must** be `0`
- VUID-VkRenderPassMultiviewCreateInfo-pViewMasks-06697
  The index of the most significant bit in each element of `pViewMasks` **must** be less than `maxMultiviewViewCount`

### Valid Usage (Implicit)

- VUID-VkRenderPassMultiviewCreateInfo-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_RENDER_PASS_MULTIVIEW_CREATE_INFO`
- VUID-VkRenderPassMultiviewCreateInfo-pViewMasks-parameter
  If `subpassCount` is not `0`, `pViewMasks` **must** be a valid pointer to an array of `subpassCount` `uint32_t` values
- VUID-VkRenderPassMultiviewCreateInfo-pViewOffsets-parameter
  If `dependencyCount` is not `0`, `pViewOffsets` **must** be a valid pointer to an array of `dependencyCount` `int32_t` values
- VUID-VkRenderPassMultiviewCreateInfo-pCorrelationMasks-parameter
  If `correlationMaskCount` is not `0`, `pCorrelationMasks` **must** be a valid pointer to an array of `correlationMaskCount` `uint32_t` values

The `VkMultiviewPerViewAttributesInfoNVX` structure is defined as:

```
// Provided by VK_KHR_dynamic_rendering with VK_NVX_multiview_per_view_attributes
typedef struct VkMultiviewPerViewAttributesInfoNVX {
    VkStructureType    sType;
    const void*        pNext;
    VkBool32           perViewAttributes;
    VkBool32           perViewAttributesPositionXOnly;
} VkMultiviewPerViewAttributesInfoNVX;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `perViewAttributes` specifies that shaders compiled for this pipeline write the attributes for all views in a single invocation of each vertex processing stage. All pipelines executed within a render pass instance that includes this bit **must** write per-view attributes to the `*PerViewNV[]` shader outputs, in addition to the non-per-view (e.g. `Position`) outputs.
- `perViewAttributesPositionXOnly` specifies that shaders compiled for this pipeline use per-view positions which only differ in value in the x component. Per-view viewport mask **can** also be used.

When dynamic render pass instances are being used, instead of specifying `VK_SUBPASS_DESCRIPTION_PER_VIEW_ATTRIBUTES_BIT_NVX` or `VK_SUBPASS_DESCRIPTION_PER_VIEW_POSITION_X_ONLY_BIT_NVX` in the subpass description flags, the per-attribute properties of the render pass instance **must** be specified by the `VkMultiviewPerViewAttributesInfoNVX` structure Include the `VkMultiviewPerViewAttributesInfoNVX` structure in the `pNext` chain of VkGraphicsPipelineCreateInfo when creating a graphics pipeline for dynamic rendering, VkRenderingInfo when starting a dynamic render pass instance, and VkCommandBufferInheritanceInfo when specifying the dynamic render pass instance parameters for secondary command buffers.

### Valid Usage (Implicit)

- VUID-VkMultiviewPerViewAttributesInfoNVX-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_MULTIVIEW_PER_VIEW_ATTRIBUTES_INFO_NVX`

If the VkRenderPassCreateInfo::`pNext` chain includes a `VkRenderPassFragmentDensityMapCreateInfoEXT` structure, then that structure includes a fragment density map attachment for the render pass.

The `VkRenderPassFragmentDensityMapCreateInfoEXT` structure is defined as:

```
// Provided by VK_EXT_fragment_density_map
typedef struct VkRenderPassFragmentDensityMapCreateInfoEXT {
    VkStructureType          sType;
    const void*              pNext;
    VkAttachmentReference    fragmentDensityMapAttachment;
} VkRenderPassFragmentDensityMapCreateInfoEXT;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- fragmentDensityMapAttachment is the fragment density map to use for the render pass.

The fragment density map is read at an implementation-dependent time with the following constraints determined by the attachment's image view flags:

- VK_IMAGE_VIEW_CREATE_FRAGMENT_DENSITY_MAP_DYNAMIC_BIT_EXT specifies that the fragment density map will be read by the device during VK_PIPELINE_STAGE_FRAGMENT_DENSITY_PROCESS_BIT_EXT
- VK_IMAGE_VIEW_CREATE_FRAGMENT_DENSITY_MAP_DEFERRED_BIT_EXT specifies that the fragment density map will be read by the host during vkEndCommandBuffer of the primary command buffer that the render pass is recorded into
- Otherwise the fragment density map will be read by the host during vkCmdBeginRenderPass

The fragment density map **may** additionally be read by the device during VK_PIPELINE_STAGE_FRAGMENT_DENSITY_PROCESS_BIT_EXT for any mode.

If this structure is not present, it is as if fragmentDensityMapAttachment was given as VK_ATTACHMENT_UNUSED.

### Valid Usage

- VUID-VkRenderPassFragmentDensityMapCreateInfoEXT-fragmentDensityMapAttachment-02548
  If fragmentDensityMapAttachment is not VK_ATTACHMENT_UNUSED, fragmentDensityMapAttachment **must** not be an element of VkSubpassDescription::pInputAttachments, VkSubpassDescription::pColorAttachments, VkSubpassDescription::pResolveAttachments, VkSubpassDescription::pDepthStencilAttachment, or VkSubpassDescription::pPreserveAttachments for any subpass
- VUID-VkRenderPassFragmentDensityMapCreateInfoEXT-fragmentDensityMapAttachment-02549
  If fragmentDensityMapAttachment is not VK_ATTACHMENT_UNUSED, layout **must** be equal to VK_IMAGE_LAYOUT_FRAGMENT_DENSITY_MAP_OPTIMAL_EXT, or VK_IMAGE_LAYOUT_GENERAL
- VUID-VkRenderPassFragmentDensityMapCreateInfoEXT-fragmentDensityMapAttachment-02550
  If fragmentDensityMapAttachment is not VK_ATTACHMENT_UNUSED, fragmentDensityMapAttachment **must** reference an attachment with a loadOp equal to VK_ATTACHMENT_LOAD_OP_LOAD or VK_ATTACHMENT_LOAD_OP_DONT_CARE
- VUID-VkRenderPassFragmentDensityMapCreateInfoEXT-fragmentDensityMapAttachment-02551
  If fragmentDensityMapAttachment is not VK_ATTACHMENT_UNUSED, fragmentDensityMapAttachment **must** reference an attachment with a storeOp equal to VK_ATTACHMENT_STORE_OP_DONT_CARE

### Valid Usage (Implicit)

- VUID-VkRenderPassFragmentDensityMapCreateInfoEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_RENDER_PASS_FRAGMENT_DENSITY_MAP_CREATE_INFO_EXT`
- VUID-VkRenderPassFragmentDensityMapCreateInfoEXT-fragmentDensityMapAttachment-parameter
  `fragmentDensityMapAttachment` **must** be a valid VkAttachmentReference structure

The `VkAttachmentDescription` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkAttachmentDescription {
    VkAttachmentDescriptionFlags    flags;
    VkFormat                        format;
    VkSampleCountFlagBits           samples;
    VkAttachmentLoadOp              loadOp;
    VkAttachmentStoreOp             storeOp;
    VkAttachmentLoadOp              stencilLoadOp;
    VkAttachmentStoreOp             stencilStoreOp;
    VkImageLayout                   initialLayout;
    VkImageLayout                   finalLayout;
} VkAttachmentDescription;
```

- `flags` is a bitmask of VkAttachmentDescriptionFlagBits specifying additional properties of the attachment.
- `format` is a VkFormat value specifying the format of the image view that will be used for the attachment.
- `samples` is a VkSampleCountFlagBits value specifying the number of samples of the image.
- `loadOp` is a VkAttachmentLoadOp value specifying how the contents of color and depth components of the attachment are treated at the beginning of the subpass where it is first used.
- `storeOp` is a VkAttachmentStoreOp value specifying how the contents of color and depth components of the attachment are treated at the end of the subpass where it is last used.
- `stencilLoadOp` is a VkAttachmentLoadOp value specifying how the contents of stencil components of the attachment are treated at the beginning of the subpass where it is first used.
- `stencilStoreOp` is a VkAttachmentStoreOp value specifying how the contents of stencil components of the attachment are treated at the end of the last subpass where it is used.
- `initialLayout` is the layout the attachment image subresource will be in when a render pass instance begins.
- `finalLayout` is the layout the attachment image subresource will be transitioned to when a render pass instance ends.

If the attachment uses a color format, then `loadOp` and `storeOp` are used, and `stencilLoadOp` and `stencilStoreOp` are ignored. If the format has depth and/or stencil components, `loadOp` and `storeOp`

apply only to the depth data, while stencilLoadOp and stencilStoreOp define how the stencil data is handled. loadOp and stencilLoadOp define the load operations for the attachment. storeOp and stencilStoreOp define the store operations for the attachment. If an attachment is not used by any subpass, loadOp, storeOp, stencilStoreOp, and stencilLoadOp will be ignored for that attachment, and no load or store ops will be performed. However, any transition specified by initialLayout and finalLayout will still be executed.

If flags includes VK_ATTACHMENT_DESCRIPTION_MAY_ALIAS_BIT, then the attachment is treated as if it shares physical memory with another attachment in the same render pass. This information limits the ability of the implementation to reorder certain operations (like layout transitions and the loadOp) such that it is not improperly reordered against other uses of the same physical memory via a different attachment. This is described in more detail below.

If a render pass uses multiple attachments that alias the same device memory, those attachments **must** each include the VK_ATTACHMENT_DESCRIPTION_MAY_ALIAS_BIT bit in their attachment description flags. Attachments aliasing the same memory occurs in multiple ways:

- Multiple attachments being assigned the same image view as part of framebuffer creation.
- Attachments using distinct image views that correspond to the same image subresource of an image.
- Attachments using views of distinct image subresources which are bound to overlapping memory ranges.

**NOTE** Render passes **must** include subpass dependencies (either directly or via a subpass dependency chain) between any two subpasses that operate on the same attachment or aliasing attachments and those subpass dependencies **must** include execution and memory dependencies separating uses of the aliases, if at least one of those subpasses writes to one of the aliases. These dependencies **must** not include the VK_DEPENDENCY_BY_REGION_BIT if the aliases are views of distinct image subresources which overlap in memory.

Multiple attachments that alias the same memory **must** not be used in a single subpass. A given attachment index **must** not be used multiple times in a single subpass, with one exception: two subpass attachments **can** use the same attachment index if at least one use is as an input attachment and neither use is as a resolve or preserve attachment. In other words, the same view **can** be used simultaneously as an input and color or depth/stencil attachment, but **must** not be used as multiple color or depth/stencil attachments nor as resolve or preserve attachments.

If a set of attachments alias each other, then all except the first to be used in the render pass **must** use an initialLayout of VK_IMAGE_LAYOUT_UNDEFINED, since the earlier uses of the other aliases make their contents undefined. Once an alias has been used and a different alias has been used after it, the first alias **must** not be used in any later subpasses. However, an application **can** assign the same image view to multiple aliasing attachment indices, which allows that image view to be used multiple times even if other aliases are used in between.

**NOTE** Once an attachment needs the VK_ATTACHMENT_DESCRIPTION_MAY_ALIAS_BIT bit, there **should** be no additional cost of introducing additional aliases, and using these additional aliases **may** allow more efficient clearing of the attachments on multiple

uses via VK_ATTACHMENT_LOAD_OP_CLEAR.

## Valid Usage

- VUID-VkAttachmentDescription-format-06699
  If format includes a color or depth component and loadOp is VK_ATTACHMENT_LOAD_OP_LOAD, then initialLayout **must** not be VK_IMAGE_LAYOUT_UNDEFINED
- VUID-VkAttachmentDescription-finalLayout-00843
  finalLayout **must** not be VK_IMAGE_LAYOUT_UNDEFINED or VK_IMAGE_LAYOUT_PREINITIALIZED
- VUID-VkAttachmentDescription-format-03280
  If format is a color format, initialLayout **must** not be VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL or VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL
- VUID-VkAttachmentDescription-format-03281
  If format is a depth/stencil format, initialLayout **must** not be VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL
- VUID-VkAttachmentDescription-format-03282
  If format is a color format, finalLayout **must** not be VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL or VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL
- VUID-VkAttachmentDescription-format-03283
  If format is a depth/stencil format, finalLayout **must** not be VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL
- VUID-VkAttachmentDescription-format-06487
  If format is a color format, initialLayout **must** not be VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_STENCIL_READ_ONLY_OPTIMAL or VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_STENCIL_ATTACHMENT_OPTIMAL
- VUID-VkAttachmentDescription-format-06488
  If format is a color format, finalLayout **must** not be VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_STENCIL_READ_ONLY_OPTIMAL or VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_STENCIL_ATTACHMENT_OPTIMAL
- VUID-VkAttachmentDescription-separateDepthStencilLayouts-03284
  If the separateDepthStencilLayouts feature is not enabled, initialLayout **must** not be VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL, VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL, VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL, or VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL,
- VUID-VkAttachmentDescription-separateDepthStencilLayouts-03285
  If the separateDepthStencilLayouts feature is not enabled, finalLayout **must** not be VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL, VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL, VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL, or VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL,
- VUID-VkAttachmentDescription-format-03286
  If format is a color format, initialLayout **must** not be

VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL, VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL, VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL, or VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL

- VUID-VkAttachmentDescription-format-03287
  If format is a color format, finalLayout **must** not be VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL, VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL, VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL, or VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL
- VUID-VkAttachmentDescription-format-06906
  If format is a depth/stencil format which includes both depth and stencil components, initialLayout **must** not be VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL or VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL
- VUID-VkAttachmentDescription-format-06907
  If format is a depth/stencil format which includes both depth and stencil components, finalLayout **must** not be VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL or VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL
- VUID-VkAttachmentDescription-format-03290
  If format is a depth/stencil format which includes only the depth component, initialLayout **must** not be VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL or VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL
- VUID-VkAttachmentDescription-format-03291
  If format is a depth/stencil format which includes only the depth component, finalLayout **must** not be VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL or VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL
- VUID-VkAttachmentDescription-synchronization2-06908
  If the synchronization2 feature is not enabled, initialLayout **must** not be VK_IMAGE_LAYOUT_ATTACHMENT_OPTIMAL_KHR or VK_IMAGE_LAYOUT_READ_ONLY_OPTIMAL_KHR
- VUID-VkAttachmentDescription-synchronization2-06909
  If the synchronization2 feature is not enabled, finalLayout **must** not be VK_IMAGE_LAYOUT_ATTACHMENT_OPTIMAL_KHR or VK_IMAGE_LAYOUT_READ_ONLY_OPTIMAL_KHR
- VUID-VkAttachmentDescription-attachmentFeedbackLoopLayout-07309
  If the attachmentFeedbackLoopLayout feature is not enabled, initialLayout **must** not be VK_IMAGE_LAYOUT_ATTACHMENT_FEEDBACK_LOOP_OPTIMAL_EXT
- VUID-VkAttachmentDescription-attachmentFeedbackLoopLayout-07310
  If the attachmentFeedbackLoopLayout feature is not enabled, finalLayout **must** not be VK_IMAGE_LAYOUT_ATTACHMENT_FEEDBACK_LOOP_OPTIMAL_EXT
- VUID-VkAttachmentDescription-samples-08745
  samples **must** be a valid VkSampleCountFlagBits value that is set in imageCreateSampleCounts (as defined in Image Creation Limits) for the given format
- VUID-VkAttachmentDescription-dynamicRenderingLocalRead-09544
  If the dynamicRenderingLocalRead feature is not enabled, initialLayout **must** not be VK_IMAGE_LAYOUT_RENDERING_LOCAL_READ_KHR
- VUID-VkAttachmentDescription-dynamicRenderingLocalRead-09545

If the dynamicRenderingLocalRead feature is not enabled, finalLayout **must** not be VK_IMAGE_LAYOUT_RENDERING_LOCAL_READ_KHR

- VUID-VkAttachmentDescription-format-06698
  format **must** not be VK_FORMAT_UNDEFINED
- VUID-VkAttachmentDescription-format-06700
  If format includes a stencil component and stencilLoadOp is VK_ATTACHMENT_LOAD_OP_LOAD, then initialLayout **must** not be VK_IMAGE_LAYOUT_UNDEFINED
- VUID-VkAttachmentDescription-format-03292
  If format is a depth/stencil format which includes only the stencil component, initialLayout **must** not be VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL or VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL
- VUID-VkAttachmentDescription-format-03293
  If format is a depth/stencil format which includes only the stencil component, finalLayout **must** not be VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL or VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL
- VUID-VkAttachmentDescription-format-06242
  If format is a depth/stencil format which includes both depth and stencil components, initialLayout **must** not be VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL or VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL
- VUID-VkAttachmentDescription-format-06243
  If format is a depth/stencil format which includes both depth and stencil components, finalLayout **must** not be VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL or VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL

## Valid Usage (Implicit)

- VUID-VkAttachmentDescription-flags-parameter
  flags **must** be a valid combination of VkAttachmentDescriptionFlagBits values
- VUID-VkAttachmentDescription-format-parameter
  format **must** be a valid VkFormat value
- VUID-VkAttachmentDescription-samples-parameter
  samples **must** be a valid VkSampleCountFlagBits value
- VUID-VkAttachmentDescription-loadOp-parameter
  loadOp **must** be a valid VkAttachmentLoadOp value
- VUID-VkAttachmentDescription-storeOp-parameter
  storeOp **must** be a valid VkAttachmentStoreOp value
- VUID-VkAttachmentDescription-stencilLoadOp-parameter
  stencilLoadOp **must** be a valid VkAttachmentLoadOp value
- VUID-VkAttachmentDescription-stencilStoreOp-parameter
  stencilStoreOp **must** be a valid VkAttachmentStoreOp value
- VUID-VkAttachmentDescription-initialLayout-parameter
  initialLayout **must** be a valid VkImageLayout value

- VUID-VkAttachmentDescription-finalLayout-parameter
  `finalLayout` **must** be a valid VkImageLayout value

Bits which **can** be set in VkAttachmentDescription::`flags`, describing additional properties of the attachment, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkAttachmentDescriptionFlagBits {
    VK_ATTACHMENT_DESCRIPTION_MAY_ALIAS_BIT = 0x00000001,
} VkAttachmentDescriptionFlagBits;
```

- `VK_ATTACHMENT_DESCRIPTION_MAY_ALIAS_BIT` specifies that the attachment aliases the same device memory as other attachments.

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkAttachmentDescriptionFlags;
```

`VkAttachmentDescriptionFlags` is a bitmask type for setting a mask of zero or more VkAttachmentDescriptionFlagBits.

The `VkRenderPassInputAttachmentAspectCreateInfo` structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkRenderPassInputAttachmentAspectCreateInfo {
    VkStructureType                             sType;
    const void*                                 pNext;
    uint32_t                                    aspectReferenceCount;
    const VkInputAttachmentAspectReference*     pAspectReferences;
} VkRenderPassInputAttachmentAspectCreateInfo;
```

or the equivalent

```
// Provided by VK_KHR_maintenance2
typedef VkRenderPassInputAttachmentAspectCreateInfo
VkRenderPassInputAttachmentAspectCreateInfoKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `aspectReferenceCount` is the number of elements in the `pAspectReferences` array.
- `pAspectReferences` is a pointer to an array of `aspectReferenceCount` VkInputAttachmentAspectReference structures containing a mask describing which aspect(s) **can** be accessed for a given input attachment within a given subpass.

To specify which aspects of an input attachment **can** be read, add a

VkRenderPassInputAttachmentAspectCreateInfo structure to the pNext chain of the VkRenderPassCreateInfo structure:

An application **can** access any aspect of an input attachment that does not have a specified aspect mask in the pAspectReferences array. Otherwise, an application **must** not access aspect(s) of an input attachment other than those in its specified aspect mask.

### Valid Usage (Implicit)

- VUID-VkRenderPassInputAttachmentAspectCreateInfo-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_RENDER_PASS_INPUT_ATTACHMENT_ASPECT_CREATE_INFO
- VUID-VkRenderPassInputAttachmentAspectCreateInfo-pAspectReferences-parameter
  pAspectReferences **must** be a valid pointer to an array of aspectReferenceCount valid VkInputAttachmentAspectReference structures
- VUID-VkRenderPassInputAttachmentAspectCreateInfo-aspectReferenceCount-arraylength
  aspectReferenceCount **must** be greater than 0

The VkInputAttachmentAspectReference structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkInputAttachmentAspectReference {
    uint32_t              subpass;
    uint32_t              inputAttachmentIndex;
    VkImageAspectFlags    aspectMask;
} VkInputAttachmentAspectReference;
```

or the equivalent

```
// Provided by VK_KHR_maintenance2
typedef VkInputAttachmentAspectReference VkInputAttachmentAspectReferenceKHR;
```

- subpass is an index into the pSubpasses array of the parent VkRenderPassCreateInfo structure.
- inputAttachmentIndex is an index into the pInputAttachments of the specified subpass.
- aspectMask is a mask of which aspect(s) **can** be accessed within the specified subpass.

This structure specifies an aspect mask for a specific input attachment of a specific subpass in the render pass.

subpass and inputAttachmentIndex index into the render pass as:

```
pCreateInfo->pSubpasses[subpass].pInputAttachments[inputAttachmentIndex]
```

### Valid Usage

- VUID-VkInputAttachmentAspectReference-aspectMask-01964
  `aspectMask` **must** not include `VK_IMAGE_ASPECT_METADATA_BIT`
- VUID-VkInputAttachmentAspectReference-aspectMask-02250
  `aspectMask` **must** not include `VK_IMAGE_ASPECT_MEMORY_PLANE_i_BIT_EXT` for any index *i*

### Valid Usage (Implicit)

- VUID-VkInputAttachmentAspectReference-aspectMask-parameter
  `aspectMask` **must** be a valid combination of VkImageAspectFlagBits values
- VUID-VkInputAttachmentAspectReference-aspectMask-requiredbitmask
  `aspectMask` **must** not be `0`

The `VkSubpassDescription` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkSubpassDescription {
    VkSubpassDescriptionFlags       flags;
    VkPipelineBindPoint             pipelineBindPoint;
    uint32_t                        inputAttachmentCount;
    const VkAttachmentReference*    pInputAttachments;
    uint32_t                        colorAttachmentCount;
    const VkAttachmentReference*    pColorAttachments;
    const VkAttachmentReference*    pResolveAttachments;
    const VkAttachmentReference*    pDepthStencilAttachment;
    uint32_t                        preserveAttachmentCount;
    const uint32_t*                 pPreserveAttachments;
} VkSubpassDescription;
```

- `flags` is a bitmask of VkSubpassDescriptionFlagBits specifying usage of the subpass.
- `pipelineBindPoint` is a VkPipelineBindPoint value specifying the pipeline type supported for this subpass.
- `inputAttachmentCount` is the number of input attachments.
- `pInputAttachments` is a pointer to an array of VkAttachmentReference structures defining the input attachments for this subpass and their layouts.
- `colorAttachmentCount` is the number of color attachments.
- `pColorAttachments` is a pointer to an array of `colorAttachmentCount` VkAttachmentReference structures defining the color attachments for this subpass and their layouts.
- `pResolveAttachments` is `NULL` or a pointer to an array of `colorAttachmentCount` VkAttachmentReference structures defining the resolve attachments for this subpass and their layouts.

- pDepthStencilAttachment is a pointer to a VkAttachmentReference structure specifying the depth/stencil attachment for this subpass and its layout.
- preserveAttachmentCount is the number of preserved attachments.
- pPreserveAttachments is a pointer to an array of preserveAttachmentCount render pass attachment indices identifying attachments that are not used by this subpass, but whose contents **must** be preserved throughout the subpass.

Each element of the pInputAttachments array corresponds to an input attachment index in a fragment shader, i.e. if a shader declares an image variable decorated with a InputAttachmentIndex value of **X**, then it uses the attachment provided in pInputAttachments[**X**]. Input attachments **must** also be bound to the pipeline in a descriptor set. If the attachment member of any element of pInputAttachments is VK_ATTACHMENT_UNUSED, the application **must** not read from the corresponding input attachment index. Fragment shaders **can** use subpass input variables to access the contents of an input attachment at the fragment's (x, y, layer) framebuffer coordinates. Input attachments **must** not be used by any subpasses within a render pass that enables render pass transform.

Each element of the pColorAttachments array corresponds to an output location in the shader, i.e. if the shader declares an output variable decorated with a Location value of **X**, then it uses the attachment provided in pColorAttachments[**X**]. If the attachment member of any element of pColorAttachments is VK_ATTACHMENT_UNUSED, or if Color Write Enable has been disabled for the corresponding attachment index, then writes to the corresponding location by a fragment shader are discarded.

If flags does not include VK_SUBPASS_DESCRIPTION_SHADER_RESOLVE_BIT_QCOM, and if pResolveAttachments is not NULL, each of its elements corresponds to a color attachment (the element in pColorAttachments at the same index), and a multisample resolve operation is defined for each attachment unless the resolve attachment index is VK_ATTACHMENT_UNUSED.

Similarly, if flags does not include VK_SUBPASS_DESCRIPTION_SHADER_RESOLVE_BIT_QCOM, and VkSubpassDescriptionDepthStencilResolve::pDepthStencilResolveAttachment is not NULL and does not have the value VK_ATTACHMENT_UNUSED, it corresponds to the depth/stencil attachment in pDepthStencilAttachment, and multisample resolve operation for depth and stencil are defined by VkSubpassDescriptionDepthStencilResolve::depthResolveMode and VkSubpassDescriptionDepthStencilResolve::stencilResolveMode, respectively. If VkSubpassDescriptionDepthStencilResolve::depthResolveMode is VK_RESOLVE_MODE_NONE or the pDepthStencilResolveAttachment does not have a depth aspect, no resolve operation is performed for the depth attachment. If VkSubpassDescriptionDepthStencilResolve::stencilResolveMode is VK_RESOLVE_MODE_NONE or the pDepthStencilResolveAttachment does not have a stencil aspect, no resolve operation is performed for the stencil attachment.

If the image subresource range referenced by the depth/stencil attachment is created with VK_IMAGE_CREATE_SAMPLE_LOCATIONS_COMPATIBLE_DEPTH_BIT_EXT, then the multisample resolve operation uses the sample locations state specified in the sampleLocationsInfo member of the element of the VkRenderPassSampleLocationsBeginInfoEXT::pPostSubpassSampleLocations for the subpass.

If pDepthStencilAttachment is NULL, or if its attachment index is VK_ATTACHMENT_UNUSED, it indicates that no depth/stencil attachment will be used in the subpass.

The contents of an attachment within the render area become undefined at the start of a subpass **S** if all of the following conditions are true:

- The attachment is used as a color, depth/stencil, or resolve attachment in any subpass in the render pass.
- There is a subpass $\mathbf{S_1}$ that uses or preserves the attachment, and a subpass dependency from $\mathbf{S_1}$ to **S**.
- The attachment is not used or preserved in subpass **S**.

In addition, the contents of an attachment within the render area become undefined at the start of a subpass **S** if all of the following conditions are true:

- `VK_SUBPASS_DESCRIPTION_SHADER_RESOLVE_BIT_QCOM` is set.
- The attachment is used as a color or depth/stencil in the subpass.

Once the contents of an attachment become undefined in subpass **S**, they remain undefined for subpasses in subpass dependency chains starting with subpass **S** until they are written again. However, they remain valid for subpasses in other subpass dependency chains starting with subpass $\mathbf{S_1}$ if those subpasses use or preserve the attachment.

### Valid Usage

- VUID-VkSubpassDescription-attachment-06912
  If the `attachment` member of an element of `pInputAttachments` is not `VK_ATTACHMENT_UNUSED`, its `layout` member **must** not be `VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL` or `VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL`
- VUID-VkSubpassDescription-attachment-06913
  If the `attachment` member of an element of `pColorAttachments` is not `VK_ATTACHMENT_UNUSED`, its `layout` member **must** not be `VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL` or `VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL`
- VUID-VkSubpassDescription-attachment-06914
  If the `attachment` member of an element of `pResolveAttachments` is not `VK_ATTACHMENT_UNUSED`, its `layout` member **must** not be `VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL` or `VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL`
- VUID-VkSubpassDescription-attachment-06915
  If the `attachment` member of `pDepthStencilAttachment` is not `VK_ATTACHMENT_UNUSED`, ts `layout` member **must** not be `VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL` or `VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL`
- VUID-VkSubpassDescription-attachment-06916
  If the `attachment` member of an element of `pColorAttachments` is not `VK_ATTACHMENT_UNUSED`, its `layout` member **must** not be `VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_STENCIL_READ_ONLY_OPTIMAL` or `VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_STENCIL_ATTACHMENT_OPTIMAL`
- VUID-VkSubpassDescription-attachment-06917

If the attachment member of an element of pResolveAttachments is not VK_ATTACHMENT_UNUSED, its layout member **must** not be VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_STENCIL_READ_ONLY_OPTIMAL or VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_STENCIL_ATTACHMENT_OPTIMAL

- VUID-VkSubpassDescription-attachment-06918
  If the attachment member of an element of pInputAttachments is not VK_ATTACHMENT_UNUSED, its layout member **must** not be VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL or VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL
- VUID-VkSubpassDescription-attachment-06919
  If the attachment member of an element of pColorAttachments is not VK_ATTACHMENT_UNUSED, its layout member **must** not be VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL, VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL, VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL, or VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL
- VUID-VkSubpassDescription-attachment-06920
  If the attachment member of an element of pResolveAttachments is not VK_ATTACHMENT_UNUSED, its layout member **must** not be VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL, VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL, VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL, or VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL
- VUID-VkSubpassDescription-attachment-06921
  If the attachment member of an element of pInputAttachments is not VK_ATTACHMENT_UNUSED, its layout member **must** not be VK_IMAGE_LAYOUT_ATTACHMENT_OPTIMAL_KHR
- VUID-VkSubpassDescription-attachment-06922
  If the attachment member of an element of pColorAttachments is not VK_ATTACHMENT_UNUSED, its layout member **must** not be VK_IMAGE_LAYOUT_READ_ONLY_OPTIMAL_KHR
- VUID-VkSubpassDescription-attachment-06923
  If the attachment member of an element of pResolveAttachments is not VK_ATTACHMENT_UNUSED, its layout member **must** not be VK_IMAGE_LAYOUT_READ_ONLY_OPTIMAL_KHR
- VUID-VkSubpassDescription-pipelineBindPoint-04952
  pipelineBindPoint **must** be VK_PIPELINE_BIND_POINT_GRAPHICS or VK_PIPELINE_BIND_POINT_SUBPASS_SHADING_HUAWEI
- VUID-VkSubpassDescription-colorAttachmentCount-00845
  colorAttachmentCount **must** be less than or equal to VkPhysicalDeviceLimits::maxColorAttachments
- VUID-VkSubpassDescription-loadOp-00846
  If the first use of an attachment in this render pass is as an input attachment, and the attachment is not also used as a color or depth/stencil attachment in the same subpass, then loadOp **must** not be VK_ATTACHMENT_LOAD_OP_CLEAR
- VUID-VkSubpassDescription-pResolveAttachments-00847
  If pResolveAttachments is not NULL, for each resolve attachment that is not VK_ATTACHMENT_UNUSED, the corresponding color attachment **must** not be VK_ATTACHMENT_UNUSED

- VUID-VkSubpassDescription-pResolveAttachments-00848
  If `pResolveAttachments` is not `NULL`, for each resolve attachment that is not `VK_ATTACHMENT_UNUSED`, the corresponding color attachment **must** not have a sample count of `VK_SAMPLE_COUNT_1_BIT`
- VUID-VkSubpassDescription-pResolveAttachments-00849
  If `pResolveAttachments` is not `NULL`, each resolve attachment that is not `VK_ATTACHMENT_UNUSED` **must** have a sample count of `VK_SAMPLE_COUNT_1_BIT`
- VUID-VkSubpassDescription-pResolveAttachments-00850
  If `pResolveAttachments` is not `NULL`, each resolve attachment that is not `VK_ATTACHMENT_UNUSED` **must** have the same VkFormat as its corresponding color attachment
- VUID-VkSubpassDescription-pColorAttachments-09430
  All attachments in `pColorAttachments` that are not `VK_ATTACHMENT_UNUSED` **must** have the same sample count
- VUID-VkSubpassDescription-pInputAttachments-02647
  All attachments in `pInputAttachments` that are not `VK_ATTACHMENT_UNUSED` **must** have image formats whose potential format features contain at least `VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BIT` or `VK_FORMAT_FEATURE_DEPTH_STENCIL_ATTACHMENT_BIT`
- VUID-VkSubpassDescription-pColorAttachments-02648
  All attachments in `pColorAttachments` that are not `VK_ATTACHMENT_UNUSED` **must** have image formats whose potential format features contain `VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BIT`
- VUID-VkSubpassDescription-pResolveAttachments-02649
  All attachments in `pResolveAttachments` that are not `VK_ATTACHMENT_UNUSED` **must** have image formats whose potential format features contain `VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BIT`
- VUID-VkSubpassDescription-pDepthStencilAttachment-02650
  If `pDepthStencilAttachment` is not `NULL` and the attachment is not `VK_ATTACHMENT_UNUSED` then it **must** have an image format whose potential format features contain `VK_FORMAT_FEATURE_DEPTH_STENCIL_ATTACHMENT_BIT`
- VUID-VkSubpassDescription-linearColorAttachment-06496
  If the `linearColorAttachment` feature is enabled and the image is created with `VK_IMAGE_TILING_LINEAR`, all attachments in `pInputAttachments` that are not `VK_ATTACHMENT_UNUSED` **must** have image formats whose potential format features **must** contain `VK_FORMAT_FEATURE_2_LINEAR_COLOR_ATTACHMENT_BIT_NV`
- VUID-VkSubpassDescription-linearColorAttachment-06497
  If the `linearColorAttachment` feature is enabled and the image is created with `VK_IMAGE_TILING_LINEAR`, all attachments in `pColorAttachments` that are not `VK_ATTACHMENT_UNUSED` **must** have image formats whose potential format features **must** contain `VK_FORMAT_FEATURE_2_LINEAR_COLOR_ATTACHMENT_BIT_NV`
- VUID-VkSubpassDescription-linearColorAttachment-06498
  If the `linearColorAttachment` feature is enabled and the image is created with `VK_IMAGE_TILING_LINEAR`, all attachments in `pResolveAttachments` that are not `VK_ATTACHMENT_UNUSED` **must** have image formats whose potential format features **must**

contain VK_FORMAT_FEATURE_2_LINEAR_COLOR_ATTACHMENT_BIT_NV

- VUID-VkSubpassDescription-None-09431
  If either of the following is enabled:

  - The VK_AMD_mixed_attachment_samples extension
  - The VK_NV_framebuffer_mixed_samples extension

  all attachments in pColorAttachments that are not VK_ATTACHMENT_UNUSED **must** have a sample count that is smaller than or equal to the sample count of pDepthStencilAttachment if it is not VK_ATTACHMENT_UNUSED

- VUID-VkSubpassDescription-pDepthStencilAttachment-01418
  If pDepthStencilAttachment is not VK_ATTACHMENT_UNUSED and any attachments in pColorAttachments are not VK_ATTACHMENT_UNUSED, they **must** have the same sample count , if none of the following are enabled:

  - The VK_AMD_mixed_attachment_samples extension
  - The VK_NV_framebuffer_mixed_samples extension

- VUID-VkSubpassDescription-attachment-00853
  Each element of pPreserveAttachments **must** not be VK_ATTACHMENT_UNUSED

- VUID-VkSubpassDescription-pPreserveAttachments-00854
  Each element of pPreserveAttachments **must** not also be an element of any other member of the subpass description

- VUID-VkSubpassDescription-layout-02519
  If any attachment is used by more than one VkAttachmentReference member, then each use **must** use the same layout

- VUID-VkSubpassDescription-flags-00856
  If flags includes VK_SUBPASS_DESCRIPTION_PER_VIEW_POSITION_X_ONLY_BIT_NVX, it **must** also include VK_SUBPASS_DESCRIPTION_PER_VIEW_ATTRIBUTES_BIT_NVX

- VUID-VkSubpassDescription-flags-03341
  If flags includes VK_SUBPASS_DESCRIPTION_SHADER_RESOLVE_BIT_QCOM, and if pResolveAttachments is not NULL, then each resolve attachment **must** be VK_ATTACHMENT_UNUSED

- VUID-VkSubpassDescription-flags-03343
  If flags includes VK_SUBPASS_DESCRIPTION_SHADER_RESOLVE_BIT_QCOM, then the subpass **must** be the last subpass in a subpass dependency chain

- VUID-VkSubpassDescription-pInputAttachments-02868
  If the render pass is created with VK_RENDER_PASS_CREATE_TRANSFORM_BIT_QCOM each of the elements of pInputAttachments **must** be VK_ATTACHMENT_UNUSED

- VUID-VkSubpassDescription-pDepthStencilAttachment-04438
  pDepthStencilAttachment and pColorAttachments **must** not contain references to the same attachment

### Valid Usage (Implicit)

- VUID-VkSubpassDescription-flags-parameter
  `flags` **must** be a valid combination of VkSubpassDescriptionFlagBits values
- VUID-VkSubpassDescription-pipelineBindPoint-parameter
  `pipelineBindPoint` **must** be a valid VkPipelineBindPoint value
- VUID-VkSubpassDescription-pInputAttachments-parameter
  If `inputAttachmentCount` is not `0`, `pInputAttachments` **must** be a valid pointer to an array of `inputAttachmentCount` valid VkAttachmentReference structures
- VUID-VkSubpassDescription-pColorAttachments-parameter
  If `colorAttachmentCount` is not `0`, `pColorAttachments` **must** be a valid pointer to an array of `colorAttachmentCount` valid VkAttachmentReference structures
- VUID-VkSubpassDescription-pResolveAttachments-parameter
  If `colorAttachmentCount` is not `0`, and `pResolveAttachments` is not `NULL`, `pResolveAttachments` **must** be a valid pointer to an array of `colorAttachmentCount` valid VkAttachmentReference structures
- VUID-VkSubpassDescription-pDepthStencilAttachment-parameter
  If `pDepthStencilAttachment` is not `NULL`, `pDepthStencilAttachment` **must** be a valid pointer to a valid VkAttachmentReference structure
- VUID-VkSubpassDescription-pPreserveAttachments-parameter
  If `preserveAttachmentCount` is not `0`, `pPreserveAttachments` **must** be a valid pointer to an array of `preserveAttachmentCount` `uint32_t` values

Bits which **can** be set in VkSubpassDescription::`flags`, specifying usage of the subpass, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkSubpassDescriptionFlagBits {
  // Provided by VK_NVX_multiview_per_view_attributes
    VK_SUBPASS_DESCRIPTION_PER_VIEW_ATTRIBUTES_BIT_NVX = 0x00000001,
  // Provided by VK_NVX_multiview_per_view_attributes
    VK_SUBPASS_DESCRIPTION_PER_VIEW_POSITION_X_ONLY_BIT_NVX = 0x00000002,
  // Provided by VK_QCOM_render_pass_shader_resolve
    VK_SUBPASS_DESCRIPTION_FRAGMENT_REGION_BIT_QCOM = 0x00000004,
  // Provided by VK_QCOM_render_pass_shader_resolve
    VK_SUBPASS_DESCRIPTION_SHADER_RESOLVE_BIT_QCOM = 0x00000008,
  // Provided by VK_EXT_rasterization_order_attachment_access
    VK_SUBPASS_DESCRIPTION_RASTERIZATION_ORDER_ATTACHMENT_COLOR_ACCESS_BIT_EXT =
0x00000010,
  // Provided by VK_EXT_rasterization_order_attachment_access
    VK_SUBPASS_DESCRIPTION_RASTERIZATION_ORDER_ATTACHMENT_DEPTH_ACCESS_BIT_EXT =
0x00000020,
  // Provided by VK_EXT_rasterization_order_attachment_access
    VK_SUBPASS_DESCRIPTION_RASTERIZATION_ORDER_ATTACHMENT_STENCIL_ACCESS_BIT_EXT =
0x00000040,
  // Provided by VK_EXT_legacy_dithering
    VK_SUBPASS_DESCRIPTION_ENABLE_LEGACY_DITHERING_BIT_EXT = 0x00000080,
  // Provided by VK_ARM_rasterization_order_attachment_access
    VK_SUBPASS_DESCRIPTION_RASTERIZATION_ORDER_ATTACHMENT_COLOR_ACCESS_BIT_ARM =
VK_SUBPASS_DESCRIPTION_RASTERIZATION_ORDER_ATTACHMENT_COLOR_ACCESS_BIT_EXT,
  // Provided by VK_ARM_rasterization_order_attachment_access
    VK_SUBPASS_DESCRIPTION_RASTERIZATION_ORDER_ATTACHMENT_DEPTH_ACCESS_BIT_ARM =
VK_SUBPASS_DESCRIPTION_RASTERIZATION_ORDER_ATTACHMENT_DEPTH_ACCESS_BIT_EXT,
  // Provided by VK_ARM_rasterization_order_attachment_access
    VK_SUBPASS_DESCRIPTION_RASTERIZATION_ORDER_ATTACHMENT_STENCIL_ACCESS_BIT_ARM =
VK_SUBPASS_DESCRIPTION_RASTERIZATION_ORDER_ATTACHMENT_STENCIL_ACCESS_BIT_EXT,
} VkSubpassDescriptionFlagBits;
```

- `VK_SUBPASS_DESCRIPTION_PER_VIEW_ATTRIBUTES_BIT_NVX` specifies that shaders compiled for this subpass write the attributes for all views in a single invocation of each pre-rasterization shader stage. All pipelines compiled against a subpass that includes this bit **must** write per-view attributes to the `*PerViewNV[]` shader outputs, in addition to the non-per-view (e.g. `Position`) outputs.
- `VK_SUBPASS_DESCRIPTION_PER_VIEW_POSITION_X_ONLY_BIT_NVX` specifies that shaders compiled for this subpass use per-view positions which only differ in value in the x component. Per-view viewport mask **can** also be used.
- `VK_SUBPASS_DESCRIPTION_FRAGMENT_REGION_BIT_QCOM` specifies that the framebuffer region is the fragment region, that is, the minimum region dependencies are by pixel rather than by sample, such that any fragment shader invocation **can** access any sample associated with that fragment shader invocation.
- `VK_SUBPASS_DESCRIPTION_SHADER_RESOLVE_BIT_QCOM` specifies that the subpass performs shader resolve operations.
- `VK_SUBPASS_DESCRIPTION_RASTERIZATION_ORDER_ATTACHMENT_COLOR_ACCESS_BIT_EXT` specifies that this

subpass supports pipelines created with VK_PIPELINE_COLOR_BLEND_STATE_CREATE_RASTERIZATION_ORDER_ATTACHMENT_ACCESS_BIT_EXT.

- VK_SUBPASS_DESCRIPTION_RASTERIZATION_ORDER_ATTACHMENT_DEPTH_ACCESS_BIT_EXT specifies that this subpass supports pipelines created with VK_PIPELINE_DEPTH_STENCIL_STATE_CREATE_RASTERIZATION_ORDER_ATTACHMENT_DEPTH_ACCESS_BIT_EXT.
- VK_SUBPASS_DESCRIPTION_RASTERIZATION_ORDER_ATTACHMENT_STENCIL_ACCESS_BIT_EXT specifies that this subpass supports pipelines created with VK_PIPELINE_DEPTH_STENCIL_STATE_CREATE_RASTERIZATION_ORDER_ATTACHMENT_STENCIL_ACCESS_BIT_EXT.
- VK_SUBPASS_DESCRIPTION_ENABLE_LEGACY_DITHERING_BIT_EXT specifies that Legacy Dithering is enabled for this subpass.

**NOTE** Shader resolve operations allow for custom resolve operations, but overdrawing pixels **may** have a performance and/or power cost. Furthermore, since the content of any depth stencil attachment or color attachment is undefined at the beginning of a shader resolve subpass, any depth testing, stencil testing, or blending operation which sources these undefined values also has undefined result value.

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkSubpassDescriptionFlags;
```

VkSubpassDescriptionFlags is a bitmask type for setting a mask of zero or more VkSubpassDescriptionFlagBits.

The VkAttachmentReference structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkAttachmentReference {
    uint32_t         attachment;
    VkImageLayout    layout;
} VkAttachmentReference;
```

- attachment is either an integer value identifying an attachment at the corresponding index in VkRenderPassCreateInfo::pAttachments, or VK_ATTACHMENT_UNUSED to signify that this attachment is not used.
- layout is a VkImageLayout value specifying the layout the attachment uses during the subpass.

### Valid Usage

- VUID-VkAttachmentReference-layout-03077
  If attachment is not VK_ATTACHMENT_UNUSED, layout **must** not be VK_IMAGE_LAYOUT_UNDEFINED, VK_IMAGE_LAYOUT_PREINITIALIZED, or VK_IMAGE_LAYOUT_PRESENT_SRC_KHR
- VUID-VkAttachmentReference-separateDepthStencilLayouts-03313
  If the separateDepthStencilLayouts feature is not enabled, and attachment is not

VK_ATTACHMENT_UNUSED, layout **must** not be VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL, VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL, VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL, or VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL,

- VUID-VkAttachmentReference-synchronization2-06910
  If the synchronization2 feature is not enabled, layout **must** not be VK_IMAGE_LAYOUT_ATTACHMENT_OPTIMAL_KHR or VK_IMAGE_LAYOUT_READ_ONLY_OPTIMAL_KHR
- VUID-VkAttachmentReference-attachmentFeedbackLoopLayout-07311
  If the attachmentFeedbackLoopLayout feature is not enabled, layout **must** not be VK_IMAGE_LAYOUT_ATTACHMENT_FEEDBACK_LOOP_OPTIMAL_EXT
- VUID-VkAttachmentReference-dynamicRenderingLocalRead-09546
  If the dynamicRenderingLocalRead feature is not enabled, layout **must** not be VK_IMAGE_LAYOUT_RENDERING_LOCAL_READ_KHR

### Valid Usage (Implicit)

- VUID-VkAttachmentReference-layout-parameter
  layout **must** be a valid VkImageLayout value

VK_SUBPASS_EXTERNAL is a special subpass index value expanding synchronization scope outside a subpass. It is described in more detail by VkSubpassDependency.

```
#define VK_SUBPASS_EXTERNAL             (~0U)
```

The VkSubpassDependency structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkSubpassDependency {
    uint32_t                srcSubpass;
    uint32_t                dstSubpass;
    VkPipelineStageFlags    srcStageMask;
    VkPipelineStageFlags    dstStageMask;
    VkAccessFlags           srcAccessMask;
    VkAccessFlags           dstAccessMask;
    VkDependencyFlags       dependencyFlags;
} VkSubpassDependency;
```

- srcSubpass is the subpass index of the first subpass in the dependency, or VK_SUBPASS_EXTERNAL.
- dstSubpass is the subpass index of the second subpass in the dependency, or VK_SUBPASS_EXTERNAL.
- srcStageMask is a bitmask of VkPipelineStageFlagBits specifying the source stage mask.
- dstStageMask is a bitmask of VkPipelineStageFlagBits specifying the destination stage mask
- srcAccessMask is a bitmask of VkAccessFlagBits specifying a source access mask.

- dstAccessMask is a bitmask of VkAccessFlagBits specifying a destination access mask.
- dependencyFlags is a bitmask of VkDependencyFlagBits.

If srcSubpass is equal to dstSubpass then the VkSubpassDependency does not directly define a dependency. Instead, it enables pipeline barriers to be used in a render pass instance within the identified subpass, where the scopes of one pipeline barrier **must** be a subset of those described by one subpass dependency. Subpass dependencies specified in this way that include framebuffer-space stages in the srcStageMask **must** only include framebuffer-space stages in dstStageMask, and **must** include VK_DEPENDENCY_BY_REGION_BIT. When a subpass dependency is specified in this way for a subpass that has more than one view in its view mask, its dependencyFlags **must** include VK_DEPENDENCY_VIEW_LOCAL_BIT.

If srcSubpass and dstSubpass are not equal, when a render pass instance which includes a subpass dependency is submitted to a queue, it defines a dependency between the subpasses identified by srcSubpass and dstSubpass.

If srcSubpass is equal to VK_SUBPASS_EXTERNAL, the first synchronization scope includes commands that occur earlier in submission order than the vkCmdBeginRenderPass used to begin the render pass instance. Otherwise, the first set of commands includes all commands submitted as part of the subpass instance identified by srcSubpass and any load, store, or multisample resolve operations on attachments used in srcSubpass. In either case, the first synchronization scope is limited to operations on the pipeline stages determined by the source stage mask specified by srcStageMask.

If dstSubpass is equal to VK_SUBPASS_EXTERNAL, the second synchronization scope includes commands that occur later in submission order than the vkCmdEndRenderPass used to end the render pass instance. Otherwise, the second set of commands includes all commands submitted as part of the subpass instance identified by dstSubpass and any load, store, and multisample resolve operations on attachments used in dstSubpass. In either case, the second synchronization scope is limited to operations on the pipeline stages determined by the destination stage mask specified by dstStageMask.

The first access scope is limited to accesses in the pipeline stages determined by the source stage mask specified by srcStageMask. It is also limited to access types in the source access mask specified by srcAccessMask.

The second access scope is limited to accesses in the pipeline stages determined by the destination stage mask specified by dstStageMask. It is also limited to access types in the destination access mask specified by dstAccessMask.

The availability and visibility operations defined by a subpass dependency affect the execution of image layout transitions within the render pass.

**NOTE** For non-attachment resources, the memory dependency expressed by subpass dependency is nearly identical to that of a VkMemoryBarrier (with matching srcAccessMask and dstAccessMask parameters) submitted as a part of a vkCmdPipelineBarrier (with matching srcStageMask and dstStageMask parameters). The only difference being that its scopes are limited to the identified subpasses rather than potentially affecting everything before and after.

For attachments however, subpass dependencies work more like a VkImageMemoryBarrier defined similarly to the VkMemoryBarrier above, the queue family indices set to VK_QUEUE_FAMILY_IGNORED, and layouts as follows:

- The equivalent to oldLayout is the attachment's layout according to the subpass description for srcSubpass.
- The equivalent to newLayout is the attachment's layout according to the subpass description for dstSubpass.

## Valid Usage

- VUID-VkSubpassDependency-srcStageMask-04090
  If the geometryShader feature is not enabled, srcStageMask **must** not contain VK_PIPELINE_STAGE_GEOMETRY_SHADER_BIT
- VUID-VkSubpassDependency-srcStageMask-04091
  If the tessellationShader feature is not enabled, srcStageMask **must** not contain VK_PIPELINE_STAGE_TESSELLATION_CONTROL_SHADER_BIT or VK_PIPELINE_STAGE_TESSELLATION_EVALUATION_SHADER_BIT
- VUID-VkSubpassDependency-srcStageMask-04092
  If the conditionalRendering feature is not enabled, srcStageMask **must** not contain VK_PIPELINE_STAGE_CONDITIONAL_RENDERING_BIT_EXT
- VUID-VkSubpassDependency-srcStageMask-04093
  If the fragmentDensityMap feature is not enabled, srcStageMask **must** not contain VK_PIPELINE_STAGE_FRAGMENT_DENSITY_PROCESS_BIT_EXT
- VUID-VkSubpassDependency-srcStageMask-04094
  If the transformFeedback feature is not enabled, srcStageMask **must** not contain VK_PIPELINE_STAGE_TRANSFORM_FEEDBACK_BIT_EXT
- VUID-VkSubpassDependency-srcStageMask-04095
  If the meshShader feature is not enabled, srcStageMask **must** not contain VK_PIPELINE_STAGE_MESH_SHADER_BIT_EXT
- VUID-VkSubpassDependency-srcStageMask-04096
  If the taskShader feature is not enabled, srcStageMask **must** not contain VK_PIPELINE_STAGE_TASK_SHADER_BIT_EXT
- VUID-VkSubpassDependency-srcStageMask-07318
  If neither the shadingRateImage or attachmentFragmentShadingRate are enabled, srcStageMask **must** not contain VK_PIPELINE_STAGE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR
- VUID-VkSubpassDependency-srcStageMask-03937
  If the synchronization2 feature is not enabled, srcStageMask **must** not be 0
- VUID-VkSubpassDependency-srcStageMask-07949
  If neither the VK_NV_ray_tracing extension or rayTracingPipeline feature are enabled, srcStageMask **must** not contain VK_PIPELINE_STAGE_RAY_TRACING_SHADER_BIT_KHR
- VUID-VkSubpassDependency-dstStageMask-04090

If the geometryShader feature is not enabled, dstStageMask **must** not contain VK_PIPELINE_STAGE_GEOMETRY_SHADER_BIT

- VUID-VkSubpassDependency-dstStageMask-04091
  If the tessellationShader feature is not enabled, dstStageMask **must** not contain VK_PIPELINE_STAGE_TESSELLATION_CONTROL_SHADER_BIT or VK_PIPELINE_STAGE_TESSELLATION_EVALUATION_SHADER_BIT
- VUID-VkSubpassDependency-dstStageMask-04092
  If the conditionalRendering feature is not enabled, dstStageMask **must** not contain VK_PIPELINE_STAGE_CONDITIONAL_RENDERING_BIT_EXT
- VUID-VkSubpassDependency-dstStageMask-04093
  If the fragmentDensityMap feature is not enabled, dstStageMask **must** not contain VK_PIPELINE_STAGE_FRAGMENT_DENSITY_PROCESS_BIT_EXT
- VUID-VkSubpassDependency-dstStageMask-04094
  If the transformFeedback feature is not enabled, dstStageMask **must** not contain VK_PIPELINE_STAGE_TRANSFORM_FEEDBACK_BIT_EXT
- VUID-VkSubpassDependency-dstStageMask-04095
  If the meshShader feature is not enabled, dstStageMask **must** not contain VK_PIPELINE_STAGE_MESH_SHADER_BIT_EXT
- VUID-VkSubpassDependency-dstStageMask-04096
  If the taskShader feature is not enabled, dstStageMask **must** not contain VK_PIPELINE_STAGE_TASK_SHADER_BIT_EXT
- VUID-VkSubpassDependency-dstStageMask-07318
  If neither the shadingRateImage or attachmentFragmentShadingRate are enabled, dstStageMask **must** not contain VK_PIPELINE_STAGE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR
- VUID-VkSubpassDependency-dstStageMask-03937
  If the synchronization2 feature is not enabled, dstStageMask **must** not be 0
- VUID-VkSubpassDependency-dstStageMask-07949
  If neither the VK_NV_ray_tracing extension or rayTracingPipeline feature are enabled, dstStageMask **must** not contain VK_PIPELINE_STAGE_RAY_TRACING_SHADER_BIT_KHR
- VUID-VkSubpassDependency-srcSubpass-00864
  srcSubpass **must** be less than or equal to dstSubpass, unless one of them is VK_SUBPASS_EXTERNAL, to avoid cyclic dependencies and ensure a valid execution order
- VUID-VkSubpassDependency-srcSubpass-00865
  srcSubpass and dstSubpass **must** not both be equal to VK_SUBPASS_EXTERNAL
- VUID-VkSubpassDependency-srcSubpass-06809
  If srcSubpass is equal to dstSubpass and srcStageMask includes a framebuffer-space stage, dstStageMask **must** only contain framebuffer-space stages
- VUID-VkSubpassDependency-srcAccessMask-00868
  Any access flag included in srcAccessMask **must** be supported by one of the pipeline stages in srcStageMask, as specified in the table of supported access types
- VUID-VkSubpassDependency-dstAccessMask-00869

Any access flag included in `dstAccessMask` **must** be supported by one of the pipeline stages in `dstStageMask`, as specified in the table of supported access types

- VUID-VkSubpassDependency-srcSubpass-02243
  If `srcSubpass` equals `dstSubpass`, and `srcStageMask` and `dstStageMask` both include a framebuffer-space stage, then `dependencyFlags` **must** include `VK_DEPENDENCY_BY_REGION_BIT`
- VUID-VkSubpassDependency-dependencyFlags-02520
  If `dependencyFlags` includes `VK_DEPENDENCY_VIEW_LOCAL_BIT`, `srcSubpass` **must** not be equal to `VK_SUBPASS_EXTERNAL`
- VUID-VkSubpassDependency-dependencyFlags-02521
  If `dependencyFlags` includes `VK_DEPENDENCY_VIEW_LOCAL_BIT`, `dstSubpass` **must** not be equal to `VK_SUBPASS_EXTERNAL`
- VUID-VkSubpassDependency-srcSubpass-00872
  If `srcSubpass` equals `dstSubpass` and that subpass has more than one bit set in the view mask, then `dependencyFlags` **must** include `VK_DEPENDENCY_VIEW_LOCAL_BIT`

### Valid Usage (Implicit)

- VUID-VkSubpassDependency-srcStageMask-parameter
  `srcStageMask` **must** be a valid combination of VkPipelineStageFlagBits values
- VUID-VkSubpassDependency-dstStageMask-parameter
  `dstStageMask` **must** be a valid combination of VkPipelineStageFlagBits values
- VUID-VkSubpassDependency-srcAccessMask-parameter
  `srcAccessMask` **must** be a valid combination of VkAccessFlagBits values
- VUID-VkSubpassDependency-dstAccessMask-parameter
  `dstAccessMask` **must** be a valid combination of VkAccessFlagBits values
- VUID-VkSubpassDependency-dependencyFlags-parameter
  `dependencyFlags` **must** be a valid combination of VkDependencyFlagBits values

When multiview is enabled, the execution of the multiple views of one subpass **may** not occur simultaneously or even back-to-back, and rather **may** be interleaved with the execution of other subpasses. The load and store operations apply to attachments on a per-view basis. For example, an attachment using `VK_ATTACHMENT_LOAD_OP_CLEAR` will have each view cleared on first use, but the first use of one view may be temporally distant from the first use of another view.

**NOTE**

A good mental model for multiview is to think of a multiview subpass as if it were a collection of individual (per-view) subpasses that are logically grouped together and described as a single multiview subpass in the API. Similarly, a multiview attachment can be thought of like several individual attachments that happen to be layers in a single image. A view-local dependency between two multiview subpasses acts like a set of one-to-one dependencies between corresponding pairs of per-view subpasses. A view-global dependency between two multiview subpasses acts like a set of N × M dependencies between all pairs of per-view subpasses in the source and destination. Thus, it is a more compact representation which also makes clear the

> commonality and reuse that is present between views in a subpass. This interpretation motivates the answers to questions like “when does the load op apply” - it is on the first use of each view of an attachment, as if each view was a separate attachment.
>
> The content of each view follows the description in attachment content behavior. In particular, if an attachment is preserved, all views within the attachment are preserved.

If any two subpasses of a render pass activate transform feedback to the same bound transform feedback buffers, a subpass dependency **must** be included (either directly or via some intermediate subpasses) between them.

If there is no subpass dependency from `VK_SUBPASS_EXTERNAL` to the first subpass that uses an attachment, then an implicit subpass dependency exists from `VK_SUBPASS_EXTERNAL` to the first subpass it is used in. The implicit subpass dependency only exists if there exists an automatic layout transition away from `initialLayout`. The subpass dependency operates as if defined with the following parameters:

```
VkSubpassDependency implicitDependency = {
    .srcSubpass = VK_SUBPASS_EXTERNAL,
    .dstSubpass = firstSubpass, // First subpass attachment is used in
    .srcStageMask = VK_PIPELINE_STAGE_NONE,
    .dstStageMask = VK_PIPELINE_STAGE_ALL_COMMANDS_BIT,
    .srcAccessMask = 0,
    .dstAccessMask = VK_ACCESS_INPUT_ATTACHMENT_READ_BIT |
                     VK_ACCESS_COLOR_ATTACHMENT_READ_BIT |
                     VK_ACCESS_COLOR_ATTACHMENT_WRITE_BIT |
                     VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_READ_BIT |
                     VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_WRITE_BIT,
    .dependencyFlags = 0
};
```

Similarly, if there is no subpass dependency from the last subpass that uses an attachment to `VK_SUBPASS_EXTERNAL`, then an implicit subpass dependency exists from the last subpass it is used in to `VK_SUBPASS_EXTERNAL`. The implicit subpass dependency only exists if there exists an automatic layout transition into `finalLayout`. The subpass dependency operates as if defined with the following parameters:

```
VkSubpassDependency implicitDependency = {
    .srcSubpass = lastSubpass, // Last subpass attachment is used in
    .dstSubpass = VK_SUBPASS_EXTERNAL,
    .srcStageMask = VK_PIPELINE_STAGE_ALL_COMMANDS_BIT,
    .dstStageMask = VK_PIPELINE_STAGE_NONE,
    .srcAccessMask = VK_ACCESS_COLOR_ATTACHMENT_WRITE_BIT |
                     VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_WRITE_BIT,
    .dstAccessMask = 0,
    .dependencyFlags = 0
```

```
};
```

As subpasses **may** overlap or execute out of order with regards to other subpasses unless a subpass dependency chain describes otherwise, the layout transitions required between subpasses **cannot** be known to an application. Instead, an application provides the layout that each attachment **must** be in at the start and end of a render pass, and the layout it **must** be in during each subpass it is used in. The implementation then **must** execute layout transitions between subpasses in order to guarantee that the images are in the layouts required by each subpass, and in the final layout at the end of the render pass.

Automatic layout transitions apply to the entire image subresource attached to the framebuffer. If multiview is not enabled and the attachment is a view of a 1D or 2D image, the automatic layout transitions apply to the number of layers specified by VkFramebufferCreateInfo::`layers`. If multiview is enabled and the attachment is a view of a 1D or 2D image, the automatic layout transitions apply to the layers corresponding to views which are used by some subpass in the render pass, even if that subpass does not reference the given attachment. If the attachment view is a 2D or 2D array view of a 3D image, even if the attachment view only refers to a subset of the slices of the selected mip level of the 3D image, automatic layout transitions apply to the entire subresource referenced which is the entire mip level in this case.

Automatic layout transitions away from the layout used in a subpass happen-after the availability operations for all dependencies with that subpass as the `srcSubpass`.

Automatic layout transitions into the layout used in a subpass happen-before the visibility operations for all dependencies with that subpass as the `dstSubpass`.

Automatic layout transitions away from `initialLayout` happen-after the availability operations for all dependencies with a `srcSubpass` equal to `VK_SUBPASS_EXTERNAL`, where `dstSubpass` uses the attachment that will be transitioned. For attachments created with `VK_ATTACHMENT_DESCRIPTION_MAY_ALIAS_BIT`, automatic layout transitions away from `initialLayout` happen-after the availability operations for all dependencies with a `srcSubpass` equal to `VK_SUBPASS_EXTERNAL`, where `dstSubpass` uses any aliased attachment.

Automatic layout transitions into `finalLayout` happen-before the visibility operations for all dependencies with a `dstSubpass` equal to `VK_SUBPASS_EXTERNAL`, where `srcSubpass` uses the attachment that will be transitioned. For attachments created with `VK_ATTACHMENT_DESCRIPTION_MAY_ALIAS_BIT`, automatic layout transitions into `finalLayout` happen-before the visibility operations for all dependencies with a `dstSubpass` equal to `VK_SUBPASS_EXTERNAL`, where `srcSubpass` uses any aliased attachment.

The image layout of the depth aspect of a depth/stencil attachment referring to an image created with `VK_IMAGE_CREATE_SAMPLE_LOCATIONS_COMPATIBLE_DEPTH_BIT_EXT` is dependent on the last sample locations used to render to the attachment, thus automatic layout transitions use the sample locations state specified in VkRenderPassSampleLocationsBeginInfoEXT.

Automatic layout transitions of an attachment referring to a depth/stencil image created with `VK_IMAGE_CREATE_SAMPLE_LOCATIONS_COMPATIBLE_DEPTH_BIT_EXT` use the sample locations the image subresource range referenced by the attachment was last rendered with. If the current render pass does not use the attachment as a depth/stencil attachment in any subpass that happens-before, the

automatic layout transition uses the sample locations state specified in the sampleLocationsInfo member of the element of the VkRenderPassSampleLocationsBeginInfoEXT::pAttachmentInitialSampleLocations array for which the attachmentIndex member equals the attachment index of the attachment, if one is specified. Otherwise, the automatic layout transition uses the sample locations state specified in the sampleLocationsInfo member of the element of the VkRenderPassSampleLocationsBeginInfoEXT::pPostSubpassSampleLocations array for which the subpassIndex member equals the index of the subpass that last used the attachment as a depth/stencil attachment, if one is specified.

If no sample locations state has been specified for an automatic layout transition performed on an attachment referring to a depth/stencil image created with VK_IMAGE_CREATE_SAMPLE_LOCATIONS_COMPATIBLE_DEPTH_BIT_EXT the contents of the depth aspect of the depth/stencil attachment become undefined as if the layout of the attachment was transitioned from the VK_IMAGE_LAYOUT_UNDEFINED layout.

If two subpasses use the same attachment, and both subpasses use the attachment in a read-only layout, no subpass dependency needs to be specified between those subpasses. If an implementation treats those layouts separately, it **must** insert an implicit subpass dependency between those subpasses to separate the uses in each layout. The subpass dependency operates as if defined with the following parameters:

```
// Used for input attachments
VkPipelineStageFlags inputAttachmentStages = VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT;
VkAccessFlags inputAttachmentDstAccess = VK_ACCESS_INPUT_ATTACHMENT_READ_BIT;

// Used for depth/stencil attachments
VkPipelineStageFlags depthStencilAttachmentStages =
VK_PIPELINE_STAGE_EARLY_FRAGMENT_TESTS_BIT |
VK_PIPELINE_STAGE_LATE_FRAGMENT_TESTS_BIT;
VkAccessFlags depthStencilAttachmentDstAccess =
VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_READ_BIT;

VkSubpassDependency implicitDependency = {
    .srcSubpass = firstSubpass;
    .dstSubpass = secondSubpass;
    .srcStageMask = inputAttachmentStages | depthStencilAttachmentStages;
    .dstStageMask = inputAttachmentStages | depthStencilAttachmentStages;
    .srcAccessMask = 0;
    .dstAccessMask = inputAttachmentDstAccess | depthStencilAttachmentDstAccess;
    .dependencyFlags = 0;
};
```

When drawing using shader objects, or when the graphics pipeline is created with VK_DYNAMIC_STATE_ATTACHMENT_FEEDBACK_LOOP_ENABLE_EXT set in VkPipelineDynamicStateCreateInfo::pDynamicStates, the application **must** specify which types of attachments that are written to during a render pass will also be accessed as non-attachments in the render pass.

To dynamically set whether a pipeline **can** access a resource as a non-attachment while it is also used as an attachment that is written to, call:

```
// Provided by VK_EXT_attachment_feedback_loop_dynamic_state
void vkCmdSetAttachmentFeedbackLoopEnableEXT(
    VkCommandBuffer                             commandBuffer,
    VkImageAspectFlags                          aspectMask);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `aspectMask` specifies the types of attachments for which feedback loops will be enabled. Attachment types whose aspects are not included in `aspectMask` will have feedback loops disabled.

For attachments that are written to in a render pass, only attachments with the aspects specified in `aspectMask` **can** be accessed as non-attachments by subsequent drawing commands.

### Valid Usage

- VUID-vkCmdSetAttachmentFeedbackLoopEnableEXT-attachmentFeedbackLoopDynamicState-08862
  The `attachmentFeedbackLoopDynamicState` feature **must** be enabled
- VUID-vkCmdSetAttachmentFeedbackLoopEnableEXT-aspectMask-08863
  `aspectMask` **must** only include `VK_IMAGE_ASPECT_NONE`, `VK_IMAGE_ASPECT_COLOR_BIT`, `VK_IMAGE_ASPECT_DEPTH_BIT`, and `VK_IMAGE_ASPECT_STENCIL_BIT`
- VUID-vkCmdSetAttachmentFeedbackLoopEnableEXT-attachmentFeedbackLoopLayout-08864
  If the `attachmentFeedbackLoopLayout` feature is not enabled, `aspectMask` **must** be `VK_IMAGE_ASPECT_NONE`

### Valid Usage (Implicit)

- VUID-vkCmdSetAttachmentFeedbackLoopEnableEXT-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdSetAttachmentFeedbackLoopEnableEXT-aspectMask-parameter
  `aspectMask` **must** be a valid combination of VkImageAspectFlagBits values
- VUID-vkCmdSetAttachmentFeedbackLoopEnableEXT-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdSetAttachmentFeedbackLoopEnableEXT-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- VUID-vkCmdSetAttachmentFeedbackLoopEnableEXT-videocoding
  This command **must** only be called outside of a video coding scope

### Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

### Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics | State |

A more extensible version of render pass creation is also defined below.

To create a render pass, call:

```
// Provided by VK_KHR_create_renderpass2
VkResult vkCreateRenderPass2KHR(
    VkDevice                                    device,
    const VkRenderPassCreateInfo2*              pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkRenderPass*                               pRenderPass);
```

- `device` is the logical device that creates the render pass.
- `pCreateInfo` is a pointer to a VkRenderPassCreateInfo2 structure describing the parameters of the render pass.
- `pAllocator` controls host memory allocation as described in the Memory Allocation chapter.
- `pRenderPass` is a pointer to a VkRenderPass handle in which the resulting render pass object is returned.

This command is functionally identical to vkCreateRenderPass, but includes extensible sub-structures that include `sType` and `pNext` parameters, allowing them to be more easily extended.

### Valid Usage

- VUID-vkCreateRenderPass2-device-10001
  `device` **must** support at least one queue family with the `VK_QUEUE_GRAPHICS_BIT` capability

### Valid Usage (Implicit)

- VUID-vkCreateRenderPass2-device-parameter

device **must** be a valid VkDevice handle

- VUID-vkCreateRenderPass2-pCreateInfo-parameter
  pCreateInfo **must** be a valid pointer to a valid VkRenderPassCreateInfo2 structure
- VUID-vkCreateRenderPass2-pAllocator-parameter
  If pAllocator is not NULL, pAllocator **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkCreateRenderPass2-pRenderPass-parameter
  pRenderPass **must** be a valid pointer to a VkRenderPass handle

## Return Codes

**Success**

- VK_SUCCESS

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY

The VkRenderPassCreateInfo2 structure is defined as:

```
typedef struct VkRenderPassCreateInfo2 {
    VkStructureType                    sType;
    const void*                        pNext;
    VkRenderPassCreateFlags            flags;
    uint32_t                           attachmentCount;
    const VkAttachmentDescription2*    pAttachments;
    uint32_t                           subpassCount;
    const VkSubpassDescription2*       pSubpasses;
    uint32_t                           dependencyCount;
    const VkSubpassDependency2*        pDependencies;
    uint32_t                           correlatedViewMaskCount;
    const uint32_t*                    pCorrelatedViewMasks;
} VkRenderPassCreateInfo2;
```

or the equivalent

```
// Provided by VK_KHR_create_renderpass2
typedef VkRenderPassCreateInfo2 VkRenderPassCreateInfo2KHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- flags is reserved for future use.
- attachmentCount is the number of attachments used by this render pass.

- pAttachments is a pointer to an array of attachmentCount VkAttachmentDescription2 structures describing the attachments used by the render pass.
- subpassCount is the number of subpasses to create.
- pSubpasses is a pointer to an array of subpassCount VkSubpassDescription2 structures describing each subpass.
- dependencyCount is the number of dependencies between pairs of subpasses.
- pDependencies is a pointer to an array of dependencyCount VkSubpassDependency2 structures describing dependencies between pairs of subpasses.
- correlatedViewMaskCount is the number of correlation masks.
- pCorrelatedViewMasks is a pointer to an array of view masks indicating sets of views that **may** be more efficient to render concurrently.

Parameters defined by this structure with the same name as those in VkRenderPassCreateInfo have the identical effect to those parameters; the child structures are variants of those used in VkRenderPassCreateInfo which add sType and pNext parameters, allowing them to be extended.

If the VkSubpassDescription2::viewMask member of any element of pSubpasses is not zero, *multiview* functionality is considered to be enabled for this render pass.

correlatedViewMaskCount and pCorrelatedViewMasks have the same effect as VkRenderPassMultiviewCreateInfo::correlationMaskCount and VkRenderPassMultiviewCreateInfo::pCorrelationMasks, respectively.

## Valid Usage

- VUID-VkRenderPassCreateInfo2-None-03049
  If any two subpasses operate on attachments with overlapping ranges of the same VkDeviceMemory object, and at least one subpass writes to that area of VkDeviceMemory, a subpass dependency **must** be included (either directly or via some intermediate subpasses) between them
- VUID-VkRenderPassCreateInfo2-attachment-03050
  If the attachment member of any element of pInputAttachments, pColorAttachments, pResolveAttachments or pDepthStencilAttachment, or the attachment indexed by any element of pPreserveAttachments in any element of pSubpasses is bound to a range of a VkDeviceMemory object that overlaps with any other attachment in any subpass (including the same subpass), the VkAttachmentDescription2 structures describing them **must** include VK_ATTACHMENT_DESCRIPTION_MAY_ALIAS_BIT in flags
- VUID-VkRenderPassCreateInfo2-attachment-03051
  If the attachment member of any element of pInputAttachments, pColorAttachments, pResolveAttachments or pDepthStencilAttachment, or any element of pPreserveAttachments in any element of pSubpasses is not VK_ATTACHMENT_UNUSED, then it **must** be less than attachmentCount
- VUID-VkRenderPassCreateInfo2-fragmentDensityMapAttachment-06472
  If the pNext chain includes a VkRenderPassFragmentDensityMapCreateInfoEXT structure and the fragmentDensityMapAttachment member is not VK_ATTACHMENT_UNUSED, then

attachment **must** be less than attachmentCount

- VUID-VkRenderPassCreateInfo2-pSubpasses-06473
  If the pSubpasses pNext chain includes a VkSubpassDescriptionDepthStencilResolve structure and the pDepthStencilResolveAttachment member is not NULL and does not have the value VK_ATTACHMENT_UNUSED, then attachment **must** be less than attachmentCount

- VUID-VkRenderPassCreateInfo2-pAttachments-02522
  For any member of pAttachments with a loadOp equal to VK_ATTACHMENT_LOAD_OP_CLEAR, the first use of that attachment **must** not specify a layout equal to VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL, VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL, or VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_STENCIL_ATTACHMENT_OPTIMAL

- VUID-VkRenderPassCreateInfo2-pAttachments-02523
  For any member of pAttachments with a stencilLoadOp equal to VK_ATTACHMENT_LOAD_OP_CLEAR, the first use of that attachment **must** not specify a layout equal to VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL, VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL, or VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_STENCIL_READ_ONLY_OPTIMAL

- VUID-VkRenderPassCreateInfo2-pDependencies-03054
  For any element of pDependencies, if the srcSubpass is not VK_SUBPASS_EXTERNAL, all stage flags included in the srcStageMask member of that dependency **must** be a pipeline stage supported by the pipeline identified by the pipelineBindPoint member of the source subpass

- VUID-VkRenderPassCreateInfo2-pDependencies-03055
  For any element of pDependencies, if the dstSubpass is not VK_SUBPASS_EXTERNAL, all stage flags included in the dstStageMask member of that dependency **must** be a pipeline stage supported by the pipeline identified by the pipelineBindPoint member of the destination subpass

- VUID-VkRenderPassCreateInfo2-pCorrelatedViewMasks-03056
  The set of bits included in any element of pCorrelatedViewMasks **must** not overlap with the set of bits included in any other element of pCorrelatedViewMasks

- VUID-VkRenderPassCreateInfo2-viewMask-03057
  If the VkSubpassDescription2::viewMask member of all elements of pSubpasses is 0, correlatedViewMaskCount **must** be 0

- VUID-VkRenderPassCreateInfo2-viewMask-03058
  The VkSubpassDescription2::viewMask member of all elements of pSubpasses **must** either all be 0, or all not be 0

- VUID-VkRenderPassCreateInfo2-viewMask-03059
  If the VkSubpassDescription2::viewMask member of all elements of pSubpasses is 0, the dependencyFlags member of any element of pDependencies **must** not include VK_DEPENDENCY_VIEW_LOCAL_BIT

- VUID-VkRenderPassCreateInfo2-pDependencies-03060
  For any element of pDependencies where its srcSubpass member equals its dstSubpass member, if the viewMask member of the corresponding element of pSubpasses includes more than one bit, its dependencyFlags member **must** include

VK_DEPENDENCY_VIEW_LOCAL_BIT

- VUID-VkRenderPassCreateInfo2-attachment-02525
  If the attachment member of any element of the pInputAttachments member of any element of pSubpasses is not VK_ATTACHMENT_UNUSED, the aspectMask member of that element of pInputAttachments **must** only include aspects that are present in images of the format specified by the element of pAttachments specified by attachment
- VUID-VkRenderPassCreateInfo2-srcSubpass-02526
  The srcSubpass member of each element of pDependencies **must** be less than subpassCount
- VUID-VkRenderPassCreateInfo2-dstSubpass-02527
  The dstSubpass member of each element of pDependencies **must** be less than subpassCount
- VUID-VkRenderPassCreateInfo2-pAttachments-04585
  If any element of pAttachments is used as a fragment shading rate attachment in any subpass, it **must** not be used as any other attachment in the render pass
- VUID-VkRenderPassCreateInfo2-pAttachments-09387
  If any element of pAttachments is used as a fragment shading rate attachment, the loadOp for that attachment **must** not be VK_ATTACHMENT_LOAD_OP_CLEAR
- VUID-VkRenderPassCreateInfo2-flags-04521
  If flags includes VK_RENDER_PASS_CREATE_TRANSFORM_BIT_QCOM, an element of pSubpasses includes an instance of VkFragmentShadingRateAttachmentInfoKHR in its pNext chain, and the pFragmentShadingRateAttachment member of that structure is not equal to NULL, the attachment member of pFragmentShadingRateAttachment **must** be VK_ATTACHMENT_UNUSED
- VUID-VkRenderPassCreateInfo2-pAttachments-04586
  If any element of pAttachments is used as a fragment shading rate attachment in any subpass, it **must** have an image format whose potential format features contain VK_FORMAT_FEATURE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR
- VUID-VkRenderPassCreateInfo2-rasterizationSamples-04905
  If the pipeline is being created with fragment shader state, and the VK_QCOM_render_pass_shader_resolve extension is enabled, and if subpass has any input attachments, and if the subpass description contains VK_SUBPASS_DESCRIPTION_FRAGMENT_REGION_BIT_QCOM, then the sample count of the input attachments **must** equal rasterizationSamples
- VUID-VkRenderPassCreateInfo2-sampleShadingEnable-04906
  If the pipeline is being created with fragment shader state, and the VK_QCOM_render_pass_shader_resolve extension is enabled, and if the subpass description contains VK_SUBPASS_DESCRIPTION_FRAGMENT_REGION_BIT_QCOM, then sampleShadingEnable **must** be false
- VUID-VkRenderPassCreateInfo2-flags-04907
  If flags includes VK_SUBPASS_DESCRIPTION_SHADER_RESOLVE_BIT_QCOM, and if pResolveAttachments is not NULL, then each resolve attachment **must** be VK_ATTACHMENT_UNUSED
- VUID-VkRenderPassCreateInfo2-flags-04908
  If flags includes VK_SUBPASS_DESCRIPTION_SHADER_RESOLVE_BIT_QCOM, and if pDepthStencilResolveAttachment is not NULL, then the depth/stencil resolve attachment

**must** be VK_ATTACHMENT_UNUSED

- VUID-VkRenderPassCreateInfo2-flags-04909
  If flags includes VK_SUBPASS_DESCRIPTION_SHADER_RESOLVE_BIT_QCOM, then the subpass **must** be the last subpass in a subpass dependency chain
- VUID-VkRenderPassCreateInfo2-attachment-06244
  If the attachment member of the pDepthStencilAttachment member of an element of pSubpasses is not VK_ATTACHMENT_UNUSED, the layout member of that same structure is either VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL or VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL, and the pNext chain of that structure does not include a VkAttachmentReferenceStencilLayout structure, then the element of pAttachments with an index equal to attachment **must** not have a format that includes both depth and stencil components
- VUID-VkRenderPassCreateInfo2-attachment-06245
  If the attachment member of the pDepthStencilAttachment member of an element of pSubpasses is not VK_ATTACHMENT_UNUSED and the layout member of that same structure is either VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL or VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL, then the element of pAttachments with an index equal to attachment **must** have a format that includes only a stencil component
- VUID-VkRenderPassCreateInfo2-attachment-06246
  If the attachment member of the pDepthStencilAttachment member of an element of pSubpasses is not VK_ATTACHMENT_UNUSED and the layout member of that same structure is either VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL or VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL, then the element of pAttachments with an index equal to attachment **must** not have a format that includes only a stencil component
- VUID-VkRenderPassCreateInfo2-pResolveAttachments-09331
  If any element of pResolveAttachments of any element of pSubpasses references an attachment description with a format of VK_FORMAT_UNDEFINED, VkRenderPassFragmentDensityMapCreateInfoEXT::fragmentDensityMapAttachment->attachment **must** be VK_ATTACHMENT_UNUSED

## Valid Usage (Implicit)

- VUID-VkRenderPassCreateInfo2-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_RENDER_PASS_CREATE_INFO_2
- VUID-VkRenderPassCreateInfo2-pNext-pNext
  Each pNext member of any structure (including this one) in the pNext chain **must** be either NULL or a pointer to a valid instance of VkRenderPassCreationControlEXT, VkRenderPassCreationFeedbackCreateInfoEXT, or VkRenderPassFragmentDensityMapCreateInfoEXT
- VUID-VkRenderPassCreateInfo2-sType-unique
  The sType value of each struct in the pNext chain **must** be unique
- VUID-VkRenderPassCreateInfo2-flags-parameter
  flags **must** be a valid combination of VkRenderPassCreateFlagBits values

- VUID-VkRenderPassCreateInfo2-pAttachments-parameter
  If `attachmentCount` is not `0`, `pAttachments` **must** be a valid pointer to an array of `attachmentCount` valid VkAttachmentDescription2 structures
- VUID-VkRenderPassCreateInfo2-pSubpasses-parameter
  `pSubpasses` **must** be a valid pointer to an array of `subpassCount` valid VkSubpassDescription2 structures
- VUID-VkRenderPassCreateInfo2-pDependencies-parameter
  If `dependencyCount` is not `0`, `pDependencies` **must** be a valid pointer to an array of `dependencyCount` valid VkSubpassDependency2 structures
- VUID-VkRenderPassCreateInfo2-pCorrelatedViewMasks-parameter
  If `correlatedViewMaskCount` is not `0`, `pCorrelatedViewMasks` **must** be a valid pointer to an array of `correlatedViewMaskCount` `uint32_t` values
- VUID-VkRenderPassCreateInfo2-subpassCount-arraylength
  `subpassCount` **must** be greater than `0`

The `VkAttachmentDescription2` structure is defined as:

```
typedef struct VkAttachmentDescription2 {
    VkStructureType                 sType;
    const void*                     pNext;
    VkAttachmentDescriptionFlags    flags;
    VkFormat                        format;
    VkSampleCountFlagBits           samples;
    VkAttachmentLoadOp              loadOp;
    VkAttachmentStoreOp             storeOp;
    VkAttachmentLoadOp              stencilLoadOp;
    VkAttachmentStoreOp             stencilStoreOp;
    VkImageLayout                   initialLayout;
    VkImageLayout                   finalLayout;
} VkAttachmentDescription2;
```

or the equivalent

```
// Provided by VK_KHR_create_renderpass2
typedef VkAttachmentDescription2 VkAttachmentDescription2KHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is a bitmask of VkAttachmentDescriptionFlagBits specifying additional properties of the attachment.
- `format` is a VkFormat value specifying the format of the image that will be used for the attachment.
- `samples` is a VkSampleCountFlagBits value specifying the number of samples of the image.

- loadOp is a VkAttachmentLoadOp value specifying how the contents of color and depth components of the attachment are treated at the beginning of the subpass where it is first used.
- storeOp is a VkAttachmentStoreOp value specifying how the contents of color and depth components of the attachment are treated at the end of the subpass where it is last used.
- stencilLoadOp is a VkAttachmentLoadOp value specifying how the contents of stencil components of the attachment are treated at the beginning of the subpass where it is first used.
- stencilStoreOp is a VkAttachmentStoreOp value specifying how the contents of stencil components of the attachment are treated at the end of the last subpass where it is used.
- initialLayout is the layout the attachment image subresource will be in when a render pass instance begins.
- finalLayout is the layout the attachment image subresource will be transitioned to when a render pass instance ends.

Parameters defined by this structure with the same name as those in VkAttachmentDescription have the identical effect to those parameters.

If the separateDepthStencilLayouts feature is enabled, and format is a depth/stencil format, initialLayout and finalLayout **can** be set to a layout that only specifies the layout of the depth aspect.

If the pNext chain includes a VkAttachmentDescriptionStencilLayout structure, then the stencilInitialLayout and stencilFinalLayout members specify the initial and final layouts of the stencil aspect of a depth/stencil format, and initialLayout and finalLayout only apply to the depth aspect. For depth-only formats, the VkAttachmentDescriptionStencilLayout structure is ignored. For stencil-only formats, the initial and final layouts of the stencil aspect are taken from the VkAttachmentDescriptionStencilLayout structure if present, or initialLayout and finalLayout if not present.

If format is a depth/stencil format, and either initialLayout or finalLayout does not specify a layout for the stencil aspect, then the application **must** specify the initial and final layouts of the stencil aspect by including a VkAttachmentDescriptionStencilLayout structure in the pNext chain.

loadOp and storeOp are ignored for fragment shading rate attachments. No access to the shading rate attachment is performed in loadOp and storeOp. Instead, access to VK_ACCESS_FRAGMENT_SHADING_RATE_ATTACHMENT_READ_BIT_KHR is performed as fragments are rasterized.

### Valid Usage

- VUID-VkAttachmentDescription2-format-06699
  If format includes a color or depth component and loadOp is VK_ATTACHMENT_LOAD_OP_LOAD, then initialLayout **must** not be VK_IMAGE_LAYOUT_UNDEFINED
- VUID-VkAttachmentDescription2-finalLayout-00843
  finalLayout **must** not be VK_IMAGE_LAYOUT_UNDEFINED or VK_IMAGE_LAYOUT_PREINITIALIZED
- VUID-VkAttachmentDescription2-format-03280
  If format is a color format, initialLayout **must** not be VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL or

VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL

- VUID-VkAttachmentDescription2-format-03281
  If format is a depth/stencil format, initialLayout **must** not be VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL
- VUID-VkAttachmentDescription2-format-03282
  If format is a color format, finalLayout **must** not be VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL or VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL
- VUID-VkAttachmentDescription2-format-03283
  If format is a depth/stencil format, finalLayout **must** not be VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL
- VUID-VkAttachmentDescription2-format-06487
  If format is a color format, initialLayout **must** not be VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_STENCIL_READ_ONLY_OPTIMAL or VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_STENCIL_ATTACHMENT_OPTIMAL
- VUID-VkAttachmentDescription2-format-06488
  If format is a color format, finalLayout **must** not be VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_STENCIL_READ_ONLY_OPTIMAL or VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_STENCIL_ATTACHMENT_OPTIMAL
- VUID-VkAttachmentDescription2-separateDepthStencilLayouts-03284
  If the separateDepthStencilLayouts feature is not enabled, initialLayout **must** not be VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL, VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL, VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL, or VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL,
- VUID-VkAttachmentDescription2-separateDepthStencilLayouts-03285
  If the separateDepthStencilLayouts feature is not enabled, finalLayout **must** not be VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL, VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL, VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL, or VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL,
- VUID-VkAttachmentDescription2-format-03286
  If format is a color format, initialLayout **must** not be VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL, VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL, VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL, or VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL
- VUID-VkAttachmentDescription2-format-03287
  If format is a color format, finalLayout **must** not be VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL, VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL, VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL, or VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL
- VUID-VkAttachmentDescription2-format-06906
  If format is a depth/stencil format which includes both depth and stencil components, initialLayout **must** not be VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL or VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL

- VUID-VkAttachmentDescription2-format-06907
  If `format` is a depth/stencil format which includes both depth and stencil components, `finalLayout` **must** not be `VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL` or `VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL`
- VUID-VkAttachmentDescription2-format-03290
  If `format` is a depth/stencil format which includes only the depth component, `initialLayout` **must** not be `VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL` or `VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL`
- VUID-VkAttachmentDescription2-format-03291
  If `format` is a depth/stencil format which includes only the depth component, `finalLayout` **must** not be `VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL` or `VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL`
- VUID-VkAttachmentDescription2-synchronization2-06908
  If the `synchronization2` feature is not enabled, `initialLayout` **must** not be `VK_IMAGE_LAYOUT_ATTACHMENT_OPTIMAL_KHR` or `VK_IMAGE_LAYOUT_READ_ONLY_OPTIMAL_KHR`
- VUID-VkAttachmentDescription2-synchronization2-06909
  If the `synchronization2` feature is not enabled, `finalLayout` **must** not be `VK_IMAGE_LAYOUT_ATTACHMENT_OPTIMAL_KHR` or `VK_IMAGE_LAYOUT_READ_ONLY_OPTIMAL_KHR`
- VUID-VkAttachmentDescription2-attachmentFeedbackLoopLayout-07309
  If the `attachmentFeedbackLoopLayout` feature is not enabled, `initialLayout` **must** not be `VK_IMAGE_LAYOUT_ATTACHMENT_FEEDBACK_LOOP_OPTIMAL_EXT`
- VUID-VkAttachmentDescription2-attachmentFeedbackLoopLayout-07310
  If the `attachmentFeedbackLoopLayout` feature is not enabled, `finalLayout` **must** not be `VK_IMAGE_LAYOUT_ATTACHMENT_FEEDBACK_LOOP_OPTIMAL_EXT`
- VUID-VkAttachmentDescription2-samples-08745
  `samples` **must** be a valid VkSampleCountFlagBits value that is set in `imageCreateSampleCounts` (as defined in Image Creation Limits) for the given `format`
- VUID-VkAttachmentDescription2-dynamicRenderingLocalRead-09544
  If the `dynamicRenderingLocalRead` feature is not enabled, `initialLayout` **must** not be `VK_IMAGE_LAYOUT_RENDERING_LOCAL_READ_KHR`
- VUID-VkAttachmentDescription2-dynamicRenderingLocalRead-09545
  If the `dynamicRenderingLocalRead` feature is not enabled, `finalLayout` **must** not be `VK_IMAGE_LAYOUT_RENDERING_LOCAL_READ_KHR`
- VUID-VkAttachmentDescription2-pNext-06704
  If the `pNext` chain does not include a VkAttachmentDescriptionStencilLayout structure, `format` includes a stencil component, and `stencilLoadOp` is `VK_ATTACHMENT_LOAD_OP_LOAD`, then `initialLayout` **must** not be `VK_IMAGE_LAYOUT_UNDEFINED`
- VUID-VkAttachmentDescription2-pNext-06705
  If the `pNext` chain includes a VkAttachmentDescriptionStencilLayout structure, `format` includes a stencil component, and `stencilLoadOp` is `VK_ATTACHMENT_LOAD_OP_LOAD`, then VkAttachmentDescriptionStencilLayout::`stencilInitialLayout` **must** not be `VK_IMAGE_LAYOUT_UNDEFINED`
- VUID-VkAttachmentDescription2-format-06249

If `format` is a depth/stencil format which includes both depth and stencil components, and `initialLayout` is `VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL` or `VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL`, the `pNext` chain **must** include a VkAttachmentDescriptionStencilLayout structure

- VUID-VkAttachmentDescription2-format-06250
  If `format` is a depth/stencil format which includes both depth and stencil components, and `finalLayout` is `VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL` or `VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL`, the `pNext` chain **must** include a VkAttachmentDescriptionStencilLayout structure
- VUID-VkAttachmentDescription2-format-06247
  If the `pNext` chain does not include a VkAttachmentDescriptionStencilLayout structure and `format` only includes a stencil component, `initialLayout` **must** not be `VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL` or `VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL`
- VUID-VkAttachmentDescription2-format-06248
  If the `pNext` chain does not include a VkAttachmentDescriptionStencilLayout structure and `format` only includes a stencil component, `finalLayout` **must** not be `VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL` or `VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL`
- VUID-VkAttachmentDescription2-format-09332
  If `externalFormatResolve` is not enabled, `format` **must** not be `VK_FORMAT_UNDEFINED`
- VUID-VkAttachmentDescription2-format-09334
  If `format` is `VK_FORMAT_UNDEFINED`, there **must** be a VkExternalFormatANDROID structure in the `pNext` chain with a `externalFormat` that is not equal to `0`

## Valid Usage (Implicit)

- VUID-VkAttachmentDescription2-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_ATTACHMENT_DESCRIPTION_2`
- VUID-VkAttachmentDescription2-pNext-pNext
  Each `pNext` member of any structure (including this one) in the `pNext` chain **must** be either `NULL` or a pointer to a valid instance of VkAttachmentDescriptionStencilLayout or VkExternalFormatANDROID
- VUID-VkAttachmentDescription2-sType-unique
  The `sType` value of each struct in the `pNext` chain **must** be unique
- VUID-VkAttachmentDescription2-flags-parameter
  `flags` **must** be a valid combination of VkAttachmentDescriptionFlagBits values
- VUID-VkAttachmentDescription2-format-parameter
  `format` **must** be a valid VkFormat value
- VUID-VkAttachmentDescription2-samples-parameter
  `samples` **must** be a valid VkSampleCountFlagBits value
- VUID-VkAttachmentDescription2-loadOp-parameter
  `loadOp` **must** be a valid VkAttachmentLoadOp value
- VUID-VkAttachmentDescription2-storeOp-parameter

storeOp **must** be a valid VkAttachmentStoreOp value

- VUID-VkAttachmentDescription2-stencilLoadOp-parameter
  stencilLoadOp **must** be a valid VkAttachmentLoadOp value
- VUID-VkAttachmentDescription2-stencilStoreOp-parameter
  stencilStoreOp **must** be a valid VkAttachmentStoreOp value
- VUID-VkAttachmentDescription2-initialLayout-parameter
  initialLayout **must** be a valid VkImageLayout value
- VUID-VkAttachmentDescription2-finalLayout-parameter
  finalLayout **must** be a valid VkImageLayout value

The VkAttachmentDescriptionStencilLayout structure is defined as:

```
typedef struct VkAttachmentDescriptionStencilLayout {
    VkStructureType    sType;
    void*              pNext;
    VkImageLayout      stencilInitialLayout;
    VkImageLayout      stencilFinalLayout;
} VkAttachmentDescriptionStencilLayout;
```

or the equivalent

```
// Provided by VK_KHR_separate_depth_stencil_layouts
typedef VkAttachmentDescriptionStencilLayout VkAttachmentDescriptionStencilLayoutKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- stencilInitialLayout is the layout the stencil aspect of the attachment image subresource will be in when a render pass instance begins.
- stencilFinalLayout is the layout the stencil aspect of the attachment image subresource will be transitioned to when a render pass instance ends.

### Valid Usage

- VUID-VkAttachmentDescriptionStencilLayout-stencilInitialLayout-03308
  stencilInitialLayout **must** not be VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL, VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL, VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL, VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL, VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL, VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_STENCIL_READ_ONLY_OPTIMAL, or VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_STENCIL_ATTACHMENT_OPTIMAL
- VUID-VkAttachmentDescriptionStencilLayout-stencilFinalLayout-03309
  stencilFinalLayout **must** not be VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL,

VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL, VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL, VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL, VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL, VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_STENCIL_READ_ONLY_OPTIMAL, or VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_STENCIL_ATTACHMENT_OPTIMAL

- VUID-VkAttachmentDescriptionStencilLayout-stencilFinalLayout-03310 stencilFinalLayout **must** not be VK_IMAGE_LAYOUT_UNDEFINED or VK_IMAGE_LAYOUT_PREINITIALIZED

## Valid Usage (Implicit)

- VUID-VkAttachmentDescriptionStencilLayout-sType-sType sType **must** be VK_STRUCTURE_TYPE_ATTACHMENT_DESCRIPTION_STENCIL_LAYOUT
- VUID-VkAttachmentDescriptionStencilLayout-stencilInitialLayout-parameter stencilInitialLayout **must** be a valid VkImageLayout value
- VUID-VkAttachmentDescriptionStencilLayout-stencilFinalLayout-parameter stencilFinalLayout **must** be a valid VkImageLayout value

The VkSubpassDescription2 structure is defined as:

```
typedef struct VkSubpassDescription2 {
    VkStructureType                 sType;
    const void*                     pNext;
    VkSubpassDescriptionFlags       flags;
    VkPipelineBindPoint             pipelineBindPoint;
    uint32_t                        viewMask;
    uint32_t                        inputAttachmentCount;
    const VkAttachmentReference2*   pInputAttachments;
    uint32_t                        colorAttachmentCount;
    const VkAttachmentReference2*   pColorAttachments;
    const VkAttachmentReference2*   pResolveAttachments;
    const VkAttachmentReference2*   pDepthStencilAttachment;
    uint32_t                        preserveAttachmentCount;
    const uint32_t*                 pPreserveAttachments;
} VkSubpassDescription2;
```

or the equivalent

```
// Provided by VK_KHR_create_renderpass2
typedef VkSubpassDescription2 VkSubpassDescription2KHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.

- flags is a bitmask of VkSubpassDescriptionFlagBits specifying usage of the subpass.
- pipelineBindPoint is a VkPipelineBindPoint value specifying the pipeline type supported for this subpass.
- viewMask is a bitfield of view indices describing which views rendering is broadcast to in this subpass, when multiview is enabled.
- inputAttachmentCount is the number of input attachments.
- pInputAttachments is a pointer to an array of VkAttachmentReference2 structures defining the input attachments for this subpass and their layouts.
- colorAttachmentCount is the number of color attachments.
- pColorAttachments is a pointer to an array of colorAttachmentCount VkAttachmentReference2 structures defining the color attachments for this subpass and their layouts.
- pResolveAttachments is NULL or a pointer to an array of colorAttachmentCount VkAttachmentReference2 structures defining the resolve attachments for this subpass and their layouts.
- pDepthStencilAttachment is a pointer to a VkAttachmentReference2 structure specifying the depth/stencil attachment for this subpass and its layout.
- preserveAttachmentCount is the number of preserved attachments.
- pPreserveAttachments is a pointer to an array of preserveAttachmentCount render pass attachment indices identifying attachments that are not used by this subpass, but whose contents **must** be preserved throughout the subpass.

Parameters defined by this structure with the same name as those in VkSubpassDescription have the identical effect to those parameters.

viewMask has the same effect for the described subpass as VkRenderPassMultiviewCreateInfo ::pViewMasks has on each corresponding subpass.

If a VkFragmentShadingRateAttachmentInfoKHR structure is included in the pNext chain, pFragmentShadingRateAttachment is not NULL, and its attachment member is not VK_ATTACHMENT_UNUSED, the identified attachment defines a fragment shading rate attachment for that subpass.

If any element of pResolveAttachments is an image specified with an VkExternalFormatANDROID, values in the corresponding color attachment will be resolved to the resolve attachment in the same manner as specified for VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID.

If the nullColorAttachmentWithExternalFormatResolve limit is VK_TRUE, values in the color attachment will be loaded from the resolve attachment at the start of rendering, and **may** also be reloaded any time after a resolve occurs or the resolve attachment is written to; if this occurs it **must** happen-before any writes to the color attachment are performed which happen-after the resolve that triggers this. If any color component in the external format is subsampled, values will be read from the nearest sample in the image when they are loaded. If the color attachment is also used as an input attachment, the same behavior applies.

Setting the color attachment to VK_ATTACHMENT_UNUSED when an external resolve attachment is used and the nullColorAttachmentWithExternalFormatResolve limit is VK_TRUE will not result in color

attachment writes to be discarded for that attachment.

When nullColorAttachmentWithExternalFormatResolve is VK_TRUE, the color output from the subpass can still be read via an input attachment; but the application cannot bind an image view for the color attachment as there is no such image view bound. Instead to access the data as an input attachment applications **can** use the resolve attachment in its place - using the resolve attachment image for the descriptor, and setting the corresponding element of pInputAttachments to the index of the resolve attachment.

Loads or input attachment reads from the resolve attachment are performed as if using a VkSamplerYcbcrConversionCreateInfo with the following parameters:

```
VkSamplerYcbcrConversionCreateInfo createInfo = {
    .sType = VK_STRUCTURE_TYPE_SAMPLER_YCBCR_CONVERSION_CREATE_INFO,
    .pNext = NULL,
    .format = VK_FORMAT_UNDEFINED,
    .ycbcrModel = VK_SAMPLER_YCBCR_MODEL_CONVERSION_RGB_IDENTITY,
    .ycbcrRange = VK_SAMPLER_YCBCR_RANGE_ITU_FULL,
    .components = {
        .r = VK_COMPONENT_SWIZZLE_B
        .g = VK_COMPONENT_SWIZZLE_R
        .b = VK_COMPONENT_SWIZZLE_G
        .a = VK_COMPONENT_SWIZZLE_IDENTITY},
    .xChromaOffset = properties.chromaOffsetX,
    .yChromaOffset = properties.chromaOffsetY,
    .chromaFilter = VK_FILTER_NEAREST,
    .forceExplicitReconstruction = ... };
```

where properties is equal to VkPhysicalDeviceExternalFormatResolvePropertiesANDROID returned by the device and forceExplicitReconstruction is effectively ignored as the VK_SAMPLER_YCBCR_MODEL_CONVERSION_RGB_IDENTITY model is used. The applied swizzle is the same effective swizzle that would be applied by the VK_SAMPLER_YCBCR_MODEL_CONVERSION_YCBCR_IDENTITY model, but no range expansion is applied.

### Valid Usage

- VUID-VkSubpassDescription2-attachment-06912
  If the attachment member of an element of pInputAttachments is not VK_ATTACHMENT_UNUSED, its layout member **must** not be VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL or VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL
- VUID-VkSubpassDescription2-attachment-06913
  If the attachment member of an element of pColorAttachments is not VK_ATTACHMENT_UNUSED, its layout member **must** not be VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL or VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL
- VUID-VkSubpassDescription2-attachment-06914
  If the attachment member of an element of pResolveAttachments is not VK_ATTACHMENT_UNUSED, its layout member **must** not be

VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL or VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL

- VUID-VkSubpassDescription2-attachment-06915
  If the attachment member of pDepthStencilAttachment is not VK_ATTACHMENT_UNUSED, ts layout member **must** not be VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL or VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL
- VUID-VkSubpassDescription2-attachment-06916
  If the attachment member of an element of pColorAttachments is not VK_ATTACHMENT_UNUSED, its layout member **must** not be VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_STENCIL_READ_ONLY_OPTIMAL or VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_STENCIL_ATTACHMENT_OPTIMAL
- VUID-VkSubpassDescription2-attachment-06917
  If the attachment member of an element of pResolveAttachments is not VK_ATTACHMENT_UNUSED, its layout member **must** not be VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_STENCIL_READ_ONLY_OPTIMAL or VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_STENCIL_ATTACHMENT_OPTIMAL
- VUID-VkSubpassDescription2-attachment-06918
  If the attachment member of an element of pInputAttachments is not VK_ATTACHMENT_UNUSED, its layout member **must** not be VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL or VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL
- VUID-VkSubpassDescription2-attachment-06919
  If the attachment member of an element of pColorAttachments is not VK_ATTACHMENT_UNUSED, its layout member **must** not be VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL, VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL, VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL, or VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL
- VUID-VkSubpassDescription2-attachment-06920
  If the attachment member of an element of pResolveAttachments is not VK_ATTACHMENT_UNUSED, its layout member **must** not be VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL, VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL, VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL, or VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL
- VUID-VkSubpassDescription2-attachment-06921
  If the attachment member of an element of pInputAttachments is not VK_ATTACHMENT_UNUSED, its layout member **must** not be VK_IMAGE_LAYOUT_ATTACHMENT_OPTIMAL_KHR
- VUID-VkSubpassDescription2-attachment-06922
  If the attachment member of an element of pColorAttachments is not VK_ATTACHMENT_UNUSED, its layout member **must** not be VK_IMAGE_LAYOUT_READ_ONLY_OPTIMAL_KHR
- VUID-VkSubpassDescription2-attachment-06923
  If the attachment member of an element of pResolveAttachments is not VK_ATTACHMENT_UNUSED, its layout member **must** not be VK_IMAGE_LAYOUT_READ_ONLY_OPTIMAL_KHR
- VUID-VkSubpassDescription2-attachment-06251
  If the attachment member of pDepthStencilAttachment is not VK_ATTACHMENT_UNUSED and its pNext chain includes a VkAttachmentReferenceStencilLayout structure, the layout

member of pDepthStencilAttachment **must** not be VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL or VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL

- VUID-VkSubpassDescription2-pipelineBindPoint-04953
  pipelineBindPoint **must** be VK_PIPELINE_BIND_POINT_GRAPHICS or VK_PIPELINE_BIND_POINT_SUBPASS_SHADING_HUAWEI
- VUID-VkSubpassDescription2-colorAttachmentCount-03063
  colorAttachmentCount **must** be less than or equal to VkPhysicalDeviceLimits::maxColorAttachments
- VUID-VkSubpassDescription2-loadOp-03064
  If the first use of an attachment in this render pass is as an input attachment, and the attachment is not also used as a color or depth/stencil attachment in the same subpass, then loadOp **must** not be VK_ATTACHMENT_LOAD_OP_CLEAR
- VUID-VkSubpassDescription2-pResolveAttachments-03067
  If pResolveAttachments is not NULL, each resolve attachment that is not VK_ATTACHMENT_UNUSED **must** have a sample count of VK_SAMPLE_COUNT_1_BIT
- VUID-VkSubpassDescription2-externalFormatResolve-09335
  If externalFormatResolve is not enabled and pResolveAttachments is not NULL, for each resolve attachment that does not have the value VK_ATTACHMENT_UNUSED, the corresponding color attachment **must** not have the value VK_ATTACHMENT_UNUSED
- VUID-VkSubpassDescription2-nullColorAttachmentWithExternalFormatResolve-09336
  If the nullColorAttachmentWithExternalFormatResolve property is VK_FALSE and pResolveAttachments is not NULL, for each resolve attachment that has a format of VK_FORMAT_UNDEFINED, the corresponding color attachment **must** not have the value VK_ATTACHMENT_UNUSED
- VUID-VkSubpassDescription2-nullColorAttachmentWithExternalFormatResolve-09337
  If the nullColorAttachmentWithExternalFormatResolve property is VK_TRUE and pResolveAttachments is not NULL, for each resolve attachment that has a format of VK_FORMAT_UNDEFINED, the corresponding color attachment **must** have the value VK_ATTACHMENT_UNUSED
- VUID-VkSubpassDescription2-externalFormatResolve-09338
  If externalFormatResolve is not enabled and pResolveAttachments is not NULL, for each resolve attachment that is not VK_ATTACHMENT_UNUSED, the corresponding color attachment **must** not have a sample count of VK_SAMPLE_COUNT_1_BIT
- VUID-VkSubpassDescription2-externalFormatResolve-09339
  If externalFormatResolve is not enabled, each element of pResolveAttachments **must** have the same VkFormat as its corresponding color attachment
- VUID-VkSubpassDescription2-multisampledRenderToSingleSampled-06869
  If the multisampledRenderToSingleSampled feature is not enabled, all attachments in pColorAttachments that are not VK_ATTACHMENT_UNUSED **must** have the same sample count
- VUID-VkSubpassDescription2-pInputAttachments-02897
  All attachments in pInputAttachments that are not VK_ATTACHMENT_UNUSED and any of the following is true:

- the externalFormatResolve feature is not enabled
- the nullColorAttachmentWithExternalFormatResolve property is VK_FALSE
- does not have a non-zero value of VkExternalFormatANDROID::externalFormat

**must** have image formats whose potential format features contain at least VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BIT or VK_FORMAT_FEATURE_DEPTH_STENCIL_ATTACHMENT_BIT

- VUID-VkSubpassDescription2-pColorAttachments-02898
  All attachments in pColorAttachments that are not VK_ATTACHMENT_UNUSED **must** have image formats whose potential format features contain VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BIT
- VUID-VkSubpassDescription2-pResolveAttachments-09343
  All attachments in pResolveAttachments that are not VK_ATTACHMENT_UNUSED and do not have an image format of VK_FORMAT_UNDEFINED **must** have image formats whose potential format features contain VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BIT
- VUID-VkSubpassDescription2-pDepthStencilAttachment-02900
  If pDepthStencilAttachment is not NULL and the attachment is not VK_ATTACHMENT_UNUSED then it **must** have an image format whose potential format features contain VK_FORMAT_FEATURE_DEPTH_STENCIL_ATTACHMENT_BIT
- VUID-VkSubpassDescription2-linearColorAttachment-06499
  If the linearColorAttachment feature is enabled and the image is created with VK_IMAGE_TILING_LINEAR, all attachments in pInputAttachments that are not VK_ATTACHMENT_UNUSED **must** have image formats whose potential format features **must** contain VK_FORMAT_FEATURE_2_LINEAR_COLOR_ATTACHMENT_BIT_NV
- VUID-VkSubpassDescription2-linearColorAttachment-06500
  If the linearColorAttachment feature is enabled and the image is created with VK_IMAGE_TILING_LINEAR, all attachments in pColorAttachments that are not VK_ATTACHMENT_UNUSED **must** have image formats whose potential format features **must** contain VK_FORMAT_FEATURE_2_LINEAR_COLOR_ATTACHMENT_BIT_NV
- VUID-VkSubpassDescription2-linearColorAttachment-06501
  If the linearColorAttachment feature is enabled and the image is created with VK_IMAGE_TILING_LINEAR, all attachments in pResolveAttachments that are not VK_ATTACHMENT_UNUSED **must** have image formats whose potential format features **must** contain VK_FORMAT_FEATURE_2_LINEAR_COLOR_ATTACHMENT_BIT_NV
- VUID-VkSubpassDescription2-None-09456
  If either of the following is enabled:

  - The VK_AMD_mixed_attachment_samples extension
  - The VK_NV_framebuffer_mixed_samples extension

  all attachments in pColorAttachments that are not VK_ATTACHMENT_UNUSED **must** have a sample count that is smaller than or equal to the sample count of pDepthStencilAttachment if it is not VK_ATTACHMENT_UNUSED
- VUID-VkSubpassDescription2-pNext-06870
  If the pNext chain includes a VkMultisampledRenderToSingleSampledInfoEXT structure

with multisampledRenderToSingleSampledEnable equal to VK_TRUE, then all attachments in pColorAttachments and pDepthStencilAttachment that are not VK_ATTACHMENT_UNUSED **must** have a sample count that is either VK_SAMPLE_COUNT_1_BIT or equal to VkMultisampledRenderToSingleSampledInfoEXT::rasterizationSamples

- VUID-VkSubpassDescription2-pNext-06871
  If the pNext chain includes a VkMultisampledRenderToSingleSampledInfoEXT structure with multisampledRenderToSingleSampledEnable equal to VK_TRUE, and pDepthStencilAttachment is not NULL, does not have the value VK_ATTACHMENT_UNUSED, and has a sample count of VK_SAMPLE_COUNT_1_BIT, the pNext chain **must** also include a VkSubpassDescriptionDepthStencilResolve structure with pDepthStencilResolveAttachment that is either NULL or has the value VK_ATTACHMENT_UNUSED
- VUID-VkSubpassDescription2-multisampledRenderToSingleSampled-06872
  All attachments in pDepthStencilAttachment or pColorAttachments that are not VK_ATTACHMENT_UNUSED **must** have the same sample count , if none of the following are enabled:
  - The VK_AMD_mixed_attachment_samples extension
  - The VK_NV_framebuffer_mixed_samples extension
  - The multisampledRenderToSingleSampled feature,
- VUID-VkSubpassDescription2-attachment-03073
  Each element of pPreserveAttachments **must** not be VK_ATTACHMENT_UNUSED
- VUID-VkSubpassDescription2-pPreserveAttachments-03074
  Each element of pPreserveAttachments **must** not also be an element of any other member of the subpass description
- VUID-VkSubpassDescription2-layout-02528
  If any attachment is used by more than one VkAttachmentReference2 member, then each use **must** use the same layout
- VUID-VkSubpassDescription2-flags-03076
  If flags includes VK_SUBPASS_DESCRIPTION_PER_VIEW_POSITION_X_ONLY_BIT_NVX, it **must** also include VK_SUBPASS_DESCRIPTION_PER_VIEW_ATTRIBUTES_BIT_NVX
- VUID-VkSubpassDescription2-attachment-02799
  If the attachment member of any element of pInputAttachments is not VK_ATTACHMENT_UNUSED, then the aspectMask member **must** be a valid combination of VkImageAspectFlagBits
- VUID-VkSubpassDescription2-attachment-02800
  If the attachment member of any element of pInputAttachments is not VK_ATTACHMENT_UNUSED, then the aspectMask member **must** not be 0
- VUID-VkSubpassDescription2-attachment-02801
  If the attachment member of any element of pInputAttachments is not VK_ATTACHMENT_UNUSED, then the aspectMask member **must** not include VK_IMAGE_ASPECT_METADATA_BIT
- VUID-VkSubpassDescription2-attachment-04563
  If the attachment member of any element of pInputAttachments is not VK_ATTACHMENT_UNUSED, then the aspectMask member **must** not include VK_IMAGE_ASPECT_MEMORY_PLANE_i_BIT_EXT for any index *i*

- VUID-VkSubpassDescription2-pDepthStencilAttachment-04440
  An attachment **must** not be used in both `pDepthStencilAttachment` and `pColorAttachments`
- VUID-VkSubpassDescription2-multiview-06558
  If the `multiview` feature is not enabled, `viewMask` **must** be `0`
- VUID-VkSubpassDescription2-viewMask-06706
  The index of the most significant bit in `viewMask` **must** be less than `maxMultiviewViewCount`
- VUID-VkSubpassDescription2-externalFormatResolve-09344
  If `externalFormatResolve` is enabled, `pResolveAttachments` is not `NULL`, and `colorAttachmentCount` is not `1`, any element of `pResolveAttachments` that is not `VK_ATTACHMENT_UNUSED`, **must** not have a format of `VK_FORMAT_UNDEFINED`
- VUID-VkSubpassDescription2-externalFormatResolve-09345
  If `externalFormatResolve` is enabled, `pResolveAttachments` is not `NULL`, any element of `pResolveAttachments` is not `VK_ATTACHMENT_UNUSED` and has a format of `VK_FORMAT_UNDEFINED`, and the corresponding element of `pColorAttachments` is not `VK_ATTACHMENT_UNUSED`, the color attachment **must** have a `samples` value of `1`
- VUID-VkSubpassDescription2-externalFormatResolve-09346
  If `externalFormatResolve` is enabled, `pResolveAttachments` is not `NULL`, and any element of `pResolveAttachments` is not `VK_ATTACHMENT_UNUSED` and has a format of `VK_FORMAT_UNDEFINED`, `viewMask` **must** be `0`
- VUID-VkSubpassDescription2-externalFormatResolve-09347
  If `externalFormatResolve` is enabled, `pResolveAttachments` is not `NULL`, and any element of `pResolveAttachments` is not `VK_ATTACHMENT_UNUSED` and has a format of `VK_FORMAT_UNDEFINED`, VkFragmentShadingRateAttachmentInfoKHR::`pFragmentShadingRateAttachment` **must** either be `NULL` or a VkAttachmentReference2 structure with an `attachment` value of `VK_ATTACHMENT_UNUSED`
- VUID-VkSubpassDescription2-externalFormatResolve-09348
  If `externalFormatResolve` is enabled, `pResolveAttachments` is not `NULL`, and any element of `pResolveAttachments` is not `VK_ATTACHMENT_UNUSED` and has a format of `VK_FORMAT_UNDEFINED`, elements of `pInputAttachments` referencing either a color attachment or resolve attachment used in this subpass **must** not include `VK_IMAGE_ASPECT_PLANE_i_BIT` for any index *i* in its `aspectMask`

## Valid Usage (Implicit)

- VUID-VkSubpassDescription2-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_SUBPASS_DESCRIPTION_2`
- VUID-VkSubpassDescription2-pNext-pNext
  Each `pNext` member of any structure (including this one) in the `pNext` chain **must** be either `NULL` or a pointer to a valid instance of VkFragmentShadingRateAttachmentInfoKHR, VkMultisampledRenderToSingleSampledInfoEXT, VkRenderPassCreationControlEXT, VkRenderPassSubpassFeedbackCreateInfoEXT, or VkSubpassDescriptionDepthStencilResolve
- VUID-VkSubpassDescription2-sType-unique

The `sType` value of each struct in the `pNext` chain **must** be unique

- VUID-VkSubpassDescription2-flags-parameter
  `flags` **must** be a valid combination of VkSubpassDescriptionFlagBits values
- VUID-VkSubpassDescription2-pipelineBindPoint-parameter
  `pipelineBindPoint` **must** be a valid VkPipelineBindPoint value
- VUID-VkSubpassDescription2-pInputAttachments-parameter
  If `inputAttachmentCount` is not `0`, `pInputAttachments` **must** be a valid pointer to an array of `inputAttachmentCount` valid VkAttachmentReference2 structures
- VUID-VkSubpassDescription2-pColorAttachments-parameter
  If `colorAttachmentCount` is not `0`, `pColorAttachments` **must** be a valid pointer to an array of `colorAttachmentCount` valid VkAttachmentReference2 structures
- VUID-VkSubpassDescription2-pResolveAttachments-parameter
  If `colorAttachmentCount` is not `0`, and `pResolveAttachments` is not `NULL`, `pResolveAttachments` **must** be a valid pointer to an array of `colorAttachmentCount` valid VkAttachmentReference2 structures
- VUID-VkSubpassDescription2-pDepthStencilAttachment-parameter
  If `pDepthStencilAttachment` is not `NULL`, `pDepthStencilAttachment` **must** be a valid pointer to a valid VkAttachmentReference2 structure
- VUID-VkSubpassDescription2-pPreserveAttachments-parameter
  If `preserveAttachmentCount` is not `0`, `pPreserveAttachments` **must** be a valid pointer to an array of `preserveAttachmentCount` `uint32_t` values

The `VkSubpassDescriptionDepthStencilResolve` structure is defined as:

```
typedef struct VkSubpassDescriptionDepthStencilResolve {
    VkStructureType                 sType;
    const void*                     pNext;
    VkResolveModeFlagBits           depthResolveMode;
    VkResolveModeFlagBits           stencilResolveMode;
    const VkAttachmentReference2*   pDepthStencilResolveAttachment;
} VkSubpassDescriptionDepthStencilResolve;
```

or the equivalent

```
// Provided by VK_KHR_depth_stencil_resolve
typedef VkSubpassDescriptionDepthStencilResolve
VkSubpassDescriptionDepthStencilResolveKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `depthResolveMode` is a VkResolveModeFlagBits value describing the depth resolve mode.
- `stencilResolveMode` is a VkResolveModeFlagBits value describing the stencil resolve mode.

- pDepthStencilResolveAttachment is NULL or a pointer to a VkAttachmentReference2 structure defining the depth/stencil resolve attachment for this subpass and its layout.

If the pNext chain of VkSubpassDescription2 includes a VkSubpassDescriptionDepthStencilResolve structure, then that structure describes multisample resolve operations for the depth/stencil attachment in a subpass. If this structure is not included in the pNext chain of VkSubpassDescription2, or if it is and either pDepthStencilResolveAttachment is NULL or its attachment index is VK_ATTACHMENT_UNUSED, it indicates that no depth/stencil resolve attachment will be used in the subpass.

### Valid Usage

- VUID-VkSubpassDescriptionDepthStencilResolve-pDepthStencilResolveAttachment-03177
  If pDepthStencilResolveAttachment is not NULL and does not have the value VK_ATTACHMENT_UNUSED, pDepthStencilAttachment **must** not be NULL or have the value VK_ATTACHMENT_UNUSED
- VUID-VkSubpassDescriptionDepthStencilResolve-pDepthStencilResolveAttachment-03179
  If pDepthStencilResolveAttachment is not NULL and does not have the value VK_ATTACHMENT_UNUSED, pDepthStencilAttachment **must** not have a sample count of VK_SAMPLE_COUNT_1_BIT
- VUID-VkSubpassDescriptionDepthStencilResolve-pDepthStencilResolveAttachment-03180
  If pDepthStencilResolveAttachment is not NULL and does not have the value VK_ATTACHMENT_UNUSED, pDepthStencilResolveAttachment **must** have a sample count of VK_SAMPLE_COUNT_1_BIT
- VUID-VkSubpassDescriptionDepthStencilResolve-pDepthStencilResolveAttachment-02651
  If pDepthStencilResolveAttachment is not NULL and does not have the value VK_ATTACHMENT_UNUSED then it **must** have an image format whose potential format features contain VK_FORMAT_FEATURE_DEPTH_STENCIL_ATTACHMENT_BIT
- VUID-VkSubpassDescriptionDepthStencilResolve-pDepthStencilResolveAttachment-03181
  If pDepthStencilResolveAttachment is not NULL and does not have the value VK_ATTACHMENT_UNUSED and VkFormat of pDepthStencilResolveAttachment has a depth component, then the VkFormat of pDepthStencilAttachment **must** have a depth component with the same number of bits and numeric format
- VUID-VkSubpassDescriptionDepthStencilResolve-pDepthStencilResolveAttachment-03182
  If pDepthStencilResolveAttachment is not NULL and does not have the value VK_ATTACHMENT_UNUSED, and VkFormat of pDepthStencilResolveAttachment has a stencil component, then the VkFormat of pDepthStencilAttachment **must** have a stencil component with the same number of bits and numeric format
- VUID-VkSubpassDescriptionDepthStencilResolve-pDepthStencilResolveAttachment-03178
  If pDepthStencilResolveAttachment is not NULL and does not have the value VK_ATTACHMENT_UNUSED, depthResolveMode and stencilResolveMode **must** not both be VK_RESOLVE_MODE_NONE
- VUID-VkSubpassDescriptionDepthStencilResolve-depthResolveMode-03183
  If pDepthStencilResolveAttachment is not NULL and does not have the value VK_ATTACHMENT_UNUSED and the VkFormat of pDepthStencilResolveAttachment has a depth

component, then the value of depthResolveMode **must** be one of the bits set in VkPhysicalDeviceDepthStencilResolveProperties::supportedDepthResolveModes or VK_RESOLVE_MODE_NONE

- VUID-VkSubpassDescriptionDepthStencilResolve-stencilResolveMode-03184
  If pDepthStencilResolveAttachment is not NULL and does not have the value VK_ATTACHMENT_UNUSED and the VkFormat of pDepthStencilResolveAttachment has a stencil component, then the value of stencilResolveMode **must** be one of the bits set in VkPhysicalDeviceDepthStencilResolveProperties::supportedStencilResolveModes or VK_RESOLVE_MODE_NONE

- VUID-VkSubpassDescriptionDepthStencilResolve-pDepthStencilResolveAttachment-03185
  If pDepthStencilResolveAttachment is not NULL and does not have the value VK_ATTACHMENT_UNUSED, the VkFormat of pDepthStencilResolveAttachment has both depth and stencil components, VkPhysicalDeviceDepthStencilResolveProperties::independentResolve is VK_FALSE, and VkPhysicalDeviceDepthStencilResolveProperties::independentResolveNone is VK_FALSE, then the values of depthResolveMode and stencilResolveMode **must** be identical

- VUID-VkSubpassDescriptionDepthStencilResolve-pDepthStencilResolveAttachment-03186
  If pDepthStencilResolveAttachment is not NULL and does not have the value VK_ATTACHMENT_UNUSED, the VkFormat of pDepthStencilResolveAttachment has both depth and stencil components, VkPhysicalDeviceDepthStencilResolveProperties::independentResolve is VK_FALSE and VkPhysicalDeviceDepthStencilResolveProperties::independentResolveNone is VK_TRUE, then the values of depthResolveMode and stencilResolveMode **must** be identical or one of them **must** be VK_RESOLVE_MODE_NONE

- VUID-VkSubpassDescriptionDepthStencilResolve-pNext-06873
  If the pNext chain of VkSubpassDescription2 includes a VkMultisampledRenderToSingleSampledInfoEXT structure, the multisampledRenderToSingleSampledEnable field is VK_TRUE, and pDepthStencilAttachment is not NULL and does not have the value VK_ATTACHMENT_UNUSED, depthResolveMode and stencilResolveMode **must** not both be VK_RESOLVE_MODE_NONE

- VUID-VkSubpassDescriptionDepthStencilResolve-pNext-06874
  If the pNext chain of VkSubpassDescription2 includes a VkMultisampledRenderToSingleSampledInfoEXT structure whose multisampledRenderToSingleSampledEnable field is VK_TRUE, and pDepthStencilAttachment is not NULL, does not have the value VK_ATTACHMENT_UNUSED, and has a VkFormat that has a depth component, then the value of depthResolveMode **must** be one of the bits set in VkPhysicalDeviceDepthStencilResolveProperties::supportedDepthResolveModes or VK_RESOLVE_MODE_NONE

- VUID-VkSubpassDescriptionDepthStencilResolve-pNext-06875
  If the pNext chain of VkSubpassDescription2 includes a VkMultisampledRenderToSingleSampledInfoEXT structure whose multisampledRenderToSingleSampledEnable field is VK_TRUE, and pDepthStencilAttachment is not NULL, does not have the value VK_ATTACHMENT_UNUSED, and has a VkFormat with a stencil component, then the value of stencilResolveMode **must** be one of the bits set in VkPhysicalDeviceDepthStencilResolveProperties::supportedStencilResolveModes or VK_RESOLVE_MODE_NONE

- VUID-VkSubpassDescriptionDepthStencilResolve-pNext-06876

If the pNext chain of VkSubpassDescription2 includes a VkMultisampledRenderToSingleSampledInfoEXT structure whose multisampledRenderToSingleSampledEnable field is VK_TRUE, pDepthStencilAttachment is not NULL, does not have the value VK_ATTACHMENT_UNUSED, and has a VkFormat with both depth and stencil components, and both VkPhysicalDeviceDepthStencilResolveProperties::independentResolve and VkPhysicalDeviceDepthStencilResolveProperties::independentResolveNone are VK_FALSE, then the values of depthResolveMode and stencilResolveMode **must** be identical

- VUID-VkSubpassDescriptionDepthStencilResolve-pNext-06877
  If the pNext chain of VkSubpassDescription2 includes a VkMultisampledRenderToSingleSampledInfoEXT structure whose multisampledRenderToSingleSampledEnable field is VK_TRUE, pDepthStencilAttachment is not NULL, does not have the value VK_ATTACHMENT_UNUSED, and has a VkFormat with both depth and stencil components, VkPhysicalDeviceDepthStencilResolveProperties::independentResolve is VK_FALSE, and VkPhysicalDeviceDepthStencilResolveProperties::independentResolveNone is VK_TRUE, then the values of depthResolveMode and stencilResolveMode **must** be identical or one of them **must** be VK_RESOLVE_MODE_NONE

### Valid Usage (Implicit)

- VUID-VkSubpassDescriptionDepthStencilResolve-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_SUBPASS_DESCRIPTION_DEPTH_STENCIL_RESOLVE
- VUID-VkSubpassDescriptionDepthStencilResolve-pDepthStencilResolveAttachment-parameter
  If pDepthStencilResolveAttachment is not NULL, pDepthStencilResolveAttachment **must** be a valid pointer to a valid VkAttachmentReference2 structure

The VkFragmentShadingRateAttachmentInfoKHR structure is defined as:

```
// Provided by VK_KHR_fragment_shading_rate
typedef struct VkFragmentShadingRateAttachmentInfoKHR {
    VkStructureType                 sType;
    const void*                     pNext;
    const VkAttachmentReference2*   pFragmentShadingRateAttachment;
    VkExtent2D                      shadingRateAttachmentTexelSize;
} VkFragmentShadingRateAttachmentInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- pFragmentShadingRateAttachment is NULL or a pointer to a VkAttachmentReference2 structure defining the fragment shading rate attachment for this subpass.
- shadingRateAttachmentTexelSize specifies the size of the portion of the framebuffer corresponding to each texel in pFragmentShadingRateAttachment.

If no shading rate attachment is specified, or if this structure is not specified, the implementation behaves as if a valid shading rate attachment was specified with all texels specifying a single pixel per fragment.

### Valid Usage

- VUID-VkFragmentShadingRateAttachmentInfoKHR-pFragmentShadingRateAttachment-04524
  If `pFragmentShadingRateAttachment` is not `NULL` and its `attachment` member is not `VK_ATTACHMENT_UNUSED`, its `layout` member **must** be equal to `VK_IMAGE_LAYOUT_GENERAL` or `VK_IMAGE_LAYOUT_FRAGMENT_SHADING_RATE_ATTACHMENT_OPTIMAL_KHR`
- VUID-VkFragmentShadingRateAttachmentInfoKHR-pFragmentShadingRateAttachment-04525
  If `pFragmentShadingRateAttachment` is not `NULL` and its `attachment` member is not `VK_ATTACHMENT_UNUSED`, `shadingRateAttachmentTexelSize.width` **must** be a power of two value
- VUID-VkFragmentShadingRateAttachmentInfoKHR-pFragmentShadingRateAttachment-04526
  If `pFragmentShadingRateAttachment` is not `NULL` and its `attachment` member is not `VK_ATTACHMENT_UNUSED`, `shadingRateAttachmentTexelSize.width` **must** be less than or equal to `maxFragmentShadingRateAttachmentTexelSize.width`
- VUID-VkFragmentShadingRateAttachmentInfoKHR-pFragmentShadingRateAttachment-04527
  If `pFragmentShadingRateAttachment` is not `NULL` and its `attachment` member is not `VK_ATTACHMENT_UNUSED`, `shadingRateAttachmentTexelSize.width` **must** be greater than or equal to `minFragmentShadingRateAttachmentTexelSize.width`
- VUID-VkFragmentShadingRateAttachmentInfoKHR-pFragmentShadingRateAttachment-04528
  If `pFragmentShadingRateAttachment` is not `NULL` and its `attachment` member is not `VK_ATTACHMENT_UNUSED`, `shadingRateAttachmentTexelSize.height` **must** be a power of two value
- VUID-VkFragmentShadingRateAttachmentInfoKHR-pFragmentShadingRateAttachment-04529
  If `pFragmentShadingRateAttachment` is not `NULL` and its `attachment` member is not `VK_ATTACHMENT_UNUSED`, `shadingRateAttachmentTexelSize.height` **must** be less than or equal to `maxFragmentShadingRateAttachmentTexelSize.height`
- VUID-VkFragmentShadingRateAttachmentInfoKHR-pFragmentShadingRateAttachment-04530
  If `pFragmentShadingRateAttachment` is not `NULL` and its `attachment` member is not `VK_ATTACHMENT_UNUSED`, `shadingRateAttachmentTexelSize.height` **must** be greater than or equal to `minFragmentShadingRateAttachmentTexelSize.height`
- VUID-VkFragmentShadingRateAttachmentInfoKHR-pFragmentShadingRateAttachment-04531
  If `pFragmentShadingRateAttachment` is not `NULL` and its `attachment` member is not `VK_ATTACHMENT_UNUSED`, the quotient of `shadingRateAttachmentTexelSize.width` and

shadingRateAttachmentTexelSize.height **must** be less than or equal to maxFragmentShadingRateAttachmentTexelSizeAspectRatio

- VUID-VkFragmentShadingRateAttachmentInfoKHR-pFragmentShadingRateAttachment-04532
  If pFragmentShadingRateAttachment is not NULL and its attachment member is not VK_ATTACHMENT_UNUSED, the quotient of shadingRateAttachmentTexelSize.height and shadingRateAttachmentTexelSize.width **must** be less than or equal to maxFragmentShadingRateAttachmentTexelSizeAspectRatio

### Valid Usage (Implicit)

- VUID-VkFragmentShadingRateAttachmentInfoKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_FRAGMENT_SHADING_RATE_ATTACHMENT_INFO_KHR
- VUID-VkFragmentShadingRateAttachmentInfoKHR-pFragmentShadingRateAttachment-parameter
  If pFragmentShadingRateAttachment is not NULL, pFragmentShadingRateAttachment **must** be a valid pointer to a valid VkAttachmentReference2 structure

If the pNext chain of VkSubpassDescription2 or VkRenderingInfo includes a VkMultisampledRenderToSingleSampledInfoEXT structure, then that structure describes how multisampled rendering is performed on single sampled attachments in that subpass.

The VkMultisampledRenderToSingleSampledInfoEXT structure is defined as:

```
// Provided by VK_EXT_multisampled_render_to_single_sampled
typedef struct VkMultisampledRenderToSingleSampledInfoEXT {
    VkStructureType          sType;
    const void*              pNext;
    VkBool32                 multisampledRenderToSingleSampledEnable;
    VkSampleCountFlagBits    rasterizationSamples;
} VkMultisampledRenderToSingleSampledInfoEXT;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- multisampledRenderToSingleSampledEnable controls whether multisampled rendering to single-sampled attachments is performed as described below.
- rasterizationSamples is a VkSampleCountFlagBits specifying the number of samples used in rasterization.

### Valid Usage

- VUID-VkMultisampledRenderToSingleSampledInfoEXT-rasterizationSamples-06878
  The value of rasterizationSamples **must** not be VK_SAMPLE_COUNT_1_BIT

- VUID-VkMultisampledRenderToSingleSampledInfoEXT-pNext-06880
  If added to the pNext chain of VkRenderingInfo, each imageView member of any element of VkRenderingInfo::pColorAttachments, VkRenderingInfo::pDepthAttachment, or VkRenderingInfo::pStencilAttachment that is not VK_NULL_HANDLE **must** have a format that supports the sample count specified in rasterizationSamples

### Valid Usage (Implicit)

- VUID-VkMultisampledRenderToSingleSampledInfoEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_MULTISAMPLED_RENDER_TO_SINGLE_SAMPLED_INFO_EXT
- VUID-VkMultisampledRenderToSingleSampledInfoEXT-rasterizationSamples-parameter
  rasterizationSamples **must** be a valid VkSampleCountFlagBits value

If the pNext chain of VkSubpassDescription2 or VkRenderingInfo includes a VkMultisampledRenderToSingleSampledInfoEXT structure whose multisampledRenderToSingleSampledEnable field is VK_TRUE, the graphics pipelines **must** have VkGraphicsPipelineCreateInfo::rasterizationSamples equal to VkMultisampledRenderToSingleSampledInfoEXT::rasterizationSamples, and the subpass attachments **can** have a sample count of VK_SAMPLE_COUNT_1_BIT. For attachments with a sample count of VK_SAMPLE_COUNT_1_BIT, multisampled rendering is performed to an intermediate multisampled image with VkMultisampledRenderToSingleSampledInfoEXT::rasterizationSamples samples, implicitly allocated by the implementation for the duration of the subpass. For such attachments:

- If loadOp equals to VK_ATTACHMENT_LOAD_OP_LOAD, samples of the implicit image are initialized by replicating the value from the corresponding pixel in the attachment.
- If storeOp or stencilStoreOp is equal to VK_ATTACHMENT_STORE_OP_STORE, the implicit image is implicitly resolved prior to storage in the attachment.

Memory constraints due to high primitive counts **may** result in an implicit split of the subpass. This is the equivalent of partial rasterization of geometry in a render pass that ends in storeOp and stencilStoreOp equal to VK_ATTACHMENT_STORE_OP_STORE, followed by another render pass with loadOp and stencilLoadOp equal to VK_ATTACHMENT_LOAD_OP_LOAD with appropriate barriers in between. When VkMultisampledRenderToSingleSampledInfoEXT is used, the implementation is allowed to resolve attachments with a sample count of VK_SAMPLE_COUNT_1_BIT and lose multisampled data on such splits. The implementation **may** similarly split the render pass at subpass boundaries even if they use the same value for VkMultisampledRenderToSingleSampledInfoEXT::rasterizationSamples.

The VkAttachmentReference2 structure is defined as:

```
typedef struct VkAttachmentReference2 {
    VkStructureType       sType;
    const void*           pNext;
    uint32_t              attachment;
    VkImageLayout         layout;
    VkImageAspectFlags    aspectMask;
} VkAttachmentReference2;
```

or the equivalent

```
// Provided by VK_KHR_create_renderpass2
typedef VkAttachmentReference2 VkAttachmentReference2KHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `attachment` is either an integer value identifying an attachment at the corresponding index in VkRenderPassCreateInfo2::`pAttachments`, or `VK_ATTACHMENT_UNUSED` to signify that this attachment is not used.
- `layout` is a VkImageLayout value specifying the layout the attachment uses during the subpass.
- `aspectMask` is a mask of which aspect(s) **can** be accessed within the specified subpass as an input attachment.

Parameters defined by this structure with the same name as those in VkAttachmentReference have the identical effect to those parameters.

`aspectMask` is ignored when this structure is used to describe anything other than an input attachment reference.

If the `separateDepthStencilLayouts` feature is enabled, and `attachment` has a depth/stencil format, `layout` **can** be set to a layout that only specifies the layout of the depth aspect.

If `layout` only specifies the layout of the depth aspect of the attachment, the layout of the stencil aspect is specified by the `stencilLayout` member of a VkAttachmentReferenceStencilLayout structure included in the `pNext` chain. Otherwise, `layout` describes the layout for all relevant image aspects.

### Valid Usage

- VUID-VkAttachmentReference2-layout-03077
  If `attachment` is not `VK_ATTACHMENT_UNUSED`, `layout` **must** not be `VK_IMAGE_LAYOUT_UNDEFINED`, `VK_IMAGE_LAYOUT_PREINITIALIZED`, or `VK_IMAGE_LAYOUT_PRESENT_SRC_KHR`
- VUID-VkAttachmentReference2-separateDepthStencilLayouts-03313
  If the `separateDepthStencilLayouts` feature is not enabled, and `attachment` is not `VK_ATTACHMENT_UNUSED`, `layout` **must** not be `VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL`, `VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL`, `VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL`, or

VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL,

- VUID-VkAttachmentReference2-synchronization2-06910
  If the synchronization2 feature is not enabled, layout **must** not be VK_IMAGE_LAYOUT_ATTACHMENT_OPTIMAL_KHR or VK_IMAGE_LAYOUT_READ_ONLY_OPTIMAL_KHR
- VUID-VkAttachmentReference2-attachmentFeedbackLoopLayout-07311
  If the attachmentFeedbackLoopLayout feature is not enabled, layout **must** not be VK_IMAGE_LAYOUT_ATTACHMENT_FEEDBACK_LOOP_OPTIMAL_EXT
- VUID-VkAttachmentReference2-dynamicRenderingLocalRead-09546
  If the dynamicRenderingLocalRead feature is not enabled, layout **must** not be VK_IMAGE_LAYOUT_RENDERING_LOCAL_READ_KHR

### Valid Usage (Implicit)

- VUID-VkAttachmentReference2-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_ATTACHMENT_REFERENCE_2
- VUID-VkAttachmentReference2-pNext-pNext
  pNext **must** be NULL or a pointer to a valid instance of VkAttachmentReferenceStencilLayout
- VUID-VkAttachmentReference2-sType-unique
  The sType value of each struct in the pNext chain **must** be unique
- VUID-VkAttachmentReference2-layout-parameter
  layout **must** be a valid VkImageLayout value

The VkAttachmentReferenceStencilLayout structure is defined as:

```
typedef struct VkAttachmentReferenceStencilLayout {
    VkStructureType    sType;
    void*              pNext;
    VkImageLayout      stencilLayout;
} VkAttachmentReferenceStencilLayout;
```

or the equivalent

```
// Provided by VK_KHR_separate_depth_stencil_layouts
typedef VkAttachmentReferenceStencilLayout VkAttachmentReferenceStencilLayoutKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- stencilLayout is a VkImageLayout value specifying the layout the stencil aspect of the attachment uses during the subpass.

### Valid Usage

- VUID-VkAttachmentReferenceStencilLayout-stencilLayout-03318
  `stencilLayout` **must** not be `VK_IMAGE_LAYOUT_UNDEFINED`, `VK_IMAGE_LAYOUT_PREINITIALIZED`, `VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL`, `VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL`, `VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL`, `VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL`, `VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL`, `VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_STENCIL_READ_ONLY_OPTIMAL`, `VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_STENCIL_ATTACHMENT_OPTIMAL`, or `VK_IMAGE_LAYOUT_PRESENT_SRC_KHR`

### Valid Usage (Implicit)

- VUID-VkAttachmentReferenceStencilLayout-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_ATTACHMENT_REFERENCE_STENCIL_LAYOUT`
- VUID-VkAttachmentReferenceStencilLayout-stencilLayout-parameter
  `stencilLayout` **must** be a valid VkImageLayout value

The `VkSubpassDependency2` structure is defined as:

```
typedef struct VkSubpassDependency2 {
    VkStructureType         sType;
    const void*             pNext;
    uint32_t                srcSubpass;
    uint32_t                dstSubpass;
    VkPipelineStageFlags    srcStageMask;
    VkPipelineStageFlags    dstStageMask;
    VkAccessFlags           srcAccessMask;
    VkAccessFlags           dstAccessMask;
    VkDependencyFlags       dependencyFlags;
    int32_t                 viewOffset;
} VkSubpassDependency2;
```

or the equivalent

```
// Provided by VK_KHR_create_renderpass2
typedef VkSubpassDependency2 VkSubpassDependency2KHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `srcSubpass` is the subpass index of the first subpass in the dependency, or `VK_SUBPASS_EXTERNAL`.
- `dstSubpass` is the subpass index of the second subpass in the dependency, or

VK_SUBPASS_EXTERNAL.

- srcStageMask is a bitmask of VkPipelineStageFlagBits specifying the source stage mask.
- dstStageMask is a bitmask of VkPipelineStageFlagBits specifying the destination stage mask
- srcAccessMask is a bitmask of VkAccessFlagBits specifying a source access mask.
- dstAccessMask is a bitmask of VkAccessFlagBits specifying a destination access mask.
- dependencyFlags is a bitmask of VkDependencyFlagBits.
- viewOffset controls which views in the source subpass the views in the destination subpass depend on.

Parameters defined by this structure with the same name as those in VkSubpassDependency have the identical effect to those parameters.

viewOffset has the same effect for the described subpass dependency as VkRenderPassMultiviewCreateInfo::pViewOffsets has on each corresponding subpass dependency.

If a VkMemoryBarrier2 is included in the pNext chain, srcStageMask, dstStageMask, srcAccessMask, and dstAccessMask parameters are ignored. The synchronization and access scopes instead are defined by the parameters of VkMemoryBarrier2.

### Valid Usage

- VUID-VkSubpassDependency2-srcStageMask-04090
  If the geometryShader feature is not enabled, srcStageMask **must** not contain VK_PIPELINE_STAGE_GEOMETRY_SHADER_BIT
- VUID-VkSubpassDependency2-srcStageMask-04091
  If the tessellationShader feature is not enabled, srcStageMask **must** not contain VK_PIPELINE_STAGE_TESSELLATION_CONTROL_SHADER_BIT or VK_PIPELINE_STAGE_TESSELLATION_EVALUATION_SHADER_BIT
- VUID-VkSubpassDependency2-srcStageMask-04092
  If the conditionalRendering feature is not enabled, srcStageMask **must** not contain VK_PIPELINE_STAGE_CONDITIONAL_RENDERING_BIT_EXT
- VUID-VkSubpassDependency2-srcStageMask-04093
  If the fragmentDensityMap feature is not enabled, srcStageMask **must** not contain VK_PIPELINE_STAGE_FRAGMENT_DENSITY_PROCESS_BIT_EXT
- VUID-VkSubpassDependency2-srcStageMask-04094
  If the transformFeedback feature is not enabled, srcStageMask **must** not contain VK_PIPELINE_STAGE_TRANSFORM_FEEDBACK_BIT_EXT
- VUID-VkSubpassDependency2-srcStageMask-04095
  If the meshShader feature is not enabled, srcStageMask **must** not contain VK_PIPELINE_STAGE_MESH_SHADER_BIT_EXT
- VUID-VkSubpassDependency2-srcStageMask-04096
  If the taskShader feature is not enabled, srcStageMask **must** not contain VK_PIPELINE_STAGE_TASK_SHADER_BIT_EXT

- VUID-VkSubpassDependency2-srcStageMask-07318
  If neither the `shadingRateImage` or `attachmentFragmentShadingRate` are enabled, `srcStageMask` **must** not contain `VK_PIPELINE_STAGE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR`
- VUID-VkSubpassDependency2-srcStageMask-03937
  If the `synchronization2` feature is not enabled, `srcStageMask` **must** not be `0`
- VUID-VkSubpassDependency2-srcStageMask-07949
  If neither the VK_NV_ray_tracing extension or `rayTracingPipeline` feature are enabled, `srcStageMask` **must** not contain `VK_PIPELINE_STAGE_RAY_TRACING_SHADER_BIT_KHR`
- VUID-VkSubpassDependency2-dstStageMask-04090
  If the `geometryShader` feature is not enabled, `dstStageMask` **must** not contain `VK_PIPELINE_STAGE_GEOMETRY_SHADER_BIT`
- VUID-VkSubpassDependency2-dstStageMask-04091
  If the `tessellationShader` feature is not enabled, `dstStageMask` **must** not contain `VK_PIPELINE_STAGE_TESSELLATION_CONTROL_SHADER_BIT` or `VK_PIPELINE_STAGE_TESSELLATION_EVALUATION_SHADER_BIT`
- VUID-VkSubpassDependency2-dstStageMask-04092
  If the `conditionalRendering` feature is not enabled, `dstStageMask` **must** not contain `VK_PIPELINE_STAGE_CONDITIONAL_RENDERING_BIT_EXT`
- VUID-VkSubpassDependency2-dstStageMask-04093
  If the `fragmentDensityMap` feature is not enabled, `dstStageMask` **must** not contain `VK_PIPELINE_STAGE_FRAGMENT_DENSITY_PROCESS_BIT_EXT`
- VUID-VkSubpassDependency2-dstStageMask-04094
  If the `transformFeedback` feature is not enabled, `dstStageMask` **must** not contain `VK_PIPELINE_STAGE_TRANSFORM_FEEDBACK_BIT_EXT`
- VUID-VkSubpassDependency2-dstStageMask-04095
  If the `meshShader` feature is not enabled, `dstStageMask` **must** not contain `VK_PIPELINE_STAGE_MESH_SHADER_BIT_EXT`
- VUID-VkSubpassDependency2-dstStageMask-04096
  If the `taskShader` feature is not enabled, `dstStageMask` **must** not contain `VK_PIPELINE_STAGE_TASK_SHADER_BIT_EXT`
- VUID-VkSubpassDependency2-dstStageMask-07318
  If neither the `shadingRateImage` or `attachmentFragmentShadingRate` are enabled, `dstStageMask` **must** not contain `VK_PIPELINE_STAGE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR`
- VUID-VkSubpassDependency2-dstStageMask-03937
  If the `synchronization2` feature is not enabled, `dstStageMask` **must** not be `0`
- VUID-VkSubpassDependency2-dstStageMask-07949
  If neither the VK_NV_ray_tracing extension or `rayTracingPipeline` feature are enabled, `dstStageMask` **must** not contain `VK_PIPELINE_STAGE_RAY_TRACING_SHADER_BIT_KHR`
- VUID-VkSubpassDependency2-srcSubpass-03084
  `srcSubpass` **must** be less than or equal to `dstSubpass`, unless one of them is

VK_SUBPASS_EXTERNAL, to avoid cyclic dependencies and ensure a valid execution order

- VUID-VkSubpassDependency2-srcSubpass-03085
  srcSubpass and dstSubpass **must** not both be equal to VK_SUBPASS_EXTERNAL
- VUID-VkSubpassDependency2-srcSubpass-06810
  If srcSubpass is equal to dstSubpass and srcStageMask includes a framebuffer-space stage, dstStageMask **must** only contain framebuffer-space stages
- VUID-VkSubpassDependency2-srcAccessMask-03088
  Any access flag included in srcAccessMask **must** be supported by one of the pipeline stages in srcStageMask, as specified in the table of supported access types
- VUID-VkSubpassDependency2-dstAccessMask-03089
  Any access flag included in dstAccessMask **must** be supported by one of the pipeline stages in dstStageMask, as specified in the table of supported access types
- VUID-VkSubpassDependency2-dependencyFlags-03090
  If dependencyFlags includes VK_DEPENDENCY_VIEW_LOCAL_BIT, srcSubpass **must** not be equal to VK_SUBPASS_EXTERNAL
- VUID-VkSubpassDependency2-dependencyFlags-03091
  If dependencyFlags includes VK_DEPENDENCY_VIEW_LOCAL_BIT, dstSubpass **must** not be equal to VK_SUBPASS_EXTERNAL
- VUID-VkSubpassDependency2-srcSubpass-02245
  If srcSubpass equals dstSubpass, and srcStageMask and dstStageMask both include a framebuffer-space stage, then dependencyFlags **must** include VK_DEPENDENCY_BY_REGION_BIT
- VUID-VkSubpassDependency2-viewOffset-02530
  If viewOffset is not equal to 0, srcSubpass **must** not be equal to dstSubpass
- VUID-VkSubpassDependency2-dependencyFlags-03092
  If dependencyFlags does not include VK_DEPENDENCY_VIEW_LOCAL_BIT, viewOffset **must** be 0

## Valid Usage (Implicit)

- VUID-VkSubpassDependency2-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_SUBPASS_DEPENDENCY_2
- VUID-VkSubpassDependency2-pNext-pNext
  pNext **must** be NULL or a pointer to a valid instance of VkMemoryBarrier2
- VUID-VkSubpassDependency2-sType-unique
  The sType value of each struct in the pNext chain **must** be unique
- VUID-VkSubpassDependency2-srcStageMask-parameter
  srcStageMask **must** be a valid combination of VkPipelineStageFlagBits values
- VUID-VkSubpassDependency2-dstStageMask-parameter
  dstStageMask **must** be a valid combination of VkPipelineStageFlagBits values
- VUID-VkSubpassDependency2-srcAccessMask-parameter
  srcAccessMask **must** be a valid combination of VkAccessFlagBits values
- VUID-VkSubpassDependency2-dstAccessMask-parameter

dstAccessMask **must** be a valid combination of VkAccessFlagBits values

- VUID-VkSubpassDependency2-dependencyFlags-parameter
  dependencyFlags **must** be a valid combination of VkDependencyFlagBits values

To destroy a render pass, call:

```
// Provided by VK_VERSION_1_0
void vkDestroyRenderPass(
    VkDevice                                    device,
    VkRenderPass                                renderPass,
    const VkAllocationCallbacks*                pAllocator);
```

- device is the logical device that destroys the render pass.
- renderPass is the handle of the render pass to destroy.
- pAllocator controls host memory allocation as described in the Memory Allocation chapter.

### Valid Usage

- VUID-vkDestroyRenderPass-renderPass-00873
  All submitted commands that refer to renderPass **must** have completed execution
- VUID-vkDestroyRenderPass-renderPass-00874
  If VkAllocationCallbacks were provided when renderPass was created, a compatible set of callbacks **must** be provided here
- VUID-vkDestroyRenderPass-renderPass-00875
  If no VkAllocationCallbacks were provided when renderPass was created, pAllocator **must** be NULL

### Valid Usage (Implicit)

- VUID-vkDestroyRenderPass-device-parameter
  device **must** be a valid VkDevice handle
- VUID-vkDestroyRenderPass-renderPass-parameter
  If renderPass is not VK_NULL_HANDLE, renderPass **must** be a valid VkRenderPass handle
- VUID-vkDestroyRenderPass-pAllocator-parameter
  If pAllocator is not NULL, pAllocator **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkDestroyRenderPass-renderPass-parent
  If renderPass is a valid handle, it **must** have been created, allocated, or retrieved from device

**Host Synchronization**

- Host access to `renderPass` **must** be externally synchronized

## 8.3. Render Pass Compatibility

Framebuffers and graphics pipelines are created based on a specific render pass object. They **must** only be used with that render pass object, or one compatible with it.

Two attachment references are compatible if they have matching format and sample count, or are both `VK_ATTACHMENT_UNUSED` or the pointer that would contain the reference is `NULL`.

Two arrays of attachment references are compatible if all corresponding pairs of attachments are compatible. If the arrays are of different lengths, attachment references not present in the smaller array are treated as `VK_ATTACHMENT_UNUSED`.

Two render passes are compatible if their corresponding color, input, resolve, and depth/stencil attachment references are compatible and if they are otherwise identical except for:

- Initial and final image layout in attachment descriptions
- Load and store operations in attachment descriptions
- Image layout in attachment references

As an additional special case, if two render passes have a single subpass, the resolve attachment reference and depth/stencil resolve mode compatibility requirements are ignored.

A framebuffer is compatible with a render pass if it was created using the same render pass or a compatible render pass.

## 8.4. Framebuffers

Render passes operate in conjunction with *framebuffers*. Framebuffers represent a collection of specific memory attachments that a render pass instance uses.

Framebuffers are represented by `VkFramebuffer` handles:

```
// Provided by VK_VERSION_1_0
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkFramebuffer)
```

To create a framebuffer, call:

```
// Provided by VK_VERSION_1_0
VkResult vkCreateFramebuffer(
    VkDevice                                    device,
    const VkFramebufferCreateInfo*              pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkFramebuffer*                              pFramebuffer);
```

- `device` is the logical device that creates the framebuffer.
- `pCreateInfo` is a pointer to a VkFramebufferCreateInfo structure describing additional information about framebuffer creation.
- `pAllocator` controls host memory allocation as described in the Memory Allocation chapter.
- `pFramebuffer` is a pointer to a VkFramebuffer handle in which the resulting framebuffer object is returned.

## Valid Usage

- VUID-vkCreateFramebuffer-device-10002
  `device` **must** support at least one queue family with the `VK_QUEUE_GRAPHICS_BIT` capability
- VUID-vkCreateFramebuffer-pCreateInfo-02777
  If `pCreateInfo->flags` does not include `VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT`, and `attachmentCount` is not `0`, each element of `pCreateInfo->pAttachments` **must** have been created on `device`

## Valid Usage (Implicit)

- VUID-vkCreateFramebuffer-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkCreateFramebuffer-pCreateInfo-parameter
  `pCreateInfo` **must** be a valid pointer to a valid VkFramebufferCreateInfo structure
- VUID-vkCreateFramebuffer-pAllocator-parameter
  If `pAllocator` is not `NULL`, `pAllocator` **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkCreateFramebuffer-pFramebuffer-parameter
  `pFramebuffer` **must** be a valid pointer to a VkFramebuffer handle

## Return Codes

### Success

- `VK_SUCCESS`

### Failure

- `VK_ERROR_OUT_OF_HOST_MEMORY`

- VK_ERROR_OUT_OF_DEVICE_MEMORY

The VkFramebufferCreateInfo structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkFramebufferCreateInfo {
    VkStructureType             sType;
    const void*                 pNext;
    VkFramebufferCreateFlags    flags;
    VkRenderPass                renderPass;
    uint32_t                    attachmentCount;
    const VkImageView*          pAttachments;
    uint32_t                    width;
    uint32_t                    height;
    uint32_t                    layers;
} VkFramebufferCreateInfo;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- flags is a bitmask of VkFramebufferCreateFlagBits
- renderPass is a render pass defining what render passes the framebuffer will be compatible with. See Render Pass Compatibility for details.
- attachmentCount is the number of attachments.
- pAttachments is a pointer to an array of VkImageView handles, each of which will be used as the corresponding attachment in a render pass instance. If flags includes VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT, this parameter is ignored.
- width, height and layers define the dimensions of the framebuffer. If the render pass uses multiview, then layers **must** be one and each attachment requires a number of layers that is greater than the maximum bit index set in the view mask in the subpasses in which it is used.

It is legal for a subpass to use no color or depth/stencil attachments, either because it has no attachment references or because all of them are VK_ATTACHMENT_UNUSED. This kind of subpass **can** use shader side effects such as image stores and atomics to produce an output. In this case, the subpass continues to use the width, height, and layers of the framebuffer to define the dimensions of the rendering area, and the rasterizationSamples from each pipeline's VkPipelineMultisampleStateCreateInfo to define the number of samples used in rasterization; however, if VkPhysicalDeviceFeatures::variableMultisampleRate is VK_FALSE, then all pipelines to be bound with the subpass **must** have the same value for VkPipelineMultisampleStateCreateInfo::rasterizationSamples. In all such cases, rasterizationSamples **must** be a valid VkSampleCountFlagBits value that is set in VkPhysicalDeviceLimits::framebufferNoAttachmentsSampleCounts.

### Valid Usage

- VUID-VkFramebufferCreateInfo-attachmentCount-00876

attachmentCount **must** be equal to the attachment count specified in renderPass

- VUID-VkFramebufferCreateInfo-flags-02778
  If flags does not include VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT and attachmentCount is not 0, pAttachments **must** be a valid pointer to an array of attachmentCount valid VkImageView handles
- VUID-VkFramebufferCreateInfo-pAttachments-00877
  If flags does not include VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT, each element of pAttachments that is used as a color attachment or resolve attachment by renderPass **must** have been created with a usage value including VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT
- VUID-VkFramebufferCreateInfo-pAttachments-02633
  If flags does not include VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT, each element of pAttachments that is used as a depth/stencil attachment by renderPass **must** have been created with a usage value including VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT
- VUID-VkFramebufferCreateInfo-pAttachments-02634
  If flags does not include VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT, each element of pAttachments that is used as a depth/stencil resolve attachment by renderPass **must** have been created with a usage value including VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT
- VUID-VkFramebufferCreateInfo-pAttachments-00879
  If renderpass is not VK_NULL_HANDLE, flags does not include VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT, each element of pAttachments that is used as an input attachment by renderPass **must** have been created with a usage value including VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT
- VUID-VkFramebufferCreateInfo-pAttachments-02552
  Each element of pAttachments that is used as a fragment density map attachment by renderPass **must** not have been created with a flags value including VK_IMAGE_CREATE_SUBSAMPLED_BIT_EXT
- VUID-VkFramebufferCreateInfo-renderPass-02553
  If renderPass has a fragment density map attachment and the fragmentDensityMapNonSubsampledImages feature is not enabled, each element of pAttachments **must** have been created with a flags value including VK_IMAGE_CREATE_SUBSAMPLED_BIT_EXT unless that element is the fragment density map attachment
- VUID-VkFramebufferCreateInfo-renderPass-06502
  If renderPass was created with fragment density map offsets other than (0,0), each element of pAttachments **must** have been created with a flags value including VK_IMAGE_CREATE_FRAGMENT_DENSITY_MAP_OFFSET_BIT_QCOM
- VUID-VkFramebufferCreateInfo-pAttachments-00880
  If flags does not include VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT, each element of pAttachments **must** have been created with a VkFormat value that matches the VkFormat specified by the corresponding VkAttachmentDescription in renderPass
- VUID-VkFramebufferCreateInfo-pAttachments-00881
  If flags does not include VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT, each element of pAttachments **must** have been created with a samples value that matches the samples value specified by the corresponding VkAttachmentDescription in renderPass

- VUID-VkFramebufferCreateInfo-flags-04533
  If `flags` does not include `VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT`, each element of `pAttachments` that is used as an input, color, resolve, or depth/stencil attachment by `renderPass` **must** have been created with a VkImageCreateInfo::`extent.width` greater than or equal to `width`
- VUID-VkFramebufferCreateInfo-flags-04534
  If `flags` does not include `VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT`, each element of `pAttachments` that is used as an input, color, resolve, or depth/stencil attachment by `renderPass` **must** have been created with a VkImageCreateInfo::`extent.height` greater than or equal to `height`
- VUID-VkFramebufferCreateInfo-flags-04535
  If `flags` does not include `VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT`, each element of `pAttachments` that is used as an input, color, resolve, or depth/stencil attachment by `renderPass` **must** have been created with a VkImageViewCreateInfo::`subresourceRange.layerCount` greater than or equal to `layers`
- VUID-VkFramebufferCreateInfo-renderPass-04536
  If `renderPass` was specified with non-zero view masks, each element of `pAttachments` that is used as an input, color, resolve, or depth/stencil attachment by `renderPass` **must** have a `layerCount` greater than the index of the most significant bit set in any of those view masks
- VUID-VkFramebufferCreateInfo-renderPass-02746
  Each element of `pAttachments` that is referenced by `fragmentDensityMapAttachment` **must** have a `layerCount` equal to `1` or if `renderPass` was specified with non-zero view masks, greater than the index of the most significant bit set in any of those view masks
- VUID-VkFramebufferCreateInfo-pAttachments-02555
  If `flags` does not include `VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT`, an element of `pAttachments` that is referenced by `fragmentDensityMapAttachment` **must** have a width at least as large as $\lceil \frac{width}{maxFragmentDensityTexelSize_{width}} \rceil$
- VUID-VkFramebufferCreateInfo-pAttachments-02556
  If `flags` does not include `VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT`, an element of `pAttachments` that is referenced by `fragmentDensityMapAttachment` **must** have a height at least as large as $\lceil \frac{height}{maxFragmentDensityTexelSize_{height}} \rceil$
- VUID-VkFramebufferCreateInfo-flags-04537
  If `flags` does not include `VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT`, and `renderPass` was specified with non-zero view masks, each element of `pAttachments` that is used as a fragment shading rate attachment by `renderPass` **must** have a `layerCount` that is either `1`, or greater than the index of the most significant bit set in any of those view masks
- VUID-VkFramebufferCreateInfo-flags-04538
  If `flags` does not include `VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT`, and `renderPass` was not specified with non-zero view masks, each element of `pAttachments` that is used as a fragment shading rate attachment by `renderPass` **must** have a `layerCount` that is either `1`, or greater than `layers`
- VUID-VkFramebufferCreateInfo-flags-04539
  If `maintenance7` is not enabled or the `robustFragmentShadingRateAttachmentAccess` limit is `VK_FALSE` or the `imageView` member of a

VkRenderingFragmentShadingRateAttachmentInfoKHR structure was created with VkImageSubresourceRange::baseMipLevel greater than 0, flags does not include VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT, an element of pAttachments that is used as a fragment shading rate attachment **must** have a width at least as large as ⌈width / texelWidth⌉, where texelWidth is the largest value of shadingRateAttachmentTexelSize.width in a VkFragmentShadingRateAttachmentInfoKHR which references that attachment

- VUID-VkFramebufferCreateInfo-flags-04540
  If maintenance7 is not enabled or the robustFragmentShadingRateAttachmentAccess limit is VK_FALSE or the imageView member of a VkRenderingFragmentShadingRateAttachmentInfoKHR structure was created with VkImageSubresourceRange::baseMipLevel greater than 0, flags does not include VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT, an element of pAttachments that is used as a fragment shading rate attachment **must** have a height at least as large as ⌈height / texelHeight⌉, where texelHeight is the largest value of shadingRateAttachmentTexelSize.height in a VkFragmentShadingRateAttachmentInfoKHR which references that attachment
- VUID-VkFramebufferCreateInfo-pAttachments-00883
  If flags does not include VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT, each element of pAttachments **must** only specify a single mip level
- VUID-VkFramebufferCreateInfo-pAttachments-00884
  If flags does not include VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT, each element of pAttachments **must** have been created with the identity swizzle
- VUID-VkFramebufferCreateInfo-width-00885
  width **must** be greater than 0
- VUID-VkFramebufferCreateInfo-width-00886
  width **must** be less than or equal to maxFramebufferWidth
- VUID-VkFramebufferCreateInfo-height-00887
  height **must** be greater than 0
- VUID-VkFramebufferCreateInfo-height-00888
  height **must** be less than or equal to maxFramebufferHeight
- VUID-VkFramebufferCreateInfo-layers-00889
  layers **must** be greater than 0
- VUID-VkFramebufferCreateInfo-layers-00890
  layers **must** be less than or equal to maxFramebufferLayers
- VUID-VkFramebufferCreateInfo-renderPass-02531
  If renderPass was specified with non-zero view masks, layers **must** be 1
- VUID-VkFramebufferCreateInfo-pAttachments-00891
  If flags does not include VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT, each element of pAttachments that is a 2D or 2D array image view taken from a 3D image **must** not be a depth/stencil format
- VUID-VkFramebufferCreateInfo-flags-03189
  If the imagelessFramebuffer feature is not enabled, flags **must** not include

VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT

- VUID-VkFramebufferCreateInfo-flags-03190
  If flags includes VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT, the pNext chain **must** include a VkFramebufferAttachmentsCreateInfo structure
- VUID-VkFramebufferCreateInfo-flags-03191
  If flags includes VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT, the attachmentImageInfoCount member of a VkFramebufferAttachmentsCreateInfo structure in the pNext chain **must** be equal to either zero or attachmentCount
- VUID-VkFramebufferCreateInfo-flags-04541
  If flags includes VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT, the width member of any element of the pAttachmentImageInfos member of a VkFramebufferAttachmentsCreateInfo structure in the pNext chain that is used as an input, color, resolve or depth/stencil attachment in renderPass **must** be greater than or equal to width
- VUID-VkFramebufferCreateInfo-flags-04542
  If flags includes VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT, the height member of any element of the pAttachmentImageInfos member of a VkFramebufferAttachmentsCreateInfo structure in the pNext chain that is used as an input, color, resolve or depth/stencil attachment in renderPass **must** be greater than or equal to height
- VUID-VkFramebufferCreateInfo-flags-03196
  If flags includes VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT, the width member of any element of the pAttachmentImageInfos member of a VkFramebufferAttachmentsCreateInfo structure in the pNext chain that is referenced by VkRenderPassFragmentDensityMapCreateInfoEXT::fragmentDensityMapAttachment in renderPass **must** be greater than or equal to $\lceil \frac{width}{maxFragmentDensityTexelSize_{width}} \rceil$
- VUID-VkFramebufferCreateInfo-flags-03197
  If flags includes VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT, the height member of any element of the pAttachmentImageInfos member of a VkFramebufferAttachmentsCreateInfo structure included in the pNext chain that is referenced by VkRenderPassFragmentDensityMapCreateInfoEXT::fragmentDensityMapAttachment in renderPass **must** be greater than or equal to $\lceil \frac{height}{maxFragmentDensityTexelSize_{height}} \rceil$
- VUID-VkFramebufferCreateInfo-flags-04543
  If maintenance7 is not enabled or the robustFragmentShadingRateAttachmentAccess limit is VK_FALSE or the imageView member of a VkRenderingFragmentShadingRateAttachmentInfoKHR structure was created with VkImageSubresourceRange::baseMipLevel greater than 0, and flags includes VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT, the width member of any element of the pAttachmentImageInfos member of a VkFramebufferAttachmentsCreateInfo structure in the pNext chain that is used as a fragment shading rate attachment **must** be greater than or equal to ⌊width / texelWidth⌋, where texelWidth is the largest value of shadingRateAttachmentTexelSize.width in a VkFragmentShadingRateAttachmentInfoKHR which references that attachment
- VUID-VkFramebufferCreateInfo-flags-04544
  If maintenance7 is not enabled or the robustFragmentShadingRateAttachmentAccess limit is VK_FALSE or the imageView member of a VkRenderingFragmentShadingRateAttachmentInfoKHR structure was created with

VkImageSubresourceRange::baseMipLevel greater than 0, and flags includes VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT, the height member of any element of the pAttachmentImageInfos member of a VkFramebufferAttachmentsCreateInfo structure in the pNext chain that is used as a fragment shading rate attachment **must** be greater than or equal to ⌊height / texelHeight⌋, where texelHeight is the largest value of shadingRateAttachmentTexelSize.height in a VkFragmentShadingRateAttachmentInfoKHR which references that attachment

- VUID-VkFramebufferCreateInfo-flags-04545
  If flags includes VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT, the layerCount member of any element of the pAttachmentImageInfos member of a VkFramebufferAttachmentsCreateInfo structure in the pNext chain that is used as a fragment shading rate attachment **must** be either 1, or greater than or equal to layers
- VUID-VkFramebufferCreateInfo-flags-04587
  If flags includes VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT and renderPass was specified with non-zero view masks, the layerCount member of any element of the pAttachmentImageInfos member of a VkFramebufferAttachmentsCreateInfo structure in the pNext chain that is used as a fragment shading rate attachment **must** be either 1, or greater than the index of the most significant bit set in any of those view masks
- VUID-VkFramebufferCreateInfo-renderPass-03198
  If multiview is enabled for renderPass and flags includes VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT, the layerCount member of any element of the pAttachmentImageInfos member of a VkFramebufferAttachmentsCreateInfo structure included in the pNext chain used as an input, color, resolve, or depth/stencil attachment in renderPass **must** be greater than the maximum bit index set in the view mask in the subpasses in which it is used in renderPass
- VUID-VkFramebufferCreateInfo-renderPass-04546
  If multiview is not enabled for renderPass and flags includes VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT, the layerCount member of any element of the pAttachmentImageInfos member of a VkFramebufferAttachmentsCreateInfo structure included in the pNext chain used as an input, color, resolve, or depth/stencil attachment in renderPass **must** be greater than or equal to layers
- VUID-VkFramebufferCreateInfo-flags-03201
  If flags includes VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT, the usage member of any element of the pAttachmentImageInfos member of a VkFramebufferAttachmentsCreateInfo structure included in the pNext chain that refers to an attachment used as a color attachment or resolve attachment by renderPass **must** include VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT
- VUID-VkFramebufferCreateInfo-flags-03202
  If flags includes VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT, the usage member of any element of the pAttachmentImageInfos member of a VkFramebufferAttachmentsCreateInfo structure included in the pNext chain that refers to an attachment used as a depth/stencil attachment by renderPass **must** include VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT
- VUID-VkFramebufferCreateInfo-flags-03203
  If flags includes VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT, the usage member of any element of the pAttachmentImageInfos member of a VkFramebufferAttachmentsCreateInfo

structure included in the pNext chain that refers to an attachment used as a depth/stencil resolve attachment by renderPass **must** include VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT

- VUID-VkFramebufferCreateInfo-flags-03204
  If flags includes VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT, the usage member of any element of the pAttachmentImageInfos member of a VkFramebufferAttachmentsCreateInfo structure included in the pNext chain that refers to an attachment used as an input attachment by renderPass **must** include VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT
- VUID-VkFramebufferCreateInfo-flags-03205
  If flags includes VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT, at least one element of the pViewFormats member of any element of the pAttachmentImageInfos member of a VkFramebufferAttachmentsCreateInfo structure included in the pNext chain **must** be equal to the corresponding value of VkAttachmentDescription::format used to create renderPass
- VUID-VkFramebufferCreateInfo-flags-04113
  If flags does not include VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT, each element of pAttachments **must** have been created with VkImageViewCreateInfo::viewType not equal to VK_IMAGE_VIEW_TYPE_3D
- VUID-VkFramebufferCreateInfo-flags-04548
  If flags does not include VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT, each element of pAttachments that is used as a fragment shading rate attachment by renderPass **must** have been created with a usage value including VK_IMAGE_USAGE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR
- VUID-VkFramebufferCreateInfo-flags-04549
  If flags includes VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT, the usage member of any element of the pAttachmentImageInfos member of a VkFramebufferAttachmentsCreateInfo structure included in the pNext chain that refers to an attachment used as a fragment shading rate attachment by renderPass **must** include VK_IMAGE_USAGE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR
- VUID-VkFramebufferCreateInfo-samples-06881
  If multisampled-render-to-single-sampled is enabled for any subpass, all color, depth/stencil and input attachments used in that subpass which have VkAttachmentDescription::samples or VkAttachmentDescription2::samples equal to VK_SAMPLE_COUNT_1_BIT **must** have been created with VK_IMAGE_CREATE_MULTISAMPLED_RENDER_TO_SINGLE_SAMPLED_BIT_EXT in their VkImageCreateInfo::flags
- VUID-VkFramebufferCreateInfo-samples-07009
  If multisampled-render-to-single-sampled is enabled for any subpass, all color, depth/stencil and input attachments used in that subpass which have VkAttachmentDescription::samples or VkAttachmentDescription2::samples equal to VK_SAMPLE_COUNT_1_BIT **must** have a format that supports the sample count specified in VkMultisampledRenderToSingleSampledInfoEXT::rasterizationSamples
- VUID-VkFramebufferCreateInfo-nullColorAttachmentWithExternalFormatResolve-09349
  If the nullColorAttachmentWithExternalFormatResolve is VK_FALSE, and flags does not include VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT, the format of the color attachment for each

subpass in renderPass that includes an external format image as a resolve attachment **must** have a format equal to the value of VkAndroidHardwareBufferFormatResolvePropertiesANDROID::colorAttachmentFormat as returned by a call to vkGetAndroidHardwareBufferPropertiesANDROID for the Android hardware buffer that was used to create the image view use as its resolve attachment

- VUID-VkFramebufferCreateInfo-pAttachments-09350
  If flags does not include VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT, then if an element of pAttachments has a format of VK_FORMAT_UNDEFINED, it **must** have been created with a VkExternalFormatANDROID::externalFormat value identical to that provided in the VkExternalFormatANDROID::externalFormat specified by the corresponding VkAttachmentDescription2 in renderPass

### Valid Usage (Implicit)

- VUID-VkFramebufferCreateInfo-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_FRAMEBUFFER_CREATE_INFO
- VUID-VkFramebufferCreateInfo-pNext-pNext
  pNext **must** be NULL or a pointer to a valid instance of VkFramebufferAttachmentsCreateInfo
- VUID-VkFramebufferCreateInfo-sType-unique
  The sType value of each struct in the pNext chain **must** be unique
- VUID-VkFramebufferCreateInfo-flags-parameter
  flags **must** be a valid combination of VkFramebufferCreateFlagBits values
- VUID-VkFramebufferCreateInfo-renderPass-parameter
  renderPass **must** be a valid VkRenderPass handle
- VUID-VkFramebufferCreateInfo-commonparent
  Both of renderPass, and the elements of pAttachments that are valid handles of non-ignored parameters **must** have been created, allocated, or retrieved from the same VkDevice

The VkFramebufferAttachmentsCreateInfo structure is defined as:

```
typedef struct VkFramebufferAttachmentsCreateInfo {
    VkStructureType                             sType;
    const void*                                 pNext;
    uint32_t                                    attachmentImageInfoCount;
    const VkFramebufferAttachmentImageInfo*     pAttachmentImageInfos;
} VkFramebufferAttachmentsCreateInfo;
```

or the equivalent

```
// Provided by VK_KHR_imageless_framebuffer
typedef VkFramebufferAttachmentsCreateInfo VkFramebufferAttachmentsCreateInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- attachmentImageInfoCount is the number of attachments being described.
- pAttachmentImageInfos is a pointer to an array of VkFramebufferAttachmentImageInfo structures, each structure describing a number of parameters of the corresponding attachment in a render pass instance.

### Valid Usage (Implicit)

- VUID-VkFramebufferAttachmentsCreateInfo-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_FRAMEBUFFER_ATTACHMENTS_CREATE_INFO
- VUID-VkFramebufferAttachmentsCreateInfo-pAttachmentImageInfos-parameter
  If attachmentImageInfoCount is not 0, pAttachmentImageInfos **must** be a valid pointer to an array of attachmentImageInfoCount valid VkFramebufferAttachmentImageInfo structures

The VkFramebufferAttachmentImageInfo structure is defined as:

```
typedef struct VkFramebufferAttachmentImageInfo {
    VkStructureType       sType;
    const void*           pNext;
    VkImageCreateFlags    flags;
    VkImageUsageFlags     usage;
    uint32_t              width;
    uint32_t              height;
    uint32_t              layerCount;
    uint32_t              viewFormatCount;
    const VkFormat*       pViewFormats;
} VkFramebufferAttachmentImageInfo;
```

or the equivalent

```
// Provided by VK_KHR_imageless_framebuffer
typedef VkFramebufferAttachmentImageInfo VkFramebufferAttachmentImageInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- flags is a bitmask of VkImageCreateFlagBits, matching the value of VkImageCreateInfo::flags used to create an image that will be used with this framebuffer.
- usage is a bitmask of VkImageUsageFlagBits, matching the value of VkImageCreateInfo::usage used to create an image used with this framebuffer.
- width is the width of the image view used for rendering.
- height is the height of the image view used for rendering.

- `layerCount` is the number of array layers of the image view used for rendering.
- `viewFormatCount` is the number of entries in the `pViewFormats` array, matching the value of VkImageFormatListCreateInfo::`viewFormatCount` used to create an image used with this framebuffer.
- `pViewFormats` is a pointer to an array of VkFormat values specifying all of the formats which **can** be used when creating views of the image, matching the value of VkImageFormatListCreateInfo::`pViewFormats` used to create an image used with this framebuffer.

Images that **can** be used with the framebuffer when beginning a render pass, as specified by VkRenderPassAttachmentBeginInfo, **must** be created with parameters that are identical to those specified here.

### Valid Usage

- VUID-VkFramebufferAttachmentImageInfo-viewFormatCount-09536
  If `viewFormatCount` is not 0, and the render pass is not being used with an external format resolve attachment, each element of `pViewFormats` **must** not be `VK_FORMAT_UNDEFINED`

### Valid Usage (Implicit)

- VUID-VkFramebufferAttachmentImageInfo-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_FRAMEBUFFER_ATTACHMENT_IMAGE_INFO`
- VUID-VkFramebufferAttachmentImageInfo-pNext-pNext
  `pNext` **must** be `NULL`
- VUID-VkFramebufferAttachmentImageInfo-flags-parameter
  `flags` **must** be a valid combination of VkImageCreateFlagBits values
- VUID-VkFramebufferAttachmentImageInfo-usage-parameter
  `usage` **must** be a valid combination of VkImageUsageFlagBits values
- VUID-VkFramebufferAttachmentImageInfo-usage-requiredbitmask
  `usage` **must** not be `0`
- VUID-VkFramebufferAttachmentImageInfo-pViewFormats-parameter
  If `viewFormatCount` is not `0`, `pViewFormats` **must** be a valid pointer to an array of `viewFormatCount` valid VkFormat values

Bits which **can** be set in VkFramebufferCreateInfo::`flags`, specifying options for framebuffers, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkFramebufferCreateFlagBits {
    VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT = 0x00000001,
  // Provided by VK_KHR_imageless_framebuffer
    VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT_KHR = VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT,
} VkFramebufferCreateFlagBits;
```

- `VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT` specifies that image views are not specified, and only attachment compatibility information will be provided via a VkFramebufferAttachmentImageInfo structure.

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkFramebufferCreateFlags;
```

`VkFramebufferCreateFlags` is a bitmask type for setting a mask of zero or more VkFramebufferCreateFlagBits.

To destroy a framebuffer, call:

```
// Provided by VK_VERSION_1_0
void vkDestroyFramebuffer(
    VkDevice                                    device,
    VkFramebuffer                               framebuffer,
    const VkAllocationCallbacks*                pAllocator);
```

- `device` is the logical device that destroys the framebuffer.
- `framebuffer` is the handle of the framebuffer to destroy.
- `pAllocator` controls host memory allocation as described in the Memory Allocation chapter.

### Valid Usage

- VUID-vkDestroyFramebuffer-framebuffer-00892
  All submitted commands that refer to `framebuffer` **must** have completed execution
- VUID-vkDestroyFramebuffer-framebuffer-00893
  If `VkAllocationCallbacks` were provided when `framebuffer` was created, a compatible set of callbacks **must** be provided here
- VUID-vkDestroyFramebuffer-framebuffer-00894
  If no `VkAllocationCallbacks` were provided when `framebuffer` was created, `pAllocator` **must** be `NULL`

### Valid Usage (Implicit)

- VUID-vkDestroyFramebuffer-device-parameter

device **must** be a valid VkDevice handle

- VUID-vkDestroyFramebuffer-framebuffer-parameter
  If framebuffer is not VK_NULL_HANDLE, framebuffer **must** be a valid VkFramebuffer handle
- VUID-vkDestroyFramebuffer-pAllocator-parameter
  If pAllocator is not NULL, pAllocator **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkDestroyFramebuffer-framebuffer-parent
  If framebuffer is a valid handle, it **must** have been created, allocated, or retrieved from device

**Host Synchronization**

- Host access to framebuffer **must** be externally synchronized

## 8.5. Render Pass Load Operations

Render pass load operations define the initial values of an attachment during a render pass instance.

Load operations for attachments with a depth/stencil format execute in the VK_PIPELINE_STAGE_EARLY_FRAGMENT_TESTS_BIT pipeline stage. Load operations for attachments with a color format execute in the VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT pipeline stage. The load operation for each sample in an attachment happens-before any recorded command which accesses the sample in that render pass instance via that attachment or an alias.

**NOTE** Because load operations always happen first, external synchronization with attachment access only needs to synchronize the load operations with previous commands; not the operations within the render pass instance. This does not apply when using VK_ATTACHMENT_LOAD_OP_NONE_KHR.

Load operations only update values within the defined render area for the render pass instance. However, any writes performed by a load operation (as defined by its access masks) to a given attachment **may** read and write back any memory locations within the image subresource bound for that attachment. For depth/stencil images, if maintenance7 is not enabled on the device or separateDepthStencilAttachmentAccess is VK_FALSE, writes to one aspect **may** also result in read-modify-write operations for the other aspect. If the subresource is in the VK_IMAGE_LAYOUT_ATTACHMENT_FEEDBACK_LOOP_OPTIMAL_EXT layout, implementations **must** not access pixels outside of the render area.

**NOTE** As entire subresources could be accessed by load operations, applications cannot safely access values outside of the render area during a render pass instance when a load operation that modifies values is used.

Load operations that **can** be used for a render pass are:

```
// Provided by VK_VERSION_1_0
typedef enum VkAttachmentLoadOp {
    VK_ATTACHMENT_LOAD_OP_LOAD = 0,
    VK_ATTACHMENT_LOAD_OP_CLEAR = 1,
    VK_ATTACHMENT_LOAD_OP_DONT_CARE = 2,
  // Provided by VK_KHR_load_store_op_none
    VK_ATTACHMENT_LOAD_OP_NONE_KHR = 1000400000,
  // Provided by VK_EXT_load_store_op_none
    VK_ATTACHMENT_LOAD_OP_NONE_EXT = VK_ATTACHMENT_LOAD_OP_NONE_KHR,
} VkAttachmentLoadOp;
```

- `VK_ATTACHMENT_LOAD_OP_LOAD` specifies that the previous contents of the image within the render area will be preserved as the initial values. For attachments with a depth/stencil format, this uses the access type `VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_READ_BIT`. For attachments with a color format, this uses the access type `VK_ACCESS_COLOR_ATTACHMENT_READ_BIT`.
- `VK_ATTACHMENT_LOAD_OP_CLEAR` specifies that the contents within the render area will be cleared to a uniform value, which is specified when a render pass instance is begun. For attachments with a depth/stencil format, this uses the access type `VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_WRITE_BIT`. For attachments with a color format, this uses the access type `VK_ACCESS_COLOR_ATTACHMENT_WRITE_BIT`.
- `VK_ATTACHMENT_LOAD_OP_DONT_CARE` specifies that the previous contents within the area need not be preserved; the contents of the attachment will be undefined inside the render area. For attachments with a depth/stencil format, this uses the access type `VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_WRITE_BIT`. For attachments with a color format, this uses the access type `VK_ACCESS_COLOR_ATTACHMENT_WRITE_BIT`.
- `VK_ATTACHMENT_LOAD_OP_NONE_KHR` specifies that the previous contents of the image will be undefined inside the render pass. No access type is used as the image is not accessed.

During a render pass instance, input and color attachments with color formats that have a component size of 8, 16, or 32 bits **must** be represented in the attachment's format throughout the instance. Attachments with other floating- or fixed-point color formats, or with depth components **may** be represented in a format with a precision higher than the attachment format, but **must** be represented with the same range. When such a component is loaded via the `loadOp`, it will be converted into an implementation-dependent format used by the render pass. Such components **must** be converted from the render pass format, to the format of the attachment, before they are resolved or stored at the end of a render pass instance via `storeOp`. Conversions occur as described in Numeric Representation and Computation and Fixed-Point Data Conversions.

## 8.6. Render Pass Store Operations

Render pass store operations define how values written to an attachment during a render pass instance are stored to memory.

Store operations for attachments with a depth/stencil format execute in the

VK_PIPELINE_STAGE_LATE_FRAGMENT_TESTS_BIT pipeline stage. Store operations for attachments with a color format execute in the VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT pipeline stage. The store operation for each sample in an attachment happens-after any recorded command which accesses the sample via that attachment or an alias.

**NOTE** Because store operations always happen after other accesses in a render pass instance, external synchronization with attachment access in an earlier render pass only needs to synchronize with the store operations; not the operations within the render pass instance. This does not apply when using VK_ATTACHMENT_STORE_OP_NONE.

Store operations only update values within the defined render area for the render pass instance. However, any writes performed by a store operation (as defined by its access masks) to a given attachment **may** read and write back any memory locations within the image subresource bound for that attachment. For depth/stencil images, if separateDepthStencilAttachmentAccess is VK_FALSE, writes to one aspect **may** also result in read-modify-write operations for the other aspect. If the subresource is in the VK_IMAGE_LAYOUT_ATTACHMENT_FEEDBACK_LOOP_OPTIMAL_EXT layout, implementations **must** not access pixels outside of the render area.

**NOTE** As entire subresources could be accessed by store operations, applications cannot safely access values outside of the render area via aliased resources during a render pass instance when a store operation that modifies values is used.

Possible values of VkAttachmentDescription::storeOp and stencilStoreOp, specifying how the contents of the attachment are treated, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkAttachmentStoreOp {
    VK_ATTACHMENT_STORE_OP_STORE = 0,
    VK_ATTACHMENT_STORE_OP_DONT_CARE = 1,
    VK_ATTACHMENT_STORE_OP_NONE = 1000301000,
  // Provided by VK_KHR_dynamic_rendering, VK_KHR_load_store_op_none
    VK_ATTACHMENT_STORE_OP_NONE_KHR = VK_ATTACHMENT_STORE_OP_NONE,
  // Provided by VK_QCOM_render_pass_store_ops
    VK_ATTACHMENT_STORE_OP_NONE_QCOM = VK_ATTACHMENT_STORE_OP_NONE,
  // Provided by VK_EXT_load_store_op_none
    VK_ATTACHMENT_STORE_OP_NONE_EXT = VK_ATTACHMENT_STORE_OP_NONE,
} VkAttachmentStoreOp;
```

- VK_ATTACHMENT_STORE_OP_STORE specifies the contents generated during the render pass and within the render area are written to memory. For attachments with a depth/stencil format, this uses the access type VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_WRITE_BIT. For attachments with a color format, this uses the access type VK_ACCESS_COLOR_ATTACHMENT_WRITE_BIT.
- VK_ATTACHMENT_STORE_OP_DONT_CARE specifies the contents within the render area are not needed after rendering, and **may** be discarded; the contents of the attachment will be undefined inside the render area. For attachments with a depth/stencil format, this uses the access type VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_WRITE_BIT. For attachments with a color format, this uses the access type VK_ACCESS_COLOR_ATTACHMENT_WRITE_BIT.

- VK_ATTACHMENT_STORE_OP_NONE specifies the contents within the render area are not accessed by the store operation as long as no values are written to the attachment during the render pass. If values are written during the render pass, this behaves identically to VK_ATTACHMENT_STORE_OP_DONT_CARE and with matching access semantics.

**NOTE** VK_ATTACHMENT_STORE_OP_DONT_CARE **can** cause contents generated during previous render passes to be discarded before reaching memory, even if no write to the attachment occurs during the current render pass.

## 8.7. Render Pass Multisample Resolve Operations

Render pass multisample resolve operations combine sample values from a single pixel in a multisample attachment and store the result to the corresponding pixel in a single sample attachment.

Multisample resolve operations for attachments execute in the VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT pipeline stage. A final resolve operation for all pixels in the render area happens-after any recorded command which writes a pixel via the multisample attachment to be resolved or an explicit alias of it in the subpass that it is specified. Any single sample attachment specified for use in a multisample resolve operation **may** have its contents modified at any point once rendering begins for the render pass instance. Reads from the multisample attachment can be synchronized with VK_ACCESS_COLOR_ATTACHMENT_READ_BIT. Access to the single sample attachment can be synchronized with VK_ACCESS_COLOR_ATTACHMENT_READ_BIT and VK_ACCESS_COLOR_ATTACHMENT_WRITE_BIT. These pipeline stage and access types are used whether the attachments are color or depth/stencil attachments.

When using render pass objects, a subpass dependency specified with the above pipeline stages and access flags will ensure synchronization with multisample resolve operations for any attachments that were last accessed by that subpass. This allows later subpasses to read resolved values as input attachments.

Resolve operations only update values within the defined render area for the render pass instance. However, any writes performed by a resolve operation (as defined by its access masks) to a given attachment **may** read and write back any memory locations within the image subresource bound for that attachment. For depth/stencil images, if separateDepthStencilAttachmentAccess is VK_FALSE, writes to one aspect **may** also result in read-modify-write operations for the other aspect. If the subresource is in the VK_IMAGE_LAYOUT_ATTACHMENT_FEEDBACK_LOOP_OPTIMAL_EXT layout, implementations **must** not access pixels outside of the render area.

**NOTE** As entire subresources could be accessed by multisample resolve operations, applications cannot safely access values outside of the render area via aliased resources during a render pass instance when a multisample resolve operation is performed.

Multisample values in a multisample attachment are combined according to the resolve mode used:

```
typedef enum VkResolveModeFlagBits {
    VK_RESOLVE_MODE_NONE = 0,
    VK_RESOLVE_MODE_SAMPLE_ZERO_BIT = 0x00000001,
    VK_RESOLVE_MODE_AVERAGE_BIT = 0x00000002,
    VK_RESOLVE_MODE_MIN_BIT = 0x00000004,
    VK_RESOLVE_MODE_MAX_BIT = 0x00000008,
  // Provided by VK_ANDROID_external_format_resolve with VK_KHR_dynamic_rendering or
VK_VERSION_1_3
    VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID = 0x00000010,
  // Provided by VK_KHR_depth_stencil_resolve
    VK_RESOLVE_MODE_NONE_KHR = VK_RESOLVE_MODE_NONE,
  // Provided by VK_KHR_depth_stencil_resolve
    VK_RESOLVE_MODE_SAMPLE_ZERO_BIT_KHR = VK_RESOLVE_MODE_SAMPLE_ZERO_BIT,
  // Provided by VK_KHR_depth_stencil_resolve
    VK_RESOLVE_MODE_AVERAGE_BIT_KHR = VK_RESOLVE_MODE_AVERAGE_BIT,
  // Provided by VK_KHR_depth_stencil_resolve
    VK_RESOLVE_MODE_MIN_BIT_KHR = VK_RESOLVE_MODE_MIN_BIT,
  // Provided by VK_KHR_depth_stencil_resolve
    VK_RESOLVE_MODE_MAX_BIT_KHR = VK_RESOLVE_MODE_MAX_BIT,
} VkResolveModeFlagBits;
```

or the equivalent

```
// Provided by VK_KHR_depth_stencil_resolve
typedef VkResolveModeFlagBits VkResolveModeFlagBitsKHR;
```

- `VK_RESOLVE_MODE_NONE` indicates that no resolve operation is done.
- `VK_RESOLVE_MODE_SAMPLE_ZERO_BIT` indicates that result of the resolve operation is equal to the value of sample 0.
- `VK_RESOLVE_MODE_AVERAGE_BIT` indicates that result of the resolve operation is the average of the sample values.
- `VK_RESOLVE_MODE_MIN_BIT` indicates that result of the resolve operation is the minimum of the sample values.
- `VK_RESOLVE_MODE_MAX_BIT` indicates that result of the resolve operation is the maximum of the sample values.
- `VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID` indicates that rather than a multisample resolve, a single sampled color attachment will be downsampled into a $Y'C_BC_R$ format image specified by an external Android format. Unlike other resolve modes, implementations can resolve multiple times during rendering, or even bypass writing to the color attachment altogether, as long as the final value is resolved to the resolve attachment. Values in the G, B, and R channels of the color attachment will be written to the Y, $C_B$, and $C_R$ channels of the external format image, respectively. Chroma values are calculated as if sampling with a linear filter from the color attachment at full rate, at the location the chroma values sit according to VkPhysicalDeviceExternalFormatResolvePropertiesANDROID::`chromaOffsetX`, VkPhysicalDeviceExternalFormatResolvePropertiesANDROID::`chromaOffsetY`, and the chroma

sample rate of the resolved image.

If no resolve mode is otherwise specified, VK_RESOLVE_MODE_AVERAGE_BIT is used.

**NOTE** No range compression or $Y'C_BC_R$ model conversion is performed by VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID; applications have to do these conversions themselves. Value outputs are expected to match those that would be read through a $Y'C_BC_R$ sampler using VK_SAMPLER_YCBCR_MODEL_CONVERSION_RGB_IDENTITY. The color space that the values should be in is defined by the platform and is not exposed via Vulkan.

```
typedef VkFlags VkResolveModeFlags;
```

or the equivalent

```
// Provided by VK_KHR_depth_stencil_resolve
typedef VkResolveModeFlags VkResolveModeFlagsKHR;
```

VkResolveModeFlags is a bitmask type for setting a mask of zero or more VkResolveModeFlagBits.

## 8.8. Render Pass Commands

An application records the commands for a render pass instance one subpass at a time, by beginning a render pass instance, iterating over the subpasses to record commands for that subpass, and then ending the render pass instance.

To begin a render pass instance, call:

```
// Provided by VK_VERSION_1_0
void vkCmdBeginRenderPass(
    VkCommandBuffer                             commandBuffer,
    const VkRenderPassBeginInfo*                pRenderPassBegin,
    VkSubpassContents                           contents);
```

- commandBuffer is the command buffer in which to record the command.
- pRenderPassBegin is a pointer to a VkRenderPassBeginInfo structure specifying the render pass to begin an instance of, and the framebuffer the instance uses.
- contents is a VkSubpassContents value specifying how the commands in the first subpass will be provided.

After beginning a render pass instance, the command buffer is ready to record the commands for the first subpass of that render pass.

## Valid Usage

- VUID-vkCmdBeginRenderPass-initialLayout-00895
  If any of the `initialLayout` or `finalLayout` member of the `VkAttachmentDescription` structures or the `layout` member of the `VkAttachmentReference` structures specified when creating the render pass specified in the `renderPass` member of `pRenderPassBegin` is `VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL` then the corresponding attachment image view of the framebuffer specified in the `framebuffer` member of `pRenderPassBegin` **must** have been created with a `usage` value including `VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT`
- VUID-vkCmdBeginRenderPass-initialLayout-01758
  If any of the `initialLayout` or `finalLayout` member of the `VkAttachmentDescription` structures or the `layout` member of the `VkAttachmentReference` structures specified when creating the render pass specified in the `renderPass` member of `pRenderPassBegin` is `VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_STENCIL_ATTACHMENT_OPTIMAL`, `VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_STENCIL_READ_ONLY_OPTIMAL`, `VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL`, or `VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL` then the corresponding attachment image view of the framebuffer specified in the `framebuffer` member of `pRenderPassBegin` **must** have been created with a `usage` value including `VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT`
- VUID-vkCmdBeginRenderPass-initialLayout-02842
  If any of the `initialLayout` or `finalLayout` member of the `VkAttachmentDescription` structures or the `layout` member of the `VkAttachmentReference` structures specified when creating the render pass specified in the `renderPass` member of `pRenderPassBegin` is `VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL`, or `VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL`, `VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL`, or `VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL` then the corresponding attachment image view of the framebuffer specified in the `framebuffer` member of `pRenderPassBegin` **must** have been created with a `usage` value including `VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT`
- VUID-vkCmdBeginRenderPass-stencilInitialLayout-02843
  If any of the `stencilInitialLayout` or `stencilFinalLayout` member of the `VkAttachmentDescriptionStencilLayout` structures or the `stencilLayout` member of the `VkAttachmentReferenceStencilLayout` structures specified when creating the render pass specified in the `renderPass` member of `pRenderPassBegin` is `VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL`, or `VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL` then the corresponding attachment image view of the framebuffer specified in the `framebuffer` member of `pRenderPassBegin` **must** have been created with a `usage` value including `VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT`
- VUID-vkCmdBeginRenderPass-initialLayout-00897
  If any of the `initialLayout` or `finalLayout` member of the `VkAttachmentDescription` structures or the `layout` member of the `VkAttachmentReference` structures specified when creating the render pass specified in the `renderPass` member of `pRenderPassBegin` is `VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL` then the corresponding attachment image view

of the framebuffer specified in the `framebuffer` member of `pRenderPassBegin` **must** have been created with a `usage` value including `VK_IMAGE_USAGE_SAMPLED_BIT` or `VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT`

- VUID-vkCmdBeginRenderPass-initialLayout-00898
  If any of the `initialLayout` or `finalLayout` member of the `VkAttachmentDescription` structures or the `layout` member of the `VkAttachmentReference` structures specified when creating the render pass specified in the `renderPass` member of `pRenderPassBegin` is `VK_IMAGE_LAYOUT_TRANSFER_SRC_OPTIMAL` then the corresponding attachment image view of the framebuffer specified in the `framebuffer` member of `pRenderPassBegin` **must** have been created with a `usage` value including `VK_IMAGE_USAGE_TRANSFER_SRC_BIT`
- VUID-vkCmdBeginRenderPass-initialLayout-00899
  If any of the `initialLayout` or `finalLayout` member of the `VkAttachmentDescription` structures or the `layout` member of the `VkAttachmentReference` structures specified when creating the render pass specified in the `renderPass` member of `pRenderPassBegin` is `VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL` then the corresponding attachment image view of the framebuffer specified in the `framebuffer` member of `pRenderPassBegin` **must** have been created with a `usage` value including `VK_IMAGE_USAGE_TRANSFER_DST_BIT`
- VUID-vkCmdBeginRenderPass-initialLayout-00900
  If the `initialLayout` member of any of the `VkAttachmentDescription` structures specified when creating the render pass specified in the `renderPass` member of `pRenderPassBegin` is not `VK_IMAGE_LAYOUT_UNDEFINED`, then each such `initialLayout` **must** be equal to the current layout of the corresponding attachment image subresource of the framebuffer specified in the `framebuffer` member of `pRenderPassBegin`
- VUID-vkCmdBeginRenderPass-srcStageMask-06451
  The `srcStageMask` members of any element of the `pDependencies` member of VkRenderPassCreateInfo used to create `renderPass` **must** be supported by the capabilities of the queue family identified by the `queueFamilyIndex` member of the VkCommandPoolCreateInfo used to create the command pool which `commandBuffer` was allocated from
- VUID-vkCmdBeginRenderPass-dstStageMask-06452
  The `dstStageMask` members of any element of the `pDependencies` member of VkRenderPassCreateInfo used to create `renderPass` **must** be supported by the capabilities of the queue family identified by the `queueFamilyIndex` member of the VkCommandPoolCreateInfo used to create the command pool which `commandBuffer` was allocated from
- VUID-vkCmdBeginRenderPass-framebuffer-02532
  For any attachment in `framebuffer` that is used by `renderPass` and is bound to memory locations that are also bound to another attachment used by `renderPass`, and if at least one of those uses causes either attachment to be written to, both attachments **must** have had the `VK_ATTACHMENT_DESCRIPTION_MAY_ALIAS_BIT` set
- VUID-vkCmdBeginRenderPass-framebuffer-09045
  If any attachments specified in `framebuffer` are used by `renderPass` and are bound to overlapping memory locations, there **must** be only one that is used as a color attachment, depth/stencil, or resolve attachment in any subpass
- VUID-vkCmdBeginRenderPass-initialLayout-07000

If any of the initialLayout or finalLayout member of the VkAttachmentDescription structures or the layout member of the VkAttachmentReference structures specified when creating the render pass specified in the renderPass member of pRenderPassBegin is VK_IMAGE_LAYOUT_ATTACHMENT_FEEDBACK_LOOP_OPTIMAL_EXT then the corresponding attachment image view of the framebuffer specified in the framebuffer member of pRenderPassBegin **must** have been created with a usage value including either the VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT or VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT and either the VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT or VK_IMAGE_USAGE_SAMPLED_BIT usage bits

- VUID-vkCmdBeginRenderPass-initialLayout-07001
  If any of the initialLayout or finalLayout member of the VkAttachmentDescription structures or the layout member of the VkAttachmentReference structures specified when creating the render pass specified in the renderPass member of pRenderPassBegin is VK_IMAGE_LAYOUT_ATTACHMENT_FEEDBACK_LOOP_OPTIMAL_EXT then the corresponding attachment image view of the framebuffer specified in the framebuffer member of pRenderPassBegin **must** have been created with a usage value the VK_IMAGE_USAGE_ATTACHMENT_FEEDBACK_LOOP_BIT_EXT usage bit
- VUID-vkCmdBeginRenderPass-initialLayout-09537
  If any of the initialLayout or finalLayout member of the VkAttachmentDescription structures or the layout member of the VkAttachmentReference structures specified when creating the render pass specified in the renderPass member of pRenderPassBegin is VK_IMAGE_LAYOUT_RENDERING_LOCAL_READ_KHR then the corresponding attachment image view of the framebuffer specified in the framebuffer member of pRenderPassBegin **must** have been created with a usage value including either VK_IMAGE_USAGE_STORAGE_BIT, or both VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT and either of VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT or VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT
- VUID-vkCmdBeginRenderPass-contents-09640
  If contents is VK_SUBPASS_CONTENTS_INLINE_AND_SECONDARY_COMMAND_BUFFERS_KHR, then at least one of the following features **must** be enabled:
  - maintenance7
  - nestedCommandBuffer

## Valid Usage (Implicit)

- VUID-vkCmdBeginRenderPass-commandBuffer-parameter
  commandBuffer **must** be a valid VkCommandBuffer handle
- VUID-vkCmdBeginRenderPass-pRenderPassBegin-parameter
  pRenderPassBegin **must** be a valid pointer to a valid VkRenderPassBeginInfo structure
- VUID-vkCmdBeginRenderPass-contents-parameter
  contents **must** be a valid VkSubpassContents value
- VUID-vkCmdBeginRenderPass-commandBuffer-recording
  commandBuffer **must** be in the recording state
- VUID-vkCmdBeginRenderPass-commandBuffer-cmdpool
  The VkCommandPool that commandBuffer was allocated from **must** support graphics

operations

- VUID-vkCmdBeginRenderPass-renderpass
  This command **must** only be called outside of a render pass instance
- VUID-vkCmdBeginRenderPass-videocoding
  This command **must** only be called outside of a video coding scope
- VUID-vkCmdBeginRenderPass-bufferlevel
  `commandBuffer` **must** be a primary `VkCommandBuffer`

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary | Outside | Outside | Graphics | Action<br>State<br>Synchronization |

Alternatively to begin a render pass, call:

```
// Provided by VK_KHR_create_renderpass2
void vkCmdBeginRenderPass2KHR(
    VkCommandBuffer                             commandBuffer,
    const VkRenderPassBeginInfo*                pRenderPassBegin,
    const VkSubpassBeginInfo*                   pSubpassBeginInfo);
```

- `commandBuffer` is the command buffer in which to record the command.
- `pRenderPassBegin` is a pointer to a VkRenderPassBeginInfo structure specifying the render pass to begin an instance of, and the framebuffer the instance uses.
- `pSubpassBeginInfo` is a pointer to a VkSubpassBeginInfo structure containing information about the subpass which is about to begin rendering.

After beginning a render pass instance, the command buffer is ready to record the commands for the first subpass of that render pass.

## Valid Usage

- VUID-vkCmdBeginRenderPass2-framebuffer-02779
  Both the `framebuffer` and `renderPass` members of `pRenderPassBegin` **must** have been created on the same VkDevice that `commandBuffer` was allocated on
- VUID-vkCmdBeginRenderPass2-initialLayout-03094
  If any of the `initialLayout` or `finalLayout` member of the `VkAttachmentDescription` structures or the `layout` member of the `VkAttachmentReference` structures specified when creating the render pass specified in the `renderPass` member of `pRenderPassBegin` is `VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL` then the corresponding attachment image view of the framebuffer specified in the `framebuffer` member of `pRenderPassBegin` **must** have been created with a `usage` value including `VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT`
- VUID-vkCmdBeginRenderPass2-initialLayout-03096
  If any of the `initialLayout` or `finalLayout` member of the `VkAttachmentDescription` structures or the `layout` member of the `VkAttachmentReference` structures specified when creating the render pass specified in the `renderPass` member of `pRenderPassBegin` is `VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_STENCIL_ATTACHMENT_OPTIMAL`, `VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_STENCIL_READ_ONLY_OPTIMAL`, `VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL`, or `VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL` then the corresponding attachment image view of the framebuffer specified in the `framebuffer` member of `pRenderPassBegin` **must** have been created with a `usage` value including `VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT`
- VUID-vkCmdBeginRenderPass2-initialLayout-02844
  If any of the `initialLayout` or `finalLayout` member of the `VkAttachmentDescription` structures or the `layout` member of the `VkAttachmentReference` structures specified when creating the render pass specified in the `renderPass` member of `pRenderPassBegin` is `VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL`, or `VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL`, `VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL`, or `VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL` then the corresponding attachment image view of the framebuffer specified in the `framebuffer` member of `pRenderPassBegin` **must** have been created with a `usage` value including `VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT`
- VUID-vkCmdBeginRenderPass2-stencilInitialLayout-02845
  If any of the `stencilInitialLayout` or `stencilFinalLayout` member of the `VkAttachmentDescriptionStencilLayout` structures or the `stencilLayout` member of the `VkAttachmentReferenceStencilLayout` structures specified when creating the render pass specified in the `renderPass` member of `pRenderPassBegin` is `VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL`, or `VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL` then the corresponding attachment image view of the framebuffer specified in the `framebuffer` member of `pRenderPassBegin` **must** have been created with a `usage` value including `VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT`
- VUID-vkCmdBeginRenderPass2-initialLayout-03097
  If any of the `initialLayout` or `finalLayout` member of the `VkAttachmentDescription`

structures or the layout member of the VkAttachmentReference structures specified when creating the render pass specified in the renderPass member of pRenderPassBegin is VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL then the corresponding attachment image view of the framebuffer specified in the framebuffer member of pRenderPassBegin **must** have been created with a usage value including VK_IMAGE_USAGE_SAMPLED_BIT or VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT

- VUID-vkCmdBeginRenderPass2-initialLayout-03098
  If any of the initialLayout or finalLayout member of the VkAttachmentDescription structures or the layout member of the VkAttachmentReference structures specified when creating the render pass specified in the renderPass member of pRenderPassBegin is VK_IMAGE_LAYOUT_TRANSFER_SRC_OPTIMAL then the corresponding attachment image view of the framebuffer specified in the framebuffer member of pRenderPassBegin **must** have been created with a usage value including VK_IMAGE_USAGE_TRANSFER_SRC_BIT
- VUID-vkCmdBeginRenderPass2-initialLayout-03099
  If any of the initialLayout or finalLayout member of the VkAttachmentDescription structures or the layout member of the VkAttachmentReference structures specified when creating the render pass specified in the renderPass member of pRenderPassBegin is VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL then the corresponding attachment image view of the framebuffer specified in the framebuffer member of pRenderPassBegin **must** have been created with a usage value including VK_IMAGE_USAGE_TRANSFER_DST_BIT
- VUID-vkCmdBeginRenderPass2-initialLayout-03100
  If the initialLayout member of any of the VkAttachmentDescription structures specified when creating the render pass specified in the renderPass member of pRenderPassBegin is not VK_IMAGE_LAYOUT_UNDEFINED, then each such initialLayout **must** be equal to the current layout of the corresponding attachment image subresource of the framebuffer specified in the framebuffer member of pRenderPassBegin
- VUID-vkCmdBeginRenderPass2-srcStageMask-06453
  The srcStageMask members of any element of the pDependencies member of VkRenderPassCreateInfo used to create renderPass **must** be supported by the capabilities of the queue family identified by the queueFamilyIndex member of the VkCommandPoolCreateInfo used to create the command pool which commandBuffer was allocated from
- VUID-vkCmdBeginRenderPass2-dstStageMask-06454
  The dstStageMask members of any element of the pDependencies member of VkRenderPassCreateInfo used to create renderPass **must** be supported by the capabilities of the queue family identified by the queueFamilyIndex member of the VkCommandPoolCreateInfo used to create the command pool which commandBuffer was allocated from
- VUID-vkCmdBeginRenderPass2-framebuffer-02533
  For any attachment in framebuffer that is used by renderPass and is bound to memory locations that are also bound to another attachment used by renderPass, and if at least one of those uses causes either attachment to be written to, both attachments **must** have had the VK_ATTACHMENT_DESCRIPTION_MAY_ALIAS_BIT set
- VUID-vkCmdBeginRenderPass2-framebuffer-09046
  If any attachments specified in framebuffer are used by renderPass and are bound to

overlapping memory locations, there **must** be only one that is used as a color attachment, depth/stencil, or resolve attachment in any subpass

- VUID-vkCmdBeginRenderPass2-initialLayout-07002
  If any of the `initialLayout` or `finalLayout` member of the `VkAttachmentDescription` structures or the `layout` member of the `VkAttachmentReference` structures specified when creating the render pass specified in the `renderPass` member of `pRenderPassBegin` is `VK_IMAGE_LAYOUT_ATTACHMENT_FEEDBACK_LOOP_OPTIMAL_EXT` then the corresponding attachment image view of the framebuffer specified in the `framebuffer` member of `pRenderPassBegin` **must** have been created with a `usage` value including either the `VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT` or `VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT` and either the `VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT` or `VK_IMAGE_USAGE_SAMPLED_BIT` usage bits
- VUID-vkCmdBeginRenderPass2-initialLayout-07003
  If any of the `initialLayout` or `finalLayout` member of the `VkAttachmentDescription` structures or the `layout` member of the `VkAttachmentReference` structures specified when creating the render pass specified in the `renderPass` member of `pRenderPassBegin` is `VK_IMAGE_LAYOUT_ATTACHMENT_FEEDBACK_LOOP_OPTIMAL_EXT` then the corresponding attachment image view of the framebuffer specified in the `framebuffer` member of `pRenderPassBegin` **must** have been created with a `usage` value the `VK_IMAGE_USAGE_ATTACHMENT_FEEDBACK_LOOP_BIT_EXT` usage bit
- VUID-vkCmdBeginRenderPass2-initialLayout-09538
  If any of the `initialLayout` or `finalLayout` member of the `VkAttachmentDescription` structures or the `layout` member of the `VkAttachmentReference` structures specified when creating the render pass specified in the `renderPass` member of `pRenderPassBegin` is `VK_IMAGE_LAYOUT_RENDERING_LOCAL_READ_KHR` then the corresponding attachment image view of the framebuffer specified in the `framebuffer` member of `pRenderPassBegin` **must** have been created with a `usage` value including either `VK_IMAGE_USAGE_STORAGE_BIT`, or both `VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT` and either of `VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT` or `VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT`

## Valid Usage (Implicit)

- VUID-vkCmdBeginRenderPass2-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdBeginRenderPass2-pRenderPassBegin-parameter
  `pRenderPassBegin` **must** be a valid pointer to a valid VkRenderPassBeginInfo structure
- VUID-vkCmdBeginRenderPass2-pSubpassBeginInfo-parameter
  `pSubpassBeginInfo` **must** be a valid pointer to a valid VkSubpassBeginInfo structure
- VUID-vkCmdBeginRenderPass2-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdBeginRenderPass2-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- VUID-vkCmdBeginRenderPass2-renderpass

This command **must** only be called outside of a render pass instance

- VUID-vkCmdBeginRenderPass2-videocoding
  This command **must** only be called outside of a video coding scope
- VUID-vkCmdBeginRenderPass2-bufferlevel
  `commandBuffer` **must** be a primary `VkCommandBuffer`

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary | Outside | Outside | Graphics | Action<br>State<br>Synchronization |

The `VkRenderPassBeginInfo` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkRenderPassBeginInfo {
    VkStructureType        sType;
    const void*            pNext;
    VkRenderPass           renderPass;
    VkFramebuffer          framebuffer;
    VkRect2D               renderArea;
    uint32_t               clearValueCount;
    const VkClearValue*    pClearValues;
} VkRenderPassBeginInfo;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `renderPass` is the render pass to begin an instance of.
- `framebuffer` is the framebuffer containing the attachments that are used with the render pass.
- `renderArea` is the render area that is affected by the render pass instance, and is described in more detail below.
- `clearValueCount` is the number of elements in `pClearValues`.

- pClearValues is a pointer to an array of clearValueCount VkClearValue structures containing clear values for each attachment, if the attachment uses a loadOp value of VK_ATTACHMENT_LOAD_OP_CLEAR or if the attachment has a depth/stencil format and uses a stencilLoadOp value of VK_ATTACHMENT_LOAD_OP_CLEAR. The array is indexed by attachment number. Only elements corresponding to cleared attachments are used. Other elements of pClearValues are ignored.

renderArea is the render area that is affected by the render pass instance. The effects of attachment load, store and multisample resolve operations are restricted to the pixels whose x and y coordinates fall within the render area on all attachments. The render area extends to all layers of framebuffer. The application **must** ensure (using scissor if necessary) that all rendering is contained within the render area. The render area, after any transform specified by VkRenderPassTransformBeginInfoQCOM::transform is applied, **must** be contained within the framebuffer dimensions.

If render pass transform is enabled, then renderArea **must** equal the framebuffer pre-transformed dimensions. After renderArea has been transformed by VkRenderPassTransformBeginInfoQCOM ::transform, the resulting render area **must** be equal to the framebuffer dimensions.

If multiview is enabled in renderPass, and multiviewPerViewRenderAreas feature is enabled, and there is an instance of VkMultiviewPerViewRenderAreasRenderPassBeginInfoQCOM included in the pNext chain with perViewRenderAreaCount not equal to 0, then the elements of VkMultiviewPerViewRenderAreasRenderPassBeginInfoQCOM::pPerViewRenderAreas override renderArea and define a render area for each view. In this case, renderArea **must** be set to an area at least as large as the union of all the per-view render areas.

If the subpassShading feature is enabled, then renderArea **must** equal the framebuffer dimensions.

| | |
|---|---|
| **NOTE** | There **may** be a performance cost for using a render area smaller than the framebuffer, unless it matches the render area granularity for the render pass. |

## Valid Usage

- VUID-VkRenderPassBeginInfo-clearValueCount-00902
  clearValueCount **must** be greater than the largest attachment index in renderPass specifying a loadOp (or stencilLoadOp, if the attachment has a depth/stencil format) of VK_ATTACHMENT_LOAD_OP_CLEAR
- VUID-VkRenderPassBeginInfo-clearValueCount-04962
  If clearValueCount is not 0, pClearValues **must** be a valid pointer to an array of clearValueCount VkClearValue unions
- VUID-VkRenderPassBeginInfo-renderPass-00904
  renderPass **must** be compatible with the renderPass member of the VkFramebufferCreateInfo structure specified when creating framebuffer
- VUID-VkRenderPassBeginInfo-None-08996
  If the pNext chain does not contain VkDeviceGroupRenderPassBeginInfo or its deviceRenderAreaCount member is equal to 0, renderArea.extent.width **must** be greater than 0

- VUID-VkRenderPassBeginInfo-None-08997
  If the `pNext` chain does not contain VkDeviceGroupRenderPassBeginInfo or its `deviceRenderAreaCount` member is equal to 0, `renderArea.extent.height` **must** be greater than 0
- VUID-VkRenderPassBeginInfo-pNext-02850
  If the `pNext` chain does not contain VkDeviceGroupRenderPassBeginInfo or its `deviceRenderAreaCount` member is equal to 0, `renderArea.offset.x` **must** be greater than or equal to 0
- VUID-VkRenderPassBeginInfo-pNext-02851
  If the `pNext` chain does not contain VkDeviceGroupRenderPassBeginInfo or its `deviceRenderAreaCount` member is equal to 0, `renderArea.offset.y` **must** be greater than or equal to 0
- VUID-VkRenderPassBeginInfo-pNext-02852
  If the `pNext` chain does not contain VkDeviceGroupRenderPassBeginInfo or its `deviceRenderAreaCount` member is equal to 0, `renderArea.offset.x` + `renderArea.extent.width` **must** be less than or equal to VkFramebufferCreateInfo::`width` the `framebuffer` was created with
- VUID-VkRenderPassBeginInfo-pNext-02853
  If the `pNext` chain does not contain VkDeviceGroupRenderPassBeginInfo or its `deviceRenderAreaCount` member is equal to 0, `renderArea.offset.y` + `renderArea.extent.height` **must** be less than or equal to VkFramebufferCreateInfo::`height` the `framebuffer` was created with
- VUID-VkRenderPassBeginInfo-pNext-02856
  If the `pNext` chain contains VkDeviceGroupRenderPassBeginInfo, `offset.x` + `extent.width` of each element of `pDeviceRenderAreas` **must** be less than or equal to VkFramebufferCreateInfo::`width` the `framebuffer` was created with
- VUID-VkRenderPassBeginInfo-pNext-02857
  If the `pNext` chain contains VkDeviceGroupRenderPassBeginInfo, `offset.y` + `extent.height` of each element of `pDeviceRenderAreas` **must** be less than or equal to VkFramebufferCreateInfo::`height` the `framebuffer` was created with
- VUID-VkRenderPassBeginInfo-framebuffer-03207
  If `framebuffer` was created with a VkFramebufferCreateInfo::`flags` value that did not include `VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT`, and the `pNext` chain includes a VkRenderPassAttachmentBeginInfo structure, its `attachmentCount` **must** be zero
- VUID-VkRenderPassBeginInfo-framebuffer-03208
  If `framebuffer` was created with a VkFramebufferCreateInfo::`flags` value that included `VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT`, the `attachmentCount` of a VkRenderPassAttachmentBeginInfo structure included in the `pNext` chain **must** be equal to the value of VkFramebufferAttachmentsCreateInfo::`attachmentImageInfoCount` used to create `framebuffer`
- VUID-VkRenderPassBeginInfo-framebuffer-02780
  If `framebuffer` was created with a VkFramebufferCreateInfo::`flags` value that included `VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT`, each element of the `pAttachments` member of a VkRenderPassAttachmentBeginInfo structure included in the `pNext` chain **must** have been

created on the same VkDevice as framebuffer and renderPass

- VUID-VkRenderPassBeginInfo-framebuffer-03209
  If framebuffer was created with a VkFramebufferCreateInfo::flags value that included VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT, each element of the pAttachments member of a VkRenderPassAttachmentBeginInfo structure included in the pNext chain **must** be a VkImageView of an image created with a value of VkImageCreateInfo::flags equal to the flags member of the corresponding element of VkFramebufferAttachmentsCreateInfo::pAttachmentImageInfos used to create framebuffer
- VUID-VkRenderPassBeginInfo-framebuffer-04627
  If framebuffer was created with a VkFramebufferCreateInfo::flags value that included VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT, each element of the pAttachments member of a VkRenderPassAttachmentBeginInfo structure included in the pNext chain **must** be a VkImageView with an inherited usage equal to the usage member of the corresponding element of VkFramebufferAttachmentsCreateInfo::pAttachmentImageInfos used to create framebuffer
- VUID-VkRenderPassBeginInfo-framebuffer-03211
  If framebuffer was created with a VkFramebufferCreateInfo::flags value that included VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT, each element of the pAttachments member of a VkRenderPassAttachmentBeginInfo structure included in the pNext chain **must** be a VkImageView with a width equal to the width member of the corresponding element of VkFramebufferAttachmentsCreateInfo::pAttachmentImageInfos used to create framebuffer
- VUID-VkRenderPassBeginInfo-framebuffer-03212
  If framebuffer was created with a VkFramebufferCreateInfo::flags value that included VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT, each element of the pAttachments member of a VkRenderPassAttachmentBeginInfo structure included in the pNext chain **must** be a VkImageView with a height equal to the height member of the corresponding element of VkFramebufferAttachmentsCreateInfo::pAttachmentImageInfos used to create framebuffer
- VUID-VkRenderPassBeginInfo-framebuffer-03213
  If framebuffer was created with a VkFramebufferCreateInfo::flags value that included VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT, each element of the pAttachments member of a VkRenderPassAttachmentBeginInfo structure included in the pNext chain **must** be a VkImageView of an image created with a value of VkImageViewCreateInfo::subresourceRange.layerCount equal to the layerCount member of the corresponding element of VkFramebufferAttachmentsCreateInfo::pAttachmentImageInfos used to create framebuffer
- VUID-VkRenderPassBeginInfo-framebuffer-03214
  If framebuffer was created with a VkFramebufferCreateInfo::flags value that included VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT, each element of the pAttachments member of a VkRenderPassAttachmentBeginInfo structure included in the pNext chain **must** be a VkImageView of an image created with a value of VkImageFormatListCreateInfo::viewFormatCount equal to the viewFormatCount member of the corresponding element of VkFramebufferAttachmentsCreateInfo::pAttachmentImageInfos used to create framebuffer
- VUID-VkRenderPassBeginInfo-framebuffer-03215
  If framebuffer was created with a VkFramebufferCreateInfo::flags value that included VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT, each element of the pAttachments member of a

VkRenderPassAttachmentBeginInfo structure included in the pNext chain **must** be a VkImageView of an image created with a set of elements in VkImageFormatListCreateInfo::pViewFormats equal to the set of elements in the pViewFormats member of the corresponding element of VkFramebufferAttachmentsCreateInfo::pAttachmentImageInfos used to create framebuffer

- VUID-VkRenderPassBeginInfo-framebuffer-03216
  If framebuffer was created with a VkFramebufferCreateInfo::flags value that included VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT, each element of the pAttachments member of a VkRenderPassAttachmentBeginInfo structure included in the pNext chain **must** be a VkImageView of an image created with a value of VkImageViewCreateInfo::format equal to the corresponding value of VkAttachmentDescription::format in renderPass
- VUID-VkRenderPassBeginInfo-framebuffer-09353
  If framebuffer was created with a VkFramebufferCreateInfo::flags value that included VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT, and the nullColorAttachmentWithExternalFormatResolve is VK_FALSE, the format of the color attachment for each subpass that includes an external format image as a resolve attachment **must** have a format equal to the value of VkAndroidHardwareBufferFormatResolvePropertiesANDROID::colorAttachmentFormat as returned by a call to vkGetAndroidHardwareBufferPropertiesANDROID for the Android hardware buffer that was used to create the image view use as its resolve attachment
- VUID-VkRenderPassBeginInfo-framebuffer-09354
  If framebuffer was created with a VkFramebufferCreateInfo::flags value that included VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT, each element of the pAttachments member of a VkRenderPassAttachmentBeginInfo structure included in the pNext chain **must** be a VkImageView of an image created with a value of VkExternalFormatANDROID::externalFormat equal to VkExternalFormatANDROID::externalFormat in the pNext chain of the corresponding VkAttachmentDescription2 structure used to create renderPass
- VUID-VkRenderPassBeginInfo-framebuffer-09047
  If framebuffer was created with a VkFramebufferCreateInfo::flags value that included VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT, each element of the pAttachments member of a VkRenderPassAttachmentBeginInfo structure included in the pNext chain **must** be a VkImageView of an image created with a value of VkImageCreateInfo::samples equal to the corresponding value of VkAttachmentDescription::samples in renderPass , or VK_SAMPLE_COUNT_1_BIT if renderPass was created with VkMultisampledRenderToSingleSampledInfoEXT structure in the pNext chain with multisampledRenderToSingleSampledEnable equal to VK_TRUE
- VUID-VkRenderPassBeginInfo-pNext-02869
  If the pNext chain includes VkRenderPassTransformBeginInfoQCOM, renderArea.offset **must** equal (0,0)
- VUID-VkRenderPassBeginInfo-pNext-02870
  If the pNext chain includes VkRenderPassTransformBeginInfoQCOM, renderArea.extent transformed by VkRenderPassTransformBeginInfoQCOM::transform **must** equal the framebuffer dimensions
- VUID-VkRenderPassBeginInfo-perViewRenderAreaCount-07859
  If the perViewRenderAreaCount member of a

VkMultiviewPerViewRenderAreasRenderPassBeginInfoQCOM structure included in the pNext chain is not 0, then the multiviewPerViewRenderAreas feature **must** be enabled

- VUID-VkRenderPassBeginInfo-perViewRenderAreaCount-07860
  If the perViewRenderAreaCount member of a VkMultiviewPerViewRenderAreasRenderPassBeginInfoQCOM structure included in the pNext chain is not 0, then renderArea **must** specify a render area that includes the union of all per view render areas
- VUID-VkRenderPassBeginInfo-pNext-09539
  If the pNext chain contains a VkRenderPassStripeBeginInfoARM structure, the union of stripe areas defined by the elements of VkRenderPassStripeInfoARM::pStripeInfos **must** cover the renderArea

### Valid Usage (Implicit)

- VUID-VkRenderPassBeginInfo-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_RENDER_PASS_BEGIN_INFO
- VUID-VkRenderPassBeginInfo-pNext-pNext
  Each pNext member of any structure (including this one) in the pNext chain **must** be either NULL or a pointer to a valid instance of VkDeviceGroupRenderPassBeginInfo, VkMultiviewPerViewRenderAreasRenderPassBeginInfoQCOM, VkRenderPassAttachmentBeginInfo, VkRenderPassSampleLocationsBeginInfoEXT, VkRenderPassStripeBeginInfoARM, or VkRenderPassTransformBeginInfoQCOM
- VUID-VkRenderPassBeginInfo-sType-unique
  The sType value of each struct in the pNext chain **must** be unique
- VUID-VkRenderPassBeginInfo-renderPass-parameter
  renderPass **must** be a valid VkRenderPass handle
- VUID-VkRenderPassBeginInfo-framebuffer-parameter
  framebuffer **must** be a valid VkFramebuffer handle
- VUID-VkRenderPassBeginInfo-commonparent
  Both of framebuffer, and renderPass **must** have been created, allocated, or retrieved from the same VkDevice

The image layout of the depth aspect of a depth/stencil attachment referring to an image created with VK_IMAGE_CREATE_SAMPLE_LOCATIONS_COMPATIBLE_DEPTH_BIT_EXT is dependent on the last sample locations used to render to the image subresource, thus preserving the contents of such depth/stencil attachments across subpass boundaries requires the application to specify these sample locations whenever a layout transition of the attachment **may** occur. This information **can** be provided by adding a VkRenderPassSampleLocationsBeginInfoEXT structure to the pNext chain of VkRenderPassBeginInfo.

The VkRenderPassSampleLocationsBeginInfoEXT structure is defined as:

```
// Provided by VK_EXT_sample_locations
typedef struct VkRenderPassSampleLocationsBeginInfoEXT {
    VkStructureType                             sType;
    const void*                                 pNext;
    uint32_t                                    attachmentInitialSampleLocationsCount;
    const VkAttachmentSampleLocationsEXT*       pAttachmentInitialSampleLocations;
    uint32_t                                    postSubpassSampleLocationsCount;
    const VkSubpassSampleLocationsEXT*          pPostSubpassSampleLocations;
} VkRenderPassSampleLocationsBeginInfoEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `attachmentInitialSampleLocationsCount` is the number of elements in the `pAttachmentInitialSampleLocations` array.
- `pAttachmentInitialSampleLocations` is a pointer to an array of `attachmentInitialSampleLocationsCount` VkAttachmentSampleLocationsEXT structures specifying the attachment indices and their corresponding sample location state. Each element of `pAttachmentInitialSampleLocations` **can** specify the sample location state to use in the automatic layout transition performed to transition a depth/stencil attachment from the initial layout of the attachment to the image layout specified for the attachment in the first subpass using it.
- `postSubpassSampleLocationsCount` is the number of elements in the `pPostSubpassSampleLocations` array.
- `pPostSubpassSampleLocations` is a pointer to an array of `postSubpassSampleLocationsCount` VkSubpassSampleLocationsEXT structures specifying the subpass indices and their corresponding sample location state. Each element of `pPostSubpassSampleLocations` **can** specify the sample location state to use in the automatic layout transition performed to transition the depth/stencil attachment used by the specified subpass to the image layout specified in a dependent subpass or to the final layout of the attachment in case the specified subpass is the last subpass using that attachment. In addition, if VkPhysicalDeviceSampleLocationsPropertiesEXT::`variableSampleLocations` is `VK_FALSE`, each element of `pPostSubpassSampleLocations` **must** specify the sample location state that matches the sample locations used by all pipelines that will be bound to a command buffer during the specified subpass. If `variableSampleLocations` is `VK_TRUE`, the sample locations used for rasterization do not depend on `pPostSubpassSampleLocations`.

## Valid Usage (Implicit)

- VUID-VkRenderPassSampleLocationsBeginInfoEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_RENDER_PASS_SAMPLE_LOCATIONS_BEGIN_INFO_EXT`
- VUID-VkRenderPassSampleLocationsBeginInfoEXT-pAttachmentInitialSampleLocations-parameter
  If `attachmentInitialSampleLocationsCount` is not `0`, `pAttachmentInitialSampleLocations` **must** be a valid pointer to an array of `attachmentInitialSampleLocationsCount` valid

VkAttachmentSampleLocationsEXT structures

- VUID-VkRenderPassSampleLocationsBeginInfoEXT-pPostSubpassSampleLocations-parameter
  If postSubpassSampleLocationsCount is not 0, pPostSubpassSampleLocations **must** be a valid pointer to an array of postSubpassSampleLocationsCount valid VkSubpassSampleLocationsEXT structures

The VkAttachmentSampleLocationsEXT structure is defined as:

```
// Provided by VK_EXT_sample_locations
typedef struct VkAttachmentSampleLocationsEXT {
    uint32_t                    attachmentIndex;
    VkSampleLocationsInfoEXT    sampleLocationsInfo;
} VkAttachmentSampleLocationsEXT;
```

- attachmentIndex is the index of the attachment for which the sample locations state is provided.
- sampleLocationsInfo is the sample locations state to use for the layout transition of the given attachment from the initial layout of the attachment to the image layout specified for the attachment in the first subpass using it.

If the image referenced by the framebuffer attachment at index attachmentIndex was not created with VK_IMAGE_CREATE_SAMPLE_LOCATIONS_COMPATIBLE_DEPTH_BIT_EXT then the values specified in sampleLocationsInfo are ignored.

### Valid Usage

- VUID-VkAttachmentSampleLocationsEXT-attachmentIndex-01531
  attachmentIndex **must** be less than the attachmentCount specified in VkRenderPassCreateInfo the render pass specified by VkRenderPassBeginInfo::renderPass was created with

### Valid Usage (Implicit)

- VUID-VkAttachmentSampleLocationsEXT-sampleLocationsInfo-parameter
  sampleLocationsInfo **must** be a valid VkSampleLocationsInfoEXT structure

The VkSubpassSampleLocationsEXT structure is defined as:

```
// Provided by VK_EXT_sample_locations
typedef struct VkSubpassSampleLocationsEXT {
    uint32_t                    subpassIndex;
    VkSampleLocationsInfoEXT    sampleLocationsInfo;
} VkSubpassSampleLocationsEXT;
```

- subpassIndex is the index of the subpass for which the sample locations state is provided.
- sampleLocationsInfo is the sample locations state to use for the layout transition of the depth/stencil attachment away from the image layout the attachment is used with in the subpass specified in subpassIndex.

If the image referenced by the depth/stencil attachment used in the subpass identified by subpassIndex was not created with VK_IMAGE_CREATE_SAMPLE_LOCATIONS_COMPATIBLE_DEPTH_BIT_EXT or if the subpass does not use a depth/stencil attachment, and VkPhysicalDeviceSampleLocationsPropertiesEXT::variableSampleLocations is VK_TRUE then the values specified in sampleLocationsInfo are ignored.

### Valid Usage

- VUID-VkSubpassSampleLocationsEXT-subpassIndex-01532
  subpassIndex **must** be less than the subpassCount specified in VkRenderPassCreateInfo the render pass specified by VkRenderPassBeginInfo::renderPass was created with

### Valid Usage (Implicit)

- VUID-VkSubpassSampleLocationsEXT-sampleLocationsInfo-parameter
  sampleLocationsInfo **must** be a valid VkSampleLocationsInfoEXT structure

To begin a render pass instance with render pass transform enabled, add the VkRenderPassTransformBeginInfoQCOM to the pNext chain of VkRenderPassBeginInfo structure passed to the vkCmdBeginRenderPass command specifying the render pass transform.

The VkRenderPassTransformBeginInfoQCOM structure is defined as:

```
// Provided by VK_QCOM_render_pass_transform
typedef struct VkRenderPassTransformBeginInfoQCOM {
    VkStructureType                 sType;
    void*                           pNext;
    VkSurfaceTransformFlagBitsKHR   transform;
} VkRenderPassTransformBeginInfoQCOM;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- transform is a VkSurfaceTransformFlagBitsKHR value describing the transform to be applied to rasterization.

### Valid Usage

- VUID-VkRenderPassTransformBeginInfoQCOM-transform-02871
  transform **must** be VK_SURFACE_TRANSFORM_IDENTITY_BIT_KHR,

VK_SURFACE_TRANSFORM_ROTATE_90_BIT_KHR, VK_SURFACE_TRANSFORM_ROTATE_180_BIT_KHR, or VK_SURFACE_TRANSFORM_ROTATE_270_BIT_KHR

- VUID-VkRenderPassTransformBeginInfoQCOM-flags-02872
  The renderpass **must** have been created with VkRenderPassCreateInfo::flags containing VK_RENDER_PASS_CREATE_TRANSFORM_BIT_QCOM

### Valid Usage (Implicit)

- VUID-VkRenderPassTransformBeginInfoQCOM-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_RENDER_PASS_TRANSFORM_BEGIN_INFO_QCOM

The VkSubpassBeginInfo structure is defined as:

```
typedef struct VkSubpassBeginInfo {
    VkStructureType      sType;
    const void*          pNext;
    VkSubpassContents    contents;
} VkSubpassBeginInfo;
```

or the equivalent

```
// Provided by VK_KHR_create_renderpass2
typedef VkSubpassBeginInfo VkSubpassBeginInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- contents is a VkSubpassContents value specifying how the commands in the next subpass will be provided.

### Valid Usage

- VUID-VkSubpassBeginInfo-contents-09382
  If contents is VK_SUBPASS_CONTENTS_INLINE_AND_SECONDARY_COMMAND_BUFFERS_KHR, then at least one of the following features **must** be enabled:
  - maintenance7
  - nestedCommandBuffer

### Valid Usage (Implicit)

- VUID-VkSubpassBeginInfo-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_SUBPASS_BEGIN_INFO

- VUID-VkSubpassBeginInfo-pNext-pNext
  `pNext` **must** be `NULL`
- VUID-VkSubpassBeginInfo-contents-parameter
  `contents` **must** be a valid VkSubpassContents value

Possible values of vkCmdBeginRenderPass::`contents`, specifying how the commands in the first subpass will be provided, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkSubpassContents {
    VK_SUBPASS_CONTENTS_INLINE = 0,
    VK_SUBPASS_CONTENTS_SECONDARY_COMMAND_BUFFERS = 1,
  // Provided by VK_KHR_maintenance7
    VK_SUBPASS_CONTENTS_INLINE_AND_SECONDARY_COMMAND_BUFFERS_KHR = 1000451000,
  // Provided by VK_EXT_nested_command_buffer
    VK_SUBPASS_CONTENTS_INLINE_AND_SECONDARY_COMMAND_BUFFERS_EXT =
VK_SUBPASS_CONTENTS_INLINE_AND_SECONDARY_COMMAND_BUFFERS_KHR,
} VkSubpassContents;
```

- `VK_SUBPASS_CONTENTS_INLINE` specifies that the contents of the subpass will be recorded inline in the primary command buffer, and secondary command buffers **must** not be executed within the subpass.
- `VK_SUBPASS_CONTENTS_SECONDARY_COMMAND_BUFFERS` specifies that the contents are recorded in secondary command buffers that will be called from the primary command buffer, and vkCmdExecuteCommands is the only valid command in the command buffer until vkCmdNextSubpass or vkCmdEndRenderPass.
- `VK_SUBPASS_CONTENTS_INLINE_AND_SECONDARY_COMMAND_BUFFERS_KHR` specifies that the contents of the subpass **can** be recorded both inline and in secondary command buffers executed from this command buffer with vkCmdExecuteCommands.

If the `pNext` chain of VkRenderPassBeginInfo or VkRenderingInfo includes a `VkDeviceGroupRenderPassBeginInfo` structure, then that structure includes a device mask and set of render areas for the render pass instance.

The `VkDeviceGroupRenderPassBeginInfo` structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkDeviceGroupRenderPassBeginInfo {
    VkStructureType    sType;
    const void*        pNext;
    uint32_t           deviceMask;
    uint32_t           deviceRenderAreaCount;
    const VkRect2D*    pDeviceRenderAreas;
} VkDeviceGroupRenderPassBeginInfo;
```

or the equivalent

```
// Provided by VK_KHR_device_group
typedef VkDeviceGroupRenderPassBeginInfo VkDeviceGroupRenderPassBeginInfoKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `deviceMask` is the device mask for the render pass instance.
- `deviceRenderAreaCount` is the number of elements in the `pDeviceRenderAreas` array.
- `pDeviceRenderAreas` is a pointer to an array of VkRect2D structures defining the render area for each physical device.

The `deviceMask` serves several purposes. It is an upper bound on the set of physical devices that **can** be used during the render pass instance, and the initial device mask when the render pass instance begins. In addition, commands transitioning to the next subpass in a render pass instance and commands ending the render pass instance, and, accordingly render pass load, store, and multisample resolve operations and subpass dependencies corresponding to the render pass instance, are executed on the physical devices included in the device mask provided here.

If `deviceRenderAreaCount` is not zero, then the elements of `pDeviceRenderAreas` override the value of VkRenderPassBeginInfo::`renderArea`, and provide a render area specific to each physical device. These render areas serve the same purpose as VkRenderPassBeginInfo::`renderArea`, including controlling the region of attachments that are cleared by `VK_ATTACHMENT_LOAD_OP_CLEAR` and that are resolved into resolve attachments.

If this structure is not present, the render pass instance's device mask is the value of VkDeviceGroupCommandBufferBeginInfo::`deviceMask`. If this structure is not present or if `deviceRenderAreaCount` is zero, VkRenderPassBeginInfo::`renderArea` is used for all physical devices.

### Valid Usage

- VUID-VkDeviceGroupRenderPassBeginInfo-deviceMask-00905
  `deviceMask` **must** be a valid device mask value
- VUID-VkDeviceGroupRenderPassBeginInfo-deviceMask-00906
  `deviceMask` **must** not be zero
- VUID-VkDeviceGroupRenderPassBeginInfo-deviceMask-00907
  `deviceMask` **must** be a subset of the command buffer's initial device mask
- VUID-VkDeviceGroupRenderPassBeginInfo-deviceRenderAreaCount-00908
  `deviceRenderAreaCount` **must** either be zero or equal to the number of physical devices in the logical device
- VUID-VkDeviceGroupRenderPassBeginInfo-offset-06166
  The `offset.x` member of any element of `pDeviceRenderAreas` **must** be greater than or equal to 0
- VUID-VkDeviceGroupRenderPassBeginInfo-offset-06167
  The `offset.y` member of any element of `pDeviceRenderAreas` **must** be greater than or equal to 0

- VUID-VkDeviceGroupRenderPassBeginInfo-offset-06168
  The sum of the `offset.x` and `extent.width` members of any element of `pDeviceRenderAreas` **must** be less than or equal to `maxFramebufferWidth`
- VUID-VkDeviceGroupRenderPassBeginInfo-offset-06169
  The sum of the `offset.y` and `extent.height` members of any element of `pDeviceRenderAreas` **must** be less than or equal to `maxFramebufferHeight`
- VUID-VkDeviceGroupRenderPassBeginInfo-extent-08998
  The `extent.width` member of any element of `pDeviceRenderAreas` **must** be greater than 0
- VUID-VkDeviceGroupRenderPassBeginInfo-extent-08999
  The `extent.height` member of any element of `pDeviceRenderAreas` **must** be greater than 0

### Valid Usage (Implicit)

- VUID-VkDeviceGroupRenderPassBeginInfo-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_DEVICE_GROUP_RENDER_PASS_BEGIN_INFO`
- VUID-VkDeviceGroupRenderPassBeginInfo-pDeviceRenderAreas-parameter
  If `deviceRenderAreaCount` is not `0`, `pDeviceRenderAreas` **must** be a valid pointer to an array of `deviceRenderAreaCount` VkRect2D structures

The `VkRenderPassAttachmentBeginInfo` structure is defined as:

```
typedef struct VkRenderPassAttachmentBeginInfo {
    VkStructureType       sType;
    const void*           pNext;
    uint32_t              attachmentCount;
    const VkImageView*    pAttachments;
} VkRenderPassAttachmentBeginInfo;
```

or the equivalent

```
// Provided by VK_KHR_imageless_framebuffer
typedef VkRenderPassAttachmentBeginInfo VkRenderPassAttachmentBeginInfoKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `attachmentCount` is the number of attachments.
- `pAttachments` is a pointer to an array of `VkImageView` handles, each of which will be used as the corresponding attachment in the render pass instance.

### Valid Usage

- VUID-VkRenderPassAttachmentBeginInfo-pAttachments-03218

Each element of pAttachments **must** only specify a single mip level

- VUID-VkRenderPassAttachmentBeginInfo-pAttachments-03219
  Each element of pAttachments **must** have been created with the identity swizzle
- VUID-VkRenderPassAttachmentBeginInfo-pAttachments-04114
  Each element of pAttachments **must** have been created with VkImageViewCreateInfo::viewType not equal to VK_IMAGE_VIEW_TYPE_3D
- VUID-VkRenderPassAttachmentBeginInfo-pAttachments-07010
  If multisampled-render-to-single-sampled is enabled for any subpass, all element of pAttachments which have a sample count equal to VK_SAMPLE_COUNT_1_BIT **must** have a format that supports the sample count specified in VkMultisampledRenderToSingleSampledInfoEXT::rasterizationSamples

### Valid Usage (Implicit)

- VUID-VkRenderPassAttachmentBeginInfo-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_RENDER_PASS_ATTACHMENT_BEGIN_INFO
- VUID-VkRenderPassAttachmentBeginInfo-pAttachments-parameter
  If attachmentCount is not 0, pAttachments **must** be a valid pointer to an array of attachmentCount valid VkImageView handles

If a render pass instance enables multiview and if the multiviewPerViewRenderAreas feature is enabled, the VkMultiviewPerViewRenderAreasRenderPassBeginInfoQCOM structure **can** be included in the pNext chain of VkRenderPassBeginInfo or VkRenderingInfo

The VkMultiviewPerViewRenderAreasRenderPassBeginInfoQCOM structure is defined as:

```
// Provided by VK_QCOM_multiview_per_view_render_areas
typedef struct VkMultiviewPerViewRenderAreasRenderPassBeginInfoQCOM {
    VkStructureType    sType;
    const void*        pNext;
    uint32_t           perViewRenderAreaCount;
    const VkRect2D*    pPerViewRenderAreas;
} VkMultiviewPerViewRenderAreasRenderPassBeginInfoQCOM;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- perViewRenderAreaCount is the number of elements in the pPerViewRenderAreas array.
- pPerViewRenderAreas is a pointer to an array of VkRect2D structures defining the render area for each view.

If perViewRenderAreaCount is not zero, then the elements of pPerViewRenderAreas override the value of VkRenderPassBeginInfo::renderArea or VkRenderingInfo::renderArea and define per-view render areas for the individual views of a multiview render pass. The render area for the view with *view*

*index* `i` is specified by `pPerViewRenderAreas`[i].

The per-view render areas define per-view regions of attachments that are loaded, stored, and resolved according to the `loadOp`, `storeOp`, and `resolveMode` values of the render pass instance. When per-view render areas are defined, the value of VkRenderPassBeginInfo::`renderArea` or VkRenderingInfo::`renderArea` **must** be set to a render area that includes the union of all per-view render areas, **may** be used by the implementation for optimizations, but does not affect loads, stores, or resolves.

If this structure is present and if `perViewRenderAreaCount` is not zero, then `perViewRenderAreaCount` **must** be at least one greater than the most significant bit set in any element of VkRenderPassMultiviewCreateInfo::`pViewMasks`. or VkRenderingInfo::`viewMask`

If this structure is not present or if `perViewRenderAreaCount` is zero, VkRenderPassBeginInfo ::`renderArea` or VkRenderingInfo::`renderArea` is used for all views.

## Valid Usage

- VUID-VkMultiviewPerViewRenderAreasRenderPassBeginInfoQCOM-offset-07861
  The `offset.x` member of any element of `pPerViewRenderAreas` **must** be greater than or equal to 0
- VUID-VkMultiviewPerViewRenderAreasRenderPassBeginInfoQCOM-offset-07862
  The `offset.y` member of any element of `pPerViewRenderAreas` **must** be greater than or equal to 0
- VUID-VkMultiviewPerViewRenderAreasRenderPassBeginInfoQCOM-offset-07863
  The sum of the `offset.x` and `extent.width` members of any element of `pPerViewRenderAreas` **must** be less than or equal to `maxFramebufferWidth`
- VUID-VkMultiviewPerViewRenderAreasRenderPassBeginInfoQCOM-offset-07864
  The sum of the `offset.y` and `extent.height` members of any element of `pPerViewRenderAreas` **must** be less than or equal to `maxFramebufferHeight`
- VUID-VkMultiviewPerViewRenderAreasRenderPassBeginInfoQCOM-pNext-07865
  If this structure is in the `pNext` chain of VkRenderPassBeginInfo and if the render pass object included an element in VkRenderPassMultiviewCreateInfo::`pViewMasks` that set bit `n`, then `perViewRenderAreaCount` **must** be at least equal to `n+1`
- VUID-VkMultiviewPerViewRenderAreasRenderPassBeginInfoQCOM-pNext-07866
  If this structure is in the `pNext` chain of VkRenderingInfo and if VkRenderingInfo::`viewMask` set bit `n`, then `perViewRenderAreaCount` **must** be at least equal to `n+1`

## Valid Usage (Implicit)

- VUID-VkMultiviewPerViewRenderAreasRenderPassBeginInfoQCOM-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_MULTIVIEW_PER_VIEW_RENDER_AREAS_RENDER_PASS_BEGIN_INFO_QCOM`
- VUID-VkMultiviewPerViewRenderAreasRenderPassBeginInfoQCOM-pPerViewRenderAreas-parameter

If perViewRenderAreaCount is not 0, pPerViewRenderAreas **must** be a valid pointer to an array of perViewRenderAreaCount VkRect2D structures

To query the render area granularity, call:

```
// Provided by VK_VERSION_1_0
void vkGetRenderAreaGranularity(
    VkDevice                                    device,
    VkRenderPass                                renderPass,
    VkExtent2D*                                 pGranularity);
```

- device is the logical device that owns the render pass.
- renderPass is a handle to a render pass.
- pGranularity is a pointer to a VkExtent2D structure in which the granularity is returned.

The conditions leading to an optimal renderArea are:

- the offset.x member in renderArea is a multiple of the width member of the returned VkExtent2D (the horizontal granularity).
- the offset.y member in renderArea is a multiple of the height member of the returned VkExtent2D (the vertical granularity).
- either the extent.width member in renderArea is a multiple of the horizontal granularity or offset.x+extent.width is equal to the width of the framebuffer in the VkRenderPassBeginInfo.
- either the extent.height member in renderArea is a multiple of the vertical granularity or offset.y+extent.height is equal to the height of the framebuffer in the VkRenderPassBeginInfo.

Subpass dependencies are not affected by the render area, and apply to the entire image subresources attached to the framebuffer as specified in the description of automatic layout transitions. Similarly, pipeline barriers are valid even if their effect extends outside the render area.

### Valid Usage (Implicit)

- VUID-vkGetRenderAreaGranularity-device-parameter
  device **must** be a valid VkDevice handle
- VUID-vkGetRenderAreaGranularity-renderPass-parameter
  renderPass **must** be a valid VkRenderPass handle
- VUID-vkGetRenderAreaGranularity-pGranularity-parameter
  pGranularity **must** be a valid pointer to a VkExtent2D structure
- VUID-vkGetRenderAreaGranularity-renderPass-parent
  renderPass **must** have been created, allocated, or retrieved from device

To transition to the next subpass in the render pass instance after recording the commands for a

subpass, call:

```
// Provided by VK_VERSION_1_0
void vkCmdNextSubpass(
    VkCommandBuffer                             commandBuffer,
    VkSubpassContents                           contents);
```

- `commandBuffer` is the command buffer in which to record the command.
- `contents` specifies how the commands in the next subpass will be provided, in the same fashion as the corresponding parameter of vkCmdBeginRenderPass.

The subpass index for a render pass begins at zero when `vkCmdBeginRenderPass` is recorded, and increments each time `vkCmdNextSubpass` is recorded.

After transitioning to the next subpass, the application **can** record the commands for that subpass.

### Valid Usage

- VUID-vkCmdNextSubpass-None-00909
  The current subpass index **must** be less than the number of subpasses in the render pass minus one
- VUID-vkCmdNextSubpass-None-02349
  This command **must** not be recorded when transform feedback is active

### Valid Usage (Implicit)

- VUID-vkCmdNextSubpass-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdNextSubpass-contents-parameter
  `contents` **must** be a valid VkSubpassContents value
- VUID-vkCmdNextSubpass-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdNextSubpass-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- VUID-vkCmdNextSubpass-renderpass
  This command **must** only be called inside of a render pass instance
- VUID-vkCmdNextSubpass-videocoding
  This command **must** only be called outside of a video coding scope
- VUID-vkCmdNextSubpass-bufferlevel
  `commandBuffer` **must** be a primary `VkCommandBuffer`

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary | Inside | Outside | Graphics | Action<br>State<br>Synchronization |

To transition to the next subpass in the render pass instance after recording the commands for a subpass, call:

```
// Provided by VK_KHR_create_renderpass2
void vkCmdNextSubpass2KHR(
    VkCommandBuffer                             commandBuffer,
    const VkSubpassBeginInfo*                   pSubpassBeginInfo,
    const VkSubpassEndInfo*                     pSubpassEndInfo);
```

- `commandBuffer` is the command buffer in which to record the command.
- `pSubpassBeginInfo` is a pointer to a VkSubpassBeginInfo structure containing information about the subpass which is about to begin rendering.
- `pSubpassEndInfo` is a pointer to a VkSubpassEndInfo structure containing information about how the previous subpass will be ended.

`vkCmdNextSubpass2KHR` is semantically identical to vkCmdNextSubpass, except that it is extensible, and that `contents` is provided as part of an extensible structure instead of as a flat parameter.

## Valid Usage

- VUID-vkCmdNextSubpass2-None-03102
  The current subpass index **must** be less than the number of subpasses in the render pass minus one
- VUID-vkCmdNextSubpass2-None-02350
  This command **must** not be recorded when transform feedback is active

## Valid Usage (Implicit)

- VUID-vkCmdNextSubpass2-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdNextSubpass2-pSubpassBeginInfo-parameter
  `pSubpassBeginInfo` **must** be a valid pointer to a valid VkSubpassBeginInfo structure
- VUID-vkCmdNextSubpass2-pSubpassEndInfo-parameter
  `pSubpassEndInfo` **must** be a valid pointer to a valid VkSubpassEndInfo structure
- VUID-vkCmdNextSubpass2-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdNextSubpass2-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- VUID-vkCmdNextSubpass2-renderpass
  This command **must** only be called inside of a render pass instance
- VUID-vkCmdNextSubpass2-videocoding
  This command **must** only be called outside of a video coding scope
- VUID-vkCmdNextSubpass2-bufferlevel
  `commandBuffer` **must** be a primary `VkCommandBuffer`

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary | Inside | Outside | Graphics | Action<br>State<br>Synchronization |

To record a command to end a render pass instance after recording the commands for the last subpass, call:

```
// Provided by VK_VERSION_1_0
void vkCmdEndRenderPass(
    VkCommandBuffer                             commandBuffer);
```

- `commandBuffer` is the command buffer in which to end the current render pass instance.

Ending a render pass instance performs any multisample resolve operations on the final subpass.

### Valid Usage

- VUID-vkCmdEndRenderPass-None-00910
  The current subpass index **must** be equal to the number of subpasses in the render pass minus one
- VUID-vkCmdEndRenderPass-None-02351
  This command **must** not be recorded when transform feedback is active
- VUID-vkCmdEndRenderPass-None-06170
  The current render pass instance **must** not have been begun with vkCmdBeginRenderingKHR
- VUID-vkCmdEndRenderPass-None-07004
  If `vkCmdBeginQuery*` was called within a subpass of the render pass, the corresponding `vkCmdEndQuery*` **must** have been called subsequently within the same subpass

### Valid Usage (Implicit)

- VUID-vkCmdEndRenderPass-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdEndRenderPass-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdEndRenderPass-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- VUID-vkCmdEndRenderPass-renderpass
  This command **must** only be called inside of a render pass instance
- VUID-vkCmdEndRenderPass-videocoding
  This command **must** only be called outside of a video coding scope
- VUID-vkCmdEndRenderPass-bufferlevel
  `commandBuffer` **must** be a primary `VkCommandBuffer`

### Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized

- Host access to the VkCommandPool that commandBuffer was allocated from **must** be externally synchronized

### Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary | Inside | Outside | Graphics | Action<br>State<br>Synchronization |

To record a command to end a render pass instance after recording the commands for the last subpass, call:

```
// Provided by VK_KHR_create_renderpass2
void vkCmdEndRenderPass2KHR(
    VkCommandBuffer                             commandBuffer,
    const VkSubpassEndInfo*                     pSubpassEndInfo);
```

- commandBuffer is the command buffer in which to end the current render pass instance.
- pSubpassEndInfo is a pointer to a VkSubpassEndInfo structure containing information about how the last subpass will be ended.

vkCmdEndRenderPass2KHR is semantically identical to vkCmdEndRenderPass, except that it is extensible.

### Valid Usage

- VUID-vkCmdEndRenderPass2-None-03103
  The current subpass index **must** be equal to the number of subpasses in the render pass minus one
- VUID-vkCmdEndRenderPass2-None-02352
  This command **must** not be recorded when transform feedback is active
- VUID-vkCmdEndRenderPass2-None-06171
  The current render pass instance **must** not have been begun with vkCmdBeginRenderingKHR
- VUID-vkCmdEndRenderPass2-None-07005
  If vkCmdBeginQuery* was called within a subpass of the render pass, the corresponding vkCmdEndQuery* **must** have been called subsequently within the same subpass

## Valid Usage (Implicit)

- VUID-vkCmdEndRenderPass2-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdEndRenderPass2-pSubpassEndInfo-parameter
  `pSubpassEndInfo` **must** be a valid pointer to a valid VkSubpassEndInfo structure
- VUID-vkCmdEndRenderPass2-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdEndRenderPass2-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- VUID-vkCmdEndRenderPass2-renderpass
  This command **must** only be called inside of a render pass instance
- VUID-vkCmdEndRenderPass2-videocoding
  This command **must** only be called outside of a video coding scope
- VUID-vkCmdEndRenderPass2-bufferlevel
  `commandBuffer` **must** be a primary `VkCommandBuffer`

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary | Inside | Outside | Graphics | Action<br>State<br>Synchronization |

The `VkSubpassEndInfo` structure is defined as:

```
typedef struct VkSubpassEndInfo {
    VkStructureType    sType;
    const void*        pNext;
} VkSubpassEndInfo;
```

or the equivalent

```
// Provided by VK_KHR_create_renderpass2
typedef VkSubpassEndInfo VkSubpassEndInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.

### Valid Usage (Implicit)

- VUID-VkSubpassEndInfo-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_SUBPASS_END_INFO
- VUID-VkSubpassEndInfo-pNext-pNext
  pNext **must** be NULL or a pointer to a valid instance of VkSubpassFragmentDensityMapOffsetEndInfoQCOM
- VUID-VkSubpassEndInfo-sType-unique
  The sType value of each struct in the pNext chain **must** be unique

If the VkSubpassEndInfo::pNext chain includes a VkSubpassFragmentDensityMapOffsetEndInfoQCOM structure, then that structure includes an array of fragment density map offsets per layer for the render pass.

The VkSubpassFragmentDensityMapOffsetEndInfoQCOM structure is defined as:

```
// Provided by VK_QCOM_fragment_density_map_offset
typedef struct VkSubpassFragmentDensityMapOffsetEndInfoQCOM {
    VkStructureType      sType;
    const void*          pNext;
    uint32_t             fragmentDensityOffsetCount;
    const VkOffset2D*    pFragmentDensityOffsets;
} VkSubpassFragmentDensityMapOffsetEndInfoQCOM;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- fragmentDensityOffsetCount is the number of offsets being specified.
- pFragmentDensityOffsets is a pointer to an array of VkOffset2D structs, each of which describes the offset per layer.

The array elements are given per layer as defined by Fetch Density Value, where index = layer. Each (x,y) offset is in framebuffer pixels and shifts the fetch of the fragment density map by that amount. Offsets can be positive or negative.

Offset values specified for any subpass that is not the last subpass in the render pass are ignored. If the VkSubpassEndInfo::pNext chain for the last subpass of a render pass does not include VkSubpassFragmentDensityMapOffsetEndInfoQCOM, or if fragmentDensityOffsetCount is zero, then the offset (0,0) is used for Fetch Density Value.

## Valid Usage

- VUID-VkSubpassFragmentDensityMapOffsetEndInfoQCOM-fragmentDensityMapOffsets-06503
  If the `fragmentDensityMapOffset` feature is not enabled or fragment density map is not enabled in the render pass, `fragmentDensityOffsetCount` **must** equal `0`
- VUID-VkSubpassFragmentDensityMapOffsetEndInfoQCOM-fragmentDensityMapAttachment-06504
  If `VkSubpassDescription::fragmentDensityMapAttachment` is not is not `VK_ATTACHMENT_UNUSED` and was not created with `VK_IMAGE_CREATE_FRAGMENT_DENSITY_MAP_OFFSET_BIT_QCOM`, `fragmentDensityOffsetCount` **must** equal `0`
- VUID-VkSubpassFragmentDensityMapOffsetEndInfoQCOM-pDepthStencilAttachment-06505
  If `VkSubpassDescription::pDepthStencilAttachment` is not is not `VK_ATTACHMENT_UNUSED` and was not created with `VK_IMAGE_CREATE_FRAGMENT_DENSITY_MAP_OFFSET_BIT_QCOM`, `fragmentDensityOffsetCount` **must** equal `0`
- VUID-VkSubpassFragmentDensityMapOffsetEndInfoQCOM-pInputAttachments-06506
  If any element of `VkSubpassDescription::pInputAttachments` is not is not `VK_ATTACHMENT_UNUSED` and was not created with `VK_IMAGE_CREATE_FRAGMENT_DENSITY_MAP_OFFSET_BIT_QCOM`, `fragmentDensityOffsetCount` **must** equal `0`
- VUID-VkSubpassFragmentDensityMapOffsetEndInfoQCOM-pColorAttachments-06507
  If any element of `VkSubpassDescription::pColorAttachments` is not is not `VK_ATTACHMENT_UNUSED` and was not created with `VK_IMAGE_CREATE_FRAGMENT_DENSITY_MAP_OFFSET_BIT_QCOM`, `fragmentDensityOffsetCount` **must** equal `0`
- VUID-VkSubpassFragmentDensityMapOffsetEndInfoQCOM-pResolveAttachments-06508
  If any element of `VkSubpassDescription::pResolveAttachments` is not is not `VK_ATTACHMENT_UNUSED` and was not created with `VK_IMAGE_CREATE_FRAGMENT_DENSITY_MAP_OFFSET_BIT_QCOM`, `fragmentDensityOffsetCount` **must** equal `0`
- VUID-VkSubpassFragmentDensityMapOffsetEndInfoQCOM-pPreserveAttachments-06509
  If any element of `VkSubpassDescription::pPreserveAttachments` is not is not `VK_ATTACHMENT_UNUSED` and was not created with `VK_IMAGE_CREATE_FRAGMENT_DENSITY_MAP_OFFSET_BIT_QCOM`, `fragmentDensityOffsetCount` **must** equal `0`
- VUID-VkSubpassFragmentDensityMapOffsetEndInfoQCOM-fragmentDensityOffsetCount-06510
  If `fragmentDensityOffsetCount` is not `0` and multiview is enabled for the render pass, `fragmentDensityOffsetCount` **must** equal the `layerCount` that was specified in creating the fragment density map attachment view
- VUID-VkSubpassFragmentDensityMapOffsetEndInfoQCOM-fragmentDensityOffsetCount-06511
  If `fragmentDensityOffsetCount` is not `0` and multiview is not enabled for the render pass,

fragmentDensityOffsetCount **must** equal 1

- VUID-VkSubpassFragmentDensityMapOffsetEndInfoQCOM-x-06512
  The x component of each element of pFragmentDensityOffsets **must** be an integer multiple of fragmentDensityOffsetGranularity.width
- VUID-VkSubpassFragmentDensityMapOffsetEndInfoQCOM-y-06513
  The y component of each element of pFragmentDensityOffsets **must** be an integer multiple of fragmentDensityOffsetGranularity.height

**Valid Usage (Implicit)**

- VUID-VkSubpassFragmentDensityMapOffsetEndInfoQCOM-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_SUBPASS_FRAGMENT_DENSITY_MAP_OFFSET_END_INFO_QCOM
- VUID-VkSubpassFragmentDensityMapOffsetEndInfoQCOM-pFragmentDensityOffsets-parameter
  If fragmentDensityOffsetCount is not 0, pFragmentDensityOffsets **must** be a valid pointer to an array of fragmentDensityOffsetCount VkOffset2D structures

## 8.9. Render Pass Creation Feedback

A VkRenderPassCreationControlEXT structure **can** be included in the pNext chain of VkRenderPassCreateInfo2 or pNext chain of VkSubpassDescription2. The VkRenderPassCreationControlEXT structure is defined as:

```
// Provided by VK_EXT_subpass_merge_feedback
typedef struct VkRenderPassCreationControlEXT {
    VkStructureType    sType;
    const void*        pNext;
    VkBool32           disallowMerging;
} VkRenderPassCreationControlEXT;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- disallowMerging is a boolean value indicating whether subpass merging will be disabled.

If a VkRenderPassCreationControlEXT structure is included in the pNext chain of VkRenderPassCreateInfo2 and its value of disallowMerging is VK_TRUE, the implementation will disable subpass merging for the entire render pass. If a VkRenderPassCreationControlEXT structure is included in the pNext chain of VkSubpassDescription2 and its value of disallowMerging is VK_TRUE, the implementation will disable merging the described subpass with previous subpasses in the render pass.

### Valid Usage (Implicit)

- VUID-VkRenderPassCreationControlEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_RENDER_PASS_CREATION_CONTROL_EXT`

To obtain feedback about the creation of a render pass, include a `VkRenderPassCreationFeedbackCreateInfoEXT` structure in the `pNext` chain of VkRenderPassCreateInfo2. The `VkRenderPassCreationFeedbackCreateInfoEXT` structure is defined as:

```
// Provided by VK_EXT_subpass_merge_feedback
typedef struct VkRenderPassCreationFeedbackCreateInfoEXT {
    VkStructureType                         sType;
    const void*                             pNext;
    VkRenderPassCreationFeedbackInfoEXT*    pRenderPassFeedback;
} VkRenderPassCreationFeedbackCreateInfoEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `pRenderPassFeedback` is a pointer to a VkRenderPassCreationFeedbackInfoEXT structure in which feedback is returned.

### Valid Usage (Implicit)

- VUID-VkRenderPassCreationFeedbackCreateInfoEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_RENDER_PASS_CREATION_FEEDBACK_CREATE_INFO_EXT`
- VUID-VkRenderPassCreationFeedbackCreateInfoEXT-pRenderPassFeedback-parameter
  `pRenderPassFeedback` **must** be a valid pointer to a VkRenderPassCreationFeedbackInfoEXT structure

The `VkRenderPassCreationFeedbackInfoEXT` structure is defined as:

```
// Provided by VK_EXT_subpass_merge_feedback
typedef struct VkRenderPassCreationFeedbackInfoEXT {
    uint32_t    postMergeSubpassCount;
} VkRenderPassCreationFeedbackInfoEXT;
```

- `postMergeSubpassCount` is the subpass count after merge.

Feedback about the creation of a subpass **can** be obtained by including a `VkRenderPassSubpassFeedbackCreateInfoEXT` structure in the `pNext` chain of VkSubpassDescription2. `VkRenderPassSubpassFeedbackCreateInfoEXT` structure is defined as:

```
// Provided by VK_EXT_subpass_merge_feedback
typedef struct VkRenderPassSubpassFeedbackCreateInfoEXT {
    VkStructureType                          sType;
    const void*                              pNext;
    VkRenderPassSubpassFeedbackInfoEXT*      pSubpassFeedback;
} VkRenderPassSubpassFeedbackCreateInfoEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `pSubpassFeedback` is a pointer to a VkRenderPassSubpassFeedbackInfoEXT structure in which feedback is returned.

### Valid Usage (Implicit)

- VUID-VkRenderPassSubpassFeedbackCreateInfoEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_RENDER_PASS_SUBPASS_FEEDBACK_CREATE_INFO_EXT`
- VUID-VkRenderPassSubpassFeedbackCreateInfoEXT-pSubpassFeedback-parameter
  `pSubpassFeedback` **must** be a valid pointer to a VkRenderPassSubpassFeedbackInfoEXT structure

The `VkRenderPassSubpassFeedbackInfoEXT` structure is defined as:

```
// Provided by VK_EXT_subpass_merge_feedback
typedef struct VkRenderPassSubpassFeedbackInfoEXT {
    VkSubpassMergeStatusEXT    subpassMergeStatus;
    char                       description[VK_MAX_DESCRIPTION_SIZE];
    uint32_t                   postMergeIndex;
} VkRenderPassSubpassFeedbackInfoEXT;
```

- `subpassMergeStatus` is a `VkSubpassMergeStatusEXT` value specifying information about whether the subpass is merged with previous subpass and the reason why it is not merged.
- `description` is an array of `VK_MAX_DESCRIPTION_SIZE` `char` containing a null-terminated UTF-8 string which provides additional details.
- `postMergeIndex` is the subpass index after the subpass merging.

Possible values of `VkRenderPassSubpassFeedbackInfoEXT`:subpassMergeStatus are:

```
// Provided by VK_EXT_subpass_merge_feedback
typedef enum VkSubpassMergeStatusEXT {
    VK_SUBPASS_MERGE_STATUS_MERGED_EXT = 0,
    VK_SUBPASS_MERGE_STATUS_DISALLOWED_EXT = 1,
    VK_SUBPASS_MERGE_STATUS_NOT_MERGED_SIDE_EFFECTS_EXT = 2,
    VK_SUBPASS_MERGE_STATUS_NOT_MERGED_SAMPLES_MISMATCH_EXT = 3,
    VK_SUBPASS_MERGE_STATUS_NOT_MERGED_VIEWS_MISMATCH_EXT = 4,
    VK_SUBPASS_MERGE_STATUS_NOT_MERGED_ALIASING_EXT = 5,
    VK_SUBPASS_MERGE_STATUS_NOT_MERGED_DEPENDENCIES_EXT = 6,
    VK_SUBPASS_MERGE_STATUS_NOT_MERGED_INCOMPATIBLE_INPUT_ATTACHMENT_EXT = 7,
    VK_SUBPASS_MERGE_STATUS_NOT_MERGED_TOO_MANY_ATTACHMENTS_EXT = 8,
    VK_SUBPASS_MERGE_STATUS_NOT_MERGED_INSUFFICIENT_STORAGE_EXT = 9,
    VK_SUBPASS_MERGE_STATUS_NOT_MERGED_DEPTH_STENCIL_COUNT_EXT = 10,
    VK_SUBPASS_MERGE_STATUS_NOT_MERGED_RESOLVE_ATTACHMENT_REUSE_EXT = 11,
    VK_SUBPASS_MERGE_STATUS_NOT_MERGED_SINGLE_SUBPASS_EXT = 12,
    VK_SUBPASS_MERGE_STATUS_NOT_MERGED_UNSPECIFIED_EXT = 13,
} VkSubpassMergeStatusEXT;
```

- `VK_SUBPASS_MERGE_STATUS_MERGED_EXT` specifies the subpass is merged with a previous subpass.
- `VK_SUBPASS_MERGE_STATUS_DISALLOWED_EXT` specifies the subpass is disallowed to merge with previous subpass. If the render pass does not allow subpass merging, then all subpass statuses are set to this value. If a subpass description does not allow subpass merging, then only that subpass's status is set to this value.
- `VK_SUBPASS_MERGE_STATUS_NOT_MERGED_SIDE_EFFECTS_EXT` specifies the subpass is not merged because it contains side effects.
- `VK_SUBPASS_MERGE_STATUS_NOT_MERGED_SAMPLES_MISMATCH_EXT` specifies the subpass is not merged because sample count is not compatible with previous subpass.
- `VK_SUBPASS_MERGE_STATUS_NOT_MERGED_VIEWS_MISMATCH_EXT` specifies the subpass is not merged because view masks do not match with previous subpass.
- `VK_SUBPASS_MERGE_STATUS_NOT_MERGED_ALIASING_EXT` specifies the subpass is not merged because of attachments aliasing between them.
- `VK_SUBPASS_MERGE_STATUS_NOT_MERGED_DEPENDENCIES_EXT` specifies the subpass is not merged because subpass dependencies do not allow merging.
- `VK_SUBPASS_MERGE_STATUS_NOT_MERGED_INCOMPATIBLE_INPUT_ATTACHMENT_EXT` specifies the subpass is not merged because input attachment is not a color attachment from previous subpass or the formats are incompatible.
- `VK_SUBPASS_MERGE_STATUS_NOT_MERGED_TOO_MANY_ATTACHMENTS_EXT` specifies the subpass is not merged because of too many attachments.
- `VK_SUBPASS_MERGE_STATUS_NOT_MERGED_INSUFFICIENT_STORAGE_EXT` specifies the subpass is not merged because of insufficient memory.
- `VK_SUBPASS_MERGE_STATUS_NOT_MERGED_DEPTH_STENCIL_COUNT_EXT` specifies the subpass is not merged because of too many depth/stencil attachments.
- `VK_SUBPASS_MERGE_STATUS_NOT_MERGED_RESOLVE_ATTACHMENT_REUSE_EXT` specifies the subpass is not

merged because a resolve attachment is reused as an input attachment in a subsequent subpass.

- `VK_SUBPASS_MERGE_STATUS_NOT_MERGED_SINGLE_SUBPASS_EXT` specifies the subpass is not merged because the render pass has only one subpass.
- `VK_SUBPASS_MERGE_STATUS_NOT_MERGED_UNSPECIFIED_EXT` specifies other reasons why subpass is not merged. It is also the recommended default value that should be reported when a subpass is not merged and when no other value is appropriate.

# 8.10. Common Render Pass Data Races (Informative)

Due to the complexity of how rendering is performed, there are several ways an application can accidentally introduce a data race, usually by doing something that may seem benign but actually cannot be supported. This section indicates a number of the more common cases as guidelines to help avoid them.

## 8.10.1. Sampling From a Read-only Attachment

Vulkan includes read-only layouts for depth/stencil images, that allow the images to be both read during a render pass for the purposes of depth/stencil tests, and read as a non-attachment.

However, because `VK_ATTACHMENT_STORE_OP_STORE` and `VK_ATTACHMENT_STORE_OP_DONT_CARE` may perform write operations, even if no recorded command writes to an attachment, reading from an image while also using it as an attachment with these store operations can result in a data race. If the reads from the non-attachment are performed in a fragment shader where the accessed samples match those covered by the fragment shader, no data race will occur as store operations are guaranteed to operate after fragment shader execution for the set of samples the fragment covers. Notably, input attachments can also be used for this case. Reading other samples or in any other shader stage can result in unexpected behavior due to the potential for a data race, and validation errors should be generated for doing so. In practice, many applications have shipped reading samples outside of the covered fragment without any observable issue, but there is no guarantee that this will always work, and it is not advisable to rely on this in new or re-worked code bases. As `VK_ATTACHMENT_STORE_OP_NONE` is guaranteed to perform no writes, applications wishing to read an image as both an attachment and a non-attachment should make use of this store operation, coupled with a load operation that also performs no writes.

## 8.10.2. Non-overlapping Access Between Resources

When relying on non-overlapping accesses between attachments and other resources, it is important to note that load and store operations have fairly wide alignment requirements - potentially affecting entire subresources and adjacent depth/stencil aspects. This makes it invalid to access a non-attachment subresource that is simultaneously being used as an attachment where either access performs a write operation.

The only exception to this is if a subresource is in the `VK_IMAGE_LAYOUT_ATTACHMENT_FEEDBACK_LOOP_OPTIMAL_EXT` image layout, in which case the overlap is defined to occur at a per-pixel granularity, and applications can read data from pixels outside the render area without introducing a data race.

### 8.10.3. Depth/Stencil and Input Attachments

When rendering to only the depth OR stencil aspect of an image, an input attachment accessing the other aspect will not cause a data race only under very specific conditions. To avoid a data race, the aspect not being written **must** be in a read-only layout, and writes to it **must** be disabled in the draw state. For example, to read from stencil while writing depth, the attachment **must** be in `VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_STENCIL_READ_ONLY_OPTIMAL` (or equivalent), and the stencil write mask **must** be set to 0. Similarly to read from depth while writing stencil, the attachment **must** be in `VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_STENCIL_ATTACHMENT_OPTIMAL` (or equivalent), and depth write enable **must** be `VK_FALSE`.

### 8.10.4. Synchronization Options

There are several synchronization options available to synchronize between accesses to resources within a render pass. Some of the options are outlined below:

- A VkSubpassDependency in a render pass object can synchronize attachment writes and multisample resolve operations from a prior subpass for subsequent input attachment reads.
- A vkCmdPipelineBarrier inside a subpass can synchronize prior attachment writes in the subpass with subsequent input attachment reads.
- A vkCmdPipelineBarrier inside a subpass can synchronize prior attachment writes in the subpass with subsequent non-attachment reads if the attachment is in the `VK_IMAGE_LAYOUT_ATTACHMENT_FEEDBACK_LOOP_OPTIMAL_EXT` image layout.
- If a subresource is used as a color and input attachment, and the pipeline performing the read was created with `VK_PIPELINE_COLOR_BLEND_STATE_CREATE_RASTERIZATION_ORDER_ATTACHMENT_ACCESS_BIT_EXT`
- If a subresource is used as a depth and input attachment, and the pipeline performing the read was created with `VK_PIPELINE_DEPTH_STENCIL_STATE_CREATE_RASTERIZATION_ORDER_ATTACHMENT_DEPTH_ACCESS_BIT_EXT`
- If a subresource is used as a stencil and input attachment, and the pipeline performing the read was created with `VK_PIPELINE_DEPTH_STENCIL_STATE_CREATE_RASTERIZATION_ORDER_ATTACHMENT_STENCIL_ACCESS_BIT_EXT`
- If a subresource is used as two separate non-attachment resources, writes to a pixel or individual sample in a fragment shader can be synchronized with access to the same pixel or sample in another fragment shader by using one of the fragment interlock execution modes.

# Chapter 9. Shaders

A shader specifies programmable operations that execute for each vertex, control point, tessellated vertex, primitive, fragment, or workgroup in the corresponding stage(s) of the graphics and compute pipelines.

Graphics pipelines include vertex shader execution as a result of primitive assembly, followed, if enabled, by tessellation control and evaluation shaders operating on patches, geometry shaders, if enabled, operating on primitives, and fragment shaders, if present, operating on fragments generated by Rasterization. In this specification, vertex, tessellation control, tessellation evaluation and geometry shaders are collectively referred to as pre-rasterization shader stages and occur in the logical pipeline before rasterization. The fragment shader occurs logically after rasterization.

Only the compute shader stage is included in a compute pipeline. Compute shaders operate on compute invocations in a workgroup.

Shaders **can** read from input variables, and read from and write to output variables. Input and output variables **can** be used to transfer data between shader stages, or to allow the shader to interact with values that exist in the execution environment. Similarly, the execution environment provides constants describing capabilities.

Shader variables are associated with execution environment-provided inputs and outputs using *built-in* decorations in the shader. The available decorations for each stage are documented in the following subsections.

## 9.1. Shader Objects

Shaders **may** be compiled and linked into pipeline objects as described in Pipelines chapter, or if the `shaderObject` feature is enabled they **may** be compiled into individual per-stage *shader objects* which **can** be bound on a command buffer independently from one another. Unlike pipelines, shader objects are not intrinsically tied to any specific set of state. Instead, state is specified dynamically in the command buffer.

Each shader object represents a single compiled shader stage, which **may optionally** be linked with one or more other stages.

Shader objects are represented by `VkShaderEXT` handles:

```
// Provided by VK_EXT_shader_object
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkShaderEXT)
```

### 9.1.1. Shader Object Creation

Shader objects **may** be created from shader code provided as SPIR-V, or in an opaque, implementation-defined binary format specific to the physical device.

To create one or more shader objects, call:

```
// Provided by VK_EXT_shader_object
VkResult vkCreateShadersEXT(
    VkDevice                                    device,
    uint32_t                                    createInfoCount,
    const VkShaderCreateInfoEXT*                pCreateInfos,
    const VkAllocationCallbacks*                pAllocator,
    VkShaderEXT*                                pShaders);
```

- `device` is the logical device that creates the shader objects.
- `createInfoCount` is the length of the `pCreateInfos` and `pShaders` arrays.
- `pCreateInfos` is a pointer to an array of VkShaderCreateInfoEXT structures.
- `pAllocator` controls host memory allocation as described in the Memory Allocation chapter.
- `pShaders` is a pointer to an array of VkShaderEXT handles in which the resulting shader objects are returned.

When this function returns, whether or not it succeeds, it is guaranteed that every element of `pShaders` will have been overwritten by either VK_NULL_HANDLE or a valid `VkShaderEXT` handle.

This means that whenever shader creation fails, the application **can** determine which shader the returned error pertains to by locating the first VK_NULL_HANDLE element in `pShaders`. It also means that an application **can** reliably clean up from a failed call by iterating over the `pShaders` array and destroying every element that is not VK_NULL_HANDLE.

### Valid Usage

- VUID-vkCreateShadersEXT-device-09669
  `device` **must** support at least one queue family with one of the `VK_QUEUE_COMPUTE_BIT` or `VK_QUEUE_GRAPHICS_BIT` capabilities
- VUID-vkCreateShadersEXT-stage-09670
  If the `stage` member of any element of `pCreateInfos` is `VK_SHADER_STAGE_COMPUTE_BIT`, `device` **must** support at least one queue family with the `VK_QUEUE_COMPUTE_BIT` capability
- VUID-vkCreateShadersEXT-stage-09671
  If the `stage` member of any element of `pCreateInfos` is `VK_SHADER_STAGE_TASK_BIT_EXT`, `VK_SHADER_STAGE_MESH_BIT_EXT`, `VK_SHADER_STAGE_VERTEX_BIT`, `VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT`, `VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT`, `VK_SHADER_STAGE_GEOMETRY_BIT`, or `VK_SHADER_STAGE_FRAGMENT_BIT`, `device` **must** support at least one queue family with the `VK_QUEUE_GRAPHICS_BIT` capability
- VUID-vkCreateShadersEXT-None-08400
  The `shaderObject` feature **must** be enabled
- VUID-vkCreateShadersEXT-pCreateInfos-08402
  If the `flags` member of any element of `pCreateInfos` includes `VK_SHADER_CREATE_LINK_STAGE_BIT_EXT`, the `flags` member of all other elements of `pCreateInfos` whose `stage` is `VK_SHADER_STAGE_VERTEX_BIT`, `VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT`, `VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT`,

VK_SHADER_STAGE_GEOMETRY_BIT, or VK_SHADER_STAGE_FRAGMENT_BIT **must** also include VK_SHADER_CREATE_LINK_STAGE_BIT_EXT

- VUID-vkCreateShadersEXT-pCreateInfos-08403
  If the flags member of any element of pCreateInfos includes VK_SHADER_CREATE_LINK_STAGE_BIT_EXT, the flags member of all other elements of pCreateInfos whose stage is VK_SHADER_STAGE_TASK_BIT_EXT or VK_SHADER_STAGE_MESH_BIT_EXT **must** also include VK_SHADER_CREATE_LINK_STAGE_BIT_EXT
- VUID-vkCreateShadersEXT-pCreateInfos-08404
  If the flags member of any element of pCreateInfos whose stage is VK_SHADER_STAGE_TASK_BIT_EXT or VK_SHADER_STAGE_MESH_BIT_EXT includes VK_SHADER_CREATE_LINK_STAGE_BIT_EXT, there **must** be no member of pCreateInfos whose stage is VK_SHADER_STAGE_VERTEX_BIT and whose flags member includes VK_SHADER_CREATE_LINK_STAGE_BIT_EXT
- VUID-vkCreateShadersEXT-pCreateInfos-08405
  If there is any element of pCreateInfos whose stage is VK_SHADER_STAGE_MESH_BIT_EXT and whose flags member includes both VK_SHADER_CREATE_LINK_STAGE_BIT_EXT and VK_SHADER_CREATE_NO_TASK_SHADER_BIT_EXT, there **must** be no element of pCreateInfos whose stage is VK_SHADER_STAGE_TASK_BIT_EXT and whose flags member includes VK_SHADER_CREATE_LINK_STAGE_BIT_EXT
- VUID-vkCreateShadersEXT-pCreateInfos-08409
  For each element of pCreateInfos whose flags member includes VK_SHADER_CREATE_LINK_STAGE_BIT_EXT, if there is any other element of pCreateInfos whose stage is logically later than the stage of the former and whose flags member also includes VK_SHADER_CREATE_LINK_STAGE_BIT_EXT, the nextStage of the former **must** be equal to the stage of the element with the logically earliest stage following the stage of the former whose flags member also includes VK_SHADER_CREATE_LINK_STAGE_BIT_EXT
- VUID-vkCreateShadersEXT-pCreateInfos-08410
  The stage member of each element of pCreateInfos whose flags member includes VK_SHADER_CREATE_LINK_STAGE_BIT_EXT **must** be unique
- VUID-vkCreateShadersEXT-pCreateInfos-08411
  The codeType member of all elements of pCreateInfos whose flags member includes VK_SHADER_CREATE_LINK_STAGE_BIT_EXT **must** be the same
- VUID-vkCreateShadersEXT-pCreateInfos-08867
  If pCreateInfos contains elements with both VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT and VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT, both elements' flags include VK_SHADER_CREATE_LINK_STAGE_BIT_EXT, both elements' codeType is VK_SHADER_CODE_TYPE_SPIRV_EXT, and the VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT stage's pCode contains an OpExecutionMode instruction specifying the type of subdivision, it **must** match the subdivision type specified in the VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT stage
- VUID-vkCreateShadersEXT-pCreateInfos-08868
  If pCreateInfos contains elements with both VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT and VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT, both elements' flags include VK_SHADER_CREATE_LINK_STAGE_BIT_EXT, both elements' codeType is VK_SHADER_CODE_TYPE_SPIRV_EXT, and the VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT stage's

pCode contains an OpExecutionMode instruction specifying the orientation of triangles, it **must** match the triangle orientation specified in the VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT stage

- VUID-vkCreateShadersEXT-pCreateInfos-08869
  If pCreateInfos contains elements with both VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT and VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT, both elements' flags include VK_SHADER_CREATE_LINK_STAGE_BIT_EXT, both elements' codeType is VK_SHADER_CODE_TYPE_SPIRV_EXT, and the VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT stage's pCode contains an OpExecutionMode instruction specifying PointMode, the VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT stage **must** also contain an OpExecutionMode instruction specifying PointMode
- VUID-vkCreateShadersEXT-pCreateInfos-08870
  If pCreateInfos contains elements with both VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT and VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT, both elements' flags include VK_SHADER_CREATE_LINK_STAGE_BIT_EXT, both elements' codeType is VK_SHADER_CODE_TYPE_SPIRV_EXT, and the VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT stage's pCode contains an OpExecutionMode instruction specifying the spacing of segments on the edges of tessellated primitives, it **must** match the segment spacing specified in the VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT stage
- VUID-vkCreateShadersEXT-pCreateInfos-08871
  If pCreateInfos contains elements with both VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT and VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT, both elements' flags include VK_SHADER_CREATE_LINK_STAGE_BIT_EXT, both elements' codeType is VK_SHADER_CODE_TYPE_SPIRV_EXT, and the VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT stage's pCode contains an OpExecutionMode instruction specifying the output patch size, it **must** match the output patch size specified in the VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT stage
- VUID-vkCreateShadersEXT-pCreateInfos-09632
  If pCreateInfos contains a VK_SHADER_STAGE_MESH_BIT_EXT with codeType of VK_SHADER_CODE_TYPE_SPIRV_EXT and VK_SHADER_CREATE_NO_TASK_SHADER_BIT_EXT is not set, then the mesh shader's entry point **must** not declare a variable with a DrawIndex BuiltIn decoration

## Valid Usage (Implicit)

- VUID-vkCreateShadersEXT-device-parameter
  device **must** be a valid VkDevice handle
- VUID-vkCreateShadersEXT-pCreateInfos-parameter
  pCreateInfos **must** be a valid pointer to an array of createInfoCount valid VkShaderCreateInfoEXT structures
- VUID-vkCreateShadersEXT-pAllocator-parameter
  If pAllocator is not NULL, pAllocator **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkCreateShadersEXT-pShaders-parameter

pShaders **must** be a valid pointer to an array of createInfoCount VkShaderEXT handles

- VUID-vkCreateShadersEXT-createInfoCount-arraylength
  createInfoCount **must** be greater than 0

## Return Codes

**Success**

- VK_SUCCESS
- VK_INCOMPATIBLE_SHADER_BINARY_EXT

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY
- VK_ERROR_INITIALIZATION_FAILED

The VkShaderCreateInfoEXT structure is defined as:

```
// Provided by VK_EXT_shader_object
typedef struct VkShaderCreateInfoEXT {
    VkStructureType                 sType;
    const void*                     pNext;
    VkShaderCreateFlagsEXT          flags;
    VkShaderStageFlagBits           stage;
    VkShaderStageFlags              nextStage;
    VkShaderCodeTypeEXT             codeType;
    size_t                          codeSize;
    const void*                     pCode;
    const char*                     pName;
    uint32_t                        setLayoutCount;
    const VkDescriptorSetLayout*    pSetLayouts;
    uint32_t                        pushConstantRangeCount;
    const VkPushConstantRange*      pPushConstantRanges;
    const VkSpecializationInfo*     pSpecializationInfo;
} VkShaderCreateInfoEXT;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- flags is a bitmask of VkShaderCreateFlagBitsEXT describing additional parameters of the shader.
- stage is a VkShaderStageFlagBits value specifying a single shader stage.
- nextStage is a bitmask of VkShaderStageFlagBits specifying zero or stages which **may** be used as a logically next bound stage when drawing with the shader bound.
- codeType is a VkShaderCodeTypeEXT value specifying the type of the shader code pointed to be

pCode.

- codeSize is the size in bytes of the shader code pointed to be pCode.
- pCode is a pointer to the shader code to use to create the shader.
- pName is a pointer to a null-terminated UTF-8 string specifying the entry point name of the shader for this stage.
- setLayoutCount is the number of descriptor set layouts pointed to by pSetLayouts.
- pSetLayouts is a pointer to an array of VkDescriptorSetLayout objects used by the shader stage.
- pushConstantRangeCount is the number of push constant ranges pointed to by pPushConstantRanges.
- pPushConstantRanges is a pointer to an array of VkPushConstantRange structures used by the shader stage.
- pSpecializationInfo is a pointer to a VkSpecializationInfo structure, as described in Specialization Constants, or NULL.

## Valid Usage

- VUID-VkShaderCreateInfoEXT-codeSize-08735
  If codeType is VK_SHADER_CODE_TYPE_SPIRV_EXT, codeSize **must** be a multiple of 4
- VUID-VkShaderCreateInfoEXT-pCode-08736
  If codeType is VK_SHADER_CODE_TYPE_SPIRV_EXT, pCode **must** point to valid SPIR-V code, formatted and packed as described by the Khronos SPIR-V Specification
- VUID-VkShaderCreateInfoEXT-pCode-08737
  If codeType is VK_SHADER_CODE_TYPE_SPIRV_EXT, pCode **must** adhere to the validation rules described by the Validation Rules within a Module section of the SPIR-V Environment appendix
- VUID-VkShaderCreateInfoEXT-pCode-08738
  If codeType is VK_SHADER_CODE_TYPE_SPIRV_EXT, pCode **must** declare the Shader capability for SPIR-V code
- VUID-VkShaderCreateInfoEXT-pCode-08739
  If codeType is VK_SHADER_CODE_TYPE_SPIRV_EXT, pCode **must** not declare any capability that is not supported by the API, as described by the Capabilities section of the SPIR-V Environment appendix
- VUID-VkShaderCreateInfoEXT-pCode-08740
  If codeType is VK_SHADER_CODE_TYPE_SPIRV_EXT, and pCode declares any of the capabilities listed in the SPIR-V Environment appendix, one of the corresponding requirements **must** be satisfied
- VUID-VkShaderCreateInfoEXT-pCode-08741
  If codeType is VK_SHADER_CODE_TYPE_SPIRV_EXT, pCode **must** not declare any SPIR-V extension that is not supported by the API, as described by the Extension section of the SPIR-V Environment appendix
- VUID-VkShaderCreateInfoEXT-pCode-08742
  If codeType is VK_SHADER_CODE_TYPE_SPIRV_EXT, and pCode declares any of the SPIR-V

extensions listed in the SPIR-V Environment appendix, one of the corresponding requirements **must** be satisfied

- VUID-VkShaderCreateInfoEXT-flags-08412
  If `stage` is not `VK_SHADER_STAGE_TASK_BIT_EXT`, `VK_SHADER_STAGE_MESH_BIT_EXT`, `VK_SHADER_STAGE_VERTEX_BIT`, `VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT`, `VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT`, `VK_SHADER_STAGE_GEOMETRY_BIT`, or `VK_SHADER_STAGE_FRAGMENT_BIT`, `flags` **must** not include `VK_SHADER_CREATE_LINK_STAGE_BIT_EXT`
- VUID-VkShaderCreateInfoEXT-flags-08486
  If `stage` is not `VK_SHADER_STAGE_FRAGMENT_BIT`, `flags` **must** not include `VK_SHADER_CREATE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_EXT`
- VUID-VkShaderCreateInfoEXT-flags-08487
  If the `attachmentFragmentShadingRate` feature is not enabled, `flags` **must** not include `VK_SHADER_CREATE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_EXT`
- VUID-VkShaderCreateInfoEXT-flags-08488
  If `stage` is not `VK_SHADER_STAGE_FRAGMENT_BIT`, `flags` **must** not include `VK_SHADER_CREATE_FRAGMENT_DENSITY_MAP_ATTACHMENT_BIT_EXT`
- VUID-VkShaderCreateInfoEXT-flags-08489
  If the `fragmentDensityMap` feature is not enabled, `flags` **must** not include `VK_SHADER_CREATE_FRAGMENT_DENSITY_MAP_ATTACHMENT_BIT_EXT`
- VUID-VkShaderCreateInfoEXT-flags-09404
  If `flags` includes `VK_SHADER_CREATE_ALLOW_VARYING_SUBGROUP_SIZE_BIT_EXT`, the `subgroupSizeControl` feature **must** be enabled
- VUID-VkShaderCreateInfoEXT-flags-09405
  If `flags` includes `VK_SHADER_CREATE_REQUIRE_FULL_SUBGROUPS_BIT_EXT`, the `computeFullSubgroups` feature **must** be enabled
- VUID-VkShaderCreateInfoEXT-flags-08992
  If `flags` includes `VK_SHADER_CREATE_REQUIRE_FULL_SUBGROUPS_BIT_EXT`, `stage` **must** be one of `VK_SHADER_STAGE_MESH_BIT_EXT`, `VK_SHADER_STAGE_TASK_BIT_EXT`, or `VK_SHADER_STAGE_COMPUTE_BIT`
- VUID-VkShaderCreateInfoEXT-flags-08485
  If `stage` is not `VK_SHADER_STAGE_COMPUTE_BIT`, `flags` **must** not include `VK_SHADER_CREATE_DISPATCH_BASE_BIT_EXT`
- VUID-VkShaderCreateInfoEXT-flags-08414
  If `stage` is not `VK_SHADER_STAGE_MESH_BIT_EXT`, `flags` **must** not include `VK_SHADER_CREATE_NO_TASK_SHADER_BIT_EXT`
- VUID-VkShaderCreateInfoEXT-flags-08416
  If `flags` includes both `VK_SHADER_CREATE_ALLOW_VARYING_SUBGROUP_SIZE_BIT_EXT` and `VK_SHADER_CREATE_REQUIRE_FULL_SUBGROUPS_BIT_EXT`, the local workgroup size in the X dimension of the shader **must** be a multiple of `maxSubgroupSize`
- VUID-VkShaderCreateInfoEXT-flags-08417
  If `flags` includes `VK_SHADER_CREATE_REQUIRE_FULL_SUBGROUPS_BIT_EXT` but not `VK_SHADER_CREATE_ALLOW_VARYING_SUBGROUP_SIZE_BIT_EXT` and no

VkShaderRequiredSubgroupSizeCreateInfoEXT structure is included in the pNext chain, the local workgroup size in the X dimension of the shader **must** be a multiple of subgroupSize

- VUID-VkShaderCreateInfoEXT-stage-08418
  stage **must** not be VK_SHADER_STAGE_ALL_GRAPHICS or VK_SHADER_STAGE_ALL
- VUID-VkShaderCreateInfoEXT-stage-08419
  If the tessellationShader feature is not enabled, stage **must** not be VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT or VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT
- VUID-VkShaderCreateInfoEXT-stage-08420
  If the geometryShader feature is not enabled, stage **must** not be VK_SHADER_STAGE_GEOMETRY_BIT
- VUID-VkShaderCreateInfoEXT-stage-08421
  If the taskShader feature is not enabled, stage **must** not be VK_SHADER_STAGE_TASK_BIT_EXT
- VUID-VkShaderCreateInfoEXT-stage-08422
  If the meshShader feature is not enabled, stage **must** not be VK_SHADER_STAGE_MESH_BIT_EXT
- VUID-VkShaderCreateInfoEXT-stage-08425
  stage **must** not be VK_SHADER_STAGE_SUBPASS_SHADING_BIT_HUAWEI
- VUID-VkShaderCreateInfoEXT-stage-08426
  stage **must** not be VK_SHADER_STAGE_CLUSTER_CULLING_BIT_HUAWEI
- VUID-VkShaderCreateInfoEXT-nextStage-08427
  If stage is VK_SHADER_STAGE_VERTEX_BIT, nextStage **must** not include any bits other than VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT, VK_SHADER_STAGE_GEOMETRY_BIT, and VK_SHADER_STAGE_FRAGMENT_BIT
- VUID-VkShaderCreateInfoEXT-nextStage-08428
  If the tessellationShader feature is not enabled, nextStage **must** not include VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT or VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT
- VUID-VkShaderCreateInfoEXT-nextStage-08429
  If the geometryShader feature is not enabled, nextStage **must** not include VK_SHADER_STAGE_GEOMETRY_BIT
- VUID-VkShaderCreateInfoEXT-nextStage-08430
  If stage is VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT, nextStage **must** not include any bits other than VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT
- VUID-VkShaderCreateInfoEXT-nextStage-08431
  If stage is VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT, nextStage **must** not include any bits other than VK_SHADER_STAGE_GEOMETRY_BIT and VK_SHADER_STAGE_FRAGMENT_BIT
- VUID-VkShaderCreateInfoEXT-nextStage-08433
  If stage is VK_SHADER_STAGE_GEOMETRY_BIT, nextStage **must** not include any bits other than VK_SHADER_STAGE_FRAGMENT_BIT
- VUID-VkShaderCreateInfoEXT-nextStage-08434
  If stage is VK_SHADER_STAGE_FRAGMENT_BIT or VK_SHADER_STAGE_COMPUTE_BIT, nextStage **must**

be 0

- VUID-VkShaderCreateInfoEXT-nextStage-08435
  If `stage` is `VK_SHADER_STAGE_TASK_BIT_EXT`, `nextStage` **must** not include any bits other than `VK_SHADER_STAGE_MESH_BIT_EXT`
- VUID-VkShaderCreateInfoEXT-nextStage-08436
  If `stage` is `VK_SHADER_STAGE_MESH_BIT_EXT`, `nextStage` **must** not include any bits other than `VK_SHADER_STAGE_FRAGMENT_BIT`
- VUID-VkShaderCreateInfoEXT-pName-08440
  If `codeType` is `VK_SHADER_CODE_TYPE_SPIRV_EXT`, `pName` **must** be the name of an `OpEntryPoint` in `pCode` with an execution model that matches `stage`
- VUID-VkShaderCreateInfoEXT-pCode-08492
  If `codeType` is `VK_SHADER_CODE_TYPE_BINARY_EXT`, `pCode` **must** be aligned to `16` bytes
- VUID-VkShaderCreateInfoEXT-pCode-08493
  If `codeType` is `VK_SHADER_CODE_TYPE_SPIRV_EXT`, `pCode` **must** be aligned to `4` bytes
- VUID-VkShaderCreateInfoEXT-pCode-08448
  If `codeType` is `VK_SHADER_CODE_TYPE_SPIRV_EXT`, and the identified entry point includes any variable in its interface that is declared with the `ClipDistance BuiltIn` decoration, that variable **must** not have an array size greater than `VkPhysicalDeviceLimits::maxClipDistances`
- VUID-VkShaderCreateInfoEXT-pCode-08449
  If `codeType` is `VK_SHADER_CODE_TYPE_SPIRV_EXT`, and the identified entry point includes any variable in its interface that is declared with the `CullDistance BuiltIn` decoration, that variable **must** not have an array size greater than `VkPhysicalDeviceLimits::maxCullDistances`
- VUID-VkShaderCreateInfoEXT-pCode-08450
  If `codeType` is `VK_SHADER_CODE_TYPE_SPIRV_EXT`, and the identified entry point includes variables in its interface that are declared with the `ClipDistance BuiltIn` decoration and variables in its interface that are declared with the `CullDistance BuiltIn` decoration, those variables **must** not have array sizes which sum to more than `VkPhysicalDeviceLimits::maxCombinedClipAndCullDistances`
- VUID-VkShaderCreateInfoEXT-pCode-08451
  If `codeType` is `VK_SHADER_CODE_TYPE_SPIRV_EXT`, and the identified entry point includes any variable in its interface that is declared with the `SampleMask BuiltIn` decoration, that variable **must** not have an array size greater than `VkPhysicalDeviceLimits::maxSampleMaskWords`
- VUID-VkShaderCreateInfoEXT-pCode-08452
  If `codeType` is `VK_SHADER_CODE_TYPE_SPIRV_EXT`, and `stage` is `VK_SHADER_STAGE_VERTEX_BIT`, the identified entry point **must** not include any input variable in its interface that is decorated with `CullDistance`
- VUID-VkShaderCreateInfoEXT-pCode-08453
  If `codeType` is `VK_SHADER_CODE_TYPE_SPIRV_EXT`, and `stage` is `VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT` or `VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT`, and the identified entry point has an

OpExecutionMode instruction specifying a patch size with OutputVertices, the patch size **must** be greater than 0 and less than or equal to VkPhysicalDeviceLimits::maxTessellationPatchSize

- VUID-VkShaderCreateInfoEXT-pCode-08454
  If codeType is VK_SHADER_CODE_TYPE_SPIRV_EXT, and stage is VK_SHADER_STAGE_GEOMETRY_BIT, the identified entry point **must** have an OpExecutionMode instruction specifying a maximum output vertex count that is greater than 0 and less than or equal to VkPhysicalDeviceLimits::maxGeometryOutputVertices
- VUID-VkShaderCreateInfoEXT-pCode-08455
  If codeType is VK_SHADER_CODE_TYPE_SPIRV_EXT, and stage is VK_SHADER_STAGE_GEOMETRY_BIT, the identified entry point **must** have an OpExecutionMode instruction specifying an invocation count that is greater than 0 and less than or equal to VkPhysicalDeviceLimits::maxGeometryShaderInvocations
- VUID-VkShaderCreateInfoEXT-pCode-08456
  If codeType is VK_SHADER_CODE_TYPE_SPIRV_EXT, and stage is a pre-rasterization shader stage, and the identified entry point writes to Layer for any primitive, it **must** write the same value to Layer for all vertices of a given primitive
- VUID-VkShaderCreateInfoEXT-pCode-08457
  If codeType is VK_SHADER_CODE_TYPE_SPIRV_EXT, and stage is a pre-rasterization shader stage, and the identified entry point writes to ViewportIndex for any primitive, it **must** write the same value to ViewportIndex for all vertices of a given primitive
- VUID-VkShaderCreateInfoEXT-pCode-08458
  If codeType is VK_SHADER_CODE_TYPE_SPIRV_EXT, and stage is VK_SHADER_STAGE_FRAGMENT_BIT, the identified entry point **must** not include any output variables in its interface decorated with CullDistance
- VUID-VkShaderCreateInfoEXT-pCode-08459
  If codeType is VK_SHADER_CODE_TYPE_SPIRV_EXT, and stage is VK_SHADER_STAGE_FRAGMENT_BIT, and the identified entry point writes to FragDepth in any execution path, all execution paths that are not exclusive to helper invocations **must** either discard the fragment, or write or initialize the value of FragDepth
- VUID-VkShaderCreateInfoEXT-pCode-08460
  If codeType is VK_SHADER_CODE_TYPE_SPIRV_EXT, the shader code in pCode **must** be valid as described by the Khronos SPIR-V Specification after applying the specializations provided in pSpecializationInfo, if any, and then converting all specialization constants into fixed constants
- VUID-VkShaderCreateInfoEXT-codeType-08872
  If codeType is VK_SHADER_CODE_TYPE_SPIRV_EXT, and stage is VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT, pCode **must** contain an OpExecutionMode instruction specifying the type of subdivision
- VUID-VkShaderCreateInfoEXT-codeType-08873
  If codeType is VK_SHADER_CODE_TYPE_SPIRV_EXT, and stage is VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT, pCode **must** contain an OpExecutionMode instruction specifying the orientation of triangles generated by the tessellator
- VUID-VkShaderCreateInfoEXT-codeType-08874

If codeType is VK_SHADER_CODE_TYPE_SPIRV_EXT, and stage is VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT, pCode **must** contain an OpExecutionMode instruction specifying the spacing of segments on the edges of tessellated primitives

- VUID-VkShaderCreateInfoEXT-codeType-08875
  If codeType is VK_SHADER_CODE_TYPE_SPIRV_EXT, and stage is VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT, pCode **must** contain an OpExecutionMode instruction specifying the output patch size
- VUID-VkShaderCreateInfoEXT-pPushConstantRanges-10063
  Any two elements of pPushConstantRanges **must** not include the same stage in stageFlags
- VUID-VkShaderCreateInfoEXT-codeType-10064
  If codeType is VK_SHADER_CODE_TYPE_SPIRV_EXT, and if a push constant block is declared in a shader, then an element of pPushConstantRanges::stageFlags **must** match pname::stage
- VUID-VkShaderCreateInfoEXT-codeType-10065
  If codeType is VK_SHADER_CODE_TYPE_SPIRV_EXT, and if a push constant block is declared in a shader, the block must be contained inside the element of pPushConstantRanges that matches the stage

## Valid Usage (Implicit)

- VUID-VkShaderCreateInfoEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_SHADER_CREATE_INFO_EXT
- VUID-VkShaderCreateInfoEXT-pNext-pNext
  Each pNext member of any structure (including this one) in the pNext chain **must** be either NULL or a pointer to a valid instance of VkPipelineShaderStageRequiredSubgroupSizeCreateInfo or VkValidationFeaturesEXT
- VUID-VkShaderCreateInfoEXT-sType-unique
  The sType value of each struct in the pNext chain **must** be unique
- VUID-VkShaderCreateInfoEXT-flags-parameter
  flags **must** be a valid combination of VkShaderCreateFlagBitsEXT values
- VUID-VkShaderCreateInfoEXT-stage-parameter
  stage **must** be a valid VkShaderStageFlagBits value
- VUID-VkShaderCreateInfoEXT-nextStage-parameter
  nextStage **must** be a valid combination of VkShaderStageFlagBits values
- VUID-VkShaderCreateInfoEXT-codeType-parameter
  codeType **must** be a valid VkShaderCodeTypeEXT value
- VUID-VkShaderCreateInfoEXT-pCode-parameter
  pCode **must** be a valid pointer to an array of codeSize bytes
- VUID-VkShaderCreateInfoEXT-pName-parameter
  If pName is not NULL, pName **must** be a null-terminated UTF-8 string
- VUID-VkShaderCreateInfoEXT-pSetLayouts-parameter
  If setLayoutCount is not 0, and pSetLayouts is not NULL, pSetLayouts **must** be a valid pointer to an array of setLayoutCount valid VkDescriptorSetLayout handles

- VUID-VkShaderCreateInfoEXT-pPushConstantRanges-parameter
  If `pushConstantRangeCount` is not `0`, and `pPushConstantRanges` is not `NULL`, `pPushConstantRanges` **must** be a valid pointer to an array of `pushConstantRangeCount` valid VkPushConstantRange structures
- VUID-VkShaderCreateInfoEXT-pSpecializationInfo-parameter
  If `pSpecializationInfo` is not `NULL`, `pSpecializationInfo` **must** be a valid pointer to a valid VkSpecializationInfo structure
- VUID-VkShaderCreateInfoEXT-codeSize-arraylength
  `codeSize` **must** be greater than `0`

```
// Provided by VK_EXT_shader_object
typedef VkFlags VkShaderCreateFlagsEXT;
```

`VkShaderCreateFlagsEXT` is a bitmask type for setting a mask of zero or more VkShaderCreateFlagBitsEXT.

Possible values of the `flags` member of VkShaderCreateInfoEXT specifying how a shader object is created, are:

```
// Provided by VK_EXT_shader_object
typedef enum VkShaderCreateFlagBitsEXT {
    VK_SHADER_CREATE_LINK_STAGE_BIT_EXT = 0x00000001,
  // Provided by VK_EXT_shader_object with VK_EXT_subgroup_size_control or
VK_VERSION_1_3
    VK_SHADER_CREATE_ALLOW_VARYING_SUBGROUP_SIZE_BIT_EXT = 0x00000002,
  // Provided by VK_EXT_shader_object with VK_EXT_subgroup_size_control or
VK_VERSION_1_3
    VK_SHADER_CREATE_REQUIRE_FULL_SUBGROUPS_BIT_EXT = 0x00000004,
  // Provided by VK_EXT_shader_object with VK_EXT_mesh_shader or VK_NV_mesh_shader
    VK_SHADER_CREATE_NO_TASK_SHADER_BIT_EXT = 0x00000008,
  // Provided by VK_EXT_shader_object with VK_KHR_device_group or VK_VERSION_1_1
    VK_SHADER_CREATE_DISPATCH_BASE_BIT_EXT = 0x00000010,
  // Provided by VK_KHR_fragment_shading_rate with VK_EXT_shader_object
    VK_SHADER_CREATE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_EXT = 0x00000020,
  // Provided by VK_EXT_fragment_density_map with VK_EXT_shader_object
    VK_SHADER_CREATE_FRAGMENT_DENSITY_MAP_ATTACHMENT_BIT_EXT = 0x00000040,
} VkShaderCreateFlagBitsEXT;
```

- `VK_SHADER_CREATE_LINK_STAGE_BIT_EXT` specifies that a shader is linked to all other shaders created in the same vkCreateShadersEXT call whose VkShaderCreateInfoEXT structures' `flags` include `VK_SHADER_CREATE_LINK_STAGE_BIT_EXT`.
- `VK_SHADER_CREATE_ALLOW_VARYING_SUBGROUP_SIZE_BIT_EXT` specifies that the `SubgroupSize` **may** vary in a task, mesh, or compute shader.
- `VK_SHADER_CREATE_REQUIRE_FULL_SUBGROUPS_BIT_EXT` specifies that the subgroup sizes **must** be launched with all invocations active in a task, mesh, or compute shader.

- VK_SHADER_CREATE_NO_TASK_SHADER_BIT_EXT specifies that a mesh shader **must** only be used without a task shader. Otherwise, the mesh shader **must** only be used with a task shader.
- VK_SHADER_CREATE_DISPATCH_BASE_BIT_EXT specifies that a compute shader **can** be used with vkCmdDispatchBase with a non-zero base workgroup.
- VK_SHADER_CREATE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_EXT specifies that a fragment shader **can** be used with a fragment shading rate attachment.
- VK_SHADER_CREATE_FRAGMENT_DENSITY_MAP_ATTACHMENT_BIT_EXT specifies that a fragment shader **can** be used with a fragment density map attachment.

**NOTE** The behavior of VK_SHADER_CREATE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_EXT and VK_SHADER_CREATE_FRAGMENT_DENSITY_MAP_ATTACHMENT_BIT_EXT differs subtly from the behavior of VK_PIPELINE_CREATE_RENDERING_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR and VK_PIPELINE_CREATE_RENDERING_FRAGMENT_DENSITY_MAP_ATTACHMENT_BIT_EXT in that the shader bit allows, but does not require the shader to be used with that type of attachment. This means that the application need not create multiple shaders when it does not know in advance whether the shader will be used with or without the attachment type, or when it needs the same shader to be compatible with usage both with and without. This **may** come at some performance cost on some implementations, so applications **should** still only set bits that are actually necessary.

Shader objects **can** be created using different types of shader code. Possible values of VkShaderCreateInfoEXT::codeType, are:

```
// Provided by VK_EXT_shader_object
typedef enum VkShaderCodeTypeEXT {
    VK_SHADER_CODE_TYPE_BINARY_EXT = 0,
    VK_SHADER_CODE_TYPE_SPIRV_EXT = 1,
} VkShaderCodeTypeEXT;
```

- VK_SHADER_CODE_TYPE_BINARY_EXT specifies shader code in an opaque, implementation-defined binary format specific to the physical device.
- VK_SHADER_CODE_TYPE_SPIRV_EXT specifies shader code in SPIR-V format.

### 9.1.2. Binary Shader Code

Binary shader code **can** be retrieved from a shader object using the command:

```
// Provided by VK_EXT_shader_object
VkResult vkGetShaderBinaryDataEXT(
    VkDevice                                    device,
    VkShaderEXT                                 shader,
    size_t*                                     pDataSize,
    void*                                       pData);
```

- device is the logical device that shader object was created from.
- shader is the shader object to retrieve binary shader code from.
- pDataSize is a pointer to a size_t value related to the size of the binary shader code, as described below.
- pData is either NULL or a pointer to a buffer.

If pData is NULL, then the size of the binary shader code of the shader object, in bytes, is returned in pDataSize. Otherwise, pDataSize **must** point to a variable set by the application to the size of the buffer, in bytes, pointed to by pData, and on return the variable is overwritten with the amount of data actually written to pData. If pDataSize is less than the size of the binary shader code, nothing is written to pData, and VK_INCOMPLETE will be returned instead of VK_SUCCESS.

**NOTE** The behavior of this command when pDataSize is too small differs from how some other getter-type commands work in Vulkan. Because shader binary data is only usable in its entirety, it would never be useful for the implementation to return partial data. Because of this, nothing is written to pData unless pDataSize is large enough to fit the data in its entirety.

Binary shader code retrieved using vkGetShaderBinaryDataEXT **can** be passed to a subsequent call to vkCreateShadersEXT on a compatible physical device by specifying VK_SHADER_CODE_TYPE_BINARY_EXT in the codeType member of VkShaderCreateInfoEXT.

The shader code returned by repeated calls to this function with the same VkShaderEXT is guaranteed to be invariant for the lifetime of the VkShaderEXT object.

### Valid Usage

- VUID-vkGetShaderBinaryDataEXT-None-08461
  The shaderObject feature **must** be enabled
- VUID-vkGetShaderBinaryDataEXT-None-08499
  If pData is not NULL, it **must** be aligned to 16 bytes

### Valid Usage (Implicit)

- VUID-vkGetShaderBinaryDataEXT-device-parameter
  device **must** be a valid VkDevice handle
- VUID-vkGetShaderBinaryDataEXT-shader-parameter
  shader **must** be a valid VkShaderEXT handle
- VUID-vkGetShaderBinaryDataEXT-pDataSize-parameter
  pDataSize **must** be a valid pointer to a size_t value
- VUID-vkGetShaderBinaryDataEXT-pData-parameter
  If the value referenced by pDataSize is not 0, and pData is not NULL, pData **must** be a valid pointer to an array of pDataSize bytes
- VUID-vkGetShaderBinaryDataEXT-shader-parent

shader **must** have been created, allocated, or retrieved from device

**Return Codes**

**Success**

- VK_SUCCESS
- VK_INCOMPLETE

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY

### 9.1.3. Binary Shader Compatibility

Binary shader compatibility means that binary shader code returned from a call to vkGetShaderBinaryDataEXT **can** be passed to a later call to vkCreateShadersEXT, potentially on a different logical and/or physical device, and that this will result in the successful creation of a shader object functionally equivalent to the shader object that the code was originally queried from.

Binary shader code queried from vkGetShaderBinaryDataEXT is not guaranteed to be compatible across all devices, but implementations are required to provide some compatibility guarantees. Applications **may** determine binary shader compatibility using either (or both) of two mechanisms.

Guaranteed compatibility of shader binaries is expressed through a combination of the shaderBinaryUUID and shaderBinaryVersion members of the VkPhysicalDeviceShaderObjectPropertiesEXT structure queried from a physical device. Binary shaders retrieved from a physical device with a certain shaderBinaryUUID are guaranteed to be compatible with all other physical devices reporting the same shaderBinaryUUID and the same or higher shaderBinaryVersion.

Whenever a new version of an implementation incorporates any changes that affect the output of vkGetShaderBinaryDataEXT, the implementation **should** either increment shaderBinaryVersion if binary shader code retrieved from older versions remains compatible with the new implementation, or else replace shaderBinaryUUID with a new value if backward compatibility has been broken. Binary shader code queried from a device with a matching shaderBinaryUUID and lower shaderBinaryVersion relative to the device on which vkCreateShadersEXT is being called **may** be suboptimal for the new device in ways that do not change shader functionality, but it is still guaranteed to be usable to successfully create the shader object(s).

| | |
|---|---|
| **NOTE** | Implementations are encouraged to share shaderBinaryUUID between devices and driver versions to the maximum extent their hardware naturally allows, and are **strongly** discouraged from ever changing the shaderBinaryUUID for the same hardware except unless absolutely necessary. |

In addition to the shader compatibility guarantees described above, it is valid for an application to

call vkCreateShadersEXT with binary shader code created on a device with a different or unknown shaderBinaryUUID and/or higher shaderBinaryVersion. In this case, the implementation **may** use any unspecified means of its choosing to determine whether the provided binary shader code is usable. If it is, vkCreateShadersEXT **must** return VK_SUCCESS, and the created shader object is guaranteed to be valid. Otherwise, in the absence of some error, vkCreateShadersEXT **must** return VK_INCOMPATIBLE_SHADER_BINARY_EXT to indicate that the provided binary shader code is not compatible with the device.

### 9.1.4. Binding Shader Objects

Once shader objects have been created, they **can** be bound to the command buffer using the command:

```
// Provided by VK_EXT_shader_object
void vkCmdBindShadersEXT(
    VkCommandBuffer                             commandBuffer,
    uint32_t                                    stageCount,
    const VkShaderStageFlagBits*                pStages,
    const VkShaderEXT*                          pShaders);
```

- commandBuffer is the command buffer that the shader object will be bound to.
- stageCount is the length of the pStages and pShaders arrays.
- pStages is a pointer to an array of VkShaderStageFlagBits values specifying one stage per array index that is affected by the corresponding value in the pShaders array.
- pShaders is a pointer to an array of VkShaderEXT handles and/or VK_NULL_HANDLE values describing the shader binding operations to be performed on each stage in pStages.

When binding linked shaders, an application **may** bind them in any combination of one or more calls to vkCmdBindShadersEXT (i.e., shaders that were created linked together do not need to be bound in the same vkCmdBindShadersEXT call).

Any shader object bound to a particular stage **may** be unbound by setting its value in pShaders to VK_NULL_HANDLE. If pShaders is NULL, vkCmdBindShadersEXT behaves as if pShaders was an array of stageCount VK_NULL_HANDLE values (i.e., any shaders bound to the stages specified in pStages are unbound).

#### Valid Usage

- VUID-vkCmdBindShadersEXT-None-08462
  The shaderObject feature **must** be enabled
- VUID-vkCmdBindShadersEXT-pStages-08463
  Every element of pStages **must** be unique
- VUID-vkCmdBindShadersEXT-pStages-08464
  pStages **must** not contain VK_SHADER_STAGE_ALL_GRAPHICS or VK_SHADER_STAGE_ALL
- VUID-vkCmdBindShadersEXT-pStages-08465

pStages **must** not contain VK_SHADER_STAGE_RAYGEN_BIT_KHR, VK_SHADER_STAGE_ANY_HIT_BIT_KHR, VK_SHADER_STAGE_CLOSEST_HIT_BIT_KHR, VK_SHADER_STAGE_MISS_BIT_KHR, VK_SHADER_STAGE_INTERSECTION_BIT_KHR, or VK_SHADER_STAGE_CALLABLE_BIT_KHR

- VUID-vkCmdBindShadersEXT-pStages-08467
  pStages **must** not contain VK_SHADER_STAGE_SUBPASS_SHADING_BIT_HUAWEI
- VUID-vkCmdBindShadersEXT-pStages-08468
  pStages **must** not contain VK_SHADER_STAGE_CLUSTER_CULLING_BIT_HUAWEI
- VUID-vkCmdBindShadersEXT-pShaders-08469
  For each element of pStages, if pShaders is not NULL, and the element of the pShaders array with the same index is not VK_NULL_HANDLE, it **must** have been created with a stage equal to the corresponding element of pStages
- VUID-vkCmdBindShadersEXT-pShaders-08470
  If pStages contains both VK_SHADER_STAGE_TASK_BIT_EXT and VK_SHADER_STAGE_VERTEX_BIT, and pShaders is not NULL, and the same index in pShaders as VK_SHADER_STAGE_TASK_BIT_EXT in pStages is not VK_NULL_HANDLE, the same index in pShaders as VK_SHADER_STAGE_VERTEX_BIT in pStages **must** be VK_NULL_HANDLE
- VUID-vkCmdBindShadersEXT-pShaders-08471
  If pStages contains both VK_SHADER_STAGE_MESH_BIT_EXT and VK_SHADER_STAGE_VERTEX_BIT, and pShaders is not NULL, and the same index in pShaders as VK_SHADER_STAGE_MESH_BIT_EXT in pStages is not VK_NULL_HANDLE, the same index in pShaders as VK_SHADER_STAGE_VERTEX_BIT in pStages **must** be VK_NULL_HANDLE
- VUID-vkCmdBindShadersEXT-pShaders-08474
  If the tessellationShader feature is not enabled, and pStages contains VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT or VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT, and pShaders is not NULL, the same index or indices in pShaders **must** be VK_NULL_HANDLE
- VUID-vkCmdBindShadersEXT-pShaders-08475
  If the geometryShader feature is not enabled, and pStages contains VK_SHADER_STAGE_GEOMETRY_BIT, and pShaders is not NULL, the same index in pShaders **must** be VK_NULL_HANDLE
- VUID-vkCmdBindShadersEXT-pShaders-08490
  If the taskShader feature is not enabled, and pStages contains VK_SHADER_STAGE_TASK_BIT_EXT, and pShaders is not NULL, the same index in pShaders **must** be VK_NULL_HANDLE
- VUID-vkCmdBindShadersEXT-pShaders-08491
  If the meshShader feature is not enabled, and pStages contains VK_SHADER_STAGE_MESH_BIT_EXT, and pShaders is not NULL, the same index in pShaders **must** be VK_NULL_HANDLE
- VUID-vkCmdBindShadersEXT-pShaders-08476
  If pStages contains VK_SHADER_STAGE_COMPUTE_BIT, the VkCommandPool that commandBuffer was allocated from **must** support compute operations
- VUID-vkCmdBindShadersEXT-pShaders-08477

If `pStages` contains `VK_SHADER_STAGE_VERTEX_BIT`, `VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT`, `VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT`, `VK_SHADER_STAGE_GEOMETRY_BIT`, or `VK_SHADER_STAGE_FRAGMENT_BIT`, the `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations

- VUID-vkCmdBindShadersEXT-pShaders-08478
  If `pStages` contains `VK_SHADER_STAGE_MESH_BIT_EXT` or `VK_SHADER_STAGE_TASK_BIT_EXT`, the `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations

## Valid Usage (Implicit)

- VUID-vkCmdBindShadersEXT-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdBindShadersEXT-pStages-parameter
  `pStages` **must** be a valid pointer to an array of `stageCount` valid VkShaderStageFlagBits values
- VUID-vkCmdBindShadersEXT-pShaders-parameter
  If `pShaders` is not `NULL`, `pShaders` **must** be a valid pointer to an array of `stageCount` valid or VK_NULL_HANDLE VkShaderEXT handles
- VUID-vkCmdBindShadersEXT-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdBindShadersEXT-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics, or compute operations
- VUID-vkCmdBindShadersEXT-videocoding
  This command **must** only be called outside of a video coding scope
- VUID-vkCmdBindShadersEXT-stageCount-arraylength
  `stageCount` **must** be greater than `0`
- VUID-vkCmdBindShadersEXT-commonparent
  Both of `commandBuffer`, and the elements of `pShaders` that are valid handles of non-ignored parameters **must** have been created, allocated, or retrieved from the same VkDevice

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

**Command Properties**

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics<br>Compute | State |

### 9.1.5. Setting State

Whenever shader objects are used to issue drawing commands, the appropriate dynamic state setting commands **must** have been called to set the relevant state in the command buffer prior to drawing:

- vkCmdSetViewportWithCountEXT
- vkCmdSetScissorWithCountEXT
- vkCmdSetRasterizerDiscardEnableEXT

If a shader is bound to the `VK_SHADER_STAGE_VERTEX_BIT` stage, the following commands **must** have been called in the command buffer prior to drawing:

- vkCmdSetVertexInputEXT
- vkCmdSetPrimitiveTopologyEXT
- vkCmdSetPatchControlPointsEXT, if `primitiveTopology` is `VK_PRIMITIVE_TOPOLOGY_PATCH_LIST`
- vkCmdSetPrimitiveRestartEnableEXT

If a shader is bound to the `VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT` stage, the following command **must** have been called in the command buffer prior to drawing:

- vkCmdSetTessellationDomainOriginEXT

If `rasterizerDiscardEnable` is `VK_FALSE`, the following commands **must** have been called in the command buffer prior to drawing:

- vkCmdSetRasterizationSamplesEXT
- vkCmdSetSampleMaskEXT
- vkCmdSetAlphaToCoverageEnableEXT
- vkCmdSetAlphaToOneEnableEXT, if the alphaToOne feature is enabled on the device
- vkCmdSetPolygonModeEXT
- vkCmdSetLineWidth, if `polygonMode` is `VK_POLYGON_MODE_LINE`, or if a shader is bound to the `VK_SHADER_STAGE_VERTEX_BIT` stage and `primitiveTopology` is a line topology, or if a shader which outputs line primitives is bound to the `VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT` or `VK_SHADER_STAGE_GEOMETRY_BIT` stage
- vkCmdSetCullModeEXT

- vkCmdSetFrontFaceEXT
- vkCmdSetDepthTestEnableEXT
- vkCmdSetDepthWriteEnableEXT
- vkCmdSetDepthCompareOpEXT, if `depthTestEnable` is `VK_TRUE`
- vkCmdSetDepthBoundsTestEnableEXT, if the depthBounds feature is enabled on the device
- vkCmdSetDepthBounds, if `depthBoundsTestEnable` is `VK_TRUE`
- vkCmdSetDepthBiasEnableEXT
- vkCmdSetDepthBias or vkCmdSetDepthBias2EXT, if `depthBiasEnable` is `VK_TRUE`
- vkCmdSetDepthClampEnableEXT, if the depthClamp feature is enabled on the device
- vkCmdSetStencilTestEnableEXT
- vkCmdSetStencilOpEXT, if `stencilTestEnable` is `VK_TRUE`
- vkCmdSetStencilCompareMask, if `stencilTestEnable` is `VK_TRUE`
- vkCmdSetStencilWriteMask, if `stencilTestEnable` is `VK_TRUE`
- vkCmdSetStencilReference, if `stencilTestEnable` is `VK_TRUE`

If a shader is bound to the `VK_SHADER_STAGE_FRAGMENT_BIT` stage, and `rasterizerDiscardEnable` is `VK_FALSE`, the following commands **must** have been called in the command buffer prior to drawing:

- vkCmdSetLogicOpEnableEXT, if the `logicOp` feature is enabled on the device
- vkCmdSetLogicOpEXT, if `logicOpEnable` is `VK_TRUE`
- vkCmdSetColorBlendEnableEXT and vkCmdSetColorWriteMaskEXT, if color attachments are bound, with values set for every color attachment in the render pass instance active at draw time
- vkCmdSetColorBlendEquationEXT or vkCmdSetColorBlendAdvancedEXT, if color attachments are bound, for every attachment whose index in `pColorBlendEnables` is a pointer to a value of `VK_TRUE`
- vkCmdSetBlendConstants, if any index in `pColorBlendEnables` is `VK_TRUE`, and the same index in `pColorBlendEquations` is a `VkColorBlendEquationEXT` structure with any VkBlendFactor member with a value of `VK_BLEND_FACTOR_CONSTANT_COLOR`, `VK_BLEND_FACTOR_ONE_MINUS_CONSTANT_COLOR`, `VK_BLEND_FACTOR_CONSTANT_ALPHA`, or `VK_BLEND_FACTOR_ONE_MINUS_CONSTANT_ALPHA`

If the `pipelineFragmentShadingRate` feature is enabled on the device, and a shader is bound to the `VK_SHADER_STAGE_FRAGMENT_BIT` stage, and `rasterizerDiscardEnable` is `VK_FALSE`, the following command **must** have been called in the command buffer prior to drawing:

- vkCmdSetFragmentShadingRateKHR

If the `geometryStreams` feature is enabled on the device, and a shader is bound to the `VK_SHADER_STAGE_GEOMETRY_BIT` stage, the following command **must** have been called in the command buffer prior to drawing:

- vkCmdSetRasterizationStreamEXT

If the VK_EXT_discard_rectangles extension is enabled on the device, and rasterizerDiscardEnable is VK_FALSE, the following commands **must** have been called in the command buffer prior to drawing:

- vkCmdSetDiscardRectangleEnableEXT
- vkCmdSetDiscardRectangleModeEXT, if discardRectangleEnable is VK_TRUE
- vkCmdSetDiscardRectangleEXT, if discardRectangleEnable is VK_TRUE

If VK_EXT_conservative_rasterization extension is enabled on the device, and rasterizerDiscardEnable is VK_FALSE, the following commands **must** have been called in the command buffer prior to drawing:

- vkCmdSetConservativeRasterizationModeEXT
- vkCmdSetExtraPrimitiveOverestimationSizeEXT, if conservativeRasterizationMode is VK_CONSERVATIVE_RASTERIZATION_MODE_OVERESTIMATE_EXT

If the depthClipEnable feature is enabled on the device, the following command **must** have been called in the command buffer prior to drawing:

- vkCmdSetDepthClipEnableEXT

If the VK_EXT_sample_locations extension is enabled on the device, and rasterizerDiscardEnable is VK_FALSE, the following commands **must** have been called in the command buffer prior to drawing:

- vkCmdSetSampleLocationsEnableEXT
- vkCmdSetSampleLocationsEXT, if sampleLocationsEnable is VK_TRUE

If the VK_EXT_provoking_vertex extension is enabled on the device, and rasterizerDiscardEnable is VK_FALSE, and a shader is bound to the VK_SHADER_STAGE_VERTEX_BIT stage, the following command **must** have been called in the command buffer prior to drawing:

- vkCmdSetProvokingVertexModeEXT

If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled on the device, and rasterizerDiscardEnable is VK_FALSE, and if polygonMode is VK_POLYGON_MODE_LINE or a shader is bound to the VK_SHADER_STAGE_VERTEX_BIT stage and primitiveTopology is a line topology or a shader which outputs line primitives is bound to the VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT or VK_SHADER_STAGE_GEOMETRY_BIT stage, the following commands **must** have been called in the command buffer prior to drawing:

- vkCmdSetLineRasterizationModeEXT
- vkCmdSetLineStippleEnableEXT
- vkCmdSetLineStippleKHR, if stippledLineEnable is VK_TRUE

If the depthClipControl feature is enabled on the device, the following command **must** have been called in the command buffer prior to drawing:

- vkCmdSetDepthClipNegativeOneToOneEXT

If the colorWriteEnable feature is enabled on the device, and a shader is bound to the

VK_SHADER_STAGE_FRAGMENT_BIT stage, and rasterizerDiscardEnable is VK_FALSE, the following command **must** have been called in the command buffer prior to drawing:

- vkCmdSetColorWriteEnableEXT, with values set for every color attachment in the render pass instance active at draw time

If the attachmentFeedbackLoopDynamicState feature is enabled on the device, and a shader is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and rasterizerDiscardEnable is VK_FALSE, the following command **must** have been called in the command buffer prior to drawing:

- vkCmdSetAttachmentFeedbackLoopEnableEXT

If the VK_NV_clip_space_w_scaling extension is enabled on the device, the following commands **must** have been called in the command buffer prior to drawing:

- vkCmdSetViewportWScalingEnableNV
- vkCmdSetViewportWScalingNV, if viewportWScalingEnable is VK_TRUE

If the VK_NV_viewport_swizzle extension is enabled on the device, the following command **must** have been called in the command buffer prior to drawing:

- vkCmdSetViewportSwizzleNV

If the VK_NV_fragment_coverage_to_color extension is enabled on the device, and a shader is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and rasterizerDiscardEnable is VK_FALSE, the following commands **must** have been called in the command buffer prior to drawing:

- vkCmdSetCoverageToColorEnableNV
- vkCmdSetCoverageToColorLocationNV, if coverageToColorEnable is VK_TRUE

If the VK_NV_framebuffer_mixed_samples extension is enabled on the device, and rasterizerDiscardEnable is VK_FALSE, the following commands **must** have been called in the command buffer prior to drawing:

- vkCmdSetCoverageModulationModeNV
- vkCmdSetCoverageModulationTableEnableNV, if coverageModulationMode is not VK_COVERAGE_MODULATION_MODE_NONE_NV
- vkCmdSetCoverageModulationTableNV, if coverageModulationTableEnable is VK_TRUE

If the coverageReductionMode feature is enabled on the device, and rasterizerDiscardEnable is VK_FALSE, the following command **must** have been called in the command buffer prior to drawing:

- vkCmdSetCoverageReductionModeNV

If the representativeFragmentTest feature is enabled on the device, and rasterizerDiscardEnable is VK_FALSE, the following command **must** have been called in the command buffer prior to drawing:

- vkCmdSetRepresentativeFragmentTestEnableNV

If the shadingRateImage feature is enabled on the device, and rasterizerDiscardEnable is VK_FALSE,

the following commands **must** have been called in the command buffer prior to drawing:

- vkCmdSetCoarseSampleOrderNV
- vkCmdSetShadingRateImageEnableNV
- vkCmdSetViewportShadingRatePaletteNV, if `shadingRateImageEnable` is `VK_TRUE`

If the `exclusiveScissor` feature is enabled on the device, the following commands **must** have been called in the command buffer prior to drawing:

- vkCmdSetExclusiveScissorEnableNV
- vkCmdSetExclusiveScissorNV, if any value in `pExclusiveScissorEnables` is `VK_TRUE`

State **can** be set either at any time before or after shader objects are bound, but all required state **must** be set prior to issuing drawing commands.

If the `commandBufferInheritance` feature is enabled, graphics and compute state is inherited from the previously executed command buffer in the queue. Any valid state inherited in this way does not need to be set again in the current command buffer.

### 9.1.6. Interaction With Pipelines

Calling vkCmdBindShadersEXT causes the pipeline bind points corresponding to each stage in `pStages` to be disturbed, meaning that any pipelines that had previously been bound to those pipeline bind points are no longer bound.

If `VK_PIPELINE_BIND_POINT_GRAPHICS` is disturbed (i.e., if `pStages` contains any graphics stage), any graphics pipeline state that the previously bound pipeline did not specify as dynamic becomes undefined, and **must** be set in the command buffer before issuing drawing commands using shader objects.

Calls to vkCmdBindPipeline likewise disturb the shader stage(s) corresponding to `pipelineBindPoint`, meaning that any shaders that had previously been bound to any of those stages are no longer bound, even if the pipeline was created without shaders for some of those stages.

### 9.1.7. Shader Object Destruction

To destroy a shader object, call:

```
// Provided by VK_EXT_shader_object
void vkDestroyShaderEXT(
    VkDevice                                    device,
    VkShaderEXT                                 shader,
    const VkAllocationCallbacks*                pAllocator);
```

- `device` is the logical device that destroys the shader object.
- `shader` is the handle of the shader object to destroy.
- `pAllocator` controls host memory allocation as described in the Memory Allocation chapter.

Destroying a shader object used by one or more command buffers in the recording or executable state causes those command buffers to move into the *invalid state*.

### Valid Usage

- VUID-vkDestroyShaderEXT-None-08481
  The `shaderObject` feature **must** be enabled
- VUID-vkDestroyShaderEXT-shader-08482
  All submitted commands that refer to `shader` **must** have completed execution
- VUID-vkDestroyShaderEXT-pAllocator-08483
  If `VkAllocationCallbacks` were provided when `shader` was created, a compatible set of callbacks **must** be provided here
- VUID-vkDestroyShaderEXT-pAllocator-08484
  If no `VkAllocationCallbacks` were provided when `shader` was created, `pAllocator` **must** be `NULL`

### Valid Usage (Implicit)

- VUID-vkDestroyShaderEXT-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkDestroyShaderEXT-shader-parameter
  If `shader` is not VK_NULL_HANDLE, `shader` **must** be a valid VkShaderEXT handle
- VUID-vkDestroyShaderEXT-pAllocator-parameter
  If `pAllocator` is not `NULL`, `pAllocator` **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkDestroyShaderEXT-shader-parent
  If `shader` is a valid handle, it **must** have been created, allocated, or retrieved from `device`

### Host Synchronization

- Host access to `shader` **must** be externally synchronized

## 9.2. Shader Modules

*Shader modules* contain *shader code* and one or more entry points. Shaders are selected from a shader module by specifying an entry point as part of pipeline creation. The stages of a pipeline **can** use shaders that come from different modules. The shader code defining a shader module **must** be in the SPIR-V format, as described by the Vulkan Environment for SPIR-V appendix.

Shader modules are represented by `VkShaderModule` handles:

```
// Provided by VK_VERSION_1_0
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkShaderModule)
```

To create a shader module, call:

```
// Provided by VK_VERSION_1_0
VkResult vkCreateShaderModule(
    VkDevice                                    device,
    const VkShaderModuleCreateInfo*             pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkShaderModule*                             pShaderModule);
```

- `device` is the logical device that creates the shader module.
- `pCreateInfo` is a pointer to a VkShaderModuleCreateInfo structure.
- `pAllocator` controls host memory allocation as described in the Memory Allocation chapter.
- `pShaderModule` is a pointer to a VkShaderModule handle in which the resulting shader module object is returned.

Once a shader module has been created, any entry points it contains **can** be used in pipeline shader stages as described in Compute Pipelines and Graphics Pipelines.

**NOTE** If the `maintenance5` feature is enabled, shader module creation can be omitted entirely. Instead, applications should provide the VkShaderModuleCreateInfo structure directly in to pipeline creation by chaining it to VkPipelineShaderStageCreateInfo. This avoids the overhead of creating and managing an additional object.

### Valid Usage

- VUID-vkCreateShaderModule-pCreateInfo-06904
  If `pCreateInfo` is not `NULL`, `pCreateInfo->pNext` **must** be `NULL` or a pointer to a valid instance of
  - VkShaderModuleValidationCacheCreateInfoEXT
  - VkValidationFeaturesEXT

### Valid Usage (Implicit)

- VUID-vkCreateShaderModule-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkCreateShaderModule-pCreateInfo-parameter
  `pCreateInfo` **must** be a valid pointer to a valid VkShaderModuleCreateInfo structure
- VUID-vkCreateShaderModule-pAllocator-parameter

If `pAllocator` is not `NULL`, `pAllocator` **must** be a valid pointer to a valid VkAllocationCallbacks structure

- VUID-vkCreateShaderModule-pShaderModule-parameter
  `pShaderModule` **must** be a valid pointer to a VkShaderModule handle

## Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_INVALID_SHADER_NV`

The `VkShaderModuleCreateInfo` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkShaderModuleCreateInfo {
    VkStructureType              sType;
    const void*                  pNext;
    VkShaderModuleCreateFlags    flags;
    size_t                       codeSize;
    const uint32_t*              pCode;
} VkShaderModuleCreateInfo;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is reserved for future use.
- `codeSize` is the size, in bytes, of the code pointed to by `pCode`.
- `pCode` is a pointer to code that is used to create the shader module. The type and format of the code is determined from the content of the memory addressed by `pCode`.

## Valid Usage

- VUID-VkShaderModuleCreateInfo-codeSize-08735
  If pCode is a pointer to SPIR-V code, `codeSize` **must** be a multiple of 4
- VUID-VkShaderModuleCreateInfo-pCode-08736
  If pCode is a pointer to SPIR-V code, `pCode` **must** point to valid SPIR-V code, formatted and packed as described by the Khronos SPIR-V Specification
- VUID-VkShaderModuleCreateInfo-pCode-08737
  If pCode is a pointer to SPIR-V code, `pCode` **must** adhere to the validation rules described

by the Validation Rules within a Module section of the SPIR-V Environment appendix

- VUID-VkShaderModuleCreateInfo-pCode-08738
  If pCode is a pointer to SPIR-V code, `pCode` **must** declare the `Shader` capability for SPIR-V code
- VUID-VkShaderModuleCreateInfo-pCode-08739
  If pCode is a pointer to SPIR-V code, `pCode` **must** not declare any capability that is not supported by the API, as described by the Capabilities section of the SPIR-V Environment appendix
- VUID-VkShaderModuleCreateInfo-pCode-08740
  If pCode is a pointer to SPIR-V code, and `pCode` declares any of the capabilities listed in the SPIR-V Environment appendix, one of the corresponding requirements **must** be satisfied
- VUID-VkShaderModuleCreateInfo-pCode-08741
  If pCode is a pointer to SPIR-V code, `pCode` **must** not declare any SPIR-V extension that is not supported by the API, as described by the Extension section of the SPIR-V Environment appendix
- VUID-VkShaderModuleCreateInfo-pCode-08742
  If pCode is a pointer to SPIR-V code, and `pCode` declares any of the SPIR-V extensions listed in the SPIR-V Environment appendix, one of the corresponding requirements **must** be satisfied
- VUID-VkShaderModuleCreateInfo-pCode-07912
  If the VK_NV_glsl_shader extension is not enabled, `pCode` **must** be a pointer to SPIR-V code
- VUID-VkShaderModuleCreateInfo-pCode-01379
  If `pCode` is a pointer to GLSL code, it **must** be valid GLSL code written to the `GL_KHR_vulkan_glsl` GLSL extension specification
- VUID-VkShaderModuleCreateInfo-codeSize-01085
  `codeSize` **must** be greater than 0

## Valid Usage (Implicit)

- VUID-VkShaderModuleCreateInfo-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_SHADER_MODULE_CREATE_INFO`
- VUID-VkShaderModuleCreateInfo-flags-zerobitmask
  `flags` **must** be `0`
- VUID-VkShaderModuleCreateInfo-pCode-parameter
  `pCode` **must** be a valid pointer to an array of $\frac{\text{codeSize}}{4}$ `uint32_t` values

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkShaderModuleCreateFlags;
```

`VkShaderModuleCreateFlags` is a bitmask type for setting a mask, but is currently reserved for future use.

To use a VkValidationCacheEXT to cache shader validation results, add a VkShaderModuleValidationCacheCreateInfoEXT structure to the pNext chain of the VkShaderModuleCreateInfo structure, specifying the cache object to use.

The VkShaderModuleValidationCacheCreateInfoEXT struct is defined as:

```
// Provided by VK_EXT_validation_cache
typedef struct VkShaderModuleValidationCacheCreateInfoEXT {
    VkStructureType         sType;
    const void*             pNext;
    VkValidationCacheEXT    validationCache;
} VkShaderModuleValidationCacheCreateInfoEXT;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- validationCache is the validation cache object from which the results of prior validation attempts will be written, and to which new validation results for this VkShaderModule will be written (if not already present).

### Valid Usage (Implicit)

- VUID-VkShaderModuleValidationCacheCreateInfoEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_SHADER_MODULE_VALIDATION_CACHE_CREATE_INFO_EXT
- VUID-VkShaderModuleValidationCacheCreateInfoEXT-validationCache-parameter
  validationCache **must** be a valid VkValidationCacheEXT handle

To destroy a shader module, call:

```
// Provided by VK_VERSION_1_0
void vkDestroyShaderModule(
    VkDevice                                    device,
    VkShaderModule                              shaderModule,
    const VkAllocationCallbacks*                pAllocator);
```

- device is the logical device that destroys the shader module.
- shaderModule is the handle of the shader module to destroy.
- pAllocator controls host memory allocation as described in the Memory Allocation chapter.

A shader module **can** be destroyed while pipelines created using its shaders are still in use.

### Valid Usage

- VUID-vkDestroyShaderModule-shaderModule-01092
  If VkAllocationCallbacks were provided when shaderModule was created, a compatible set

of callbacks **must** be provided here

- VUID-vkDestroyShaderModule-shaderModule-01093
  If no `VkAllocationCallbacks` were provided when `shaderModule` was created, `pAllocator` **must** be `NULL`

### Valid Usage (Implicit)

- VUID-vkDestroyShaderModule-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkDestroyShaderModule-shaderModule-parameter
  If `shaderModule` is not VK_NULL_HANDLE, `shaderModule` **must** be a valid VkShaderModule handle
- VUID-vkDestroyShaderModule-pAllocator-parameter
  If `pAllocator` is not `NULL`, `pAllocator` **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkDestroyShaderModule-shaderModule-parent
  If `shaderModule` is a valid handle, it **must** have been created, allocated, or retrieved from `device`

### Host Synchronization

- Host access to `shaderModule` **must** be externally synchronized

## 9.3. Shader Module Identifiers

Shader modules have unique identifiers associated with them. To query an implementation provided identifier, call:

```
// Provided by VK_EXT_shader_module_identifier
void vkGetShaderModuleIdentifierEXT(
    VkDevice                                    device,
    VkShaderModule                              shaderModule,
    VkShaderModuleIdentifierEXT*                pIdentifier);
```

- `device` is the logical device that created the shader module.
- `shaderModule` is the handle of the shader module.
- `pIdentifier` is a pointer to the returned VkShaderModuleIdentifierEXT.

The identifier returned by the implementation **must** only depend on `shaderIdentifierAlgorithmUUID` and information provided in the VkShaderModuleCreateInfo which created `shaderModule`. The implementation **may** return equal identifiers for two different VkShaderModuleCreateInfo structures if the difference does not affect pipeline compilation. Identifiers are only meaningful on

different VkDevice objects if the device the identifier was queried from had the same shaderModuleIdentifierAlgorithmUUID as the device consuming the identifier.

### Valid Usage

- VUID-vkGetShaderModuleIdentifierEXT-shaderModuleIdentifier-06884
  shaderModuleIdentifier feature **must** be enabled

### Valid Usage (Implicit)

- VUID-vkGetShaderModuleIdentifierEXT-device-parameter
  device **must** be a valid VkDevice handle
- VUID-vkGetShaderModuleIdentifierEXT-shaderModule-parameter
  shaderModule **must** be a valid VkShaderModule handle
- VUID-vkGetShaderModuleIdentifierEXT-pIdentifier-parameter
  pIdentifier **must** be a valid pointer to a VkShaderModuleIdentifierEXT structure
- VUID-vkGetShaderModuleIdentifierEXT-shaderModule-parent
  shaderModule **must** have been created, allocated, or retrieved from device

VkShaderModuleCreateInfo structures have unique identifiers associated with them. To query an implementation provided identifier, call:

```
// Provided by VK_EXT_shader_module_identifier
void vkGetShaderModuleCreateInfoIdentifierEXT(
    VkDevice                                    device,
    const VkShaderModuleCreateInfo*             pCreateInfo,
    VkShaderModuleIdentifierEXT*                pIdentifier);
```

- device is the logical device that **can** create a VkShaderModule from pCreateInfo.
- pCreateInfo is a pointer to a VkShaderModuleCreateInfo structure.
- pIdentifier is a pointer to the returned VkShaderModuleIdentifierEXT.

The identifier returned by implementation **must** only depend on shaderIdentifierAlgorithmUUID and information provided in the VkShaderModuleCreateInfo. The implementation **may** return equal identifiers for two different VkShaderModuleCreateInfo structures if the difference does not affect pipeline compilation. Identifiers are only meaningful on different VkDevice objects if the device the identifier was queried from had the same shaderModuleIdentifierAlgorithmUUID as the device consuming the identifier.

The identifier returned by the implementation in vkGetShaderModuleCreateInfoIdentifierEXT **must** be equal to the identifier returned by vkGetShaderModuleIdentifierEXT given equivalent definitions of VkShaderModuleCreateInfo and any chained pNext structures.

### Valid Usage

- VUID-vkGetShaderModuleCreateInfoIdentifierEXT-shaderModuleIdentifier-06885 `shaderModuleIdentifier` feature **must** be enabled

### Valid Usage (Implicit)

- VUID-vkGetShaderModuleCreateInfoIdentifierEXT-device-parameter `device` **must** be a valid VkDevice handle
- VUID-vkGetShaderModuleCreateInfoIdentifierEXT-pCreateInfo-parameter `pCreateInfo` **must** be a valid pointer to a valid VkShaderModuleCreateInfo structure
- VUID-vkGetShaderModuleCreateInfoIdentifierEXT-pIdentifier-parameter `pIdentifier` **must** be a valid pointer to a VkShaderModuleIdentifierEXT structure

VkShaderModuleIdentifierEXT represents a shader module identifier returned by the implementation.

```
// Provided by VK_EXT_shader_module_identifier
typedef struct VkShaderModuleIdentifierEXT {
    VkStructureType    sType;
    void*              pNext;
    uint32_t           identifierSize;
    uint8_t            identifier[VK_MAX_SHADER_MODULE_IDENTIFIER_SIZE_EXT];
} VkShaderModuleIdentifierEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `identifierSize` is the size, in bytes, of valid data returned in `identifier`.
- `identifier` is a buffer of opaque data specifying an identifier.

Any returned values beyond the first `identifierSize` bytes are undefined. Implementations **must** return an `identifierSize` greater than 0, and less-or-equal to `VK_MAX_SHADER_MODULE_IDENTIFIER_SIZE_EXT`.

Two identifiers are considered equal if `identifierSize` is equal and the first `identifierSize` bytes of `identifier` compare equal.

Implementations **may** return a different `identifierSize` for different modules. Implementations **should** ensure that `identifierSize` is large enough to uniquely define a shader module.

### Valid Usage (Implicit)

- VUID-VkShaderModuleIdentifierEXT-sType-sType

sType **must** be VK_STRUCTURE_TYPE_SHADER_MODULE_IDENTIFIER_EXT

- VUID-VkShaderModuleIdentifierEXT-pNext-pNext
  pNext **must** be NULL

VK_MAX_SHADER_MODULE_IDENTIFIER_SIZE_EXT is the length in bytes of a shader module identifier, as returned in VkShaderModuleIdentifierEXT::identifierSize.

```
#define VK_MAX_SHADER_MODULE_IDENTIFIER_SIZE_EXT 32U
```

## 9.4. Binding Shaders

Before a shader can be used it **must** be first bound to the command buffer.

Calling vkCmdBindPipeline binds all stages corresponding to the VkPipelineBindPoint. Calling vkCmdBindShadersEXT binds all stages in pStages

The following table describes the relationship between shader stages and pipeline bind points:

| Shader stage | Pipeline bind point | behavior controlled |
|---|---|---|
| • VK_SHADER_STAGE_VERTEX_BIT<br>• VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT<br>• VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT<br>• VK_SHADER_STAGE_GEOMETRY_BIT<br>• VK_SHADER_STAGE_FRAGMENT_BIT<br>• VK_SHADER_STAGE_TASK_BIT_EXT<br>• VK_SHADER_STAGE_MESH_BIT_EXT | VK_PIPELINE_BIND_POINT_GRAPHICS | all drawing commands |
| • VK_SHADER_STAGE_COMPUTE_BIT | VK_PIPELINE_BIND_POINT_COMPUTE | all dispatch commands |

| Shader stage | Pipeline bind point | behavior controlled |
|---|---|---|
| • VK_SHADER_STAGE_ANY_HIT_BIT_KHR<br>• VK_SHADER_STAGE_CALLABLE_BIT_KHR<br>• VK_SHADER_STAGE_CLOSEST_HIT_BIT_KHR<br>• VK_SHADER_STAGE_INTERSECTION_BIT_KHR<br>• VK_SHADER_STAGE_MISS_BIT_KHR<br>• VK_SHADER_STAGE_RAYGEN_BIT_KHR | VK_PIPELINE_BIND_POINT_RAY_TRACING_KHR | vkCmdTraceRaysKHR and vkCmdTraceRaysIndirectKHR |
| • VK_SHADER_STAGE_SUBPASS_SHADING_BIT_HUAWEI<br>• VK_SHADER_STAGE_CLUSTER_CULLING_BIT_HUAWEI | VK_PIPELINE_BIND_POINT_SUBPASS_SHADING_HUAWEI | vkCmdSubpassShadingHUAWEI |
| • VK_SHADER_STAGE_COMPUTE_BIT | VK_PIPELINE_BIND_POINT_EXECUTION_GRAPH_AMDX | all execution graph commands |

# 9.5. Shader Execution

At each stage of the pipeline, multiple invocations of a shader **may** execute simultaneously. Further, invocations of a single shader produced as the result of different commands **may** execute simultaneously. The relative execution order of invocations of the same shader type is undefined. Shader invocations **may** complete in a different order than that in which the primitives they originated from were drawn or dispatched by the application. However, fragment shader outputs are written to attachments in rasterization order.

The relative execution order of invocations of different shader types is largely undefined. However, when invoking a shader whose inputs are generated from a previous pipeline stage, the shader invocations from the previous stage are guaranteed to have executed far enough to generate input values for all required inputs.

## 9.5.1. Shader Termination

A shader invocation that is *terminated* has finished executing instructions.

Executing OpReturn in the entry point, or executing OpTerminateInvocation in any function will terminate an invocation. Implementations **may** also terminate a shader invocation when OpKill is executed in any function; otherwise it becomes a helper invocation.

In addition to the above conditions, helper invocations **may** be terminated when all non-helper invocations in the same derivative group either terminate or become helper invocations.

A shader stage for a given command completes execution when all invocations for that stage have terminated.

| | |
|---|---|
| **NOTE** | `OpKill` will behave the same as either `OpTerminateInvocation` or `OpDemoteToHelperInvocation` depending on the implementation. It is recommended that shader authors use `OpTerminateInvocation` or `OpDemoteToHelperInvocation` instead of `OpKill` whenever possible to produce more predictable behavior. |

## 9.6. Shader Memory Access Ordering

The order in which image or buffer memory is read or written by shaders is largely undefined. For some shader types (vertex, tessellation evaluation, and in some cases, fragment), even the number of shader invocations that **may** perform loads and stores is undefined.

In particular, the following rules apply:

- Vertex and tessellation evaluation shaders will be invoked at least once for each unique vertex, as defined in those sections.
- Fragment shaders will be invoked zero or more times, as defined in that section.
- The relative execution order of invocations of the same shader type is undefined. A store issued by a shader when working on primitive B might complete prior to a store for primitive A, even if primitive A is specified prior to primitive B. This applies even to fragment shaders; while fragment shader outputs are always written to the framebuffer in rasterization order, stores executed by fragment shader invocations are not.
- The relative execution order of invocations of different shader types is largely undefined.

| | |
|---|---|
| **NOTE** | The above limitations on shader invocation order make some forms of synchronization between shader invocations within a single set of primitives unimplementable. For example, having one invocation poll memory written by another invocation assumes that the other invocation has been launched and will complete its writes in finite time. |

The Memory Model appendix defines the terminology and rules for how to correctly communicate between shader invocations, such as when a write is Visible-To a read, and what constitutes a Data Race.

Applications **must** not cause a data race.

The SPIR-V **SubgroupMemory**, **CrossWorkgroupMemory**, and **AtomicCounterMemory** memory semantics are ignored. Sequentially consistent atomics and barriers are not supported and **SequentiallyConsistent** is treated as **AcquireRelease**. **SequentiallyConsistent should** not be used.

## 9.7. Shader Inputs and Outputs

Data is passed into and out of shaders using variables with input or output storage class,

respectively. User-defined inputs and outputs are connected between stages by matching their `Location` decorations. Additionally, data **can** be provided by or communicated to special functions provided by the execution environment using `BuiltIn` decorations.

In many cases, the same `BuiltIn` decoration **can** be used in multiple shader stages with similar meaning. The specific behavior of variables decorated as `BuiltIn` is documented in the following sections.

# 9.8. Task Shaders

Task shaders operate in conjunction with the mesh shaders to produce a collection of primitives that will be processed by subsequent stages of the graphics pipeline. Its primary purpose is to create a variable amount of subsequent mesh shader invocations.

Task shaders are invoked via the execution of the programmable mesh shading pipeline.

The task shader has no fixed-function inputs other than variables identifying the specific workgroup and invocation. In the `TaskNV Execution Model` the number of mesh shader workgroups to create is specified via a `TaskCountNV` decorated output variable. In the `TaskEXT Execution Model` the number of mesh shader workgroups to create is specified via the `OpEmitMeshTasksEXT` instruction.

The task shader can write additional outputs to task memory, which can be read by all of the mesh shader workgroups it created.

## 9.8.1. Task Shader Execution

Task workloads are formed from groups of work items called workgroups and processed by the task shader in the current graphics pipeline. A workgroup is a collection of shader invocations that execute the same shader, potentially in parallel. Task shaders execute in *global workgroups* which are divided into a number of *local workgroups* with a size that **can** be set by assigning a value to the `LocalSize` or `LocalSizeId` execution mode or via an object decorated by the `WorkgroupSize` decoration. An invocation within a local workgroup **can** share data with other members of the local workgroup through shared variables and issue memory and control flow barriers to synchronize with other members of the local workgroup. If the subpass includes multiple views in its view mask, a Task shader using `TaskEXT Execution Model` **may** be invoked separately for each view.

# 9.9. Mesh Shaders

Mesh shaders operate in workgroups to produce a collection of primitives that will be processed by subsequent stages of the graphics pipeline. Each workgroup emits zero or more output primitives and the group of vertices and their associated data required for each output primitive.

Mesh shaders are invoked via the execution of the programmable mesh shading pipeline.

The only inputs available to the mesh shader are variables identifying the specific workgroup and invocation and, if applicable, any outputs written to task memory by the task shader that spawned the mesh shader's workgroup. The mesh shader can operate without a task shader as well.

The invocations of the mesh shader workgroup write an output mesh, comprising a set of

primitives with per-primitive attributes, a set of vertices with per-vertex attributes, and an array of indices identifying the mesh vertices that belong to each primitive. The primitives of this mesh are then processed by subsequent graphics pipeline stages, where the outputs of the mesh shader form an interface with the fragment shader.

### 9.9.1. Mesh Shader Execution

Mesh workloads are formed from groups of work items called workgroups and processed by the mesh shader in the current graphics pipeline. A workgroup is a collection of shader invocations that execute the same shader, potentially in parallel. Mesh shaders execute in *global workgroups* which are divided into a number of *local workgroups* with a size that **can** be set by assigning a value to the `LocalSize` or `LocalSizeId` execution mode or via an object decorated by the `WorkgroupSize` decoration. An invocation within a local workgroup **can** share data with other members of the local workgroup through shared variables and issue memory and control flow barriers to synchronize with other members of the local workgroup.

The *global workgroups* may be generated explicitly via the API, or implicitly through the task shader's work creation mechanism. If the subpass includes multiple views in its view mask, a Mesh shader using `MeshEXT Execution Model` **may** be invoked separately for each view.

## 9.10. Cluster Culling Shaders

Cluster Culling shaders are invoked via the execution of the Programmable Cluster Culling Shading pipeline.

The only inputs available to the cluster culling shader are variables identifying the specific workgroup and invocation.

Cluster Culling shaders operate in workgroups to perform cluster-based culling and produce zero or more cluster drawing command that will be processed by subsequent stages of the graphics pipeline.

The Cluster Drawing Command(CDC) is very similar to the MDI command, invocations in workgroup can emit zero of more CDC to draw zero or more visible cluster.

### 9.10.1. Cluster Culling Shader Execution

Cluster Culling workloads are formed from groups of work items called workgroups and processed by the cluster culling shader in the current graphics pipeline. A workgroup is a collection of shader invocations that execute the same shader, potentially in parallel. Cluster Culling shaders execute in *global workgroups* which are divided into a number of *local workgroups* with a size that **can** be set by assigning a value to the `LocalSize` or `LocalSizeId` execution mode or via an object decorated by the `WorkgroupSize` decoration. An invocation within a local workgroup **can** share data with other members of the local workgroup through shared variables and issue memory and control flow barriers to synchronize with other members of the local workgroup.

# 9.11. Vertex Shaders

Each vertex shader invocation operates on one vertex and its associated vertex attribute data, and outputs one vertex and associated data. Graphics pipelines using primitive shading **must** include a vertex shader, and the vertex shader stage is always the first shader stage in the graphics pipeline.

## 9.11.1. Vertex Shader Execution

A vertex shader **must** be executed at least once for each vertex specified by a drawing command. If the subpass includes multiple views in its view mask, the shader **may** be invoked separately for each view. During execution, the shader is presented with the index of the vertex and instance for which it has been invoked. Input variables declared in the vertex shader are filled by the implementation with the values of vertex attributes associated with the invocation being executed.

If the same vertex is specified multiple times in a drawing command (e.g. by including the same index value multiple times in an index buffer) the implementation **may** reuse the results of vertex shading if it can statically determine that the vertex shader invocations will produce identical results.

| | |
|---|---|
| **NOTE** | It is implementation-dependent when and if results of vertex shading are reused, and thus how many times the vertex shader will be executed. This is true also if the vertex shader contains stores or atomic operations (see `vertexPipelineStoresAndAtomics`). |

# 9.12. Tessellation Control Shaders

The tessellation control shader is used to read an input patch provided by the application and to produce an output patch. Each tessellation control shader invocation operates on an input patch (after all control points in the patch are processed by a vertex shader) and its associated data, and outputs a single control point of the output patch and its associated data, and **can** also output additional per-patch data. The input patch is sized according to the `patchControlPoints` member of VkPipelineTessellationStateCreateInfo, as part of input assembly.

The input patch can also be dynamically sized with `patchControlPoints` parameter of vkCmdSetPatchControlPointsEXT.

To dynamically set the number of control points per patch, call:

```
// Provided by VK_EXT_extended_dynamic_state2, VK_EXT_shader_object
void vkCmdSetPatchControlPointsEXT(
    VkCommandBuffer                             commandBuffer,
    uint32_t                                    patchControlPoints);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `patchControlPoints` specifies the number of control points per patch.

This command sets the number of control points per patch for subsequent drawing commands

when drawing using shader objects, or when the graphics pipeline is created with `VK_DYNAMIC_STATE_PATCH_CONTROL_POINTS_EXT` set in VkPipelineDynamicStateCreateInfo::`pDynamicStates`. Otherwise, this state is specified by the VkPipelineTessellationStateCreateInfo::`patchControlPoints` value used to create the currently active pipeline.

## Valid Usage

- VUID-vkCmdSetPatchControlPointsEXT-None-09422
  At least one of the following **must** be true:
  - The `extendedDynamicState2PatchControlPoints` feature is enabled
  - The `shaderObject` feature is enabled
- VUID-vkCmdSetPatchControlPointsEXT-patchControlPoints-04874
  `patchControlPoints` **must** be greater than zero and less than or equal to `VkPhysicalDeviceLimits`::`maxTessellationPatchSize`

## Valid Usage (Implicit)

- VUID-vkCmdSetPatchControlPointsEXT-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdSetPatchControlPointsEXT-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdSetPatchControlPointsEXT-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- VUID-vkCmdSetPatchControlPointsEXT-videocoding
  This command **must** only be called outside of a video coding scope

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics | State |

The size of the output patch is controlled by the `OpExecutionMode OutputVertices` specified in the tessellation control or tessellation evaluation shaders, which **must** be specified in at least one of the shaders. The size of the input and output patches **must** each be greater than zero and less than or equal to `VkPhysicalDeviceLimits::maxTessellationPatchSize`.

### 9.12.1. Tessellation Control Shader Execution

A tessellation control shader is invoked at least once for each *output* vertex in a patch. If the subpass includes multiple views in its view mask, the shader **may** be invoked separately for each view.

Inputs to the tessellation control shader are generated by the vertex shader. Each invocation of the tessellation control shader **can** read the attributes of any incoming vertices and their associated data. The invocations corresponding to a given patch execute logically in parallel, with undefined relative execution order. However, the `OpControlBarrier` instruction **can** be used to provide limited control of the execution order by synchronizing invocations within a patch, effectively dividing tessellation control shader execution into a set of phases. Tessellation control shaders will read undefined values if one invocation reads a per-vertex or per-patch output written by another invocation at any point during the same phase, or if two invocations attempt to write different values to the same per-patch output in a single phase.

## 9.13. Tessellation Evaluation Shaders

The Tessellation Evaluation Shader operates on an input patch of control points and their associated data, and a single input barycentric coordinate indicating the invocation's relative position within the subdivided patch, and outputs a single vertex and its associated data.

### 9.13.1. Tessellation Evaluation Shader Execution

A tessellation evaluation shader is invoked at least once for each unique vertex generated by the tessellator. If the subpass includes multiple views in its view mask, the shader **may** be invoked separately for each view.

## 9.14. Geometry Shaders

The geometry shader operates on a group of vertices and their associated data assembled from a single input primitive, and emits zero or more output primitives and the group of vertices and their associated data required for each output primitive.

### 9.14.1. Geometry Shader Execution

A geometry shader is invoked at least once for each primitive produced by the tessellation stages, or at least once for each primitive generated by primitive assembly when tessellation is not in use. A shader can request that the geometry shader runs multiple instances. A geometry shader is invoked at least once for each instance. If the subpass includes multiple views in its view mask, the shader **may** be invoked separately for each view.

## 9.15. Fragment Shaders

Fragment shaders are invoked as a fragment operation in a graphics pipeline. Each fragment shader invocation operates on a single fragment and its associated data. With few exceptions, fragment shaders do not have access to any data associated with other fragments and are considered to execute in isolation of fragment shader invocations associated with other fragments.

## 9.16. Compute Shaders

Compute shaders are invoked via vkCmdDispatch and vkCmdDispatchIndirect commands. In general, they have access to similar resources as shader stages executing as part of a graphics pipeline.

Compute workloads are formed from groups of work items called workgroups and processed by the compute shader in the current compute pipeline. A workgroup is a collection of shader invocations that execute the same shader, potentially in parallel. Compute shaders execute in *global workgroups* which are divided into a number of *local workgroups* with a size that **can** be set by assigning a value to the `LocalSize` or `LocalSizeId` execution mode or via an object decorated by the `WorkgroupSize` decoration. An invocation within a local workgroup **can** share data with other members of the local workgroup through shared variables and issue memory and control flow barriers to synchronize with other members of the local workgroup.

## 9.17. Ray Generation Shaders

A ray generation shader is similar to a compute shader. Its main purpose is to execute ray tracing queries using pipeline trace ray instructions (such as `OpTraceRayKHR`) and process the results.

### 9.17.1. Ray Generation Shader Execution

One ray generation shader is executed per ray tracing dispatch. Its location in the shader binding table (see Shader Binding Table for details) is passed directly into vkCmdTraceRaysKHR using the `pRaygenShaderBindingTable` parameter or vkCmdTraceRaysNV using the `raygenShaderBindingTableBuffer` and `raygenShaderBindingOffset` parameters .

## 9.18. Intersection Shaders

Intersection shaders enable the implementation of arbitrary, application defined geometric primitives. An intersection shader for a primitive is executed whenever its axis-aligned bounding box is hit by a ray.

Like other ray tracing shader domains, an intersection shader operates on a single ray at a time. It also operates on a single primitive at a time. It is therefore the purpose of an intersection shader to compute the ray-primitive intersections and report them. To report an intersection, the shader calls the `OpReportIntersectionKHR` instruction.

An intersection shader communicates with any-hit and closest shaders by generating attribute values that they **can** read. Intersection shaders **cannot** read or modify the ray payload.

### 9.18.1. Intersection Shader Execution

The order in which intersections are found along a ray, and therefore the order in which intersection shaders are executed, is unspecified.

The intersection shader of the closest AABB which intersects the ray is guaranteed to be executed at some point during traversal, unless the ray is forcibly terminated.

## 9.19. Any-Hit Shaders

The any-hit shader is executed after the intersection shader reports an intersection that lies within the current $[t_{min},t_{max}]$ of the ray. The main use of any-hit shaders is to programmatically decide whether or not an intersection will be accepted. The intersection will be accepted unless the shader calls the `OpIgnoreIntersectionKHR` instruction. Any-hit shaders have read-only access to the attributes generated by the corresponding intersection shader, and **can** read or modify the ray payload.

### 9.19.1. Any-Hit Shader Execution

The order in which intersections are found along a ray, and therefore the order in which any-hit shaders are executed, is unspecified.

The any-hit shader of the closest hit is guaranteed to be executed at some point during traversal, unless the ray is forcibly terminated.

## 9.20. Closest Hit Shaders

Closest hit shaders have read-only access to the attributes generated by the corresponding intersection shader, and **can** read or modify the ray payload. They also have access to a number of system-generated values. Closest hit shaders **can** call pipeline trace ray instructions to recursively trace rays.

### 9.20.1. Closest Hit Shader Execution

Exactly one closest hit shader is executed when traversal is finished and an intersection has been found and accepted.

## 9.21. Miss Shaders

Miss shaders **can** access the ray payload and **can** trace new rays through the pipeline trace ray instructions, but **cannot** access attributes since they are not associated with an intersection.

### 9.21.1. Miss Shader Execution

A miss shader is executed instead of a closest hit shader if no intersection was found during traversal.

# 9.22. Callable Shaders

Callable shaders **can** access a callable payload that works similarly to ray payloads to do subroutine work.

### 9.22.1. Callable Shader Execution

A callable shader is executed by calling `OpExecuteCallableKHR` from an allowed shader stage.

# 9.23. Interpolation Decorations

Variables in the `Input` storage class in a fragment shader's interface are interpolated from the values specified by the primitive being rasterized.

| | |
|---|---|
| **NOTE** | Interpolation decorations can be present on input and output variables in pre-rasterization shaders but have no effect on the interpolation performed. |

An undecorated input variable will be interpolated with perspective-correct interpolation according to the primitive type being rasterized. Lines and polygons are interpolated in the same way as the primitive's clip coordinates. If the `NoPerspective` decoration is present, linear interpolation is instead used for lines and polygons. For points, as there is only a single vertex, input values are never interpolated and instead take the value written for the single vertex.

If the `Flat` decoration is present on an input variable, the value is not interpolated, and instead takes its value directly from the provoking vertex. Fragment shader inputs that are signed or unsigned integers, integer vectors, or any double-precision floating-point type **must** be decorated with `Flat`.

Interpolation of input variables is performed at an implementation-defined position within the fragment area being shaded. The position is further constrained as follows:

- If the `Centroid` decoration is used, the interpolation position used for the variable **must** also fall within the bounds of the primitive being rasterized.
- If the `Sample` decoration is used, the interpolation position used for the variable **must** be at the position of the sample being shaded by the current fragment shader invocation.
- If a sample count of 1 is used, the interpolation position **must** be at the center of the fragment area.

| | |
|---|---|
| **NOTE** | As `Centroid` restricts the possible interpolation position to the covered area of the primitive, the position can be forced to vary between neighboring fragments when it otherwise would not. Derivatives calculated based on these differing locations can produce inconsistent results compared to undecorated inputs. It is recommended that input variables used in derivative calculations are not decorated with `Centroid`. |

If the `PerVertexKHR` decoration is present on an input variable, the value is not interpolated, and instead values from all input vertices are available in an array. Each index of the array corresponds to one of the vertices of the primitive that produced the fragment.

If the `CustomInterpAMD` decoration is present on an input variable, the value **cannot** be accessed directly; instead the extended instruction `InterpolateAtVertexAMD` **must** be used to obtain values from the input vertices.

## 9.24. Static Use

A SPIR-V module declares a global object in memory using the `OpVariable` instruction, which results in a pointer `x` to that object. A specific entry point in a SPIR-V module is said to *statically use* that object if that entry point's call tree contains a function containing a instruction with `x` as an `id` operand. A shader entry point also *statically uses* any variables explicitly declared in its interface.

## 9.25. Scope

A *scope* describes a set of shader invocations, where each such set is a *scope instance*. Each invocation belongs to one or more scope instances, but belongs to no more than one scope instance for each scope.

The operations available between invocations in a given scope instance vary, with smaller scopes generally able to perform more operations, and with greater efficiency.

### 9.25.1. Cross Device

All invocations executed in a Vulkan instance fall into a single *cross device scope instance*.

Whilst the `CrossDevice` scope is defined in SPIR-V, it is disallowed in Vulkan. API synchronization commands **can** be used to communicate between devices.

### 9.25.2. Device

All invocations executed on a single device form a *device scope instance*.

If the `vulkanMemoryModel` and `vulkanMemoryModelDeviceScope` features are enabled, this scope is represented in SPIR-V by the `Device Scope`, which **can** be used as a `Memory Scope` for barrier and atomic operations.

If both the `shaderDeviceClock` and `vulkanMemoryModelDeviceScope` features are enabled, using the `Device Scope` with the `OpReadClockKHR` instruction will read from a clock that is consistent across invocations in the same device scope instance.

There is no method to synchronize the execution of these invocations within SPIR-V, and this **can** only be done with API synchronization primitives.

Invocations executing on different devices in a device group operate in separate device scope instances.

### 9.25.3. Queue Family

Invocations executed by queues in a given queue family form a *queue family scope instance*.

This scope is identified in SPIR-V as the `QueueFamily Scope` if the `vulkanMemoryModel` feature is enabled, or if not, the `Device Scope`, which **can** be used as a `Memory Scope` for barrier and atomic operations.

If the `shaderDeviceClock` feature is enabled, but the `vulkanMemoryModelDeviceScope` feature is not enabled, using the `Device Scope` with the `OpReadClockKHR` instruction will read from a clock that is consistent across invocations in the same queue family scope instance.

There is no method to synchronize the execution of these invocations within SPIR-V, and this **can** only be done with API synchronization primitives.

Each invocation in a queue family scope instance **must** be in the same device scope instance.

### 9.25.4. Command

Any shader invocations executed as the result of a single command such as vkCmdDispatch or vkCmdDraw form a *command scope instance*. For indirect drawing commands with `drawCount` greater than one, invocations from separate draws are in separate command scope instances. For ray tracing shaders, an invocation group is an implementation-dependent subset of the set of shader invocations of a given shader stage which are produced by a single trace rays command.

There is no specific `Scope` for communication across invocations in a command scope instance. As this has a clear boundary at the API level, coordination here **can** be performed in the API, rather than in SPIR-V.

Each invocation in a command scope instance **must** be in the same queue-family scope instance.

For shaders without defined workgroups, this set of invocations forms an *invocation group* as defined in the SPIR-V specification.

### 9.25.5. Primitive

Any fragment shader invocations executed as the result of rasterization of a single primitive form a *primitive scope instance*.

There is no specific `Scope` for communication across invocations in a primitive scope instance.

Any generated helper invocations are included in this scope instance.

Each invocation in a primitive scope instance **must** be in the same command scope instance.

Any input variables decorated with `Flat` are uniform within a primitive scope instance.

### 9.25.6. Shader Call

Any shader-call-related invocations that are executed in one or more ray tracing execution models form a *shader call scope instance*.

The `ShaderCallKHR Scope` can be used as `Memory Scope` for barrier and atomic operations.

Each invocation in a shader call scope instance **must** be in the same queue family scope instance.

### 9.25.7. Workgroup

A *local workgroup* is a set of invocations that can synchronize and share data with each other using memory in the `Workgroup` storage class.

The `Workgroup Scope` can be used as both an `Execution Scope` and `Memory Scope` for barrier and atomic operations.

Each invocation in a local workgroup **must** be in the same command scope instance.

Only task, mesh, and compute shaders have defined workgroups - other shader types **cannot** use workgroup functionality. For shaders that have defined workgroups, this set of invocations forms an *invocation group* as defined in the SPIR-V specification.

When variables declared with the `Workgroup` storage class are explicitly laid out (hence they are also decorated with `Block`), the amount of storage consumed is the size of the largest Block variable, not counting any padding at the end. The amount of storage consumed by the non-Block variables declared with the `Workgroup` storage class is implementation-dependent. However, the amount of storage consumed may not exceed the largest block size that would be obtained if all active non-Block variables declared with `Workgroup` storage class were assigned offsets in an arbitrary order by successively taking the smallest valid offset according to the Standard Storage Buffer Layout rules, and with `Boolean` values considered as 32-bit integer values for the purpose of this calculation. (This is equivalent to using the GLSL std430 layout rules.)

### 9.25.8. Subgroup

A *subgroup* (see the subsection “Control Flow” of section 2 of the SPIR-V 1.3 Revision 1 specification) is a set of invocations that can synchronize and share data with each other efficiently.

The `Subgroup Scope` can be used as both an `Execution Scope` and `Memory Scope` for barrier and atomic operations. Other subgroup features allow the use of group operations with subgroup scope.

If the `shaderSubgroupClock` feature is enabled, using the `Subgroup Scope` with the `OpReadClockKHR` instruction will read from a clock that is consistent across invocations in the same subgroup.

For shaders that have defined workgroups, each invocation in a subgroup **must** be in the same local workgroup.

In other shader stages, each invocation in a subgroup **must** be in the same device scope instance.

Only shader stages that support subgroup operations have defined subgroups.

**NOTE**

Subgroups are not guaranteed to be a subset of a single command in shaders that do not have defined workgroups. Values that are guaranteed to be uniform for a given command or sub command may then not be uniform for the subgroup, and vice versa. As such, applications must take care when dealing with mixed uniformity.

A somewhat common example of this would something like trying to optimize access to per-draw data using subgroup operations:

```
buffer { uint draw_data[]; };

flat in int vDrawID; // Passed through from vertex shader

void main()
{
    uint local_draw_data = subgroupBroadcastFirst(draw_data
[local_draw_data]);
}
```

This can be done in an attempt to optimize the shader to only perform the loads once per subgroup. However, if the implementation packs multiple draws into a single subgroup, invocations from draws with a different drawID are now receiving data from the wrong invocation. Applications should rely on implementations to do this kind of optimization automatically where the implementation can, rather than trying to force it.

### 9.25.9. Quad

A *quad scope instance* is formed of four shader invocations.

In a fragment shader, each invocation in a quad scope instance is formed of invocations in neighboring framebuffer locations ($x_i$, $y_i$), where:

- i is the index of the invocation within the scope instance.
- w and h are the number of pixels the fragment covers in the x and y axes.
- w and h are identical for all participating invocations.
- $(x_0) = (x_1 - w) = (x_2) = (x_3 - w)$
- $(y_0) = (y_1) = (y_2 - h) = (y_3 - h)$
- Each invocation has the same layer and sample indices.

In a mesh, task, or compute shader, if the `DerivativeGroupQuadsKHR` execution mode is specified, each invocation in a quad scope instance is formed of invocations with adjacent local invocation IDs ($x_i$, $y_i$), where:

- i is the index of the invocation within the quad scope instance.
- $(x_0) = (x_1 - 1) = (x_2) = (x_3 - 1)$
- $(y_0) = (y_1) = (y_2 - 1) = (y_3 - 1)$
- $x_0$ and $y_0$ are integer multiples of 2.
- Each invocation has the same z coordinate.

In a mesh, task, or compute shader, if the `DerivativeGroupLinearKHR` execution mode is specified, each invocation in a quad scope instance is formed of invocations with adjacent local invocation indices ($l_i$), where:

- i is the index of the invocation within the quad scope instance.
- $(l_0) = (l_1 - 1) = (l_2 - 2) = (l_3 - 3)$
- $l_0$ is an integer multiple of 4.

In all shaders, each invocation in a quad scope instance is formed of invocations in adjacent subgroup invocation indices ($s_i$), where:

- i is the index of the invocation within the quad scope instance.
- $(s_0) = (s_1 - 1) = (s_2 - 2) = (s_3 - 3)$
- $s_0$ is an integer multiple of 4.

Each invocation in a quad scope instance **must** be in the same subgroup.

In a fragment shader, each invocation in a quad scope instance **must** be in the same primitive scope instance.

Fragment , mesh, task, and compute shaders have defined quad scope instances. If the `quadOperationsInAllStages` limit is supported, any shader stages that support subgroup operations also have defined quad scope instances.

### 9.25.10. Fragment Interlock

A *fragment interlock scope instance* is formed of fragment shader invocations based on their framebuffer locations (x,y,layer,sample), executed by commands inside a single subpass.

The specific set of invocations included varies based on the execution mode as follows:

- If the `SampleInterlockOrderedEXT` or `SampleInterlockUnorderedEXT` execution modes are used, only invocations with identical framebuffer locations (x,y,layer,sample) are included.
- If the `PixelInterlockOrderedEXT` or `PixelInterlockUnorderedEXT` execution modes are used, fragments with different sample ids are also included.
- If the `ShadingRateInterlockOrderedEXT` or `ShadingRateInterlockUnorderedEXT` execution modes are used, fragments from neighboring framebuffer locations are also included. The shading rate image or fragment shading rate determines these fragments.

Only fragment shaders with one of the above execution modes have defined fragment interlock scope instances.

There is no specific `Scope` value for communication across invocations in a fragment interlock scope instance. However, this is implicitly used as a memory scope by `OpBeginInvocationInterlockEXT` and `OpEndInvocationInterlockEXT`.

Each invocation in a fragment interlock scope instance **must** be in the same queue family scope instance.

### 9.25.11. Invocation

The smallest *scope* is a single invocation; this is represented by the `Invocation Scope` in SPIR-V.

Fragment shader invocations **must** be in a primitive scope instance.

Invocations in fragment shaders that have a defined fragment interlock scope **must** be in a fragment interlock scope instance.

Invocations in shaders that have defined workgroups **must** be in a local workgroup.

Invocations in shaders that have a defined subgroup scope **must** be in a subgroup.

Invocations in shaders that have a defined quad scope **must** be in a quad scope instance.

All invocations in all stages **must** be in a command scope instance.

# 9.26. Group Operations

*Group operations* are executed by multiple invocations within a scope instance; with each invocation involved in calculating the result. This provides a mechanism for efficient communication between invocations in a particular scope instance.

Group operations all take a `Scope` defining the desired scope instance to operate within. Only the `Subgroup` scope **can** be used for these operations; the `subgroupSupportedOperations` limit defines which types of operation **can** be used.

## 9.26.1. Basic Group Operations

Basic group operations include the use of `OpGroupNonUniformElect`, `OpControlBarrier`, `OpMemoryBarrier`, and atomic operations.

`OpGroupNonUniformElect` **can** be used to choose a single invocation to perform a task for the whole group. Only the invocation with the lowest id in the group will return `true`.

The Memory Model appendix defines the operation of barriers and atomics.

## 9.26.2. Vote Group Operations

The vote group operations allow invocations within a group to compare values across a group. The types of votes enabled are:

- Do all active group invocations agree that an expression is true?
- Do any active group invocations evaluate an expression to true?
- Do all active group invocations have the same value of an expression?

| | |
|---|---|
| **NOTE** | These operations are useful in combination with control flow in that they allow for developers to check whether conditions match across the group and choose potentially faster code-paths in these cases. |

## 9.26.3. Arithmetic Group Operations

The arithmetic group operations allow invocations to perform scans and reductions across a group.

The operators supported are add, mul, min, max, and, or, xor.

For reductions, every invocation in a group will obtain the cumulative result of these operators applied to all values in the group. For exclusive scans, each invocation in a group will obtain the cumulative result of these operators applied to all values in invocations with a lower index in the group. Inclusive scans are identical to exclusive scans, except the cumulative result includes the operator applied to the value in the current invocation.

The order in which these operators are applied is implementation-dependent.

### 9.26.4. Ballot Group Operations

The ballot group operations allow invocations to perform more complex votes across the group. The ballot functionality allows all invocations within a group to provide a boolean value and get as a result what each invocation provided as their boolean value. The broadcast functionality allows values to be broadcast from an invocation to all other invocations within the group.

### 9.26.5. Shuffle Group Operations

The shuffle group operations allow invocations to read values from other invocations within a group.

### 9.26.6. Shuffle Relative Group Operations

The shuffle relative group operations allow invocations to read values from other invocations within the group relative to the current invocation in the group. The relative operations supported allow data to be shifted up and down through the invocations within a group.

### 9.26.7. Clustered Group Operations

The clustered group operations allow invocations to perform an operation among partitions of a group, such that the operation is only performed within the group invocations within a partition. The partitions for clustered group operations are consecutive power-of-two size groups of invocations and the cluster size **must** be known at pipeline creation time. The operations supported are add, mul, min, max, and, or, xor.

### 9.26.8. Rotate Group Operations

The rotate group operations allow invocations to read values from other invocations within the group relative to the current invocation and modulo the size of the group. Clustered rotate group operations perform the same operation within individual partitions of a group.

The partitions for clustered rotate group operations are consecutive power-of-two size groups of invocations and the cluster size **must** be known at pipeline creation time.

## 9.27. Quad Group Operations

Quad group operations (`OpGroupNonUniformQuad*`) are a specialized type of group operations that only operate on quad scope instances. Whilst these instructions do include a `Scope` parameter, this

scope is always overridden; only the quad scope instance is included in its execution scope.

Fragment shaders that statically execute either `OpGroupNonUniformQuadBroadcast` or `OpGroupNonUniformQuadSwap` **must** launch sufficient invocations to ensure their correct operation; additional helper invocations are launched for framebuffer locations not covered by rasterized fragments if necessary.

The index used to select participating invocations is i, as described for a quad scope instance, defined as the *quad index* in the SPIR-V specification.

For `OpGroupNonUniformQuadBroadcast` this value is equal to `Index`. For `OpGroupNonUniformQuadSwap`, it is equal to the implicit `Index` used by each participating invocation.

## 9.28. Derivative Operations

Derivative operations calculate the partial derivative for an expression P as a function of an invocation's x and y coordinates.

Derivative operations operate on a set of invocations known as a *derivative group* as defined in the SPIR-V specification.

A derivative group in a fragment shader is equivalent to the quad scope instance if the `QuadDerivativesKHR` execution mode is specified, otherwise it is equivalent to the primitive scope instance. A derivative group in a mesh, task, or compute shader is equivalent to the quad scope instance.

Derivatives are calculated assuming that P is piecewise linear and continuous within the derivative group.

The following control-flow restrictions apply to derivative operations:

- If the `QuadDerivativesKHR` execution mode is specified, dynamic instances of any derivative operations **must** be executed in control flow that is uniform within the current quad scope instance.
- If the `QuadDerivativesKHR` execution mode is not specified:
  - dynamic instances of explicit derivative instructions (`OpDPdx*`, `OpDPdy*`, and `OpFwidth*`) **must** be executed in control flow that is uniform within a derivative group.
  - dynamic instances of implicit derivative operations **can** be executed in control flow that is not uniform within the derivative group, but results are undefined.

Fragment shaders that statically execute derivative operations **must** launch sufficient invocations to ensure their correct operation; additional helper invocations are launched for framebuffer locations not covered by rasterized fragments if necessary.

| | |
|---|---|
| **NOTE** | In a mesh, task, or compute shader, it is the application's responsibility to ensure that sufficient invocations are launched. |

Derivative operations calculate their results as the difference between the result of P across

invocations in the quad. For fine derivative operations (`OpDPdxFine` and `OpDPdyFine`), the values of $DPdx(P_i)$ are calculated as

$$DPdx(P_0) = DPdx(P_1) = P_1 - P_0$$

$$DPdx(P_2) = DPdx(P_3) = P_3 - P_2$$

and the values of $DPdy(P_i)$ are calculated as

$$DPdy(P_0) = DPdy(P_2) = P_2 - P_0$$

$$DPdy(P_1) = DPdy(P_3) = P_3 - P_1$$

where i is the index of each invocation as described in Quad.

Coarse derivative operations (`OpDPdxCoarse` and `OpDPdyCoarse`), calculate their results in roughly the same manner, but **may** only calculate two values instead of four (one for each of DPdx and DPdy), reusing the same result no matter the originating invocation. If an implementation does this, it **should** use the fine derivative calculations described for $P_0$.

**NOTE**

Derivative values are calculated between fragments rather than pixels. If the fragment shader invocations involved in the calculation cover multiple pixels, these operations cover a wider area, resulting in larger derivative values. This in turn will result in a coarser LOD being selected for image sampling operations using derivatives.

Applications may want to account for this when using multi-pixel fragments; if pixel derivatives are desired, applications should use explicit derivative operations and divide the results by the size of the fragment in each dimension as follows:

$$DPdx(P_n)' = DPdx(P_n) / w$$

$$DPdy(P_n)' = DPdy(P_n) / h$$

where w and h are the size of the fragments in the quad, and $DPdx(P_n)'$ and $DPdy(P_n)'$ are the pixel derivatives.

The results for `OpDPdx` and `OpDPdy` **may** be calculated as either fine or coarse derivatives, with implementations favoring the most efficient approach. Implementations **must** choose coarse or fine consistently between the two.

Executing `OpFwidthFine`, `OpFwidthCoarse`, or `OpFwidth` is equivalent to executing the corresponding `OpDPdx*` and `OpDPdy*` instructions, taking the absolute value of the results, and summing them.

Executing an OpImage*Sample*ImplicitLod instruction is equivalent to executing OpDPdx(Coordinate) and OpDPdy(Coordinate), and passing the results as the Grad operands dx and dy.

| | |
|---|---|
| **NOTE** | It is expected that using the ImplicitLod variants of sampling functions will be substantially more efficient than using the ExplicitLod variants with explicitly generated derivatives. |

## 9.29. Helper Invocations

When performing derivative or quad group operations in a fragment shader, additional invocations **may** be spawned in order to ensure correct results. These additional invocations are known as *helper invocations* and **can** be identified by a non-zero value in the HelperInvocation built-in. Stores and atomics performed by helper invocations **must** not have any effect on memory except for the Function, Private and Output storage classes, and values returned by atomic instructions in helper invocations are undefined.

| | |
|---|---|
| **NOTE** | While storage to Output storage class has an effect even in helper invocations, it does not mean that helper invocations have an effect on the framebuffer. Output variables in fragment shaders can be read from as well, and they behave more like Private variables for the duration of the shader invocation. |

If the MaximallyReconvergesKHR execution mode is applied to the entry point, helper invocations **must** remain active for all instructions for the lifetime of the quad scope instance they are a part of. If the MaximallyReconvergesKHR execution mode is not applied to the entry point, helper invocations **may** be considered inactive for group operations other than derivative and quad group operations. All invocations in a quad scope instance **may** become permanently inactive at any point once the only remaining invocations in that quad scope instance are helper invocations.

## 9.30. Cooperative Matrices

A *cooperative matrix* type is a SPIR-V type where the storage for and computations performed on the matrix are spread across the invocations in a scope instance. These types give the implementation freedom in how to optimize matrix multiplies.

SPIR-V defines the types and instructions, but does not specify rules about what sizes/combinations are valid, and it is expected that different implementations **may** support different sizes.

To enumerate the supported cooperative matrix types and operations, call:

```
// Provided by VK_KHR_cooperative_matrix
VkResult vkGetPhysicalDeviceCooperativeMatrixPropertiesKHR(
    VkPhysicalDevice                            physicalDevice,
    uint32_t*                                   pPropertyCount,
    VkCooperativeMatrixPropertiesKHR*           pProperties);
```

- physicalDevice is the physical device.

- pPropertyCount is a pointer to an integer related to the number of cooperative matrix properties available or queried.
- pProperties is either NULL or a pointer to an array of VkCooperativeMatrixPropertiesKHR structures.

If pProperties is NULL, then the number of cooperative matrix properties available is returned in pPropertyCount. Otherwise, pPropertyCount **must** point to a variable set by the application to the number of elements in the pProperties array, and on return the variable is overwritten with the number of structures actually written to pProperties. If pPropertyCount is less than the number of cooperative matrix properties available, at most pPropertyCount structures will be written, and VK_INCOMPLETE will be returned instead of VK_SUCCESS, to indicate that not all the available cooperative matrix properties were returned.

### Valid Usage (Implicit)

- VUID-vkGetPhysicalDeviceCooperativeMatrixPropertiesKHR-physicalDevice-parameter
  physicalDevice **must** be a valid VkPhysicalDevice handle
- VUID-vkGetPhysicalDeviceCooperativeMatrixPropertiesKHR-pPropertyCount-parameter
  pPropertyCount **must** be a valid pointer to a uint32_t value
- VUID-vkGetPhysicalDeviceCooperativeMatrixPropertiesKHR-pProperties-parameter
  If the value referenced by pPropertyCount is not 0, and pProperties is not NULL, pProperties **must** be a valid pointer to an array of pPropertyCount VkCooperativeMatrixPropertiesKHR structures

### Return Codes

**Success**

- VK_SUCCESS
- VK_INCOMPLETE

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY

To enumerate the supported cooperative matrix types and operations, call:

```
// Provided by VK_NV_cooperative_matrix
VkResult vkGetPhysicalDeviceCooperativeMatrixPropertiesNV(
    VkPhysicalDevice                            physicalDevice,
    uint32_t*                                   pPropertyCount,
    VkCooperativeMatrixPropertiesNV*            pProperties);
```

- physicalDevice is the physical device.

- pPropertyCount is a pointer to an integer related to the number of cooperative matrix properties available or queried.
- pProperties is either NULL or a pointer to an array of VkCooperativeMatrixPropertiesNV structures.

If pProperties is NULL, then the number of cooperative matrix properties available is returned in pPropertyCount. Otherwise, pPropertyCount **must** point to a variable set by the application to the number of elements in the pProperties array, and on return the variable is overwritten with the number of structures actually written to pProperties. If pPropertyCount is less than the number of cooperative matrix properties available, at most pPropertyCount structures will be written, and VK_INCOMPLETE will be returned instead of VK_SUCCESS, to indicate that not all the available cooperative matrix properties were returned.

### Valid Usage (Implicit)

- VUID-vkGetPhysicalDeviceCooperativeMatrixPropertiesNV-physicalDevice-parameter
  physicalDevice **must** be a valid VkPhysicalDevice handle
- VUID-vkGetPhysicalDeviceCooperativeMatrixPropertiesNV-pPropertyCount-parameter
  pPropertyCount **must** be a valid pointer to a uint32_t value
- VUID-vkGetPhysicalDeviceCooperativeMatrixPropertiesNV-pProperties-parameter
  If the value referenced by pPropertyCount is not 0, and pProperties is not NULL, pProperties **must** be a valid pointer to an array of pPropertyCount VkCooperativeMatrixPropertiesNV structures

### Return Codes

**Success**

- VK_SUCCESS
- VK_INCOMPLETE

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY

Each VkCooperativeMatrixPropertiesKHR or VkCooperativeMatrixPropertiesNV structure describes a single supported combination of types for a matrix multiply/add operation ( OpCooperativeMatrixMulAddKHR or OpCooperativeMatrixMulAddNV ). The multiply **can** be described in terms of the following variables and types (in SPIR-V pseudocode):

```
    %A is of type OpTypeCooperativeMatrixKHR %AType %scope %MSize %KSize %MatrixAKHR
    %B is of type OpTypeCooperativeMatrixKHR %BType %scope %KSize %NSize %MatrixBKHR
    %C is of type OpTypeCooperativeMatrixKHR %CType %scope %MSize %NSize
%MatrixAccumulatorKHR
    %Result is of type OpTypeCooperativeMatrixKHR %ResultType %scope %MSize %NSize
```

```
%MatrixAccumulatorKHR

    %Result = %A * %B + %C // using OpCooperativeMatrixMulAddKHR
```

```
    %A is of type OpTypeCooperativeMatrixNV %AType %scope %MSize %KSize
    %B is of type OpTypeCooperativeMatrixNV %BType %scope %KSize %NSize
    %C is of type OpTypeCooperativeMatrixNV %CType %scope %MSize %NSize
    %D is of type OpTypeCooperativeMatrixNV %DType %scope %MSize %NSize

    %D = %A * %B + %C // using OpCooperativeMatrixMulAddNV
```

A matrix multiply with these dimensions is known as an *MxNxK* matrix multiply.

The `VkCooperativeMatrixPropertiesKHR` structure is defined as:

```
// Provided by VK_KHR_cooperative_matrix
typedef struct VkCooperativeMatrixPropertiesKHR {
    VkStructureType       sType;
    void*                 pNext;
    uint32_t              MSize;
    uint32_t              NSize;
    uint32_t              KSize;
    VkComponentTypeKHR    AType;
    VkComponentTypeKHR    BType;
    VkComponentTypeKHR    CType;
    VkComponentTypeKHR    ResultType;
    VkBool32              saturatingAccumulation;
    VkScopeKHR            scope;
} VkCooperativeMatrixPropertiesKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `MSize` is the number of rows in matrices `A`, `C`, and `Result`.
- `KSize` is the number of columns in matrix `A` and rows in matrix `B`.
- `NSize` is the number of columns in matrices `B`, `C`, `Result`.
- `AType` is the component type of matrix `A`, of type VkComponentTypeKHR.
- `BType` is the component type of matrix `B`, of type VkComponentTypeKHR.
- `CType` is the component type of matrix `C`, of type VkComponentTypeKHR.
- `ResultType` is the component type of matrix `Result`, of type VkComponentTypeKHR.
- `saturatingAccumulation` indicates whether the `SaturatingAccumulation` operand to `OpCooperativeMatrixMulAddKHR` **must** be present or not. If it is `VK_TRUE`, the `SaturatingAccumulation` operand **must** be present. If it is `VK_FALSE`, the `SaturatingAccumulation` operand **must** not be present.

- scope is the scope of all the matrix types, of type VkScopeKHR.

If some types are preferred over other types (e.g. for performance), they **should** appear earlier in the list enumerated by vkGetPhysicalDeviceCooperativeMatrixPropertiesKHR.

At least one entry in the list **must** have power of two values for all of MSize, KSize, and NSize.

scope **must** be VK_SCOPE_SUBGROUP_KHR.

### Valid Usage (Implicit)

- VUID-VkCooperativeMatrixPropertiesKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_COOPERATIVE_MATRIX_PROPERTIES_KHR
- VUID-VkCooperativeMatrixPropertiesKHR-pNext-pNext
  pNext **must** be NULL

The VkCooperativeMatrixPropertiesNV structure is defined as:

```
// Provided by VK_NV_cooperative_matrix
typedef struct VkCooperativeMatrixPropertiesNV {
    VkStructureType      sType;
    void*                pNext;
    uint32_t             MSize;
    uint32_t             NSize;
    uint32_t             KSize;
    VkComponentTypeNV    AType;
    VkComponentTypeNV    BType;
    VkComponentTypeNV    CType;
    VkComponentTypeNV    DType;
    VkScopeNV            scope;
} VkCooperativeMatrixPropertiesNV;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- MSize is the number of rows in matrices A, C, and D.
- KSize is the number of columns in matrix A and rows in matrix B.
- NSize is the number of columns in matrices B, C, D.
- AType is the component type of matrix A, of type VkComponentTypeNV.
- BType is the component type of matrix B, of type VkComponentTypeNV.
- CType is the component type of matrix C, of type VkComponentTypeNV.
- DType is the component type of matrix D, of type VkComponentTypeNV.
- scope is the scope of all the matrix types, of type VkScopeNV.

If some types are preferred over other types (e.g. for performance), they **should** appear earlier in

the list enumerated by vkGetPhysicalDeviceCooperativeMatrixPropertiesNV.

At least one entry in the list **must** have power of two values for all of `MSize`, `KSize`, and `NSize`.

### Valid Usage (Implicit)

- VUID-VkCooperativeMatrixPropertiesNV-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_COOPERATIVE_MATRIX_PROPERTIES_NV`
- VUID-VkCooperativeMatrixPropertiesNV-pNext-pNext
  `pNext` **must** be `NULL`

Possible values for VkScopeKHR include:

```
// Provided by VK_KHR_cooperative_matrix
typedef enum VkScopeKHR {
    VK_SCOPE_DEVICE_KHR = 1,
    VK_SCOPE_WORKGROUP_KHR = 2,
    VK_SCOPE_SUBGROUP_KHR = 3,
    VK_SCOPE_QUEUE_FAMILY_KHR = 5,
  // Provided by VK_NV_cooperative_matrix
    VK_SCOPE_DEVICE_NV = VK_SCOPE_DEVICE_KHR,
  // Provided by VK_NV_cooperative_matrix
    VK_SCOPE_WORKGROUP_NV = VK_SCOPE_WORKGROUP_KHR,
  // Provided by VK_NV_cooperative_matrix
    VK_SCOPE_SUBGROUP_NV = VK_SCOPE_SUBGROUP_KHR,
  // Provided by VK_NV_cooperative_matrix
    VK_SCOPE_QUEUE_FAMILY_NV = VK_SCOPE_QUEUE_FAMILY_KHR,
} VkScopeKHR;
```

or the equivalent

```
// Provided by VK_NV_cooperative_matrix
typedef VkScopeKHR VkScopeNV;
```

- `VK_SCOPE_DEVICE_KHR` corresponds to SPIR-V `Device` scope.
- `VK_SCOPE_WORKGROUP_KHR` corresponds to SPIR-V `Workgroup` scope.
- `VK_SCOPE_SUBGROUP_KHR` corresponds to SPIR-V `Subgroup` scope.
- `VK_SCOPE_QUEUE_FAMILY_KHR` corresponds to SPIR-V `QueueFamily` scope.

All enum values match the corresponding SPIR-V value.

Possible values for VkComponentTypeKHR include:

```
// Provided by VK_KHR_cooperative_matrix
typedef enum VkComponentTypeKHR {
    VK_COMPONENT_TYPE_FLOAT16_KHR = 0,
    VK_COMPONENT_TYPE_FLOAT32_KHR = 1,
    VK_COMPONENT_TYPE_FLOAT64_KHR = 2,
    VK_COMPONENT_TYPE_SINT8_KHR = 3,
    VK_COMPONENT_TYPE_SINT16_KHR = 4,
    VK_COMPONENT_TYPE_SINT32_KHR = 5,
    VK_COMPONENT_TYPE_SINT64_KHR = 6,
    VK_COMPONENT_TYPE_UINT8_KHR = 7,
    VK_COMPONENT_TYPE_UINT16_KHR = 8,
    VK_COMPONENT_TYPE_UINT32_KHR = 9,
    VK_COMPONENT_TYPE_UINT64_KHR = 10,
  // Provided by VK_NV_cooperative_matrix
    VK_COMPONENT_TYPE_FLOAT16_NV = VK_COMPONENT_TYPE_FLOAT16_KHR,
  // Provided by VK_NV_cooperative_matrix
    VK_COMPONENT_TYPE_FLOAT32_NV = VK_COMPONENT_TYPE_FLOAT32_KHR,
  // Provided by VK_NV_cooperative_matrix
    VK_COMPONENT_TYPE_FLOAT64_NV = VK_COMPONENT_TYPE_FLOAT64_KHR,
  // Provided by VK_NV_cooperative_matrix
    VK_COMPONENT_TYPE_SINT8_NV = VK_COMPONENT_TYPE_SINT8_KHR,
  // Provided by VK_NV_cooperative_matrix
    VK_COMPONENT_TYPE_SINT16_NV = VK_COMPONENT_TYPE_SINT16_KHR,
  // Provided by VK_NV_cooperative_matrix
    VK_COMPONENT_TYPE_SINT32_NV = VK_COMPONENT_TYPE_SINT32_KHR,
  // Provided by VK_NV_cooperative_matrix
    VK_COMPONENT_TYPE_SINT64_NV = VK_COMPONENT_TYPE_SINT64_KHR,
  // Provided by VK_NV_cooperative_matrix
    VK_COMPONENT_TYPE_UINT8_NV = VK_COMPONENT_TYPE_UINT8_KHR,
  // Provided by VK_NV_cooperative_matrix
    VK_COMPONENT_TYPE_UINT16_NV = VK_COMPONENT_TYPE_UINT16_KHR,
  // Provided by VK_NV_cooperative_matrix
    VK_COMPONENT_TYPE_UINT32_NV = VK_COMPONENT_TYPE_UINT32_KHR,
  // Provided by VK_NV_cooperative_matrix
    VK_COMPONENT_TYPE_UINT64_NV = VK_COMPONENT_TYPE_UINT64_KHR,
} VkComponentTypeKHR;
```

or the equivalent

```
// Provided by VK_NV_cooperative_matrix
typedef VkComponentTypeKHR VkComponentTypeNV;
```

- `VK_COMPONENT_TYPE_FLOAT16_KHR` corresponds to SPIR-V `OpTypeFloat` 16.
- `VK_COMPONENT_TYPE_FLOAT32_KHR` corresponds to SPIR-V `OpTypeFloat` 32.
- `VK_COMPONENT_TYPE_FLOAT64_KHR` corresponds to SPIR-V `OpTypeFloat` 64.
- `VK_COMPONENT_TYPE_SINT8_KHR` corresponds to SPIR-V `OpTypeInt` 8 0/1.

- VK_COMPONENT_TYPE_SINT16_KHR corresponds to SPIR-V OpTypeInt 16 0/1.
- VK_COMPONENT_TYPE_SINT32_KHR corresponds to SPIR-V OpTypeInt 32 0/1.
- VK_COMPONENT_TYPE_SINT64_KHR corresponds to SPIR-V OpTypeInt 64 0/1.
- VK_COMPONENT_TYPE_UINT8_KHR corresponds to SPIR-V OpTypeInt 8 0/1.
- VK_COMPONENT_TYPE_UINT16_KHR corresponds to SPIR-V OpTypeInt 16 0/1.
- VK_COMPONENT_TYPE_UINT32_KHR corresponds to SPIR-V OpTypeInt 32 0/1.
- VK_COMPONENT_TYPE_UINT64_KHR corresponds to SPIR-V OpTypeInt 64 0/1.

## 9.31. Validation Cache

Validation cache objects allow the result of internal validation to be reused, both within a single application run and between multiple runs. Reuse within a single run is achieved by passing the same validation cache object when creating supported Vulkan objects. Reuse across runs of an application is achieved by retrieving validation cache contents in one run of an application, saving the contents, and using them to preinitialize a validation cache on a subsequent run. The contents of the validation cache objects are managed by the validation layers. Applications **can** manage the host memory consumed by a validation cache object and control the amount of data retrieved from a validation cache object.

Validation cache objects are represented by VkValidationCacheEXT handles:

```
// Provided by VK_EXT_validation_cache
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkValidationCacheEXT)
```

To create validation cache objects, call:

```
// Provided by VK_EXT_validation_cache
VkResult vkCreateValidationCacheEXT(
    VkDevice                                    device,
    const VkValidationCacheCreateInfoEXT*       pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkValidationCacheEXT*                       pValidationCache);
```

- device is the logical device that creates the validation cache object.
- pCreateInfo is a pointer to a VkValidationCacheCreateInfoEXT structure containing the initial parameters for the validation cache object.
- pAllocator controls host memory allocation as described in the Memory Allocation chapter.
- pValidationCache is a pointer to a VkValidationCacheEXT handle in which the resulting validation cache object is returned.

**NOTE** Applications **can** track and manage the total host memory size of a validation cache object using the pAllocator. Applications **can** limit the amount of data retrieved from a validation cache object in vkGetValidationCacheDataEXT. Implementations

> **should** not internally limit the total number of entries added to a validation cache object or the total host memory consumed.

Once created, a validation cache **can** be passed to the `vkCreateShaderModule` command by adding this object to the VkShaderModuleCreateInfo structure's `pNext` chain. If a VkShaderModuleValidationCacheCreateInfoEXT object is included in the VkShaderModuleCreateInfo::`pNext` chain, and its `validationCache` field is not VK_NULL_HANDLE, the implementation will query it for possible reuse opportunities and update it with new content. The use of the validation cache object in these commands is internally synchronized, and the same validation cache object **can** be used in multiple threads simultaneously.

**NOTE** Implementations **should** make every effort to limit any critical sections to the actual accesses to the cache, which is expected to be significantly shorter than the duration of the `vkCreateShaderModule` command.

### Valid Usage (Implicit)

- VUID-vkCreateValidationCacheEXT-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkCreateValidationCacheEXT-pCreateInfo-parameter
  `pCreateInfo` **must** be a valid pointer to a valid VkValidationCacheCreateInfoEXT structure
- VUID-vkCreateValidationCacheEXT-pAllocator-parameter
  If `pAllocator` is not `NULL`, `pAllocator` **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkCreateValidationCacheEXT-pValidationCache-parameter
  `pValidationCache` **must** be a valid pointer to a VkValidationCacheEXT handle

### Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- `VK_ERROR_OUT_OF_HOST_MEMORY`

The `VkValidationCacheCreateInfoEXT` structure is defined as:

```
// Provided by VK_EXT_validation_cache
typedef struct VkValidationCacheCreateInfoEXT {
    VkStructureType                    sType;
    const void*                        pNext;
    VkValidationCacheCreateFlagsEXT    flags;
    size_t                             initialDataSize;
    const void*                        pInitialData;
} VkValidationCacheCreateInfoEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is reserved for future use.
- `initialDataSize` is the number of bytes in `pInitialData`. If `initialDataSize` is zero, the validation cache will initially be empty.
- `pInitialData` is a pointer to previously retrieved validation cache data. If the validation cache data is incompatible (as defined below) with the device, the validation cache will be initially empty. If `initialDataSize` is zero, `pInitialData` is ignored.

### Valid Usage

- VUID-VkValidationCacheCreateInfoEXT-initialDataSize-01534
  If `initialDataSize` is not `0`, it **must** be equal to the size of `pInitialData`, as returned by `vkGetValidationCacheDataEXT` when `pInitialData` was originally retrieved
- VUID-VkValidationCacheCreateInfoEXT-initialDataSize-01535
  If `initialDataSize` is not `0`, `pInitialData` **must** have been retrieved from a previous call to `vkGetValidationCacheDataEXT`

### Valid Usage (Implicit)

- VUID-VkValidationCacheCreateInfoEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_VALIDATION_CACHE_CREATE_INFO_EXT`
- VUID-VkValidationCacheCreateInfoEXT-pNext-pNext
  `pNext` **must** be `NULL`
- VUID-VkValidationCacheCreateInfoEXT-flags-zerobitmask
  `flags` **must** be `0`
- VUID-VkValidationCacheCreateInfoEXT-pInitialData-parameter
  If `initialDataSize` is not `0`, `pInitialData` **must** be a valid pointer to an array of `initialDataSize` bytes

```
// Provided by VK_EXT_validation_cache
typedef VkFlags VkValidationCacheCreateFlagsEXT;
```

VkValidationCacheCreateFlagsEXT is a bitmask type for setting a mask, but is currently reserved for future use.

Validation cache objects **can** be merged using the command:

```
// Provided by VK_EXT_validation_cache
VkResult vkMergeValidationCachesEXT(
    VkDevice                                    device,
    VkValidationCacheEXT                        dstCache,
    uint32_t                                    srcCacheCount,
    const VkValidationCacheEXT*                 pSrcCaches);
```

- device is the logical device that owns the validation cache objects.
- dstCache is the handle of the validation cache to merge results into.
- srcCacheCount is the length of the pSrcCaches array.
- pSrcCaches is a pointer to an array of validation cache handles, which will be merged into dstCache. The previous contents of dstCache are included after the merge.

| | |
|---|---|
| **NOTE** | The details of the merge operation are implementation-dependent, but implementations **should** merge the contents of the specified validation caches and prune duplicate entries. |

### Valid Usage

- VUID-vkMergeValidationCachesEXT-dstCache-01536
  dstCache **must** not appear in the list of source caches

### Valid Usage (Implicit)

- VUID-vkMergeValidationCachesEXT-device-parameter
  device **must** be a valid VkDevice handle
- VUID-vkMergeValidationCachesEXT-dstCache-parameter
  dstCache **must** be a valid VkValidationCacheEXT handle
- VUID-vkMergeValidationCachesEXT-pSrcCaches-parameter
  pSrcCaches **must** be a valid pointer to an array of srcCacheCount valid VkValidationCacheEXT handles
- VUID-vkMergeValidationCachesEXT-srcCacheCount-arraylength
  srcCacheCount **must** be greater than 0
- VUID-vkMergeValidationCachesEXT-dstCache-parent
  dstCache **must** have been created, allocated, or retrieved from device
- VUID-vkMergeValidationCachesEXT-pSrcCaches-parent
  Each element of pSrcCaches **must** have been created, allocated, or retrieved from device

### Host Synchronization

- Host access to dstCache **must** be externally synchronized

### Return Codes

**Success**

- VK_SUCCESS

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY

Data **can** be retrieved from a validation cache object using the command:

```
// Provided by VK_EXT_validation_cache
VkResult vkGetValidationCacheDataEXT(
    VkDevice                                    device,
    VkValidationCacheEXT                        validationCache,
    size_t*                                     pDataSize,
    void*                                       pData);
```

- device is the logical device that owns the validation cache.
- validationCache is the validation cache to retrieve data from.
- pDataSize is a pointer to a value related to the amount of data in the validation cache, as described below.
- pData is either NULL or a pointer to a buffer.

If pData is NULL, then the maximum size of the data that **can** be retrieved from the validation cache, in bytes, is returned in pDataSize. Otherwise, pDataSize **must** point to a variable set by the application to the size of the buffer, in bytes, pointed to by pData, and on return the variable is overwritten with the amount of data actually written to pData. If pDataSize is less than the maximum size that **can** be retrieved by the validation cache, at most pDataSize bytes will be written to pData, and vkGetValidationCacheDataEXT will return VK_INCOMPLETE instead of VK_SUCCESS, to indicate that not all of the validation cache was returned.

Any data written to pData is valid and **can** be provided as the pInitialData member of the VkValidationCacheCreateInfoEXT structure passed to vkCreateValidationCacheEXT.

Two calls to vkGetValidationCacheDataEXT with the same parameters **must** retrieve the same data unless a command that modifies the contents of the cache is called between them.

Applications **can** store the data retrieved from the validation cache, and use these data, possibly in a future run of the application, to populate new validation cache objects. The results of validation, however, **may** depend on the vendor ID, device ID, driver version, and other details of the device.

To enable applications to detect when previously retrieved data is incompatible with the device, the initial bytes written to `pData` **must** be a header consisting of the following members:

*Table 12. Layout for validation cache header version* `VK_VALIDATION_CACHE_HEADER_VERSION_ONE_EXT`

| Offset | Size | Meaning |
|---|---|---|
| 0 | 4 | length in bytes of the entire validation cache header written as a stream of bytes, with the least significant byte first |
| 4 | 4 | a VkValidationCacheHeaderVersionEXT value written as a stream of bytes, with the least significant byte first |
| 8 | `VK_UUID_SIZE` | a layer commit ID expressed as a UUID, which uniquely identifies the version of the validation layers used to generate these validation results |

The first four bytes encode the length of the entire validation cache header, in bytes. This value includes all fields in the header including the validation cache version field and the size of the length field.

The next four bytes encode the validation cache version, as described for VkValidationCacheHeaderVersionEXT. A consumer of the validation cache **should** use the cache version to interpret the remainder of the cache header.

If `pDataSize` is less than what is necessary to store this header, nothing will be written to `pData` and zero will be written to `pDataSize`.

### Valid Usage (Implicit)

- VUID-vkGetValidationCacheDataEXT-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkGetValidationCacheDataEXT-validationCache-parameter
  `validationCache` **must** be a valid VkValidationCacheEXT handle
- VUID-vkGetValidationCacheDataEXT-pDataSize-parameter
  `pDataSize` **must** be a valid pointer to a `size_t` value
- VUID-vkGetValidationCacheDataEXT-pData-parameter
  If the value referenced by `pDataSize` is not `0`, and `pData` is not `NULL`, `pData` **must** be a valid pointer to an array of `pDataSize` bytes
- VUID-vkGetValidationCacheDataEXT-validationCache-parent
  `validationCache` **must** have been created, allocated, or retrieved from `device`

### Return Codes

**Success**

- VK_SUCCESS
- VK_INCOMPLETE

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY

Possible values of the second group of four bytes in the header returned by vkGetValidationCacheDataEXT, encoding the validation cache version, are:

```
// Provided by VK_EXT_validation_cache
typedef enum VkValidationCacheHeaderVersionEXT {
    VK_VALIDATION_CACHE_HEADER_VERSION_ONE_EXT = 1,
} VkValidationCacheHeaderVersionEXT;
```

- VK_VALIDATION_CACHE_HEADER_VERSION_ONE_EXT specifies version one of the validation cache.

To destroy a validation cache, call:

```
// Provided by VK_EXT_validation_cache
void vkDestroyValidationCacheEXT(
    VkDevice                                    device,
    VkValidationCacheEXT                        validationCache,
    const VkAllocationCallbacks*                pAllocator);
```

- device is the logical device that destroys the validation cache object.
- validationCache is the handle of the validation cache to destroy.
- pAllocator controls host memory allocation as described in the Memory Allocation chapter.

### Valid Usage

- VUID-vkDestroyValidationCacheEXT-validationCache-01537
  If VkAllocationCallbacks were provided when validationCache was created, a compatible set of callbacks **must** be provided here
- VUID-vkDestroyValidationCacheEXT-validationCache-01538
  If no VkAllocationCallbacks were provided when validationCache was created, pAllocator **must** be NULL

### Valid Usage (Implicit)

- VUID-vkDestroyValidationCacheEXT-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkDestroyValidationCacheEXT-validationCache-parameter
  If `validationCache` is not VK_NULL_HANDLE, `validationCache` **must** be a valid VkValidationCacheEXT handle
- VUID-vkDestroyValidationCacheEXT-pAllocator-parameter
  If `pAllocator` is not `NULL`, `pAllocator` **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkDestroyValidationCacheEXT-validationCache-parent
  If `validationCache` is a valid handle, it **must** have been created, allocated, or retrieved from `device`

### Host Synchronization

- Host access to `validationCache` **must** be externally synchronized

# 9.32. CUDA Modules

## 9.32.1. Creating a CUDA Module

CUDA modules **must** contain some kernel code and **must** expose at least one function entry point.

CUDA modules are represented by `VkCudaModuleNV` handles:

```
// Provided by VK_NV_cuda_kernel_launch
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkCudaModuleNV)
```

To create a CUDA module, call:

```
// Provided by VK_NV_cuda_kernel_launch
VkResult vkCreateCudaModuleNV(
    VkDevice                                    device,
    const VkCudaModuleCreateInfoNV*             pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkCudaModuleNV*                             pModule);
```

- `device` is the logical device that creates the shader module.
- `pCreateInfo` is a pointer to a VkCudaModuleCreateInfoNV structure.
- `pAllocator` controls host memory allocation as described in the Memory Allocation chapter.

- pModule is a pointer to a VkCudaModuleNV handle in which the resulting CUDA module object is returned.

Once a CUDA module has been created, the application **may** create the function entry point, which **must** refer to one function in the module.

### Valid Usage (Implicit)

- VUID-vkCreateCudaModuleNV-device-parameter
  device **must** be a valid VkDevice handle
- VUID-vkCreateCudaModuleNV-pCreateInfo-parameter
  pCreateInfo **must** be a valid pointer to a valid VkCudaModuleCreateInfoNV structure
- VUID-vkCreateCudaModuleNV-pAllocator-parameter
  If pAllocator is not NULL, pAllocator **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkCreateCudaModuleNV-pModule-parameter
  pModule **must** be a valid pointer to a VkCudaModuleNV handle

### Return Codes

**Success**

- VK_SUCCESS

**Failure**

- VK_ERROR_INITIALIZATION_FAILED
- VK_ERROR_OUT_OF_HOST_MEMORY

The VkCudaModuleCreateInfoNV structure is defined as:

```
// Provided by VK_NV_cuda_kernel_launch
typedef struct VkCudaModuleCreateInfoNV {
    VkStructureType    sType;
    const void*        pNext;
    size_t             dataSize;
    const void*        pData;
} VkCudaModuleCreateInfoNV;
```

- sType is a VkStructureType value identifying this structure.
- pNext **may** be NULL or **may** be a pointer to a structure extending this structure.
- dataSize is the length of the pData array.
- pData is a pointer to CUDA code

### Valid Usage

- VUID-VkCudaModuleCreateInfoNV-dataSize-09413
  `dataSize` **must** be the total size in bytes of the PTX files or binary cache passed to `pData`

### Valid Usage (Implicit)

- VUID-VkCudaModuleCreateInfoNV-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_CUDA_MODULE_CREATE_INFO_NV`
- VUID-VkCudaModuleCreateInfoNV-pNext-pNext
  `pNext` **must** be `NULL`
- VUID-VkCudaModuleCreateInfoNV-pData-parameter
  `pData` **must** be a valid pointer to an array of `dataSize` bytes
- VUID-VkCudaModuleCreateInfoNV-dataSize-arraylength
  `dataSize` **must** be greater than `0`

## 9.32.2. Creating a CUDA Function Handle

CUDA functions are represented by `VkCudaFunctionNV` handles. Handles to `global` functions **may** then be used to issue a kernel launch (i.e. dispatch) from a commandbuffer. See Dispatching Command for CUDA PTX kernel.

```
// Provided by VK_NV_cuda_kernel_launch
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkCudaFunctionNV)
```

To create a CUDA function, call:

```
// Provided by VK_NV_cuda_kernel_launch
VkResult vkCreateCudaFunctionNV(
    VkDevice                                    device,
    const VkCudaFunctionCreateInfoNV*           pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkCudaFunctionNV*                           pFunction);
```

- `device` is the logical device that creates the shader module.
- `pCreateInfo` is a pointer to a VkCudaFunctionCreateInfoNV structure.
- `pAllocator` controls host memory allocation as described in the Memory Allocation chapter.
- `pFunction` is a pointer to a VkCudaFunctionNV handle in which the resulting CUDA function object is returned.

## Valid Usage (Implicit)

- VUID-vkCreateCudaFunctionNV-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkCreateCudaFunctionNV-pCreateInfo-parameter
  `pCreateInfo` **must** be a valid pointer to a valid VkCudaFunctionCreateInfoNV structure
- VUID-vkCreateCudaFunctionNV-pAllocator-parameter
  If `pAllocator` is not `NULL`, `pAllocator` **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkCreateCudaFunctionNV-pFunction-parameter
  `pFunction` **must** be a valid pointer to a VkCudaFunctionNV handle

## Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- `VK_ERROR_INITIALIZATION_FAILED`
- `VK_ERROR_OUT_OF_HOST_MEMORY`

The `VkCudaFunctionCreateInfoNV` structure is defined as:

```
// Provided by VK_NV_cuda_kernel_launch
typedef struct VkCudaFunctionCreateInfoNV {
    VkStructureType    sType;
    const void*        pNext;
    VkCudaModuleNV     module;
    const char*        pName;
} VkCudaFunctionCreateInfoNV;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `module` is the CUDA VkCudaModuleNV module in which the function resides.
- `pName` is a null-terminated UTF-8 string containing the name of the shader entry point for this stage.

## Valid Usage (Implicit)

- VUID-VkCudaFunctionCreateInfoNV-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_CUDA_FUNCTION_CREATE_INFO_NV`
- VUID-VkCudaFunctionCreateInfoNV-pNext-pNext

pNext **must** be NULL

- VUID-VkCudaFunctionCreateInfoNV-module-parameter
  module **must** be a valid VkCudaModuleNV handle
- VUID-VkCudaFunctionCreateInfoNV-pName-parameter
  pName **must** be a null-terminated UTF-8 string

### 9.32.3. Destroying a CUDA Function

To destroy a CUDA function handle, call:

```
// Provided by VK_NV_cuda_kernel_launch
void vkDestroyCudaFunctionNV(
    VkDevice                                    device,
    VkCudaFunctionNV                            function,
    const VkAllocationCallbacks*                pAllocator);
```

- device is the logical device that destroys the Function.
- function is the handle of the CUDA function to destroy.
- pAllocator controls host memory allocation as described in the Memory Allocation chapter.

**Valid Usage (Implicit)**

- VUID-vkDestroyCudaFunctionNV-device-parameter
  device **must** be a valid VkDevice handle
- VUID-vkDestroyCudaFunctionNV-function-parameter
  function **must** be a valid VkCudaFunctionNV handle
- VUID-vkDestroyCudaFunctionNV-pAllocator-parameter
  If pAllocator is not NULL, pAllocator **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkDestroyCudaFunctionNV-function-parent
  function **must** have been created, allocated, or retrieved from device

### 9.32.4. Destroying a CUDA Module

To destroy a CUDA shader module, call:

```
// Provided by VK_NV_cuda_kernel_launch
void vkDestroyCudaModuleNV(
    VkDevice                                    device,
    VkCudaModuleNV                              module,
    const VkAllocationCallbacks*                pAllocator);
```

- device is the logical device that destroys the shader module.
- module is the handle of the CUDA module to destroy.
- pAllocator controls host memory allocation as described in the Memory Allocation chapter.

**Valid Usage (Implicit)**

- VUID-vkDestroyCudaModuleNV-device-parameter
  device **must** be a valid VkDevice handle
- VUID-vkDestroyCudaModuleNV-module-parameter
  module **must** be a valid VkCudaModuleNV handle
- VUID-vkDestroyCudaModuleNV-pAllocator-parameter
  If pAllocator is not NULL, pAllocator **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkDestroyCudaModuleNV-module-parent
  module **must** have been created, allocated, or retrieved from device

### 9.32.5. Reading back CUDA Module Cache

After uploading the PTX kernel code, the module compiles the code to generate a binary cache with all the necessary information for the device to execute it. It is possible to read back this cache for later use, such as accelerating the initialization of further executions.

To get the CUDA module cache call:

```
// Provided by VK_NV_cuda_kernel_launch
VkResult vkGetCudaModuleCacheNV(
    VkDevice                                    device,
    VkCudaModuleNV                              module,
    size_t*                                     pCacheSize,
    void*                                       pCacheData);
```

- device is the logical device that destroys the Function.
- module is the CUDA module.
- pCacheSize is a pointer containing the amount of bytes to be copied in pCacheData
- pCacheData is a pointer to a buffer in which to copy the binary cache

If pCacheData is NULL, then the size of the binary cache, in bytes, is returned in pCacheSize. Otherwise, pCacheSize **must** point to a variable set by the application to the size of the buffer, in bytes, pointed to by pCacheData, and on return the variable is overwritten with the amount of data actually written to pCacheData. If pCacheSize is less than the size of the binary shader code, nothing is written to pCacheData, and VK_INCOMPLETE will be returned instead of VK_SUCCESS.

The returned cache **may** then be used later for further initialization of the CUDA module, by sending this cache *instead* of the PTX code when using vkCreateCudaModuleNV.

NOTE

Using the binary cache instead of the original PTX code **should** significantly speed up initialization of the CUDA module, given that the whole compilation and validation will not be necessary.

As with VkPipelineCache, the binary cache depends on the specific implementation. The application **must** assume the cache upload might fail in many circumstances and thus **may** have to get ready for falling back to the original PTX code if necessary. Most often, the cache **may** succeed if the same device driver and architecture is used between the cache generation from PTX and the use of this cache. In the event of a new driver version or if using a different device architecture, this cache **may** become invalid.

### Valid Usage (Implicit)

- VUID-vkGetCudaModuleCacheNV-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkGetCudaModuleCacheNV-module-parameter
  `module` **must** be a valid VkCudaModuleNV handle
- VUID-vkGetCudaModuleCacheNV-pCacheSize-parameter
  `pCacheSize` **must** be a valid pointer to a `size_t` value
- VUID-vkGetCudaModuleCacheNV-pCacheData-parameter
  If the value referenced by `pCacheSize` is not `0`, and `pCacheData` is not `NULL`, `pCacheData` **must** be a valid pointer to an array of `pCacheSize` bytes
- VUID-vkGetCudaModuleCacheNV-module-parent
  `module` **must** have been created, allocated, or retrieved from `device`

### Return Codes

**Success**

- `VK_SUCCESS`
- `VK_INCOMPLETE`

**Failure**

- `VK_ERROR_INITIALIZATION_FAILED`

## 9.32.6. Limitations

CUDA and Vulkan do not use the device in the same configuration. The following limitations **must** be taken into account:

- It is not possible to read or write global parameters from Vulkan. The only way to share resources or send values to the PTX kernel is to pass them as arguments of the function. See Resources sharing between CUDA Kernel and Vulkan for more details.

- No calls to functions external to the module PTX are supported.
- Vulkan disables some shader/kernel exceptions, which could break CUDA kernels relying on exceptions.
- CUDA kernels submitted to Vulkan are limited to the amount of shared memory, which can be queried from the physical capabilities. It may be less than what CUDA can offer.
- CUDA instruction-level preemption (CILP) does not work.
- CUDA Unified Memory will not work in this extension.
- CUDA Dynamic parallelism is not supported.
- `vk*DispatchIndirect` is not available.

# Chapter 10. Pipelines

The following figure shows a block diagram of the Vulkan pipelines. Some Vulkan commands specify geometric objects to be drawn or computational work to be performed, while others specify state controlling how objects are handled by the various pipeline stages, or control data transfer between memory organized as images and buffers. Commands are effectively sent through a processing pipeline, either a *graphics pipeline*, a *ray tracing pipeline*, or a *compute pipeline*.

The graphics pipeline can be operated in two modes, as either *primitive shading* or *mesh shading* pipeline.

**Primitive Shading**

The first stage of the graphics pipeline (Input Assembler) assembles vertices to form geometric primitives such as points, lines, and triangles, based on a requested primitive topology. In the next stage (Vertex Shader) vertices **can** be transformed, computing positions and attributes for each vertex. If tessellation and/or geometry shaders are supported, they **can** then generate multiple primitives from a single input primitive, possibly changing the primitive topology or generating additional attribute data in the process.

**Cluster Culling Shading**

When using the Cluster Culling Shader, a compute-like shader will perform cluster-based culling, a set of new built-in output variables are used to express visible cluster, in addition, a new built-in function is used to emit these variables from the cluster culling shader to the Input Assembler(IA) stage, then IA can use these variables to fetches vertices of visible cluster and drive vertex shader to work.

**Mesh Shading**

When using the *mesh shading* pipeline input primitives are not assembled implicitly, but explicitly through the (Mesh Shader). The work on the mesh pipeline is initiated by the application drawing a set of mesh tasks.

If an optional (Task Shader) is active, each task triggers the execution of a task shader workgroup that will generate a new set of tasks upon completion. Each of these spawned tasks, or each of the original dispatched tasks if no task shader is present, triggers the execution of a mesh shader workgroup that produces an output mesh with a variable-sized number of primitives assembled from vertices stored in the output mesh.

**Common**

The final resulting primitives are clipped to a clip volume in preparation for the next stage, Rasterization. The rasterizer produces a series of *fragments* associated with a region of the framebuffer, from a two-dimensional description of a point, line segment, or triangle. These fragments are processed by fragment operations to determine whether generated values will be written to the framebuffer. Fragment shading determines the values to be written to the framebuffer attachments. Framebuffer operations then read and write the color and depth/stencil attachments of the framebuffer for a given subpass of a render pass instance. The attachments **can**

be used as input attachments in the fragment shader in a later subpass of the same render pass.

The compute pipeline is a separate pipeline from the graphics pipeline, which operates on one-, two-, or three-dimensional workgroups which **can** read from and write to buffer and image memory.

This ordering is meant only as a tool for describing Vulkan, not as a strict rule of how Vulkan is implemented, and we present it only as a means to organize the various operations of the pipelines. Actual ordering guarantees between pipeline stages are explained in detail in the synchronization chapter.

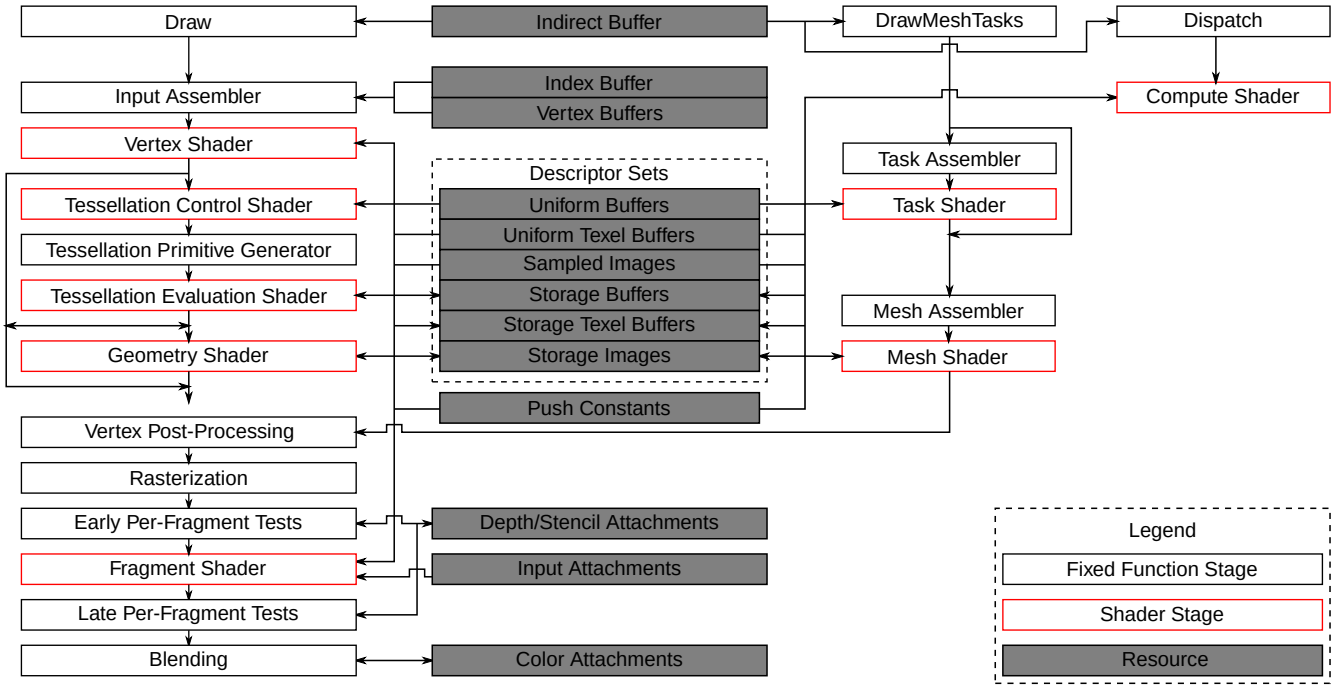



*Figure 2. Block diagram of the Vulkan pipeline*

Each pipeline is controlled by a monolithic object created from a description of all of the shader stages and any relevant fixed-function stages. Linking the whole pipeline together allows the optimization of shaders based on their input/outputs and eliminates expensive draw time state validation.

A pipeline object is bound to the current state using vkCmdBindPipeline. Any pipeline object state that is specified as dynamic is not applied to the current state when the pipeline object is bound, but is instead set by dynamic state setting commands.

If the `commandBufferInheritance` feature is not enabled, then no state, including dynamic state, is inherited from one command buffer to another.

If the `commandBufferInheritance` feature is enabled, then all graphics and compute state that is valid at the end of the command buffer executed in a queue is inherited and valid at beginning of the command buffer next executed in the same queue. This applies to both primary and secondary command buffers, where a primary command buffer submitted to a queue will inherit state from the previously submitted command buffer to that queue, secondary command buffers will inherit state from the primary or seconard command buffer they are executed in, and after a seconard command buffer is executed, its state inherited by the primary or secondary command buffer that executed it. Command buffers executed in one queue do not inherit state from any command

buffers executed in another queue.

Compute, ray tracing, and graphics pipelines are each represented by VkPipeline handles:

```
// Provided by VK_VERSION_1_0
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkPipeline)
```

## 10.1. Multiple Pipeline Creation

Multiple pipelines **can** be created in a single call by commands such as vkCreateExecutionGraphPipelinesAMDX, vkCreateRayTracingPipelinesKHR, vkCreateRayTracingPipelinesNV, vkCreateComputePipelines, and vkCreateGraphicsPipelines.

The creation commands are passed an array pCreateInfos of Vk*PipelineCreateInfo structures specifying parameters of each pipeline to be created, and return a corresponding array of handles in pPipelines. Each element index *i* of pPipelines is created based on the corresponding element *i* of pCreateInfos.

Applications **can** group together similar pipelines to be created in a single call, and implementations are encouraged to look for reuse opportunities when creating a group.

When attempting to create many pipelines in a single command, it is possible that creation **may** fail for a subset of them. In this case, the corresponding elements of pPipelines will be set to VK_NULL_HANDLE. If creation fails for a pipeline despite valid arguments (for example, due to out of memory errors), the VkResult code returned by the pipeline creation command will indicate why. The implementation will attempt to create all pipelines, and only return VK_NULL_HANDLE values for those that actually failed.

If creation fails for a pipeline that has the VK_PIPELINE_CREATE_EARLY_RETURN_ON_FAILURE_BIT set in its Vk*PipelineCreateInfo, pipelines at an index in the pPipelines array greater than or equal to that of the failing pipeline will be set to VK_NULL_HANDLE.

If creation fails for multiple pipelines, the returned VkResult **must** be the return value of any one of the pipelines which did not succeed. An application **can** reliably clean up from a failed call by iterating over the pPipelines array and destroying every element that is not VK_NULL_HANDLE.

If the entire command fails and no pipelines are created, all elements of pPipelines will be set to VK_NULL_HANDLE.

## 10.2. Compute Pipelines

Compute pipelines consist of a single static compute shader stage and the pipeline layout.

The compute pipeline represents a compute shader and is created by calling vkCreateComputePipelines with module and pName selecting an entry point from a shader module, where that entry point defines a valid compute shader, in the VkPipelineShaderStageCreateInfo structure contained within the VkComputePipelineCreateInfo structure.

To create compute pipelines, call:

```
// Provided by VK_VERSION_1_0
VkResult vkCreateComputePipelines(
    VkDevice                                    device,
    VkPipelineCache                             pipelineCache,
    uint32_t                                    createInfoCount,
    const VkComputePipelineCreateInfo*          pCreateInfos,
    const VkAllocationCallbacks*                pAllocator,
    VkPipeline*                                 pPipelines);
```

- `device` is the logical device that creates the compute pipelines.
- `pipelineCache` is either VK_NULL_HANDLE, indicating that pipeline caching is disabled; or the handle of a valid pipeline cache object, in which case use of that cache is enabled for the duration of the command.
- `createInfoCount` is the length of the `pCreateInfos` and `pPipelines` arrays.
- `pCreateInfos` is a pointer to an array of VkComputePipelineCreateInfo structures.
- `pAllocator` controls host memory allocation as described in the Memory Allocation chapter.
- `pPipelines` is a pointer to an array of VkPipeline handles in which the resulting compute pipeline objects are returned.

Pipelines are created and returned as described for Multiple Pipeline Creation.

### Valid Usage

- VUID-vkCreateComputePipelines-device-09661
  `device` **must** support at least one queue family with the `VK_QUEUE_COMPUTE_BIT` capability
- VUID-vkCreateComputePipelines-flags-00695
  If the `flags` member of any element of `pCreateInfos` contains the `VK_PIPELINE_CREATE_DERIVATIVE_BIT` flag, and the `basePipelineIndex` member of that same element is not `-1`, `basePipelineIndex` **must** be less than the index into `pCreateInfos` that corresponds to that element
- VUID-vkCreateComputePipelines-flags-00696
  If the `flags` member of any element of `pCreateInfos` contains the `VK_PIPELINE_CREATE_DERIVATIVE_BIT` flag, the base pipeline **must** have been created with the `VK_PIPELINE_CREATE_ALLOW_DERIVATIVES_BIT` flag set
- VUID-vkCreateComputePipelines-pipelineCache-02873
  If `pipelineCache` was created with `VK_PIPELINE_CACHE_CREATE_EXTERNALLY_SYNCHRONIZED_BIT`, host access to `pipelineCache` **must** be externally synchronized
- VUID-vkCreateComputePipelines-pNext-09616
  If VkPipelineBinaryInfoKHR::`binaryCount` is not `0` for any element of `pCreateInfos`, `pipelineCache` **must** be VK_NULL_HANDLE
- VUID-vkCreateComputePipelines-pNext-09617

If a VkPipelineCreateFlags2CreateInfoKHR structure with the VK_PIPELINE_CREATE_2_CAPTURE_DATA_BIT_KHR flag set is included in the pNext chain of any element of pCreateInfos, pipelineCache **must** be VK_NULL_HANDLE

- VUID-vkCreateComputePipelines-binaryCount-09620
  If VkPipelineBinaryInfoKHR::binaryCount is not 0 for any element of pCreateInfos, VK_PIPELINE_CREATION_FEEDBACK_APPLICATION_PIPELINE_CACHE_HIT_BIT **must** not be set in the flags of that element
- VUID-vkCreateComputePipelines-binaryCount-09621
  If VkPipelineBinaryInfoKHR::binaryCount is not 0 for any element of pCreateInfos, VK_PIPELINE_CREATION_FEEDBACK_BASE_PIPELINE_ACCELERATION_BIT **must** not be set in the flags of that element
- VUID-vkCreateComputePipelines-binaryCount-09622
  If VkPipelineBinaryInfoKHR::binaryCount is not 0 for any element of pCreateInfos, VK_PIPELINE_CREATE_FAIL_ON_PIPELINE_COMPILE_REQUIRED_BIT_EXT **must** not be set in the flags of that element

## Valid Usage (Implicit)

- VUID-vkCreateComputePipelines-device-parameter
  device **must** be a valid VkDevice handle
- VUID-vkCreateComputePipelines-pipelineCache-parameter
  If pipelineCache is not VK_NULL_HANDLE, pipelineCache **must** be a valid VkPipelineCache handle
- VUID-vkCreateComputePipelines-pCreateInfos-parameter
  pCreateInfos **must** be a valid pointer to an array of createInfoCount valid VkComputePipelineCreateInfo structures
- VUID-vkCreateComputePipelines-pAllocator-parameter
  If pAllocator is not NULL, pAllocator **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkCreateComputePipelines-pPipelines-parameter
  pPipelines **must** be a valid pointer to an array of createInfoCount VkPipeline handles
- VUID-vkCreateComputePipelines-createInfoCount-arraylength
  createInfoCount **must** be greater than 0
- VUID-vkCreateComputePipelines-pipelineCache-parent
  If pipelineCache is a valid handle, it **must** have been created, allocated, or retrieved from device

## Return Codes

**Success**

- VK_SUCCESS

- VK_PIPELINE_COMPILE_REQUIRED_EXT

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY
- VK_ERROR_INVALID_SHADER_NV

The VkComputePipelineCreateInfo structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkComputePipelineCreateInfo {
    VkStructureType                    sType;
    const void*                        pNext;
    VkPipelineCreateFlags              flags;
    VkPipelineShaderStageCreateInfo    stage;
    VkPipelineLayout                   layout;
    VkPipeline                         basePipelineHandle;
    int32_t                            basePipelineIndex;
} VkComputePipelineCreateInfo;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- flags is a bitmask of VkPipelineCreateFlagBits specifying how the pipeline will be generated.
- stage is a VkPipelineShaderStageCreateInfo structure describing the compute shader.
- layout is the description of binding locations used by both the pipeline and descriptor sets used with the pipeline.
- basePipelineHandle is a pipeline to derive from.
- basePipelineIndex is an index into the pCreateInfos parameter to use as a pipeline to derive from.

The parameters basePipelineHandle and basePipelineIndex are described in more detail in Pipeline Derivatives.

If the pNext chain includes a VkPipelineCreateFlags2CreateInfoKHR structure, VkPipelineCreateFlags2CreateInfoKHR::flags from that structure is used instead of flags from this structure.

## Valid Usage

- VUID-VkComputePipelineCreateInfo-None-09497
  If the pNext chain does not include a VkPipelineCreateFlags2CreateInfoKHR structure, flags **must** be a valid combination of VkPipelineCreateFlagBits values
- VUID-VkComputePipelineCreateInfo-flags-07984
  If flags contains the VK_PIPELINE_CREATE_DERIVATIVE_BIT flag, and basePipelineIndex is -1,

basePipelineHandle **must** be a valid compute VkPipeline handle

- VUID-VkComputePipelineCreateInfo-flags-07985
  If flags contains the VK_PIPELINE_CREATE_DERIVATIVE_BIT flag, and basePipelineHandle is VK_NULL_HANDLE, basePipelineIndex **must** be a valid index into the calling command's pCreateInfos parameter
- VUID-VkComputePipelineCreateInfo-flags-07986
  If flags contains the VK_PIPELINE_CREATE_DERIVATIVE_BIT flag, basePipelineIndex **must** be -1 or basePipelineHandle **must** be VK_NULL_HANDLE
- VUID-VkComputePipelineCreateInfo-layout-07987
  If a push constant block is declared in a shader, a push constant range in layout **must** match the shader stage
- VUID-VkComputePipelineCreateInfo-layout-10069
  If a push constant block is declared in a shader, the block must be contained inside the push constant range in layout that matches the stage
- VUID-VkComputePipelineCreateInfo-layout-07988
  If a resource variables is declared in a shader, a descriptor slot in layout **must** match the shader stage
- VUID-VkComputePipelineCreateInfo-layout-07990
  If a resource variables is declared in a shader, and the descriptor type is not VK_DESCRIPTOR_TYPE_MUTABLE_EXT, a descriptor slot in layout **must** match the descriptor type
- VUID-VkComputePipelineCreateInfo-layout-07991
  If a resource variables is declared in a shader as an array, a descriptor slot in layout **must** match the descriptor count

- VUID-VkComputePipelineCreateInfo-flags-03365
  flags **must** not include VK_PIPELINE_CREATE_RAY_TRACING_NO_NULL_ANY_HIT_SHADERS_BIT_KHR
- VUID-VkComputePipelineCreateInfo-flags-03366
  flags **must** not include VK_PIPELINE_CREATE_RAY_TRACING_NO_NULL_CLOSEST_HIT_SHADERS_BIT_KHR
- VUID-VkComputePipelineCreateInfo-flags-03367
  flags **must** not include VK_PIPELINE_CREATE_RAY_TRACING_NO_NULL_MISS_SHADERS_BIT_KHR
- VUID-VkComputePipelineCreateInfo-flags-03368
  flags **must** not include VK_PIPELINE_CREATE_RAY_TRACING_NO_NULL_INTERSECTION_SHADERS_BIT_KHR
- VUID-VkComputePipelineCreateInfo-flags-03369
  flags **must** not include VK_PIPELINE_CREATE_RAY_TRACING_SKIP_TRIANGLES_BIT_KHR
- VUID-VkComputePipelineCreateInfo-flags-03370
  flags **must** not include VK_PIPELINE_CREATE_RAY_TRACING_SKIP_AABBS_BIT_KHR
- VUID-VkComputePipelineCreateInfo-flags-03576
  flags **must** not include VK_PIPELINE_CREATE_RAY_TRACING_SHADER_GROUP_HANDLE_CAPTURE_REPLAY_BIT_KHR

- VUID-VkComputePipelineCreateInfo-flags-04945
  flags **must** not include VK_PIPELINE_CREATE_RAY_TRACING_ALLOW_MOTION_BIT_NV
- VUID-VkComputePipelineCreateInfo-flags-09007
  If the VkPhysicalDeviceDeviceGeneratedCommandsComputeFeaturesNV::deviceGeneratedComputePipelines is not enabled, flags **must** not include VK_PIPELINE_CREATE_INDIRECT_BINDABLE_BIT_NV
- VUID-VkComputePipelineCreateInfo-flags-09008
  If flags includes VK_PIPELINE_CREATE_INDIRECT_BINDABLE_BIT_NV, then the pNext chain **must** include a pointer to a valid instance of VkComputePipelineIndirectBufferInfoNV specifying the address where the pipeline's metadata will be saved
- VUID-VkComputePipelineCreateInfo-pipelineCreationCacheControl-02875
  If the pipelineCreationCacheControl feature is not enabled, flags **must** not include VK_PIPELINE_CREATE_FAIL_ON_PIPELINE_COMPILE_REQUIRED_BIT or VK_PIPELINE_CREATE_EARLY_RETURN_ON_FAILURE_BIT
- VUID-VkComputePipelineCreateInfo-stage-00701
  The stage member of stage **must** be VK_SHADER_STAGE_COMPUTE_BIT
- VUID-VkComputePipelineCreateInfo-stage-00702
  The shader code for the entry point identified by stage and the rest of the state identified by this structure **must** adhere to the pipeline linking rules described in the Shader Interfaces chapter
- VUID-VkComputePipelineCreateInfo-layout-01687
  The number of resources in layout accessible to the compute shader stage **must** be less than or equal to VkPhysicalDeviceLimits::maxPerStageResources
- VUID-VkComputePipelineCreateInfo-shaderEnqueue-09177
  If shaderEnqueue is not enabled, flags **must** not include VK_PIPELINE_CREATE_LIBRARY_BIT_KHR
- VUID-VkComputePipelineCreateInfo-flags-09178
  If flags does not include VK_PIPELINE_CREATE_LIBRARY_BIT_KHR, the shader specified by stage **must** not declare the ShaderEnqueueAMDX capability
- VUID-VkComputePipelineCreateInfo-pipelineStageCreationFeedbackCount-06566
  If VkPipelineCreationFeedbackCreateInfo::pipelineStageCreationFeedbackCount is not 0, it **must** be 1
- VUID-VkComputePipelineCreateInfo-flags-07367
  flags **must** not include VK_PIPELINE_CREATE_RAY_TRACING_OPACITY_MICROMAP_BIT_EXT
- VUID-VkComputePipelineCreateInfo-flags-07996
  flags **must** not include VK_PIPELINE_CREATE_RAY_TRACING_DISPLACEMENT_MICROMAP_BIT_NV

## Valid Usage (Implicit)

- VUID-VkComputePipelineCreateInfo-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_COMPUTE_PIPELINE_CREATE_INFO
- VUID-VkComputePipelineCreateInfo-pNext-pNext

Each pNext member of any structure (including this one) in the pNext chain **must** be either NULL or a pointer to a valid instance of VkComputePipelineIndirectBufferInfoNV, VkPipelineBinaryInfoKHR, VkPipelineCompilerControlCreateInfoAMD, VkPipelineCreateFlags2CreateInfoKHR, VkPipelineCreationFeedbackCreateInfo, VkPipelineRobustnessCreateInfoEXT, or VkSubpassShadingPipelineCreateInfoHUAWEI

- VUID-VkComputePipelineCreateInfo-sType-unique
  The sType value of each struct in the pNext chain **must** be unique
- VUID-VkComputePipelineCreateInfo-stage-parameter
  stage **must** be a valid VkPipelineShaderStageCreateInfo structure
- VUID-VkComputePipelineCreateInfo-layout-parameter
  layout **must** be a valid VkPipelineLayout handle
- VUID-VkComputePipelineCreateInfo-commonparent
  Both of basePipelineHandle, and layout that are valid handles of non-ignored parameters **must** have been created, allocated, or retrieved from the same VkDevice

The VkPipelineShaderStageCreateInfo structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkPipelineShaderStageCreateInfo {
    VkStructureType                     sType;
    const void*                         pNext;
    VkPipelineShaderStageCreateFlags    flags;
    VkShaderStageFlagBits               stage;
    VkShaderModule                      module;
    const char*                         pName;
    const VkSpecializationInfo*         pSpecializationInfo;
} VkPipelineShaderStageCreateInfo;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- flags is a bitmask of VkPipelineShaderStageCreateFlagBits specifying how the pipeline shader stage will be generated.
- stage is a VkShaderStageFlagBits value specifying a single pipeline stage.
- module is optionally a VkShaderModule object containing the shader code for this stage.
- pName is a pointer to a null-terminated UTF-8 string specifying the entry point name of the shader for this stage.
- pSpecializationInfo is a pointer to a VkSpecializationInfo structure, as described in Specialization Constants, or NULL.

If module is not VK_NULL_HANDLE, the shader code used by the pipeline is defined by module. If module is VK_NULL_HANDLE, the shader code is defined by the chained VkShaderModuleCreateInfo if present.

If the shaderModuleIdentifier feature is enabled, applications **can** omit shader code for stage and instead provide a module identifier. This is done by including a VkPipelineShaderStageModuleIdentifierCreateInfoEXT struct with identifierSize not equal to 0 in the pNext chain. A shader stage created in this way is equivalent to one created using a shader module with the same identifier. The identifier allows an implementation to look up a pipeline without consuming a valid SPIR-V module. If a pipeline is not found, pipeline compilation is not possible and the implementation **must** fail as specified by VK_PIPELINE_CREATE_FAIL_ON_PIPELINE_COMPILE_REQUIRED_BIT.

When an identifier is used in lieu of a shader module, implementations **may** fail pipeline compilation with VK_PIPELINE_COMPILE_REQUIRED for any reason.

**NOTE** The rationale for the relaxed requirement on implementations to return a pipeline with VkPipelineShaderStageModuleIdentifierCreateInfoEXT is that layers or tools may intercept pipeline creation calls and require the full SPIR-V context to operate correctly. ICDs are not expected to fail pipeline compilation if the pipeline exists in a cache somewhere.

Applications **can** use identifiers when creating pipelines with VK_PIPELINE_CREATE_LIBRARY_BIT_KHR. When creating such pipelines, VK_SUCCESS **may** be returned, but subsequently fail when referencing the pipeline in a VkPipelineLibraryCreateInfoKHR struct. Applications **must** allow pipeline compilation to fail during link steps with VK_PIPELINE_CREATE_FAIL_ON_PIPELINE_COMPILE_REQUIRED_BIT as it **may** not be possible to determine if a pipeline **can** be created from identifiers until the link step.

### Valid Usage

- VUID-VkPipelineShaderStageCreateInfo-stage-00704
  If the geometryShader feature is not enabled, stage **must** not be VK_SHADER_STAGE_GEOMETRY_BIT
- VUID-VkPipelineShaderStageCreateInfo-stage-00705
  If the tessellationShader feature is not enabled, stage **must** not be VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT or VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT
- VUID-VkPipelineShaderStageCreateInfo-stage-02091
  If the meshShaders feature is not enabled, stage **must** not be VK_SHADER_STAGE_MESH_BIT_EXT
- VUID-VkPipelineShaderStageCreateInfo-stage-02092
  If the taskShaders feature is not enabled, stage **must** not be VK_SHADER_STAGE_TASK_BIT_EXT
- VUID-VkPipelineShaderStageCreateInfo-clustercullingShader-07813
  If the clustercullingShader feature is not enabled, stage **must** not be VK_SHADER_STAGE_CLUSTER_CULLING_BIT_HUAWEI
- VUID-VkPipelineShaderStageCreateInfo-stage-00706
  stage **must** not be VK_SHADER_STAGE_ALL_GRAPHICS, or VK_SHADER_STAGE_ALL
- VUID-VkPipelineShaderStageCreateInfo-pName-00707
  pName **must** be the name of an OpEntryPoint in module with an execution model that

matches stage

- VUID-VkPipelineShaderStageCreateInfo-maxClipDistances-00708
  If the identified entry point includes any variable in its interface that is declared with the ClipDistance BuiltIn decoration, that variable **must** not have an array size greater than VkPhysicalDeviceLimits::maxClipDistances
- VUID-VkPipelineShaderStageCreateInfo-maxCullDistances-00709
  If the identified entry point includes any variable in its interface that is declared with the CullDistance BuiltIn decoration, that variable **must** not have an array size greater than VkPhysicalDeviceLimits::maxCullDistances
- VUID-VkPipelineShaderStageCreateInfo-maxCombinedClipAndCullDistances-00710
  If the identified entry point includes variables in its interface that are declared with the ClipDistance BuiltIn decoration and variables in its interface that are declared with the CullDistance BuiltIn decoration, those variables **must** not have array sizes which sum to more than VkPhysicalDeviceLimits::maxCombinedClipAndCullDistances
- VUID-VkPipelineShaderStageCreateInfo-maxSampleMaskWords-00711
  If the identified entry point includes any variable in its interface that is declared with the SampleMask BuiltIn decoration, that variable **must** not have an array size greater than VkPhysicalDeviceLimits::maxSampleMaskWords
- VUID-VkPipelineShaderStageCreateInfo-stage-00713
  If stage is VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT or VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT, and the identified entry point has an OpExecutionMode instruction specifying a patch size with OutputVertices, the patch size **must** be greater than 0 and less than or equal to VkPhysicalDeviceLimits::maxTessellationPatchSize
- VUID-VkPipelineShaderStageCreateInfo-stage-00714
  If stage is VK_SHADER_STAGE_GEOMETRY_BIT, the identified entry point **must** have an OpExecutionMode instruction specifying a maximum output vertex count that is greater than 0 and less than or equal to VkPhysicalDeviceLimits::maxGeometryOutputVertices
- VUID-VkPipelineShaderStageCreateInfo-stage-00715
  If stage is VK_SHADER_STAGE_GEOMETRY_BIT, the identified entry point **must** have an OpExecutionMode instruction specifying an invocation count that is greater than 0 and less than or equal to VkPhysicalDeviceLimits::maxGeometryShaderInvocations
- VUID-VkPipelineShaderStageCreateInfo-stage-02596
  If stage is either VK_SHADER_STAGE_VERTEX_BIT, VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT, VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT, or VK_SHADER_STAGE_GEOMETRY_BIT, and the identified entry point writes to Layer for any primitive, it **must** write the same value to Layer for all vertices of a given primitive
- VUID-VkPipelineShaderStageCreateInfo-stage-02597
  If stage is either VK_SHADER_STAGE_VERTEX_BIT, VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT, VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT, or VK_SHADER_STAGE_GEOMETRY_BIT, and the identified entry point writes to ViewportIndex for any primitive, it **must** write the same value to ViewportIndex for all vertices of a given primitive
- VUID-VkPipelineShaderStageCreateInfo-stage-06685
  If stage is VK_SHADER_STAGE_FRAGMENT_BIT, and the identified entry point writes to FragDepth

in any execution path, all execution paths that are not exclusive to helper invocations **must** either discard the fragment, or write or initialize the value of `FragDepth`

- VUID-VkPipelineShaderStageCreateInfo-stage-06686
  If `stage` is `VK_SHADER_STAGE_FRAGMENT_BIT`, and the identified entry point writes to `FragStencilRefEXT` in any execution path, all execution paths that are not exclusive to helper invocations **must** either discard the fragment, or write or initialize the value of `FragStencilRefEXT`
- VUID-VkPipelineShaderStageCreateInfo-flags-02784
  If `flags` has the `VK_PIPELINE_SHADER_STAGE_CREATE_ALLOW_VARYING_SUBGROUP_SIZE_BIT` flag set, the `subgroupSizeControl` feature **must** be enabled
- VUID-VkPipelineShaderStageCreateInfo-flags-02785
  If `flags` has the `VK_PIPELINE_SHADER_STAGE_CREATE_REQUIRE_FULL_SUBGROUPS_BIT` flag set, the `computeFullSubgroups` feature **must** be enabled
- VUID-VkPipelineShaderStageCreateInfo-flags-08988
  If `flags` includes `VK_PIPELINE_SHADER_STAGE_CREATE_REQUIRE_FULL_SUBGROUPS_BIT`, `stage` **must** be one of `VK_SHADER_STAGE_MESH_BIT_EXT`, `VK_SHADER_STAGE_TASK_BIT_EXT`, or `VK_SHADER_STAGE_COMPUTE_BIT`
- VUID-VkPipelineShaderStageCreateInfo-pNext-02754
  If a VkPipelineShaderStageRequiredSubgroupSizeCreateInfo structure is included in the `pNext` chain, `flags` **must** not have the `VK_PIPELINE_SHADER_STAGE_CREATE_ALLOW_VARYING_SUBGROUP_SIZE_BIT` flag set
- VUID-VkPipelineShaderStageCreateInfo-pNext-02755
  If a VkPipelineShaderStageRequiredSubgroupSizeCreateInfo structure is included in the `pNext` chain, the `subgroupSizeControl` feature **must** be enabled, and `stage` **must** be a valid bit specified in `requiredSubgroupSizeStages`
- VUID-VkPipelineShaderStageCreateInfo-pNext-02756
  If a VkPipelineShaderStageRequiredSubgroupSizeCreateInfo structure is included in the `pNext` chain and `stage` is `VK_SHADER_STAGE_COMPUTE_BIT`, `VK_SHADER_STAGE_MESH_BIT_EXT`, or `VK_SHADER_STAGE_TASK_BIT_EXT`, the local workgroup size of the shader **must** be less than or equal to the product of VkPipelineShaderStageRequiredSubgroupSizeCreateInfo::`requiredSubgroupSize` and `maxComputeWorkgroupSubgroups`
- VUID-VkPipelineShaderStageCreateInfo-pNext-02757
  If a VkPipelineShaderStageRequiredSubgroupSizeCreateInfo structure is included in the `pNext` chain, and `flags` has the `VK_PIPELINE_SHADER_STAGE_CREATE_REQUIRE_FULL_SUBGROUPS_BIT` flag set, the local workgroup size in the X dimension of the pipeline **must** be a multiple of VkPipelineShaderStageRequiredSubgroupSizeCreateInfo::`requiredSubgroupSize`
- VUID-VkPipelineShaderStageCreateInfo-flags-02758
  If `flags` has both the `VK_PIPELINE_SHADER_STAGE_CREATE_REQUIRE_FULL_SUBGROUPS_BIT` and `VK_PIPELINE_SHADER_STAGE_CREATE_ALLOW_VARYING_SUBGROUP_SIZE_BIT` flags set, the local workgroup size in the X dimension of the pipeline **must** be a multiple of `maxSubgroupSize`
- VUID-VkPipelineShaderStageCreateInfo-flags-02759
  If `flags` has the `VK_PIPELINE_SHADER_STAGE_CREATE_REQUIRE_FULL_SUBGROUPS_BIT` flag set and `flags` does not have the `VK_PIPELINE_SHADER_STAGE_CREATE_ALLOW_VARYING_SUBGROUP_SIZE_BIT`

flag set and no VkPipelineShaderStageRequiredSubgroupSizeCreateInfo structure is included in the pNext chain, the local workgroup size in the X dimension of the pipeline **must** be a multiple of subgroupSize

- VUID-VkPipelineShaderStageCreateInfo-module-08987
  If module uses the OpTypeCooperativeMatrixKHR instruction with a Scope equal to Subgroup, then the local workgroup size in the X dimension of the pipeline **must** be a multiple of subgroupSize
- VUID-VkPipelineShaderStageCreateInfo-stage-08771
  If a shader module identifier is not specified for this stage, module **must** be a valid VkShaderModule , or the pNext chain of the parent Vk*CreateInfo structure **must** set VkPipelineBinaryInfoKHR::binaryCount to a value greater than 0, if none of the following features are enabled:
  - graphicsPipelineLibrary
  - maintenance5
- VUID-VkPipelineShaderStageCreateInfo-stage-06845
  If a shader module identifier is not specified for this stage, module **must** be a valid VkShaderModule, or there **must** be a valid VkShaderModuleCreateInfo structure in the pNext chain , or the pNext chain of the parent Vk*CreateInfo structure **must** set VkPipelineBinaryInfoKHR::binaryCount to a value greater than 0,
- VUID-VkPipelineShaderStageCreateInfo-stage-06844
  If a shader module identifier is specified for this stage, the pNext chain **must** not include a VkShaderModuleCreateInfo structure
- VUID-VkPipelineShaderStageCreateInfo-stage-06848
  If a shader module identifier is specified for this stage, module **must** be VK_NULL_HANDLE
- VUID-VkPipelineShaderStageCreateInfo-pSpecializationInfo-06849
  If a shader module identifier is not specified, the shader code used by the pipeline **must** be valid as described by the Khronos SPIR-V Specification after applying the specializations provided in pSpecializationInfo, if any, and then converting all specialization constants into fixed constants

## Valid Usage (Implicit)

- VUID-VkPipelineShaderStageCreateInfo-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PIPELINE_SHADER_STAGE_CREATE_INFO
- VUID-VkPipelineShaderStageCreateInfo-pNext-pNext
  Each pNext member of any structure (including this one) in the pNext chain **must** be either NULL or a pointer to a valid instance of VkDebugUtilsObjectNameInfoEXT, VkPipelineRobustnessCreateInfoEXT, VkPipelineShaderStageModuleIdentifierCreateInfoEXT, VkPipelineShaderStageNodeCreateInfoAMDX, VkPipelineShaderStageRequiredSubgroupSizeCreateInfo, VkShaderModuleCreateInfo, or VkShaderModuleValidationCacheCreateInfoEXT
- VUID-VkPipelineShaderStageCreateInfo-sType-unique

The sType value of each struct in the pNext chain **must** be unique

- VUID-VkPipelineShaderStageCreateInfo-flags-parameter
  flags **must** be a valid combination of VkPipelineShaderStageCreateFlagBits values
- VUID-VkPipelineShaderStageCreateInfo-stage-parameter
  stage **must** be a valid VkShaderStageFlagBits value
- VUID-VkPipelineShaderStageCreateInfo-module-parameter
  If module is not VK_NULL_HANDLE, module **must** be a valid VkShaderModule handle
- VUID-VkPipelineShaderStageCreateInfo-pName-parameter
  pName **must** be a null-terminated UTF-8 string
- VUID-VkPipelineShaderStageCreateInfo-pSpecializationInfo-parameter
  If pSpecializationInfo is not NULL, pSpecializationInfo **must** be a valid pointer to a valid VkSpecializationInfo structure

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkPipelineShaderStageCreateFlags;
```

VkPipelineShaderStageCreateFlags is a bitmask type for setting a mask of zero or more VkPipelineShaderStageCreateFlagBits.

Possible values of the flags member of VkPipelineShaderStageCreateInfo specifying how a pipeline shader stage is created, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkPipelineShaderStageCreateFlagBits {
    VK_PIPELINE_SHADER_STAGE_CREATE_ALLOW_VARYING_SUBGROUP_SIZE_BIT = 0x00000001,
    VK_PIPELINE_SHADER_STAGE_CREATE_REQUIRE_FULL_SUBGROUPS_BIT = 0x00000002,
  // Provided by VK_EXT_subgroup_size_control
    VK_PIPELINE_SHADER_STAGE_CREATE_ALLOW_VARYING_SUBGROUP_SIZE_BIT_EXT =
VK_PIPELINE_SHADER_STAGE_CREATE_ALLOW_VARYING_SUBGROUP_SIZE_BIT,
  // Provided by VK_EXT_subgroup_size_control
    VK_PIPELINE_SHADER_STAGE_CREATE_REQUIRE_FULL_SUBGROUPS_BIT_EXT =
VK_PIPELINE_SHADER_STAGE_CREATE_REQUIRE_FULL_SUBGROUPS_BIT,
} VkPipelineShaderStageCreateFlagBits;
```

- VK_PIPELINE_SHADER_STAGE_CREATE_ALLOW_VARYING_SUBGROUP_SIZE_BIT specifies that the SubgroupSize **may** vary in the shader stage.
- VK_PIPELINE_SHADER_STAGE_CREATE_REQUIRE_FULL_SUBGROUPS_BIT specifies that the subgroup sizes **must** be launched with all invocations active in the task, mesh, or compute stage.

**NOTE** If VK_PIPELINE_SHADER_STAGE_CREATE_ALLOW_VARYING_SUBGROUP_SIZE_BIT_EXT and VK_PIPELINE_SHADER_STAGE_CREATE_REQUIRE_FULL_SUBGROUPS_BIT_EXT are specified and minSubgroupSize does not equal maxSubgroupSize and no required subgroup size is specified, then the only way to guarantee that the 'X' dimension of the local workgroup size is a multiple of SubgroupSize is to make it a multiple of

> `maxSubgroupSize`. Under these conditions, you are guaranteed full subgroups but not any particular subgroup size.

Bits which **can** be set by commands and structures, specifying one or more shader stages, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkShaderStageFlagBits {
    VK_SHADER_STAGE_VERTEX_BIT = 0x00000001,
    VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT = 0x00000002,
    VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT = 0x00000004,
    VK_SHADER_STAGE_GEOMETRY_BIT = 0x00000008,
    VK_SHADER_STAGE_FRAGMENT_BIT = 0x00000010,
    VK_SHADER_STAGE_COMPUTE_BIT = 0x00000020,
    VK_SHADER_STAGE_ALL_GRAPHICS = 0x0000001F,
    VK_SHADER_STAGE_ALL = 0x7FFFFFFF,
  // Provided by VK_KHR_ray_tracing_pipeline
    VK_SHADER_STAGE_RAYGEN_BIT_KHR = 0x00000100,
  // Provided by VK_KHR_ray_tracing_pipeline
    VK_SHADER_STAGE_ANY_HIT_BIT_KHR = 0x00000200,
  // Provided by VK_KHR_ray_tracing_pipeline
    VK_SHADER_STAGE_CLOSEST_HIT_BIT_KHR = 0x00000400,
  // Provided by VK_KHR_ray_tracing_pipeline
    VK_SHADER_STAGE_MISS_BIT_KHR = 0x00000800,
  // Provided by VK_KHR_ray_tracing_pipeline
    VK_SHADER_STAGE_INTERSECTION_BIT_KHR = 0x00001000,
  // Provided by VK_KHR_ray_tracing_pipeline
    VK_SHADER_STAGE_CALLABLE_BIT_KHR = 0x00002000,
  // Provided by VK_EXT_mesh_shader
    VK_SHADER_STAGE_TASK_BIT_EXT = 0x00000040,
  // Provided by VK_EXT_mesh_shader
    VK_SHADER_STAGE_MESH_BIT_EXT = 0x00000080,
  // Provided by VK_HUAWEI_subpass_shading
    VK_SHADER_STAGE_SUBPASS_SHADING_BIT_HUAWEI = 0x00004000,
  // Provided by VK_HUAWEI_cluster_culling_shader
    VK_SHADER_STAGE_CLUSTER_CULLING_BIT_HUAWEI = 0x00080000,
  // Provided by VK_NV_ray_tracing
    VK_SHADER_STAGE_RAYGEN_BIT_NV = VK_SHADER_STAGE_RAYGEN_BIT_KHR,
  // Provided by VK_NV_ray_tracing
    VK_SHADER_STAGE_ANY_HIT_BIT_NV = VK_SHADER_STAGE_ANY_HIT_BIT_KHR,
  // Provided by VK_NV_ray_tracing
    VK_SHADER_STAGE_CLOSEST_HIT_BIT_NV = VK_SHADER_STAGE_CLOSEST_HIT_BIT_KHR,
  // Provided by VK_NV_ray_tracing
    VK_SHADER_STAGE_MISS_BIT_NV = VK_SHADER_STAGE_MISS_BIT_KHR,
  // Provided by VK_NV_ray_tracing
    VK_SHADER_STAGE_INTERSECTION_BIT_NV = VK_SHADER_STAGE_INTERSECTION_BIT_KHR,
  // Provided by VK_NV_ray_tracing
    VK_SHADER_STAGE_CALLABLE_BIT_NV = VK_SHADER_STAGE_CALLABLE_BIT_KHR,
  // Provided by VK_NV_mesh_shader
    VK_SHADER_STAGE_TASK_BIT_NV = VK_SHADER_STAGE_TASK_BIT_EXT,
  // Provided by VK_NV_mesh_shader
    VK_SHADER_STAGE_MESH_BIT_NV = VK_SHADER_STAGE_MESH_BIT_EXT,
} VkShaderStageFlagBits;
```

- `VK_SHADER_STAGE_VERTEX_BIT` specifies the vertex stage.

- VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT specifies the tessellation control stage.
- VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT specifies the tessellation evaluation stage.
- VK_SHADER_STAGE_GEOMETRY_BIT specifies the geometry stage.
- VK_SHADER_STAGE_FRAGMENT_BIT specifies the fragment stage.
- VK_SHADER_STAGE_COMPUTE_BIT specifies the compute stage.
- VK_SHADER_STAGE_ALL_GRAPHICS is a combination of bits used as shorthand to specify all graphics stages defined above (excluding the compute stage).
- VK_SHADER_STAGE_ALL is a combination of bits used as shorthand to specify all shader stages supported by the device, including all additional stages which are introduced by extensions.
- VK_SHADER_STAGE_TASK_BIT_EXT specifies the task stage.
- VK_SHADER_STAGE_MESH_BIT_EXT specifies the mesh stage.
- VK_SHADER_STAGE_CLUSTER_CULLING_BIT_HUAWEI specifies the cluster culling stage.
- VK_SHADER_STAGE_RAYGEN_BIT_KHR specifies the ray generation stage.
- VK_SHADER_STAGE_ANY_HIT_BIT_KHR specifies the any-hit stage.
- VK_SHADER_STAGE_CLOSEST_HIT_BIT_KHR specifies the closest hit stage.
- VK_SHADER_STAGE_MISS_BIT_KHR specifies the miss stage.
- VK_SHADER_STAGE_INTERSECTION_BIT_KHR specifies the intersection stage.
- VK_SHADER_STAGE_CALLABLE_BIT_KHR specifies the callable stage.

**NOTE** VK_SHADER_STAGE_ALL_GRAPHICS only includes the original five graphics stages included in Vulkan 1.0, and not any stages added by extensions. Thus, it may not have the desired effect in all cases.

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkShaderStageFlags;
```

VkShaderStageFlags is a bitmask type for setting a mask of zero or more VkShaderStageFlagBits.

The VkPipelineShaderStageRequiredSubgroupSizeCreateInfo structure is defined as:

```
typedef struct VkPipelineShaderStageRequiredSubgroupSizeCreateInfo {
    VkStructureType    sType;
    void*              pNext;
    uint32_t           requiredSubgroupSize;
} VkPipelineShaderStageRequiredSubgroupSizeCreateInfo;
```

or the equivalent

```
// Provided by VK_EXT_subgroup_size_control
typedef VkPipelineShaderStageRequiredSubgroupSizeCreateInfo
VkPipelineShaderStageRequiredSubgroupSizeCreateInfoEXT;
```

or the equiavlent

```
// Provided by VK_EXT_shader_object
typedef VkPipelineShaderStageRequiredSubgroupSizeCreateInfo
VkShaderRequiredSubgroupSizeCreateInfoEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `requiredSubgroupSize` is an unsigned integer value specifying the required subgroup size for the newly created pipeline shader stage.

If a `VkPipelineShaderStageRequiredSubgroupSizeCreateInfo` structure is included in the `pNext` chain of VkPipelineShaderStageCreateInfo, it specifies that the pipeline shader stage being compiled has a required subgroup size.

If a `VkShaderRequiredSubgroupSizeCreateInfoEXT` structure is included in the `pNext` chain of VkShaderCreateInfoEXT, it specifies that the shader being compiled has a required subgroup size.

### Valid Usage

- VUID-VkPipelineShaderStageRequiredSubgroupSizeCreateInfo-requiredSubgroupSize-02760
  `requiredSubgroupSize` **must** be a power-of-two integer
- VUID-VkPipelineShaderStageRequiredSubgroupSizeCreateInfo-requiredSubgroupSize-02761
  `requiredSubgroupSize` **must** be greater or equal to `minSubgroupSize`
- VUID-VkPipelineShaderStageRequiredSubgroupSizeCreateInfo-requiredSubgroupSize-02762
  `requiredSubgroupSize` **must** be less than or equal to `maxSubgroupSize`

### Valid Usage (Implicit)

- VUID-VkPipelineShaderStageRequiredSubgroupSizeCreateInfo-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PIPELINE_SHADER_STAGE_REQUIRED_SUBGROUP_SIZE_CREATE_INFO`

A subpass shading pipeline is a compute pipeline which **must** be called only in a subpass of a render pass with work dimensions specified by render area size. The subpass shading pipeline shader is a compute shader allowed to access input attachments specified in the calling subpass. To

create a subpass shading pipeline, call vkCreateComputePipelines with VkSubpassShadingPipelineCreateInfoHUAWEI in the pNext chain of VkComputePipelineCreateInfo.

The VkSubpassShadingPipelineCreateInfoHUAWEI structure is defined as:

```
// Provided by VK_HUAWEI_subpass_shading
typedef struct VkSubpassShadingPipelineCreateInfoHUAWEI {
    VkStructureType    sType;
    void*              pNext;
    VkRenderPass       renderPass;
    uint32_t           subpass;
} VkSubpassShadingPipelineCreateInfoHUAWEI;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- renderPass is a handle to a render pass object describing the environment in which the pipeline will be used. The pipeline **must** only be used with a render pass instance compatible with the one provided. See Render Pass Compatibility for more information.
- subpass is the index of the subpass in the render pass where this pipeline will be used.

### Valid Usage

- VUID-VkSubpassShadingPipelineCreateInfoHUAWEI-subpass-04946
  subpass **must** be created with VK_PIPELINE_BIND_POINT_SUBPASS_SHADING_HUAWEI bind point

### Valid Usage (Implicit)

- VUID-VkSubpassShadingPipelineCreateInfoHUAWEI-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_SUBPASS_SHADING_PIPELINE_CREATE_INFO_HUAWEI
- VUID-VkSubpassShadingPipelineCreateInfoHUAWEI-renderPass-parameter
  renderPass **must** be a valid VkRenderPass handle

A subpass shading pipeline's workgroup size is a 2D vector with number of power-of-two in width and height. The maximum number of width and height is implementation-dependent, and **may** vary for different formats and sample counts of attachments in a render pass.

To query the maximum workgroup size, call:

```
// Provided by VK_HUAWEI_subpass_shading
VkResult vkGetDeviceSubpassShadingMaxWorkgroupSizeHUAWEI(
    VkDevice                                    device,
    VkRenderPass                                renderpass,
    VkExtent2D*                                 pMaxWorkgroupSize);
```

- device is a handle to a local device object that was used to create the given render pass.
- renderPass is a handle to a render pass object describing the environment in which the pipeline will be used. The pipeline **must** only be used with a render pass instance compatible with the one provided. See Render Pass Compatibility for more information.
- pMaxWorkgroupSize is a pointer to a VkExtent2D structure.

### Valid Usage (Implicit)

- VUID-vkGetDeviceSubpassShadingMaxWorkgroupSizeHUAWEI-device-parameter
  device **must** be a valid VkDevice handle
- VUID-vkGetDeviceSubpassShadingMaxWorkgroupSizeHUAWEI-renderpass-parameter
  renderpass **must** be a valid VkRenderPass handle
- VUID-vkGetDeviceSubpassShadingMaxWorkgroupSizeHUAWEI-pMaxWorkgroupSize-parameter
  pMaxWorkgroupSize **must** be a valid pointer to VkExtent2D structures
- VUID-vkGetDeviceSubpassShadingMaxWorkgroupSizeHUAWEI-renderpass-parent
  renderpass **must** have been created, allocated, or retrieved from device

### Return Codes

**Success**

- VK_SUCCESS

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY
- VK_ERROR_SURFACE_LOST_KHR

The VkPipelineRobustnessCreateInfoEXT structure is defined as:

```
// Provided by VK_EXT_pipeline_robustness
typedef struct VkPipelineRobustnessCreateInfoEXT {
    VkStructureType                          sType;
    const void*                              pNext;
    VkPipelineRobustnessBufferBehaviorEXT    storageBuffers;
    VkPipelineRobustnessBufferBehaviorEXT    uniformBuffers;
    VkPipelineRobustnessBufferBehaviorEXT    vertexInputs;
    VkPipelineRobustnessImageBehaviorEXT     images;
} VkPipelineRobustnessCreateInfoEXT;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.

- storageBuffers sets the behavior of out of bounds accesses made to resources bound as:
  - VK_DESCRIPTOR_TYPE_STORAGE_BUFFER
  - VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER
  - VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC
- uniformBuffers describes the behavior of out of bounds accesses made to resources bound as:
  - VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER
  - VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER
  - VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC
  - VK_DESCRIPTOR_TYPE_INLINE_UNIFORM_BLOCK
- vertexInputs describes the behavior of out of bounds accesses made to vertex input attributes
- images describes the behavior of out of bounds accesses made to resources bound as:
  - VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE
  - VK_DESCRIPTOR_TYPE_STORAGE_IMAGE

Resources bound as VK_DESCRIPTOR_TYPE_MUTABLE_EXT will have the robustness behavior that covers its active descriptor type.

The scope of the effect of VkPipelineRobustnessCreateInfoEXT depends on which structure's pNext chain it is included in.

- VkGraphicsPipelineCreateInfo, VkRayTracingPipelineCreateInfoKHR, VkComputePipelineCreateInfo: The robustness behavior described by VkPipelineRobustnessCreateInfoEXT applies to all accesses through this pipeline
- VkPipelineShaderStageCreateInfo: The robustness behavior described by VkPipelineRobustnessCreateInfoEXT applies to all accesses emanating from the shader code of this shader stage

If VkPipelineRobustnessCreateInfoEXT is specified for both a pipeline and a pipeline stage, the VkPipelineRobustnessCreateInfoEXT specified for the pipeline stage will take precedence.

When VkPipelineRobustnessCreateInfoEXT is specified for a pipeline, it only affects the subset of the pipeline that is specified by the create info, as opposed to subsets linked from pipeline libraries. For VkGraphicsPipelineCreateInfo, that subset is specified by VkGraphicsPipelineLibraryCreateInfoEXT::flags. For VkRayTracingPipelineCreateInfoKHR, that subset is specified by the specific stages in VkRayTracingPipelineCreateInfoKHR::pStages.

### Valid Usage

- VUID-VkPipelineRobustnessCreateInfoEXT-pipelineRobustness-06926
  If the pipelineRobustness feature is not enabled, storageBuffers **must** be VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_DEVICE_DEFAULT_EXT
- VUID-VkPipelineRobustnessCreateInfoEXT-pipelineRobustness-06927
  If the pipelineRobustness feature is not enabled, uniformBuffers **must** be

VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_DEVICE_DEFAULT_EXT

- VUID-VkPipelineRobustnessCreateInfoEXT-pipelineRobustness-06928
  If the pipelineRobustness feature is not enabled, vertexInputs **must** be VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_DEVICE_DEFAULT_EXT
- VUID-VkPipelineRobustnessCreateInfoEXT-pipelineRobustness-06929
  If the pipelineRobustness feature is not enabled, images **must** be VK_PIPELINE_ROBUSTNESS_IMAGE_BEHAVIOR_DEVICE_DEFAULT_EXT
- VUID-VkPipelineRobustnessCreateInfoEXT-robustImageAccess-06930
  If the robustImageAccess feature is not supported, images **must** not be VK_PIPELINE_ROBUSTNESS_IMAGE_BEHAVIOR_ROBUST_IMAGE_ACCESS_EXT
- VUID-VkPipelineRobustnessCreateInfoEXT-robustBufferAccess2-06931
  If the robustBufferAccess2 feature is not supported, storageBuffers **must** not be VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_2_EXT
- VUID-VkPipelineRobustnessCreateInfoEXT-robustBufferAccess2-06932
  If the robustBufferAccess2 feature is not supported, uniformBuffers **must** not be VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_2_EXT
- VUID-VkPipelineRobustnessCreateInfoEXT-robustBufferAccess2-06933
  If the robustBufferAccess2 feature is not supported, vertexInputs **must** not be VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_2_EXT
- VUID-VkPipelineRobustnessCreateInfoEXT-robustImageAccess2-06934
  If the robustImageAccess2 feature is not supported, images **must** not be VK_PIPELINE_ROBUSTNESS_IMAGE_BEHAVIOR_ROBUST_IMAGE_ACCESS_2_EXT

### Valid Usage (Implicit)

- VUID-VkPipelineRobustnessCreateInfoEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PIPELINE_ROBUSTNESS_CREATE_INFO_EXT
- VUID-VkPipelineRobustnessCreateInfoEXT-storageBuffers-parameter
  storageBuffers **must** be a valid VkPipelineRobustnessBufferBehaviorEXT value
- VUID-VkPipelineRobustnessCreateInfoEXT-uniformBuffers-parameter
  uniformBuffers **must** be a valid VkPipelineRobustnessBufferBehaviorEXT value
- VUID-VkPipelineRobustnessCreateInfoEXT-vertexInputs-parameter
  vertexInputs **must** be a valid VkPipelineRobustnessBufferBehaviorEXT value
- VUID-VkPipelineRobustnessCreateInfoEXT-images-parameter
  images **must** be a valid VkPipelineRobustnessImageBehaviorEXT value

Possible values of the storageBuffers, uniformBuffers, and vertexInputs members of VkPipelineRobustnessCreateInfoEXT are:

```
// Provided by VK_EXT_pipeline_robustness
typedef enum VkPipelineRobustnessBufferBehaviorEXT {
    VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_DEVICE_DEFAULT_EXT = 0,
    VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_DISABLED_EXT = 1,
    VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_EXT = 2,
    VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_2_EXT = 3,
} VkPipelineRobustnessBufferBehaviorEXT;
```

- `VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_DEVICE_DEFAULT_EXT` specifies that this pipeline stage follows the behavior of robustness features that are enabled on the device that created this pipeline
- `VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_DISABLED_EXT` specifies that buffer accesses by this pipeline stage to the relevant resource types **must** not be out of bounds
- `VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_EXT` specifies that out of bounds accesses by this pipeline stage to the relevant resource types behave as if the `robustBufferAccess` feature is enabled
- `VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_2_EXT` specifies that out of bounds accesses by this pipeline stage to the relevant resource types behave as if the `robustBufferAccess2` feature is enabled

Possible values of the `images` member of VkPipelineRobustnessCreateInfoEXT are:

```
// Provided by VK_EXT_pipeline_robustness
typedef enum VkPipelineRobustnessImageBehaviorEXT {
    VK_PIPELINE_ROBUSTNESS_IMAGE_BEHAVIOR_DEVICE_DEFAULT_EXT = 0,
    VK_PIPELINE_ROBUSTNESS_IMAGE_BEHAVIOR_DISABLED_EXT = 1,
    VK_PIPELINE_ROBUSTNESS_IMAGE_BEHAVIOR_ROBUST_IMAGE_ACCESS_EXT = 2,
    VK_PIPELINE_ROBUSTNESS_IMAGE_BEHAVIOR_ROBUST_IMAGE_ACCESS_2_EXT = 3,
} VkPipelineRobustnessImageBehaviorEXT;
```

- `VK_PIPELINE_ROBUSTNESS_IMAGE_BEHAVIOR_DEVICE_DEFAULT_EXT` specifies that this pipeline stage follows the behavior of robustness features that are enabled on the device that created this pipeline
- `VK_PIPELINE_ROBUSTNESS_IMAGE_BEHAVIOR_DISABLED_EXT` specifies that image accesses by this pipeline stage to the relevant resource types **must** not be out of bounds
- `VK_PIPELINE_ROBUSTNESS_IMAGE_BEHAVIOR_ROBUST_IMAGE_ACCESS_EXT` specifies that out of bounds accesses by this pipeline stage to images behave as if the `robustImageAccess` feature is enabled
- `VK_PIPELINE_ROBUSTNESS_IMAGE_BEHAVIOR_ROBUST_IMAGE_ACCESS_2_EXT` specifies that out of bounds accesses by this pipeline stage to images behave as if the `robustImageAccess2` feature is enabled

An identifier **can** be provided instead of shader code in an attempt to compile pipelines without providing complete SPIR-V to the implementation.

The `VkPipelineShaderStageModuleIdentifierCreateInfoEXT` structure is defined as:

```
// Provided by VK_EXT_shader_module_identifier
typedef struct VkPipelineShaderStageModuleIdentifierCreateInfoEXT {
    VkStructureType    sType;
    const void*        pNext;
    uint32_t           identifierSize;
    const uint8_t*     pIdentifier;
} VkPipelineShaderStageModuleIdentifierCreateInfoEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `identifierSize` is the size, in bytes, of the buffer pointed to by `pIdentifier`.
- `pIdentifier` is a pointer to a buffer of opaque data specifying an identifier.

Any identifier **can** be used. If the pipeline being created with identifier requires compilation to complete the pipeline creation call, pipeline compilation **must** fail as defined by `VK_PIPELINE_CREATE_FAIL_ON_PIPELINE_COMPILE_REQUIRED_BIT`.

`pIdentifier` and `identifierSize` **can** be obtained from an VkShaderModuleIdentifierEXT queried earlier.

### Valid Usage

- VUID-VkPipelineShaderStageModuleIdentifierCreateInfoEXT-pNext-06850
  If this structure is included in a `pNext` chain and `identifierSize` is not equal to 0, the `shaderModuleIdentifier` feature **must** be enabled
- VUID-VkPipelineShaderStageModuleIdentifierCreateInfoEXT-pNext-06851
  If this struct is included in a `pNext` chain of VkPipelineShaderStageCreateInfo and `identifierSize` is not equal to 0, the pipeline **must** be created with the `VK_PIPELINE_CREATE_FAIL_ON_PIPELINE_COMPILE_REQUIRED_BIT` flag set
- VUID-VkPipelineShaderStageModuleIdentifierCreateInfoEXT-identifierSize-06852
  `identifierSize` **must** be less-or-equal to `VK_MAX_SHADER_MODULE_IDENTIFIER_SIZE_EXT`

### Valid Usage (Implicit)

- VUID-VkPipelineShaderStageModuleIdentifierCreateInfoEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PIPELINE_SHADER_STAGE_MODULE_IDENTIFIER_CREATE_INFO_EXT`
- VUID-VkPipelineShaderStageModuleIdentifierCreateInfoEXT-pIdentifier-parameter
  If `identifierSize` is not `0`, `pIdentifier` **must** be a valid pointer to an array of `identifierSize` `uint8_t` values

If a compute pipeline is going to be used in Device-Generated Commands by specifying its pipeline token with VkBindPipelineIndirectCommandNV, then that pipeline's associated metadata **must** be

saved at a specified buffer device address for later use in indirect command generation. The buffer device address **must** be specified at the time of compute pipeline creation with VkComputePipelineIndirectBufferInfoNV structure in the pNext chain of VkComputePipelineCreateInfo.

The VkComputePipelineIndirectBufferInfoNV structure is defined as:

```
// Provided by VK_NV_device_generated_commands_compute
typedef struct VkComputePipelineIndirectBufferInfoNV {
    VkStructureType    sType;
    const void*        pNext;
    VkDeviceAddress    deviceAddress;
    VkDeviceSize       size;
    VkDeviceAddress    pipelineDeviceAddressCaptureReplay;
} VkComputePipelineIndirectBufferInfoNV;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- deviceAddress is the address where the pipeline's metadata will be stored.
- size is the size of pipeline's metadata that was queried using vkGetPipelineIndirectMemoryRequirementsNV.
- pipelineDeviceAddressCaptureReplay is the device address where pipeline's metadata was originally saved and can now be used to re-populate deviceAddress for replay.

If pipelineDeviceAddressCaptureReplay is zero, no specific address is requested. If pipelineDeviceAddressCaptureReplay is not zero, then it **must** be an address retrieved from an identically created pipeline on the same implementation. The pipeline metadata **must** also be placed on an identically created buffer and at the same offset using the vkCmdUpdatePipelineIndirectBufferNV command.

### Valid Usage

- VUID-VkComputePipelineIndirectBufferInfoNV-deviceGeneratedComputePipelines-09009
  The VkPhysicalDeviceDeviceGeneratedCommandsComputeFeaturesNV::deviceGeneratedComputePipelines feature **must** be enabled
- VUID-VkComputePipelineIndirectBufferInfoNV-flags-09010
  The pipeline creation flags in VkComputePipelineCreateInfo::flags **must** include VK_PIPELINE_CREATE_INDIRECT_BINDABLE_BIT_NV
- VUID-VkComputePipelineIndirectBufferInfoNV-deviceAddress-09011
  deviceAddress **must** be aligned to the VkMemoryRequirements2::alignment, as returned by vkGetPipelineIndirectMemoryRequirementsNV
- VUID-VkComputePipelineIndirectBufferInfoNV-deviceAddress-09012
  deviceAddress **must** have been allocated from a buffer that was created with usage VK_BUFFER_USAGE_TRANSFER_DST_BIT and VK_BUFFER_USAGE_INDIRECT_BUFFER_BIT

- VUID-VkComputePipelineIndirectBufferInfoNV-size-09013
  size **must** be greater than or equal to the VkMemoryRequirements2::size, as returned by vkGetPipelineIndirectMemoryRequirementsNV
- VUID-VkComputePipelineIndirectBufferInfoNV-pipelineDeviceAddressCaptureReplay-09014
  If pipelineDeviceAddressCaptureReplay is non-zero then the VkPhysicalDeviceDeviceGeneratedCommandsComputeFeaturesNV::deviceGeneratedComputeCaptureReplay feature **must** be enabled
- VUID-VkComputePipelineIndirectBufferInfoNV-pipelineDeviceAddressCaptureReplay-09015
  If pipelineDeviceAddressCaptureReplay is non-zero then that address **must** have been allocated with flag VK_MEMORY_ALLOCATE_DEVICE_ADDRESS_CAPTURE_REPLAY_BIT set
- VUID-VkComputePipelineIndirectBufferInfoNV-pipelineDeviceAddressCaptureReplay-09016
  If pipelineDeviceAddressCaptureReplay is non-zero, the pipeline **must** have been recreated for replay
- VUID-VkComputePipelineIndirectBufferInfoNV-pipelineDeviceAddressCaptureReplay-09017
  pipelineDeviceAddressCaptureReplay **must** satisfy the alignment and size requirements similar to deviceAddress

### Valid Usage (Implicit)

- VUID-VkComputePipelineIndirectBufferInfoNV-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_COMPUTE_PIPELINE_INDIRECT_BUFFER_INFO_NV

To save a compute pipeline's metadata at a device address call:

```
// Provided by VK_NV_device_generated_commands_compute
void vkCmdUpdatePipelineIndirectBufferNV(
    VkCommandBuffer                             commandBuffer,
    VkPipelineBindPoint                         pipelineBindPoint,
    VkPipeline                                  pipeline);
```

- commandBuffer is the command buffer into which the command will be recorded.
- pipelineBindPoint is a VkPipelineBindPoint value specifying the type of pipeline whose metadata will be saved.
- pipeline is the pipeline whose metadata will be saved.

vkCmdUpdatePipelineIndirectBufferNV is only allowed outside of a render pass. This command is treated as a "transfer" operation for the purposes of synchronization barriers. The writes to the address **must** be synchronized using stages VK_PIPELINE_STAGE_2_COPY_BIT and VK_PIPELINE_STAGE_COMMAND_PREPROCESS_BIT_NV and with access masks VK_ACCESS_MEMORY_WRITE_BIT

and VK_ACCESS_COMMAND_PREPROCESS_READ_BIT_NV respectively before using the results in preprocessing.

## Valid Usage

- VUID-vkCmdUpdatePipelineIndirectBufferNV-pipelineBindPoint-09018
  pipelineBindPoint **must** be VK_PIPELINE_BIND_POINT_COMPUTE
- VUID-vkCmdUpdatePipelineIndirectBufferNV-pipeline-09019
  pipeline **must** have been created with VK_PIPELINE_CREATE_INDIRECT_BINDABLE_BIT_NV flag set
- VUID-vkCmdUpdatePipelineIndirectBufferNV-pipeline-09020
  pipeline **must** have been created with VkComputePipelineIndirectBufferInfoNV structure specifying a valid address where its metadata will be saved
- VUID-vkCmdUpdatePipelineIndirectBufferNV-deviceGeneratedComputePipelines-09021
  The VkPhysicalDeviceDeviceGeneratedCommandsComputeFeaturesNV::deviceGeneratedComputePipelines feature **must** be enabled

## Valid Usage (Implicit)

- VUID-vkCmdUpdatePipelineIndirectBufferNV-commandBuffer-parameter
  commandBuffer **must** be a valid VkCommandBuffer handle
- VUID-vkCmdUpdatePipelineIndirectBufferNV-pipelineBindPoint-parameter
  pipelineBindPoint **must** be a valid VkPipelineBindPoint value
- VUID-vkCmdUpdatePipelineIndirectBufferNV-pipeline-parameter
  pipeline **must** be a valid VkPipeline handle
- VUID-vkCmdUpdatePipelineIndirectBufferNV-commandBuffer-recording
  commandBuffer **must** be in the recording state
- VUID-vkCmdUpdatePipelineIndirectBufferNV-commandBuffer-cmdpool
  The VkCommandPool that commandBuffer was allocated from **must** support transfer, graphics, or compute operations
- VUID-vkCmdUpdatePipelineIndirectBufferNV-renderpass
  This command **must** only be called outside of a render pass instance
- VUID-vkCmdUpdatePipelineIndirectBufferNV-videocoding
  This command **must** only be called outside of a video coding scope
- VUID-vkCmdUpdatePipelineIndirectBufferNV-commonparent
  Both of commandBuffer, and pipeline **must** have been created, allocated, or retrieved from the same VkDevice

## Host Synchronization

- Host access to commandBuffer **must** be externally synchronized

- Host access to the VkCommandPool that commandBuffer was allocated from **must** be externally synchronized

**Command Properties**

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Outside | Outside | Transfer<br>Graphics<br>Compute | Action |

## 10.3. Graphics Pipelines

Graphics pipelines consist of multiple shader stages, multiple fixed-function pipeline stages, and a pipeline layout.

To create graphics pipelines, call:

```
// Provided by VK_VERSION_1_0
VkResult vkCreateGraphicsPipelines(
    VkDevice                                    device,
    VkPipelineCache                             pipelineCache,
    uint32_t                                    createInfoCount,
    const VkGraphicsPipelineCreateInfo*         pCreateInfos,
    const VkAllocationCallbacks*                pAllocator,
    VkPipeline*                                 pPipelines);
```

- device is the logical device that creates the graphics pipelines.
- pipelineCache is either VK_NULL_HANDLE, indicating that pipeline caching is disabled; or the handle of a valid pipeline cache object, in which case use of that cache is enabled for the duration of the command.
- createInfoCount is the length of the pCreateInfos and pPipelines arrays.
- pCreateInfos is a pointer to an array of VkGraphicsPipelineCreateInfo structures.
- pAllocator controls host memory allocation as described in the Memory Allocation chapter.
- pPipelines is a pointer to an array of VkPipeline handles in which the resulting graphics pipeline objects are returned.

The VkGraphicsPipelineCreateInfo structure includes an array of VkPipelineShaderStageCreateInfo structures for each of the desired active shader stages, as well as creation information for all relevant fixed-function stages, and a pipeline layout.

Pipelines are created and returned as described for Multiple Pipeline Creation.

## Valid Usage

- VUID-vkCreateGraphicsPipelines-device-09662
  `device` **must** support at least one queue family with the `VK_QUEUE_GRAPHICS_BIT` capability
- VUID-vkCreateGraphicsPipelines-flags-00720
  If the `flags` member of any element of `pCreateInfos` contains the `VK_PIPELINE_CREATE_DERIVATIVE_BIT` flag, and the `basePipelineIndex` member of that same element is not `-1`, `basePipelineIndex` **must** be less than the index into `pCreateInfos` that corresponds to that element
- VUID-vkCreateGraphicsPipelines-flags-00721
  If the `flags` member of any element of `pCreateInfos` contains the `VK_PIPELINE_CREATE_DERIVATIVE_BIT` flag, the base pipeline **must** have been created with the `VK_PIPELINE_CREATE_ALLOW_DERIVATIVES_BIT` flag set
- VUID-vkCreateGraphicsPipelines-pipelineCache-02876
  If `pipelineCache` was created with `VK_PIPELINE_CACHE_CREATE_EXTERNALLY_SYNCHRONIZED_BIT`, host access to `pipelineCache` **must** be externally synchronized
- VUID-vkCreateGraphicsPipelines-pNext-09616
  If VkPipelineBinaryInfoKHR::`binaryCount` is not `0` for any element of `pCreateInfos`, `pipelineCache` **must** be VK_NULL_HANDLE
- VUID-vkCreateGraphicsPipelines-pNext-09617
  If a VkPipelineCreateFlags2CreateInfoKHR structure with the `VK_PIPELINE_CREATE_2_CAPTURE_DATA_BIT_KHR` flag set is included in the `pNext` chain of any element of `pCreateInfos`, `pipelineCache` **must** be VK_NULL_HANDLE
- VUID-vkCreateGraphicsPipelines-binaryCount-09620
  If VkPipelineBinaryInfoKHR::`binaryCount` is not `0` for any element of `pCreateInfos`, `VK_PIPELINE_CREATION_FEEDBACK_APPLICATION_PIPELINE_CACHE_HIT_BIT` **must** not be set in the `flags` of that element
- VUID-vkCreateGraphicsPipelines-binaryCount-09621
  If VkPipelineBinaryInfoKHR::`binaryCount` is not `0` for any element of `pCreateInfos`, `VK_PIPELINE_CREATION_FEEDBACK_BASE_PIPELINE_ACCELERATION_BIT` **must** not be set in the `flags` of that element
- VUID-vkCreateGraphicsPipelines-binaryCount-09622
  If VkPipelineBinaryInfoKHR::`binaryCount` is not `0` for any element of `pCreateInfos`, `VK_PIPELINE_CREATE_FAIL_ON_PIPELINE_COMPILE_REQUIRED_BIT_EXT` **must** not be set in the `flags` of that element

**NOTE** An implicit cache may be provided by the implementation or a layer. For this reason, it is still valid to set `VK_PIPELINE_CREATE_FAIL_ON_PIPELINE_COMPILE_REQUIRED_BIT` on `flags` for any element of `pCreateInfos` while passing VK_NULL_HANDLE for `pipelineCache`.

## Valid Usage (Implicit)

- VUID-vkCreateGraphicsPipelines-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkCreateGraphicsPipelines-pipelineCache-parameter
  If `pipelineCache` is not VK_NULL_HANDLE, `pipelineCache` **must** be a valid VkPipelineCache handle
- VUID-vkCreateGraphicsPipelines-pCreateInfos-parameter
  `pCreateInfos` **must** be a valid pointer to an array of `createInfoCount` valid VkGraphicsPipelineCreateInfo structures
- VUID-vkCreateGraphicsPipelines-pAllocator-parameter
  If `pAllocator` is not `NULL`, `pAllocator` **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkCreateGraphicsPipelines-pPipelines-parameter
  `pPipelines` **must** be a valid pointer to an array of `createInfoCount` VkPipeline handles
- VUID-vkCreateGraphicsPipelines-createInfoCount-arraylength
  `createInfoCount` **must** be greater than `0`
- VUID-vkCreateGraphicsPipelines-pipelineCache-parent
  If `pipelineCache` is a valid handle, it **must** have been created, allocated, or retrieved from `device`

## Return Codes

### Success

- `VK_SUCCESS`
- `VK_PIPELINE_COMPILE_REQUIRED_EXT`

### Failure

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_INVALID_SHADER_NV`

The `VkGraphicsPipelineCreateInfo` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkGraphicsPipelineCreateInfo {
    VkStructureType                                 sType;
    const void*                                     pNext;
    VkPipelineCreateFlags                           flags;
    uint32_t                                        stageCount;
    const VkPipelineShaderStageCreateInfo*          pStages;
    const VkPipelineVertexInputStateCreateInfo*     pVertexInputState;
    const VkPipelineInputAssemblyStateCreateInfo*   pInputAssemblyState;
    const VkPipelineTessellationStateCreateInfo*    pTessellationState;
    const VkPipelineViewportStateCreateInfo*        pViewportState;
    const VkPipelineRasterizationStateCreateInfo*   pRasterizationState;
    const VkPipelineMultisampleStateCreateInfo*     pMultisampleState;
    const VkPipelineDepthStencilStateCreateInfo*    pDepthStencilState;
    const VkPipelineColorBlendStateCreateInfo*      pColorBlendState;
    const VkPipelineDynamicStateCreateInfo*         pDynamicState;
    VkPipelineLayout                                layout;
    VkRenderPass                                    renderPass;
    uint32_t                                        subpass;
    VkPipeline                                      basePipelineHandle;
    int32_t                                         basePipelineIndex;
} VkGraphicsPipelineCreateInfo;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is a bitmask of VkPipelineCreateFlagBits specifying how the pipeline will be generated.
- `stageCount` is the number of entries in the `pStages` array.
- `pStages` is a pointer to an array of `stageCount` VkPipelineShaderStageCreateInfo structures describing the set of the shader stages to be included in the graphics pipeline.
- `pVertexInputState` is a pointer to a VkPipelineVertexInputStateCreateInfo structure. It is ignored if the pipeline includes a mesh shader stage. It **can** be `NULL` if the pipeline is created with the `VK_DYNAMIC_STATE_VERTEX_INPUT_EXT` dynamic state set.
- `pInputAssemblyState` is a pointer to a VkPipelineInputAssemblyStateCreateInfo structure which determines input assembly behavior for vertex shading, as described in Drawing Commands. If the `VK_EXT_extended_dynamic_state3` extension is enabled, it **can** be `NULL` if the pipeline is created with both `VK_DYNAMIC_STATE_PRIMITIVE_RESTART_ENABLE`, and `VK_DYNAMIC_STATE_PRIMITIVE_TOPOLOGY` dynamic states set and `dynamicPrimitiveTopologyUnrestricted` is `VK_TRUE`. It is ignored if the pipeline includes a mesh shader stage.
- `pTessellationState` is a pointer to a VkPipelineTessellationStateCreateInfo structure defining tessellation state used by tessellation shaders. It **can** be `NULL` if the pipeline is created with the `VK_DYNAMIC_STATE_PATCH_CONTROL_POINTS_EXT` dynamic state set.
- `pViewportState` is a pointer to a VkPipelineViewportStateCreateInfo structure defining viewport state used when rasterization is enabled. If the `VK_EXT_extended_dynamic_state3` extension is enabled, it **can** be `NULL` if the pipeline is created with both `VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT`, and `VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT` dynamic states

set.

- pRasterizationState is a pointer to a VkPipelineRasterizationStateCreateInfo structure defining rasterization state. If the VK_EXT_extended_dynamic_state3 extension is enabled, it **can** be NULL if the pipeline is created with all of VK_DYNAMIC_STATE_DEPTH_CLAMP_ENABLE_EXT, VK_DYNAMIC_STATE_RASTERIZER_DISCARD_ENABLE, VK_DYNAMIC_STATE_POLYGON_MODE_EXT, VK_DYNAMIC_STATE_CULL_MODE, VK_DYNAMIC_STATE_FRONT_FACE, VK_DYNAMIC_STATE_DEPTH_BIAS_ENABLE, VK_DYNAMIC_STATE_DEPTH_BIAS, and VK_DYNAMIC_STATE_LINE_WIDTH dynamic states set.
- pMultisampleState is a pointer to a VkPipelineMultisampleStateCreateInfo structure defining multisample state used when rasterization is enabled. If the VK_EXT_extended_dynamic_state3 extension is enabled, it **can** be NULL if the pipeline is created with all of VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT, VK_DYNAMIC_STATE_SAMPLE_MASK_EXT, and VK_DYNAMIC_STATE_ALPHA_TO_COVERAGE_ENABLE_EXT dynamic states set, and either alphaToOne is disabled on the device or VK_DYNAMIC_STATE_ALPHA_TO_ONE_ENABLE_EXT is set, in which case VkPipelineMultisampleStateCreateInfo::sampleShadingEnable is assumed to be VK_FALSE.
- pDepthStencilState is a pointer to a VkPipelineDepthStencilStateCreateInfo structure defining depth/stencil state used when rasterization is enabled for depth or stencil attachments accessed during rendering. If the VK_EXT_extended_dynamic_state3 extension is enabled, it **can** be NULL if the pipeline is created with all of VK_DYNAMIC_STATE_DEPTH_TEST_ENABLE, VK_DYNAMIC_STATE_DEPTH_WRITE_ENABLE, VK_DYNAMIC_STATE_DEPTH_COMPARE_OP, VK_DYNAMIC_STATE_DEPTH_BOUNDS_TEST_ENABLE, VK_DYNAMIC_STATE_STENCIL_TEST_ENABLE, VK_DYNAMIC_STATE_STENCIL_OP, and VK_DYNAMIC_STATE_DEPTH_BOUNDS dynamic states set.
- pColorBlendState is a pointer to a VkPipelineColorBlendStateCreateInfo structure defining color blend state used when rasterization is enabled for any color attachments accessed during rendering. If the VK_EXT_extended_dynamic_state3 extension is enabled, it **can** be NULL if the pipeline is created with all of VK_DYNAMIC_STATE_LOGIC_OP_ENABLE_EXT, VK_DYNAMIC_STATE_LOGIC_OP_EXT, VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT, VK_DYNAMIC_STATE_COLOR_BLEND_EQUATION_EXT, VK_DYNAMIC_STATE_COLOR_WRITE_MASK_EXT, and VK_DYNAMIC_STATE_BLEND_CONSTANTS dynamic states set.
- pDynamicState is a pointer to a VkPipelineDynamicStateCreateInfo structure defining which properties of the pipeline state object are dynamic and **can** be changed independently of the pipeline state. This **can** be NULL, which means no state in the pipeline is considered dynamic.
- layout is the description of binding locations used by both the pipeline and descriptor sets used with the pipeline.
- renderPass is a handle to a render pass object describing the environment in which the pipeline will be used. The pipeline **must** only be used with a render pass instance compatible with the one provided. See Render Pass Compatibility for more information.
- subpass is the index of the subpass in the render pass where this pipeline will be used.
- basePipelineHandle is a pipeline to derive from.
- basePipelineIndex is an index into the pCreateInfos parameter to use as a pipeline to derive from.

The parameters basePipelineHandle and basePipelineIndex are described in more detail in Pipeline Derivatives.

If any shader stage fails to compile, the compile log will be reported back to the application, and VK_ERROR_INVALID_SHADER_NV will be generated.

| | |
|---|---|
| **NOTE** | With VK_EXT_extended_dynamic_state3, it is possible that many of the VkGraphicsPipelineCreateInfo members above **can** be NULL because all their state is dynamic and therefore ignored. This is optional so the application **can** still use a valid pointer if it needs to set the pNext or flags fields to specify state for other extensions. |

The state required for a graphics pipeline is divided into vertex input state, pre-rasterization shader state, fragment shader state, and fragment output state.

*Vertex Input State*

Vertex input state is defined by:

- VkPipelineVertexInputStateCreateInfo
- VkPipelineInputAssemblyStateCreateInfo

If this pipeline specifies pre-rasterization state either directly or by including it as a pipeline library and its pStages includes a vertex shader, this state **must** be specified to create a complete graphics pipeline.

If a pipeline includes VK_GRAPHICS_PIPELINE_LIBRARY_VERTEX_INPUT_INTERFACE_BIT_EXT in VkGraphicsPipelineLibraryCreateInfoEXT::flags either explicitly or as a default, and either the conditions requiring this state for a complete graphics pipeline are met or this pipeline does not specify pre-rasterization state in any way, that pipeline **must** specify this state directly.

*Pre-Rasterization Shader State*

Pre-rasterization shader state is defined by:

- VkPipelineShaderStageCreateInfo entries for:
  - Vertex shaders
  - Tessellation control shaders
  - Tessellation evaluation shaders
  - Geometry shaders
  - Task shaders
  - Mesh shaders
- Within the VkPipelineLayout, all descriptor sets with pre-rasterization shader bindings if VK_PIPELINE_LAYOUT_CREATE_INDEPENDENT_SETS_BIT_EXT was specified.
  - If VK_PIPELINE_LAYOUT_CREATE_INDEPENDENT_SETS_BIT_EXT was not specified, the full pipeline layout **must** be specified.
- VkPipelineViewportStateCreateInfo
- VkPipelineRasterizationStateCreateInfo
- VkPipelineTessellationStateCreateInfo

- VkRenderPass and subpass parameter
- The viewMask parameter of VkPipelineRenderingCreateInfo (formats are ignored)
- VkPipelineDiscardRectangleStateCreateInfoEXT
- VkPipelineFragmentShadingRateStateCreateInfoKHR

This state **must** be specified to create a complete graphics pipeline.

If either the pNext chain includes a VkGraphicsPipelineLibraryCreateInfoEXT structure with VK_GRAPHICS_PIPELINE_LIBRARY_PRE_RASTERIZATION_SHADERS_BIT_EXT included in flags, or it is not specified and would default to include that value, this state **must** be specified in the pipeline.

*Fragment Shader State*

Fragment shader state is defined by:

- A VkPipelineShaderStageCreateInfo entry for the fragment shader
- Within the VkPipelineLayout, all descriptor sets with fragment shader bindings if VK_PIPELINE_LAYOUT_CREATE_INDEPENDENT_SETS_BIT_EXT was specified.
  - If VK_PIPELINE_LAYOUT_CREATE_INDEPENDENT_SETS_BIT_EXT was not specified, the full pipeline layout **must** be specified.
- VkPipelineMultisampleStateCreateInfo if sample shading is enabled or renderpass is not VK_NULL_HANDLE
- VkPipelineDepthStencilStateCreateInfo
- VkRenderPass and subpass parameter
- The viewMask parameter of VkPipelineRenderingCreateInfo (formats are ignored)
- VkPipelineFragmentShadingRateStateCreateInfoKHR
- VkPipelineFragmentShadingRateEnumStateCreateInfoNV
- VkPipelineRepresentativeFragmentTestStateCreateInfoNV
- Inclusion/omission of the VK_PIPELINE_RASTERIZATION_STATE_CREATE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR flag
- Inclusion/omission of the VK_PIPELINE_RASTERIZATION_STATE_CREATE_FRAGMENT_DENSITY_MAP_ATTACHMENT_BIT_EXT flag
- VkRenderingInputAttachmentIndexInfoKHR

If a pipeline specifies pre-rasterization state either directly or by including it as a pipeline library and rasterizerDiscardEnable is set to VK_FALSE or VK_DYNAMIC_STATE_RASTERIZER_DISCARD_ENABLE is used, this state **must** be specified to create a complete graphics pipeline.

If a pipeline includes VK_GRAPHICS_PIPELINE_LIBRARY_FRAGMENT_SHADER_BIT_EXT in VkGraphicsPipelineLibraryCreateInfoEXT::flags either explicitly or as a default, and either the conditions requiring this state for a complete graphics pipeline are met or this pipeline does not specify pre-rasterization state in any way, that pipeline **must** specify this state directly.

*Fragment Output State*

Fragment output state is defined by:

- VkPipelineColorBlendStateCreateInfo
- VkRenderPass and `subpass` parameter
- VkPipelineMultisampleStateCreateInfo
- VkPipelineRenderingCreateInfo
- VkAttachmentSampleCountInfoAMD
- VkAttachmentSampleCountInfoNV
- VkExternalFormatANDROID
- Inclusion/omission of the `VK_PIPELINE_CREATE_COLOR_ATTACHMENT_FEEDBACK_LOOP_BIT_EXT` and `VK_PIPELINE_CREATE_DEPTH_STENCIL_ATTACHMENT_FEEDBACK_LOOP_BIT_EXT` flags
- Inclusion/omission of the `VK_PIPELINE_CREATE_2_ENABLE_LEGACY_DITHERING_BIT_EXT` flag
- VkRenderingAttachmentLocationInfoKHR

If a pipeline specifies pre-rasterization state either directly or by including it as a pipeline library and `rasterizerDiscardEnable` is set to `VK_FALSE` or `VK_DYNAMIC_STATE_RASTERIZER_DISCARD_ENABLE` is used, this state **must** be specified to create a complete graphics pipeline.

If a pipeline includes `VK_GRAPHICS_PIPELINE_LIBRARY_FRAGMENT_OUTPUT_INTERFACE_BIT_EXT` in VkGraphicsPipelineLibraryCreateInfoEXT::`flags` either explicitly or as a default, and either the conditions requiring this state for a complete graphics pipeline are met or this pipeline does not specify pre-rasterization state in any way, that pipeline **must** specify this state directly.

*Dynamic State*

Dynamic state values set via `pDynamicState` **must** be ignored if the state they correspond to is not otherwise statically set by one of the state subsets used to create the pipeline. Additionally, setting dynamic state values **must** not modify whether state in a linked library is static or dynamic; this is set and unchangeable when the library is created. For example, if a pipeline only included pre-rasterization shader state, then any dynamic state value corresponding to depth or stencil testing has no effect. Any linked library that has dynamic state enabled that same dynamic state **must** also be enabled in all the other linked libraries to which that dynamic state applies.

*Complete Graphics Pipelines*

A complete graphics pipeline always includes pre-rasterization shader state, with other subsets included depending on that state as specified in the above sections.

*Graphics Pipeline Library Layouts*

If different subsets are linked together with pipeline layouts created with `VK_PIPELINE_LAYOUT_CREATE_INDEPENDENT_SETS_BIT_EXT`, the final effective pipeline layout is effectively the union of the linked pipeline layouts. When binding descriptor sets for this pipeline, the pipeline layout used **must** be compatible with this union. This pipeline layout **can** be overridden when linking with `VK_PIPELINE_CREATE_LINK_TIME_OPTIMIZATION_BIT_EXT` by providing a VkPipelineLayout that is compatible with this union other than `VK_PIPELINE_LAYOUT_CREATE_INDEPENDENT_SETS_BIT_EXT`, or when linking without `VK_PIPELINE_CREATE_LINK_TIME_OPTIMIZATION_BIT_EXT` by providing a VkPipelineLayout that is fully compatible with this union.

If the pNext chain includes a VkPipelineCreateFlags2CreateInfoKHR structure, VkPipelineCreateFlags2CreateInfoKHR::flags from that structure is used instead of flags from this structure.

## Valid Usage

- VUID-VkGraphicsPipelineCreateInfo-None-09497
  If the pNext chain does not include a VkPipelineCreateFlags2CreateInfoKHR structure, flags **must** be a valid combination of VkPipelineCreateFlagBits values
- VUID-VkGraphicsPipelineCreateInfo-flags-07984
  If flags contains the VK_PIPELINE_CREATE_DERIVATIVE_BIT flag, and basePipelineIndex is -1, basePipelineHandle **must** be a valid graphics VkPipeline handle
- VUID-VkGraphicsPipelineCreateInfo-flags-07985
  If flags contains the VK_PIPELINE_CREATE_DERIVATIVE_BIT flag, and basePipelineHandle is VK_NULL_HANDLE, basePipelineIndex **must** be a valid index into the calling command's pCreateInfos parameter
- VUID-VkGraphicsPipelineCreateInfo-flags-07986
  If flags contains the VK_PIPELINE_CREATE_DERIVATIVE_BIT flag, basePipelineIndex **must** be -1 or basePipelineHandle **must** be VK_NULL_HANDLE
- VUID-VkGraphicsPipelineCreateInfo-layout-07987
  If a push constant block is declared in a shader, a push constant range in layout **must** match the shader stage
- VUID-VkGraphicsPipelineCreateInfo-layout-10069
  If a push constant block is declared in a shader, the block must be contained inside the push constant range in layout that matches the stage
- VUID-VkGraphicsPipelineCreateInfo-layout-07988
  If a resource variables is declared in a shader, a descriptor slot in layout **must** match the shader stage
- VUID-VkGraphicsPipelineCreateInfo-layout-07990
  If a resource variables is declared in a shader, and the descriptor type is not VK_DESCRIPTOR_TYPE_MUTABLE_EXT, a descriptor slot in layout **must** match the descriptor type
- VUID-VkGraphicsPipelineCreateInfo-layout-07991
  If a resource variables is declared in a shader as an array, a descriptor slot in layout **must** match the descriptor count
- VUID-VkGraphicsPipelineCreateInfo-stage-02096
  If the pipeline requires pre-rasterization shader state the stage member of one element of pStages **must** be VK_SHADER_STAGE_VERTEX_BIT or VK_SHADER_STAGE_MESH_BIT_EXT
- VUID-VkGraphicsPipelineCreateInfo-pStages-02095
  If the pipeline requires pre-rasterization shader state the geometric shader stages provided in pStages **must** be either from the mesh shading pipeline (stage is VK_SHADER_STAGE_TASK_BIT_EXT or VK_SHADER_STAGE_MESH_BIT_EXT) or from the primitive shading pipeline (stage is VK_SHADER_STAGE_VERTEX_BIT,

VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT, VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT, or VK_SHADER_STAGE_GEOMETRY_BIT)

- VUID-VkGraphicsPipelineCreateInfo-pStages-09631
  If the pipeline requires pre-rasterization shader state and pStages contains both VK_SHADER_STAGE_TASK_BIT_EXT and VK_SHADER_STAGE_MESH_BIT_EXT, then the mesh shader's entry point **must** not declare a variable with a DrawIndex BuiltIn decoration
- VUID-VkGraphicsPipelineCreateInfo-TaskNV-07063
  The shader stages for VK_SHADER_STAGE_TASK_BIT_EXT or VK_SHADER_STAGE_MESH_BIT_EXT **must** use either the TaskNV and MeshNV Execution Model or the TaskEXT and MeshEXT Execution Model, but **must** not use both
- VUID-VkGraphicsPipelineCreateInfo-pStages-00729
  If the pipeline requires pre-rasterization shader state and pStages includes a tessellation control shader stage, it **must** include a tessellation evaluation shader stage
- VUID-VkGraphicsPipelineCreateInfo-pStages-00730
  If the pipeline requires pre-rasterization shader state and pStages includes a tessellation evaluation shader stage, it **must** include a tessellation control shader stage
- VUID-VkGraphicsPipelineCreateInfo-pStages-09022
  If the pipeline requires pre-rasterization shader state and pStages includes a tessellation control shader stage, and the VK_EXT_extended_dynamic_state3 extension is not enabled or the VK_DYNAMIC_STATE_PATCH_CONTROL_POINTS_EXT dynamic state is not set, pTessellationState **must** be a valid pointer to a valid VkPipelineTessellationStateCreateInfo structure
- VUID-VkGraphicsPipelineCreateInfo-pTessellationState-09023
  If pTessellationState is not NULL it **must** be a pointer to a valid VkPipelineTessellationStateCreateInfo structure
- VUID-VkGraphicsPipelineCreateInfo-pStages-00732
  If the pipeline requires pre-rasterization shader state and pStages includes tessellation shader stages, the shader code of at least one stage **must** contain an OpExecutionMode instruction specifying the type of subdivision in the pipeline
- VUID-VkGraphicsPipelineCreateInfo-pStages-00733
  If the pipeline requires pre-rasterization shader state and pStages includes tessellation shader stages, and the shader code of both stages contain an OpExecutionMode instruction specifying the type of subdivision in the pipeline, they **must** both specify the same subdivision mode
- VUID-VkGraphicsPipelineCreateInfo-pStages-00734
  If the pipeline requires pre-rasterization shader state and pStages includes tessellation shader stages, the shader code of at least one stage **must** contain an OpExecutionMode instruction specifying the output patch size in the pipeline
- VUID-VkGraphicsPipelineCreateInfo-pStages-00735
  If the pipeline requires pre-rasterization shader state and pStages includes tessellation shader stages, and the shader code of both contain an OpExecutionMode instruction specifying the out patch size in the pipeline, they **must** both specify the same patch size
- VUID-VkGraphicsPipelineCreateInfo-pStages-08888

If the pipeline is being created with pre-rasterization shader state and vertex input state and pStages includes tessellation shader stages, and either VK_DYNAMIC_STATE_PRIMITIVE_TOPOLOGY dynamic state is not enabled or dynamicPrimitiveTopologyUnrestricted is VK_FALSE, the topology member of pInputAssembly **must** be VK_PRIMITIVE_TOPOLOGY_PATCH_LIST

- VUID-VkGraphicsPipelineCreateInfo-topology-08889
  If the pipeline is being created with pre-rasterization shader state and vertex input state and the topology member of pInputAssembly is VK_PRIMITIVE_TOPOLOGY_PATCH_LIST, and either VK_DYNAMIC_STATE_PRIMITIVE_TOPOLOGY dynamic state is not enabled or dynamicPrimitiveTopologyUnrestricted is VK_FALSE, then pStages **must** include tessellation shader stages
- VUID-VkGraphicsPipelineCreateInfo-TessellationEvaluation-07723
  If the pipeline is being created with a TessellationEvaluation Execution Model, no Geometry Execution Model, uses the PointMode Execution Mode, and shaderTessellationAndGeometryPointSize is enabled, a PointSize decorated variable **must** be written to if maintenance5 is not enabled
- VUID-VkGraphicsPipelineCreateInfo-topology-08773
  If the pipeline is being created with a Vertex Execution Model and no TessellationEvaluation or Geometry Execution Model, and the topology member of pInputAssembly is VK_PRIMITIVE_TOPOLOGY_POINT_LIST, and either VK_DYNAMIC_STATE_PRIMITIVE_TOPOLOGY dynamic state is not enabled or dynamicPrimitiveTopologyUnrestricted is VK_FALSE, a PointSize decorated variable **must** be written to if maintenance5 is not enabled
- VUID-VkGraphicsPipelineCreateInfo-maintenance5-08775
  If maintenance5 is enabled and a PointSize decorated variable is written to, all execution paths **must** write to a PointSize decorated variable
- VUID-VkGraphicsPipelineCreateInfo-TessellationEvaluation-07724
  If the pipeline is being created with a TessellationEvaluation Execution Model, no Geometry Execution Model, uses the PointMode Execution Mode, and shaderTessellationAndGeometryPointSize is not enabled, a PointSize decorated variable **must** not be written to
- VUID-VkGraphicsPipelineCreateInfo-shaderTessellationAndGeometryPointSize-08776
  If the pipeline is being created with a Geometry Execution Model, uses the OutputPoints Execution Mode, and shaderTessellationAndGeometryPointSize is enabled, a PointSize decorated variable **must** be written to for every vertex emitted if maintenance5 is not enabled
- VUID-VkGraphicsPipelineCreateInfo-Geometry-07726
  If the pipeline is being created with a Geometry Execution Model, uses the OutputPoints Execution Mode, and shaderTessellationAndGeometryPointSize is not enabled, a PointSize decorated variable **must** not be written to
- VUID-VkGraphicsPipelineCreateInfo-pStages-00738
  If the pipeline requires pre-rasterization shader state and pStages includes a geometry shader stage, and does not include any tessellation shader stages, its shader code **must** contain an OpExecutionMode instruction specifying an input primitive type that is compatible with the primitive topology specified in pInputAssembly

- VUID-VkGraphicsPipelineCreateInfo-pStages-00739
  If the pipeline requires pre-rasterization shader state and `pStages` includes a geometry shader stage, and also includes tessellation shader stages, its shader code **must** contain an `OpExecutionMode` instruction specifying an input primitive type that is compatible with the primitive topology that is output by the tessellation stages
- VUID-VkGraphicsPipelineCreateInfo-pStages-00740
  If the pipeline requires pre-rasterization shader state and fragment shader state, it includes both a fragment shader and a geometry shader, and the fragment shader code reads from an input variable that is decorated with `PrimitiveId`, then the geometry shader code **must** write to a matching output variable, decorated with `PrimitiveId`, in all execution paths
- VUID-VkGraphicsPipelineCreateInfo-PrimitiveId-06264
  If the pipeline requires pre-rasterization shader state, it includes a mesh shader and the fragment shader code reads from an input variable that is decorated with `PrimitiveId`, then the mesh shader code **must** write to a matching output variable, decorated with `PrimitiveId`, in all execution paths
- VUID-VkGraphicsPipelineCreateInfo-renderPass-06038
  If `renderPass` is not VK_NULL_HANDLE and the pipeline is being created with fragment shader state the fragment shader **must** not read from any input attachment that is defined as `VK_ATTACHMENT_UNUSED` in `subpass`
- VUID-VkGraphicsPipelineCreateInfo-pStages-00742
  If the pipeline requires pre-rasterization shader state and multiple pre-rasterization shader stages are included in `pStages`, the shader code for the entry points identified by those `pStages` and the rest of the state identified by this structure **must** adhere to the pipeline linking rules described in the Shader Interfaces chapter
- VUID-VkGraphicsPipelineCreateInfo-None-04889
  If the pipeline requires pre-rasterization shader state and fragment shader state, the fragment shader and last pre-rasterization shader stage and any relevant state **must** adhere to the pipeline linking rules described in the Shader Interfaces chapter
- VUID-VkGraphicsPipelineCreateInfo-renderPass-06041
  If `renderPass` is not VK_NULL_HANDLE, and the pipeline is being created with fragment output interface state, then for each color attachment in the subpass, if the potential format features of the format of the corresponding attachment description do not contain `VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT`, then the `blendEnable` member of the corresponding element of the `pAttachments` member of `pColorBlendState` **must** be `VK_FALSE`
- VUID-VkGraphicsPipelineCreateInfo-renderPass-07609
  If `renderPass` is not VK_NULL_HANDLE, the pipeline is being created with fragment output interface state, the `pColorBlendState` pointer is not `NULL`, the `attachmentCount` member of `pColorBlendState` is not ignored, and the subpass uses color attachments, the `attachmentCount` member of `pColorBlendState` **must** be equal to the `colorAttachmentCount` used to create `subpass`
- VUID-VkGraphicsPipelineCreateInfo-pDynamicStates-04130
  If the pipeline requires pre-rasterization shader state, and `pViewportState->pViewports` is not dynamic, then `pViewportState->pViewports` **must** be a valid pointer to an array of `pViewportState->viewportCount` valid `VkViewport` structures

- VUID-VkGraphicsPipelineCreateInfo-pDynamicStates-04131
  If the pipeline requires pre-rasterization shader state, and `pViewportState->pScissors` is not dynamic, then `pViewportState->pScissors` **must** be a valid pointer to an array of `pViewportState->scissorCount` `VkRect2D` structures
- VUID-VkGraphicsPipelineCreateInfo-pDynamicStates-00749
  If the pipeline requires pre-rasterization shader state, and the `wideLines` feature is not enabled, and no element of the `pDynamicStates` member of `pDynamicState` is `VK_DYNAMIC_STATE_LINE_WIDTH`, the `lineWidth` member of `pRasterizationState` **must** be `1.0`
- VUID-VkGraphicsPipelineCreateInfo-rasterizerDiscardEnable-09024
  If the pipeline requires pre-rasterization shader state, and the `VK_DYNAMIC_STATE_RASTERIZER_DISCARD_ENABLE` dynamic state is enabled or the `rasterizerDiscardEnable` member of `pRasterizationState` is `VK_FALSE`, and related dynamic state is not set, `pViewportState` **must** be a valid pointer to a valid VkPipelineViewportStateCreateInfo structure
- VUID-VkGraphicsPipelineCreateInfo-pViewportState-09025
  If `pViewportState` is not `NULL` it **must** be a valid pointer to a valid VkPipelineViewportStateCreateInfo structure
- VUID-VkGraphicsPipelineCreateInfo-pMultisampleState-09026
  If the pipeline requires fragment output interface state, and the `VK_EXT_extended_dynamic_state3` extension is not enabled or any of the `VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT`, `VK_DYNAMIC_STATE_SAMPLE_MASK_EXT`, or `VK_DYNAMIC_STATE_ALPHA_TO_COVERAGE_ENABLE_EXT` dynamic states is not set, or alphaToOne is enabled on the device and `VK_DYNAMIC_STATE_ALPHA_TO_ONE_ENABLE_EXT` is not set, `pMultisampleState` **must** be a valid pointer to a valid VkPipelineMultisampleStateCreateInfo structure
- VUID-VkGraphicsPipelineCreateInfo-pMultisampleState-09027
  If `pMultisampleState` is not `NULL` it **must** be a valid pointer to a valid VkPipelineMultisampleStateCreateInfo structure
- VUID-VkGraphicsPipelineCreateInfo-alphaToCoverageEnable-08891
  If the pipeline is being created with fragment shader state, the VkPipelineMultisampleStateCreateInfo::`alphaToCoverageEnable` is not ignored and is `VK_TRUE`, then the Fragment Output Interface **must** contain a variable for the alpha `Component` word in `Location` 0 at `Index` 0
- VUID-VkGraphicsPipelineCreateInfo-renderPass-09028
  If `renderPass` is not VK_NULL_HANDLE, the pipeline is being created with fragment shader state, and `subpass` uses a depth/stencil attachment, and related dynamic state is not set, `pDepthStencilState` **must** be a valid pointer to a valid VkPipelineDepthStencilStateCreateInfo structure
- VUID-VkGraphicsPipelineCreateInfo-pDepthStencilState-09029
  If `pDepthStencilState` is not `NULL` it **must** be a valid pointer to a valid VkPipelineDepthStencilStateCreateInfo structure
- VUID-VkGraphicsPipelineCreateInfo-renderPass-09030
  If `renderPass` is not VK_NULL_HANDLE, the pipeline is being created with fragment output interface state, and `subpass` uses color attachments, and related dynamic state is not set,

pColorBlendState **must** be a valid pointer to a valid VkPipelineColorBlendStateCreateInfo structure

- VUID-VkGraphicsPipelineCreateInfo-pDynamicStates-00754
  If the pipeline requires pre-rasterization shader state, the depthBiasClamp feature is not enabled, no element of the pDynamicStates member of pDynamicState is VK_DYNAMIC_STATE_DEPTH_BIAS, and the depthBiasEnable member of pRasterizationState is VK_TRUE, the depthBiasClamp member of pRasterizationState **must** be 0.0
- VUID-VkGraphicsPipelineCreateInfo-pDynamicStates-02510
  If the pipeline requires fragment shader state, the VK_EXT_depth_range_unrestricted extension is not enabled and no element of the pDynamicStates member of pDynamicState is VK_DYNAMIC_STATE_DEPTH_BOUNDS, and the depthBoundsTestEnable member of pDepthStencilState is VK_TRUE, the minDepthBounds and maxDepthBounds members of pDepthStencilState **must** be between 0.0 and 1.0, inclusive
- VUID-VkGraphicsPipelineCreateInfo-pDynamicStates-07610
  If the pipeline requires fragment shader state or fragment output interface state, and rasterizationSamples and sampleLocationsInfo are not dynamic, and VkPipelineSampleLocationsStateCreateInfoEXT::sampleLocationsEnable included in the pNext chain of pMultisampleState is VK_TRUE, sampleLocationsInfo.sampleLocationGridSize.width **must** evenly divide VkMultisamplePropertiesEXT::sampleLocationGridSize.width as returned by vkGetPhysicalDeviceMultisamplePropertiesEXT with a samples parameter equaling rasterizationSamples
- VUID-VkGraphicsPipelineCreateInfo-pDynamicStates-07611
  If the pipeline requires fragment shader state or fragment output interface state, and rasterizationSamples and sampleLocationsInfo are not dynamic, and VkPipelineSampleLocationsStateCreateInfoEXT::sampleLocationsEnable the included in the pNext chain of pMultisampleState is VK_TRUE or VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT is used, sampleLocationsInfo.sampleLocationGridSize.height **must** evenly divide VkMultisamplePropertiesEXT::sampleLocationGridSize.height as returned by vkGetPhysicalDeviceMultisamplePropertiesEXT with a samples parameter equaling rasterizationSamples
- VUID-VkGraphicsPipelineCreateInfo-pDynamicStates-07612
  If the pipeline requires fragment shader state or fragment output interface state, and rasterizationSamples and sampleLocationsInfo are not dynamic, and VkPipelineSampleLocationsStateCreateInfoEXT::sampleLocationsEnable included in the pNext chain of pMultisampleState is VK_TRUE or VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT is used, sampleLocationsInfo.sampleLocationsPerPixel **must** equal rasterizationSamples
- VUID-VkGraphicsPipelineCreateInfo-sampleLocationsEnable-01524
  If the pipeline requires fragment shader state, and the sampleLocationsEnable member of a VkPipelineSampleLocationsStateCreateInfoEXT structure included in the pNext chain of pMultisampleState is VK_TRUE, the fragment shader code **must** not statically use the extended instruction InterpolateAtSample
- VUID-VkGraphicsPipelineCreateInfo-multisampledRenderToSingleSampled-06853

If the pipeline requires fragment output interface state, and none of the VK_AMD_mixed_attachment_samples extension, the VK_NV_framebuffer_mixed_samples extension, or the multisampledRenderToSingleSampled feature are enabled, rasterizationSamples is not dynamic, and if subpass uses color and/or depth/stencil attachments, then the rasterizationSamples member of pMultisampleState **must** be the same as the sample count for those subpass attachments

- VUID-VkGraphicsPipelineCreateInfo-subpass-01505
  If the pipeline requires fragment output interface state, and the VK_AMD_mixed_attachment_samples extension is enabled, rasterizationSamples is not dynamic, and if subpass uses color and/or depth/stencil attachments, then the rasterizationSamples member of pMultisampleState **must** equal the maximum of the sample counts of those subpass attachments

- VUID-VkGraphicsPipelineCreateInfo-renderPass-06854
  If renderPass is not VK_NULL_HANDLE, the VK_EXT_multisampled_render_to_single_sampled extension is enabled, rasterizationSamples is not dynamic, and subpass has a VkMultisampledRenderToSingleSampledInfoEXT structure included in the VkSubpassDescription2::pNext chain with multisampledRenderToSingleSampledEnable equal to VK_TRUE, then the rasterizationSamples member of pMultisampleState **must** be equal to VkMultisampledRenderToSingleSampledInfoEXT::rasterizationSamples

- VUID-VkGraphicsPipelineCreateInfo-subpass-01411
  If the pipeline requires fragment output interface state, the VK_NV_framebuffer_mixed_samples extension is enabled, rasterizationSamples is not dynamic, and if subpass has a depth/stencil attachment and depth test, stencil test, or depth bounds test are enabled, then the rasterizationSamples member of pMultisampleState **must** be the same as the sample count of the depth/stencil attachment

- VUID-VkGraphicsPipelineCreateInfo-subpass-01412
  If the pipeline requires fragment output interface state, the VK_NV_framebuffer_mixed_samples extension is enabled, rasterizationSamples is not dynamic, and if subpass has any color attachments, then the rasterizationSamples member of pMultisampleState **must** be greater than or equal to the sample count for those subpass attachments

- VUID-VkGraphicsPipelineCreateInfo-coverageReductionMode-02722
  If the pipeline requires fragment output interface state, the VK_NV_coverage_reduction_mode extension is enabled, and rasterizationSamples is not dynamic, the coverage reduction mode specified by VkPipelineCoverageReductionStateCreateInfoNV::coverageReductionMode, the rasterizationSamples member of pMultisampleState and the sample counts for the color and depth/stencil attachments (if the subpass has them) **must** be a valid combination returned by vkGetPhysicalDeviceSupportedFramebufferMixedSamplesCombinationsNV

- VUID-VkGraphicsPipelineCreateInfo-subpass-00758
  If the pipeline requires fragment output interface state, rasterizationSamples is not dynamic, and subpass does not use any color and/or depth/stencil attachments, then the rasterizationSamples member of pMultisampleState **must** follow the rules for a zero-attachment subpass

- VUID-VkGraphicsPipelineCreateInfo-renderPass-06046

If renderPass is not VK_NULL_HANDLE, subpass **must** be a valid subpass within renderPass

- VUID-VkGraphicsPipelineCreateInfo-renderPass-06047
  If renderPass is not VK_NULL_HANDLE, the pipeline is being created with pre-rasterization shader state, subpass viewMask is not 0, and multiviewTessellationShader is not enabled, then pStages **must** not include tessellation shaders
- VUID-VkGraphicsPipelineCreateInfo-renderPass-06048
  If renderPass is not VK_NULL_HANDLE, the pipeline is being created with pre-rasterization shader state, subpass viewMask is not 0, and multiviewGeometryShader is not enabled, then pStages **must** not include a geometry shader
- VUID-VkGraphicsPipelineCreateInfo-renderPass-06050
  If renderPass is not VK_NULL_HANDLE and the pipeline is being created with pre-rasterization shader state, and subpass viewMask is not 0, then all of the shaders in the pipeline **must** not include variables decorated with the Layer built-in decoration in their interfaces
- VUID-VkGraphicsPipelineCreateInfo-renderPass-07064
  If renderPass is not VK_NULL_HANDLE, the pipeline is being created with pre-rasterization shader state, subpass viewMask is not 0, and multiviewMeshShader is not enabled, then pStages **must** not include a mesh shader
- VUID-VkGraphicsPipelineCreateInfo-flags-00764
  flags **must** not contain the VK_PIPELINE_CREATE_DISPATCH_BASE flag
- VUID-VkGraphicsPipelineCreateInfo-pStages-01565
  If the pipeline requires fragment shader state and an input attachment was referenced by an aspectMask at renderPass creation time, the fragment shader **must** only read from the aspects that were specified for that input attachment
- VUID-VkGraphicsPipelineCreateInfo-layout-01688
  The number of resources in layout accessible to each shader stage that is used by the pipeline **must** be less than or equal to VkPhysicalDeviceLimits::maxPerStageResources
- VUID-VkGraphicsPipelineCreateInfo-pDynamicStates-01715
  If the pipeline requires pre-rasterization shader state, and no element of the pDynamicStates member of pDynamicState is VK_DYNAMIC_STATE_VIEWPORT_W_SCALING_NV, and the viewportWScalingEnable member of a VkPipelineViewportWScalingStateCreateInfoNV structure, included in the pNext chain of pViewportState, is VK_TRUE, the pViewportWScalings member of the VkPipelineViewportWScalingStateCreateInfoNV **must** be a pointer to an array of VkPipelineViewportWScalingStateCreateInfoNV::viewportCount valid VkViewportWScalingNV structures
- VUID-VkGraphicsPipelineCreateInfo-pDynamicStates-04056
  If the pipeline requires pre-rasterization shader state, and no element of the pDynamicStates member of pDynamicState is VK_DYNAMIC_STATE_EXCLUSIVE_SCISSOR_NV, and if pViewportState->pNext chain includes a VkPipelineViewportExclusiveScissorStateCreateInfoNV structure, and if its exclusiveScissorCount member is not 0, then its pExclusiveScissors member **must** be a valid pointer to an array of exclusiveScissorCount VkRect2D structures
- VUID-VkGraphicsPipelineCreateInfo-pDynamicStates-07854
  If VK_DYNAMIC_STATE_EXCLUSIVE_SCISSOR_ENABLE_NV is included in the pDynamicStates array

then the implementation **must** support at least specVersion 2 of the VK_NV_scissor_exclusive extension

- VUID-VkGraphicsPipelineCreateInfo-pDynamicStates-04057
  If the pipeline requires pre-rasterization shader state, and no element of the pDynamicStates member of pDynamicState is VK_DYNAMIC_STATE_VIEWPORT_SHADING_RATE_PALETTE_NV, and if pViewportState->pNext chain includes a VkPipelineViewportShadingRateImageStateCreateInfoNV structure, then its pShadingRatePalettes member **must** be a valid pointer to an array of viewportCount valid VkShadingRatePaletteNV structures
- VUID-VkGraphicsPipelineCreateInfo-pDynamicStates-04058
  If the pipeline requires pre-rasterization shader state, and no element of the pDynamicStates member of pDynamicState is VK_DYNAMIC_STATE_DISCARD_RECTANGLE_EXT, and if pNext chain includes a VkPipelineDiscardRectangleStateCreateInfoEXT structure, and if its discardRectangleCount member is not 0, then its pDiscardRectangles member **must** be a valid pointer to an array of discardRectangleCount VkRect2D structures
- VUID-VkGraphicsPipelineCreateInfo-pDynamicStates-07855
  If VK_DYNAMIC_STATE_DISCARD_RECTANGLE_ENABLE_EXT is included in the pDynamicStates array then the implementation **must** support at least specVersion 2 of the VK_EXT_discard_rectangles extension
- VUID-VkGraphicsPipelineCreateInfo-pDynamicStates-07856
  If VK_DYNAMIC_STATE_DISCARD_RECTANGLE_MODE_EXT is included in the pDynamicStates array then the implementation **must** support at least specVersion 2 of the VK_EXT_discard_rectangles extension
- VUID-VkGraphicsPipelineCreateInfo-pStages-02097
  If the pipeline requires vertex input state, and pVertexInputState is not dynamic, then pVertexInputState **must** be a valid pointer to a valid VkPipelineVertexInputStateCreateInfo structure
- VUID-VkGraphicsPipelineCreateInfo-Input-07904
  If the pipeline is being created with vertex input state and pVertexInputState is not dynamic, then all variables with the Input storage class decorated with Location in the Vertex Execution Model OpEntryPoint **must** contain a location in VkVertexInputAttributeDescription::location
- VUID-VkGraphicsPipelineCreateInfo-Input-08733
  If the pipeline requires vertex input state and pVertexInputState is not dynamic, then the numeric type associated with all Input variables of the corresponding Location in the Vertex Execution Model OpEntryPoint **must** be the same as VkVertexInputAttributeDescription::format
- VUID-VkGraphicsPipelineCreateInfo-pVertexInputState-08929
  If the pipeline is being created with vertex input state and pVertexInputState is not dynamic, and VkVertexInputAttributeDescription::format has a 64-bit component, then the scalar width associated with all Input variables of the corresponding Location in the Vertex Execution Model OpEntryPoint **must** be 64-bit
- VUID-VkGraphicsPipelineCreateInfo-pVertexInputState-08930
  If the pipeline is being created with vertex input state and pVertexInputState is not

dynamic, and the scalar width associated with a `Location` decorated `Input` variable in the `Vertex Execution Model OpEntryPoint` is 64-bit, then the corresponding VkVertexInputAttributeDescription::`format` **must** have a 64-bit component

- VUID-VkGraphicsPipelineCreateInfo-pVertexInputState-09198
  If the pipeline is being created with vertex input state and `pVertexInputState` is not dynamic, and VkVertexInputAttributeDescription::`format` has a 64-bit component, then all `Input` variables at the corresponding `Location` in the `Vertex Execution Model OpEntryPoint` **must** not use components that are not present in the format
- VUID-VkGraphicsPipelineCreateInfo-dynamicPrimitiveTopologyUnrestricted-09031
  If the pipeline requires vertex input state, and related dynamic state is not set, `pInputAssemblyState` **must** be a valid pointer to a valid VkPipelineInputAssemblyStateCreateInfo structure
- VUID-VkGraphicsPipelineCreateInfo-pInputAssemblyState-09032
  If `pInputAssemblyState` is not `NULL` it **must** be a valid pointer to a valid VkPipelineInputAssemblyStateCreateInfo structure
- VUID-VkGraphicsPipelineCreateInfo-pStages-02317
  If the pipeline requires pre-rasterization shader state, the `Xfb` execution mode **can** be specified by no more than one shader stage in `pStages`
- VUID-VkGraphicsPipelineCreateInfo-pStages-02318
  If the pipeline requires pre-rasterization shader state, and any shader stage in `pStages` specifies `Xfb` execution mode it **must** be the last pre-rasterization shader stage
- VUID-VkGraphicsPipelineCreateInfo-rasterizationStream-02319
  If the pipeline requires pre-rasterization shader state, and a VkPipelineRasterizationStateStreamCreateInfoEXT::`rasterizationStream` value other than zero is specified, all variables in the output interface of the entry point being compiled decorated with `Position`, `PointSize`, `ClipDistance`, or `CullDistance` **must** be decorated with identical `Stream` values that match the `rasterizationStream`
- VUID-VkGraphicsPipelineCreateInfo-rasterizationStream-02320
  If the pipeline requires pre-rasterization shader state, and VkPipelineRasterizationStateStreamCreateInfoEXT::`rasterizationStream` is zero, or not specified, all variables in the output interface of the entry point being compiled decorated with `Position`, `PointSize`, `ClipDistance`, or `CullDistance` **must** be decorated with a `Stream` value of zero, or **must** not specify the `Stream` decoration
- VUID-VkGraphicsPipelineCreateInfo-geometryStreams-02321
  If the pipeline requires pre-rasterization shader state, and the last pre-rasterization shader stage is a geometry shader, and that geometry shader uses the `GeometryStreams` capability, then `VkPhysicalDeviceTransformFeedbackFeaturesEXT`::`geometryStreams` feature **must** be enabled
- VUID-VkGraphicsPipelineCreateInfo-None-02322
  If the pipeline requires pre-rasterization shader state, and there are any mesh shader stages in the pipeline there **must** not be any shader stage in the pipeline with a `Xfb` execution mode
- VUID-VkGraphicsPipelineCreateInfo-lineRasterizationMode-02766
  If the pipeline requires pre-rasterization shader state and at least one of fragment output

interface state or fragment shader state, and pMultisampleState is not NULL, the lineRasterizationMode member of a VkPipelineRasterizationLineStateCreateInfoKHR structure included in the pNext chain of pRasterizationState is VK_LINE_RASTERIZATION_MODE_BRESENHAM_KHR or VK_LINE_RASTERIZATION_MODE_RECTANGULAR_SMOOTH_KHR, then the alphaToCoverageEnable, alphaToOneEnable, and sampleShadingEnable members of pMultisampleState **must** all be VK_FALSE

- VUID-VkGraphicsPipelineCreateInfo-stippledLineEnable-02767
  If the pipeline requires pre-rasterization shader state, the stippledLineEnable member of VkPipelineRasterizationLineStateCreateInfoKHR is VK_TRUE, and no element of the pDynamicStates member of pDynamicState is VK_DYNAMIC_STATE_LINE_STIPPLE_EXT, then the lineStippleFactor member of VkPipelineRasterizationLineStateCreateInfoKHR **must** be in the range [1,256]
- VUID-VkGraphicsPipelineCreateInfo-flags-03372
  flags **must** not include VK_PIPELINE_CREATE_RAY_TRACING_NO_NULL_ANY_HIT_SHADERS_BIT_KHR
- VUID-VkGraphicsPipelineCreateInfo-flags-03373
  flags **must** not include VK_PIPELINE_CREATE_RAY_TRACING_NO_NULL_CLOSEST_HIT_SHADERS_BIT_KHR
- VUID-VkGraphicsPipelineCreateInfo-flags-03374
  flags **must** not include VK_PIPELINE_CREATE_RAY_TRACING_NO_NULL_MISS_SHADERS_BIT_KHR
- VUID-VkGraphicsPipelineCreateInfo-flags-03375
  flags **must** not include VK_PIPELINE_CREATE_RAY_TRACING_NO_NULL_INTERSECTION_SHADERS_BIT_KHR
- VUID-VkGraphicsPipelineCreateInfo-flags-03376
  flags **must** not include VK_PIPELINE_CREATE_RAY_TRACING_SKIP_TRIANGLES_BIT_KHR
- VUID-VkGraphicsPipelineCreateInfo-flags-03377
  flags **must** not include VK_PIPELINE_CREATE_RAY_TRACING_SKIP_AABBS_BIT_KHR
- VUID-VkGraphicsPipelineCreateInfo-flags-03577
  flags **must** not include VK_PIPELINE_CREATE_RAY_TRACING_SHADER_GROUP_HANDLE_CAPTURE_REPLAY_BIT_KHR
- VUID-VkGraphicsPipelineCreateInfo-flags-04947
  flags **must** not include VK_PIPELINE_CREATE_RAY_TRACING_ALLOW_MOTION_BIT_NV
- VUID-VkGraphicsPipelineCreateInfo-pDynamicStates-03378
  If the extendedDynamicState feature is not enabled, there **must** be no element of the pDynamicStates member of pDynamicState set to VK_DYNAMIC_STATE_CULL_MODE, VK_DYNAMIC_STATE_FRONT_FACE, VK_DYNAMIC_STATE_PRIMITIVE_TOPOLOGY, VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT, VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT, VK_DYNAMIC_STATE_VERTEX_INPUT_BINDING_STRIDE, VK_DYNAMIC_STATE_DEPTH_TEST_ENABLE, VK_DYNAMIC_STATE_DEPTH_WRITE_ENABLE, VK_DYNAMIC_STATE_DEPTH_COMPARE_OP, VK_DYNAMIC_STATE_DEPTH_BOUNDS_TEST_ENABLE, VK_DYNAMIC_STATE_STENCIL_TEST_ENABLE, or VK_DYNAMIC_STATE_STENCIL_OP
- VUID-VkGraphicsPipelineCreateInfo-pDynamicStates-03379
  If the pipeline requires pre-rasterization shader state, and

VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT is included in the pDynamicStates array then viewportCount **must** be zero

- VUID-VkGraphicsPipelineCreateInfo-pDynamicStates-03380
  If the pipeline requires pre-rasterization shader state, and VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT is included in the pDynamicStates array then scissorCount **must** be zero
- VUID-VkGraphicsPipelineCreateInfo-pDynamicStates-04132
  If the pipeline requires pre-rasterization shader state, and VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT is included in the pDynamicStates array then VK_DYNAMIC_STATE_VIEWPORT **must** not be present
- VUID-VkGraphicsPipelineCreateInfo-pDynamicStates-04133
  If the pipeline requires pre-rasterization shader state, and VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT is included in the pDynamicStates array then VK_DYNAMIC_STATE_SCISSOR **must** not be present
- VUID-VkGraphicsPipelineCreateInfo-pDynamicStates-07065
  If the pipeline requires pre-rasterization shader state, and includes a mesh shader, there **must** be no element of the pDynamicStates member of pDynamicState set to VK_DYNAMIC_STATE_PRIMITIVE_TOPOLOGY, or VK_DYNAMIC_STATE_VERTEX_INPUT_BINDING_STRIDE
- VUID-VkGraphicsPipelineCreateInfo-pDynamicStates-04868
  If the extendedDynamicState2 feature is not enabled, there **must** be no element of the pDynamicStates member of pDynamicState set to VK_DYNAMIC_STATE_DEPTH_BIAS_ENABLE, VK_DYNAMIC_STATE_PRIMITIVE_RESTART_ENABLE, or VK_DYNAMIC_STATE_RASTERIZER_DISCARD_ENABLE
- VUID-VkGraphicsPipelineCreateInfo-pDynamicStates-04869
  If the extendedDynamicState2LogicOp feature is not enabled, there **must** be no element of the pDynamicStates member of pDynamicState set to VK_DYNAMIC_STATE_LOGIC_OP_EXT
- VUID-VkGraphicsPipelineCreateInfo-pDynamicStates-04870
  If the extendedDynamicState2PatchControlPoints feature is not enabled, there **must** be no element of the pDynamicStates member of pDynamicState set to VK_DYNAMIC_STATE_PATCH_CONTROL_POINTS_EXT
- VUID-VkGraphicsPipelineCreateInfo-pDynamicStates-07066
  If the pipeline requires pre-rasterization shader state, and includes a mesh shader, there **must** be no element of the pDynamicStates member of pDynamicState set to VK_DYNAMIC_STATE_PRIMITIVE_RESTART_ENABLE, or VK_DYNAMIC_STATE_PATCH_CONTROL_POINTS_EXT
- VUID-VkGraphicsPipelineCreateInfo-flags-02877
  If flags includes VK_PIPELINE_CREATE_INDIRECT_BINDABLE_BIT_NV, then the deviceGeneratedCommands feature **must** be enabled
- VUID-VkGraphicsPipelineCreateInfo-flags-02966
  If the pipeline requires pre-rasterization shader state and flags includes VK_PIPELINE_CREATE_INDIRECT_BINDABLE_BIT_NV, then all stages **must** not specify Xfb execution mode
- VUID-VkGraphicsPipelineCreateInfo-libraryCount-06648
  If the pipeline is not created with a complete set of state, or

VkPipelineLibraryCreateInfoKHR::libraryCount is not 0, VkGraphicsPipelineShaderGroupsCreateInfoNV::groupCount and VkGraphicsPipelineShaderGroupsCreateInfoNV::pipelineCount **must** be 0

- VUID-VkGraphicsPipelineCreateInfo-libraryCount-06649
  If the pipeline is created with a complete set of state, and VkPipelineLibraryCreateInfoKHR::libraryCount is 0, and the pNext chain includes an instance of VkGraphicsPipelineShaderGroupsCreateInfoNV, VkGraphicsPipelineShaderGroupsCreateInfoNV::groupCount **must** be greater than 0
- VUID-VkGraphicsPipelineCreateInfo-pipelineCreationCacheControl-02878
  If the pipelineCreationCacheControl feature is not enabled, flags **must** not include VK_PIPELINE_CREATE_FAIL_ON_PIPELINE_COMPILE_REQUIRED_BIT or VK_PIPELINE_CREATE_EARLY_RETURN_ON_FAILURE_BIT
- VUID-VkGraphicsPipelineCreateInfo-pipelineProtectedAccess-07368
  If the pipelineProtectedAccess feature is not enabled, flags **must** not include VK_PIPELINE_CREATE_NO_PROTECTED_ACCESS_BIT_EXT or VK_PIPELINE_CREATE_PROTECTED_ACCESS_ONLY_BIT_EXT
- VUID-VkGraphicsPipelineCreateInfo-flags-07369
  flags **must** not include both VK_PIPELINE_CREATE_NO_PROTECTED_ACCESS_BIT_EXT and VK_PIPELINE_CREATE_PROTECTED_ACCESS_ONLY_BIT_EXT
- VUID-VkGraphicsPipelineCreateInfo-pDynamicState-04494
  If the pipeline requires pre-rasterization shader state or fragment shader state and VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR is not included in pDynamicState->pDynamicStates, VkPipelineFragmentShadingRateStateCreateInfoKHR::fragmentSize.width **must** be greater than or equal to 1
- VUID-VkGraphicsPipelineCreateInfo-pDynamicState-04495
  If the pipeline requires pre-rasterization shader state or fragment shader state and VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR is not included in pDynamicState->pDynamicStates, VkPipelineFragmentShadingRateStateCreateInfoKHR::fragmentSize.height **must** be greater than or equal to 1
- VUID-VkGraphicsPipelineCreateInfo-pDynamicState-04496
  If the pipeline requires pre-rasterization shader state or fragment shader state and VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR is not included in pDynamicState->pDynamicStates, VkPipelineFragmentShadingRateStateCreateInfoKHR::fragmentSize.width **must** be a power-of-two value
- VUID-VkGraphicsPipelineCreateInfo-pDynamicState-04497
  If the pipeline requires pre-rasterization shader state or fragment shader state and VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR is not included in pDynamicState->pDynamicStates, VkPipelineFragmentShadingRateStateCreateInfoKHR::fragmentSize.height **must** be a power-of-two value
- VUID-VkGraphicsPipelineCreateInfo-pDynamicState-04498
  If the pipeline requires pre-rasterization shader state or fragment shader state and VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR is not included in pDynamicState->pDynamicStates, VkPipelineFragmentShadingRateStateCreateInfoKHR::fragmentSize.width **must** be less than or equal to 4

- VUID-VkGraphicsPipelineCreateInfo-pDynamicState-04499
  If the pipeline requires pre-rasterization shader state or fragment shader state and VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR is not included in pDynamicState->pDynamicStates, VkPipelineFragmentShadingRateStateCreateInfoKHR::fragmentSize.height **must** be less than or equal to 4
- VUID-VkGraphicsPipelineCreateInfo-pDynamicState-04500
  If the pipeline requires pre-rasterization shader state or fragment shader state and VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR is not included in pDynamicState->pDynamicStates, and the pipelineFragmentShadingRate feature is not enabled, VkPipelineFragmentShadingRateStateCreateInfoKHR::fragmentSize.width and VkPipelineFragmentShadingRateStateCreateInfoKHR::fragmentSize.height **must** both be equal to 1
- VUID-VkGraphicsPipelineCreateInfo-pDynamicState-06567
  If the pipeline requires pre-rasterization shader state or fragment shader state and VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR is not included in pDynamicState->pDynamicStates, VkPipelineFragmentShadingRateStateCreateInfoKHR::combinerOps[0] **must** be a valid VkFragmentShadingRateCombinerOpKHR value
- VUID-VkGraphicsPipelineCreateInfo-pDynamicState-06568
  If the pipeline requires pre-rasterization shader state or fragment shader state and VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR is not included in pDynamicState->pDynamicStates, VkPipelineFragmentShadingRateStateCreateInfoKHR::combinerOps[1] **must** be a valid VkFragmentShadingRateCombinerOpKHR value
- VUID-VkGraphicsPipelineCreateInfo-pDynamicState-04501
  If the pipeline requires pre-rasterization shader state or fragment shader state and VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR is not included in pDynamicState->pDynamicStates, and the primitiveFragmentShadingRate feature is not enabled, VkPipelineFragmentShadingRateStateCreateInfoKHR::combinerOps[0] **must** be VK_FRAGMENT_SHADING_RATE_COMBINER_OP_KEEP_KHR
- VUID-VkGraphicsPipelineCreateInfo-pDynamicState-04502
  If the pipeline requires pre-rasterization shader state or fragment shader state and VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR is not included in pDynamicState->pDynamicStates, and the attachmentFragmentShadingRate feature is not enabled, VkPipelineFragmentShadingRateStateCreateInfoKHR::combinerOps[1] **must** be VK_FRAGMENT_SHADING_RATE_COMBINER_OP_KEEP_KHR
- VUID-VkGraphicsPipelineCreateInfo-primitiveFragmentShadingRateWithMultipleViewports-04503
  If the pipeline requires pre-rasterization shader state and the primitiveFragmentShadingRateWithMultipleViewports limit is not supported, VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT is not included in pDynamicState->pDynamicStates, and VkPipelineViewportStateCreateInfo::viewportCount is greater than 1, entry points specified in pStages **must** not write to the PrimitiveShadingRateKHR built-in
- VUID-VkGraphicsPipelineCreateInfo-primitiveFragmentShadingRateWithMultipleViewports-04504
  If the pipeline requires pre-rasterization shader state and the primitiveFragmentShadingRateWithMultipleViewports limit is not supported, and entry

points specified in pStages write to the ViewportIndex built-in, they **must** not also write to the PrimitiveShadingRateKHR built-in

- VUID-VkGraphicsPipelineCreateInfo-primitiveFragmentShadingRateWithMultipleViewports-04505
  If the pipeline requires pre-rasterization shader state and the primitiveFragmentShadingRateWithMultipleViewports limit is not supported, and entry points specified in pStages write to the ViewportMaskNV built-in, they **must** not also write to the PrimitiveShadingRateKHR built-in

- VUID-VkGraphicsPipelineCreateInfo-fragmentShadingRateNonTrivialCombinerOps-04506
  If the pipeline requires pre-rasterization shader state or fragment shader state, the fragmentShadingRateNonTrivialCombinerOps limit is not supported, and VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR is not included in pDynamicState->pDynamicStates, elements of VkPipelineFragmentShadingRateStateCreateInfoKHR::combinerOps **must** be VK_FRAGMENT_SHADING_RATE_COMBINER_OP_KEEP_KHR or VK_FRAGMENT_SHADING_RATE_COMBINER_OP_REPLACE_KHR

- VUID-VkGraphicsPipelineCreateInfo-None-06569
  If the pipeline requires fragment shader state and VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR is not included in pDynamicState->pDynamicStates, VkPipelineFragmentShadingRateEnumStateCreateInfoNV::shadingRateType **must** be a valid VkFragmentShadingRateTypeNV value

- VUID-VkGraphicsPipelineCreateInfo-pDynamicState-06570
  If the pipeline requires fragment shader state and VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR is not included in pDynamicState->pDynamicStates, VkPipelineFragmentShadingRateEnumStateCreateInfoNV::shadingRate **must** be a valid VkFragmentShadingRateNV value

- VUID-VkGraphicsPipelineCreateInfo-pDynamicState-06571
  If the pipeline requires fragment shader state and VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR is not included in pDynamicState->pDynamicStates, VkPipelineFragmentShadingRateEnumStateCreateInfoNV::combinerOps[0] **must** be a valid VkFragmentShadingRateCombinerOpKHR value

- VUID-VkGraphicsPipelineCreateInfo-pDynamicState-06572
  If the pipeline requires fragment shader state and VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR is not included in pDynamicState->pDynamicStates, VkPipelineFragmentShadingRateEnumStateCreateInfoNV::combinerOps[1] **must** be a valid VkFragmentShadingRateCombinerOpKHR value

- VUID-VkGraphicsPipelineCreateInfo-pDynamicState-04569
  If the pipeline requires fragment shader state and VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR is not included in pDynamicState->pDynamicStates, and the fragmentShadingRateEnums feature is not enabled, VkPipelineFragmentShadingRateEnumStateCreateInfoNV::shadingRateType **must** be equal to VK_FRAGMENT_SHADING_RATE_TYPE_FRAGMENT_SIZE_NV

- VUID-VkGraphicsPipelineCreateInfo-pDynamicState-04570
  If the pipeline requires fragment shader state and VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR is not included in pDynamicState->pDynamicStates, and the pipelineFragmentShadingRate feature is not enabled,

VkPipelineFragmentShadingRateEnumStateCreateInfoNV::shadingRate **must** be equal to VK_FRAGMENT_SHADING_RATE_1_INVOCATION_PER_PIXEL_NV

- VUID-VkGraphicsPipelineCreateInfo-pDynamicState-04571
  If the pipeline requires fragment shader state and VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR is not included in pDynamicState->pDynamicStates, and the primitiveFragmentShadingRate feature is not enabled, VkPipelineFragmentShadingRateEnumStateCreateInfoNV::combinerOps[0] **must** be VK_FRAGMENT_SHADING_RATE_COMBINER_OP_KEEP_KHR
- VUID-VkGraphicsPipelineCreateInfo-pDynamicState-04572
  If the pipeline requires fragment shader state and VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR is not included in pDynamicState->pDynamicStates, and the attachmentFragmentShadingRate feature is not enabled, VkPipelineFragmentShadingRateEnumStateCreateInfoNV::combinerOps[1] **must** be VK_FRAGMENT_SHADING_RATE_COMBINER_OP_KEEP_KHR
- VUID-VkGraphicsPipelineCreateInfo-fragmentShadingRateNonTrivialCombinerOps-04573
  If the pipeline requires fragment shader state, and the fragmentShadingRateNonTrivialCombinerOps limit is not supported and VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR is not included in pDynamicState->pDynamicStates, elements of VkPipelineFragmentShadingRateEnumStateCreateInfoNV::combinerOps **must** be VK_FRAGMENT_SHADING_RATE_COMBINER_OP_KEEP_KHR or VK_FRAGMENT_SHADING_RATE_COMBINER_OP_REPLACE_KHR
- VUID-VkGraphicsPipelineCreateInfo-None-04574
  If the pipeline requires fragment shader state, and the supersampleFragmentShadingRates feature is not enabled, VkPipelineFragmentShadingRateEnumStateCreateInfoNV::shadingRate **must** not be equal to VK_FRAGMENT_SHADING_RATE_2_INVOCATIONS_PER_PIXEL_NV, VK_FRAGMENT_SHADING_RATE_4_INVOCATIONS_PER_PIXEL_NV, VK_FRAGMENT_SHADING_RATE_8_INVOCATIONS_PER_PIXEL_NV, or VK_FRAGMENT_SHADING_RATE_16_INVOCATIONS_PER_PIXEL_NV
- VUID-VkGraphicsPipelineCreateInfo-None-04575
  If the pipeline requires fragment shader state, and the noInvocationFragmentShadingRates feature is not enabled, VkPipelineFragmentShadingRateEnumStateCreateInfoNV::shadingRate **must** not be equal to VK_FRAGMENT_SHADING_RATE_NO_INVOCATIONS_NV
- VUID-VkGraphicsPipelineCreateInfo-pDynamicStates-03578
  All elements of the pDynamicStates member of pDynamicState **must** not be VK_DYNAMIC_STATE_RAY_TRACING_PIPELINE_STACK_SIZE_KHR
- VUID-VkGraphicsPipelineCreateInfo-pDynamicStates-04807
  If the pipeline requires pre-rasterization shader state and the vertexInputDynamicState feature is not enabled, there **must** be no element of the pDynamicStates member of pDynamicState set to VK_DYNAMIC_STATE_VERTEX_INPUT_EXT
- VUID-VkGraphicsPipelineCreateInfo-pDynamicStates-07067
  If the pipeline requires pre-rasterization shader state, and includes a mesh shader, there **must** be no element of the pDynamicStates member of pDynamicState set to VK_DYNAMIC_STATE_VERTEX_INPUT_EXT
- VUID-VkGraphicsPipelineCreateInfo-pDynamicStates-04800

If the colorWriteEnable feature is not enabled, there **must** be no element of the pDynamicStates member of pDynamicState set to VK_DYNAMIC_STATE_COLOR_WRITE_ENABLE_EXT

- VUID-VkGraphicsPipelineCreateInfo-rasterizationSamples-04899
  If the pipeline requires fragment shader state, and the VK_QCOM_render_pass_shader_resolve extension is enabled, rasterizationSamples is not dynamic, and if subpass has any input attachments, and if the subpass description contains VK_SUBPASS_DESCRIPTION_FRAGMENT_REGION_BIT_QCOM, then the sample count of the input attachments **must** equal rasterizationSamples
- VUID-VkGraphicsPipelineCreateInfo-sampleShadingEnable-04900
  If the pipeline requires fragment shader state, and the VK_QCOM_render_pass_shader_resolve extension is enabled, and if the subpass description contains VK_SUBPASS_DESCRIPTION_FRAGMENT_REGION_BIT_QCOM, then sampleShadingEnable **must** be false
- VUID-VkGraphicsPipelineCreateInfo-flags-04901
  If flags includes VK_SUBPASS_DESCRIPTION_SHADER_RESOLVE_BIT_QCOM, then the subpass **must** be the last subpass in a subpass dependency chain
- VUID-VkGraphicsPipelineCreateInfo-flags-04902
  If flags includes VK_SUBPASS_DESCRIPTION_SHADER_RESOLVE_BIT_QCOM, and if pResolveAttachments is not NULL, then each resolve attachment **must** be VK_ATTACHMENT_UNUSED
- VUID-VkGraphicsPipelineCreateInfo-dynamicRendering-06576
  If the dynamicRendering feature is not enabled and the pipeline requires pre-rasterization shader state, fragment shader state, or fragment output interface state, renderPass **must** not be VK_NULL_HANDLE
- VUID-VkGraphicsPipelineCreateInfo-multiview-06577
  If the multiview feature is not enabled, the pipeline requires pre-rasterization shader state, fragment shader state, or fragment output interface state, and renderPass is VK_NULL_HANDLE, VkPipelineRenderingCreateInfo::viewMask **must** be 0
- VUID-VkGraphicsPipelineCreateInfo-renderPass-06578
  If the pipeline requires pre-rasterization shader state, fragment shader state, or fragment output interface state, and renderPass is VK_NULL_HANDLE, the index of the most significant bit in VkPipelineRenderingCreateInfo::viewMask **must** be less than maxMultiviewViewCount
- VUID-VkGraphicsPipelineCreateInfo-renderPass-06579
  If the pipeline requires fragment output interface state, and renderPass is VK_NULL_HANDLE, and VkPipelineRenderingCreateInfo::colorAttachmentCount is not 0, VkPipelineRenderingCreateInfo::pColorAttachmentFormats **must** be a valid pointer to an array of colorAttachmentCount valid VkFormat values
- VUID-VkGraphicsPipelineCreateInfo-renderPass-06580
  If the pipeline requires fragment output interface state, and renderPass is VK_NULL_HANDLE, each element of VkPipelineRenderingCreateInfo::pColorAttachmentFormats **must** be a valid VkFormat value
- VUID-VkGraphicsPipelineCreateInfo-renderPass-06582
  If the pipeline requires fragment output interface state, renderPass is VK_NULL_HANDLE,

and any element of VkPipelineRenderingCreateInfo::pColorAttachmentFormats is not VK_FORMAT_UNDEFINED, that format **must** be a format with potential format features that include VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BIT or VK_FORMAT_FEATURE_2_LINEAR_COLOR_ATTACHMENT_BIT_NV

- VUID-VkGraphicsPipelineCreateInfo-renderPass-06583
  If the pipeline requires fragment output interface state, and renderPass is VK_NULL_HANDLE, VkPipelineRenderingCreateInfo::depthAttachmentFormat **must** be a valid VkFormat value
- VUID-VkGraphicsPipelineCreateInfo-renderPass-06584
  If the pipeline requires fragment output interface state, and renderPass is VK_NULL_HANDLE, VkPipelineRenderingCreateInfo::stencilAttachmentFormat **must** be a valid VkFormat value
- VUID-VkGraphicsPipelineCreateInfo-renderPass-06585
  If the pipeline requires fragment output interface state, renderPass is VK_NULL_HANDLE, and VkPipelineRenderingCreateInfo::depthAttachmentFormat is not VK_FORMAT_UNDEFINED, it **must** be a format with potential format features that include VK_FORMAT_FEATURE_DEPTH_STENCIL_ATTACHMENT_BIT
- VUID-VkGraphicsPipelineCreateInfo-renderPass-06586
  If the pipeline requires fragment output interface state, renderPass is VK_NULL_HANDLE, and VkPipelineRenderingCreateInfo::stencilAttachmentFormat is not VK_FORMAT_UNDEFINED, it **must** be a format with potential format features that include VK_FORMAT_FEATURE_DEPTH_STENCIL_ATTACHMENT_BIT
- VUID-VkGraphicsPipelineCreateInfo-renderPass-06587
  If the pipeline requires fragment output interface state, renderPass is VK_NULL_HANDLE, and VkPipelineRenderingCreateInfo::depthAttachmentFormat is not VK_FORMAT_UNDEFINED, it **must** be a format that includes a depth component
- VUID-VkGraphicsPipelineCreateInfo-renderPass-06588
  If the pipeline requires fragment output interface state, renderPass is VK_NULL_HANDLE, and VkPipelineRenderingCreateInfo::stencilAttachmentFormat is not VK_FORMAT_UNDEFINED, it **must** be a format that includes a stencil component
- VUID-VkGraphicsPipelineCreateInfo-renderPass-06589
  If the pipeline requires fragment output interface state, renderPass is VK_NULL_HANDLE, VkPipelineRenderingCreateInfo::depthAttachmentFormat is not VK_FORMAT_UNDEFINED, and VkPipelineRenderingCreateInfo::stencilAttachmentFormat is not VK_FORMAT_UNDEFINED, depthAttachmentFormat **must** equal stencilAttachmentFormat
- VUID-VkGraphicsPipelineCreateInfo-renderPass-09033
  If renderPass is VK_NULL_HANDLE, the pipeline is being created with fragment shader state and fragment output interface state, and either of VkPipelineRenderingCreateInfo::depthAttachmentFormat or VkPipelineRenderingCreateInfo::stencilAttachmentFormat are not VK_FORMAT_UNDEFINED, and the VK_EXT_extended_dynamic_state3 extension is not enabled or any of the VK_DYNAMIC_STATE_DEPTH_TEST_ENABLE, VK_DYNAMIC_STATE_DEPTH_WRITE_ENABLE, VK_DYNAMIC_STATE_DEPTH_COMPARE_OP, VK_DYNAMIC_STATE_DEPTH_BOUNDS_TEST_ENABLE, VK_DYNAMIC_STATE_STENCIL_TEST_ENABLE, VK_DYNAMIC_STATE_STENCIL_OP, or VK_DYNAMIC_STATE_DEPTH_BOUNDS dynamic states are not set, pDepthStencilState **must** be a valid pointer to a valid VkPipelineDepthStencilStateCreateInfo structure

- VUID-VkGraphicsPipelineCreateInfo-pDepthStencilState-09034
  If `pDepthStencilState` is not `NULL` it **must** be a valid pointer to a valid VkPipelineDepthStencilStateCreateInfo structure
- VUID-VkGraphicsPipelineCreateInfo-renderPass-09035
  If `renderPass` is VK_NULL_HANDLE and the pipeline is being created with fragment shader state but not fragment output interface state, and the `VK_EXT_extended_dynamic_state3` extension is not enabled, or any of the `VK_DYNAMIC_STATE_DEPTH_TEST_ENABLE`, `VK_DYNAMIC_STATE_DEPTH_WRITE_ENABLE`, `VK_DYNAMIC_STATE_DEPTH_COMPARE_OP`, `VK_DYNAMIC_STATE_DEPTH_BOUNDS_TEST_ENABLE`, `VK_DYNAMIC_STATE_STENCIL_TEST_ENABLE`, `VK_DYNAMIC_STATE_STENCIL_OP`, or `VK_DYNAMIC_STATE_DEPTH_BOUNDS` dynamic states are not set, `pDepthStencilState` **must** be a valid pointer to a valid VkPipelineDepthStencilStateCreateInfo structure
- VUID-VkGraphicsPipelineCreateInfo-pDepthStencilState-09036
  If `pDepthStencilState` is not `NULL` it **must** be a valid pointer to a valid VkPipelineDepthStencilStateCreateInfo structure
- VUID-VkGraphicsPipelineCreateInfo-renderPass-09037
  If `renderPass` is VK_NULL_HANDLE, the pipeline is being created with fragment output interface state, and any element of VkPipelineRenderingCreateInfo::`pColorAttachmentFormats` is not `VK_FORMAT_UNDEFINED`, and the `VK_EXT_extended_dynamic_state3` extension is not enabled, or any of the `VK_DYNAMIC_STATE_LOGIC_OP_ENABLE_EXT`, `VK_DYNAMIC_STATE_LOGIC_OP_EXT`, `VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT`, `VK_DYNAMIC_STATE_COLOR_BLEND_EQUATION_EXT`, `VK_DYNAMIC_STATE_COLOR_WRITE_MASK_EXT`, or `VK_DYNAMIC_STATE_BLEND_CONSTANTS` dynamic states are not set, `pColorBlendState` **must** be a valid pointer to a valid VkPipelineColorBlendStateCreateInfo structure
- VUID-VkGraphicsPipelineCreateInfo-pColorBlendState-09038
  If `pColorBlendState` is not `NULL` it **must** be a valid pointer to a valid VkPipelineColorBlendStateCreateInfo structure
- VUID-VkGraphicsPipelineCreateInfo-renderPass-06055
  If `renderPass` is VK_NULL_HANDLE, `pColorBlendState` is not dynamic, and the pipeline is being created with fragment output interface state, `pColorBlendState->attachmentCount` **must** be equal to VkPipelineRenderingCreateInfo::`colorAttachmentCount`
- VUID-VkGraphicsPipelineCreateInfo-renderPass-06057
  If `renderPass` is VK_NULL_HANDLE, the pipeline is being created with pre-rasterization shader state, VkPipelineRenderingCreateInfo::`viewMask` is not `0`, and the `multiviewTessellationShader` feature is not enabled, then `pStages` **must** not include tessellation shaders
- VUID-VkGraphicsPipelineCreateInfo-renderPass-06058
  If `renderPass` is VK_NULL_HANDLE, the pipeline is being created with pre-rasterization shader state, VkPipelineRenderingCreateInfo::`viewMask` is not `0`, and the `multiviewGeometryShader` feature is not enabled, then `pStages` **must** not include a geometry shader
- VUID-VkGraphicsPipelineCreateInfo-renderPass-06059
  If `renderPass` is VK_NULL_HANDLE, the pipeline is being created with pre-rasterization shader state, and VkPipelineRenderingCreateInfo::`viewMask` is not `0`, all of the shaders in

the pipeline **must** not include variables decorated with the `Layer` built-in decoration in their interfaces

- VUID-VkGraphicsPipelineCreateInfo-renderPass-07720
  If `renderPass` is VK_NULL_HANDLE, the pipeline is being created with pre-rasterization shader state, and VkPipelineRenderingCreateInfo::`viewMask` is not `0`, and `multiviewMeshShader` is not enabled, then `pStages` **must** not include a mesh shader

- VUID-VkGraphicsPipelineCreateInfo-renderPass-06061
  If the `dynamicRenderingLocalRead` feature is not enabled, the pipeline requires fragment shader state, and `renderPass` is VK_NULL_HANDLE, fragment shaders in `pStages` **must** not include the `InputAttachment` capability

- VUID-VkGraphicsPipelineCreateInfo-renderPass-08710
  If the pipeline requires fragment shader state and `renderPass` is not VK_NULL_HANDLE, fragment shaders in `pStages` **must** not include any of the `TileImageColorReadAccessEXT`, `TileImageDepthReadAccessEXT`, or `TileImageStencilReadAccessEXT` capabilities

- VUID-VkGraphicsPipelineCreateInfo-renderPass-06062
  If the pipeline requires fragment output interface state and `renderPass` is VK_NULL_HANDLE, for each color attachment format defined by the `pColorAttachmentFormats` member of VkPipelineRenderingCreateInfo, if its potential format features do not contain `VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT`, then the `blendEnable` member of the corresponding element of the `pAttachments` member of `pColorBlendState` **must** be `VK_FALSE`

- VUID-VkGraphicsPipelineCreateInfo-renderPass-06063
  If the pipeline requires fragment output interface state and `renderPass` is VK_NULL_HANDLE, if the `pNext` chain includes VkAttachmentSampleCountInfoAMD or `VkAttachmentSampleCountInfoNV`, the `colorAttachmentCount` member of that structure **must** be equal to the value of VkPipelineRenderingCreateInfo::`colorAttachmentCount`

- VUID-VkGraphicsPipelineCreateInfo-flags-06591
  If `pStages` includes a fragment shader stage, and the fragment shader declares the `EarlyFragmentTests` execution mode, the `flags` member of VkPipelineDepthStencilStateCreateInfo **must** not include `VK_PIPELINE_DEPTH_STENCIL_STATE_CREATE_RASTERIZATION_ORDER_ATTACHMENT_DEPTH_ACCESS_BIT_EXT` or `VK_PIPELINE_DEPTH_STENCIL_STATE_CREATE_RASTERIZATION_ORDER_ATTACHMENT_STENCIL_ACCESS_BIT_EXT`

- VUID-VkGraphicsPipelineCreateInfo-flags-06482
  If the `dynamicRenderingLocalRead` feature is not enabled, the pipeline requires fragment output interface state, and the `flags` member of VkPipelineColorBlendStateCreateInfo includes `VK_PIPELINE_COLOR_BLEND_STATE_CREATE_RASTERIZATION_ORDER_ATTACHMENT_ACCESS_BIT_EXT`, `renderPass` **must** not be VK_NULL_HANDLE

- VUID-VkGraphicsPipelineCreateInfo-None-09526
  If the `dynamicRenderingLocalRead` feature is not enabled, the pipeline requires fragment output interface state, and the `flags` member of VkPipelineDepthStencilStateCreateInfo includes `VK_PIPELINE_DEPTH_STENCIL_STATE_CREATE_RASTERIZATION_ORDER_ATTACHMENT_DEPTH_ACCESS_BI`

T_EXT or VK_PIPELINE_DEPTH_STENCIL_STATE_CREATE_RASTERIZATION_ORDER_ATTACHMENT_STENCIL_ACCESS_BIT_EXT, renderPass **must** not be VK_NULL_HANDLE

- VUID-VkGraphicsPipelineCreateInfo-pColorAttachmentSamples-06592
  If the fragment output interface state, elements of the pColorAttachmentSamples member of VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV **must** be valid VkSampleCountFlagBits values
- VUID-VkGraphicsPipelineCreateInfo-depthStencilAttachmentSamples-06593
  If the fragment output interface state and the depthStencilAttachmentSamples member of VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV is not 0, it **must** be a valid VkSampleCountFlagBits value
- VUID-VkGraphicsPipelineCreateInfo-renderPass-09527
  If the pipeline requires fragment output interface state, renderPass is not VK_NULL_HANDLE, and the flags member of VkPipelineColorBlendStateCreateInfo includes VK_PIPELINE_COLOR_BLEND_STATE_CREATE_RASTERIZATION_ORDER_ATTACHMENT_ACCESS_BIT_EXT subpass **must** have been created with VK_SUBPASS_DESCRIPTION_RASTERIZATION_ORDER_ATTACHMENT_COLOR_ACCESS_BIT_EXT
- VUID-VkGraphicsPipelineCreateInfo-renderPass-09528
  If the pipeline requires fragment shader state, renderPass is not VK_NULL_HANDLE, and the flags member of VkPipelineDepthStencilStateCreateInfo includes VK_PIPELINE_DEPTH_STENCIL_STATE_CREATE_RASTERIZATION_ORDER_ATTACHMENT_DEPTH_ACCESS_BIT_EXT, subpass **must** have been created with VK_SUBPASS_DESCRIPTION_RASTERIZATION_ORDER_ATTACHMENT_DEPTH_ACCESS_BIT_EXT
- VUID-VkGraphicsPipelineCreateInfo-renderPass-09529
  If the pipeline requires fragment shader state, renderPass is not VK_NULL_HANDLE, and the flags member of VkPipelineDepthStencilStateCreateInfo includes VK_PIPELINE_DEPTH_STENCIL_STATE_CREATE_RASTERIZATION_ORDER_ATTACHMENT_STENCIL_ACCESS_BIT_EXT, subpass **must** have been created with VK_SUBPASS_DESCRIPTION_RASTERIZATION_ORDER_ATTACHMENT_STENCIL_ACCESS_BIT_EXT
- VUID-VkGraphicsPipelineCreateInfo-pipelineStageCreationFeedbackCount-06594
  If VkPipelineCreationFeedbackCreateInfo::pipelineStageCreationFeedbackCount is not 0, it **must** be equal to stageCount
- VUID-VkGraphicsPipelineCreateInfo-renderPass-06595
  If renderPass is VK_NULL_HANDLE, the pipeline is being created with pre-rasterization shader state or fragment shader state, and VkMultiviewPerViewAttributesInfoNVX::perViewAttributesPositionXOnly is VK_TRUE then VkMultiviewPerViewAttributesInfoNVX::perViewAttributes **must** also be VK_TRUE
- VUID-VkGraphicsPipelineCreateInfo-flags-06596
  If VkGraphicsPipelineLibraryCreateInfoEXT::flags includes only one of VK_GRAPHICS_PIPELINE_LIBRARY_PRE_RASTERIZATION_SHADERS_BIT_EXT or VK_GRAPHICS_PIPELINE_LIBRARY_FRAGMENT_SHADER_BIT_EXT, and an element of VkPipelineLibraryCreateInfoKHR::pLibraries includes the other flag, the value of VkMultiviewPerViewAttributesInfoNVX::perViewAttributes specified in both this pipeline and the library **must** be equal

- VUID-VkGraphicsPipelineCreateInfo-pLibraries-06597
  If one element of VkPipelineLibraryCreateInfoKHR::pLibraries includes VK_GRAPHICS_PIPELINE_LIBRARY_PRE_RASTERIZATION_SHADERS_BIT_EXT and another element includes VK_GRAPHICS_PIPELINE_LIBRARY_FRAGMENT_SHADER_BIT_EXT, the value of VkMultiviewPerViewAttributesInfoNVX::perViewAttributes specified in both libraries **must** be equal
- VUID-VkGraphicsPipelineCreateInfo-flags-06598
  If VkGraphicsPipelineLibraryCreateInfoEXT::flags includes only one of VK_GRAPHICS_PIPELINE_LIBRARY_PRE_RASTERIZATION_SHADERS_BIT_EXT or VK_GRAPHICS_PIPELINE_LIBRARY_FRAGMENT_SHADER_BIT_EXT, and an element of VkPipelineLibraryCreateInfoKHR::pLibraries includes the other flag, the value of VkMultiviewPerViewAttributesInfoNVX::perViewAttributesPositionXOnly specified in both this pipeline and the library **must** be equal
- VUID-VkGraphicsPipelineCreateInfo-pLibraries-06599
  If one element of VkPipelineLibraryCreateInfoKHR::pLibraries includes VK_GRAPHICS_PIPELINE_LIBRARY_PRE_RASTERIZATION_SHADERS_BIT_EXT and another element includes VK_GRAPHICS_PIPELINE_LIBRARY_FRAGMENT_SHADER_BIT_EXT, the value of VkMultiviewPerViewAttributesInfoNVX::perViewAttributesPositionXOnly specified in both libraries **must** be equal
- VUID-VkGraphicsPipelineCreateInfo-pStages-06600
  If the pipeline requires pre-rasterization shader state or fragment shader state, pStages **must** be a valid pointer to an array of stageCount valid VkPipelineShaderStageCreateInfo structures
- VUID-VkGraphicsPipelineCreateInfo-stageCount-09587
  If the pipeline does not require pre-rasterization shader state or fragment shader state, stageCount **must** be zero
- VUID-VkGraphicsPipelineCreateInfo-pRasterizationState-06601
  If the pipeline requires pre-rasterization shader state, and related dynamic state is not set, pRasterizationState **must** be a valid pointer to a valid VkPipelineRasterizationStateCreateInfo structure
- VUID-VkGraphicsPipelineCreateInfo-pRasterizationState-09039
  If VkGraphicsPipelineLibraryCreateInfoEXT::flags includes VK_GRAPHICS_PIPELINE_LIBRARY_PRE_RASTERIZATION_SHADERS_BIT_EXT, and related dynamic state is not set, then pMultisampleState **must** be a valid pointer to a valid VkPipelineMultisampleStateCreateInfo structure
- VUID-VkGraphicsPipelineCreateInfo-pRasterizationState-09040
  If pRasterizationState is not NULL it **must** be a valid pointer to a valid VkPipelineRasterizationStateCreateInfo structure
- VUID-VkGraphicsPipelineCreateInfo-layout-06602
  If the pipeline requires fragment shader state or pre-rasterization shader state, layout **must** be a valid VkPipelineLayout handle
- VUID-VkGraphicsPipelineCreateInfo-renderPass-06603
  If the pipeline requires pre-rasterization shader state, fragment shader state, or fragment output state, and renderPass is not VK_NULL_HANDLE, renderPass **must** be a valid

VkRenderPass handle

- VUID-VkGraphicsPipelineCreateInfo-stageCount-09530
  If the pipeline requires pre-rasterization shader state, `stageCount` **must** be greater than `0`
- VUID-VkGraphicsPipelineCreateInfo-graphicsPipelineLibrary-06606
  If the `graphicsPipelineLibrary` feature is not enabled, `flags` **must** not include `VK_PIPELINE_CREATE_LIBRARY_BIT_KHR`
- VUID-VkGraphicsPipelineCreateInfo-flags-06608
  If the pipeline defines, or includes as libraries, all the state subsets required for a complete graphics pipeline, `flags` **must** not include `VK_PIPELINE_CREATE_LIBRARY_BIT_KHR`
- VUID-VkGraphicsPipelineCreateInfo-flags-06609
  If `flags` includes `VK_PIPELINE_CREATE_LINK_TIME_OPTIMIZATION_BIT_EXT`, pipeline libraries included via VkPipelineLibraryCreateInfoKHR **must** have been created with `VK_PIPELINE_CREATE_RETAIN_LINK_TIME_OPTIMIZATION_INFO_BIT_EXT`
- VUID-VkGraphicsPipelineCreateInfo-flags-09245
  If `flags` includes `VK_PIPELINE_CREATE_RETAIN_LINK_TIME_OPTIMIZATION_INFO_BIT_EXT`, `flags` **must** also include `VK_PIPELINE_CREATE_LIBRARY_BIT_KHR`
- VUID-VkGraphicsPipelineCreateInfo-flags-06610
  If `flags` includes `VK_PIPELINE_CREATE_RETAIN_LINK_TIME_OPTIMIZATION_INFO_BIT_EXT`, pipeline libraries included via VkPipelineLibraryCreateInfoKHR **must** have been created with `VK_PIPELINE_CREATE_RETAIN_LINK_TIME_OPTIMIZATION_INFO_BIT_EXT`
- VUID-VkGraphicsPipelineCreateInfo-pLibraries-06611
  Any pipeline libraries included via VkPipelineLibraryCreateInfoKHR::`pLibraries` **must** not include any state subset already defined by this structure or defined by any other pipeline library in VkPipelineLibraryCreateInfoKHR::`pLibraries`
- VUID-VkGraphicsPipelineCreateInfo-flags-06612
  If VkGraphicsPipelineLibraryCreateInfoEXT::`flags` includes only one of `VK_GRAPHICS_PIPELINE_LIBRARY_PRE_RASTERIZATION_SHADERS_BIT_EXT` or `VK_GRAPHICS_PIPELINE_LIBRARY_FRAGMENT_SHADER_BIT_EXT`, and an element of VkPipelineLibraryCreateInfoKHR::`pLibraries` includes the other flag, and `layout` was not created with `VK_PIPELINE_LAYOUT_CREATE_INDEPENDENT_SETS_BIT_EXT`, then the `layout` used by this pipeline and the library **must** be *identically defined*
- VUID-VkGraphicsPipelineCreateInfo-pLibraries-06613
  If one element of VkPipelineLibraryCreateInfoKHR::`pLibraries` includes `VK_GRAPHICS_PIPELINE_LIBRARY_PRE_RASTERIZATION_SHADERS_BIT_EXT` and another element includes `VK_GRAPHICS_PIPELINE_LIBRARY_FRAGMENT_SHADER_BIT_EXT`, and the `layout` specified by either library was not created with `VK_PIPELINE_LAYOUT_CREATE_INDEPENDENT_SETS_BIT_EXT`, then the `layout` used by each library **must** be *identically defined*
- VUID-VkGraphicsPipelineCreateInfo-flags-06614
  If VkGraphicsPipelineLibraryCreateInfoEXT::`flags` includes only one of `VK_GRAPHICS_PIPELINE_LIBRARY_PRE_RASTERIZATION_SHADERS_BIT_EXT` or `VK_GRAPHICS_PIPELINE_LIBRARY_FRAGMENT_SHADER_BIT_EXT`, an element of VkPipelineLibraryCreateInfoKHR::`pLibraries` includes the other subset, and `layout` was created with `VK_PIPELINE_LAYOUT_CREATE_INDEPENDENT_SETS_BIT_EXT`, then the `layout` used by

the library **must** also have been created with VK_PIPELINE_LAYOUT_CREATE_INDEPENDENT_SETS_BIT_EXT

- VUID-VkGraphicsPipelineCreateInfo-pLibraries-06615
  If one element of VkPipelineLibraryCreateInfoKHR::pLibraries includes VK_GRAPHICS_PIPELINE_LIBRARY_PRE_RASTERIZATION_SHADERS_BIT_EXT and another element includes VK_GRAPHICS_PIPELINE_LIBRARY_FRAGMENT_SHADER_BIT_EXT, and the layout specified by either library was created with VK_PIPELINE_LAYOUT_CREATE_INDEPENDENT_SETS_BIT_EXT, then the layout used by both libraries **must** have been created with VK_PIPELINE_LAYOUT_CREATE_INDEPENDENT_SETS_BIT_EXT
- VUID-VkGraphicsPipelineCreateInfo-flags-06616
  If VkGraphicsPipelineLibraryCreateInfoEXT::flags includes only one of VK_GRAPHICS_PIPELINE_LIBRARY_PRE_RASTERIZATION_SHADERS_BIT_EXT or VK_GRAPHICS_PIPELINE_LIBRARY_FRAGMENT_SHADER_BIT_EXT, an element of VkPipelineLibraryCreateInfoKHR::pLibraries includes the other subset, and layout was created with VK_PIPELINE_LAYOUT_CREATE_INDEPENDENT_SETS_BIT_EXT, elements of the pSetLayouts array which layout was created with that are not VK_NULL_HANDLE **must** be *identically defined* to the element at the same index of pSetLayouts used to create the library's layout
- VUID-VkGraphicsPipelineCreateInfo-pLibraries-06617
  If one element of VkPipelineLibraryCreateInfoKHR::pLibraries includes VK_GRAPHICS_PIPELINE_LIBRARY_PRE_RASTERIZATION_SHADERS_BIT_EXT and another element includes VK_GRAPHICS_PIPELINE_LIBRARY_FRAGMENT_SHADER_BIT_EXT, and the layout specified by either library was created with VK_PIPELINE_LAYOUT_CREATE_INDEPENDENT_SETS_BIT_EXT, elements of the pSetLayouts array which either layout was created with that are not VK_NULL_HANDLE **must** be *identically defined* to the element at the same index of pSetLayouts used to create the other library's layout
- VUID-VkGraphicsPipelineCreateInfo-flags-06618
  If VkGraphicsPipelineLibraryCreateInfoEXT::flags includes only one of VK_GRAPHICS_PIPELINE_LIBRARY_PRE_RASTERIZATION_SHADERS_BIT_EXT or VK_GRAPHICS_PIPELINE_LIBRARY_FRAGMENT_SHADER_BIT_EXT, and an element of VkPipelineLibraryCreateInfoKHR::pLibraries includes the other flag, any descriptor set layout *N* specified by layout in both this pipeline and the library which include bindings accessed by shader stages in each **must** be *identically defined*
- VUID-VkGraphicsPipelineCreateInfo-pLibraries-06619
  If one element of VkPipelineLibraryCreateInfoKHR::pLibraries includes VK_GRAPHICS_PIPELINE_LIBRARY_PRE_RASTERIZATION_SHADERS_BIT_EXT and another element includes VK_GRAPHICS_PIPELINE_LIBRARY_FRAGMENT_SHADER_BIT_EXT, any descriptor set layout *N* specified by layout in both libraries which include bindings accessed by shader stages in each **must** be *identically defined*
- VUID-VkGraphicsPipelineCreateInfo-flags-06620
  If VkGraphicsPipelineLibraryCreateInfoEXT::flags includes only one of VK_GRAPHICS_PIPELINE_LIBRARY_PRE_RASTERIZATION_SHADERS_BIT_EXT or VK_GRAPHICS_PIPELINE_LIBRARY_FRAGMENT_SHADER_BIT_EXT, and an element of VkPipelineLibraryCreateInfoKHR::pLibraries includes the other flag, push constants specified in layout in both this pipeline and the library which are available to shader

stages in each **must** be *identically defined*

- VUID-VkGraphicsPipelineCreateInfo-pLibraries-06621
  If one element of VkPipelineLibraryCreateInfoKHR::`pLibraries` includes `VK_GRAPHICS_PIPELINE_LIBRARY_PRE_RASTERIZATION_SHADERS_BIT_EXT` and another element includes `VK_GRAPHICS_PIPELINE_LIBRARY_FRAGMENT_SHADER_BIT_EXT`, push constants specified in `layout` in both this pipeline and the library which are available to shader stages in each **must** be *identically defined*

- VUID-VkGraphicsPipelineCreateInfo-flags-06679
  If VkGraphicsPipelineLibraryCreateInfoEXT::`flags` includes only one of `VK_GRAPHICS_PIPELINE_LIBRARY_PRE_RASTERIZATION_SHADERS_BIT_EXT` or `VK_GRAPHICS_PIPELINE_LIBRARY_FRAGMENT_SHADER_BIT_EXT`, an element of VkPipelineLibraryCreateInfoKHR::`pLibraries` includes the other subset, any element of the `pSetLayouts` array when `layout` was created and the corresponding element of the `pSetLayouts` array used to create the library's `layout` **must** not both be VK_NULL_HANDLE

- VUID-VkGraphicsPipelineCreateInfo-pLibraries-06681
  If one element of VkPipelineLibraryCreateInfoKHR::`pLibraries` includes `VK_GRAPHICS_PIPELINE_LIBRARY_PRE_RASTERIZATION_SHADERS_BIT_EXT` and another element includes `VK_GRAPHICS_PIPELINE_LIBRARY_FRAGMENT_SHADER_BIT_EXT`, and any element of the `pSetLayouts` array used to create each library's `layout` was VK_NULL_HANDLE, then the corresponding element of the `pSetLayouts` array used to create the other library's `layout` **must** not be VK_NULL_HANDLE

- VUID-VkGraphicsPipelineCreateInfo-flags-06756
  If VkGraphicsPipelineLibraryCreateInfoEXT::`flags` includes only one of `VK_GRAPHICS_PIPELINE_LIBRARY_PRE_RASTERIZATION_SHADERS_BIT_EXT` or `VK_GRAPHICS_PIPELINE_LIBRARY_FRAGMENT_SHADER_BIT_EXT`, an element of VkPipelineLibraryCreateInfoKHR::`pLibraries` includes the other subset, and any element of the `pSetLayouts` array which `layout` was created with was VK_NULL_HANDLE, then the corresponding element of the `pSetLayouts` array used to create the library's `layout` **must** not have shader bindings for shaders in the other subset

- VUID-VkGraphicsPipelineCreateInfo-flags-06757
  If VkGraphicsPipelineLibraryCreateInfoEXT::`flags` includes only one of `VK_GRAPHICS_PIPELINE_LIBRARY_PRE_RASTERIZATION_SHADERS_BIT_EXT` or `VK_GRAPHICS_PIPELINE_LIBRARY_FRAGMENT_SHADER_BIT_EXT`, an element of VkPipelineLibraryCreateInfoKHR::`pLibraries` includes the other subset, and any element of the `pSetLayouts` array used to create the library's `layout` was VK_NULL_HANDLE, then the corresponding element of the `pSetLayouts` array used to create this pipeline's `layout` **must** not have shader bindings for shaders in the other subset

- VUID-VkGraphicsPipelineCreateInfo-pLibraries-06758
  If one element of VkPipelineLibraryCreateInfoKHR::`pLibraries` includes `VK_GRAPHICS_PIPELINE_LIBRARY_PRE_RASTERIZATION_SHADERS_BIT_EXT` and another element includes `VK_GRAPHICS_PIPELINE_LIBRARY_FRAGMENT_SHADER_BIT_EXT`, and any element of the `pSetLayouts` array used to create each library's `layout` was VK_NULL_HANDLE, then the corresponding element of the `pSetLayouts` array used to create the other library's `layout` **must** not have shader bindings for shaders in the other subset

- VUID-VkGraphicsPipelineCreateInfo-flags-06682

If VkGraphicsPipelineLibraryCreateInfoEXT::flags includes both VK_GRAPHICS_PIPELINE_LIBRARY_PRE_RASTERIZATION_SHADERS_BIT_EXT and VK_GRAPHICS_PIPELINE_LIBRARY_FRAGMENT_SHADER_BIT_EXT, layout **must** have been created with no elements of the pSetLayouts array set to VK_NULL_HANDLE

- VUID-VkGraphicsPipelineCreateInfo-flags-06683
  If VkGraphicsPipelineLibraryCreateInfoEXT::flags includes VK_GRAPHICS_PIPELINE_LIBRARY_PRE_RASTERIZATION_SHADERS_BIT_EXT and pRasterizationState->rasterizerDiscardEnable is VK_TRUE, layout **must** have been created with no elements of the pSetLayouts array set to VK_NULL_HANDLE

- VUID-VkGraphicsPipelineCreateInfo-flags-06684
  If VkGraphicsPipelineLibraryCreateInfoEXT::flags includes at least one of and no more than two of VK_GRAPHICS_PIPELINE_LIBRARY_PRE_RASTERIZATION_SHADERS_BIT_EXT, VK_GRAPHICS_PIPELINE_LIBRARY_FRAGMENT_SHADER_BIT_EXT, or VK_GRAPHICS_PIPELINE_LIBRARY_FRAGMENT_OUTPUT_INTERFACE_BIT_EXT, and an element of VkPipelineLibraryCreateInfoKHR::pLibraries includes one of the other flags, the value of subpass **must** be equal to that used to create the library

- VUID-VkGraphicsPipelineCreateInfo-pLibraries-06623
  If one element of VkPipelineLibraryCreateInfoKHR::pLibraries includes at least one of and no more than two of VK_GRAPHICS_PIPELINE_LIBRARY_PRE_RASTERIZATION_SHADERS_BIT_EXT, VK_GRAPHICS_PIPELINE_LIBRARY_FRAGMENT_SHADER_BIT_EXT, or VK_GRAPHICS_PIPELINE_LIBRARY_FRAGMENT_OUTPUT_INTERFACE_BIT_EXT, and another element of VkPipelineLibraryCreateInfoKHR::pLibraries includes one of the other flags, the value of subpass used to create each library **must** be identical

- VUID-VkGraphicsPipelineCreateInfo-renderpass-06624
  If renderpass is not VK_NULL_HANDLE, VkGraphicsPipelineLibraryCreateInfoEXT::flags includes at least one of and no more than two of VK_GRAPHICS_PIPELINE_LIBRARY_PRE_RASTERIZATION_SHADERS_BIT_EXT, VK_GRAPHICS_PIPELINE_LIBRARY_FRAGMENT_SHADER_BIT_EXT, or VK_GRAPHICS_PIPELINE_LIBRARY_FRAGMENT_OUTPUT_INTERFACE_BIT_EXT, and an element of VkPipelineLibraryCreateInfoKHR::pLibraries includes one of the other flags, renderPass **must** be compatible with that used to create the library

- VUID-VkGraphicsPipelineCreateInfo-renderpass-06625
  If renderpass is VK_NULL_HANDLE, VkGraphicsPipelineLibraryCreateInfoEXT::flags includes at least one of and no more than two of VK_GRAPHICS_PIPELINE_LIBRARY_PRE_RASTERIZATION_SHADERS_BIT_EXT, VK_GRAPHICS_PIPELINE_LIBRARY_FRAGMENT_SHADER_BIT_EXT, or VK_GRAPHICS_PIPELINE_LIBRARY_FRAGMENT_OUTPUT_INTERFACE_BIT_EXT, and an element of VkPipelineLibraryCreateInfoKHR::pLibraries includes one of the other flags, the value of renderPass used to create that library **must** also be VK_NULL_HANDLE

- VUID-VkGraphicsPipelineCreateInfo-flags-06626
  If VkGraphicsPipelineLibraryCreateInfoEXT::flags includes at least one of and no more than two of VK_GRAPHICS_PIPELINE_LIBRARY_PRE_RASTERIZATION_SHADERS_BIT_EXT, VK_GRAPHICS_PIPELINE_LIBRARY_FRAGMENT_SHADER_BIT_EXT, or VK_GRAPHICS_PIPELINE_LIBRARY_FRAGMENT_OUTPUT_INTERFACE_BIT_EXT, an element of

VkPipelineLibraryCreateInfoKHR::pLibraries includes one of the other flags, and renderPass is VK_NULL_HANDLE, the value of VkPipelineRenderingCreateInfo::viewMask used by this pipeline and that specified by the library **must** be identical

- VUID-VkGraphicsPipelineCreateInfo-pLibraries-06627
  If one element of VkPipelineLibraryCreateInfoKHR::pLibraries includes at least one of and no more than two of VK_GRAPHICS_PIPELINE_LIBRARY_PRE_RASTERIZATION_SHADERS_BIT_EXT, VK_GRAPHICS_PIPELINE_LIBRARY_FRAGMENT_SHADER_BIT_EXT, or VK_GRAPHICS_PIPELINE_LIBRARY_FRAGMENT_OUTPUT_INTERFACE_BIT_EXT, another element of VkPipelineLibraryCreateInfoKHR::pLibraries includes one of the other flags, and renderPass was VK_NULL_HANDLE for both libraries, the value of VkPipelineRenderingCreateInfo::viewMask set by each library **must** be identical

- VUID-VkGraphicsPipelineCreateInfo-pLibraries-06628
  If one element of VkPipelineLibraryCreateInfoKHR::pLibraries includes at least one of and no more than two of VK_GRAPHICS_PIPELINE_LIBRARY_PRE_RASTERIZATION_SHADERS_BIT_EXT, VK_GRAPHICS_PIPELINE_LIBRARY_FRAGMENT_SHADER_BIT_EXT, or VK_GRAPHICS_PIPELINE_LIBRARY_FRAGMENT_OUTPUT_INTERFACE_BIT_EXT, and another element of VkPipelineLibraryCreateInfoKHR::pLibraries includes one of the other flags, the renderPass objects used to create each library **must** be compatible or all equal to VK_NULL_HANDLE

- VUID-VkGraphicsPipelineCreateInfo-renderpass-06631
  If renderPass is not VK_NULL_HANDLE, the pipeline requires fragment shader state, and the VK_EXT_extended_dynamic_state3 extension is not enabled or any of the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT, VK_DYNAMIC_STATE_SAMPLE_MASK_EXT, or VK_DYNAMIC_STATE_ALPHA_TO_COVERAGE_ENABLE_EXT dynamic states is not set, or alphaToOne is enabled on the device and VK_DYNAMIC_STATE_ALPHA_TO_ONE_ENABLE_EXT is not set, then pMultisampleState **must** be a valid pointer to a valid VkPipelineMultisampleStateCreateInfo structure

- VUID-VkGraphicsPipelineCreateInfo-Input-06632
  If the pipeline requires fragment shader state with a fragment shader that either enables sample shading or decorates any variable in the Input storage class with Sample, and the VK_EXT_extended_dynamic_state3 extension is not enabled or any of the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT, VK_DYNAMIC_STATE_SAMPLE_MASK_EXT, or VK_DYNAMIC_STATE_ALPHA_TO_COVERAGE_ENABLE_EXT dynamic states is not set, or alphaToOne is enabled on the device and VK_DYNAMIC_STATE_ALPHA_TO_ONE_ENABLE_EXT is not set, then pMultisampleState **must** be a valid pointer to a valid VkPipelineMultisampleStateCreateInfo structure

- VUID-VkGraphicsPipelineCreateInfo-flags-06633
  If VkGraphicsPipelineLibraryCreateInfoEXT::flags includes VK_GRAPHICS_PIPELINE_LIBRARY_FRAGMENT_SHADER_BIT_EXT with a pMultisampleState that was not NULL, and an element of VkPipelineLibraryCreateInfoKHR::pLibraries was created with VK_GRAPHICS_PIPELINE_LIBRARY_FRAGMENT_OUTPUT_INTERFACE_BIT_EXT, pMultisampleState **must** be *identically defined* to that used to create the library

- VUID-VkGraphicsPipelineCreateInfo-pLibraries-06634

If an element of VkPipelineLibraryCreateInfoKHR::pLibraries was created with VK_GRAPHICS_PIPELINE_LIBRARY_FRAGMENT_SHADER_BIT_EXT with a pMultisampleState that was not NULL, and if VkGraphicsPipelineLibraryCreateInfoEXT::flags includes VK_GRAPHICS_PIPELINE_LIBRARY_FRAGMENT_OUTPUT_INTERFACE_BIT_EXT, pMultisampleState **must** be *identically defined* to that used to create the library

- VUID-VkGraphicsPipelineCreateInfo-pLibraries-06635
  If one element of VkPipelineLibraryCreateInfoKHR::pLibraries was created with VK_GRAPHICS_PIPELINE_LIBRARY_FRAGMENT_SHADER_BIT_EXT with a pMultisampleState that was not NULL, and if a different element of VkPipelineLibraryCreateInfoKHR::pLibraries was created with VK_GRAPHICS_PIPELINE_LIBRARY_FRAGMENT_OUTPUT_INTERFACE_BIT_EXT, the pMultisampleState used to create each library **must** be *identically defined*
- VUID-VkGraphicsPipelineCreateInfo-pLibraries-06636
  If one element of VkPipelineLibraryCreateInfoKHR::pLibraries was created with VK_GRAPHICS_PIPELINE_LIBRARY_FRAGMENT_OUTPUT_INTERFACE_BIT_EXT and a value of pMultisampleState->sampleShadingEnable equal VK_TRUE, and if a different element of VkPipelineLibraryCreateInfoKHR::pLibraries was created with VK_GRAPHICS_PIPELINE_LIBRARY_FRAGMENT_SHADER_BIT_EXT, the pMultisampleState used to create each library **must** be *identically defined*
- VUID-VkGraphicsPipelineCreateInfo-flags-06637
  If VkGraphicsPipelineLibraryCreateInfoEXT::flags includes VK_GRAPHICS_PIPELINE_LIBRARY_FRAGMENT_OUTPUT_INTERFACE_BIT_EXT, pMultisampleState->sampleShadingEnable is VK_TRUE, and an element of VkPipelineLibraryCreateInfoKHR::pLibraries was created with VK_GRAPHICS_PIPELINE_LIBRARY_FRAGMENT_SHADER_BIT_EXT, pMultisampleState **must** be *identically defined* to that used to create the library
- VUID-VkGraphicsPipelineCreateInfo-pLibraries-09567
  If one element of VkPipelineLibraryCreateInfoKHR::pLibraries was created with VK_GRAPHICS_PIPELINE_LIBRARY_FRAGMENT_OUTPUT_INTERFACE_BIT_EXT and a value of pMultisampleState->sampleShadingEnable equal VK_TRUE, and if VkGraphicsPipelineLibraryCreateInfoEXT::flags includes VK_GRAPHICS_PIPELINE_LIBRARY_FRAGMENT_SHADER_BIT_EXT, pMultisampleState **must** be *identically defined* to that used to create the library
- VUID-VkGraphicsPipelineCreateInfo-flags-06638
  If VkGraphicsPipelineLibraryCreateInfoEXT::flags includes only one of VK_GRAPHICS_PIPELINE_LIBRARY_PRE_RASTERIZATION_SHADERS_BIT_EXT or VK_GRAPHICS_PIPELINE_LIBRARY_FRAGMENT_SHADER_BIT_EXT, and an element of VkPipelineLibraryCreateInfoKHR::pLibraries includes the other flag, values specified in VkPipelineFragmentShadingRateStateCreateInfoKHR for both this pipeline and that library **must** be identical
- VUID-VkGraphicsPipelineCreateInfo-pLibraries-06639
  If one element of VkPipelineLibraryCreateInfoKHR::pLibraries includes VK_GRAPHICS_PIPELINE_LIBRARY_PRE_RASTERIZATION_SHADERS_BIT_EXT and another element includes VK_GRAPHICS_PIPELINE_LIBRARY_FRAGMENT_SHADER_BIT_EXT, values specified in VkPipelineFragmentShadingRateStateCreateInfoKHR for both this pipeline and that library **must** be identical
- VUID-VkGraphicsPipelineCreateInfo-flags-06640

If VkGraphicsPipelineLibraryCreateInfoEXT::flags includes VK_GRAPHICS_PIPELINE_LIBRARY_PRE_RASTERIZATION_SHADERS_BIT_EXT or VK_GRAPHICS_PIPELINE_LIBRARY_FRAGMENT_SHADER_BIT_EXT, pStages **must** be a valid pointer to an array of stageCount valid VkPipelineShaderStageCreateInfo structures

- VUID-VkGraphicsPipelineCreateInfo-flags-06642
  If VkGraphicsPipelineLibraryCreateInfoEXT::flags includes VK_GRAPHICS_PIPELINE_LIBRARY_PRE_RASTERIZATION_SHADERS_BIT_EXT or VK_GRAPHICS_PIPELINE_LIBRARY_FRAGMENT_SHADER_BIT_EXT, layout **must** be a valid VkPipelineLayout handle
- VUID-VkGraphicsPipelineCreateInfo-flags-06643
  If VkGraphicsPipelineLibraryCreateInfoEXT::flags includes VK_GRAPHICS_PIPELINE_LIBRARY_PRE_RASTERIZATION_SHADERS_BIT_EXT, or VK_GRAPHICS_PIPELINE_LIBRARY_FRAGMENT_SHADER_BIT_EXT, VK_GRAPHICS_PIPELINE_LIBRARY_FRAGMENT_OUTPUT_INTERFACE_BIT_EXT, and renderPass is not VK_NULL_HANDLE, renderPass **must** be a valid VkRenderPass handle
- VUID-VkGraphicsPipelineCreateInfo-flags-06644
  If VkGraphicsPipelineLibraryCreateInfoEXT::flags includes VK_GRAPHICS_PIPELINE_LIBRARY_PRE_RASTERIZATION_SHADERS_BIT_EXT or VK_GRAPHICS_PIPELINE_LIBRARY_FRAGMENT_SHADER_BIT_EXT, stageCount **must** be greater than 0
- VUID-VkGraphicsPipelineCreateInfo-flags-06645
  If VkGraphicsPipelineLibraryCreateInfoEXT::flags is non-zero, if flags includes VK_PIPELINE_CREATE_CAPTURE_INTERNAL_REPRESENTATIONS_BIT_KHR, any libraries **must** have also been created with VK_PIPELINE_CREATE_CAPTURE_INTERNAL_REPRESENTATIONS_BIT_KHR
- VUID-VkGraphicsPipelineCreateInfo-pLibraries-06646
  If VkPipelineLibraryCreateInfoKHR::pLibraries includes more than one library, and any library was created with VK_PIPELINE_CREATE_CAPTURE_INTERNAL_REPRESENTATIONS_BIT_KHR, all libraries **must** have also been created with VK_PIPELINE_CREATE_CAPTURE_INTERNAL_REPRESENTATIONS_BIT_KHR
- VUID-VkGraphicsPipelineCreateInfo-pLibraries-06647
  If VkPipelineLibraryCreateInfoKHR::pLibraries includes at least one library, VkGraphicsPipelineLibraryCreateInfoEXT::flags is non-zero, and any library was created with VK_PIPELINE_CREATE_CAPTURE_INTERNAL_REPRESENTATIONS_BIT_KHR, flags **must** include VK_PIPELINE_CREATE_CAPTURE_INTERNAL_REPRESENTATIONS_BIT_KHR
- VUID-VkGraphicsPipelineCreateInfo-None-07826
  If the pipeline includes a complete set of state, and there are no libraries included in VkPipelineLibraryCreateInfoKHR::pLibraries, then VkPipelineLayout **must** be a valid pipeline layout
- VUID-VkGraphicsPipelineCreateInfo-layout-07827
  If the pipeline includes a complete set of state specified entirely by libraries, and each library was created with a VkPipelineLayout created without VK_PIPELINE_LAYOUT_CREATE_INDEPENDENT_SETS_BIT_EXT, then layout **must** be compatible with the layouts in those libraries
- VUID-VkGraphicsPipelineCreateInfo-flags-06729
  If flags includes VK_PIPELINE_CREATE_LINK_TIME_OPTIMIZATION_BIT_EXT, the pipeline

includes a complete set of state specified entirely by libraries, and each library was created with a VkPipelineLayout created with VK_PIPELINE_LAYOUT_CREATE_INDEPENDENT_SETS_BIT_EXT, then layout **must** be compatible with the union of the libraries' pipeline layouts other than the inclusion/exclusion of VK_PIPELINE_LAYOUT_CREATE_INDEPENDENT_SETS_BIT_EXT

- VUID-VkGraphicsPipelineCreateInfo-flags-06730
  If flags does not include VK_PIPELINE_CREATE_LINK_TIME_OPTIMIZATION_BIT_EXT, the pipeline includes a complete set of state specified entirely by libraries, and each library was created with a VkPipelineLayout created with VK_PIPELINE_LAYOUT_CREATE_INDEPENDENT_SETS_BIT_EXT, then layout **must** be compatible with the union of the libraries' pipeline layouts
- VUID-VkGraphicsPipelineCreateInfo-conservativePointAndLineRasterization-08892
  If conservativePointAndLineRasterization is not supported; the pipeline is being created with vertex input state and pre-rasterization shader state; the pipeline does not include a geometry shader; and the value of VkPipelineInputAssemblyStateCreateInfo::topology is VK_PRIMITIVE_TOPOLOGY_POINT_LIST, VK_PRIMITIVE_TOPOLOGY_LINE_LIST, or VK_PRIMITIVE_TOPOLOGY_LINE_STRIP, and either VK_DYNAMIC_STATE_PRIMITIVE_TOPOLOGY dynamic state is not enabled or dynamicPrimitiveTopologyUnrestricted is VK_FALSE, then VkPipelineRasterizationConservativeStateCreateInfoEXT::conservativeRasterizationMode **must** be VK_CONSERVATIVE_RASTERIZATION_MODE_DISABLED_EXT
- VUID-VkGraphicsPipelineCreateInfo-conservativePointAndLineRasterization-06760
  If conservativePointAndLineRasterization is not supported, the pipeline requires pre-rasterization shader state, and the pipeline includes a geometry shader with either the OutputPoints or OutputLineStrip execution modes, VkPipelineRasterizationConservativeStateCreateInfoEXT::conservativeRasterizationMode **must** be VK_CONSERVATIVE_RASTERIZATION_MODE_DISABLED_EXT
- VUID-VkGraphicsPipelineCreateInfo-conservativePointAndLineRasterization-06761
  If conservativePointAndLineRasterization is not supported, the pipeline requires pre-rasterization shader state, and the pipeline includes a mesh shader with either the OutputPoints or OutputLinesNV execution modes, VkPipelineRasterizationConservativeStateCreateInfoEXT::conservativeRasterizationMode **must** be VK_CONSERVATIVE_RASTERIZATION_MODE_DISABLED_EXT
- VUID-VkGraphicsPipelineCreateInfo-pStages-06894
  If the pipeline requires pre-rasterization shader state but not fragment shader state, elements of pStages **must** not have stage set to VK_SHADER_STAGE_FRAGMENT_BIT
- VUID-VkGraphicsPipelineCreateInfo-pStages-06895
  If the pipeline requires fragment shader state but not pre-rasterization shader state, elements of pStages **must** not have stage set to a shader stage which participates in pre-rasterization
- VUID-VkGraphicsPipelineCreateInfo-pStages-06896
  If the pipeline requires pre-rasterization shader state, all elements of pStages **must** have a stage set to a shader stage which participates in fragment shader state or pre-rasterization shader state
- VUID-VkGraphicsPipelineCreateInfo-stage-06897
  If the pipeline requires fragment shader state and/or pre-rasterization shader state, any

value of stage **must** not be set in more than one element of pStages

- VUID-VkGraphicsPipelineCreateInfo-extendedDynamicState3TessellationDomainOrigin-07370
  If the extendedDynamicState3TessellationDomainOrigin feature is not enabled, there **must** be no element of the pDynamicStates member of pDynamicState set to VK_DYNAMIC_STATE_TESSELLATION_DOMAIN_ORIGIN_EXT
- VUID-VkGraphicsPipelineCreateInfo-extendedDynamicState3DepthClampEnable-07371
  If the extendedDynamicState3DepthClampEnable feature is not enabled, there **must** be no element of the pDynamicStates member of pDynamicState set to VK_DYNAMIC_STATE_DEPTH_CLAMP_ENABLE_EXT
- VUID-VkGraphicsPipelineCreateInfo-extendedDynamicState3PolygonMode-07372
  If the extendedDynamicState3PolygonMode feature is not enabled, there **must** be no element of the pDynamicStates member of pDynamicState set to VK_DYNAMIC_STATE_POLYGON_MODE_EXT
- VUID-VkGraphicsPipelineCreateInfo-extendedDynamicState3RasterizationSamples-07373
  If the extendedDynamicState3RasterizationSamples feature is not enabled, there **must** be no element of the pDynamicStates member of pDynamicState set to VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT
- VUID-VkGraphicsPipelineCreateInfo-extendedDynamicState3SampleMask-07374
  If the extendedDynamicState3SampleMask feature is not enabled, there **must** be no element of the pDynamicStates member of pDynamicState set to VK_DYNAMIC_STATE_SAMPLE_MASK_EXT
- VUID-VkGraphicsPipelineCreateInfo-extendedDynamicState3AlphaToCoverageEnable-07375
  If the extendedDynamicState3AlphaToCoverageEnable feature is not enabled, there **must** be no element of the pDynamicStates member of pDynamicState set to VK_DYNAMIC_STATE_ALPHA_TO_COVERAGE_ENABLE_EXT
- VUID-VkGraphicsPipelineCreateInfo-extendedDynamicState3AlphaToOneEnable-07376
  If the extendedDynamicState3AlphaToOneEnable feature is not enabled, there **must** be no element of the pDynamicStates member of pDynamicState set to VK_DYNAMIC_STATE_ALPHA_TO_ONE_ENABLE_EXT
- VUID-VkGraphicsPipelineCreateInfo-extendedDynamicState3LogicOpEnable-07377
  If the extendedDynamicState3LogicOpEnable feature is not enabled, there **must** be no element of the pDynamicStates member of pDynamicState set to VK_DYNAMIC_STATE_LOGIC_OP_ENABLE_EXT
- VUID-VkGraphicsPipelineCreateInfo-extendedDynamicState3ColorBlendEnable-07378
  If the extendedDynamicState3ColorBlendEnable feature is not enabled, there **must** be no element of the pDynamicStates member of pDynamicState set to VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT
- VUID-VkGraphicsPipelineCreateInfo-extendedDynamicState3ColorBlendEquation-07379
  If the extendedDynamicState3ColorBlendEquation feature is not enabled, there **must** be no element of the pDynamicStates member of pDynamicState set to VK_DYNAMIC_STATE_COLOR_BLEND_EQUATION_EXT
- VUID-VkGraphicsPipelineCreateInfo-extendedDynamicState3ColorWriteMask-07380
  If the extendedDynamicState3ColorWriteMask feature is not enabled, there **must** be no

element of the pDynamicStates member of pDynamicState set to VK_DYNAMIC_STATE_COLOR_WRITE_MASK_EXT

- VUID-VkGraphicsPipelineCreateInfo-extendedDynamicState3RasterizationStream-07381
  If the extendedDynamicState3RasterizationStream feature is not enabled, there **must** be no element of the pDynamicStates member of pDynamicState set to VK_DYNAMIC_STATE_RASTERIZATION_STREAM_EXT
- VUID-VkGraphicsPipelineCreateInfo-extendedDynamicState3ConservativeRasterizationMode-07382
  If the extendedDynamicState3ConservativeRasterizationMode feature is not enabled, there **must** be no element of the pDynamicStates member of pDynamicState set to VK_DYNAMIC_STATE_CONSERVATIVE_RASTERIZATION_MODE_EXT
- VUID-VkGraphicsPipelineCreateInfo-extendedDynamicState3ExtraPrimitiveOverestimationSize-07383
  If the extendedDynamicState3ExtraPrimitiveOverestimationSize feature is not enabled, there **must** be no element of the pDynamicStates member of pDynamicState set to VK_DYNAMIC_STATE_EXTRA_PRIMITIVE_OVERESTIMATION_SIZE_EXT
- VUID-VkGraphicsPipelineCreateInfo-pDynamicState-09639
  If the pipeline requires pre-rasterization shader state, pDynamicState includes VK_DYNAMIC_STATE_CONSERVATIVE_RASTERIZATION_MODE_EXT, and pDynamicState does not include VK_DYNAMIC_STATE_EXTRA_PRIMITIVE_OVERESTIMATION_SIZE_EXT, pRasterizationState **must** include a VkPipelineRasterizationConservativeStateCreateInfoEXT in its pNext chain
- VUID-VkGraphicsPipelineCreateInfo-extendedDynamicState3DepthClipEnable-07384
  If the extendedDynamicState3DepthClipEnable feature is not enabled, there **must** be no element of the pDynamicStates member of pDynamicState set to VK_DYNAMIC_STATE_DEPTH_CLIP_ENABLE_EXT
- VUID-VkGraphicsPipelineCreateInfo-extendedDynamicState3SampleLocationsEnable-07385
  If the extendedDynamicState3SampleLocationsEnable feature is not enabled, there **must** be no element of the pDynamicStates member of pDynamicState set to VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT
- VUID-VkGraphicsPipelineCreateInfo-extendedDynamicState3ColorBlendAdvanced-07386
  If the extendedDynamicState3ColorBlendAdvanced feature is not enabled, there **must** be no element of the pDynamicStates member of pDynamicState set to VK_DYNAMIC_STATE_COLOR_BLEND_ADVANCED_EXT
- VUID-VkGraphicsPipelineCreateInfo-extendedDynamicState3ProvokingVertexMode-07387
  If the extendedDynamicState3ProvokingVertexMode feature is not enabled, there **must** be no element of the pDynamicStates member of pDynamicState set to VK_DYNAMIC_STATE_PROVOKING_VERTEX_MODE_EXT
- VUID-VkGraphicsPipelineCreateInfo-extendedDynamicState3LineRasterizationMode-07388
  If the extendedDynamicState3LineRasterizationMode feature is not enabled, there **must** be no element of the pDynamicStates member of pDynamicState set to VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT
- VUID-VkGraphicsPipelineCreateInfo-extendedDynamicState3LineStippleEnable-07389

If the extendedDynamicState3LineStippleEnable feature is not enabled, there **must** be no element of the pDynamicStates member of pDynamicState set to VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT

- VUID-VkGraphicsPipelineCreateInfo-extendedDynamicState3DepthClipNegativeOneToOne-07390
  If the extendedDynamicState3DepthClipNegativeOneToOne feature is not enabled, there **must** be no element of the pDynamicStates member of pDynamicState set to VK_DYNAMIC_STATE_DEPTH_CLIP_NEGATIVE_ONE_TO_ONE_EXT
- VUID-VkGraphicsPipelineCreateInfo-extendedDynamicState3ViewportWScalingEnable-07391
  If the extendedDynamicState3ViewportWScalingEnable feature is not enabled, there **must** be no element of the pDynamicStates member of pDynamicState set to VK_DYNAMIC_STATE_VIEWPORT_W_SCALING_ENABLE_NV
- VUID-VkGraphicsPipelineCreateInfo-extendedDynamicState3ViewportSwizzle-07392
  If the extendedDynamicState3ViewportSwizzle feature is not enabled, there **must** be no element of the pDynamicStates member of pDynamicState set to VK_DYNAMIC_STATE_VIEWPORT_SWIZZLE_NV
- VUID-VkGraphicsPipelineCreateInfo-extendedDynamicState3CoverageToColorEnable-07393
  If the extendedDynamicState3CoverageToColorEnable feature is not enabled, there **must** be no element of the pDynamicStates member of pDynamicState set to VK_DYNAMIC_STATE_COVERAGE_TO_COLOR_ENABLE_NV
- VUID-VkGraphicsPipelineCreateInfo-extendedDynamicState3CoverageToColorLocation-07394
  If the extendedDynamicState3CoverageToColorLocation feature is not enabled, there **must** be no element of the pDynamicStates member of pDynamicState set to VK_DYNAMIC_STATE_COVERAGE_TO_COLOR_LOCATION_NV
- VUID-VkGraphicsPipelineCreateInfo-extendedDynamicState3CoverageModulationMode-07395
  If the extendedDynamicState3CoverageModulationMode feature is not enabled, there **must** be no element of the pDynamicStates member of pDynamicState set to VK_DYNAMIC_STATE_COVERAGE_MODULATION_MODE_NV
- VUID-VkGraphicsPipelineCreateInfo-extendedDynamicState3CoverageModulationTableEnable-07396
  If the extendedDynamicState3CoverageModulationTableEnable feature is not enabled, there **must** be no element of the pDynamicStates member of pDynamicState set to VK_DYNAMIC_STATE_COVERAGE_MODULATION_TABLE_ENABLE_NV
- VUID-VkGraphicsPipelineCreateInfo-extendedDynamicState3CoverageModulationTable-07397
  If the extendedDynamicState3CoverageModulationTable feature is not enabled, there **must** be no element of the pDynamicStates member of pDynamicState set to VK_DYNAMIC_STATE_COVERAGE_MODULATION_TABLE_NV
- VUID-VkGraphicsPipelineCreateInfo-extendedDynamicState3CoverageReductionMode-07398

If the extendedDynamicState3CoverageReductionMode feature is not enabled, there **must** be no element of the pDynamicStates member of pDynamicState set to VK_DYNAMIC_STATE_COVERAGE_REDUCTION_MODE_NV

- VUID-VkGraphicsPipelineCreateInfo-extendedDynamicState3RepresentativeFragmentTestEnable-07399
  If the extendedDynamicState3RepresentativeFragmentTestEnable feature is not enabled, there **must** be no element of the pDynamicStates member of pDynamicState set to VK_DYNAMIC_STATE_REPRESENTATIVE_FRAGMENT_TEST_ENABLE_NV
- VUID-VkGraphicsPipelineCreateInfo-extendedDynamicState3ShadingRateImageEnable-07400
  If the extendedDynamicState3ShadingRateImageEnable feature is not enabled, there **must** be no element of the pDynamicStates member of pDynamicState set to VK_DYNAMIC_STATE_SHADING_RATE_IMAGE_ENABLE_NV
- VUID-VkGraphicsPipelineCreateInfo-flags-07401
  flags **must** not include VK_PIPELINE_CREATE_RAY_TRACING_OPACITY_MICROMAP_BIT_EXT
- VUID-VkGraphicsPipelineCreateInfo-flags-07997
  flags **must** not include VK_PIPELINE_CREATE_RAY_TRACING_DISPLACEMENT_MICROMAP_BIT_NV
- VUID-VkGraphicsPipelineCreateInfo-pDynamicStates-07730
  If the pipeline requires pre-rasterization shader state, and no element of the pDynamicStates member of pDynamicState is VK_DYNAMIC_STATE_VIEWPORT or VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT, and if multiviewPerViewViewports is enabled, then the index of the most significant bit in each element of VkRenderPassMultiviewCreateInfo::pViewMasks **must** be less than pViewportState->viewportCount
- VUID-VkGraphicsPipelineCreateInfo-pDynamicStates-07731
  If the pipeline requires pre-rasterization shader state, and no element of the pDynamicStates member of pDynamicState is VK_DYNAMIC_STATE_SCISSOR or VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT, and if multiviewPerViewViewports is enabled, then the index of the most significant bit in each element of VkRenderPassMultiviewCreateInfo::pViewMasks **must** be less than pViewportState->scissorCount
- VUID-VkGraphicsPipelineCreateInfo-pStages-08711
  If pStages includes a fragment shader stage, VK_DYNAMIC_STATE_DEPTH_WRITE_ENABLE is not set in VkPipelineDynamicStateCreateInfo::pDynamicStates, and the fragment shader declares the EarlyFragmentTests execution mode and uses OpDepthAttachmentReadEXT, the depthWriteEnable member of VkPipelineDepthStencilStateCreateInfo **must** be VK_FALSE
- VUID-VkGraphicsPipelineCreateInfo-pStages-08712
  If pStages includes a fragment shader stage, VK_DYNAMIC_STATE_STENCIL_WRITE_MASK is not set in VkPipelineDynamicStateCreateInfo::pDynamicStates, and the fragment shader declares the EarlyFragmentTests execution mode and uses OpStencilAttachmentReadEXT, the value of VkStencilOpState::writeMask for both front and back in VkPipelineDepthStencilStateCreateInfo **must** be 0
- VUID-VkGraphicsPipelineCreateInfo-renderPass-08744
  If renderPass is VK_NULL_HANDLE, the pipeline requires fragment output state or

fragment shader state, the pipeline enables sample shading, rasterizationSamples is not dynamic, and the pNext chain includes a VkPipelineRenderingCreateInfo structure, rasterizationSamples **must** be a valid VkSampleCountFlagBits value that is set in imageCreateSampleCounts (as defined in Image Creation Limits) for every element of depthAttachmentFormat, stencilAttachmentFormat and the pColorAttachmentFormats array which is not VK_FORMAT_UNDEFINED

- VUID-VkGraphicsPipelineCreateInfo-flags-08897
  If VkGraphicsPipelineLibraryCreateInfoEXT::flags includes VK_GRAPHICS_PIPELINE_LIBRARY_VERTEX_INPUT_INTERFACE_BIT_EXT, pre-rasterization shader state is specified either in a library or by the inclusion of VK_GRAPHICS_PIPELINE_LIBRARY_PRE_RASTERIZATION_SHADERS_BIT_EXT, and that state includes a vertex shader stage in pStages, the pipeline **must** define vertex input state
- VUID-VkGraphicsPipelineCreateInfo-flags-08898
  If VkGraphicsPipelineLibraryCreateInfoEXT::flags includes VK_GRAPHICS_PIPELINE_LIBRARY_VERTEX_INPUT_INTERFACE_BIT_EXT, and pre-rasterization shader state is not specified, the pipeline **must** define vertex input state
- VUID-VkGraphicsPipelineCreateInfo-flags-08899
  If flags does not include VK_PIPELINE_CREATE_LIBRARY_BIT_KHR, pre-rasterization shader state is specified either in a library or by the inclusion of VK_GRAPHICS_PIPELINE_LIBRARY_PRE_RASTERIZATION_SHADERS_BIT_EXT, and that state includes a vertex shader stage in pStages, the pipeline **must** either define vertex input state or include that state in a linked pipeline library
- VUID-VkGraphicsPipelineCreateInfo-flags-08900
  If VkGraphicsPipelineLibraryCreateInfoEXT::flags includes VK_GRAPHICS_PIPELINE_LIBRARY_PRE_RASTERIZATION_SHADERS_BIT_EXT the pipeline **must** define pre-rasterization shader state
- VUID-VkGraphicsPipelineCreateInfo-flags-08901
  If flags does not include VK_PIPELINE_CREATE_LIBRARY_BIT_KHR, the pipeline **must** either define pre-rasterization shader state or include that state in a linked pipeline library
- VUID-VkGraphicsPipelineCreateInfo-flags-08903
  If VkGraphicsPipelineLibraryCreateInfoEXT::flags includes VK_GRAPHICS_PIPELINE_LIBRARY_FRAGMENT_SHADER_BIT_EXT, pre-rasterization shader state is specified either in a library or by the inclusion of VK_GRAPHICS_PIPELINE_LIBRARY_PRE_RASTERIZATION_SHADERS_BIT_EXT, and that state either includes VK_DYNAMIC_STATE_RASTERIZER_DISCARD_ENABLE or has pRasterizationState->rasterizerDiscardEnable set to VK_FALSE, the pipeline **must** define fragment shader state
- VUID-VkGraphicsPipelineCreateInfo-flags-08904
  If VkGraphicsPipelineLibraryCreateInfoEXT::flags includes VK_GRAPHICS_PIPELINE_LIBRARY_FRAGMENT_SHADER_BIT_EXT, and pre-rasterization shader state is not specified, the pipeline **must** define fragment shader state
- VUID-VkGraphicsPipelineCreateInfo-flags-08906
  If VkGraphicsPipelineLibraryCreateInfoEXT::flags includes VK_GRAPHICS_PIPELINE_LIBRARY_FRAGMENT_SHADER_BIT_EXT, pre-rasterization shader state is specified either in a library or by the inclusion of VK_GRAPHICS_PIPELINE_LIBRARY_FRAGMENT_OUTPUT_INTERFACE_BIT_EXT, and that state either

includes VK_DYNAMIC_STATE_RASTERIZER_DISCARD_ENABLE or has pRasterizationState->rasterizerDiscardEnable set to VK_FALSE, the pipeline **must** define fragment output interface state

- VUID-VkGraphicsPipelineCreateInfo-flags-08907
  If VkGraphicsPipelineLibraryCreateInfoEXT::flags includes VK_GRAPHICS_PIPELINE_LIBRARY_FRAGMENT_SHADER_BIT_EXT, and pre-rasterization shader state is not specified, the pipeline **must** define fragment output interface state
- VUID-VkGraphicsPipelineCreateInfo-flags-08909
  If flags does not include VK_PIPELINE_CREATE_LIBRARY_BIT_KHR, pre-rasterization shader state is specified either in a library or by the inclusion of VK_GRAPHICS_PIPELINE_LIBRARY_PRE_RASTERIZATION_SHADERS_BIT_EXT, and that state either includes VK_DYNAMIC_STATE_RASTERIZER_DISCARD_ENABLE or has pRasterizationState->rasterizerDiscardEnable set to VK_FALSE, the pipeline **must** define fragment output interface state and fragment shader state or include those states in linked pipeline libraries
- VUID-VkGraphicsPipelineCreateInfo-None-09043
  If pDynamicState->pDynamicStates does not include VK_DYNAMIC_STATE_COLOR_WRITE_MASK_EXT, and the format of any color attachment is VK_FORMAT_E5B9G9R9_UFLOAT_PACK32, the colorWriteMask member of the corresponding element of pColorBlendState->pAttachments **must** either include all of VK_COLOR_COMPONENT_R_BIT, VK_COLOR_COMPONENT_G_BIT, and VK_COLOR_COMPONENT_B_BIT, or none of them
- VUID-VkGraphicsPipelineCreateInfo-externalFormatResolve-09301
  If the externalFormatResolve feature is enabled, the pipeline requires fragment output interface state, renderPass is VK_NULL_HANDLE, and VkExternalFormatANDROID::externalFormat is not 0, VkPipelineRenderingCreateInfo::viewMask **must** be 0
- VUID-VkGraphicsPipelineCreateInfo-externalFormatResolve-09304
  If the externalFormatResolve feature is enabled, the pipeline requires fragment output interface state, renderPass is VK_NULL_HANDLE, VkExternalFormatANDROID::externalFormat is not 0, and rasterizationSamples is not dynamic, VkPipelineMultisampleStateCreateInfo::rasterizationSamples **must** be 1
- VUID-VkGraphicsPipelineCreateInfo-externalFormatResolve-09305
  If the externalFormatResolve feature is enabled, the pipeline requires fragment output interface state, renderPass is VK_NULL_HANDLE, and VkExternalFormatANDROID::externalFormat is not 0, and blendEnable is not dynamic, the blendEnable member of each element of pColorBlendState->pAttachments **must** be VK_FALSE
- VUID-VkGraphicsPipelineCreateInfo-externalFormatResolve-09306
  If the externalFormatResolve feature is enabled, the pipeline requires fragment output interface state, renderPass is VK_NULL_HANDLE, and VkExternalFormatANDROID::externalFormat is not 0, and pDynamicState->pDynamicStates does not include VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR, VkPipelineFragmentShadingRateStateCreateInfoKHR::width **must** be 1
- VUID-VkGraphicsPipelineCreateInfo-externalFormatResolve-09307
  If the externalFormatResolve feature is enabled, the pipeline requires fragment output interface state, renderPass is VK_NULL_HANDLE, and VkExternalFormatANDROID::externalFormat is not 0, and pDynamicState->pDynamicStates does not include

VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR, VkPipelineFragmentShadingRateStateCreateInfoKHR::height **must** be 1

- VUID-VkGraphicsPipelineCreateInfo-externalFormatResolve-09308
  If the externalFormatResolve feature is enabled, the pipeline requires pre-rasterization shader state and fragment output interface state, renderPass is VK_NULL_HANDLE, and VkExternalFormatANDROID::externalFormat is not 0, the last pre-rasterization shader stage **must** not statically use a variable with the PrimitiveShadingRateKHR built-in
- VUID-VkGraphicsPipelineCreateInfo-externalFormatResolve-09309
  If the externalFormatResolve feature is enabled, the pipeline requires fragment output interface state, renderPass is VK_NULL_HANDLE, and VkExternalFormatANDROID::externalFormat is not 0, VkPipelineRenderingCreateInfo::colorAttachmentCount **must** be 1
- VUID-VkGraphicsPipelineCreateInfo-externalFormatResolve-09310
  If the externalFormatResolve feature is enabled, the pipeline requires fragment shader state and fragment output interface state, renderPass is VK_NULL_HANDLE, and VkExternalFormatANDROID::externalFormat is not 0, the fragment shader **must** not declare the DepthReplacing or StencilRefReplacingEXT execution modes
- VUID-VkGraphicsPipelineCreateInfo-externalFormatResolve-09313
  If the externalFormatResolve feature is enabled, the pipeline requires fragment output interface state, renderPass is not VK_NULL_HANDLE, subpass includes an external format resolve attachment, and rasterizationSamples is not dynamic, VkPipelineMultisampleStateCreateInfo::rasterizationSamples **must** be VK_SAMPLE_COUNT_1_BIT
- VUID-VkGraphicsPipelineCreateInfo-externalFormatResolve-09314
  If the externalFormatResolve feature is enabled, the pipeline requires fragment output interface state, renderPass is not VK_NULL_HANDLE, subpass includes an external format resolve attachment, and blendEnable is not dynamic, the blendEnable member of each element of pColorBlendState->pAttachments **must** be VK_FALSE
- VUID-VkGraphicsPipelineCreateInfo-externalFormatResolve-09315
  If the externalFormatResolve feature is enabled, the pipeline requires fragment output interface state, renderPass is not VK_NULL_HANDLE, subpass includes an external format resolve attachment, and pDynamicState->pDynamicStates does not include VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR, VkPipelineFragmentShadingRateStateCreateInfoKHR::width **must** be 1
- VUID-VkGraphicsPipelineCreateInfo-externalFormatResolve-09316
  If the externalFormatResolve feature is enabled, the pipeline requires fragment output interface state, renderPass is not VK_NULL_HANDLE, subpass includes an external format resolve attachment, and pDynamicState->pDynamicStates does not include VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR, VkPipelineFragmentShadingRateStateCreateInfoKHR::height **must** be 1
- VUID-VkGraphicsPipelineCreateInfo-externalFormatResolve-09317
  If the externalFormatResolve feature is enabled, the pipeline requires pre-rasterization shader state and fragment output interface state, renderPass is not VK_NULL_HANDLE, and subpass includes an external format resolve attachment, the last pre-rasterization shader stage **must** not statically use a variable with the PrimitiveShadingRateKHR built-in

- VUID-VkGraphicsPipelineCreateInfo-renderPass-09531
  If the pipeline is being created with fragment shader state and fragment output state, the value of `renderPass` is VK_NULL_HANDLE, and VkRenderingInputAttachmentIndexInfoKHR is included, VkRenderingInputAttachmentIndexInfoKHR::`colorAttachmentCount` **must** be equal to VkPipelineRenderingCreateInfo::`colorAttachmentCount`
- VUID-VkGraphicsPipelineCreateInfo-renderPass-09652
  If the pipeline is being created with fragment shader state and fragment output state, the value of `renderPass` is VK_NULL_HANDLE, and VkRenderingInputAttachmentIndexInfoKHR is not included, the fragment shader **must** not contain any input attachments with a `InputAttachmentIndex` greater than or equal to VkPipelineRenderingCreateInfo::`colorAttachmentCount`
- VUID-VkGraphicsPipelineCreateInfo-renderPass-09532
  If the pipeline is being created with fragment output state, and the value of `renderPass` is VK_NULL_HANDLE, VkRenderingAttachmentLocationInfoKHR::`colorAttachmentCount` **must** be equal to VkPipelineRenderingCreateInfo::`colorAttachmentCount`

## Valid Usage (Implicit)

- VUID-VkGraphicsPipelineCreateInfo-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_GRAPHICS_PIPELINE_CREATE_INFO`
- VUID-VkGraphicsPipelineCreateInfo-pNext-pNext
  Each `pNext` member of any structure (including this one) in the `pNext` chain **must** be either `NULL` or a pointer to a valid instance of VkAttachmentSampleCountInfoAMD, VkExternalFormatANDROID, VkGraphicsPipelineLibraryCreateInfoEXT, VkGraphicsPipelineShaderGroupsCreateInfoNV, VkMultiviewPerViewAttributesInfoNVX, VkPipelineBinaryInfoKHR, VkPipelineCompilerControlCreateInfoAMD, VkPipelineCreateFlags2CreateInfoKHR, VkPipelineCreationFeedbackCreateInfo, VkPipelineDiscardRectangleStateCreateInfoEXT, VkPipelineFragmentShadingRateEnumStateCreateInfoNV, VkPipelineFragmentShadingRateStateCreateInfoKHR, VkPipelineLibraryCreateInfoKHR, VkPipelineRenderingCreateInfo, VkPipelineRepresentativeFragmentTestStateCreateInfoNV, VkPipelineRobustnessCreateInfoEXT, VkRenderingAttachmentLocationInfoKHR, or VkRenderingInputAttachmentIndexInfoKHR
- VUID-VkGraphicsPipelineCreateInfo-sType-unique
  The `sType` value of each struct in the `pNext` chain **must** be unique
- VUID-VkGraphicsPipelineCreateInfo-pDynamicState-parameter
  If `pDynamicState` is not `NULL`, `pDynamicState` **must** be a valid pointer to a valid VkPipelineDynamicStateCreateInfo structure
- VUID-VkGraphicsPipelineCreateInfo-commonparent
  Each of `basePipelineHandle`, `layout`, and `renderPass` that are valid handles of non-ignored parameters **must** have been created, allocated, or retrieved from the same VkDevice

The VkPipelineRenderingCreateInfo structure is defined as:

```
typedef struct VkPipelineRenderingCreateInfo {
    VkStructureType    sType;
    const void*        pNext;
    uint32_t           viewMask;
    uint32_t           colorAttachmentCount;
    const VkFormat*    pColorAttachmentFormats;
    VkFormat           depthAttachmentFormat;
    VkFormat           stencilAttachmentFormat;
} VkPipelineRenderingCreateInfo;
```

or the equivalent

```
// Provided by VK_KHR_dynamic_rendering
typedef VkPipelineRenderingCreateInfo VkPipelineRenderingCreateInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- viewMask is the viewMask used for rendering.
- colorAttachmentCount is the number of entries in pColorAttachmentFormats
- pColorAttachmentFormats is a pointer to an array of VkFormat values defining the format of color attachments used in this pipeline.
- depthAttachmentFormat is a VkFormat value defining the format of the depth attachment used in this pipeline.
- stencilAttachmentFormat is a VkFormat value defining the format of the stencil attachment used in this pipeline.

When a pipeline is created without a VkRenderPass, if the pNext chain of VkGraphicsPipelineCreateInfo includes this structure, it specifies the view mask and format of attachments used for rendering. If this structure is not specified, and the pipeline does not include a VkRenderPass, viewMask and colorAttachmentCount are 0, and depthAttachmentFormat and stencilAttachmentFormat are VK_FORMAT_UNDEFINED. If a graphics pipeline is created with a valid VkRenderPass, parameters of this structure are ignored.

If depthAttachmentFormat, stencilAttachmentFormat, or any element of pColorAttachmentFormats is VK_FORMAT_UNDEFINED, it indicates that the corresponding attachment is unused within the render pass. Valid formats indicate that an attachment **can** be used - but it is still valid to set the attachment to NULL when beginning rendering.

If the render pass is going to be used with an external format resolve attachment, a VkExternalFormatANDROID structure **must** also be included in the pNext chain of VkGraphicsPipelineCreateInfo, defining the external format of the resolve attachment that will be used.

### Valid Usage

- VUID-VkPipelineRenderingCreateInfo-colorAttachmentCount-09533
  `colorAttachmentCount` **must** be less than or equal to `maxColorAttachments`

### Valid Usage (Implicit)

- VUID-VkPipelineRenderingCreateInfo-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PIPELINE_RENDERING_CREATE_INFO`

The `VkPipelineCreateFlags2CreateInfoKHR` structure is defined as:

```
// Provided by VK_KHR_maintenance5
typedef struct VkPipelineCreateFlags2CreateInfoKHR {
    VkStructureType              sType;
    const void*                  pNext;
    VkPipelineCreateFlags2KHR    flags;
} VkPipelineCreateFlags2CreateInfoKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is a bitmask of VkPipelineCreateFlagBits2KHR specifying how a pipeline will be generated.

If this structure is included in the `pNext` chain of a pipeline creation structure, `flags` is used instead of the corresponding `flags` value passed in that creation structure, allowing additional creation flags to be specified.

### Valid Usage (Implicit)

- VUID-VkPipelineCreateFlags2CreateInfoKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PIPELINE_CREATE_FLAGS_2_CREATE_INFO_KHR`
- VUID-VkPipelineCreateFlags2CreateInfoKHR-flags-parameter
  `flags` **must** be a valid combination of VkPipelineCreateFlagBits2KHR values
- VUID-VkPipelineCreateFlags2CreateInfoKHR-flags-requiredbitmask
  `flags` **must** not be `0`

Bits which **can** be set in VkPipelineCreateFlags2CreateInfoKHR::`flags`, specifying how a pipeline is created, are:

```
// Provided by VK_KHR_maintenance5
// Flag bits for VkPipelineCreateFlagBits2KHR
typedef VkFlags64 VkPipelineCreateFlagBits2KHR;
static const VkPipelineCreateFlagBits2KHR
```

```
VK_PIPELINE_CREATE_2_DISABLE_OPTIMIZATION_BIT_KHR = 0x00000001ULL;
static const VkPipelineCreateFlagBits2KHR
VK_PIPELINE_CREATE_2_ALLOW_DERIVATIVES_BIT_KHR = 0x00000002ULL;
static const VkPipelineCreateFlagBits2KHR VK_PIPELINE_CREATE_2_DERIVATIVE_BIT_KHR =
0x00000004ULL;
// Provided by VK_EXT_legacy_dithering with (VK_KHR_dynamic_rendering or
VK_VERSION_1_3) and VK_KHR_maintenance5
static const VkPipelineCreateFlagBits2KHR
VK_PIPELINE_CREATE_2_ENABLE_LEGACY_DITHERING_BIT_EXT = 0x400000000ULL;
// Provided by VK_KHR_maintenance5 with VK_VERSION_1_1 or VK_KHR_device_group
static const VkPipelineCreateFlagBits2KHR
VK_PIPELINE_CREATE_2_VIEW_INDEX_FROM_DEVICE_INDEX_BIT_KHR = 0x00000008ULL;
// Provided by VK_KHR_maintenance5 with VK_VERSION_1_1 or VK_KHR_device_group
static const VkPipelineCreateFlagBits2KHR VK_PIPELINE_CREATE_2_DISPATCH_BASE_BIT_KHR =
0x00000010ULL;
// Provided by VK_KHR_maintenance5 with VK_NV_ray_tracing
static const VkPipelineCreateFlagBits2KHR VK_PIPELINE_CREATE_2_DEFER_COMPILE_BIT_NV =
0x00000020ULL;
// Provided by VK_KHR_maintenance5 with VK_KHR_pipeline_executable_properties
static const VkPipelineCreateFlagBits2KHR
VK_PIPELINE_CREATE_2_CAPTURE_STATISTICS_BIT_KHR = 0x00000040ULL;
// Provided by VK_KHR_maintenance5 with VK_KHR_pipeline_executable_properties
static const VkPipelineCreateFlagBits2KHR
VK_PIPELINE_CREATE_2_CAPTURE_INTERNAL_REPRESENTATIONS_BIT_KHR = 0x00000080ULL;
// Provided by VK_KHR_maintenance5 with VK_VERSION_1_3 or
VK_EXT_pipeline_creation_cache_control
static const VkPipelineCreateFlagBits2KHR
VK_PIPELINE_CREATE_2_FAIL_ON_PIPELINE_COMPILE_REQUIRED_BIT_KHR = 0x00000100ULL;
// Provided by VK_KHR_maintenance5 with VK_VERSION_1_3 or
VK_EXT_pipeline_creation_cache_control
static const VkPipelineCreateFlagBits2KHR
VK_PIPELINE_CREATE_2_EARLY_RETURN_ON_FAILURE_BIT_KHR = 0x00000200ULL;
// Provided by VK_KHR_maintenance5 with VK_EXT_graphics_pipeline_library
static const VkPipelineCreateFlagBits2KHR
VK_PIPELINE_CREATE_2_LINK_TIME_OPTIMIZATION_BIT_EXT = 0x00000400ULL;
// Provided by VK_KHR_maintenance5 with VK_EXT_graphics_pipeline_library
static const VkPipelineCreateFlagBits2KHR
VK_PIPELINE_CREATE_2_RETAIN_LINK_TIME_OPTIMIZATION_INFO_BIT_EXT = 0x00800000ULL;
// Provided by VK_KHR_maintenance5 with VK_KHR_pipeline_library
static const VkPipelineCreateFlagBits2KHR VK_PIPELINE_CREATE_2_LIBRARY_BIT_KHR =
0x00000800ULL;
// Provided by VK_KHR_maintenance5 with VK_KHR_ray_tracing_pipeline
static const VkPipelineCreateFlagBits2KHR
VK_PIPELINE_CREATE_2_RAY_TRACING_SKIP_TRIANGLES_BIT_KHR = 0x00001000ULL;
// Provided by VK_KHR_maintenance5 with VK_KHR_ray_tracing_pipeline
static const VkPipelineCreateFlagBits2KHR
VK_PIPELINE_CREATE_2_RAY_TRACING_SKIP_AABBS_BIT_KHR = 0x00002000ULL;
// Provided by VK_KHR_maintenance5 with VK_KHR_ray_tracing_pipeline
static const VkPipelineCreateFlagBits2KHR
VK_PIPELINE_CREATE_2_RAY_TRACING_NO_NULL_ANY_HIT_SHADERS_BIT_KHR = 0x00004000ULL;
// Provided by VK_KHR_maintenance5 with VK_KHR_ray_tracing_pipeline
```

```
static const VkPipelineCreateFlagBits2KHR
VK_PIPELINE_CREATE_2_RAY_TRACING_NO_NULL_CLOSEST_HIT_SHADERS_BIT_KHR = 0x00008000ULL;
// Provided by VK_KHR_maintenance5 with VK_KHR_ray_tracing_pipeline
static const VkPipelineCreateFlagBits2KHR
VK_PIPELINE_CREATE_2_RAY_TRACING_NO_NULL_MISS_SHADERS_BIT_KHR = 0x00010000ULL;
// Provided by VK_KHR_maintenance5 with VK_KHR_ray_tracing_pipeline
static const VkPipelineCreateFlagBits2KHR
VK_PIPELINE_CREATE_2_RAY_TRACING_NO_NULL_INTERSECTION_SHADERS_BIT_KHR = 0x00020000ULL;
// Provided by VK_KHR_maintenance5 with VK_KHR_ray_tracing_pipeline
static const VkPipelineCreateFlagBits2KHR
VK_PIPELINE_CREATE_2_RAY_TRACING_SHADER_GROUP_HANDLE_CAPTURE_REPLAY_BIT_KHR =
0x00080000ULL;
// Provided by VK_KHR_maintenance5 with VK_NV_device_generated_commands
static const VkPipelineCreateFlagBits2KHR
VK_PIPELINE_CREATE_2_INDIRECT_BINDABLE_BIT_NV = 0x00040000ULL;
// Provided by VK_KHR_maintenance5 with VK_NV_ray_tracing_motion_blur
static const VkPipelineCreateFlagBits2KHR
VK_PIPELINE_CREATE_2_RAY_TRACING_ALLOW_MOTION_BIT_NV = 0x00100000ULL;
// Provided by VK_KHR_maintenance5 with (VK_KHR_dynamic_rendering or VK_VERSION_1_3)
and VK_KHR_fragment_shading_rate
static const VkPipelineCreateFlagBits2KHR
VK_PIPELINE_CREATE_2_RENDERING_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR =
0x00200000ULL;
// Provided by VK_KHR_maintenance5 with (VK_KHR_dynamic_rendering or VK_VERSION_1_3)
and VK_EXT_fragment_density_map
static const VkPipelineCreateFlagBits2KHR
VK_PIPELINE_CREATE_2_RENDERING_FRAGMENT_DENSITY_MAP_ATTACHMENT_BIT_EXT =
0x00400000ULL;
// Provided by VK_KHR_maintenance5 with VK_EXT_opacity_micromap
static const VkPipelineCreateFlagBits2KHR
VK_PIPELINE_CREATE_2_RAY_TRACING_OPACITY_MICROMAP_BIT_EXT = 0x01000000ULL;
// Provided by VK_KHR_maintenance5 with VK_EXT_attachment_feedback_loop_layout
static const VkPipelineCreateFlagBits2KHR
VK_PIPELINE_CREATE_2_COLOR_ATTACHMENT_FEEDBACK_LOOP_BIT_EXT = 0x02000000ULL;
// Provided by VK_KHR_maintenance5 with VK_EXT_attachment_feedback_loop_layout
static const VkPipelineCreateFlagBits2KHR
VK_PIPELINE_CREATE_2_DEPTH_STENCIL_ATTACHMENT_FEEDBACK_LOOP_BIT_EXT = 0x04000000ULL;
// Provided by VK_KHR_maintenance5 with VK_EXT_pipeline_protected_access
static const VkPipelineCreateFlagBits2KHR
VK_PIPELINE_CREATE_2_NO_PROTECTED_ACCESS_BIT_EXT = 0x08000000ULL;
// Provided by VK_KHR_maintenance5 with VK_EXT_pipeline_protected_access
static const VkPipelineCreateFlagBits2KHR
VK_PIPELINE_CREATE_2_PROTECTED_ACCESS_ONLY_BIT_EXT = 0x40000000ULL;
// Provided by VK_KHR_maintenance5 with VK_NV_displacement_micromap
static const VkPipelineCreateFlagBits2KHR
VK_PIPELINE_CREATE_2_RAY_TRACING_DISPLACEMENT_MICROMAP_BIT_NV = 0x10000000ULL;
// Provided by VK_KHR_maintenance5 with VK_EXT_descriptor_buffer
static const VkPipelineCreateFlagBits2KHR
VK_PIPELINE_CREATE_2_DESCRIPTOR_BUFFER_BIT_EXT = 0x20000000ULL;
// Provided by VK_KHR_pipeline_binary
static const VkPipelineCreateFlagBits2KHR VK_PIPELINE_CREATE_2_CAPTURE_DATA_BIT_KHR =
```

```
0x80000000ULL;
```

- VK_PIPELINE_CREATE_2_DISABLE_OPTIMIZATION_BIT_KHR specifies that the created pipeline will not be optimized. Using this flag **may** reduce the time taken to create the pipeline.
- VK_PIPELINE_CREATE_2_ALLOW_DERIVATIVES_BIT_KHR specifies that the pipeline to be created is allowed to be the parent of a pipeline that will be created in a subsequent pipeline creation call.
- VK_PIPELINE_CREATE_2_DERIVATIVE_BIT_KHR specifies that the pipeline to be created will be a child of a previously created parent pipeline.
- VK_PIPELINE_CREATE_2_VIEW_INDEX_FROM_DEVICE_INDEX_BIT_KHR specifies that any shader input variables decorated as ViewIndex will be assigned values as if they were decorated as DeviceIndex.
- VK_PIPELINE_CREATE_2_DISPATCH_BASE_BIT_KHR specifies that a compute pipeline **can** be used with vkCmdDispatchBase with a non-zero base workgroup.
- VK_PIPELINE_CREATE_2_DEFER_COMPILE_BIT_NV specifies that a pipeline is created with all shaders in the deferred state. Before using the pipeline the application **must** call vkCompileDeferredNV exactly once on each shader in the pipeline before using the pipeline.
- VK_PIPELINE_CREATE_2_CAPTURE_STATISTICS_BIT_KHR specifies that the shader compiler should capture statistics for the pipeline executables produced by the compile process which **can** later be retrieved by calling vkGetPipelineExecutableStatisticsKHR. Enabling this flag **must** not affect the final compiled pipeline but **may** disable pipeline caching or otherwise affect pipeline creation time.
- VK_PIPELINE_CREATE_2_CAPTURE_INTERNAL_REPRESENTATIONS_BIT_KHR specifies that the shader compiler should capture the internal representations of pipeline executables produced by the compile process which **can** later be retrieved by calling vkGetPipelineExecutableInternalRepresentationsKHR. Enabling this flag **must** not affect the final compiled pipeline but **may** disable pipeline caching or otherwise affect pipeline creation time. When capturing IR from pipelines created with pipeline libraries, there is no guarantee that IR from libraries **can** be retrieved from the linked pipeline. Applications **should** retrieve IR from each library, and any linked pipelines, separately.
- VK_PIPELINE_CREATE_2_LIBRARY_BIT_KHR specifies that the pipeline **cannot** be used directly, and instead defines a *pipeline library* that **can** be combined with other pipelines using the VkPipelineLibraryCreateInfoKHR structure. This is available in ray tracing and graphics pipelines.
- VK_PIPELINE_CREATE_2_RAY_TRACING_NO_NULL_ANY_HIT_SHADERS_BIT_KHR specifies that an any-hit shader will always be present when an any-hit shader would be executed. A NULL any-hit shader is an any-hit shader which is effectively VK_SHADER_UNUSED_KHR, such as from a shader group consisting entirely of zeros.
- VK_PIPELINE_CREATE_2_RAY_TRACING_NO_NULL_CLOSEST_HIT_SHADERS_BIT_KHR specifies that a closest hit shader will always be present when a closest hit shader would be executed. A NULL closest hit shader is a closest hit shader which is effectively VK_SHADER_UNUSED_KHR, such as from a shader group consisting entirely of zeros.
- VK_PIPELINE_CREATE_2_RAY_TRACING_NO_NULL_MISS_SHADERS_BIT_KHR specifies that a miss shader will always be present when a miss shader would be executed. A NULL miss shader is a miss

shader which is effectively VK_SHADER_UNUSED_KHR, such as from a shader group consisting entirely of zeros.

- VK_PIPELINE_CREATE_2_RAY_TRACING_NO_NULL_INTERSECTION_SHADERS_BIT_KHR specifies that an intersection shader will always be present when an intersection shader would be executed. A NULL intersection shader is an intersection shader which is effectively VK_SHADER_UNUSED_KHR, such as from a shader group consisting entirely of zeros.
- VK_PIPELINE_CREATE_2_RAY_TRACING_SKIP_TRIANGLES_BIT_KHR specifies that triangle primitives will be skipped during traversal using pipeline trace ray instructions.
- VK_PIPELINE_CREATE_2_RAY_TRACING_SKIP_AABBS_BIT_KHR specifies that AABB primitives will be skipped during traversal using pipeline trace ray instructions.
- VK_PIPELINE_CREATE_2_RAY_TRACING_SHADER_GROUP_HANDLE_CAPTURE_REPLAY_BIT_KHR specifies that the shader group handles **can** be saved and reused on a subsequent run (e.g. for trace capture and replay).
- VK_PIPELINE_CREATE_2_INDIRECT_BINDABLE_BIT_NV specifies that the pipeline can be used in combination with Device-Generated Commands.
- VK_PIPELINE_CREATE_2_FAIL_ON_PIPELINE_COMPILE_REQUIRED_BIT_KHR specifies that pipeline creation will fail if a compile is required for creation of a valid VkPipeline object; VK_PIPELINE_COMPILE_REQUIRED will be returned by pipeline creation, and the VkPipeline will be set to VK_NULL_HANDLE.
- When creating multiple pipelines, VK_PIPELINE_CREATE_2_EARLY_RETURN_ON_FAILURE_BIT_KHR specifies that control will be returned to the application if any individual pipeline returns a result which is not VK_SUCCESS rather than continuing to create additional pipelines.
- VK_PIPELINE_CREATE_2_RAY_TRACING_ALLOW_MOTION_BIT_NV specifies that the pipeline is allowed to use OpTraceRayMotionNV.
- VK_PIPELINE_CREATE_2_RENDERING_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR specifies that the pipeline will be used with a fragment shading rate attachment.
- VK_PIPELINE_CREATE_2_RENDERING_FRAGMENT_DENSITY_MAP_ATTACHMENT_BIT_EXT specifies that the pipeline will be used with a fragment density map attachment.
- VK_PIPELINE_CREATE_2_LINK_TIME_OPTIMIZATION_BIT_EXT specifies that pipeline libraries being linked into this library **should** have link time optimizations applied. If this bit is omitted, implementations **should** instead perform linking as rapidly as possible.
- VK_PIPELINE_CREATE_2_RETAIN_LINK_TIME_OPTIMIZATION_INFO_BIT_EXT specifies that pipeline libraries should retain any information necessary to later perform an optimal link with VK_PIPELINE_CREATE_2_LINK_TIME_OPTIMIZATION_BIT_EXT.
- VK_PIPELINE_CREATE_2_DESCRIPTOR_BUFFER_BIT_EXT specifies that a pipeline will be used with descriptor buffers, rather than descriptor sets.
- VK_PIPELINE_CREATE_2_COLOR_ATTACHMENT_FEEDBACK_LOOP_BIT_EXT specifies that the pipeline **may** be used with an attachment feedback loop including color attachments.
- VK_PIPELINE_CREATE_2_DEPTH_STENCIL_ATTACHMENT_FEEDBACK_LOOP_BIT_EXT specifies that the pipeline **may** be used with an attachment feedback loop including depth-stencil attachments.
- VK_PIPELINE_CREATE_2_RAY_TRACING_OPACITY_MICROMAP_BIT_EXT specifies that the ray tracing

pipeline **can** be used with acceleration structures which reference an opacity micromap array.

- VK_PIPELINE_CREATE_2_RAY_TRACING_DISPLACEMENT_MICROMAP_BIT_NV specifies that the ray tracing pipeline **can** be used with acceleration structures which reference a displacement micromap array.
- VK_PIPELINE_CREATE_2_NO_PROTECTED_ACCESS_BIT_EXT specifies that the pipeline **must** not be bound to a protected command buffer.
- VK_PIPELINE_CREATE_2_PROTECTED_ACCESS_ONLY_BIT_EXT specifies that the pipeline **must** not be bound to an unprotected command buffer.
- VK_PIPELINE_CREATE_2_CAPTURE_DATA_BIT_KHR specifies that VkPipelineBinaryKHR objects **can** be created from the pipeline. If VK_PIPELINE_CREATE_2_CAPTURE_DATA_BIT_KHR is used, implementations **should** not store pipeline data to an internal cache, if such a cache exists as stated by pipelineBinaryInternalCache. If pipelineBinaryPrefersInternalCache is VK_TRUE, applications **should** not use VK_PIPELINE_CREATE_2_CAPTURE_DATA_BIT_KHR.
- VK_PIPELINE_CREATE_2_ENABLE_LEGACY_DITHERING_BIT_EXT specifies that the pipeline will be used in a render pass that is begun with VK_RENDERING_ENABLE_LEGACY_DITHERING_BIT_EXT.

It is valid to set both VK_PIPELINE_CREATE_2_ALLOW_DERIVATIVES_BIT_KHR and VK_PIPELINE_CREATE_2_DERIVATIVE_BIT_KHR. This allows a pipeline to be both a parent and possibly a child in a pipeline hierarchy. See Pipeline Derivatives for more information.

When an implementation is looking up a pipeline in a pipeline cache, if that pipeline is being created using linked libraries, implementations **should** always return an equivalent pipeline created with VK_PIPELINE_CREATE_2_LINK_TIME_OPTIMIZATION_BIT_EXT if available, whether or not that bit was specified.

**NOTE** Using VK_PIPELINE_CREATE_2_LINK_TIME_OPTIMIZATION_BIT_EXT (or not) when linking pipeline libraries is intended as a performance tradeoff between host and device. If the bit is omitted, linking should be faster and produce a pipeline more rapidly, but performance of the pipeline on the target device may be reduced. If the bit is included, linking may be slower but should produce a pipeline with device performance comparable to a monolithically created pipeline. Using both options can allow latency-sensitive applications to generate a suboptimal but usable pipeline quickly, and then perform an optimal link in the background, substituting the result for the suboptimally linked pipeline as soon as it is available.

```
// Provided by VK_KHR_maintenance5
typedef VkFlags64 VkPipelineCreateFlags2KHR;
```

VkPipelineCreateFlags2KHR is a bitmask type for setting a mask of zero or more VkPipelineCreateFlagBits2KHR.

Bits which **can** be set in

- VkGraphicsPipelineCreateInfo::flags
- VkComputePipelineCreateInfo::flags

- VkRayTracingPipelineCreateInfoKHR::flags
- VkRayTracingPipelineCreateInfoNV::flags

specify how a pipeline is created, and are:

```
// Provided by VK_VERSION_1_0
typedef enum VkPipelineCreateFlagBits {
    VK_PIPELINE_CREATE_DISABLE_OPTIMIZATION_BIT = 0x00000001,
    VK_PIPELINE_CREATE_ALLOW_DERIVATIVES_BIT = 0x00000002,
    VK_PIPELINE_CREATE_DERIVATIVE_BIT = 0x00000004,
  // Provided by VK_VERSION_1_1
    VK_PIPELINE_CREATE_VIEW_INDEX_FROM_DEVICE_INDEX_BIT = 0x00000008,
  // Provided by VK_VERSION_1_1
    VK_PIPELINE_CREATE_DISPATCH_BASE_BIT = 0x00000010,
    VK_PIPELINE_CREATE_FAIL_ON_PIPELINE_COMPILE_REQUIRED_BIT = 0x00000100,
    VK_PIPELINE_CREATE_EARLY_RETURN_ON_FAILURE_BIT = 0x00000200,
  // Provided by VK_KHR_dynamic_rendering with VK_KHR_fragment_shading_rate
    VK_PIPELINE_CREATE_RENDERING_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR =
0x00200000,
  // Provided by VK_KHR_dynamic_rendering with VK_EXT_fragment_density_map
    VK_PIPELINE_CREATE_RENDERING_FRAGMENT_DENSITY_MAP_ATTACHMENT_BIT_EXT = 0x00400000,
  // Provided by VK_KHR_ray_tracing_pipeline
    VK_PIPELINE_CREATE_RAY_TRACING_NO_NULL_ANY_HIT_SHADERS_BIT_KHR = 0x00004000,
  // Provided by VK_KHR_ray_tracing_pipeline
    VK_PIPELINE_CREATE_RAY_TRACING_NO_NULL_CLOSEST_HIT_SHADERS_BIT_KHR = 0x00008000,
  // Provided by VK_KHR_ray_tracing_pipeline
    VK_PIPELINE_CREATE_RAY_TRACING_NO_NULL_MISS_SHADERS_BIT_KHR = 0x00010000,
  // Provided by VK_KHR_ray_tracing_pipeline
    VK_PIPELINE_CREATE_RAY_TRACING_NO_NULL_INTERSECTION_SHADERS_BIT_KHR = 0x00020000,
  // Provided by VK_KHR_ray_tracing_pipeline
    VK_PIPELINE_CREATE_RAY_TRACING_SKIP_TRIANGLES_BIT_KHR = 0x00001000,
  // Provided by VK_KHR_ray_tracing_pipeline
    VK_PIPELINE_CREATE_RAY_TRACING_SKIP_AABBS_BIT_KHR = 0x00002000,
  // Provided by VK_KHR_ray_tracing_pipeline
    VK_PIPELINE_CREATE_RAY_TRACING_SHADER_GROUP_HANDLE_CAPTURE_REPLAY_BIT_KHR =
0x00080000,
  // Provided by VK_NV_ray_tracing
    VK_PIPELINE_CREATE_DEFER_COMPILE_BIT_NV = 0x00000020,
  // Provided by VK_KHR_pipeline_executable_properties
    VK_PIPELINE_CREATE_CAPTURE_STATISTICS_BIT_KHR = 0x00000040,
  // Provided by VK_KHR_pipeline_executable_properties
    VK_PIPELINE_CREATE_CAPTURE_INTERNAL_REPRESENTATIONS_BIT_KHR = 0x00000080,
  // Provided by VK_NV_device_generated_commands
    VK_PIPELINE_CREATE_INDIRECT_BINDABLE_BIT_NV = 0x00040000,
  // Provided by VK_KHR_pipeline_library
    VK_PIPELINE_CREATE_LIBRARY_BIT_KHR = 0x00000800,
  // Provided by VK_EXT_descriptor_buffer
    VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT = 0x20000000,
  // Provided by VK_EXT_graphics_pipeline_library
    VK_PIPELINE_CREATE_RETAIN_LINK_TIME_OPTIMIZATION_INFO_BIT_EXT = 0x00800000,
```

```
  // Provided by VK_EXT_graphics_pipeline_library
    VK_PIPELINE_CREATE_LINK_TIME_OPTIMIZATION_BIT_EXT = 0x00000400,
  // Provided by VK_NV_ray_tracing_motion_blur
    VK_PIPELINE_CREATE_RAY_TRACING_ALLOW_MOTION_BIT_NV = 0x00100000,
  // Provided by VK_EXT_attachment_feedback_loop_layout
    VK_PIPELINE_CREATE_COLOR_ATTACHMENT_FEEDBACK_LOOP_BIT_EXT = 0x02000000,
  // Provided by VK_EXT_attachment_feedback_loop_layout
    VK_PIPELINE_CREATE_DEPTH_STENCIL_ATTACHMENT_FEEDBACK_LOOP_BIT_EXT = 0x04000000,
  // Provided by VK_EXT_opacity_micromap
    VK_PIPELINE_CREATE_RAY_TRACING_OPACITY_MICROMAP_BIT_EXT = 0x01000000,
#ifdef VK_ENABLE_BETA_EXTENSIONS
  // Provided by VK_NV_displacement_micromap
    VK_PIPELINE_CREATE_RAY_TRACING_DISPLACEMENT_MICROMAP_BIT_NV = 0x10000000,
#endif
  // Provided by VK_EXT_pipeline_protected_access
    VK_PIPELINE_CREATE_NO_PROTECTED_ACCESS_BIT_EXT = 0x08000000,
  // Provided by VK_EXT_pipeline_protected_access
    VK_PIPELINE_CREATE_PROTECTED_ACCESS_ONLY_BIT_EXT = 0x40000000,
  // Provided by VK_VERSION_1_1
    VK_PIPELINE_CREATE_DISPATCH_BASE = VK_PIPELINE_CREATE_DISPATCH_BASE_BIT,
  // Provided by VK_KHR_dynamic_rendering with VK_KHR_fragment_shading_rate
  // VK_PIPELINE_RASTERIZATION_STATE_CREATE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR
is a deprecated alias
    VK_PIPELINE_RASTERIZATION_STATE_CREATE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR =
VK_PIPELINE_CREATE_RENDERING_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR,
  // Provided by VK_KHR_dynamic_rendering with VK_EXT_fragment_density_map
  // VK_PIPELINE_RASTERIZATION_STATE_CREATE_FRAGMENT_DENSITY_MAP_ATTACHMENT_BIT_EXT is
a deprecated alias
    VK_PIPELINE_RASTERIZATION_STATE_CREATE_FRAGMENT_DENSITY_MAP_ATTACHMENT_BIT_EXT =
VK_PIPELINE_CREATE_RENDERING_FRAGMENT_DENSITY_MAP_ATTACHMENT_BIT_EXT,
  // Provided by VK_KHR_device_group
    VK_PIPELINE_CREATE_VIEW_INDEX_FROM_DEVICE_INDEX_BIT_KHR =
VK_PIPELINE_CREATE_VIEW_INDEX_FROM_DEVICE_INDEX_BIT,
  // Provided by VK_KHR_device_group
    VK_PIPELINE_CREATE_DISPATCH_BASE_KHR = VK_PIPELINE_CREATE_DISPATCH_BASE,
  // Provided by VK_EXT_pipeline_creation_cache_control
    VK_PIPELINE_CREATE_FAIL_ON_PIPELINE_COMPILE_REQUIRED_BIT_EXT =
VK_PIPELINE_CREATE_FAIL_ON_PIPELINE_COMPILE_REQUIRED_BIT,
  // Provided by VK_EXT_pipeline_creation_cache_control
    VK_PIPELINE_CREATE_EARLY_RETURN_ON_FAILURE_BIT_EXT =
VK_PIPELINE_CREATE_EARLY_RETURN_ON_FAILURE_BIT,
} VkPipelineCreateFlagBits;
```

- `VK_PIPELINE_CREATE_DISABLE_OPTIMIZATION_BIT` specifies that the created pipeline will not be optimized. Using this flag **may** reduce the time taken to create the pipeline.
- `VK_PIPELINE_CREATE_ALLOW_DERIVATIVES_BIT` specifies that the pipeline to be created is allowed to be the parent of a pipeline that will be created in a subsequent pipeline creation call.
- `VK_PIPELINE_CREATE_DERIVATIVE_BIT` specifies that the pipeline to be created will be a child of a previously created parent pipeline.

- VK_PIPELINE_CREATE_VIEW_INDEX_FROM_DEVICE_INDEX_BIT specifies that any shader input variables decorated as ViewIndex will be assigned values as if they were decorated as DeviceIndex.
- VK_PIPELINE_CREATE_DISPATCH_BASE specifies that a compute pipeline **can** be used with vkCmdDispatchBase with a non-zero base workgroup.
- VK_PIPELINE_CREATE_DEFER_COMPILE_BIT_NV specifies that a pipeline is created with all shaders in the deferred state. Before using the pipeline the application **must** call vkCompileDeferredNV exactly once on each shader in the pipeline before using the pipeline.
- VK_PIPELINE_CREATE_CAPTURE_STATISTICS_BIT_KHR specifies that the shader compiler should capture statistics for the pipeline executables produced by the compile process which **can** later be retrieved by calling vkGetPipelineExecutableStatisticsKHR. Enabling this flag **must** not affect the final compiled pipeline but **may** disable pipeline caching or otherwise affect pipeline creation time.
- VK_PIPELINE_CREATE_CAPTURE_INTERNAL_REPRESENTATIONS_BIT_KHR specifies that the shader compiler should capture the internal representations of pipeline executables produced by the compile process which **can** later be retrieved by calling vkGetPipelineExecutableInternalRepresentationsKHR. Enabling this flag **must** not affect the final compiled pipeline but **may** disable pipeline caching or otherwise affect pipeline creation time. When capturing IR from pipelines created with pipeline libraries, there is no guarantee that IR from libraries **can** be retrieved from the linked pipeline. Applications **should** retrieve IR from each library, and any linked pipelines, separately.
- VK_PIPELINE_CREATE_LIBRARY_BIT_KHR specifies that the pipeline **cannot** be used directly, and instead defines a *pipeline library* that **can** be combined with other pipelines using the VkPipelineLibraryCreateInfoKHR structure. This is available in ray tracing and graphics pipelines.
- VK_PIPELINE_CREATE_RAY_TRACING_NO_NULL_ANY_HIT_SHADERS_BIT_KHR specifies that an any-hit shader will always be present when an any-hit shader would be executed. A NULL any-hit shader is an any-hit shader which is effectively VK_SHADER_UNUSED_KHR, such as from a shader group consisting entirely of zeros.
- VK_PIPELINE_CREATE_RAY_TRACING_NO_NULL_CLOSEST_HIT_SHADERS_BIT_KHR specifies that a closest hit shader will always be present when a closest hit shader would be executed. A NULL closest hit shader is a closest hit shader which is effectively VK_SHADER_UNUSED_KHR, such as from a shader group consisting entirely of zeros.
- VK_PIPELINE_CREATE_RAY_TRACING_NO_NULL_MISS_SHADERS_BIT_KHR specifies that a miss shader will always be present when a miss shader would be executed. A NULL miss shader is a miss shader which is effectively VK_SHADER_UNUSED_KHR, such as from a shader group consisting entirely of zeros.
- VK_PIPELINE_CREATE_RAY_TRACING_NO_NULL_INTERSECTION_SHADERS_BIT_KHR specifies that an intersection shader will always be present when an intersection shader would be executed. A NULL intersection shader is an intersection shader which is effectively VK_SHADER_UNUSED_KHR, such as from a shader group consisting entirely of zeros.
- VK_PIPELINE_CREATE_RAY_TRACING_SKIP_TRIANGLES_BIT_KHR specifies that triangle primitives will be skipped during traversal using pipeline trace ray instructions.
- VK_PIPELINE_CREATE_RAY_TRACING_SKIP_AABBS_BIT_KHR specifies that AABB primitives will be

skipped during traversal using pipeline trace ray instructions.

- VK_PIPELINE_CREATE_RAY_TRACING_SHADER_GROUP_HANDLE_CAPTURE_REPLAY_BIT_KHR specifies that the shader group handles **can** be saved and reused on a subsequent run (e.g. for trace capture and replay).
- VK_PIPELINE_CREATE_INDIRECT_BINDABLE_BIT_NV specifies that the pipeline can be used in combination with Device-Generated Commands.
- VK_PIPELINE_CREATE_FAIL_ON_PIPELINE_COMPILE_REQUIRED_BIT specifies that pipeline creation will fail if a compile is required for creation of a valid VkPipeline object; VK_PIPELINE_COMPILE_REQUIRED will be returned by pipeline creation, and the VkPipeline will be set to VK_NULL_HANDLE.
- When creating multiple pipelines, VK_PIPELINE_CREATE_EARLY_RETURN_ON_FAILURE_BIT specifies that control will be returned to the application if any individual pipeline returns a result which is not VK_SUCCESS rather than continuing to create additional pipelines.
- VK_PIPELINE_CREATE_RAY_TRACING_ALLOW_MOTION_BIT_NV specifies that the pipeline is allowed to use OpTraceRayMotionNV.
- VK_PIPELINE_CREATE_RENDERING_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR specifies that the pipeline will be used with a fragment shading rate attachment and dynamic rendering.
- VK_PIPELINE_CREATE_RENDERING_FRAGMENT_DENSITY_MAP_ATTACHMENT_BIT_EXT specifies that the pipeline will be used with a fragment density map attachment and dynamic rendering.
- VK_PIPELINE_CREATE_LINK_TIME_OPTIMIZATION_BIT_EXT specifies that pipeline libraries being linked into this library **should** have link time optimizations applied. If this bit is omitted, implementations **should** instead perform linking as rapidly as possible.
- VK_PIPELINE_CREATE_RETAIN_LINK_TIME_OPTIMIZATION_INFO_BIT_EXT specifies that pipeline libraries should retain any information necessary to later perform an optimal link with VK_PIPELINE_CREATE_LINK_TIME_OPTIMIZATION_BIT_EXT.
- VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT specifies that a pipeline will be used with descriptor buffers, rather than descriptor sets.
- VK_PIPELINE_CREATE_COLOR_ATTACHMENT_FEEDBACK_LOOP_BIT_EXT specifies that the pipeline **may** be used with an attachment feedback loop including color attachments. It is ignored if VK_DYNAMIC_STATE_ATTACHMENT_FEEDBACK_LOOP_ENABLE_EXT is set in pDynamicStates.
- VK_PIPELINE_CREATE_DEPTH_STENCIL_ATTACHMENT_FEEDBACK_LOOP_BIT_EXT specifies that the pipeline **may** be used with an attachment feedback loop including depth-stencil attachments. It is ignored if VK_DYNAMIC_STATE_ATTACHMENT_FEEDBACK_LOOP_ENABLE_EXT is set in pDynamicStates.
- VK_PIPELINE_CREATE_RAY_TRACING_OPACITY_MICROMAP_BIT_EXT specifies that the ray tracing pipeline **can** be used with acceleration structures which reference an opacity micromap array.
- VK_PIPELINE_CREATE_RAY_TRACING_DISPLACEMENT_MICROMAP_BIT_NV specifies that the ray tracing pipeline **can** be used with acceleration structures which reference a displacement micromap array.
- VK_PIPELINE_CREATE_NO_PROTECTED_ACCESS_BIT_EXT specifies that the pipeline **must** not be bound to a protected command buffer.
- VK_PIPELINE_CREATE_PROTECTED_ACCESS_ONLY_BIT_EXT specifies that the pipeline **must** not be

bound to an unprotected command buffer.

It is valid to set both VK_PIPELINE_CREATE_ALLOW_DERIVATIVES_BIT and VK_PIPELINE_CREATE_DERIVATIVE_BIT. This allows a pipeline to be both a parent and possibly a child in a pipeline hierarchy. See Pipeline Derivatives for more information.

When an implementation is looking up a pipeline in a pipeline cache, if that pipeline is being created using linked libraries, implementations **should** always return an equivalent pipeline created with VK_PIPELINE_CREATE_LINK_TIME_OPTIMIZATION_BIT_EXT if available, whether or not that bit was specified.

**NOTE** Using VK_PIPELINE_CREATE_LINK_TIME_OPTIMIZATION_BIT_EXT (or not) when linking pipeline libraries is intended as a performance tradeoff between host and device. If the bit is omitted, linking should be faster and produce a pipeline more rapidly, but performance of the pipeline on the target device may be reduced. If the bit is included, linking may be slower but should produce a pipeline with device performance comparable to a monolithically created pipeline. Using both options can allow latency-sensitive applications to generate a suboptimal but usable pipeline quickly, and then perform an optimal link in the background, substituting the result for the suboptimally linked pipeline as soon as it is available.

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkPipelineCreateFlags;
```

VkPipelineCreateFlags is a bitmask type for setting a mask of zero or more VkPipelineCreateFlagBits.

The VkPipelineBinaryInfoKHR structure is defined as:

```
// Provided by VK_KHR_pipeline_binary
typedef struct VkPipelineBinaryInfoKHR {
    VkStructureType                 sType;
    const void*                     pNext;
    uint32_t                        binaryCount;
    const VkPipelineBinaryKHR*      pPipelineBinaries;
} VkPipelineBinaryInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- binaryCount is the number of elements in the pPipelineBinaries array.
- pPipelineBinaries is a pointer to an array of VkPipelineBinaryKHR handles.

If a VkPipelineBinaryInfoKHR structure with a binaryCount greater than 0 is included in the pNext chain of any Vk*PipelineCreateInfo structure when creating a pipeline, implementations **must** use the data in pPipelineBinaries instead of recalculating it. Any shader module identifiers or shader modules declared in VkPipelineShaderStageCreateInfo instances are ignored.

If this structure is not included in the pNext chain, it is equivalent to specifying this structure with a binaryCount of 0.

### Valid Usage

- VUID-VkPipelineBinaryInfoKHR-binaryCount-09603
  binaryCount and the order of the elements in pPipelineBinaries **must** exactly match that returned by vkCreatePipelineBinariesKHR for the matching Vk*PipelineCreateInfo structure and its pNext chain, ignoring the presence of the VkPipelineBinaryInfoKHR structure, the presence of the VK_PIPELINE_CREATE_2_CAPTURE_DATA_BIT_KHR flag, and absence of any shader module identifiers or shader modules, for the same global pipeline key, from either:
  - VkPipelineBinaryCreateInfoKHR::pPipelineCreateInfo, or
  - VkPipelineBinaryCreateInfoKHR::pipeline

### Valid Usage (Implicit)

- VUID-VkPipelineBinaryInfoKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PIPELINE_BINARY_INFO_KHR
- VUID-VkPipelineBinaryInfoKHR-pPipelineBinaries-parameter
  If binaryCount is not 0, pPipelineBinaries **must** be a valid pointer to an array of binaryCount valid VkPipelineBinaryKHR handles

The VkGraphicsPipelineLibraryCreateInfoEXT structure is defined as:

```
// Provided by VK_EXT_graphics_pipeline_library
typedef struct VkGraphicsPipelineLibraryCreateInfoEXT {
    VkStructureType                     sType;
    const void*                         pNext;
    VkGraphicsPipelineLibraryFlagsEXT   flags;
} VkGraphicsPipelineLibraryCreateInfoEXT;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- flags is a bitmask of VkGraphicsPipelineLibraryFlagBitsEXT specifying the subsets of the graphics pipeline that are being compiled.

If a VkGraphicsPipelineLibraryCreateInfoEXT structure is included in the pNext chain of VkGraphicsPipelineCreateInfo, it specifies the subsets of the graphics pipeline being created, excluding any subsets from linked pipeline libraries. If the pipeline is created with pipeline libraries, state from those libraries is aggregated with said subset.

If this structure is omitted, and either VkGraphicsPipelineCreateInfo::flags includes VK_PIPELINE_CREATE_LIBRARY_BIT_KHR or the VkGraphicsPipelineCreateInfo::pNext chain includes a

VkPipelineLibraryCreateInfoKHR structure with a libraryCount greater than 0, it is as if flags is 0. Otherwise if this structure is omitted, it is as if flags includes all possible subsets of the graphics pipeline (i.e. a complete graphics pipeline).

### Valid Usage (Implicit)

- VUID-VkGraphicsPipelineLibraryCreateInfoEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_GRAPHICS_PIPELINE_LIBRARY_CREATE_INFO_EXT
- VUID-VkGraphicsPipelineLibraryCreateInfoEXT-flags-parameter
  flags **must** be a valid combination of VkGraphicsPipelineLibraryFlagBitsEXT values
- VUID-VkGraphicsPipelineLibraryCreateInfoEXT-flags-requiredbitmask
  flags **must** not be 0

```
// Provided by VK_EXT_graphics_pipeline_library
typedef VkFlags VkGraphicsPipelineLibraryFlagsEXT;
```

VkGraphicsPipelineLibraryFlagsEXT is a bitmask type for setting a mask of zero or more VkGraphicsPipelineLibraryFlagBitsEXT.

Possible values of the flags member of VkGraphicsPipelineLibraryCreateInfoEXT, specifying the subsets of a graphics pipeline to compile are:

```
// Provided by VK_EXT_graphics_pipeline_library
typedef enum VkGraphicsPipelineLibraryFlagBitsEXT {
    VK_GRAPHICS_PIPELINE_LIBRARY_VERTEX_INPUT_INTERFACE_BIT_EXT = 0x00000001,
    VK_GRAPHICS_PIPELINE_LIBRARY_PRE_RASTERIZATION_SHADERS_BIT_EXT = 0x00000002,
    VK_GRAPHICS_PIPELINE_LIBRARY_FRAGMENT_SHADER_BIT_EXT = 0x00000004,
    VK_GRAPHICS_PIPELINE_LIBRARY_FRAGMENT_OUTPUT_INTERFACE_BIT_EXT = 0x00000008,
} VkGraphicsPipelineLibraryFlagBitsEXT;
```

- VK_GRAPHICS_PIPELINE_LIBRARY_VERTEX_INPUT_INTERFACE_BIT_EXT specifies that a pipeline will include vertex input interface state.
- VK_GRAPHICS_PIPELINE_LIBRARY_PRE_RASTERIZATION_SHADERS_BIT_EXT specifies that a pipeline will include pre-rasterization shader state.
- VK_GRAPHICS_PIPELINE_LIBRARY_FRAGMENT_SHADER_BIT_EXT specifies that a pipeline will include fragment shader state.
- VK_GRAPHICS_PIPELINE_LIBRARY_FRAGMENT_OUTPUT_INTERFACE_BIT_EXT specifies that a pipeline will include fragment output interface state.

The VkPipelineDynamicStateCreateInfo structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkPipelineDynamicStateCreateInfo {
    VkStructureType                     sType;
    const void*                         pNext;
    VkPipelineDynamicStateCreateFlags   flags;
    uint32_t                            dynamicStateCount;
    const VkDynamicState*               pDynamicStates;
} VkPipelineDynamicStateCreateInfo;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is reserved for future use.
- `dynamicStateCount` is the number of elements in the `pDynamicStates` array.
- `pDynamicStates` is a pointer to an array of VkDynamicState values specifying which pieces of pipeline state will use the values from dynamic state commands rather than from pipeline state creation information.

### Valid Usage

- VUID-VkPipelineDynamicStateCreateInfo-pDynamicStates-01442
  Each element of `pDynamicStates` **must** be unique

### Valid Usage (Implicit)

- VUID-VkPipelineDynamicStateCreateInfo-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PIPELINE_DYNAMIC_STATE_CREATE_INFO`
- VUID-VkPipelineDynamicStateCreateInfo-pNext-pNext
  `pNext` **must** be `NULL`
- VUID-VkPipelineDynamicStateCreateInfo-flags-zerobitmask
  `flags` **must** be `0`
- VUID-VkPipelineDynamicStateCreateInfo-pDynamicStates-parameter
  If `dynamicStateCount` is not `0`, `pDynamicStates` **must** be a valid pointer to an array of `dynamicStateCount` valid VkDynamicState values

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkPipelineDynamicStateCreateFlags;
```

`VkPipelineDynamicStateCreateFlags` is a bitmask type for setting a mask, but is currently reserved for future use.

The source of different pieces of dynamic state is specified by the VkPipelineDynamicStateCreateInfo::`pDynamicStates` property of the currently active pipeline, each

of whose elements **must** be one of the values:

```
// Provided by VK_VERSION_1_0
typedef enum VkDynamicState {
    VK_DYNAMIC_STATE_VIEWPORT = 0,
    VK_DYNAMIC_STATE_SCISSOR = 1,
    VK_DYNAMIC_STATE_LINE_WIDTH = 2,
    VK_DYNAMIC_STATE_DEPTH_BIAS = 3,
    VK_DYNAMIC_STATE_BLEND_CONSTANTS = 4,
    VK_DYNAMIC_STATE_DEPTH_BOUNDS = 5,
    VK_DYNAMIC_STATE_STENCIL_COMPARE_MASK = 6,
    VK_DYNAMIC_STATE_STENCIL_WRITE_MASK = 7,
    VK_DYNAMIC_STATE_STENCIL_REFERENCE = 8,
    VK_DYNAMIC_STATE_CULL_MODE = 1000267000,
    VK_DYNAMIC_STATE_FRONT_FACE = 1000267001,
    VK_DYNAMIC_STATE_PRIMITIVE_TOPOLOGY = 1000267002,
    VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT = 1000267003,
    VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT = 1000267004,
    VK_DYNAMIC_STATE_VERTEX_INPUT_BINDING_STRIDE = 1000267005,
    VK_DYNAMIC_STATE_DEPTH_TEST_ENABLE = 1000267006,
    VK_DYNAMIC_STATE_DEPTH_WRITE_ENABLE = 1000267007,
    VK_DYNAMIC_STATE_DEPTH_COMPARE_OP = 1000267008,
    VK_DYNAMIC_STATE_DEPTH_BOUNDS_TEST_ENABLE = 1000267009,
    VK_DYNAMIC_STATE_STENCIL_TEST_ENABLE = 1000267010,
    VK_DYNAMIC_STATE_STENCIL_OP = 1000267011,
    VK_DYNAMIC_STATE_RASTERIZER_DISCARD_ENABLE = 1000377001,
    VK_DYNAMIC_STATE_DEPTH_BIAS_ENABLE = 1000377002,
    VK_DYNAMIC_STATE_PRIMITIVE_RESTART_ENABLE = 1000377004,
  // Provided by VK_NV_clip_space_w_scaling
    VK_DYNAMIC_STATE_VIEWPORT_W_SCALING_NV = 1000087000,
  // Provided by VK_EXT_discard_rectangles
    VK_DYNAMIC_STATE_DISCARD_RECTANGLE_EXT = 1000099000,
  // Provided by VK_EXT_discard_rectangles
    VK_DYNAMIC_STATE_DISCARD_RECTANGLE_ENABLE_EXT = 1000099001,
  // Provided by VK_EXT_discard_rectangles
    VK_DYNAMIC_STATE_DISCARD_RECTANGLE_MODE_EXT = 1000099002,
  // Provided by VK_EXT_sample_locations
    VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT = 1000143000,
  // Provided by VK_KHR_ray_tracing_pipeline
    VK_DYNAMIC_STATE_RAY_TRACING_PIPELINE_STACK_SIZE_KHR = 1000347000,
  // Provided by VK_NV_shading_rate_image
    VK_DYNAMIC_STATE_VIEWPORT_SHADING_RATE_PALETTE_NV = 1000164004,
  // Provided by VK_NV_shading_rate_image
    VK_DYNAMIC_STATE_VIEWPORT_COARSE_SAMPLE_ORDER_NV = 1000164006,
  // Provided by VK_NV_scissor_exclusive
    VK_DYNAMIC_STATE_EXCLUSIVE_SCISSOR_ENABLE_NV = 1000205000,
  // Provided by VK_NV_scissor_exclusive
    VK_DYNAMIC_STATE_EXCLUSIVE_SCISSOR_NV = 1000205001,
  // Provided by VK_KHR_fragment_shading_rate
    VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR = 1000226000,
```

```
  // Provided by VK_EXT_vertex_input_dynamic_state
    VK_DYNAMIC_STATE_VERTEX_INPUT_EXT = 1000352000,
  // Provided by VK_EXT_extended_dynamic_state2
    VK_DYNAMIC_STATE_PATCH_CONTROL_POINTS_EXT = 1000377000,
  // Provided by VK_EXT_extended_dynamic_state2
    VK_DYNAMIC_STATE_LOGIC_OP_EXT = 1000377003,
  // Provided by VK_EXT_color_write_enable
    VK_DYNAMIC_STATE_COLOR_WRITE_ENABLE_EXT = 1000381000,
  // Provided by VK_EXT_extended_dynamic_state3
    VK_DYNAMIC_STATE_DEPTH_CLAMP_ENABLE_EXT = 1000455003,
  // Provided by VK_EXT_extended_dynamic_state3
    VK_DYNAMIC_STATE_POLYGON_MODE_EXT = 1000455004,
  // Provided by VK_EXT_extended_dynamic_state3
    VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT = 1000455005,
  // Provided by VK_EXT_extended_dynamic_state3
    VK_DYNAMIC_STATE_SAMPLE_MASK_EXT = 1000455006,
  // Provided by VK_EXT_extended_dynamic_state3
    VK_DYNAMIC_STATE_ALPHA_TO_COVERAGE_ENABLE_EXT = 1000455007,
  // Provided by VK_EXT_extended_dynamic_state3
    VK_DYNAMIC_STATE_ALPHA_TO_ONE_ENABLE_EXT = 1000455008,
  // Provided by VK_EXT_extended_dynamic_state3
    VK_DYNAMIC_STATE_LOGIC_OP_ENABLE_EXT = 1000455009,
  // Provided by VK_EXT_extended_dynamic_state3
    VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT = 1000455010,
  // Provided by VK_EXT_extended_dynamic_state3
    VK_DYNAMIC_STATE_COLOR_BLEND_EQUATION_EXT = 1000455011,
  // Provided by VK_EXT_extended_dynamic_state3
    VK_DYNAMIC_STATE_COLOR_WRITE_MASK_EXT = 1000455012,
  // Provided by VK_EXT_extended_dynamic_state3 with VK_KHR_maintenance2 or
VK_VERSION_1_1
    VK_DYNAMIC_STATE_TESSELLATION_DOMAIN_ORIGIN_EXT = 1000455002,
  // Provided by VK_EXT_extended_dynamic_state3 with VK_EXT_transform_feedback
    VK_DYNAMIC_STATE_RASTERIZATION_STREAM_EXT = 1000455013,
  // Provided by VK_EXT_conservative_rasterization with VK_EXT_extended_dynamic_state3
    VK_DYNAMIC_STATE_CONSERVATIVE_RASTERIZATION_MODE_EXT = 1000455014,
  // Provided by VK_EXT_conservative_rasterization with VK_EXT_extended_dynamic_state3
    VK_DYNAMIC_STATE_EXTRA_PRIMITIVE_OVERESTIMATION_SIZE_EXT = 1000455015,
  // Provided by VK_EXT_depth_clip_enable with VK_EXT_extended_dynamic_state3
    VK_DYNAMIC_STATE_DEPTH_CLIP_ENABLE_EXT = 1000455016,
  // Provided by VK_EXT_extended_dynamic_state3 with VK_EXT_sample_locations
    VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT = 1000455017,
  // Provided by VK_EXT_blend_operation_advanced with VK_EXT_extended_dynamic_state3
    VK_DYNAMIC_STATE_COLOR_BLEND_ADVANCED_EXT = 1000455018,
  // Provided by VK_EXT_extended_dynamic_state3 with VK_EXT_provoking_vertex
    VK_DYNAMIC_STATE_PROVOKING_VERTEX_MODE_EXT = 1000455019,
  // Provided by VK_EXT_extended_dynamic_state3 with VK_EXT_line_rasterization
    VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT = 1000455020,
  // Provided by VK_EXT_extended_dynamic_state3 with VK_EXT_line_rasterization
    VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT = 1000455021,
  // Provided by VK_EXT_depth_clip_control with VK_EXT_extended_dynamic_state3
    VK_DYNAMIC_STATE_DEPTH_CLIP_NEGATIVE_ONE_TO_ONE_EXT = 1000455022,
```

```
  // Provided by VK_EXT_extended_dynamic_state3 with VK_NV_clip_space_w_scaling
    VK_DYNAMIC_STATE_VIEWPORT_W_SCALING_ENABLE_NV = 1000455023,
  // Provided by VK_EXT_extended_dynamic_state3 with VK_NV_viewport_swizzle
    VK_DYNAMIC_STATE_VIEWPORT_SWIZZLE_NV = 1000455024,
  // Provided by VK_EXT_extended_dynamic_state3 with VK_NV_fragment_coverage_to_color
    VK_DYNAMIC_STATE_COVERAGE_TO_COLOR_ENABLE_NV = 1000455025,
  // Provided by VK_EXT_extended_dynamic_state3 with VK_NV_fragment_coverage_to_color
    VK_DYNAMIC_STATE_COVERAGE_TO_COLOR_LOCATION_NV = 1000455026,
  // Provided by VK_EXT_extended_dynamic_state3 with VK_NV_framebuffer_mixed_samples
    VK_DYNAMIC_STATE_COVERAGE_MODULATION_MODE_NV = 1000455027,
  // Provided by VK_EXT_extended_dynamic_state3 with VK_NV_framebuffer_mixed_samples
    VK_DYNAMIC_STATE_COVERAGE_MODULATION_TABLE_ENABLE_NV = 1000455028,
  // Provided by VK_EXT_extended_dynamic_state3 with VK_NV_framebuffer_mixed_samples
    VK_DYNAMIC_STATE_COVERAGE_MODULATION_TABLE_NV = 1000455029,
  // Provided by VK_EXT_extended_dynamic_state3 with VK_NV_shading_rate_image
    VK_DYNAMIC_STATE_SHADING_RATE_IMAGE_ENABLE_NV = 1000455030,
  // Provided by VK_EXT_extended_dynamic_state3 with
VK_NV_representative_fragment_test
    VK_DYNAMIC_STATE_REPRESENTATIVE_FRAGMENT_TEST_ENABLE_NV = 1000455031,
  // Provided by VK_EXT_extended_dynamic_state3 with VK_NV_coverage_reduction_mode
    VK_DYNAMIC_STATE_COVERAGE_REDUCTION_MODE_NV = 1000455032,
  // Provided by VK_EXT_attachment_feedback_loop_dynamic_state
    VK_DYNAMIC_STATE_ATTACHMENT_FEEDBACK_LOOP_ENABLE_EXT = 1000524000,
  // Provided by VK_KHR_line_rasterization
    VK_DYNAMIC_STATE_LINE_STIPPLE_KHR = 1000259000,
  // Provided by VK_EXT_line_rasterization
    VK_DYNAMIC_STATE_LINE_STIPPLE_EXT = VK_DYNAMIC_STATE_LINE_STIPPLE_KHR,
  // Provided by VK_EXT_extended_dynamic_state
    VK_DYNAMIC_STATE_CULL_MODE_EXT = VK_DYNAMIC_STATE_CULL_MODE,
  // Provided by VK_EXT_extended_dynamic_state
    VK_DYNAMIC_STATE_FRONT_FACE_EXT = VK_DYNAMIC_STATE_FRONT_FACE,
  // Provided by VK_EXT_extended_dynamic_state
    VK_DYNAMIC_STATE_PRIMITIVE_TOPOLOGY_EXT = VK_DYNAMIC_STATE_PRIMITIVE_TOPOLOGY,
  // Provided by VK_EXT_extended_dynamic_state
    VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT_EXT = VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT,
  // Provided by VK_EXT_extended_dynamic_state
    VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT_EXT = VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT,
  // Provided by VK_EXT_extended_dynamic_state
    VK_DYNAMIC_STATE_VERTEX_INPUT_BINDING_STRIDE_EXT =
VK_DYNAMIC_STATE_VERTEX_INPUT_BINDING_STRIDE,
  // Provided by VK_EXT_extended_dynamic_state
    VK_DYNAMIC_STATE_DEPTH_TEST_ENABLE_EXT = VK_DYNAMIC_STATE_DEPTH_TEST_ENABLE,
  // Provided by VK_EXT_extended_dynamic_state
    VK_DYNAMIC_STATE_DEPTH_WRITE_ENABLE_EXT = VK_DYNAMIC_STATE_DEPTH_WRITE_ENABLE,
  // Provided by VK_EXT_extended_dynamic_state
    VK_DYNAMIC_STATE_DEPTH_COMPARE_OP_EXT = VK_DYNAMIC_STATE_DEPTH_COMPARE_OP,
  // Provided by VK_EXT_extended_dynamic_state
    VK_DYNAMIC_STATE_DEPTH_BOUNDS_TEST_ENABLE_EXT =
VK_DYNAMIC_STATE_DEPTH_BOUNDS_TEST_ENABLE,
  // Provided by VK_EXT_extended_dynamic_state
    VK_DYNAMIC_STATE_STENCIL_TEST_ENABLE_EXT = VK_DYNAMIC_STATE_STENCIL_TEST_ENABLE,
```

```
  // Provided by VK_EXT_extended_dynamic_state
    VK_DYNAMIC_STATE_STENCIL_OP_EXT = VK_DYNAMIC_STATE_STENCIL_OP,
  // Provided by VK_EXT_extended_dynamic_state2
    VK_DYNAMIC_STATE_RASTERIZER_DISCARD_ENABLE_EXT =
VK_DYNAMIC_STATE_RASTERIZER_DISCARD_ENABLE,
  // Provided by VK_EXT_extended_dynamic_state2
    VK_DYNAMIC_STATE_DEPTH_BIAS_ENABLE_EXT = VK_DYNAMIC_STATE_DEPTH_BIAS_ENABLE,
  // Provided by VK_EXT_extended_dynamic_state2
    VK_DYNAMIC_STATE_PRIMITIVE_RESTART_ENABLE_EXT =
VK_DYNAMIC_STATE_PRIMITIVE_RESTART_ENABLE,
} VkDynamicState;
```

- `VK_DYNAMIC_STATE_VIEWPORT` specifies that the `pViewports` state in VkPipelineViewportStateCreateInfo will be ignored and **must** be set dynamically with vkCmdSetViewport before any drawing commands. The number of viewports used by a pipeline is still specified by the `viewportCount` member of VkPipelineViewportStateCreateInfo.
- `VK_DYNAMIC_STATE_SCISSOR` specifies that the `pScissors` state in VkPipelineViewportStateCreateInfo will be ignored and **must** be set dynamically with vkCmdSetScissor before any drawing commands. The number of scissor rectangles used by a pipeline is still specified by the `scissorCount` member of VkPipelineViewportStateCreateInfo.
- `VK_DYNAMIC_STATE_LINE_WIDTH` specifies that the `lineWidth` state in VkPipelineRasterizationStateCreateInfo will be ignored and **must** be set dynamically with vkCmdSetLineWidth before any drawing commands that generate line primitives for the rasterizer.
- `VK_DYNAMIC_STATE_DEPTH_BIAS` specifies that any instance of VkDepthBiasRepresentationInfoEXT included in the `pNext` chain of VkPipelineRasterizationStateCreateInfo as well as the `depthBiasConstantFactor`, `depthBiasClamp` and `depthBiasSlopeFactor` states in VkPipelineRasterizationStateCreateInfo will be ignored and **must** be set dynamically with vkCmdSetDepthBias or vkCmdSetDepthBias2EXT before any draws are performed with depth bias enabled.
- `VK_DYNAMIC_STATE_BLEND_CONSTANTS` specifies that the `blendConstants` state in VkPipelineColorBlendStateCreateInfo will be ignored and **must** be set dynamically with vkCmdSetBlendConstants before any draws are performed with a pipeline state with `VkPipelineColorBlendAttachmentState` member `blendEnable` set to `VK_TRUE` and any of the blend functions using a constant blend color.
- `VK_DYNAMIC_STATE_DEPTH_BOUNDS` specifies that the `minDepthBounds` and `maxDepthBounds` states of VkPipelineDepthStencilStateCreateInfo will be ignored and **must** be set dynamically with vkCmdSetDepthBounds before any draws are performed with a pipeline state with VkPipelineDepthStencilStateCreateInfo member `depthBoundsTestEnable` set to `VK_TRUE`.
- `VK_DYNAMIC_STATE_STENCIL_COMPARE_MASK` specifies that the `compareMask` state in VkPipelineDepthStencilStateCreateInfo for both `front` and `back` will be ignored and **must** be set dynamically with vkCmdSetStencilCompareMask before any draws are performed with a pipeline state with VkPipelineDepthStencilStateCreateInfo member `stencilTestEnable` set to `VK_TRUE`
- `VK_DYNAMIC_STATE_STENCIL_WRITE_MASK` specifies that the `writeMask` state in

VkPipelineDepthStencilStateCreateInfo for both front and back will be ignored and **must** be set dynamically with vkCmdSetStencilWriteMask before any draws are performed with a pipeline state with VkPipelineDepthStencilStateCreateInfo member stencilTestEnable set to VK_TRUE

- VK_DYNAMIC_STATE_STENCIL_REFERENCE specifies that the reference state in VkPipelineDepthStencilStateCreateInfo for both front and back will be ignored and **must** be set dynamically with vkCmdSetStencilReference before any draws are performed with a pipeline state with VkPipelineDepthStencilStateCreateInfo member stencilTestEnable set to VK_TRUE
- VK_DYNAMIC_STATE_VIEWPORT_W_SCALING_NV specifies that the pViewportWScalings state in VkPipelineViewportWScalingStateCreateInfoNV will be ignored and **must** be set dynamically with vkCmdSetViewportWScalingNV before any draws are performed with a pipeline state with VkPipelineViewportWScalingStateCreateInfoNV member viewportScalingEnable set to VK_TRUE
- VK_DYNAMIC_STATE_DISCARD_RECTANGLE_EXT specifies that the pDiscardRectangles state in VkPipelineDiscardRectangleStateCreateInfoEXT will be ignored and **must** be set dynamically with vkCmdSetDiscardRectangleEXT before any draw or clear commands.
- VK_DYNAMIC_STATE_DISCARD_RECTANGLE_ENABLE_EXT specifies that the presence of the VkPipelineDiscardRectangleStateCreateInfoEXT structure in the VkGraphicsPipelineCreateInfo chain with a discardRectangleCount greater than zero does not implicitly enable discard rectangles and they **must** be enabled dynamically with vkCmdSetDiscardRectangleEnableEXT before any draw commands. This is available on implementations that support at least specVersion 2 of the VK_EXT_discard_rectangles extension.
- VK_DYNAMIC_STATE_DISCARD_RECTANGLE_MODE_EXT specifies that the discardRectangleMode state in VkPipelineDiscardRectangleStateCreateInfoEXT will be ignored and **must** be set dynamically with vkCmdSetDiscardRectangleModeEXT before any draw commands. This is available on implementations that support at least specVersion 2 of the VK_EXT_discard_rectangles extension.
- VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT specifies that the sampleLocationsInfo state in VkPipelineSampleLocationsStateCreateInfoEXT will be ignored and **must** be set dynamically with vkCmdSetSampleLocationsEXT before any draw or clear commands. Enabling custom sample locations is still indicated by the sampleLocationsEnable member of VkPipelineSampleLocationsStateCreateInfoEXT.
- VK_DYNAMIC_STATE_EXCLUSIVE_SCISSOR_NV specifies that the pExclusiveScissors state in VkPipelineViewportExclusiveScissorStateCreateInfoNV will be ignored and **must** be set dynamically with vkCmdSetExclusiveScissorNV before any drawing commands.
- VK_DYNAMIC_STATE_EXCLUSIVE_SCISSOR_ENABLE_NV specifies that the exclusive scissors **must** be explicitly enabled with vkCmdSetExclusiveScissorEnableNV and the exclusiveScissorCount value in VkPipelineViewportExclusiveScissorStateCreateInfoNV will not implicitly enable them. This is available on implementations that support at least specVersion 2 of the VK_NV_scissor_exclusive extension.
- VK_DYNAMIC_STATE_VIEWPORT_SHADING_RATE_PALETTE_NV specifies that the pShadingRatePalettes state in VkPipelineViewportShadingRateImageStateCreateInfoNV will be ignored and **must** be set dynamically with vkCmdSetViewportShadingRatePaletteNV before any drawing commands.
- VK_DYNAMIC_STATE_VIEWPORT_COARSE_SAMPLE_ORDER_NV specifies that the coarse sample order state in VkPipelineViewportCoarseSampleOrderStateCreateInfoNV will be ignored and **must** be set dynamically with vkCmdSetCoarseSampleOrderNV before any drawing commands.

- VK_DYNAMIC_STATE_LINE_STIPPLE_EXT specifies that the lineStippleFactor and lineStipplePattern state in VkPipelineRasterizationLineStateCreateInfoKHR will be ignored and **must** be set dynamically with vkCmdSetLineStippleKHR before any draws are performed with a pipeline state with VkPipelineRasterizationLineStateCreateInfoKHR member stippledLineEnable set to VK_TRUE.
- VK_DYNAMIC_STATE_CULL_MODE specifies that the cullMode state in VkPipelineRasterizationStateCreateInfo will be ignored and **must** be set dynamically with vkCmdSetCullModeEXT before any drawing commands.
- VK_DYNAMIC_STATE_FRONT_FACE specifies that the frontFace state in VkPipelineRasterizationStateCreateInfo will be ignored and **must** be set dynamically with vkCmdSetFrontFaceEXT before any drawing commands.
- VK_DYNAMIC_STATE_PRIMITIVE_TOPOLOGY specifies that the topology state in VkPipelineInputAssemblyStateCreateInfo only specifies the topology class, and the specific topology order and adjacency **must** be set dynamically with vkCmdSetPrimitiveTopologyEXT before any drawing commands.
- VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT specifies that the viewportCount and pViewports state in VkPipelineViewportStateCreateInfo will be ignored and **must** be set dynamically with vkCmdSetViewportWithCountEXT before any draw call.
- VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT specifies that the scissorCount and pScissors state in VkPipelineViewportStateCreateInfo will be ignored and **must** be set dynamically with vkCmdSetScissorWithCountEXT before any draw call.
- VK_DYNAMIC_STATE_VERTEX_INPUT_BINDING_STRIDE specifies that the stride state in VkVertexInputBindingDescription will be ignored and **must** be set dynamically with vkCmdBindVertexBuffers2EXT before any draw call.
- VK_DYNAMIC_STATE_DEPTH_TEST_ENABLE specifies that the depthTestEnable state in VkPipelineDepthStencilStateCreateInfo will be ignored and **must** be set dynamically with vkCmdSetDepthTestEnableEXT before any draw call.
- VK_DYNAMIC_STATE_DEPTH_WRITE_ENABLE specifies that the depthWriteEnable state in VkPipelineDepthStencilStateCreateInfo will be ignored and **must** be set dynamically with vkCmdSetDepthWriteEnableEXT before any draw call.
- VK_DYNAMIC_STATE_DEPTH_COMPARE_OP specifies that the depthCompareOp state in VkPipelineDepthStencilStateCreateInfo will be ignored and **must** be set dynamically with vkCmdSetDepthCompareOpEXT before any draw call.
- VK_DYNAMIC_STATE_DEPTH_BOUNDS_TEST_ENABLE specifies that the depthBoundsTestEnable state in VkPipelineDepthStencilStateCreateInfo will be ignored and **must** be set dynamically with vkCmdSetDepthBoundsTestEnableEXT before any draw call.
- VK_DYNAMIC_STATE_STENCIL_TEST_ENABLE specifies that the stencilTestEnable state in VkPipelineDepthStencilStateCreateInfo will be ignored and **must** be set dynamically with vkCmdSetStencilTestEnableEXT before any draw call.
- VK_DYNAMIC_STATE_STENCIL_OP specifies that the failOp, passOp, depthFailOp, and compareOp states in VkPipelineDepthStencilStateCreateInfo for both front and back will be ignored and **must** be set dynamically with vkCmdSetStencilOpEXT before any draws are performed with a pipeline state with VkPipelineDepthStencilStateCreateInfo member stencilTestEnable set to VK_TRUE

- VK_DYNAMIC_STATE_PATCH_CONTROL_POINTS_EXT specifies that the patchControlPoints state in VkPipelineTessellationStateCreateInfo will be ignored and **must** be set dynamically with vkCmdSetPatchControlPointsEXT before any drawing commands.
- VK_DYNAMIC_STATE_RASTERIZER_DISCARD_ENABLE specifies that the rasterizerDiscardEnable state in VkPipelineRasterizationStateCreateInfo will be ignored and **must** be set dynamically with vkCmdSetRasterizerDiscardEnableEXT before any drawing commands.
- VK_DYNAMIC_STATE_DEPTH_BIAS_ENABLE specifies that the depthBiasEnable state in VkPipelineRasterizationStateCreateInfo will be ignored and **must** be set dynamically with vkCmdSetDepthBiasEnableEXT before any drawing commands.
- VK_DYNAMIC_STATE_LOGIC_OP_EXT specifies that the logicOp state in VkPipelineColorBlendStateCreateInfo will be ignored and **must** be set dynamically with vkCmdSetLogicOpEXT before any drawing commands.
- VK_DYNAMIC_STATE_PRIMITIVE_RESTART_ENABLE specifies that the primitiveRestartEnable state in VkPipelineInputAssemblyStateCreateInfo will be ignored and **must** be set dynamically with vkCmdSetPrimitiveRestartEnableEXT before any drawing commands.
- VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR specifies that state in VkPipelineFragmentShadingRateStateCreateInfoKHR and VkPipelineFragmentShadingRateEnumStateCreateInfoNV will be ignored and **must** be set dynamically with vkCmdSetFragmentShadingRateKHR or vkCmdSetFragmentShadingRateEnumNV before any drawing commands.
- VK_DYNAMIC_STATE_RAY_TRACING_PIPELINE_STACK_SIZE_KHR specifies that the default stack size computation for the pipeline will be ignored and **must** be set dynamically with vkCmdSetRayTracingPipelineStackSizeKHR before any ray tracing calls are performed.
- VK_DYNAMIC_STATE_VERTEX_INPUT_EXT specifies that the pVertexInputState state will be ignored and **must** be set dynamically with vkCmdSetVertexInputEXT before any drawing commands
- VK_DYNAMIC_STATE_COLOR_WRITE_ENABLE_EXT specifies that the pColorWriteEnables state in VkPipelineColorWriteCreateInfoEXT will be ignored and **must** be set dynamically with vkCmdSetColorWriteEnableEXT before any draw call.
- VK_DYNAMIC_STATE_TESSELLATION_DOMAIN_ORIGIN_EXT specifies that the domainOrigin state in VkPipelineTessellationDomainOriginStateCreateInfo will be ignored and **must** be set dynamically with vkCmdSetTessellationDomainOriginEXT before any draw call.
- VK_DYNAMIC_STATE_DEPTH_CLAMP_ENABLE_EXT specifies that the depthClampEnable state in VkPipelineRasterizationStateCreateInfo will be ignored and **must** be set dynamically with vkCmdSetDepthClampEnableEXT before any draw call.
- VK_DYNAMIC_STATE_POLYGON_MODE_EXT specifies that the polygonMode state in VkPipelineRasterizationStateCreateInfo will be ignored and **must** be set dynamically with vkCmdSetPolygonModeEXT before any draw call.
- VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT specifies that the rasterizationSamples state in VkPipelineMultisampleStateCreateInfo will be ignored and **must** be set dynamically with vkCmdSetRasterizationSamplesEXT before any draw call.
- VK_DYNAMIC_STATE_SAMPLE_MASK_EXT specifies that the pSampleMask state in VkPipelineMultisampleStateCreateInfo will be ignored and **must** be set dynamically with vkCmdSetSampleMaskEXT before any draw call.

- VK_DYNAMIC_STATE_ALPHA_TO_COVERAGE_ENABLE_EXT specifies that the alphaToCoverageEnable state in VkPipelineMultisampleStateCreateInfo will be ignored and **must** be set dynamically with vkCmdSetAlphaToCoverageEnableEXT before any draw call.
- VK_DYNAMIC_STATE_ALPHA_TO_ONE_ENABLE_EXT specifies that the alphaToOneEnable state in VkPipelineMultisampleStateCreateInfo will be ignored and **must** be set dynamically with vkCmdSetAlphaToOneEnableEXT before any draw call.
- VK_DYNAMIC_STATE_LOGIC_OP_ENABLE_EXT specifies that the logicOpEnable state in VkPipelineColorBlendStateCreateInfo will be ignored and **must** be set dynamically with vkCmdSetLogicOpEnableEXT before any draw call.
- VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT specifies that the blendEnable state in VkPipelineColorBlendAttachmentState will be ignored and **must** be set dynamically with vkCmdSetColorBlendEnableEXT before any draw call.
- VK_DYNAMIC_STATE_COLOR_BLEND_EQUATION_EXT specifies that the srcColorBlendFactor, dstColorBlendFactor, colorBlendOp, srcAlphaBlendFactor, dstAlphaBlendFactor, and alphaBlendOp states in VkPipelineColorBlendAttachmentState will be ignored and **must** be set dynamically with vkCmdSetColorBlendEquationEXT before any draw call.
- VK_DYNAMIC_STATE_COLOR_WRITE_MASK_EXT specifies that the colorWriteMask state in VkPipelineColorBlendAttachmentState will be ignored and **must** be set dynamically with vkCmdSetColorWriteMaskEXT before any draw call.
- VK_DYNAMIC_STATE_RASTERIZATION_STREAM_EXT specifies that the rasterizationStream state in VkPipelineRasterizationStateStreamCreateInfoEXT will be ignored and **must** be set dynamically with vkCmdSetRasterizationStreamEXT before any draw call.
- VK_DYNAMIC_STATE_CONSERVATIVE_RASTERIZATION_MODE_EXT specifies that the conservativeRasterizationMode state in VkPipelineRasterizationConservativeStateCreateInfoEXT will be ignored and **must** be set dynamically with vkCmdSetConservativeRasterizationModeEXT before any draw call.
- VK_DYNAMIC_STATE_EXTRA_PRIMITIVE_OVERESTIMATION_SIZE_EXT specifies that the extraPrimitiveOverestimationSize state in VkPipelineRasterizationConservativeStateCreateInfoEXT will be ignored and **must** be set dynamically with vkCmdSetExtraPrimitiveOverestimationSizeEXT before any draw call.
- VK_DYNAMIC_STATE_DEPTH_CLIP_ENABLE_EXT specifies that the depthClipEnable state in VkPipelineRasterizationDepthClipStateCreateInfoEXT will be ignored and **must** be set dynamically with vkCmdSetDepthClipEnableEXT before any draw call.
- VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT specifies that the sampleLocationsEnable state in VkPipelineSampleLocationsStateCreateInfoEXT will be ignored and **must** be set dynamically with vkCmdSetSampleLocationsEnableEXT before any draw call.
- VK_DYNAMIC_STATE_COLOR_BLEND_ADVANCED_EXT specifies that the colorBlendOp state in VkPipelineColorBlendAttachmentState, and srcPremultiplied, dstPremultiplied, and blendOverlap states in VkPipelineColorBlendAdvancedStateCreateInfoEXT will be ignored and **must** be set dynamically with vkCmdSetColorBlendAdvancedEXT before any draw call.
- VK_DYNAMIC_STATE_PROVOKING_VERTEX_MODE_EXT specifies that the provokingVertexMode state in VkPipelineRasterizationProvokingVertexStateCreateInfoEXT will be ignored and **must** be set dynamically with vkCmdSetProvokingVertexModeEXT before any draw call.

- VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT specifies that the lineRasterizationMode state in VkPipelineRasterizationLineStateCreateInfoKHR will be ignored and **must** be set dynamically with vkCmdSetLineRasterizationModeEXT before any draw call.
- VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT specifies that the stippledLineEnable state in VkPipelineRasterizationLineStateCreateInfoKHR will be ignored and **must** be set dynamically with vkCmdSetLineStippleEnableEXT before any draw call.
- VK_DYNAMIC_STATE_DEPTH_CLIP_NEGATIVE_ONE_TO_ONE_EXT specifies that the negativeOneToOne state in VkPipelineViewportDepthClipControlCreateInfoEXT will be ignored and **must** be set dynamically with vkCmdSetDepthClipNegativeOneToOneEXT before any draw call.
- VK_DYNAMIC_STATE_VIEWPORT_W_SCALING_ENABLE_NV specifies that the viewportWScalingEnable state in VkPipelineViewportWScalingStateCreateInfoNV will be ignored and **must** be set dynamically with vkCmdSetViewportWScalingEnableNV before any draw call.
- VK_DYNAMIC_STATE_VIEWPORT_SWIZZLE_NV specifies that the viewportCount, and pViewportSwizzles states in VkPipelineViewportSwizzleStateCreateInfoNV will be ignored and **must** be set dynamically with vkCmdSetViewportSwizzleNV before any draw call.
- VK_DYNAMIC_STATE_COVERAGE_TO_COLOR_ENABLE_NV specifies that the coverageToColorEnable state in VkPipelineCoverageToColorStateCreateInfoNV will be ignored and **must** be set dynamically with vkCmdSetCoverageToColorEnableNV before any draw call.
- VK_DYNAMIC_STATE_COVERAGE_TO_COLOR_LOCATION_NV specifies that the coverageToColorLocation state in VkPipelineCoverageToColorStateCreateInfoNV will be ignored and **must** be set dynamically with vkCmdSetCoverageToColorLocationNV before any draw call.
- VK_DYNAMIC_STATE_COVERAGE_MODULATION_MODE_NV specifies that the coverageModulationMode state in VkPipelineCoverageModulationStateCreateInfoNV will be ignored and **must** be set dynamically with vkCmdSetCoverageModulationModeNV before any draw call.
- VK_DYNAMIC_STATE_COVERAGE_MODULATION_TABLE_ENABLE_NV specifies that the coverageModulationTableEnable state in VkPipelineCoverageModulationStateCreateInfoNV will be ignored and **must** be set dynamically with vkCmdSetCoverageModulationTableEnableNV before any draw call.
- VK_DYNAMIC_STATE_COVERAGE_MODULATION_TABLE_NV specifies that the coverageModulationTableCount, and pCoverageModulationTable states in VkPipelineCoverageModulationStateCreateInfoNV will be ignored and **must** be set dynamically with vkCmdSetCoverageModulationTableNV before any draw call.
- VK_DYNAMIC_STATE_SHADING_RATE_IMAGE_ENABLE_NV specifies that the shadingRateImageEnable state in VkPipelineViewportShadingRateImageStateCreateInfoNV will be ignored and **must** be set dynamically with vkCmdSetShadingRateImageEnableNV before any draw call.
- VK_DYNAMIC_STATE_REPRESENTATIVE_FRAGMENT_TEST_ENABLE_NV specifies that the representativeFragmentTestEnable state in VkPipelineRepresentativeFragmentTestStateCreateInfoNV will be ignored and **must** be set dynamically with vkCmdSetRepresentativeFragmentTestEnableNV before any draw call.
- VK_DYNAMIC_STATE_COVERAGE_REDUCTION_MODE_NV specifies that the coverageReductionMode state in VkPipelineCoverageReductionStateCreateInfoNV will be ignored and **must** be set dynamically with vkCmdSetCoverageReductionModeNV before any draw call.
- VK_DYNAMIC_STATE_ATTACHMENT_FEEDBACK_LOOP_ENABLE_EXT specifies that the

VK_PIPELINE_CREATE_COLOR_ATTACHMENT_FEEDBACK_LOOP_BIT_EXT and VK_PIPELINE_CREATE_DEPTH_STENCIL_ATTACHMENT_FEEDBACK_LOOP_BIT_EXT flags will be ignored and **must** be set dynamically with vkCmdSetAttachmentFeedbackLoopEnableEXT before any draw call.

### 10.3.1. Valid Combinations of Stages for Graphics Pipelines

Primitive processing can be handled either on a per primitive basis by the vertex, tessellation, and geometry shader stages, or on a per mesh basis using task and mesh shader stages. If the pipeline includes a mesh shader stage, it uses the mesh pipeline, otherwise it uses the primitive pipeline.

If a task shader is omitted, the task shading stage is skipped.

If tessellation shader stages are omitted, the tessellation shading and fixed-function stages of the pipeline are skipped.

If a geometry shader is omitted, the geometry shading stage is skipped.

If a fragment shader is omitted, fragment color outputs have undefined values, and the fragment depth value is determined by Fragment Operations state. This **can** be useful for depth-only rendering.

Presence of a shader stage in a pipeline is indicated by including a valid VkPipelineShaderStageCreateInfo with `module` and `pName` selecting an entry point from a shader module, where that entry point is valid for the stage specified by `stage`.

Presence of some of the fixed-function stages in the pipeline is implicitly derived from enabled shaders and provided state. For example, the fixed-function tessellator is always present when the pipeline has valid Tessellation Control and Tessellation Evaluation shaders.

*For example:*

- Depth/stencil-only rendering in a subpass with no color attachments
  - Active Pipeline Shader Stages
    - Vertex Shader
  - Required: Fixed-Function Pipeline Stages
    - VkPipelineVertexInputStateCreateInfo
    - VkPipelineInputAssemblyStateCreateInfo
    - VkPipelineViewportStateCreateInfo
    - VkPipelineRasterizationStateCreateInfo
    - VkPipelineMultisampleStateCreateInfo
    - VkPipelineDepthStencilStateCreateInfo
- Color-only rendering in a subpass with no depth/stencil attachment
  - Active Pipeline Shader Stages
    - Vertex Shader

        - Fragment Shader
    - Required: Fixed-Function Pipeline Stages
        - VkPipelineVertexInputStateCreateInfo
        - VkPipelineInputAssemblyStateCreateInfo
        - VkPipelineViewportStateCreateInfo
        - VkPipelineRasterizationStateCreateInfo
        - VkPipelineMultisampleStateCreateInfo
        - VkPipelineColorBlendStateCreateInfo
- Rendering pipeline with tessellation and geometry shaders
    - Active Pipeline Shader Stages
        - Vertex Shader
        - Tessellation Control Shader
        - Tessellation Evaluation Shader
        - Geometry Shader
        - Fragment Shader
    - Required: Fixed-Function Pipeline Stages
        - VkPipelineVertexInputStateCreateInfo
        - VkPipelineInputAssemblyStateCreateInfo
        - VkPipelineTessellationStateCreateInfo
        - VkPipelineViewportStateCreateInfo
        - VkPipelineRasterizationStateCreateInfo
        - VkPipelineMultisampleStateCreateInfo
        - VkPipelineDepthStencilStateCreateInfo
        - VkPipelineColorBlendStateCreateInfo
- Rendering pipeline with task and mesh shaders
    - Active Pipeline Shader Stages
        - Task Shader
        - Mesh Shader
        - Fragment Shader
    - Required: Fixed-Function Pipeline Stages
        - VkPipelineViewportStateCreateInfo
        - VkPipelineRasterizationStateCreateInfo
        - VkPipelineMultisampleStateCreateInfo
        - VkPipelineDepthStencilStateCreateInfo
        - VkPipelineColorBlendStateCreateInfo

### 10.3.2. Graphics Pipeline Shader Groups

Graphics pipelines can contain multiple shader groups that can be bound individually. Each shader group behaves as if it was a pipeline using the shader group's state. When the pipeline is bound by regular means, it behaves as if the state of group `0` is active, use vkCmdBindPipelineShaderGroupNV to bind an individual shader group.

The primary purpose of shader groups is allowing the device to bind different pipeline state using Device-Generated Commands.

The `VkGraphicsPipelineShaderGroupsCreateInfoNV` structure is defined as:

```
// Provided by VK_NV_device_generated_commands
typedef struct VkGraphicsPipelineShaderGroupsCreateInfoNV {
    VkStructureType                             sType;
    const void*                                 pNext;
    uint32_t                                    groupCount;
    const VkGraphicsShaderGroupCreateInfoNV*    pGroups;
    uint32_t                                    pipelineCount;
    const VkPipeline*                           pPipelines;
} VkGraphicsPipelineShaderGroupsCreateInfoNV;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `groupCount` is the number of elements in the `pGroups` array.
- `pGroups` is a pointer to an array of VkGraphicsShaderGroupCreateInfoNV structures specifying which state of the original VkGraphicsPipelineCreateInfo each shader group overrides.
- `pipelineCount` is the number of elements in the `pPipelines` array.
- `pPipelines` is a pointer to an array of graphics `VkPipeline` structures which are referenced within the created pipeline, including all their shader groups.

When referencing shader groups by index, groups defined in the referenced pipelines are treated as if they were defined as additional entries in `pGroups`. They are appended in the order they appear in the `pPipelines` array and in the `pGroups` array when those pipelines were defined.

The application **must** maintain the lifetime of all such referenced pipelines based on the pipelines that make use of them.

#### Valid Usage

- VUID-VkGraphicsPipelineShaderGroupsCreateInfoNV-groupCount-02879
  `groupCount` **must** be at least `1` and as maximum `VkPhysicalDeviceDeviceGeneratedCommandsPropertiesNV::maxGraphicsShaderGroupCount`
- VUID-VkGraphicsPipelineShaderGroupsCreateInfoNV-groupCount-02880
  The sum of `groupCount` including those groups added from referenced `pPipelines` **must** also be as maximum `VkPhysicalDeviceDeviceGeneratedCommandsPropertiesNV`

::maxGraphicsShaderGroupCount

- VUID-VkGraphicsPipelineShaderGroupsCreateInfoNV-pGroups-02881
  The state of the first element of pGroups **must** match its equivalent within the parent's VkGraphicsPipelineCreateInfo
- VUID-VkGraphicsPipelineShaderGroupsCreateInfoNV-pGroups-02882
  Each element of pGroups **must** in combination with the rest of the pipeline state yield a valid state configuration
- VUID-VkGraphicsPipelineShaderGroupsCreateInfoNV-pGroups-02884
  All elements of pGroups **must** use the same shader stage combinations unless any mesh shader stage is used, then either combination of task and mesh or just mesh shader is valid
- VUID-VkGraphicsPipelineShaderGroupsCreateInfoNV-pGroups-02885
  Mesh and regular primitive shading stages cannot be mixed across pGroups
- VUID-VkGraphicsPipelineShaderGroupsCreateInfoNV-pPipelines-02886
  Each element of pPipelines **must** have been created with identical state to the pipeline currently created except the state that can be overridden by VkGraphicsShaderGroupCreateInfoNV
- VUID-VkGraphicsPipelineShaderGroupsCreateInfoNV-deviceGeneratedCommands-02887
  The deviceGeneratedCommands feature **must** be enabled

## Valid Usage (Implicit)

- VUID-VkGraphicsPipelineShaderGroupsCreateInfoNV-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_GRAPHICS_PIPELINE_SHADER_GROUPS_CREATE_INFO_NV
- VUID-VkGraphicsPipelineShaderGroupsCreateInfoNV-pGroups-parameter
  If groupCount is not 0, pGroups **must** be a valid pointer to an array of groupCount valid VkGraphicsShaderGroupCreateInfoNV structures
- VUID-VkGraphicsPipelineShaderGroupsCreateInfoNV-pPipelines-parameter
  If pipelineCount is not 0, pPipelines **must** be a valid pointer to an array of pipelineCount valid VkPipeline handles

The VkGraphicsShaderGroupCreateInfoNV structure provides the state overrides for each shader group. Each shader group behaves like a pipeline that was created from its state as well as the remaining parent's state. It is defined as:

```
// Provided by VK_NV_device_generated_commands
typedef struct VkGraphicsShaderGroupCreateInfoNV {
    VkStructureType                                 sType;
    const void*                                     pNext;
    uint32_t                                        stageCount;
    const VkPipelineShaderStageCreateInfo*          pStages;
    const VkPipelineVertexInputStateCreateInfo*     pVertexInputState;
    const VkPipelineTessellationStateCreateInfo*    pTessellationState;
} VkGraphicsShaderGroupCreateInfoNV;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `stageCount` is the number of entries in the `pStages` array.
- `pStages` is a pointer to an array VkPipelineShaderStageCreateInfo structures specifying the set of the shader stages to be included in this shader group.
- `pVertexInputState` is a pointer to a VkPipelineVertexInputStateCreateInfo structure.
- `pTessellationState` is a pointer to a VkPipelineTessellationStateCreateInfo structure, and is ignored if the shader group does not include a tessellation control shader stage and tessellation evaluation shader stage.

## Valid Usage

- VUID-VkGraphicsShaderGroupCreateInfoNV-stageCount-02888
  For `stageCount`, the same restrictions as in VkGraphicsPipelineCreateInfo::`stageCount` apply
- VUID-VkGraphicsShaderGroupCreateInfoNV-pStages-02889
  For `pStages`, the same restrictions as in VkGraphicsPipelineCreateInfo::`pStages` apply
- VUID-VkGraphicsShaderGroupCreateInfoNV-pVertexInputState-02890
  For `pVertexInputState`, the same restrictions as in VkGraphicsPipelineCreateInfo::`pVertexInputState` apply
- VUID-VkGraphicsShaderGroupCreateInfoNV-pTessellationState-02891
  For `pTessellationState`, the same restrictions as in VkGraphicsPipelineCreateInfo::`pTessellationState` apply

## Valid Usage (Implicit)

- VUID-VkGraphicsShaderGroupCreateInfoNV-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_GRAPHICS_SHADER_GROUP_CREATE_INFO_NV`
- VUID-VkGraphicsShaderGroupCreateInfoNV-pNext-pNext
  `pNext` **must** be `NULL`
- VUID-VkGraphicsShaderGroupCreateInfoNV-pStages-parameter
  `pStages` **must** be a valid pointer to an array of `stageCount` valid

VkPipelineShaderStageCreateInfo structures

- VUID-VkGraphicsShaderGroupCreateInfoNV-stageCount-arraylength
  stageCount **must** be greater than 0

## 10.4. Ray Tracing Pipelines

Ray tracing pipelines consist of multiple shader stages, fixed-function traversal stages, and a pipeline layout.

VK_SHADER_UNUSED_KHR is a special shader index used to indicate that a ray generation, miss, or callable shader member is not used.

```
#define VK_SHADER_UNUSED_KHR              (~0U)
```

or the equivalent

```
#define VK_SHADER_UNUSED_NV               VK_SHADER_UNUSED_KHR
```

To create ray tracing pipelines, call:

```
// Provided by VK_NV_ray_tracing
VkResult vkCreateRayTracingPipelinesNV(
    VkDevice                                    device,
    VkPipelineCache                             pipelineCache,
    uint32_t                                    createInfoCount,
    const VkRayTracingPipelineCreateInfoNV*     pCreateInfos,
    const VkAllocationCallbacks*                pAllocator,
    VkPipeline*                                 pPipelines);
```

- device is the logical device that creates the ray tracing pipelines.
- pipelineCache is either VK_NULL_HANDLE, indicating that pipeline caching is disabled, or the handle of a valid pipeline cache object, in which case use of that cache is enabled for the duration of the command.
- createInfoCount is the length of the pCreateInfos and pPipelines arrays.
- pCreateInfos is a pointer to an array of VkRayTracingPipelineCreateInfoNV structures.
- pAllocator controls host memory allocation as described in the Memory Allocation chapter.
- pPipelines is a pointer to an array in which the resulting ray tracing pipeline objects are returned.

Pipelines are created and returned as described for Multiple Pipeline Creation.

## Valid Usage

- VUID-vkCreateRayTracingPipelinesNV-device-09677
  `device` **must** support at least one queue family with the `VK_QUEUE_COMPUTE_BIT` capability
- VUID-vkCreateRayTracingPipelinesNV-flags-03415
  If the `flags` member of any element of `pCreateInfos` contains the `VK_PIPELINE_CREATE_DERIVATIVE_BIT` flag, and the `basePipelineIndex` member of that same element is not `-1`, `basePipelineIndex` **must** be less than the index into `pCreateInfos` that corresponds to that element
- VUID-vkCreateRayTracingPipelinesNV-flags-03416
  If the `flags` member of any element of `pCreateInfos` contains the `VK_PIPELINE_CREATE_DERIVATIVE_BIT` flag, the base pipeline **must** have been created with the `VK_PIPELINE_CREATE_ALLOW_DERIVATIVES_BIT` flag set
- VUID-vkCreateRayTracingPipelinesNV-flags-03816
  `flags` **must** not contain the `VK_PIPELINE_CREATE_DISPATCH_BASE` flag
- VUID-vkCreateRayTracingPipelinesNV-pipelineCache-02903
  If `pipelineCache` was created with `VK_PIPELINE_CACHE_CREATE_EXTERNALLY_SYNCHRONIZED_BIT`, host access to `pipelineCache` **must** be externally synchronized
- VUID-vkCreateRayTracingPipelinesNV-pNext-09616
  If VkPipelineBinaryInfoKHR::`binaryCount` is not `0` for any element of `pCreateInfos`, `pipelineCache` **must** be VK_NULL_HANDLE
- VUID-vkCreateRayTracingPipelinesNV-pNext-09617
  If a VkPipelineCreateFlags2CreateInfoKHR structure with the `VK_PIPELINE_CREATE_2_CAPTURE_DATA_BIT_KHR` flag set is included in the `pNext` chain of any element of `pCreateInfos`, `pipelineCache` **must** be VK_NULL_HANDLE
- VUID-vkCreateRayTracingPipelinesNV-binaryCount-09620
  If VkPipelineBinaryInfoKHR::`binaryCount` is not `0` for any element of `pCreateInfos`, `VK_PIPELINE_CREATION_FEEDBACK_APPLICATION_PIPELINE_CACHE_HIT_BIT` **must** not be set in the `flags` of that element
- VUID-vkCreateRayTracingPipelinesNV-binaryCount-09621
  If VkPipelineBinaryInfoKHR::`binaryCount` is not `0` for any element of `pCreateInfos`, `VK_PIPELINE_CREATION_FEEDBACK_BASE_PIPELINE_ACCELERATION_BIT` **must** not be set in the `flags` of that element
- VUID-vkCreateRayTracingPipelinesNV-binaryCount-09622
  If VkPipelineBinaryInfoKHR::`binaryCount` is not `0` for any element of `pCreateInfos`, `VK_PIPELINE_CREATE_FAIL_ON_PIPELINE_COMPILE_REQUIRED_BIT_EXT` **must** not be set in the `flags` of that element
- VUID-vkCreateRayTracingPipelinesNV-pNext-10150
  If a VkPipelineCreateFlags2CreateInfoKHR structure is included in the `pNext` chain of any element of `pCreateInfos`, `VK_PIPELINE_CREATE_2_CAPTURE_DATA_BIT_KHR` flag **must** not be set

## Valid Usage (Implicit)

- VUID-vkCreateRayTracingPipelinesNV-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkCreateRayTracingPipelinesNV-pipelineCache-parameter
  If `pipelineCache` is not VK_NULL_HANDLE, `pipelineCache` **must** be a valid VkPipelineCache handle
- VUID-vkCreateRayTracingPipelinesNV-pCreateInfos-parameter
  `pCreateInfos` **must** be a valid pointer to an array of `createInfoCount` valid VkRayTracingPipelineCreateInfoNV structures
- VUID-vkCreateRayTracingPipelinesNV-pAllocator-parameter
  If `pAllocator` is not `NULL`, `pAllocator` **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkCreateRayTracingPipelinesNV-pPipelines-parameter
  `pPipelines` **must** be a valid pointer to an array of `createInfoCount` VkPipeline handles
- VUID-vkCreateRayTracingPipelinesNV-createInfoCount-arraylength
  `createInfoCount` **must** be greater than `0`
- VUID-vkCreateRayTracingPipelinesNV-pipelineCache-parent
  If `pipelineCache` is a valid handle, it **must** have been created, allocated, or retrieved from `device`

## Return Codes

### Success

- `VK_SUCCESS`
- `VK_PIPELINE_COMPILE_REQUIRED_EXT`

### Failure

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_INVALID_SHADER_NV`

To create ray tracing pipelines, call:

```
// Provided by VK_KHR_ray_tracing_pipeline
VkResult vkCreateRayTracingPipelinesKHR(
    VkDevice                                    device,
    VkDeferredOperationKHR                      deferredOperation,
    VkPipelineCache                             pipelineCache,
    uint32_t                                    createInfoCount,
    const VkRayTracingPipelineCreateInfoKHR*    pCreateInfos,
    const VkAllocationCallbacks*                pAllocator,
    VkPipeline*                                 pPipelines);
```

- `device` is the logical device that creates the ray tracing pipelines.
- `deferredOperation` is VK_NULL_HANDLE or the handle of a valid VkDeferredOperationKHR request deferral object for this command.
- `pipelineCache` is either VK_NULL_HANDLE, indicating that pipeline caching is disabled, or the handle of a valid pipeline cache object, in which case use of that cache is enabled for the duration of the command.
- `createInfoCount` is the length of the `pCreateInfos` and `pPipelines` arrays.
- `pCreateInfos` is a pointer to an array of VkRayTracingPipelineCreateInfoKHR structures.
- `pAllocator` controls host memory allocation as described in the Memory Allocation chapter.
- `pPipelines` is a pointer to an array in which the resulting ray tracing pipeline objects are returned.

The `VK_ERROR_INVALID_OPAQUE_CAPTURE_ADDRESS` error is returned if the implementation is unable to reuse the shader group handles provided in VkRayTracingShaderGroupCreateInfoKHR::`pShaderGroupCaptureReplayHandle` when VkPhysicalDeviceRayTracingPipelineFeaturesKHR::`rayTracingPipelineShaderGroupHandleCaptureReplay` is enabled.

Pipelines are created and returned as described for Multiple Pipeline Creation.

## Valid Usage

- VUID-vkCreateRayTracingPipelinesKHR-device-09677
  `device` **must** support at least one queue family with the `VK_QUEUE_COMPUTE_BIT` capability
- VUID-vkCreateRayTracingPipelinesKHR-flags-03415
  If the `flags` member of any element of `pCreateInfos` contains the `VK_PIPELINE_CREATE_DERIVATIVE_BIT` flag, and the `basePipelineIndex` member of that same element is not `-1`, `basePipelineIndex` **must** be less than the index into `pCreateInfos` that corresponds to that element
- VUID-vkCreateRayTracingPipelinesKHR-flags-03416
  If the `flags` member of any element of `pCreateInfos` contains the `VK_PIPELINE_CREATE_DERIVATIVE_BIT` flag, the base pipeline **must** have been created with the `VK_PIPELINE_CREATE_ALLOW_DERIVATIVES_BIT` flag set
- VUID-vkCreateRayTracingPipelinesKHR-flags-03816
  `flags` **must** not contain the `VK_PIPELINE_CREATE_DISPATCH_BASE` flag

- VUID-vkCreateRayTracingPipelinesKHR-pipelineCache-02903
  If `pipelineCache` was created with `VK_PIPELINE_CACHE_CREATE_EXTERNALLY_SYNCHRONIZED_BIT`, host access to `pipelineCache` **must** be externally synchronized

- VUID-vkCreateRayTracingPipelinesKHR-deferredOperation-03678
  Any previous deferred operation that was associated with `deferredOperation` **must** be complete

- VUID-vkCreateRayTracingPipelinesKHR-pNext-09616
  If VkPipelineBinaryInfoKHR::`binaryCount` is not `0` for any element of `pCreateInfos`, `pipelineCache` **must** be VK_NULL_HANDLE
- VUID-vkCreateRayTracingPipelinesKHR-pNext-09617
  If a VkPipelineCreateFlags2CreateInfoKHR structure with the `VK_PIPELINE_CREATE_2_CAPTURE_DATA_BIT_KHR` flag set is included in the `pNext` chain of any element of `pCreateInfos`, `pipelineCache` **must** be VK_NULL_HANDLE
- VUID-vkCreateRayTracingPipelinesKHR-binaryCount-09620
  If VkPipelineBinaryInfoKHR::`binaryCount` is not `0` for any element of `pCreateInfos`, `VK_PIPELINE_CREATION_FEEDBACK_APPLICATION_PIPELINE_CACHE_HIT_BIT` **must** not be set in the `flags` of that element
- VUID-vkCreateRayTracingPipelinesKHR-binaryCount-09621
  If VkPipelineBinaryInfoKHR::`binaryCount` is not `0` for any element of `pCreateInfos`, `VK_PIPELINE_CREATION_FEEDBACK_BASE_PIPELINE_ACCELERATION_BIT` **must** not be set in the `flags` of that element
- VUID-vkCreateRayTracingPipelinesKHR-binaryCount-09622
  If VkPipelineBinaryInfoKHR::`binaryCount` is not `0` for any element of `pCreateInfos`, `VK_PIPELINE_CREATE_FAIL_ON_PIPELINE_COMPILE_REQUIRED_BIT_EXT` **must** not be set in the `flags` of that element
- VUID-vkCreateRayTracingPipelinesKHR-rayTracingPipeline-03586
  The `rayTracingPipeline` feature **must** be enabled
- VUID-vkCreateRayTracingPipelinesKHR-deferredOperation-03587
  If `deferredOperation` is not VK_NULL_HANDLE, the `flags` member of elements of `pCreateInfos` **must** not include `VK_PIPELINE_CREATE_EARLY_RETURN_ON_FAILURE_BIT`

## Valid Usage (Implicit)

- VUID-vkCreateRayTracingPipelinesKHR-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkCreateRayTracingPipelinesKHR-deferredOperation-parameter
  If `deferredOperation` is not VK_NULL_HANDLE, `deferredOperation` **must** be a valid VkDeferredOperationKHR handle
- VUID-vkCreateRayTracingPipelinesKHR-pipelineCache-parameter
  If `pipelineCache` is not VK_NULL_HANDLE, `pipelineCache` **must** be a valid VkPipelineCache handle
- VUID-vkCreateRayTracingPipelinesKHR-pCreateInfos-parameter

pCreateInfos **must** be a valid pointer to an array of createInfoCount valid VkRayTracingPipelineCreateInfoKHR structures

- VUID-vkCreateRayTracingPipelinesKHR-pAllocator-parameter
  If pAllocator is not NULL, pAllocator **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkCreateRayTracingPipelinesKHR-pPipelines-parameter
  pPipelines **must** be a valid pointer to an array of createInfoCount VkPipeline handles
- VUID-vkCreateRayTracingPipelinesKHR-createInfoCount-arraylength
  createInfoCount **must** be greater than 0
- VUID-vkCreateRayTracingPipelinesKHR-deferredOperation-parent
  If deferredOperation is a valid handle, it **must** have been created, allocated, or retrieved from device
- VUID-vkCreateRayTracingPipelinesKHR-pipelineCache-parent
  If pipelineCache is a valid handle, it **must** have been created, allocated, or retrieved from device

## Return Codes

### Success

- VK_SUCCESS
- VK_OPERATION_DEFERRED_KHR
- VK_OPERATION_NOT_DEFERRED_KHR
- VK_PIPELINE_COMPILE_REQUIRED_EXT

### Failure

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY
- VK_ERROR_INVALID_OPAQUE_CAPTURE_ADDRESS

The VkRayTracingPipelineCreateInfoNV structure is defined as:

```
// Provided by VK_NV_ray_tracing
typedef struct VkRayTracingPipelineCreateInfoNV {
    VkStructureType                                 sType;
    const void*                                     pNext;
    VkPipelineCreateFlags                           flags;
    uint32_t                                        stageCount;
    const VkPipelineShaderStageCreateInfo*          pStages;
    uint32_t                                        groupCount;
    const VkRayTracingShaderGroupCreateInfoNV*      pGroups;
    uint32_t                                        maxRecursionDepth;
    VkPipelineLayout                                layout;
    VkPipeline                                      basePipelineHandle;
    int32_t                                         basePipelineIndex;
} VkRayTracingPipelineCreateInfoNV;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is a bitmask of VkPipelineCreateFlagBits specifying how the pipeline will be generated.
- `stageCount` is the number of entries in the `pStages` array.
- `pStages` is a pointer to an array of VkPipelineShaderStageCreateInfo structures specifying the set of the shader stages to be included in the ray tracing pipeline.
- `groupCount` is the number of entries in the `pGroups` array.
- `pGroups` is a pointer to an array of VkRayTracingShaderGroupCreateInfoNV structures describing the set of the shader stages to be included in each shader group in the ray tracing pipeline.
- `maxRecursionDepth` is the maximum recursion depth of shaders executed by this pipeline.
- `layout` is the description of binding locations used by both the pipeline and descriptor sets used with the pipeline.
- `basePipelineHandle` is a pipeline to derive from.
- `basePipelineIndex` is an index into the `pCreateInfos` parameter to use as a pipeline to derive from.

The parameters `basePipelineHandle` and `basePipelineIndex` are described in more detail in Pipeline Derivatives.

If the `pNext` chain includes a VkPipelineCreateFlags2CreateInfoKHR structure, VkPipelineCreateFlags2CreateInfoKHR::`flags` from that structure is used instead of `flags` from this structure.

### Valid Usage

- VUID-VkRayTracingPipelineCreateInfoNV-None-09497
  If the `pNext` chain does not include a VkPipelineCreateFlags2CreateInfoKHR structure, `flags` **must** be a valid combination of VkPipelineCreateFlagBits values

- VUID-VkRayTracingPipelineCreateInfoNV-flags-07984
  If `flags` contains the `VK_PIPELINE_CREATE_DERIVATIVE_BIT` flag, and `basePipelineIndex` is -1, `basePipelineHandle` **must** be a valid ray tracing `VkPipeline` handle
- VUID-VkRayTracingPipelineCreateInfoNV-flags-07985
  If `flags` contains the `VK_PIPELINE_CREATE_DERIVATIVE_BIT` flag, and `basePipelineHandle` is VK_NULL_HANDLE, `basePipelineIndex` **must** be a valid index into the calling command's `pCreateInfos` parameter
- VUID-VkRayTracingPipelineCreateInfoNV-flags-07986
  If `flags` contains the `VK_PIPELINE_CREATE_DERIVATIVE_BIT` flag, `basePipelineIndex` **must** be -1 or `basePipelineHandle` **must** be VK_NULL_HANDLE
- VUID-VkRayTracingPipelineCreateInfoNV-layout-07987
  If a push constant block is declared in a shader, a push constant range in `layout` **must** match the shader stage
- VUID-VkRayTracingPipelineCreateInfoNV-layout-10069
  If a push constant block is declared in a shader, the block must be contained inside the push constant range in `layout` that matches the stage
- VUID-VkRayTracingPipelineCreateInfoNV-layout-07988
  If a resource variables is declared in a shader, a descriptor slot in `layout` **must** match the shader stage
- VUID-VkRayTracingPipelineCreateInfoNV-layout-07990
  If a resource variables is declared in a shader, and the descriptor type is not `VK_DESCRIPTOR_TYPE_MUTABLE_EXT`, a descriptor slot in `layout` **must** match the descriptor type
- VUID-VkRayTracingPipelineCreateInfoNV-layout-07991
  If a resource variables is declared in a shader as an array, a descriptor slot in `layout` **must** match the descriptor count
- VUID-VkRayTracingPipelineCreateInfoNV-pStages-03426
  The shader code for the entry points identified by `pStages`, and the rest of the state identified by this structure **must** adhere to the pipeline linking rules described in the Shader Interfaces chapter
- VUID-VkRayTracingPipelineCreateInfoNV-layout-03428
  The number of resources in `layout` accessible to each shader stage that is used by the pipeline **must** be less than or equal to VkPhysicalDeviceLimits::`maxPerStageResources`
- VUID-VkRayTracingPipelineCreateInfoNV-flags-02904
  `flags` **must** not include `VK_PIPELINE_CREATE_INDIRECT_BINDABLE_BIT_NV`
- VUID-VkRayTracingPipelineCreateInfoNV-pipelineCreationCacheControl-02905
  If the `pipelineCreationCacheControl` feature is not enabled, `flags` **must** not include `VK_PIPELINE_CREATE_FAIL_ON_PIPELINE_COMPILE_REQUIRED_BIT` or `VK_PIPELINE_CREATE_EARLY_RETURN_ON_FAILURE_BIT`
- VUID-VkRayTracingPipelineCreateInfoNV-stage-06232
  The `stage` member of at least one element of `pStages` **must** be `VK_SHADER_STAGE_RAYGEN_BIT_KHR`

- VUID-VkRayTracingPipelineCreateInfoNV-flags-03456
  `flags` **must** not include `VK_PIPELINE_CREATE_LIBRARY_BIT_KHR`
- VUID-VkRayTracingPipelineCreateInfoNV-maxRecursionDepth-03457
  `maxRecursionDepth` **must** be less than or equal to VkPhysicalDeviceRayTracingPropertiesNV::`maxRecursionDepth`
- VUID-VkRayTracingPipelineCreateInfoNV-flags-03458
  `flags` **must** not include `VK_PIPELINE_CREATE_RAY_TRACING_NO_NULL_ANY_HIT_SHADERS_BIT_KHR`
- VUID-VkRayTracingPipelineCreateInfoNV-flags-03459
  `flags` **must** not include `VK_PIPELINE_CREATE_RAY_TRACING_NO_NULL_CLOSEST_HIT_SHADERS_BIT_KHR`
- VUID-VkRayTracingPipelineCreateInfoNV-flags-03460
  `flags` **must** not include `VK_PIPELINE_CREATE_RAY_TRACING_NO_NULL_MISS_SHADERS_BIT_KHR`
- VUID-VkRayTracingPipelineCreateInfoNV-flags-03461
  `flags` **must** not include `VK_PIPELINE_CREATE_RAY_TRACING_NO_NULL_INTERSECTION_SHADERS_BIT_KHR`
- VUID-VkRayTracingPipelineCreateInfoNV-flags-03462
  `flags` **must** not include `VK_PIPELINE_CREATE_RAY_TRACING_SKIP_AABBS_BIT_KHR`
- VUID-VkRayTracingPipelineCreateInfoNV-flags-03463
  `flags` **must** not include `VK_PIPELINE_CREATE_RAY_TRACING_SKIP_TRIANGLES_BIT_KHR`
- VUID-VkRayTracingPipelineCreateInfoNV-flags-03588
  `flags` **must** not include `VK_PIPELINE_CREATE_RAY_TRACING_SHADER_GROUP_HANDLE_CAPTURE_REPLAY_BIT_KHR`
- VUID-VkRayTracingPipelineCreateInfoNV-flags-04948
  `flags` **must** not include `VK_PIPELINE_CREATE_RAY_TRACING_ALLOW_MOTION_BIT_NV`
- VUID-VkRayTracingPipelineCreateInfoNV-flags-02957
  `flags` **must** not include both `VK_PIPELINE_CREATE_DEFER_COMPILE_BIT_NV` and `VK_PIPELINE_CREATE_FAIL_ON_PIPELINE_COMPILE_REQUIRED_BIT` at the same time
- VUID-VkRayTracingPipelineCreateInfoNV-pipelineStageCreationFeedbackCount-06651
  If VkPipelineCreationFeedbackCreateInfo::`pipelineStageCreationFeedbackCount` is not `0`, it **must** be equal to `stageCount`
- VUID-VkRayTracingPipelineCreateInfoNV-stage-06898
  The `stage` value in all `pStages` elements **must** be one of `VK_SHADER_STAGE_RAYGEN_BIT_KHR`, `VK_SHADER_STAGE_ANY_HIT_BIT_KHR`, `VK_SHADER_STAGE_CLOSEST_HIT_BIT_KHR`, `VK_SHADER_STAGE_MISS_BIT_KHR`, `VK_SHADER_STAGE_INTERSECTION_BIT_KHR`, or `VK_SHADER_STAGE_CALLABLE_BIT_KHR`
- VUID-VkRayTracingPipelineCreateInfoNV-flags-07402
  `flags` **must** not include `VK_PIPELINE_CREATE_RAY_TRACING_OPACITY_MICROMAP_BIT_EXT`
- VUID-VkRayTracingPipelineCreateInfoNV-flags-07998
  `flags` **must** not include `VK_PIPELINE_CREATE_RAY_TRACING_DISPLACEMENT_MICROMAP_BIT_NV`

### Valid Usage (Implicit)

- VUID-VkRayTracingPipelineCreateInfoNV-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_RAY_TRACING_PIPELINE_CREATE_INFO_NV`
- VUID-VkRayTracingPipelineCreateInfoNV-pNext-pNext
  Each `pNext` member of any structure (including this one) in the `pNext` chain **must** be either `NULL` or a pointer to a valid instance of VkPipelineCreateFlags2CreateInfoKHR or VkPipelineCreationFeedbackCreateInfo
- VUID-VkRayTracingPipelineCreateInfoNV-sType-unique
  The `sType` value of each struct in the `pNext` chain **must** be unique
- VUID-VkRayTracingPipelineCreateInfoNV-pStages-parameter
  `pStages` **must** be a valid pointer to an array of `stageCount` valid VkPipelineShaderStageCreateInfo structures
- VUID-VkRayTracingPipelineCreateInfoNV-pGroups-parameter
  `pGroups` **must** be a valid pointer to an array of `groupCount` valid VkRayTracingShaderGroupCreateInfoNV structures
- VUID-VkRayTracingPipelineCreateInfoNV-layout-parameter
  `layout` **must** be a valid VkPipelineLayout handle
- VUID-VkRayTracingPipelineCreateInfoNV-stageCount-arraylength
  `stageCount` **must** be greater than `0`
- VUID-VkRayTracingPipelineCreateInfoNV-groupCount-arraylength
  `groupCount` **must** be greater than `0`
- VUID-VkRayTracingPipelineCreateInfoNV-commonparent
  Both of `basePipelineHandle`, and `layout` that are valid handles of non-ignored parameters **must** have been created, allocated, or retrieved from the same VkDevice

The `VkRayTracingPipelineCreateInfoKHR` structure is defined as:

```
// Provided by VK_KHR_ray_tracing_pipeline
typedef struct VkRayTracingPipelineCreateInfoKHR {
    VkStructureType                                     sType;
    const void*                                         pNext;
    VkPipelineCreateFlags                               flags;
    uint32_t                                            stageCount;
    const VkPipelineShaderStageCreateInfo*              pStages;
    uint32_t                                            groupCount;
    const VkRayTracingShaderGroupCreateInfoKHR*         pGroups;
    uint32_t                                            maxPipelineRayRecursionDepth;
    const VkPipelineLibraryCreateInfoKHR*               pLibraryInfo;
    const VkRayTracingPipelineInterfaceCreateInfoKHR*   pLibraryInterface;
    const VkPipelineDynamicStateCreateInfo*             pDynamicState;
    VkPipelineLayout                                    layout;
    VkPipeline                                          basePipelineHandle;
    int32_t                                             basePipelineIndex;
} VkRayTracingPipelineCreateInfoKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is a bitmask of VkPipelineCreateFlagBits specifying how the pipeline will be generated.
- `stageCount` is the number of entries in the `pStages` array.
- `pStages` is a pointer to an array of `stageCount` VkPipelineShaderStageCreateInfo structures describing the set of the shader stages to be included in the ray tracing pipeline.
- `groupCount` is the number of entries in the `pGroups` array.
- `pGroups` is a pointer to an array of `groupCount` VkRayTracingShaderGroupCreateInfoKHR structures describing the set of the shader stages to be included in each shader group in the ray tracing pipeline.
- `maxPipelineRayRecursionDepth` is the maximum recursion depth of shaders executed by this pipeline.
- `pLibraryInfo` is a pointer to a VkPipelineLibraryCreateInfoKHR structure defining pipeline libraries to include.
- `pLibraryInterface` is a pointer to a VkRayTracingPipelineInterfaceCreateInfoKHR structure defining additional information when using pipeline libraries.
- `pDynamicState` is a pointer to a VkPipelineDynamicStateCreateInfo structure, and is used to indicate which properties of the pipeline state object are dynamic and **can** be changed independently of the pipeline state. This **can** be `NULL`, which means no state in the pipeline is considered dynamic.
- `layout` is the description of binding locations used by both the pipeline and descriptor sets used with the pipeline.
- `basePipelineHandle` is a pipeline to derive from.
- `basePipelineIndex` is an index into the `pCreateInfos` parameter to use as a pipeline to derive from.

The parameters basePipelineHandle and basePipelineIndex are described in more detail in Pipeline Derivatives.

When VK_PIPELINE_CREATE_LIBRARY_BIT_KHR is specified, this pipeline defines a *pipeline library* which **cannot** be bound as a ray tracing pipeline directly. Instead, pipeline libraries define common shaders and shader groups which **can** be included in future pipeline creation.

If pipeline libraries are included in pLibraryInfo, shaders defined in those libraries are treated as if they were defined as additional entries in pStages, appended in the order they appear in the pLibraries array and in the pStages array when those libraries were defined.

When referencing shader groups in order to obtain a shader group handle, groups defined in those libraries are treated as if they were defined as additional entries in pGroups, appended in the order they appear in the pLibraries array and in the pGroups array when those libraries were defined. The shaders these groups reference are set when the pipeline library is created, referencing those specified in the pipeline library, not in the pipeline that includes it.

The default stack size for a pipeline if VK_DYNAMIC_STATE_RAY_TRACING_PIPELINE_STACK_SIZE_KHR is not provided is computed as described in Ray Tracing Pipeline Stack.

If the pNext chain includes a VkPipelineCreateFlags2CreateInfoKHR structure, VkPipelineCreateFlags2CreateInfoKHR::flags from that structure is used instead of flags from this structure.

## Valid Usage

- VUID-VkRayTracingPipelineCreateInfoKHR-None-09497
  If the pNext chain does not include a VkPipelineCreateFlags2CreateInfoKHR structure, flags **must** be a valid combination of VkPipelineCreateFlagBits values
- VUID-VkRayTracingPipelineCreateInfoKHR-flags-07984
  If flags contains the VK_PIPELINE_CREATE_DERIVATIVE_BIT flag, and basePipelineIndex is -1, basePipelineHandle **must** be a valid ray tracing VkPipeline handle
- VUID-VkRayTracingPipelineCreateInfoKHR-flags-07985
  If flags contains the VK_PIPELINE_CREATE_DERIVATIVE_BIT flag, and basePipelineHandle is VK_NULL_HANDLE, basePipelineIndex **must** be a valid index into the calling command's pCreateInfos parameter
- VUID-VkRayTracingPipelineCreateInfoKHR-flags-07986
  If flags contains the VK_PIPELINE_CREATE_DERIVATIVE_BIT flag, basePipelineIndex **must** be -1 or basePipelineHandle **must** be VK_NULL_HANDLE
- VUID-VkRayTracingPipelineCreateInfoKHR-layout-07987
  If a push constant block is declared in a shader, a push constant range in layout **must** match the shader stage
- VUID-VkRayTracingPipelineCreateInfoKHR-layout-10069
  If a push constant block is declared in a shader, the block must be contained inside the push constant range in layout that matches the stage
- VUID-VkRayTracingPipelineCreateInfoKHR-layout-07988

If a resource variables is declared in a shader, a descriptor slot in `layout` **must** match the shader stage

- VUID-VkRayTracingPipelineCreateInfoKHR-layout-07990
  If a resource variables is declared in a shader, and the descriptor type is not `VK_DESCRIPTOR_TYPE_MUTABLE_EXT`, a descriptor slot in `layout` **must** match the descriptor type
- VUID-VkRayTracingPipelineCreateInfoKHR-layout-07991
  If a resource variables is declared in a shader as an array, a descriptor slot in `layout` **must** match the descriptor count
- VUID-VkRayTracingPipelineCreateInfoKHR-pStages-03426
  The shader code for the entry points identified by `pStages`, and the rest of the state identified by this structure **must** adhere to the pipeline linking rules described in the Shader Interfaces chapter
- VUID-VkRayTracingPipelineCreateInfoKHR-layout-03428
  The number of resources in `layout` accessible to each shader stage that is used by the pipeline **must** be less than or equal to VkPhysicalDeviceLimits::`maxPerStageResources`
- VUID-VkRayTracingPipelineCreateInfoKHR-flags-02904
  `flags` **must** not include `VK_PIPELINE_CREATE_INDIRECT_BINDABLE_BIT_NV`
- VUID-VkRayTracingPipelineCreateInfoKHR-pipelineCreationCacheControl-02905
  If the `pipelineCreationCacheControl` feature is not enabled, `flags` **must** not include `VK_PIPELINE_CREATE_FAIL_ON_PIPELINE_COMPILE_REQUIRED_BIT` or `VK_PIPELINE_CREATE_EARLY_RETURN_ON_FAILURE_BIT`
- VUID-VkRayTracingPipelineCreateInfoKHR-stage-03425
  If `flags` does not include `VK_PIPELINE_CREATE_LIBRARY_BIT_KHR`, the `stage` member of at least one element of `pStages`, including those implicitly added by `pLibraryInfo`, **must** be `VK_SHADER_STAGE_RAYGEN_BIT_KHR`
- VUID-VkRayTracingPipelineCreateInfoKHR-maxPipelineRayRecursionDepth-03589
  `maxPipelineRayRecursionDepth` **must** be less than or equal to VkPhysicalDeviceRayTracingPipelinePropertiesKHR::`maxRayRecursionDepth`
- VUID-VkRayTracingPipelineCreateInfoKHR-flags-03465
  If `flags` includes `VK_PIPELINE_CREATE_LIBRARY_BIT_KHR`, `pLibraryInterface` **must** not be `NULL`
- VUID-VkRayTracingPipelineCreateInfoKHR-pLibraryInfo-03590
  If `pLibraryInfo` is not `NULL` and its `libraryCount` member is greater than `0`, `pLibraryInterface` **must** not be `NULL`
- VUID-VkRayTracingPipelineCreateInfoKHR-pLibraries-03591
  Each element of `pLibraryInfo->pLibraries` **must** have been created with the value of `maxPipelineRayRecursionDepth` equal to that in this pipeline
- VUID-VkRayTracingPipelineCreateInfoKHR-pLibraryInfo-03592
  If `pLibraryInfo` is not `NULL`, each element of its `pLibraries` member **must** have been created with a `layout` that is compatible with the `layout` in this pipeline
- VUID-VkRayTracingPipelineCreateInfoKHR-pLibraryInfo-03593
  If `pLibraryInfo` is not `NULL`, each element of its `pLibraries` member **must** have been created

with values of the maxPipelineRayPayloadSize and maxPipelineRayHitAttributeSize members of pLibraryInterface equal to those in this pipeline

- VUID-VkRayTracingPipelineCreateInfoKHR-flags-03594
  If flags includes VK_PIPELINE_CREATE_RAY_TRACING_SHADER_GROUP_HANDLE_CAPTURE_REPLAY_BIT_KHR, each element of pLibraryInfo->pLibraries **must** have been created with the VK_PIPELINE_CREATE_RAY_TRACING_SHADER_GROUP_HANDLE_CAPTURE_REPLAY_BIT_KHR bit set
- VUID-VkRayTracingPipelineCreateInfoKHR-flags-04718
  If flags includes VK_PIPELINE_CREATE_RAY_TRACING_SKIP_AABBS_BIT_KHR, each element of pLibraryInfo->pLibraries **must** have been created with the VK_PIPELINE_CREATE_RAY_TRACING_SKIP_AABBS_BIT_KHR bit set
- VUID-VkRayTracingPipelineCreateInfoKHR-flags-04719
  If flags includes VK_PIPELINE_CREATE_RAY_TRACING_SKIP_TRIANGLES_BIT_KHR, each element of pLibraryInfo->pLibraries **must** have been created with the VK_PIPELINE_CREATE_RAY_TRACING_SKIP_TRIANGLES_BIT_KHR bit set
- VUID-VkRayTracingPipelineCreateInfoKHR-flags-04720
  If flags includes VK_PIPELINE_CREATE_RAY_TRACING_NO_NULL_ANY_HIT_SHADERS_BIT_KHR, each element of pLibraryInfo->pLibraries **must** have been created with the VK_PIPELINE_CREATE_RAY_TRACING_NO_NULL_ANY_HIT_SHADERS_BIT_KHR bit set
- VUID-VkRayTracingPipelineCreateInfoKHR-flags-04721
  If flags includes VK_PIPELINE_CREATE_RAY_TRACING_NO_NULL_CLOSEST_HIT_SHADERS_BIT_KHR, each element of pLibraryInfo->pLibraries **must** have been created with the VK_PIPELINE_CREATE_RAY_TRACING_NO_NULL_CLOSEST_HIT_SHADERS_BIT_KHR bit set
- VUID-VkRayTracingPipelineCreateInfoKHR-flags-04722
  If flags includes VK_PIPELINE_CREATE_RAY_TRACING_NO_NULL_INTERSECTION_SHADERS_BIT_KHR, each element of pLibraryInfo->pLibraries **must** have been created with the VK_PIPELINE_CREATE_RAY_TRACING_NO_NULL_INTERSECTION_SHADERS_BIT_KHR bit set
- VUID-VkRayTracingPipelineCreateInfoKHR-flags-04723
  If flags includes VK_PIPELINE_CREATE_RAY_TRACING_NO_NULL_MISS_SHADERS_BIT_KHR, each element of pLibraryInfo->pLibraries **must** have been created with the VK_PIPELINE_CREATE_RAY_TRACING_NO_NULL_MISS_SHADERS_BIT_KHR bit set
- VUID-VkRayTracingPipelineCreateInfoKHR-pLibraryInfo-03595
  If the VK_KHR_pipeline_library extension is not enabled, pLibraryInfo and pLibraryInterface **must** be NULL
- VUID-VkRayTracingPipelineCreateInfoKHR-flags-03470
  If flags includes VK_PIPELINE_CREATE_RAY_TRACING_NO_NULL_ANY_HIT_SHADERS_BIT_KHR, for any element of pGroups with a type of VK_RAY_TRACING_SHADER_GROUP_TYPE_TRIANGLES_HIT_GROUP_KHR or VK_RAY_TRACING_SHADER_GROUP_TYPE_PROCEDURAL_HIT_GROUP_KHR, the anyHitShader of that element **must** not be VK_SHADER_UNUSED_KHR
- VUID-VkRayTracingPipelineCreateInfoKHR-flags-03471
  If flags includes VK_PIPELINE_CREATE_RAY_TRACING_NO_NULL_CLOSEST_HIT_SHADERS_BIT_KHR, for any element of pGroups with a type of VK_RAY_TRACING_SHADER_GROUP_TYPE_TRIANGLES_HIT_GROUP_KHR or

VK_RAY_TRACING_SHADER_GROUP_TYPE_PROCEDURAL_HIT_GROUP_KHR, the closestHitShader of that element **must** not be VK_SHADER_UNUSED_KHR

- VUID-VkRayTracingPipelineCreateInfoKHR-rayTraversalPrimitiveCulling-03596
  If the rayTraversalPrimitiveCulling feature is not enabled, flags **must** not include VK_PIPELINE_CREATE_RAY_TRACING_SKIP_AABBS_BIT_KHR
- VUID-VkRayTracingPipelineCreateInfoKHR-rayTraversalPrimitiveCulling-03597
  If the rayTraversalPrimitiveCulling feature is not enabled, flags **must** not include VK_PIPELINE_CREATE_RAY_TRACING_SKIP_TRIANGLES_BIT_KHR
- VUID-VkRayTracingPipelineCreateInfoKHR-flags-06546
  flags **must** not include both VK_PIPELINE_CREATE_RAY_TRACING_SKIP_TRIANGLES_BIT_KHR and VK_PIPELINE_CREATE_RAY_TRACING_SKIP_AABBS_BIT_KHR
- VUID-VkRayTracingPipelineCreateInfoKHR-flags-03598
  If flags includes VK_PIPELINE_CREATE_RAY_TRACING_SHADER_GROUP_HANDLE_CAPTURE_REPLAY_BIT_KHR, rayTracingPipelineShaderGroupHandleCaptureReplay **must** be enabled
- VUID-VkRayTracingPipelineCreateInfoKHR-rayTracingPipelineShaderGroupHandleCaptureReplay-03599
  If VkPhysicalDeviceRayTracingPipelineFeaturesKHR::rayTracingPipelineShaderGroupHandleCaptureReplay is VK_TRUE and the pShaderGroupCaptureReplayHandle member of any element of pGroups is not NULL, flags **must** include VK_PIPELINE_CREATE_RAY_TRACING_SHADER_GROUP_HANDLE_CAPTURE_REPLAY_BIT_KHR
- VUID-VkRayTracingPipelineCreateInfoKHR-pLibraryInfo-07999
  If pLibraryInfo is NULL or its libraryCount is 0, stageCount **must** not be 0
- VUID-VkRayTracingPipelineCreateInfoKHR-flags-08700
  If flags does not include VK_PIPELINE_CREATE_LIBRARY_BIT_KHR and either pLibraryInfo is NULL or its libraryCount is 0, groupCount **must** not be 0
- VUID-VkRayTracingPipelineCreateInfoKHR-pDynamicStates-03602
  Any element of the pDynamicStates member of pDynamicState **must** be VK_DYNAMIC_STATE_RAY_TRACING_PIPELINE_STACK_SIZE_KHR
- VUID-VkRayTracingPipelineCreateInfoKHR-pipelineStageCreationFeedbackCount-06652
  If VkPipelineCreationFeedbackCreateInfo::pipelineStageCreationFeedbackCount is not 0, it **must** be equal to stageCount
- VUID-VkRayTracingPipelineCreateInfoKHR-stage-06899
  The stage value in all pStages elements **must** be one of VK_SHADER_STAGE_RAYGEN_BIT_KHR, VK_SHADER_STAGE_ANY_HIT_BIT_KHR, VK_SHADER_STAGE_CLOSEST_HIT_BIT_KHR, VK_SHADER_STAGE_MISS_BIT_KHR, VK_SHADER_STAGE_INTERSECTION_BIT_KHR, or VK_SHADER_STAGE_CALLABLE_BIT_KHR
- VUID-VkRayTracingPipelineCreateInfoKHR-flags-07403
  If flags includes VK_PIPELINE_CREATE_RAY_TRACING_OPACITY_MICROMAP_BIT_EXT, each element of pLibraryInfo->pLibraries **must** have been created with the VK_PIPELINE_CREATE_RAY_TRACING_OPACITY_MICROMAP_BIT_EXT bit set
- VUID-VkRayTracingPipelineCreateInfoKHR-flags-08701

If `flags` includes `VK_PIPELINE_CREATE_RAY_TRACING_DISPLACEMENT_MICROMAP_BIT_NV`, each element of `pLibraryInfo->pLibraries` **must** have been created with the `VK_PIPELINE_CREATE_RAY_TRACING_DISPLACEMENT_MICROMAP_BIT_NV` bit set

### Valid Usage (Implicit)

- VUID-VkRayTracingPipelineCreateInfoKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_RAY_TRACING_PIPELINE_CREATE_INFO_KHR`
- VUID-VkRayTracingPipelineCreateInfoKHR-pNext-pNext
  Each `pNext` member of any structure (including this one) in the `pNext` chain **must** be either `NULL` or a pointer to a valid instance of VkPipelineBinaryInfoKHR, VkPipelineCreateFlags2CreateInfoKHR, VkPipelineCreationFeedbackCreateInfo, or VkPipelineRobustnessCreateInfoEXT
- VUID-VkRayTracingPipelineCreateInfoKHR-sType-unique
  The `sType` value of each struct in the `pNext` chain **must** be unique
- VUID-VkRayTracingPipelineCreateInfoKHR-pStages-parameter
  If `stageCount` is not `0`, `pStages` **must** be a valid pointer to an array of `stageCount` valid VkPipelineShaderStageCreateInfo structures
- VUID-VkRayTracingPipelineCreateInfoKHR-pGroups-parameter
  If `groupCount` is not `0`, `pGroups` **must** be a valid pointer to an array of `groupCount` valid VkRayTracingShaderGroupCreateInfoKHR structures
- VUID-VkRayTracingPipelineCreateInfoKHR-pLibraryInfo-parameter
  If `pLibraryInfo` is not `NULL`, `pLibraryInfo` **must** be a valid pointer to a valid VkPipelineLibraryCreateInfoKHR structure
- VUID-VkRayTracingPipelineCreateInfoKHR-pLibraryInterface-parameter
  If `pLibraryInterface` is not `NULL`, `pLibraryInterface` **must** be a valid pointer to a valid VkRayTracingPipelineInterfaceCreateInfoKHR structure
- VUID-VkRayTracingPipelineCreateInfoKHR-pDynamicState-parameter
  If `pDynamicState` is not `NULL`, `pDynamicState` **must** be a valid pointer to a valid VkPipelineDynamicStateCreateInfo structure
- VUID-VkRayTracingPipelineCreateInfoKHR-layout-parameter
  `layout` **must** be a valid VkPipelineLayout handle
- VUID-VkRayTracingPipelineCreateInfoKHR-commonparent
  Both of `basePipelineHandle`, and `layout` that are valid handles of non-ignored parameters **must** have been created, allocated, or retrieved from the same VkDevice

The `VkRayTracingShaderGroupCreateInfoNV` structure is defined as:

```
// Provided by VK_NV_ray_tracing
typedef struct VkRayTracingShaderGroupCreateInfoNV {
    VkStructureType                   sType;
    const void*                       pNext;
    VkRayTracingShaderGroupTypeKHR    type;
    uint32_t                          generalShader;
    uint32_t                          closestHitShader;
    uint32_t                          anyHitShader;
    uint32_t                          intersectionShader;
} VkRayTracingShaderGroupCreateInfoNV;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `type` is the type of hit group specified in this structure.
- `generalShader` is the index of the ray generation, miss, or callable shader from VkRayTracingPipelineCreateInfoNV::`pStages` in the group if the shader group has `type` of `VK_RAY_TRACING_SHADER_GROUP_TYPE_GENERAL_NV`, and `VK_SHADER_UNUSED_NV` otherwise.
- `closestHitShader` is the optional index of the closest hit shader from VkRayTracingPipelineCreateInfoNV::`pStages` in the group if the shader group has `type` of `VK_RAY_TRACING_SHADER_GROUP_TYPE_TRIANGLES_HIT_GROUP_NV` or `VK_RAY_TRACING_SHADER_GROUP_TYPE_PROCEDURAL_HIT_GROUP_NV`, and `VK_SHADER_UNUSED_NV` otherwise.
- `anyHitShader` is the optional index of the any-hit shader from VkRayTracingPipelineCreateInfoNV::`pStages` in the group if the shader group has `type` of `VK_RAY_TRACING_SHADER_GROUP_TYPE_TRIANGLES_HIT_GROUP_NV` or `VK_RAY_TRACING_SHADER_GROUP_TYPE_PROCEDURAL_HIT_GROUP_NV`, and `VK_SHADER_UNUSED_NV` otherwise.
- `intersectionShader` is the index of the intersection shader from VkRayTracingPipelineCreateInfoNV::`pStages` in the group if the shader group has `type` of `VK_RAY_TRACING_SHADER_GROUP_TYPE_PROCEDURAL_HIT_GROUP_NV`, and `VK_SHADER_UNUSED_NV` otherwise.

## Valid Usage

- VUID-VkRayTracingShaderGroupCreateInfoNV-type-02413
  If `type` is `VK_RAY_TRACING_SHADER_GROUP_TYPE_GENERAL_NV` then `generalShader` **must** be a valid index into VkRayTracingPipelineCreateInfoNV::`pStages` referring to a shader of `VK_SHADER_STAGE_RAYGEN_BIT_NV`, `VK_SHADER_STAGE_MISS_BIT_NV`, or `VK_SHADER_STAGE_CALLABLE_BIT_NV`
- VUID-VkRayTracingShaderGroupCreateInfoNV-type-02414
  If `type` is `VK_RAY_TRACING_SHADER_GROUP_TYPE_GENERAL_NV` then `closestHitShader`, `anyHitShader`, and `intersectionShader` **must** be `VK_SHADER_UNUSED_NV`
- VUID-VkRayTracingShaderGroupCreateInfoNV-type-02415
  If `type` is `VK_RAY_TRACING_SHADER_GROUP_TYPE_PROCEDURAL_HIT_GROUP_NV` then `intersectionShader` **must** be a valid index into VkRayTracingPipelineCreateInfoNV::`pStages` referring to a shader of `VK_SHADER_STAGE_INTERSECTION_BIT_NV`

- VUID-VkRayTracingShaderGroupCreateInfoNV-type-02416
  If `type` is `VK_RAY_TRACING_SHADER_GROUP_TYPE_TRIANGLES_HIT_GROUP_NV` then `intersectionShader` **must** be `VK_SHADER_UNUSED_NV`
- VUID-VkRayTracingShaderGroupCreateInfoNV-closestHitShader-02417
  `closestHitShader` **must** be either `VK_SHADER_UNUSED_NV` or a valid index into VkRayTracingPipelineCreateInfoNV::`pStages` referring to a shader of `VK_SHADER_STAGE_CLOSEST_HIT_BIT_NV`
- VUID-VkRayTracingShaderGroupCreateInfoNV-anyHitShader-02418
  `anyHitShader` **must** be either `VK_SHADER_UNUSED_NV` or a valid index into VkRayTracingPipelineCreateInfoNV::`pStages` referring to a shader of `VK_SHADER_STAGE_ANY_HIT_BIT_NV`

### Valid Usage (Implicit)

- VUID-VkRayTracingShaderGroupCreateInfoNV-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_RAY_TRACING_SHADER_GROUP_CREATE_INFO_NV`
- VUID-VkRayTracingShaderGroupCreateInfoNV-pNext-pNext
  `pNext` **must** be `NULL`
- VUID-VkRayTracingShaderGroupCreateInfoNV-type-parameter
  `type` **must** be a valid VkRayTracingShaderGroupTypeKHR value

The `VkRayTracingShaderGroupCreateInfoKHR` structure is defined as:

```
// Provided by VK_KHR_ray_tracing_pipeline
typedef struct VkRayTracingShaderGroupCreateInfoKHR {
    VkStructureType                   sType;
    const void*                       pNext;
    VkRayTracingShaderGroupTypeKHR    type;
    uint32_t                          generalShader;
    uint32_t                          closestHitShader;
    uint32_t                          anyHitShader;
    uint32_t                          intersectionShader;
    const void*                       pShaderGroupCaptureReplayHandle;
} VkRayTracingShaderGroupCreateInfoKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `type` is the type of hit group specified in this structure.
- `generalShader` is the index of the ray generation, miss, or callable shader from VkRayTracingPipelineCreateInfoKHR::`pStages` in the group if the shader group has `type` of `VK_RAY_TRACING_SHADER_GROUP_TYPE_GENERAL_KHR`, and `VK_SHADER_UNUSED_KHR` otherwise.
- `closestHitShader` is the optional index of the closest hit shader from VkRayTracingPipelineCreateInfoKHR::`pStages` in the group if the shader group has `type` of

VK_RAY_TRACING_SHADER_GROUP_TYPE_TRIANGLES_HIT_GROUP_KHR or VK_RAY_TRACING_SHADER_GROUP_TYPE_PROCEDURAL_HIT_GROUP_KHR, and VK_SHADER_UNUSED_KHR otherwise.

- anyHitShader is the optional index of the any-hit shader from VkRayTracingPipelineCreateInfoKHR::pStages in the group if the shader group has type of VK_RAY_TRACING_SHADER_GROUP_TYPE_TRIANGLES_HIT_GROUP_KHR or VK_RAY_TRACING_SHADER_GROUP_TYPE_PROCEDURAL_HIT_GROUP_KHR, and VK_SHADER_UNUSED_KHR otherwise.
- intersectionShader is the index of the intersection shader from VkRayTracingPipelineCreateInfoKHR::pStages in the group if the shader group has type of VK_RAY_TRACING_SHADER_GROUP_TYPE_PROCEDURAL_HIT_GROUP_KHR, and VK_SHADER_UNUSED_KHR otherwise.
- pShaderGroupCaptureReplayHandle is NULL or a pointer to replay information for this shader group queried from vkGetRayTracingCaptureReplayShaderGroupHandlesKHR, as described in Ray Tracing Capture Replay. Ignored if VkPhysicalDeviceRayTracingPipelineFeaturesKHR::rayTracingPipelineShaderGroupHandleCaptureReplay is VK_FALSE.

If the pipeline is created with VK_PIPELINE_CREATE_LIBRARY_BIT_KHR and the pipelineLibraryGroupHandles feature is enabled, pShaderGroupCaptureReplayHandle is inherited by all pipelines which link against this pipeline and remains bitwise identical for any pipeline which references this pipeline library.

## Valid Usage

- VUID-VkRayTracingShaderGroupCreateInfoKHR-type-03474
  If type is VK_RAY_TRACING_SHADER_GROUP_TYPE_GENERAL_KHR then generalShader **must** be a valid index into VkRayTracingPipelineCreateInfoKHR::pStages referring to a shader of VK_SHADER_STAGE_RAYGEN_BIT_KHR, VK_SHADER_STAGE_MISS_BIT_KHR, or VK_SHADER_STAGE_CALLABLE_BIT_KHR
- VUID-VkRayTracingShaderGroupCreateInfoKHR-type-03475
  If type is VK_RAY_TRACING_SHADER_GROUP_TYPE_GENERAL_KHR then closestHitShader, anyHitShader, and intersectionShader **must** be VK_SHADER_UNUSED_KHR
- VUID-VkRayTracingShaderGroupCreateInfoKHR-type-03476
  If type is VK_RAY_TRACING_SHADER_GROUP_TYPE_PROCEDURAL_HIT_GROUP_KHR then intersectionShader **must** be a valid index into VkRayTracingPipelineCreateInfoKHR::pStages referring to a shader of VK_SHADER_STAGE_INTERSECTION_BIT_KHR
- VUID-VkRayTracingShaderGroupCreateInfoKHR-type-03477
  If type is VK_RAY_TRACING_SHADER_GROUP_TYPE_TRIANGLES_HIT_GROUP_KHR then intersectionShader **must** be VK_SHADER_UNUSED_KHR
- VUID-VkRayTracingShaderGroupCreateInfoKHR-closestHitShader-03478
  closestHitShader **must** be either VK_SHADER_UNUSED_KHR or a valid index into VkRayTracingPipelineCreateInfoKHR::pStages referring to a shader of VK_SHADER_STAGE_CLOSEST_HIT_BIT_KHR
- VUID-VkRayTracingShaderGroupCreateInfoKHR-anyHitShader-03479

anyHitShader **must** be either VK_SHADER_UNUSED_KHR or a valid index into VkRayTracingPipelineCreateInfoKHR::pStages referring to a shader of VK_SHADER_STAGE_ANY_HIT_BIT_KHR

- VUID-VkRayTracingShaderGroupCreateInfoKHR-rayTracingPipelineShaderGroupHandleCaptureReplayMixed-03603
  If VkPhysicalDeviceRayTracingPipelineFeaturesKHR::rayTracingPipelineShaderGroupHandleCaptureReplayMixed is VK_FALSE then pShaderGroupCaptureReplayHandle **must** not be provided if it has not been provided on a previous call to ray tracing pipeline creation
- VUID-VkRayTracingShaderGroupCreateInfoKHR-rayTracingPipelineShaderGroupHandleCaptureReplayMixed-03604
  If VkPhysicalDeviceRayTracingPipelineFeaturesKHR::rayTracingPipelineShaderGroupHandleCaptureReplayMixed is VK_FALSE then the caller **must** guarantee that no ray tracing pipeline creation commands with pShaderGroupCaptureReplayHandle provided execute simultaneously with ray tracing pipeline creation commands without pShaderGroupCaptureReplayHandle provided

### Valid Usage (Implicit)

- VUID-VkRayTracingShaderGroupCreateInfoKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_RAY_TRACING_SHADER_GROUP_CREATE_INFO_KHR
- VUID-VkRayTracingShaderGroupCreateInfoKHR-pNext-pNext
  pNext **must** be NULL
- VUID-VkRayTracingShaderGroupCreateInfoKHR-type-parameter
  type **must** be a valid VkRayTracingShaderGroupTypeKHR value

Possible values of type in VkRayTracingShaderGroupCreateInfoKHR are:

```
// Provided by VK_KHR_ray_tracing_pipeline
typedef enum VkRayTracingShaderGroupTypeKHR {
    VK_RAY_TRACING_SHADER_GROUP_TYPE_GENERAL_KHR = 0,
    VK_RAY_TRACING_SHADER_GROUP_TYPE_TRIANGLES_HIT_GROUP_KHR = 1,
    VK_RAY_TRACING_SHADER_GROUP_TYPE_PROCEDURAL_HIT_GROUP_KHR = 2,
  // Provided by VK_NV_ray_tracing
    VK_RAY_TRACING_SHADER_GROUP_TYPE_GENERAL_NV =
VK_RAY_TRACING_SHADER_GROUP_TYPE_GENERAL_KHR,
  // Provided by VK_NV_ray_tracing
    VK_RAY_TRACING_SHADER_GROUP_TYPE_TRIANGLES_HIT_GROUP_NV =
VK_RAY_TRACING_SHADER_GROUP_TYPE_TRIANGLES_HIT_GROUP_KHR,
  // Provided by VK_NV_ray_tracing
    VK_RAY_TRACING_SHADER_GROUP_TYPE_PROCEDURAL_HIT_GROUP_NV =
VK_RAY_TRACING_SHADER_GROUP_TYPE_PROCEDURAL_HIT_GROUP_KHR,
} VkRayTracingShaderGroupTypeKHR;
```

or the equivalent

```
// Provided by VK_NV_ray_tracing
typedef VkRayTracingShaderGroupTypeKHR VkRayTracingShaderGroupTypeNV;
```

- VK_RAY_TRACING_SHADER_GROUP_TYPE_GENERAL_KHR indicates a shader group with a single VK_SHADER_STAGE_RAYGEN_BIT_KHR, VK_SHADER_STAGE_MISS_BIT_KHR, or VK_SHADER_STAGE_CALLABLE_BIT_KHR shader in it.
- VK_RAY_TRACING_SHADER_GROUP_TYPE_TRIANGLES_HIT_GROUP_KHR specifies a shader group that only hits triangles and **must** not contain an intersection shader, only closest hit and any-hit shaders.
- VK_RAY_TRACING_SHADER_GROUP_TYPE_PROCEDURAL_HIT_GROUP_KHR specifies a shader group that only intersects with custom geometry and **must** contain an intersection shader and **may** contain closest hit and any-hit shaders.

**NOTE** For current group types, the hit group type could be inferred from the presence or absence of the intersection shader, but we provide the type explicitly for future hit groups that do not have that property.

The VkRayTracingPipelineInterfaceCreateInfoKHR structure is defined as:

```
// Provided by VK_KHR_ray_tracing_pipeline
typedef struct VkRayTracingPipelineInterfaceCreateInfoKHR {
    VkStructureType    sType;
    const void*        pNext;
    uint32_t           maxPipelineRayPayloadSize;
    uint32_t           maxPipelineRayHitAttributeSize;
} VkRayTracingPipelineInterfaceCreateInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- maxPipelineRayPayloadSize is the maximum payload size in bytes used by any shader in the pipeline.
- maxPipelineRayHitAttributeSize is the maximum attribute structure size in bytes used by any shader in the pipeline.

maxPipelineRayPayloadSize is calculated as the maximum number of bytes used by any block declared in the RayPayloadKHR or IncomingRayPayloadKHR storage classes. maxPipelineRayHitAttributeSize is calculated as the maximum number of bytes used by any block declared in the HitAttributeKHR storage class. As variables in these storage classes do not have explicit offsets, the size should be calculated as if each variable has a scalar alignment equal to the largest scalar alignment of any of the block's members.

**NOTE** There is no explicit upper limit for maxPipelineRayPayloadSize, but in practice it should be kept as small as possible. Similar to invocation local memory, it must be allocated for each shader invocation and for devices which support many simultaneous invocations, this storage can rapidly be exhausted, resulting in

failure.

### Valid Usage

- VUID-VkRayTracingPipelineInterfaceCreateInfoKHR-maxPipelineRayHitAttributeSize-03605
  `maxPipelineRayHitAttributeSize` **must** be less than or equal to VkPhysicalDeviceRayTracingPipelinePropertiesKHR::`maxRayHitAttributeSize`

### Valid Usage (Implicit)

- VUID-VkRayTracingPipelineInterfaceCreateInfoKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_RAY_TRACING_PIPELINE_INTERFACE_CREATE_INFO_KHR`
- VUID-VkRayTracingPipelineInterfaceCreateInfoKHR-pNext-pNext
  `pNext` **must** be `NULL`

To query the opaque handles of shaders in the ray tracing pipeline, call:

```
// Provided by VK_KHR_ray_tracing_pipeline
VkResult vkGetRayTracingShaderGroupHandlesKHR(
    VkDevice                                    device,
    VkPipeline                                  pipeline,
    uint32_t                                    firstGroup,
    uint32_t                                    groupCount,
    size_t                                      dataSize,
    void*                                       pData);
```

or the equivalent command

```
// Provided by VK_NV_ray_tracing
VkResult vkGetRayTracingShaderGroupHandlesNV(
    VkDevice                                    device,
    VkPipeline                                  pipeline,
    uint32_t                                    firstGroup,
    uint32_t                                    groupCount,
    size_t                                      dataSize,
    void*                                       pData);
```

- `device` is the logical device containing the ray tracing pipeline.
- `pipeline` is the ray tracing pipeline object containing the shaders.
- `firstGroup` is the index of the first group to retrieve a handle for from the VkRayTracingPipelineCreateInfoKHR::`pGroups` or VkRayTracingPipelineCreateInfoNV::`pGroups` array.

- groupCount is the number of shader handles to retrieve.
- dataSize is the size in bytes of the buffer pointed to by pData.
- pData is a pointer to an application-allocated buffer where the results will be written.

On success, an array of groupCount shader handles will be written to pData, with each element being of size VkPhysicalDeviceRayTracingPipelinePropertiesKHR::shaderGroupHandleSize.

If pipeline was created with VK_PIPELINE_CREATE_LIBRARY_BIT_KHR and the pipelineLibraryGroupHandles feature is enabled applications **can** query group handles from that pipeline, even if the pipeline is a library and is never bound to a command buffer. These group handles remain bitwise identical for any pipeline which references the pipeline library. Group indices are assigned as-if the pipeline was created without VK_PIPELINE_CREATE_LIBRARY_BIT_KHR.

### Valid Usage

- VUID-vkGetRayTracingShaderGroupHandlesKHR-pipeline-04619
  pipeline **must** be a ray tracing pipeline
- VUID-vkGetRayTracingShaderGroupHandlesKHR-firstGroup-04050
  firstGroup **must** be less than the number of shader groups in pipeline
- VUID-vkGetRayTracingShaderGroupHandlesKHR-firstGroup-02419
  The sum of firstGroup and groupCount **must** be less than or equal to the number of shader groups in pipeline
- VUID-vkGetRayTracingShaderGroupHandlesKHR-dataSize-02420
  dataSize **must** be at least VkPhysicalDeviceRayTracingPipelinePropertiesKHR::shaderGroupHandleSize × groupCount
- VUID-vkGetRayTracingShaderGroupHandlesKHR-pipeline-07828
  If the pipelineLibraryGroupHandles feature is not enabled, pipeline **must** not have been created with VK_PIPELINE_CREATE_LIBRARY_BIT_KHR

### Valid Usage (Implicit)

- VUID-vkGetRayTracingShaderGroupHandlesKHR-device-parameter
  device **must** be a valid VkDevice handle
- VUID-vkGetRayTracingShaderGroupHandlesKHR-pipeline-parameter
  pipeline **must** be a valid VkPipeline handle
- VUID-vkGetRayTracingShaderGroupHandlesKHR-pData-parameter
  pData **must** be a valid pointer to an array of dataSize bytes
- VUID-vkGetRayTracingShaderGroupHandlesKHR-dataSize-arraylength
  dataSize **must** be greater than 0
- VUID-vkGetRayTracingShaderGroupHandlesKHR-pipeline-parent
  pipeline **must** have been created, allocated, or retrieved from device

### Return Codes

**Success**

- VK_SUCCESS

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY

To query the opaque capture data of shader groups in a ray tracing pipeline, call:

```
// Provided by VK_KHR_ray_tracing_pipeline
VkResult vkGetRayTracingCaptureReplayShaderGroupHandlesKHR(
    VkDevice                                    device,
    VkPipeline                                  pipeline,
    uint32_t                                    firstGroup,
    uint32_t                                    groupCount,
    size_t                                      dataSize,
    void*                                       pData);
```

- device is the logical device containing the ray tracing pipeline.
- pipeline is the ray tracing pipeline object containing the shaders.
- firstGroup is the index of the first group to retrieve a handle for from the VkRayTracingPipelineCreateInfoKHR::pGroups array.
- groupCount is the number of shader handles to retrieve.
- dataSize is the size in bytes of the buffer pointed to by pData.
- pData is a pointer to an application-allocated buffer where the results will be written.

On success, an array of groupCount shader handles will be written to pData, with each element being of size VkPhysicalDeviceRayTracingPipelinePropertiesKHR::shaderGroupHandleCaptureReplaySize.

Once queried, this opaque data **can** be provided at pipeline creation time (in a subsequent execution), using VkRayTracingShaderGroupCreateInfoKHR::pShaderGroupCaptureReplayHandle, as described in Ray Tracing Capture Replay.

If pipeline was created with VK_PIPELINE_CREATE_LIBRARY_BIT_KHR and the pipelineLibraryGroupHandles feature is enabled applications **can** query capture replay group handles from that pipeline. The capture replay handle remains bitwise identical for any pipeline which references the pipeline library. Group indices are assigned as-if the pipeline was created without VK_PIPELINE_CREATE_LIBRARY_BIT_KHR.

### Valid Usage

- VUID-vkGetRayTracingCaptureReplayShaderGroupHandlesKHR-pipeline-04620

pipeline **must** be a ray tracing pipeline

- VUID-vkGetRayTracingCaptureReplayShaderGroupHandlesKHR-firstGroup-04051
  firstGroup **must** be less than the number of shader groups in pipeline
- VUID-vkGetRayTracingCaptureReplayShaderGroupHandlesKHR-firstGroup-03483
  The sum of firstGroup and groupCount **must** be less than or equal to the number of shader groups in pipeline
- VUID-vkGetRayTracingCaptureReplayShaderGroupHandlesKHR-dataSize-03484
  dataSize **must** be at least VkPhysicalDeviceRayTracingPipelinePropertiesKHR::shaderGroupHandleCaptureReplaySize × groupCount
- VUID-vkGetRayTracingCaptureReplayShaderGroupHandlesKHR-rayTracingPipelineShaderGroupHandleCaptureReplay-03606
  VkPhysicalDeviceRayTracingPipelineFeaturesKHR::rayTracingPipelineShaderGroupHandleCaptureReplay **must** be enabled to call this function
- VUID-vkGetRayTracingCaptureReplayShaderGroupHandlesKHR-pipeline-03607
  pipeline **must** have been created with a flags that included VK_PIPELINE_CREATE_RAY_TRACING_SHADER_GROUP_HANDLE_CAPTURE_REPLAY_BIT_KHR
- VUID-vkGetRayTracingCaptureReplayShaderGroupHandlesKHR-pipeline-07829
  If the pipelineLibraryGroupHandles feature is not enabled, pipeline **must** not have been created with VK_PIPELINE_CREATE_LIBRARY_BIT_KHR

## Valid Usage (Implicit)

- VUID-vkGetRayTracingCaptureReplayShaderGroupHandlesKHR-device-parameter
  device **must** be a valid VkDevice handle
- VUID-vkGetRayTracingCaptureReplayShaderGroupHandlesKHR-pipeline-parameter
  pipeline **must** be a valid VkPipeline handle
- VUID-vkGetRayTracingCaptureReplayShaderGroupHandlesKHR-pData-parameter
  pData **must** be a valid pointer to an array of dataSize bytes
- VUID-vkGetRayTracingCaptureReplayShaderGroupHandlesKHR-dataSize-arraylength
  dataSize **must** be greater than 0
- VUID-vkGetRayTracingCaptureReplayShaderGroupHandlesKHR-pipeline-parent
  pipeline **must** have been created, allocated, or retrieved from device

## Return Codes

### Success

- VK_SUCCESS

### Failure

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY

Ray tracing pipelines **can** contain more shaders than a graphics or compute pipeline, so to allow parallel compilation of shaders within a pipeline, an application **can** choose to defer compilation until a later point in time.

To compile a deferred shader in a pipeline call:

```
// Provided by VK_NV_ray_tracing
VkResult vkCompileDeferredNV(
    VkDevice                                    device,
    VkPipeline                                  pipeline,
    uint32_t                                    shader);
```

- `device` is the logical device containing the ray tracing pipeline.
- `pipeline` is the ray tracing pipeline object containing the shaders.
- `shader` is the index of the shader to compile.

## Valid Usage

- VUID-vkCompileDeferredNV-pipeline-04621
  `pipeline` **must** be a ray tracing pipeline
- VUID-vkCompileDeferredNV-pipeline-02237
  `pipeline` **must** have been created with `VK_PIPELINE_CREATE_DEFER_COMPILE_BIT_NV`
- VUID-vkCompileDeferredNV-shader-02238
  `shader` **must** not have been called as a deferred compile before

## Valid Usage (Implicit)

- VUID-vkCompileDeferredNV-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkCompileDeferredNV-pipeline-parameter
  `pipeline` **must** be a valid VkPipeline handle
- VUID-vkCompileDeferredNV-pipeline-parent
  `pipeline` **must** have been created, allocated, or retrieved from `device`

## Return Codes

### Success

- `VK_SUCCESS`

### Failure

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`

To query the pipeline stack size of shaders in a shader group in the ray tracing pipeline, call:

```
// Provided by VK_KHR_ray_tracing_pipeline
VkDeviceSize vkGetRayTracingShaderGroupStackSizeKHR(
    VkDevice                                    device,
    VkPipeline                                  pipeline,
    uint32_t                                    group,
    VkShaderGroupShaderKHR                      groupShader);
```

- `device` is the logical device containing the ray tracing pipeline.
- `pipeline` is the ray tracing pipeline object containing the shaders groups.
- `group` is the index of the shader group to query.
- `groupShader` is the type of shader from the group to query.

The return value is the ray tracing pipeline stack size in bytes for the specified shader as called from the specified shader group.

### Valid Usage

- VUID-vkGetRayTracingShaderGroupStackSizeKHR-pipeline-04622
  `pipeline` **must** be a ray tracing pipeline
- VUID-vkGetRayTracingShaderGroupStackSizeKHR-group-03608
  The value of `group` **must** be less than the number of shader groups in `pipeline`
- VUID-vkGetRayTracingShaderGroupStackSizeKHR-groupShader-03609
  The shader identified by `groupShader` in `group` **must** not be `VK_SHADER_UNUSED_KHR`

### Valid Usage (Implicit)

- VUID-vkGetRayTracingShaderGroupStackSizeKHR-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkGetRayTracingShaderGroupStackSizeKHR-pipeline-parameter
  `pipeline` **must** be a valid VkPipeline handle
- VUID-vkGetRayTracingShaderGroupStackSizeKHR-groupShader-parameter
  `groupShader` **must** be a valid VkShaderGroupShaderKHR value
- VUID-vkGetRayTracingShaderGroupStackSizeKHR-pipeline-parent
  `pipeline` **must** have been created, allocated, or retrieved from `device`

Possible values of `groupShader` in vkGetRayTracingShaderGroupStackSizeKHR are:

```
// Provided by VK_KHR_ray_tracing_pipeline
typedef enum VkShaderGroupShaderKHR {
    VK_SHADER_GROUP_SHADER_GENERAL_KHR = 0,
    VK_SHADER_GROUP_SHADER_CLOSEST_HIT_KHR = 1,
    VK_SHADER_GROUP_SHADER_ANY_HIT_KHR = 2,
    VK_SHADER_GROUP_SHADER_INTERSECTION_KHR = 3,
} VkShaderGroupShaderKHR;
```

- VK_SHADER_GROUP_SHADER_GENERAL_KHR uses the shader specified in the group with VkRayTracingShaderGroupCreateInfoKHR::generalShader
- VK_SHADER_GROUP_SHADER_CLOSEST_HIT_KHR uses the shader specified in the group with VkRayTracingShaderGroupCreateInfoKHR::closestHitShader
- VK_SHADER_GROUP_SHADER_ANY_HIT_KHR uses the shader specified in the group with VkRayTracingShaderGroupCreateInfoKHR::anyHitShader
- VK_SHADER_GROUP_SHADER_INTERSECTION_KHR uses the shader specified in the group with VkRayTracingShaderGroupCreateInfoKHR::intersectionShader

To dynamically set the stack size for a ray tracing pipeline, call:

```
// Provided by VK_KHR_ray_tracing_pipeline
void vkCmdSetRayTracingPipelineStackSizeKHR(
    VkCommandBuffer                             commandBuffer,
    uint32_t                                    pipelineStackSize);
```

- commandBuffer is the command buffer into which the command will be recorded.
- pipelineStackSize is the stack size to use for subsequent ray tracing trace commands.

This command sets the stack size for subsequent ray tracing commands when the ray tracing pipeline is created with VK_DYNAMIC_STATE_RAY_TRACING_PIPELINE_STACK_SIZE_KHR set in VkPipelineDynamicStateCreateInfo::pDynamicStates. Otherwise, the stack size is computed as described in Ray Tracing Pipeline Stack.

### Valid Usage

- VUID-vkCmdSetRayTracingPipelineStackSizeKHR-pipelineStackSize-03610
  pipelineStackSize **must** be large enough for any dynamic execution through the shaders in the ray tracing pipeline used by a subsequent trace call

### Valid Usage (Implicit)

- VUID-vkCmdSetRayTracingPipelineStackSizeKHR-commandBuffer-parameter
  commandBuffer **must** be a valid VkCommandBuffer handle
- VUID-vkCmdSetRayTracingPipelineStackSizeKHR-commandBuffer-recording

commandBuffer **must** be in the recording state

- VUID-vkCmdSetRayTracingPipelineStackSizeKHR-commandBuffer-cmdpool
  The VkCommandPool that commandBuffer was allocated from **must** support compute operations
- VUID-vkCmdSetRayTracingPipelineStackSizeKHR-renderpass
  This command **must** only be called outside of a render pass instance
- VUID-vkCmdSetRayTracingPipelineStackSizeKHR-videocoding
  This command **must** only be called outside of a video coding scope

### Host Synchronization

- Host access to commandBuffer **must** be externally synchronized
- Host access to the VkCommandPool that commandBuffer was allocated from **must** be externally synchronized

### Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Outside | Outside | Compute | State |

## 10.5. Pipeline Destruction

To destroy a pipeline, call:

```
// Provided by VK_VERSION_1_0
void vkDestroyPipeline(
    VkDevice                                    device,
    VkPipeline                                  pipeline,
    const VkAllocationCallbacks*                pAllocator);
```

- device is the logical device that destroys the pipeline.
- pipeline is the handle of the pipeline to destroy.
- pAllocator controls host memory allocation as described in the Memory Allocation chapter.

### Valid Usage

- VUID-vkDestroyPipeline-pipeline-00765
  All submitted commands that refer to pipeline **must** have completed execution

- VUID-vkDestroyPipeline-pipeline-00766
  If `VkAllocationCallbacks` were provided when `pipeline` was created, a compatible set of callbacks **must** be provided here
- VUID-vkDestroyPipeline-pipeline-00767
  If no `VkAllocationCallbacks` were provided when `pipeline` was created, `pAllocator` **must** be `NULL`

### Valid Usage (Implicit)

- VUID-vkDestroyPipeline-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkDestroyPipeline-pipeline-parameter
  If `pipeline` is not VK_NULL_HANDLE, `pipeline` **must** be a valid VkPipeline handle
- VUID-vkDestroyPipeline-pAllocator-parameter
  If `pAllocator` is not `NULL`, `pAllocator` **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkDestroyPipeline-pipeline-parent
  If `pipeline` is a valid handle, it **must** have been created, allocated, or retrieved from `device`

### Host Synchronization

- Host access to `pipeline` **must** be externally synchronized

## 10.6. Pipeline Derivatives

A pipeline derivative is a child pipeline created from a parent pipeline, where the child and parent are expected to have much commonality.

The goal of derivative pipelines is that they be cheaper to create using the parent as a starting point, and that it be more efficient (on either host or device) to switch/bind between children of the same parent.

A derivative pipeline is created by setting the `VK_PIPELINE_CREATE_DERIVATIVE_BIT` flag in the `Vk*PipelineCreateInfo` structure. If this is set, then exactly one of `basePipelineHandle` or `basePipelineIndex` members of the structure **must** have a valid handle/index, and specifies the parent pipeline. If `basePipelineHandle` is used, the parent pipeline **must** have already been created. If `basePipelineIndex` is used, then the parent is being created in the same command. VK_NULL_HANDLE acts as the invalid handle for `basePipelineHandle`, and -1 is the invalid index for `basePipelineIndex`. If `basePipelineIndex` is used, the base pipeline **must** appear earlier in the array. The base pipeline **must** have been created with the `VK_PIPELINE_CREATE_ALLOW_DERIVATIVES_BIT` flag set.

# 10.7. Pipeline Cache

Pipeline cache objects allow the result of pipeline construction to be reused between pipelines and between runs of an application. Reuse between pipelines is achieved by passing the same pipeline cache object when creating multiple related pipelines. Reuse across runs of an application is achieved by retrieving pipeline cache contents in one run of an application, saving the contents, and using them to preinitialize a pipeline cache on a subsequent run. The contents of the pipeline cache objects are managed by the implementation. Applications **can** manage the host memory consumed by a pipeline cache object and control the amount of data retrieved from a pipeline cache object.

Pipeline cache objects are represented by `VkPipelineCache` handles:

```
// Provided by VK_VERSION_1_0
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkPipelineCache)
```

## 10.7.1. Creating a Pipeline Cache

To create pipeline cache objects, call:

```
// Provided by VK_VERSION_1_0
VkResult vkCreatePipelineCache(
    VkDevice                                    device,
    const VkPipelineCacheCreateInfo*            pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkPipelineCache*                            pPipelineCache);
```

- `device` is the logical device that creates the pipeline cache object.
- `pCreateInfo` is a pointer to a VkPipelineCacheCreateInfo structure containing initial parameters for the pipeline cache object.
- `pAllocator` controls host memory allocation as described in the Memory Allocation chapter.
- `pPipelineCache` is a pointer to a VkPipelineCache handle in which the resulting pipeline cache object is returned.

**NOTE** Applications **can** track and manage the total host memory size of a pipeline cache object using the `pAllocator`. Applications **can** limit the amount of data retrieved from a pipeline cache object in `vkGetPipelineCacheData`. Implementations **should** not internally limit the total number of entries added to a pipeline cache object or the total host memory consumed.

Once created, a pipeline cache **can** be passed to the vkCreateGraphicsPipelines vkCreateRayTracingPipelinesKHR, vkCreateRayTracingPipelinesNV, and vkCreateComputePipelines commands. If the pipeline cache passed into these commands is not VK_NULL_HANDLE, the implementation will query it for possible reuse opportunities and update it with new content. The use of the pipeline cache object in these commands is internally synchronized, and the same

pipeline cache object **can** be used in multiple threads simultaneously.

If `flags` of `pCreateInfo` includes `VK_PIPELINE_CACHE_CREATE_EXTERNALLY_SYNCHRONIZED_BIT`, all commands that modify the returned pipeline cache object **must** be externally synchronized.

| | |
|---|---|
| **NOTE** | Implementations **should** make every effort to limit any critical sections to the actual accesses to the cache, which is expected to be significantly shorter than the duration of the `vkCreate*Pipelines` commands. |

### Valid Usage (Implicit)

- VUID-vkCreatePipelineCache-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkCreatePipelineCache-pCreateInfo-parameter
  `pCreateInfo` **must** be a valid pointer to a valid VkPipelineCacheCreateInfo structure
- VUID-vkCreatePipelineCache-pAllocator-parameter
  If `pAllocator` is not `NULL`, `pAllocator` **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkCreatePipelineCache-pPipelineCache-parameter
  `pPipelineCache` **must** be a valid pointer to a VkPipelineCache handle

### Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`

The `VkPipelineCacheCreateInfo` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkPipelineCacheCreateInfo {
    VkStructureType                 sType;
    const void*                     pNext;
    VkPipelineCacheCreateFlags      flags;
    size_t                          initialDataSize;
    const void*                     pInitialData;
} VkPipelineCacheCreateInfo;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.

- `flags` is a bitmask of VkPipelineCacheCreateFlagBits specifying the behavior of the pipeline cache.
- `initialDataSize` is the number of bytes in `pInitialData`. If `initialDataSize` is zero, the pipeline cache will initially be empty.
- `pInitialData` is a pointer to previously retrieved pipeline cache data. If the pipeline cache data is incompatible (as defined below) with the device, the pipeline cache will be initially empty. If `initialDataSize` is zero, `pInitialData` is ignored.

### Valid Usage

- VUID-VkPipelineCacheCreateInfo-initialDataSize-00768
  If `initialDataSize` is not `0`, it **must** be equal to the size of `pInitialData`, as returned by `vkGetPipelineCacheData` when `pInitialData` was originally retrieved
- VUID-VkPipelineCacheCreateInfo-initialDataSize-00769
  If `initialDataSize` is not `0`, `pInitialData` **must** have been retrieved from a previous call to `vkGetPipelineCacheData`
- VUID-VkPipelineCacheCreateInfo-pipelineCreationCacheControl-02892
  If the `pipelineCreationCacheControl` feature is not enabled, `flags` **must** not include `VK_PIPELINE_CACHE_CREATE_EXTERNALLY_SYNCHRONIZED_BIT`

### Valid Usage (Implicit)

- VUID-VkPipelineCacheCreateInfo-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PIPELINE_CACHE_CREATE_INFO`
- VUID-VkPipelineCacheCreateInfo-pNext-pNext
  `pNext` **must** be `NULL`
- VUID-VkPipelineCacheCreateInfo-flags-parameter
  `flags` **must** be a valid combination of VkPipelineCacheCreateFlagBits values
- VUID-VkPipelineCacheCreateInfo-pInitialData-parameter
  If `initialDataSize` is not `0`, `pInitialData` **must** be a valid pointer to an array of `initialDataSize` bytes

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkPipelineCacheCreateFlags;
```

`VkPipelineCacheCreateFlags` is a bitmask type for setting a mask of zero or more VkPipelineCacheCreateFlagBits.

Bits which **can** be set in VkPipelineCacheCreateInfo::`flags`, specifying behavior of the pipeline cache, are:

```
// Provided by VK_EXT_pipeline_creation_cache_control
typedef enum VkPipelineCacheCreateFlagBits {
    VK_PIPELINE_CACHE_CREATE_EXTERNALLY_SYNCHRONIZED_BIT = 0x00000001,
  // Provided by VK_EXT_pipeline_creation_cache_control
    VK_PIPELINE_CACHE_CREATE_EXTERNALLY_SYNCHRONIZED_BIT_EXT =
VK_PIPELINE_CACHE_CREATE_EXTERNALLY_SYNCHRONIZED_BIT,
} VkPipelineCacheCreateFlagBits;
```

- `VK_PIPELINE_CACHE_CREATE_EXTERNALLY_SYNCHRONIZED_BIT` specifies that all commands that modify the created VkPipelineCache will be externally synchronized. When set, the implementation **may** skip any unnecessary processing needed to support simultaneous modification from multiple threads where allowed.

### 10.7.2. Merging Pipeline Caches

Pipeline cache objects **can** be merged using the command:

```
// Provided by VK_VERSION_1_0
VkResult vkMergePipelineCaches(
    VkDevice                                    device,
    VkPipelineCache                             dstCache,
    uint32_t                                    srcCacheCount,
    const VkPipelineCache*                      pSrcCaches);
```

- `device` is the logical device that owns the pipeline cache objects.
- `dstCache` is the handle of the pipeline cache to merge results into.
- `srcCacheCount` is the length of the `pSrcCaches` array.
- `pSrcCaches` is a pointer to an array of pipeline cache handles, which will be merged into `dstCache`. The previous contents of `dstCache` are included after the merge.

| | |
|---|---|
| **NOTE** | The details of the merge operation are implementation-dependent, but implementations **should** merge the contents of the specified pipelines and prune duplicate entries. |

**Valid Usage**

- VUID-vkMergePipelineCaches-dstCache-00770
  `dstCache` **must** not appear in the list of source caches

**Valid Usage (Implicit)**

- VUID-vkMergePipelineCaches-device-parameter
  `device` **must** be a valid VkDevice handle

- VUID-vkMergePipelineCaches-dstCache-parameter
  `dstCache` **must** be a valid VkPipelineCache handle
- VUID-vkMergePipelineCaches-pSrcCaches-parameter
  `pSrcCaches` **must** be a valid pointer to an array of `srcCacheCount` valid VkPipelineCache handles
- VUID-vkMergePipelineCaches-srcCacheCount-arraylength
  `srcCacheCount` **must** be greater than `0`
- VUID-vkMergePipelineCaches-dstCache-parent
  `dstCache` **must** have been created, allocated, or retrieved from `device`
- VUID-vkMergePipelineCaches-pSrcCaches-parent
  Each element of `pSrcCaches` **must** have been created, allocated, or retrieved from `device`

### Host Synchronization

- Host access to `dstCache` **must** be externally synchronized

### Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`

## 10.7.3. Retrieving Pipeline Cache Data

Data **can** be retrieved from a pipeline cache object using the command:

```
// Provided by VK_VERSION_1_0
VkResult vkGetPipelineCacheData(
    VkDevice                                    device,
    VkPipelineCache                             pipelineCache,
    size_t*                                     pDataSize,
    void*                                       pData);
```

- `device` is the logical device that owns the pipeline cache.
- `pipelineCache` is the pipeline cache to retrieve data from.
- `pDataSize` is a pointer to a `size_t` value related to the amount of data in the pipeline cache, as described below.
- `pData` is either `NULL` or a pointer to a buffer.

If pData is NULL, then the maximum size of the data that **can** be retrieved from the pipeline cache, in bytes, is returned in pDataSize. Otherwise, pDataSize **must** point to a variable set by the application to the size of the buffer, in bytes, pointed to by pData, and on return the variable is overwritten with the amount of data actually written to pData. If pDataSize is less than the maximum size that **can** be retrieved by the pipeline cache, at most pDataSize bytes will be written to pData, and VK_INCOMPLETE will be returned instead of VK_SUCCESS, to indicate that not all of the pipeline cache was returned.

Any data written to pData is valid and **can** be provided as the pInitialData member of the VkPipelineCacheCreateInfo structure passed to vkCreatePipelineCache.

Two calls to vkGetPipelineCacheData with the same parameters **must** retrieve the same data unless a command that modifies the contents of the cache is called between them.

The initial bytes written to pData **must** be a header as described in the Pipeline Cache Header section.

If pDataSize is less than what is necessary to store this header, nothing will be written to pData and zero will be written to pDataSize.

## Valid Usage (Implicit)

- VUID-vkGetPipelineCacheData-device-parameter
  device **must** be a valid VkDevice handle
- VUID-vkGetPipelineCacheData-pipelineCache-parameter
  pipelineCache **must** be a valid VkPipelineCache handle
- VUID-vkGetPipelineCacheData-pDataSize-parameter
  pDataSize **must** be a valid pointer to a size_t value
- VUID-vkGetPipelineCacheData-pData-parameter
  If the value referenced by pDataSize is not 0, and pData is not NULL, pData **must** be a valid pointer to an array of pDataSize bytes
- VUID-vkGetPipelineCacheData-pipelineCache-parent
  pipelineCache **must** have been created, allocated, or retrieved from device

## Return Codes

### Success

- VK_SUCCESS
- VK_INCOMPLETE

### Failure

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY

### 10.7.4. Pipeline Cache Header

Applications **can** store the data retrieved from the pipeline cache, and use these data, possibly in a future run of the application, to populate new pipeline cache objects. The results of pipeline compiles, however, **may** depend on the vendor ID, device ID, driver version, and other details of the device. To enable applications to detect when previously retrieved data is incompatible with the device, the pipeline cache data **must** begin with a valid pipeline cache header.

**NOTE** Structures described in this section are not part of the Vulkan API and are only used to describe the representation of data elements in pipeline cache data. Accordingly, the valid usage clauses defined for structures defined in this section do not define valid usage conditions for APIs accepting pipeline cache data as input, as providing invalid pipeline cache data as input to any Vulkan API commands will result in the provided pipeline cache data being ignored.

Version one of the pipeline cache header is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkPipelineCacheHeaderVersionOne {
    uint32_t                        headerSize;
    VkPipelineCacheHeaderVersion    headerVersion;
    uint32_t                        vendorID;
    uint32_t                        deviceID;
    uint8_t                         pipelineCacheUUID[VK_UUID_SIZE];
} VkPipelineCacheHeaderVersionOne;
```

- `headerSize` is the length in bytes of the pipeline cache header.
- `headerVersion` is a VkPipelineCacheHeaderVersion value specifying the version of the header. A consumer of the pipeline cache **should** use the cache version to interpret the remainder of the cache header.
- `vendorID` is the `VkPhysicalDeviceProperties::vendorID` of the implementation.
- `deviceID` is the `VkPhysicalDeviceProperties::deviceID` of the implementation.
- `pipelineCacheUUID` is the `VkPhysicalDeviceProperties::pipelineCacheUUID` of the implementation.

Unlike most structures declared by the Vulkan API, all fields of this structure are written with the least significant byte first, regardless of host byte-order.

The C language specification does not define the packing of structure members. This layout assumes tight structure member packing, with members laid out in the order listed in the structure, and the intended size of the structure is 32 bytes. If a compiler produces code that diverges from that pattern, applications **must** employ another method to set values at the correct offsets.

#### Valid Usage

- VUID-VkPipelineCacheHeaderVersionOne-headerSize-04967
  `headerSize` **must** be 32

- VUID-VkPipelineCacheHeaderVersionOne-headerVersion-04968
  headerVersion **must** be VK_PIPELINE_CACHE_HEADER_VERSION_ONE
- VUID-VkPipelineCacheHeaderVersionOne-headerSize-08990
  headerSize **must** not exceed the size of the pipeline cache

### Valid Usage (Implicit)

- VUID-VkPipelineCacheHeaderVersionOne-headerVersion-parameter
  headerVersion **must** be a valid VkPipelineCacheHeaderVersion value

Possible values of the headerVersion value of the pipeline cache header are:

```
// Provided by VK_VERSION_1_0
typedef enum VkPipelineCacheHeaderVersion {
    VK_PIPELINE_CACHE_HEADER_VERSION_ONE = 1,
} VkPipelineCacheHeaderVersion;
```

- VK_PIPELINE_CACHE_HEADER_VERSION_ONE specifies version one of the pipeline cache, described by VkPipelineCacheHeaderVersionOne.

## 10.7.5. Destroying a Pipeline Cache

To destroy a pipeline cache, call:

```
// Provided by VK_VERSION_1_0
void vkDestroyPipelineCache(
    VkDevice                                    device,
    VkPipelineCache                             pipelineCache,
    const VkAllocationCallbacks*                pAllocator);
```

- device is the logical device that destroys the pipeline cache object.
- pipelineCache is the handle of the pipeline cache to destroy.
- pAllocator controls host memory allocation as described in the Memory Allocation chapter.

### Valid Usage

- VUID-vkDestroyPipelineCache-pipelineCache-00771
  If VkAllocationCallbacks were provided when pipelineCache was created, a compatible set of callbacks **must** be provided here
- VUID-vkDestroyPipelineCache-pipelineCache-00772
  If no VkAllocationCallbacks were provided when pipelineCache was created, pAllocator **must** be NULL

### Valid Usage (Implicit)

- VUID-vkDestroyPipelineCache-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkDestroyPipelineCache-pipelineCache-parameter
  If `pipelineCache` is not VK_NULL_HANDLE, `pipelineCache` **must** be a valid VkPipelineCache handle
- VUID-vkDestroyPipelineCache-pAllocator-parameter
  If `pAllocator` is not `NULL`, `pAllocator` **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkDestroyPipelineCache-pipelineCache-parent
  If `pipelineCache` is a valid handle, it **must** have been created, allocated, or retrieved from `device`

### Host Synchronization

- Host access to `pipelineCache` **must** be externally synchronized

## 10.8. Pipeline Binaries

Pipeline binary objects allow the result of pipeline construction to be reused between pipelines and between runs of an application. Reuse is achieved by extracting pipeline binaries from a VkPipeline object, associating them with a corresponding VkPipelineBinaryKeyKHR and then adding a VkPipelineBinaryInfoKHR to the `pNext` chain of any `Vk*PipelineCreateInfo` when creating a pipeline. Pipeline binaries can be reused between runs by extracting `VkPipelineBinaryDataKHR` from `VkPipelineBinaryKHR` objects, saving the contents, and then using them to create a `VkPipelineBinaryKHR` object on subsequent runs.

When creating a pipeline that includes VkPipelineBinaryInfoKHR in the `pNext` chain, or has the `VK_PIPELINE_CREATE_2_CAPTURE_DATA_BIT_KHR` flag set, the use of VkPipelineCache objects is not allowed.

Pipeline binary objects are represented by `VkPipelineBinaryKHR` handles:

```
// Provided by VK_KHR_pipeline_binary
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkPipelineBinaryKHR)
```

### 10.8.1. Generating the Pipeline Key

To generate the key for a particular pipeline creation info, call:

```
// Provided by VK_KHR_pipeline_binary
VkResult vkGetPipelineKeyKHR(
    VkDevice                                    device,
    const VkPipelineCreateInfoKHR*              pPipelineCreateInfo,
    VkPipelineBinaryKeyKHR*                     pPipelineKey);
```

- `device` is the logical device that creates the pipeline object.
- `pPipelineCreateInfo` is `NULL` or a pointer to a VkPipelineCreateInfoKHR structure.
- `pPipelineKey` is a pointer to a VkPipelineBinaryKeyKHR structure in which the resulting key is returned.

If `pPipelineCreateInfo` is `NULL`, then the implementation **must** return the global key that applies to all pipelines. If the key obtained in this way changes between saving and restoring data obtained from vkGetPipelineBinaryDataKHR in a different VkDevice, then the application **must** assume that the restored data is invalid and cannot be passed to vkCreatePipelineBinariesKHR. Otherwise the application **can** assume the data is still valid.

If `pPipelineCreateInfo` is not `NULL`, the key obtained functions as a method to compare two pipeline creation info structures. Implementations **may** not compare parts of a pipeline creation info which would not contribute to the final binary output. If a shader module identifier is used instead of a shader module, the `pPipelineKey` generated **must** be equal to the key generated when using the shader module from which the identifier was queried. If the content of two `pPipelineKey` are equal, pipelines created with the two `pPipelineCreateInfo->pname`:pNext create infos **must** produce the same VkPipelineBinaryKHR contents.

The pipeline key is distinct from pipeline binary key. Pipeline binary keys **can** only be obtained after compilation. The pipeline key is intended to optionally allow associating pipeline create info with multiple pipeline binary keys.

### Valid Usage

- VUID-vkGetPipelineKeyKHR-pNext-09605
  The `pNext` chain of `pPipelineCreateInfo` **must** not set VkPipelineBinaryInfoKHR::`binaryCount` to a value greater than `0`

### Valid Usage (Implicit)

- VUID-vkGetPipelineKeyKHR-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkGetPipelineKeyKHR-pPipelineCreateInfo-parameter
  If `pPipelineCreateInfo` is not `NULL`, `pPipelineCreateInfo` **must** be a valid pointer to a valid VkPipelineCreateInfoKHR structure
- VUID-vkGetPipelineKeyKHR-pPipelineKey-parameter
  `pPipelineKey` **must** be a valid pointer to a VkPipelineBinaryKeyKHR structure

### Return Codes

**Success**

- VK_SUCCESS

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY

The VkPipelineCreateInfoKHR structure is defined as:

```
// Provided by VK_KHR_pipeline_binary
typedef struct VkPipelineCreateInfoKHR {
    VkStructureType    sType;
    void*              pNext;
} VkPipelineCreateInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is a pointer to a structure extending this structure.

### Valid Usage

- VUID-VkPipelineCreateInfoKHR-pNext-09604
  pNext **must** be pointer to a valid instance of VkGraphicsPipelineCreateInfo, VkExecutionGraphPipelineCreateInfoAMDX, VkRayTracingPipelineCreateInfoKHR, or VkComputePipelineCreateInfo

### Valid Usage (Implicit)

- VUID-VkPipelineCreateInfoKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PIPELINE_CREATE_INFO_KHR

The VkPipelineBinaryKeyKHR structure is defined as:

```
// Provided by VK_KHR_pipeline_binary
typedef struct VkPipelineBinaryKeyKHR {
    VkStructureType    sType;
    void*              pNext;
    uint32_t           keySize;
    uint8_t            key[VK_MAX_PIPELINE_BINARY_KEY_SIZE_KHR];
} VkPipelineBinaryKeyKHR;
```

- sType is a VkStructureType value identifying this structure.

- pNext is NULL or a pointer to a structure extending this structure.
- keySize is the size, in bytes, of valid data returned in key.
- key is a buffer of opaque data specifying a pipeline binary key.

Any returned values beyond the first keySize bytes are undefined. Implementations **must** return a keySize greater than 0, and less-or-equal to VK_MAX_PIPELINE_BINARY_KEY_SIZE_KHR.

Two keys are considered equal if keySize is equal and the first keySize bytes of key compare equal.

Implementations **may** return a different keySize for different binaries.

Implementations **should** ensure that keySize is large enough to uniquely identify a pipeline binary.

**Valid Usage (Implicit)**

- VUID-VkPipelineBinaryKeyKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PIPELINE_BINARY_KEY_KHR
- VUID-VkPipelineBinaryKeyKHR-pNext-pNext
  pNext **must** be NULL

VK_MAX_PIPELINE_BINARY_KEY_SIZE_KHR is the length in bytes of a binary key, as returned in VkPipelineBinaryKeyKHR::keySize.

```
#define VK_MAX_PIPELINE_BINARY_KEY_SIZE_KHR 32U
```

### 10.8.2. Creating Pipeline Binaries

To create pipeline binary objects, call:

```
// Provided by VK_KHR_pipeline_binary
VkResult vkCreatePipelineBinariesKHR(
    VkDevice                                    device,
    const VkPipelineBinaryCreateInfoKHR*        pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkPipelineBinaryHandlesInfoKHR*             pBinaries);
```

- device is the logical device that creates the pipeline binary objects.
- pCreateInfo is a pointer to a VkPipelineBinaryCreateInfoKHR structure that contains the data to create the pipeline binaries from.
- pAllocator controls host memory allocation as described in the Memory Allocation chapter.
- pBinaries is a pointer to a VkPipelineBinaryHandlesInfoKHR structure in which the resulting pipeline binaries are returned.

The implementation will attempt to create all pipeline binaries. If creation fails for any pipeline

binary, then:

- The corresponding entry in the `pPipelineBinaries` output array will be filled with VK_NULL_HANDLE.
- The `VkResult` returned by vkCreatePipelineBinariesKHR will contain the error value for the first entry in the output array in `pBinaries` containing VK_NULL_HANDLE.

### Valid Usage (Implicit)

- VUID-vkCreatePipelineBinariesKHR-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkCreatePipelineBinariesKHR-pCreateInfo-parameter
  `pCreateInfo` **must** be a valid pointer to a valid VkPipelineBinaryCreateInfoKHR structure
- VUID-vkCreatePipelineBinariesKHR-pAllocator-parameter
  If `pAllocator` is not `NULL`, `pAllocator` **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkCreatePipelineBinariesKHR-pBinaries-parameter
  `pBinaries` **must** be a valid pointer to a VkPipelineBinaryHandlesInfoKHR structure

### Return Codes

**Success**

- `VK_SUCCESS`
- `VK_INCOMPLETE`
- `VK_PIPELINE_BINARY_MISSING_KHR`

**Failure**

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_INITIALIZATION_FAILED`

The `VkPipelineBinaryHandlesInfoKHR` structure is defined as:

```
// Provided by VK_KHR_pipeline_binary
typedef struct VkPipelineBinaryHandlesInfoKHR {
    VkStructureType         sType;
    const void*             pNext;
    uint32_t                pipelineBinaryCount;
    VkPipelineBinaryKHR*    pPipelineBinaries;
} VkPipelineBinaryHandlesInfoKHR;
```

- `sType` is a VkStructureType value identifying this structure.

- pNext is NULL or a pointer to a structure extending this structure.
- pipelineBinaryCount is the number of binaries associated with this pipeline or the number of entries in the pPipelineBinaries array.
- pPipelineBinaries is NULL or a pointer to an array of VkPipelineBinaryKHR handles in which the resulting pipeline binaries are returned.

If pPipelineBinaries is NULL, the number of binaries that would be created is returned in pipelineBinaryCount. Otherwise, pipelineBinaryCount **must** be set to the number of entries in the pPipelineBinaries array, and on return from vkCreatePipelineBinariesKHR pipelineBinaryCount is overwritten with the number of handles actually written to pPipelineBinaries. If the value of pipelineBinaryCount is less than the number of binaries that would have been created, at most pipelineBinaryCount handles will be written to pPipelineBinaries and VK_INCOMPLETE will be returned instead of VK_SUCCESS, to indicate that pPipelineBinaries was not large enough to create all the binaries.

### Valid Usage (Implicit)

- VUID-VkPipelineBinaryHandlesInfoKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PIPELINE_BINARY_HANDLES_INFO_KHR
- VUID-VkPipelineBinaryHandlesInfoKHR-pNext-pNext
  pNext **must** be NULL
- VUID-VkPipelineBinaryHandlesInfoKHR-pPipelineBinaries-parameter
  If pipelineBinaryCount is not 0, and pPipelineBinaries is not NULL, pPipelineBinaries **must** be a valid pointer to an array of pipelineBinaryCount VkPipelineBinaryKHR handles

The VkPipelineBinaryCreateInfoKHR structure is defined as:

```
// Provided by VK_KHR_pipeline_binary
typedef struct VkPipelineBinaryCreateInfoKHR {
    VkStructureType                          sType;
    const void*                              pNext;
    const VkPipelineBinaryKeysAndDataKHR*    pKeysAndDataInfo;
    VkPipeline                               pipeline;
    const VkPipelineCreateInfoKHR*           pPipelineCreateInfo;
} VkPipelineBinaryCreateInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- pKeysAndDataInfo is NULL or a pointer to a VkPipelineBinaryKeysAndDataKHR structure that contains keys and data to create the pipeline binaries from.
- pipeline is VK_NULL_HANDLE or a VkPipeline that contains data to create the pipeline binaries from.
- pPipelineCreateInfo is NULL or a pointer to a VkPipelineCreateInfoKHR structure with the pipeline creation info. This is used to probe the implementation's internal cache for pipeline

binaries.

When pPipelineCreateInfo is not NULL, an implementation will attempt to retrieve pipeline binary data from an internal cache external to the application if pipelineBinaryInternalCache is VK_TRUE. Applications **can** use this to determine if a pipeline **can** be created without compilation. If the implementation fails to create a pipeline binary due to missing an internal cache entry, VK_PIPELINE_BINARY_MISSING_KHR is returned. If creation succeeds, the resulting binary **can** be used to create a pipeline. VK_PIPELINE_BINARY_MISSING_KHR **may** be returned for any reason in this situation, even if creating a pipeline binary with the same parameters that succeeded earlier.

If pipelineBinaryPrecompiledInternalCache is VK_TRUE, the implementation **may** be able to create pipeline binaries even when pPipelineCreateInfo has not been used to create binaries before by the application.

| | |
|---|---|
| **NOTE** | On some platforms, internal pipeline caches may be pre-populated before running the application. |

## Valid Usage

- VUID-VkPipelineBinaryCreateInfoKHR-pipeline-09607
  If pipeline is not VK_NULL_HANDLE, pipeline **must** have been created with VK_PIPELINE_CREATE_2_CAPTURE_DATA_BIT_KHR
- VUID-VkPipelineBinaryCreateInfoKHR-pipeline-09608
  If pipeline is not VK_NULL_HANDLE, vkReleaseCapturedPipelineDataKHR **must** not have been called on pipeline prior to this command
- VUID-VkPipelineBinaryCreateInfoKHR-pipelineBinaryInternalCache-09609
  If pipelineBinaryInternalCache is VK_FALSE pPipelineCreateInfo **must** be NULL
- VUID-VkPipelineBinaryCreateInfoKHR-device-09610
  If device was created with VkDevicePipelineBinaryInternalCacheControlKHR::disableInternalCache set to VK_TRUE, pPipelineCreateInfo **must** be NULL
- VUID-VkPipelineBinaryCreateInfoKHR-pKeysAndDataInfo-09619
  One and only one of pKeysAndDataInfo, pipeline, or pPipelineCreateInfo **must** be non-NULL
- VUID-VkPipelineBinaryCreateInfoKHR-pPipelineCreateInfo-09606
  If pPipelineCreateInfo is not NULL, the pNext chain of pPipelineCreateInfo **must** not set VkPipelineBinaryInfoKHR::binaryCount to a value greater than 0

## Valid Usage (Implicit)

- VUID-VkPipelineBinaryCreateInfoKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PIPELINE_BINARY_CREATE_INFO_KHR
- VUID-VkPipelineBinaryCreateInfoKHR-pNext-pNext
  pNext **must** be NULL
- VUID-VkPipelineBinaryCreateInfoKHR-pKeysAndDataInfo-parameter
  If pKeysAndDataInfo is not NULL, pKeysAndDataInfo **must** be a valid pointer to a valid

VkPipelineBinaryKeysAndDataKHR structure

- VUID-VkPipelineBinaryCreateInfoKHR-pipeline-parameter
  If pipeline is not VK_NULL_HANDLE, pipeline **must** be a valid VkPipeline handle
- VUID-VkPipelineBinaryCreateInfoKHR-pPipelineCreateInfo-parameter
  If pPipelineCreateInfo is not NULL, pPipelineCreateInfo **must** be a valid pointer to a valid VkPipelineCreateInfoKHR structure

The VkPipelineBinaryKeysAndDataKHR structure is defined as:

```
// Provided by VK_KHR_pipeline_binary
typedef struct VkPipelineBinaryKeysAndDataKHR {
    uint32_t                            binaryCount;
    const VkPipelineBinaryKeyKHR*       pPipelineBinaryKeys;
    const VkPipelineBinaryDataKHR*      pPipelineBinaryData;
} VkPipelineBinaryKeysAndDataKHR;
```

- binaryCount is the size of the pPipelineBinaryKeys and pPipelineBinaryData arrays
- pPipelineBinaryKeys is a pointer to an array of VkPipelineBinaryKeyKHR structures containing the pipeline binary keys
- pPipelineBinaryData is a pointer to an array of VkPipelineBinaryDataKHR structures containing the pipeline binary data

### Valid Usage (Implicit)

- VUID-VkPipelineBinaryKeysAndDataKHR-pPipelineBinaryKeys-parameter
  pPipelineBinaryKeys **must** be a valid pointer to an array of binaryCount valid VkPipelineBinaryKeyKHR structures
- VUID-VkPipelineBinaryKeysAndDataKHR-pPipelineBinaryData-parameter
  pPipelineBinaryData **must** be a valid pointer to an array of binaryCount valid VkPipelineBinaryDataKHR structures
- VUID-VkPipelineBinaryKeysAndDataKHR-binaryCount-arraylength
  binaryCount **must** be greater than 0

The VkPipelineBinaryDataKHR structure is defined as:

```
// Provided by VK_KHR_pipeline_binary
typedef struct VkPipelineBinaryDataKHR {
    size_t    dataSize;
    void*     pData;
} VkPipelineBinaryDataKHR;
```

- dataSize is the size of the pData buffer in bytes.

- pData is a pointer to a buffer of size bytes that contains pipeline binary data obtained from vkGetPipelineBinaryDataKHR.

**Valid Usage (Implicit)**

- VUID-VkPipelineBinaryDataKHR-pData-parameter
  pData **must** be a valid pointer to an array of dataSize bytes
- VUID-VkPipelineBinaryDataKHR-dataSize-arraylength
  dataSize **must** be greater than 0

### 10.8.3. Retrieving Pipeline Binary Data

Data **can** be retrieved from a pipeline binary object using the command:

```
// Provided by VK_KHR_pipeline_binary
VkResult vkGetPipelineBinaryDataKHR(
    VkDevice                                    device,
    const VkPipelineBinaryDataInfoKHR*          pInfo,
    VkPipelineBinaryKeyKHR*                     pPipelineBinaryKey,
    size_t*                                     pPipelineBinaryDataSize,
    void*                                       pPipelineBinaryData);
```

- device is the logical device that created the pipeline binary.
- pInfo is a pointer to a VkPipelineBinaryDataInfoKHR structure which describes the pipeline binary to get data from.
- pPipelineBinaryKey is a pointer to a VkPipelineBinaryKeyKHR structure where the key for this binary will be written.
- pPipelineBinaryDataSize is a pointer to a size_t value related to the amount of data in the pipeline binary, as described below.
- pPipelineBinaryData is either NULL or a pointer to a buffer.

If pPipelineBinaryData is NULL, then the size of the data, in bytes, that is required to store the binary is returned in pPipelineBinaryDataSize. Otherwise, pPipelineBinaryDataSize **must** contain the size of the buffer, in bytes, pointed to by pPipelineBinaryData, and on return pPipelineBinaryDataSize is overwritten with the size of the data, in bytes, that is required to store the binary. If pPipelineBinaryDataSize is less than the size that is required to store the binary, nothing is written to pPipelineBinaryData and VK_ERROR_NOT_ENOUGH_SPACE_KHR will be returned, instead of VK_SUCCESS.

If pipelineBinaryCompressedData is VK_FALSE, implementations **should** not return compressed pipeline binary data to the application.

**Valid Usage (Implicit)**

- VUID-vkGetPipelineBinaryDataKHR-device-parameter

device **must** be a valid VkDevice handle

- VUID-vkGetPipelineBinaryDataKHR-pInfo-parameter
  pInfo **must** be a valid pointer to a valid VkPipelineBinaryDataInfoKHR structure
- VUID-vkGetPipelineBinaryDataKHR-pPipelineBinaryKey-parameter
  pPipelineBinaryKey **must** be a valid pointer to a VkPipelineBinaryKeyKHR structure
- VUID-vkGetPipelineBinaryDataKHR-pPipelineBinaryDataSize-parameter
  pPipelineBinaryDataSize **must** be a valid pointer to a size_t value
- VUID-vkGetPipelineBinaryDataKHR-pPipelineBinaryData-parameter
  If the value referenced by pPipelineBinaryDataSize is not 0, and pPipelineBinaryData is not NULL, pPipelineBinaryData **must** be a valid pointer to an array of pPipelineBinaryDataSize bytes

## Return Codes

**Success**

- VK_SUCCESS

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY
- VK_ERROR_NOT_ENOUGH_SPACE_KHR

The VkPipelineBinaryDataInfoKHR structure is defined as:

```
// Provided by VK_KHR_pipeline_binary
typedef struct VkPipelineBinaryDataInfoKHR {
    VkStructureType        sType;
    void*                  pNext;
    VkPipelineBinaryKHR    pipelineBinary;
} VkPipelineBinaryDataInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- pipelineBinary is the pipeline binary to get data from.

## Valid Usage (Implicit)

- VUID-VkPipelineBinaryDataInfoKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PIPELINE_BINARY_DATA_INFO_KHR
- VUID-VkPipelineBinaryDataInfoKHR-pNext-pNext
  pNext **must** be NULL
- VUID-VkPipelineBinaryDataInfoKHR-pipelineBinary-parameter

pipelineBinary **must** be a valid VkPipelineBinaryKHR handle

### 10.8.4. Releasing Captured Pipeline Binary Data

To release pipeline resources captured with VK_PIPELINE_CREATE_2_CAPTURE_DATA_BIT_KHR, call:

```
// Provided by VK_KHR_pipeline_binary
VkResult vkReleaseCapturedPipelineDataKHR(
    VkDevice                                    device,
    const VkReleaseCapturedPipelineDataInfoKHR* pInfo,
    const VkAllocationCallbacks*                pAllocator);
```

- device is the logical device that created the pipeline object.
- pInfo is a pointer to a VkReleaseCapturedPipelineDataInfoKHR structure which describes the pipeline to release the data from.
- pAllocator controls host memory allocation as described in the Memory Allocation chapter.

The implementation **may** free any resources captured as a result of creating the pipeline with VK_PIPELINE_CREATE_2_CAPTURE_DATA_BIT_KHR and put the pipeline into a state as if VK_PIPELINE_CREATE_2_CAPTURE_DATA_BIT_KHR had not been provided at pipeline creation time.

| | |
|---|---|
| **NOTE** | Any resources captured as a result of creating the pipeline with VK_PIPELINE_CREATE_2_CAPTURE_DATA_BIT_KHR are implicitly freed by vkDestroyPipeline. |

#### Valid Usage

- VUID-vkReleaseCapturedPipelineDataKHR-pipeline-09611
  If VkAllocationCallbacks were provided when pipeline was created, a compatible set of callbacks **must** be provided in pAllocator
- VUID-vkReleaseCapturedPipelineDataKHR-pipeline-09612
  If no VkAllocationCallbacks were provided when pipeline was created, pAllocator **must** be NULL

#### Valid Usage (Implicit)

- VUID-vkReleaseCapturedPipelineDataKHR-device-parameter
  device **must** be a valid VkDevice handle
- VUID-vkReleaseCapturedPipelineDataKHR-pInfo-parameter
  pInfo **must** be a valid pointer to a valid VkReleaseCapturedPipelineDataInfoKHR structure
- VUID-vkReleaseCapturedPipelineDataKHR-pAllocator-parameter
  If pAllocator is not NULL, pAllocator **must** be a valid pointer to a valid VkAllocationCallbacks structure

## Host Synchronization

- Host access to `pInfo->pipeline` **must** be externally synchronized

## Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

None

The `VkReleaseCapturedPipelineDataInfoKHR` structure is defined as:

```
// Provided by VK_KHR_pipeline_binary
typedef struct VkReleaseCapturedPipelineDataInfoKHR {
    VkStructureType    sType;
    void*              pNext;
    VkPipeline         pipeline;
} VkReleaseCapturedPipelineDataInfoKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `pipeline` the handle of the pipeline object to release the data from.

## Valid Usage

- VUID-VkReleaseCapturedPipelineDataInfoKHR-pipeline-09613
  `pipeline` **must** have been created with `VK_PIPELINE_CREATE_2_CAPTURE_DATA_BIT_KHR`
- VUID-VkReleaseCapturedPipelineDataInfoKHR-pipeline-09618
  `pipeline` **must** not have been used in a previous call to `vkReleaseCapturedPipelineDataKHR`

## Valid Usage (Implicit)

- VUID-VkReleaseCapturedPipelineDataInfoKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_RELEASE_CAPTURED_PIPELINE_DATA_INFO_KHR`
- VUID-VkReleaseCapturedPipelineDataInfoKHR-pNext-pNext
  `pNext` **must** be `NULL`
- VUID-VkReleaseCapturedPipelineDataInfoKHR-pipeline-parameter
  `pipeline` **must** be a valid VkPipeline handle

### 10.8.5. Destroying Pipeline Binaries

To destroy a VkPipelineBinaryKHR, call:

```
// Provided by VK_KHR_pipeline_binary
void vkDestroyPipelineBinaryKHR(
    VkDevice                                    device,
    VkPipelineBinaryKHR                         pipelineBinary,
    const VkAllocationCallbacks*                pAllocator);
```

- `device` is the logical device that created the pipeline binary object.
- `pipelineBinary` is the handle of the pipeline binary object to destroy.
- `pAllocator` controls host memory allocation as described in the Memory Allocation chapter.

#### Valid Usage

- VUID-vkDestroyPipelineBinaryKHR-pipelineBinary-09614
  If `VkAllocationCallbacks` were provided when `pipelineBinary` was created, a compatible set of callbacks **must** be provided here.
- VUID-vkDestroyPipelineBinaryKHR-pipelineBinary-09615
  If no VkAllocationCallbacks were provided when `pipelineBinary` was created, `pAllocator` **must** be `NULL`

#### Valid Usage (Implicit)

- VUID-vkDestroyPipelineBinaryKHR-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkDestroyPipelineBinaryKHR-pipelineBinary-parameter
  If `pipelineBinary` is not VK_NULL_HANDLE, `pipelineBinary` **must** be a valid VkPipelineBinaryKHR handle
- VUID-vkDestroyPipelineBinaryKHR-pAllocator-parameter
  If `pAllocator` is not `NULL`, `pAllocator` **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkDestroyPipelineBinaryKHR-pipelineBinary-parent
  If `pipelineBinary` is a valid handle, it **must** have been created, allocated, or retrieved from `device`

#### Host Synchronization

- Host access to `pipelineBinary` **must** be externally synchronized

## 10.9. Specialization Constants

Specialization constants are a mechanism whereby constants in a SPIR-V module **can** have their constant value specified at the time the `VkPipeline` is created. This allows a SPIR-V module to have constants that **can** be modified while executing an application that uses the Vulkan API.

| NOTE | Specialization constants are useful to allow a compute shader to have its local workgroup size changed at runtime by the user, for example. |
|---|---|

Each VkPipelineShaderStageCreateInfo structure contains a `pSpecializationInfo` member, which **can** be `NULL` to indicate no specialization constants, or point to a `VkSpecializationInfo` structure.

The `VkSpecializationInfo` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkSpecializationInfo {
    uint32_t                            mapEntryCount;
    const VkSpecializationMapEntry*     pMapEntries;
    size_t                              dataSize;
    const void*                         pData;
} VkSpecializationInfo;
```

- `mapEntryCount` is the number of entries in the `pMapEntries` array.
- `pMapEntries` is a pointer to an array of `VkSpecializationMapEntry` structures, which map constant IDs to offsets in `pData`.
- `dataSize` is the byte size of the `pData` buffer.
- `pData` contains the actual constant values to specialize with.

### Valid Usage

- VUID-VkSpecializationInfo-offset-00773
  The `offset` member of each element of `pMapEntries` **must** be less than `dataSize`
- VUID-VkSpecializationInfo-pMapEntries-00774
  The `size` member of each element of `pMapEntries` **must** be less than or equal to `dataSize` minus `offset`
- VUID-VkSpecializationInfo-constantID-04911
  The `constantID` value of each element of `pMapEntries` **must** be unique within `pMapEntries`

### Valid Usage (Implicit)

- VUID-VkSpecializationInfo-pMapEntries-parameter
  If `mapEntryCount` is not `0`, `pMapEntries` **must** be a valid pointer to an array of `mapEntryCount` valid VkSpecializationMapEntry structures

- VUID-VkSpecializationInfo-pData-parameter
  If `dataSize` is not `0`, `pData` **must** be a valid pointer to an array of `dataSize` bytes

The `VkSpecializationMapEntry` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkSpecializationMapEntry {
    uint32_t    constantID;
    uint32_t    offset;
    size_t      size;
} VkSpecializationMapEntry;
```

- `constantID` is the ID of the specialization constant in SPIR-V.
- `offset` is the byte offset of the specialization constant value within the supplied data buffer.
- `size` is the byte size of the specialization constant value within the supplied data buffer.

If a `constantID` value is not a specialization constant ID used in the shader, that map entry does not affect the behavior of the pipeline.

### Valid Usage

- VUID-VkSpecializationMapEntry-constantID-00776
  For a `constantID` specialization constant declared in a shader, `size` **must** match the byte size of the `constantID`. If the specialization constant is of type `boolean`, `size` **must** be the byte size of VkBool32

In human readable SPIR-V:

```
OpDecorate %x SpecId 13 ; decorate .x component of WorkgroupSize with ID 13
OpDecorate %y SpecId 42 ; decorate .y component of WorkgroupSize with ID 42
OpDecorate %z SpecId 3  ; decorate .z component of WorkgroupSize with ID 3
OpDecorate %wgsize BuiltIn WorkgroupSize ; decorate WorkgroupSize onto constant
%i32 = OpTypeInt 32 0 ; declare an unsigned 32-bit type
%uvec3 = OpTypeVector %i32 3 ; declare a 3 element vector type of unsigned 32-bit
%x = OpSpecConstant %i32 1 ; declare the .x component of WorkgroupSize
%y = OpSpecConstant %i32 1 ; declare the .y component of WorkgroupSize
%z = OpSpecConstant %i32 1 ; declare the .z component of WorkgroupSize
%wgsize = OpSpecConstantComposite %uvec3 %x %y %z ; declare WorkgroupSize
```

From the above we have three specialization constants, one for each of the x, y & z elements of the WorkgroupSize vector.

Now to specialize the above via the specialization constants mechanism:

```
const VkSpecializationMapEntry entries[] =
```

```
{
    {
        .constantID = 13,
        .offset = 0 * sizeof(uint32_t),
        .size = sizeof(uint32_t)
    },
    {
        .constantID = 42,
        .offset = 1 * sizeof(uint32_t),
        .size = sizeof(uint32_t)
    },
    {
        .constantID = 3,
        .offset = 2 * sizeof(uint32_t),
        .size = sizeof(uint32_t)
    }
};

const uint32_t data[] = { 16, 8, 4 }; // our workgroup size is 16x8x4

const VkSpecializationInfo info =
{
    .mapEntryCount = 3,
    .pMapEntries  = entries,
    .dataSize = 3 * sizeof(uint32_t),
    .pData = data,
};
```

Then when calling vkCreateComputePipelines, and passing the `VkSpecializationInfo` we defined as the `pSpecializationInfo` parameter of VkPipelineShaderStageCreateInfo, we will create a compute pipeline with the runtime specified local workgroup size.

Another example would be that an application has a SPIR-V module that has some platform-dependent constants they wish to use.

In human readable SPIR-V:

```
OpDecorate %1 SpecId 0  ; decorate our signed 32-bit integer constant
OpDecorate %2 SpecId 12 ; decorate our 32-bit floating-point constant
%i32 = OpTypeInt 32 1   ; declare a signed 32-bit type
%float = OpTypeFloat 32 ; declare a 32-bit floating-point type
%1 = OpSpecConstant %i32 -1 ; some signed 32-bit integer constant
%2 = OpSpecConstant %float 0.5 ; some 32-bit floating-point constant
```

From the above we have two specialization constants, one is a signed 32-bit integer and the second is a 32-bit floating-point value.

Now to specialize the above via the specialization constants mechanism:

```
struct SpecializationData {
    int32_t data0;
    float data1;
};

const VkSpecializationMapEntry entries[] =
{
    {
        .constantID = 0,
        .offset = offsetof(SpecializationData, data0),
        .size = sizeof(SpecializationData::data0)
    },
    {
        .constantID = 12,
        .offset = offsetof(SpecializationData, data1),
        .size = sizeof(SpecializationData::data1)
    }
};

SpecializationData data;
data.data0 = -42;    // set the data for the 32-bit integer
data.data1 = 42.0f;  // set the data for the 32-bit floating-point

const VkSpecializationInfo info =
{
    .mapEntryCount = 2,
    .pMapEntries = entries,
    .dataSize = sizeof(data),
    .pdata = &data,
};
```

It is legal for a SPIR-V module with specializations to be compiled into a pipeline where no specialization information was provided. SPIR-V specialization constants contain default values such that if a specialization is not provided, the default value will be used. In the examples above, it would be valid for an application to only specialize some of the specialization constants within the SPIR-V module, and let the other constants use their default values encoded within the OpSpecConstant declarations.

## 10.10. Pipeline Libraries

A pipeline library is a special pipeline that was created using the VK_PIPELINE_CREATE_LIBRARY_BIT_KHR and cannot be bound, instead it defines a set of pipeline state which can be linked into other pipelines. For ray tracing pipelines this includes shaders and shader groups. For graphics pipelines this includes distinct library types defined by VkGraphicsPipelineLibraryFlagBitsEXT. The application **must** maintain the lifetime of a pipeline library based on the pipelines that link with it.

This linkage is achieved by using the following structure within the appropriate creation

mechanisms:

The VkPipelineLibraryCreateInfoKHR structure is defined as:

```
// Provided by VK_KHR_pipeline_library
typedef struct VkPipelineLibraryCreateInfoKHR {
    VkStructureType      sType;
    const void*          pNext;
    uint32_t             libraryCount;
    const VkPipeline*    pLibraries;
} VkPipelineLibraryCreateInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- libraryCount is the number of pipeline libraries in pLibraries.
- pLibraries is a pointer to an array of VkPipeline structures specifying pipeline libraries to use when creating a pipeline.

### Valid Usage

- VUID-VkPipelineLibraryCreateInfoKHR-pLibraries-03381
  Each element of pLibraries **must** have been created with VK_PIPELINE_CREATE_LIBRARY_BIT_KHR
- VUID-VkPipelineLibraryCreateInfoKHR-pLibraries-06855
  If any library in pLibraries was created with a shader stage with VkPipelineShaderStageModuleIdentifierCreateInfoEXT and identifierSize not equal to 0, the pipeline **must** be created with the VK_PIPELINE_CREATE_FAIL_ON_PIPELINE_COMPILE_REQUIRED_BIT flag set
- VUID-VkPipelineLibraryCreateInfoKHR-pLibraries-08096
  If any element of pLibraries was created with VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT, all elements **must** have been created with VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT
- VUID-VkPipelineLibraryCreateInfoKHR-pipeline-07404
  If pipeline is being created with VK_PIPELINE_CREATE_NO_PROTECTED_ACCESS_BIT_EXT, every element of pLibraries **must** have been created with VK_PIPELINE_CREATE_NO_PROTECTED_ACCESS_BIT_EXT
- VUID-VkPipelineLibraryCreateInfoKHR-pipeline-07405
  If pipeline is being created without VK_PIPELINE_CREATE_NO_PROTECTED_ACCESS_BIT_EXT, every element of pLibraries **must** have been created without VK_PIPELINE_CREATE_NO_PROTECTED_ACCESS_BIT_EXT
- VUID-VkPipelineLibraryCreateInfoKHR-pipeline-07406
  If pipeline is being created with VK_PIPELINE_CREATE_PROTECTED_ACCESS_ONLY_BIT_EXT, every element of pLibraries **must** have been created with VK_PIPELINE_CREATE_PROTECTED_ACCESS_ONLY_BIT_EXT

- VUID-VkPipelineLibraryCreateInfoKHR-pipeline-07407
  If `pipeline` is being created without `VK_PIPELINE_CREATE_PROTECTED_ACCESS_ONLY_BIT_EXT`, every element of `pLibraries` **must** have been created without `VK_PIPELINE_CREATE_PROTECTED_ACCESS_ONLY_BIT_EXT`

### Valid Usage (Implicit)

- VUID-VkPipelineLibraryCreateInfoKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PIPELINE_LIBRARY_CREATE_INFO_KHR`
- VUID-VkPipelineLibraryCreateInfoKHR-pLibraries-parameter
  If `libraryCount` is not `0`, `pLibraries` **must** be a valid pointer to an array of `libraryCount` valid VkPipeline handles

Pipelines created with `VK_PIPELINE_CREATE_LIBRARY_BIT_KHR` libraries **can** depend on other pipeline libraries in VkPipelineLibraryCreateInfoKHR.

A pipeline library is considered in-use, as long as one of the linking pipelines is in-use. This applies recursively if a pipeline library includes other pipeline libraries.

## 10.11. Pipeline Binding

Once a pipeline has been created, it **can** be bound to the command buffer using the command:

```
// Provided by VK_VERSION_1_0
void vkCmdBindPipeline(
    VkCommandBuffer                             commandBuffer,
    VkPipelineBindPoint                         pipelineBindPoint,
    VkPipeline                                  pipeline);
```

- `commandBuffer` is the command buffer that the pipeline will be bound to.
- `pipelineBindPoint` is a VkPipelineBindPoint value specifying to which bind point the pipeline is bound. Binding one does not disturb the others.
- `pipeline` is the pipeline to be bound.

Once bound, a pipeline binding affects subsequent commands that interact with the given pipeline type in the command buffer until a different pipeline of the same type is bound to the bind point, or until the pipeline bind point is disturbed by binding a shader object as described in Interaction with Pipelines. Commands that do not interact with the given pipeline type **must** not be affected by the pipeline state.

### Valid Usage

- VUID-vkCmdBindPipeline-pipelineBindPoint-00777
  If `pipelineBindPoint` is `VK_PIPELINE_BIND_POINT_COMPUTE`, the `VkCommandPool` that

commandBuffer was allocated from **must** support compute operations

- VUID-vkCmdBindPipeline-pipelineBindPoint-00778
  If pipelineBindPoint is VK_PIPELINE_BIND_POINT_GRAPHICS, the VkCommandPool that commandBuffer was allocated from **must** support graphics operations
- VUID-vkCmdBindPipeline-pipelineBindPoint-00779
  If pipelineBindPoint is VK_PIPELINE_BIND_POINT_COMPUTE, pipeline **must** be a compute pipeline
- VUID-vkCmdBindPipeline-pipelineBindPoint-00780
  If pipelineBindPoint is VK_PIPELINE_BIND_POINT_GRAPHICS, pipeline **must** be a graphics pipeline
- VUID-vkCmdBindPipeline-pipeline-00781
  If the variableMultisampleRate feature is not supported, pipeline is a graphics pipeline, the current subpass uses no attachments, and this is not the first call to this function with a graphics pipeline after transitioning to the current subpass, then the sample count specified by this pipeline **must** match that set in the previous pipeline
- VUID-vkCmdBindPipeline-variableSampleLocations-01525
  If VkPhysicalDeviceSampleLocationsPropertiesEXT::variableSampleLocations is VK_FALSE, and pipeline is a graphics pipeline created with a VkPipelineSampleLocationsStateCreateInfoEXT structure having its sampleLocationsEnable member set to VK_TRUE but without VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT enabled then the current render pass instance **must** have been begun by specifying a VkRenderPassSampleLocationsBeginInfoEXT structure whose pPostSubpassSampleLocations member contains an element with a subpassIndex matching the current subpass index and the sampleLocationsInfo member of that element **must** match the sampleLocationsInfo specified in VkPipelineSampleLocationsStateCreateInfoEXT when the pipeline was created
- VUID-vkCmdBindPipeline-None-02323
  This command **must** not be recorded when transform feedback is active
- VUID-vkCmdBindPipeline-pipelineBindPoint-02391
  If pipelineBindPoint is VK_PIPELINE_BIND_POINT_RAY_TRACING_KHR, the VkCommandPool that commandBuffer was allocated from **must** support compute operations
- VUID-vkCmdBindPipeline-pipelineBindPoint-02392
  If pipelineBindPoint is VK_PIPELINE_BIND_POINT_RAY_TRACING_KHR, pipeline **must** be a ray tracing pipeline
- VUID-vkCmdBindPipeline-pipelineBindPoint-06721
  If pipelineBindPoint is VK_PIPELINE_BIND_POINT_RAY_TRACING_KHR, commandBuffer **must** not be a protected command buffer
- VUID-vkCmdBindPipeline-pipelineProtectedAccess-07408
  If the pipelineProtectedAccess feature is enabled, and commandBuffer is a protected command buffer, pipeline **must** have been created without VK_PIPELINE_CREATE_NO_PROTECTED_ACCESS_BIT_EXT
- VUID-vkCmdBindPipeline-pipelineProtectedAccess-07409
  If the pipelineProtectedAccess feature is enabled, and commandBuffer is not a protected

command buffer, pipeline **must** have been created without VK_PIPELINE_CREATE_PROTECTED_ACCESS_ONLY_BIT_EXT

- VUID-vkCmdBindPipeline-pipeline-03382
  pipeline **must** not have been created with VK_PIPELINE_CREATE_LIBRARY_BIT_KHR set
- VUID-vkCmdBindPipeline-commandBuffer-04808
  If commandBuffer is a secondary command buffer with VkCommandBufferInheritanceViewportScissorInfoNV::viewportScissor2D enabled and pipelineBindPoint is VK_PIPELINE_BIND_POINT_GRAPHICS, then the pipeline **must** have been created with VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT or VK_DYNAMIC_STATE_VIEWPORT, and VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT or VK_DYNAMIC_STATE_SCISSOR enabled
- VUID-vkCmdBindPipeline-commandBuffer-04809
  If commandBuffer is a secondary command buffer with VkCommandBufferInheritanceViewportScissorInfoNV::viewportScissor2D enabled and pipelineBindPoint is VK_PIPELINE_BIND_POINT_GRAPHICS and pipeline was created with VkPipelineDiscardRectangleStateCreateInfoEXT structure and its discardRectangleCount member is not 0, or the pipeline was created with VK_DYNAMIC_STATE_DISCARD_RECTANGLE_ENABLE_EXT enabled, then the pipeline **must** have been created with VK_DYNAMIC_STATE_DISCARD_RECTANGLE_EXT enabled
- VUID-vkCmdBindPipeline-pipelineBindPoint-04881
  If pipelineBindPoint is VK_PIPELINE_BIND_POINT_GRAPHICS and the provokingVertexModePerPipeline limit is VK_FALSE, then pipeline's VkPipelineRasterizationProvokingVertexStateCreateInfoEXT::provokingVertexMode **must** be the same as that of any other pipelines previously bound to this bind point within the current render pass instance, including any pipeline already bound when beginning the render pass instance
- VUID-vkCmdBindPipeline-pipelineBindPoint-04949
  If pipelineBindPoint is VK_PIPELINE_BIND_POINT_SUBPASS_SHADING_HUAWEI, the VkCommandPool that commandBuffer was allocated from **must** support compute operations
- VUID-vkCmdBindPipeline-pipelineBindPoint-04950
  If pipelineBindPoint is VK_PIPELINE_BIND_POINT_SUBPASS_SHADING_HUAWEI, pipeline **must** be a subpass shading pipeline

## Valid Usage (Implicit)

- VUID-vkCmdBindPipeline-commandBuffer-parameter
  commandBuffer **must** be a valid VkCommandBuffer handle
- VUID-vkCmdBindPipeline-pipelineBindPoint-parameter
  pipelineBindPoint **must** be a valid VkPipelineBindPoint value
- VUID-vkCmdBindPipeline-pipeline-parameter
  pipeline **must** be a valid VkPipeline handle
- VUID-vkCmdBindPipeline-commandBuffer-recording
  commandBuffer **must** be in the recording state
- VUID-vkCmdBindPipeline-commandBuffer-cmdpool

The VkCommandPool that commandBuffer was allocated from **must** support graphics, or compute operations

- VUID-vkCmdBindPipeline-videocoding
  This command **must** only be called outside of a video coding scope
- VUID-vkCmdBindPipeline-commonparent
  Both of commandBuffer, and pipeline **must** have been created, allocated, or retrieved from the same VkDevice

## Host Synchronization

- Host access to commandBuffer **must** be externally synchronized
- Host access to the VkCommandPool that commandBuffer was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics<br>Compute | State |

Possible values of vkCmdBindPipeline::pipelineBindPoint, specifying the bind point of a pipeline object, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkPipelineBindPoint {
    VK_PIPELINE_BIND_POINT_GRAPHICS = 0,
    VK_PIPELINE_BIND_POINT_COMPUTE = 1,
#ifdef VK_ENABLE_BETA_EXTENSIONS
  // Provided by VK_AMDX_shader_enqueue
    VK_PIPELINE_BIND_POINT_EXECUTION_GRAPH_AMDX = 1000134000,
#endif
  // Provided by VK_KHR_ray_tracing_pipeline
    VK_PIPELINE_BIND_POINT_RAY_TRACING_KHR = 1000165000,
  // Provided by VK_HUAWEI_subpass_shading
    VK_PIPELINE_BIND_POINT_SUBPASS_SHADING_HUAWEI = 1000369003,
  // Provided by VK_NV_ray_tracing
    VK_PIPELINE_BIND_POINT_RAY_TRACING_NV = VK_PIPELINE_BIND_POINT_RAY_TRACING_KHR,
} VkPipelineBindPoint;
```

- VK_PIPELINE_BIND_POINT_COMPUTE specifies binding as a compute pipeline.
- VK_PIPELINE_BIND_POINT_GRAPHICS specifies binding as a graphics pipeline.

- VK_PIPELINE_BIND_POINT_RAY_TRACING_KHR specifies binding as a ray tracing pipeline.
- VK_PIPELINE_BIND_POINT_SUBPASS_SHADING_HUAWEI specifies binding as a subpass shading pipeline.
- VK_PIPELINE_BIND_POINT_EXECUTION_GRAPH_AMDX specifies binding as an execution graph pipeline.

For pipelines that were created with the support of multiple shader groups (see Graphics Pipeline Shader Groups), the regular vkCmdBindPipeline command will bind Shader Group 0. To explicitly bind a shader group use:

```
// Provided by VK_NV_device_generated_commands
void vkCmdBindPipelineShaderGroupNV(
    VkCommandBuffer                             commandBuffer,
    VkPipelineBindPoint                         pipelineBindPoint,
    VkPipeline                                  pipeline,
    uint32_t                                    groupIndex);
```

- commandBuffer is the command buffer that the pipeline will be bound to.
- pipelineBindPoint is a VkPipelineBindPoint value specifying the bind point to which the pipeline will be bound.
- pipeline is the pipeline to be bound.
- groupIndex is the shader group to be bound.

### Valid Usage

- VUID-vkCmdBindPipelineShaderGroupNV-groupIndex-02893
  groupIndex **must** be 0 or less than the effective VkGraphicsPipelineShaderGroupsCreateInfoNV::groupCount including the referenced pipelines
- VUID-vkCmdBindPipelineShaderGroupNV-pipelineBindPoint-02894
  The pipelineBindPoint **must** be VK_PIPELINE_BIND_POINT_GRAPHICS
- VUID-vkCmdBindPipelineShaderGroupNV-groupIndex-02895
  The same restrictions as vkCmdBindPipeline apply as if the bound pipeline was created only with the Shader Group from the groupIndex information
- VUID-vkCmdBindPipelineShaderGroupNV-deviceGeneratedCommands-02896
  The deviceGeneratedCommands feature **must** be enabled

### Valid Usage (Implicit)

- VUID-vkCmdBindPipelineShaderGroupNV-commandBuffer-parameter
  commandBuffer **must** be a valid VkCommandBuffer handle
- VUID-vkCmdBindPipelineShaderGroupNV-pipelineBindPoint-parameter
  pipelineBindPoint **must** be a valid VkPipelineBindPoint value
- VUID-vkCmdBindPipelineShaderGroupNV-pipeline-parameter

pipeline **must** be a valid VkPipeline handle

- VUID-vkCmdBindPipelineShaderGroupNV-commandBuffer-recording
  commandBuffer **must** be in the recording state
- VUID-vkCmdBindPipelineShaderGroupNV-commandBuffer-cmdpool
  The VkCommandPool that commandBuffer was allocated from **must** support graphics, or compute operations
- VUID-vkCmdBindPipelineShaderGroupNV-videocoding
  This command **must** only be called outside of a video coding scope
- VUID-vkCmdBindPipelineShaderGroupNV-commonparent
  Both of commandBuffer, and pipeline **must** have been created, allocated, or retrieved from the same VkDevice

### Host Synchronization

- Host access to commandBuffer **must** be externally synchronized
- Host access to the VkCommandPool that commandBuffer was allocated from **must** be externally synchronized

### Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics<br>Compute | State |

### 10.11.1. Interaction With Shader Objects

If the shaderObject feature is enabled, applications **can** use both pipelines and shader objects at the same time. The interaction between pipelines and shader objects is described in Interaction with Pipelines.

## 10.12. Dynamic State

When a pipeline object is bound, any pipeline object state that is not specified as dynamic is applied to the command buffer state. Pipeline object state that is specified as dynamic is not applied to the command buffer state at this time.

Instead, dynamic state **can** be modified at any time and persists for the lifetime of the command buffer, or until modified by another dynamic state setting command, or made invalid by binding a pipeline in which that state is statically specified.

If the commandBufferInheritance feature is enabled, all valid state from the previously executed

command buffer in the queue is inherited into the next command buffer executed in the same queue. This inherited state does not need to be set again before draw or dispatch commands.

When a pipeline object is bound, the following applies to each state parameter:

- If the state is not specified as dynamic in the new pipeline object, then that command buffer state is overwritten by the state in the new pipeline object. Before any draw or dispatch call with this pipeline there **must** not have been any calls to any of the corresponding dynamic state setting commands after this pipeline was bound.
- If the state is specified as dynamic in the new pipeline object, then that command buffer state is not disturbed. Before any draw or dispatch call with this pipeline there **must** have been at least one call to each of the corresponding dynamic state setting commands. The state-setting commands **must** be recorded after command buffer recording was begun, or after the last command binding a pipeline object with that state specified as static, whichever was the latter.
- If the state is not included (corresponding pointer in VkGraphicsPipelineCreateInfo was `NULL` or was ignored) in the new pipeline object, then that command buffer state is not disturbed. For example, mesh shading pipelines do not include vertex input state and therefore do not disturb any such command buffer state.

Dynamic state that does not affect the result of operations **can** be left undefined.

| | |
|---|---|
| **NOTE** | For example, if blending is disabled by the pipeline object state then the dynamic color blend constants do not need to be specified in the command buffer, even if this state is specified as dynamic in the pipeline object. |
| **NOTE** | Applications running on Vulkan implementations advertising an VkPhysicalDeviceDriverProperties::`conformanceVersion` less than 1.3.8.0 should be aware that rebinding the currently bound pipeline object may not reapply static state. |

## 10.13. Pipeline Properties and Shader Information

When a pipeline is created, its state and shaders are compiled into zero or more device-specific executables, which are used when executing commands against that pipeline. To query the properties of these pipeline executables, call:

```
// Provided by VK_KHR_pipeline_executable_properties
VkResult vkGetPipelineExecutablePropertiesKHR(
    VkDevice                                    device,
    const VkPipelineInfoKHR*                    pPipelineInfo,
    uint32_t*                                   pExecutableCount,
    VkPipelineExecutablePropertiesKHR*          pProperties);
```

- `device` is the device that created the pipeline.
- `pPipelineInfo` describes the pipeline being queried.

- pExecutableCount is a pointer to an integer related to the number of pipeline executables available or queried, as described below.
- pProperties is either NULL or a pointer to an array of VkPipelineExecutablePropertiesKHR structures.

If pProperties is NULL, then the number of pipeline executables associated with the pipeline is returned in pExecutableCount. Otherwise, pExecutableCount **must** point to a variable set by the application to the number of elements in the pProperties array, and on return the variable is overwritten with the number of structures actually written to pProperties. If pExecutableCount is less than the number of pipeline executables associated with the pipeline, at most pExecutableCount structures will be written, and VK_INCOMPLETE will be returned instead of VK_SUCCESS, to indicate that not all the available properties were returned.

## Valid Usage

- VUID-vkGetPipelineExecutablePropertiesKHR-pipelineExecutableInfo-03270
  The pipelineExecutableInfo feature **must** be enabled
- VUID-vkGetPipelineExecutablePropertiesKHR-pipeline-03271
  The pipeline member of pPipelineInfo **must** have been created with device

## Valid Usage (Implicit)

- VUID-vkGetPipelineExecutablePropertiesKHR-device-parameter
  device **must** be a valid VkDevice handle
- VUID-vkGetPipelineExecutablePropertiesKHR-pPipelineInfo-parameter
  pPipelineInfo **must** be a valid pointer to a valid VkPipelineInfoKHR structure
- VUID-vkGetPipelineExecutablePropertiesKHR-pExecutableCount-parameter
  pExecutableCount **must** be a valid pointer to a uint32_t value
- VUID-vkGetPipelineExecutablePropertiesKHR-pProperties-parameter
  If the value referenced by pExecutableCount is not 0, and pProperties is not NULL, pProperties **must** be a valid pointer to an array of pExecutableCount VkPipelineExecutablePropertiesKHR structures

## Return Codes

### Success

- VK_SUCCESS
- VK_INCOMPLETE

### Failure

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY

The VkPipelineExecutablePropertiesKHR structure is defined as:

```
// Provided by VK_KHR_pipeline_executable_properties
typedef struct VkPipelineExecutablePropertiesKHR {
    VkStructureType       sType;
    void*                 pNext;
    VkShaderStageFlags    stages;
    char                  name[VK_MAX_DESCRIPTION_SIZE];
    char                  description[VK_MAX_DESCRIPTION_SIZE];
    uint32_t              subgroupSize;
} VkPipelineExecutablePropertiesKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- stages is a bitmask of zero or more VkShaderStageFlagBits indicating which shader stages (if any) were principally used as inputs to compile this pipeline executable.
- name is an array of VK_MAX_DESCRIPTION_SIZE char containing a null-terminated UTF-8 string which is a short human readable name for this pipeline executable.
- description is an array of VK_MAX_DESCRIPTION_SIZE char containing a null-terminated UTF-8 string which is a human readable description for this pipeline executable.
- subgroupSize is the subgroup size with which this pipeline executable is dispatched.

Not all implementations have a 1:1 mapping between shader stages and pipeline executables and some implementations **may** reduce a given shader stage to fixed function hardware programming such that no pipeline executable is available. No guarantees are provided about the mapping between shader stages and pipeline executables and stages **should** be considered a best effort hint. Because the application **cannot** rely on the stages field to provide an exact description, name and description provide a human readable name and description which more accurately describes the given pipeline executable.

### Valid Usage (Implicit)

- VUID-VkPipelineExecutablePropertiesKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PIPELINE_EXECUTABLE_PROPERTIES_KHR
- VUID-VkPipelineExecutablePropertiesKHR-pNext-pNext
  pNext **must** be NULL

To query the pipeline properties call:

```
// Provided by VK_EXT_pipeline_properties
VkResult vkGetPipelinePropertiesEXT(
    VkDevice                                    device,
    const VkPipelineInfoEXT*                    pPipelineInfo,
    VkBaseOutStructure*                         pPipelineProperties);
```

- `device` is the logical device that created the pipeline.
- `pPipelineInfo` is a pointer to a VkPipelineInfoEXT structure which describes the pipeline being queried.
- `pPipelineProperties` is a pointer to a VkBaseOutStructure structure in which the pipeline properties will be written.

To query a pipeline's `pipelineIdentifier` pass a VkPipelinePropertiesIdentifierEXT structure in `pPipelineProperties`. Each pipeline is associated with a `pipelineIdentifier` and the identifier is implementation specific.

## Valid Usage

- VUID-vkGetPipelinePropertiesEXT-pipeline-06738
  The `pipeline` member of `pPipelineInfo` **must** have been created with `device`
- VUID-vkGetPipelinePropertiesEXT-pPipelineProperties-06739
  `pPipelineProperties` **must** be a valid pointer to a VkPipelinePropertiesIdentifierEXT structure
- VUID-vkGetPipelinePropertiesEXT-None-06766
  The `pipelinePropertiesIdentifier` feature **must** be enabled

## Valid Usage (Implicit)

- VUID-vkGetPipelinePropertiesEXT-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkGetPipelinePropertiesEXT-pPipelineInfo-parameter
  `pPipelineInfo` **must** be a valid pointer to a valid VkPipelineInfoEXT structure

## Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- `VK_ERROR_OUT_OF_HOST_MEMORY`

The `VkPipelinePropertiesIdentifierEXT` structure is defined as:

```
// Provided by VK_EXT_pipeline_properties
typedef struct VkPipelinePropertiesIdentifierEXT {
    VkStructureType    sType;
    void*              pNext;
    uint8_t            pipelineIdentifier[VK_UUID_SIZE];
} VkPipelinePropertiesIdentifierEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `pipelineIdentifier` is an array of `VK_UUID_SIZE` `uint8_t` values into which the pipeline identifier will be written.

### Valid Usage (Implicit)

- VUID-VkPipelinePropertiesIdentifierEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PIPELINE_PROPERTIES_IDENTIFIER_EXT`
- VUID-VkPipelinePropertiesIdentifierEXT-pNext-pNext
  `pNext` **must** be `NULL`

The `VkPipelineInfoKHR` structure is defined as:

```
// Provided by VK_KHR_pipeline_executable_properties
typedef struct VkPipelineInfoKHR {
    VkStructureType    sType;
    const void*        pNext;
    VkPipeline         pipeline;
} VkPipelineInfoKHR;
```

or the equivalent

```
// Provided by VK_EXT_pipeline_properties
typedef VkPipelineInfoKHR VkPipelineInfoEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `pipeline` is a `VkPipeline` handle.

### Valid Usage (Implicit)

- VUID-VkPipelineInfoKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PIPELINE_INFO_KHR`
- VUID-VkPipelineInfoKHR-pNext-pNext

pNext **must** be NULL

- VUID-VkPipelineInfoKHR-pipeline-parameter
  pipeline **must** be a valid VkPipeline handle

Each pipeline executable **may** have a set of statistics associated with it that are generated by the pipeline compilation process. These statistics **may** include things such as instruction counts, amount of spilling (if any), maximum number of simultaneous threads, or anything else which **may** aid developers in evaluating the expected performance of a shader. To query the compile time statistics associated with a pipeline executable, call:

```
// Provided by VK_KHR_pipeline_executable_properties
VkResult vkGetPipelineExecutableStatisticsKHR(
    VkDevice                                    device,
    const VkPipelineExecutableInfoKHR*          pExecutableInfo,
    uint32_t*                                   pStatisticCount,
    VkPipelineExecutableStatisticKHR*           pStatistics);
```

- device is the device that created the pipeline.
- pExecutableInfo describes the pipeline executable being queried.
- pStatisticCount is a pointer to an integer related to the number of statistics available or queried, as described below.
- pStatistics is either NULL or a pointer to an array of VkPipelineExecutableStatisticKHR structures.

If pStatistics is NULL, then the number of statistics associated with the pipeline executable is returned in pStatisticCount. Otherwise, pStatisticCount **must** point to a variable set by the application to the number of elements in the pStatistics array, and on return the variable is overwritten with the number of structures actually written to pStatistics. If pStatisticCount is less than the number of statistics associated with the pipeline executable, at most pStatisticCount structures will be written, and VK_INCOMPLETE will be returned instead of VK_SUCCESS, to indicate that not all the available statistics were returned.

### Valid Usage

- VUID-vkGetPipelineExecutableStatisticsKHR-pipelineExecutableInfo-03272
  The pipelineExecutableInfo feature **must** be enabled
- VUID-vkGetPipelineExecutableStatisticsKHR-pipeline-03273
  The pipeline member of pExecutableInfo **must** have been created with device
- VUID-vkGetPipelineExecutableStatisticsKHR-pipeline-03274
  The pipeline member of pExecutableInfo **must** have been created with VK_PIPELINE_CREATE_CAPTURE_STATISTICS_BIT_KHR

## Valid Usage (Implicit)

- VUID-vkGetPipelineExecutableStatisticsKHR-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkGetPipelineExecutableStatisticsKHR-pExecutableInfo-parameter
  `pExecutableInfo` **must** be a valid pointer to a valid VkPipelineExecutableInfoKHR structure
- VUID-vkGetPipelineExecutableStatisticsKHR-pStatisticCount-parameter
  `pStatisticCount` **must** be a valid pointer to a `uint32_t` value
- VUID-vkGetPipelineExecutableStatisticsKHR-pStatistics-parameter
  If the value referenced by `pStatisticCount` is not `0`, and `pStatistics` is not `NULL`, `pStatistics` **must** be a valid pointer to an array of `pStatisticCount` VkPipelineExecutableStatisticKHR structures

## Return Codes

**Success**

- `VK_SUCCESS`
- `VK_INCOMPLETE`

**Failure**

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`

The `VkPipelineExecutableInfoKHR` structure is defined as:

```
// Provided by VK_KHR_pipeline_executable_properties
typedef struct VkPipelineExecutableInfoKHR {
    VkStructureType    sType;
    const void*        pNext;
    VkPipeline         pipeline;
    uint32_t           executableIndex;
} VkPipelineExecutableInfoKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `pipeline` is the pipeline to query.
- `executableIndex` is the index of the pipeline executable to query in the array of executable properties returned by vkGetPipelineExecutablePropertiesKHR.

### Valid Usage

- VUID-VkPipelineExecutableInfoKHR-executableIndex-03275
  `executableIndex` **must** be less than the number of pipeline executables associated with `pipeline` as returned in the `pEXECUTABLECOUNT` parameter of `vkGetPipelineExecutablePropertiesKHR`

### Valid Usage (Implicit)

- VUID-VkPipelineExecutableInfoKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PIPELINE_EXECUTABLE_INFO_KHR`
- VUID-VkPipelineExecutableInfoKHR-pNext-pNext
  `pNext` **must** be `NULL`
- VUID-VkPipelineExecutableInfoKHR-pipeline-parameter
  `pipeline` **must** be a valid VkPipeline handle

The `VkPipelineExecutableStatisticKHR` structure is defined as:

```
// Provided by VK_KHR_pipeline_executable_properties
typedef struct VkPipelineExecutableStatisticKHR {
    VkStructureType                             sType;
    void*                                       pNext;
    char                                        name[VK_MAX_DESCRIPTION_SIZE];
    char                                        description[VK_MAX_DESCRIPTION_SIZE];
    VkPipelineExecutableStatisticFormatKHR      format;
    VkPipelineExecutableStatisticValueKHR       value;
} VkPipelineExecutableStatisticKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `name` is an array of `VK_MAX_DESCRIPTION_SIZE` `char` containing a null-terminated UTF-8 string which is a short human readable name for this statistic.
- `description` is an array of `VK_MAX_DESCRIPTION_SIZE` `char` containing a null-terminated UTF-8 string which is a human readable description for this statistic.
- `format` is a VkPipelineExecutableStatisticFormatKHR value specifying the format of the data found in `value`.
- `value` is the value of this statistic.

### Valid Usage (Implicit)

- VUID-VkPipelineExecutableStatisticKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PIPELINE_EXECUTABLE_STATISTIC_KHR`

- VUID-VkPipelineExecutableStatisticKHR-pNext-pNext
  pNext **must** be NULL

The VkPipelineExecutableStatisticFormatKHR enum is defined as:

```
// Provided by VK_KHR_pipeline_executable_properties
typedef enum VkPipelineExecutableStatisticFormatKHR {
    VK_PIPELINE_EXECUTABLE_STATISTIC_FORMAT_BOOL32_KHR = 0,
    VK_PIPELINE_EXECUTABLE_STATISTIC_FORMAT_INT64_KHR = 1,
    VK_PIPELINE_EXECUTABLE_STATISTIC_FORMAT_UINT64_KHR = 2,
    VK_PIPELINE_EXECUTABLE_STATISTIC_FORMAT_FLOAT64_KHR = 3,
} VkPipelineExecutableStatisticFormatKHR;
```

- VK_PIPELINE_EXECUTABLE_STATISTIC_FORMAT_BOOL32_KHR specifies that the statistic is returned as a 32-bit boolean value which **must** be either VK_TRUE or VK_FALSE and **should** be read from the b32 field of VkPipelineExecutableStatisticValueKHR.
- VK_PIPELINE_EXECUTABLE_STATISTIC_FORMAT_INT64_KHR specifies that the statistic is returned as a signed 64-bit integer and **should** be read from the i64 field of VkPipelineExecutableStatisticValueKHR.
- VK_PIPELINE_EXECUTABLE_STATISTIC_FORMAT_UINT64_KHR specifies that the statistic is returned as an unsigned 64-bit integer and **should** be read from the u64 field of VkPipelineExecutableStatisticValueKHR.
- VK_PIPELINE_EXECUTABLE_STATISTIC_FORMAT_FLOAT64_KHR specifies that the statistic is returned as a 64-bit floating-point value and **should** be read from the f64 field of VkPipelineExecutableStatisticValueKHR.

The VkPipelineExecutableStatisticValueKHR union is defined as:

```
// Provided by VK_KHR_pipeline_executable_properties
typedef union VkPipelineExecutableStatisticValueKHR {
    VkBool32    b32;
    int64_t     i64;
    uint64_t    u64;
    double      f64;
} VkPipelineExecutableStatisticValueKHR;
```

- b32 is the 32-bit boolean value if the VkPipelineExecutableStatisticFormatKHR is VK_PIPELINE_EXECUTABLE_STATISTIC_FORMAT_BOOL32_KHR.
- i64 is the signed 64-bit integer value if the VkPipelineExecutableStatisticFormatKHR is VK_PIPELINE_EXECUTABLE_STATISTIC_FORMAT_INT64_KHR.
- u64 is the unsigned 64-bit integer value if the VkPipelineExecutableStatisticFormatKHR is VK_PIPELINE_EXECUTABLE_STATISTIC_FORMAT_UINT64_KHR.
- f64 is the 64-bit floating-point value if the VkPipelineExecutableStatisticFormatKHR is VK_PIPELINE_EXECUTABLE_STATISTIC_FORMAT_FLOAT64_KHR.

Each pipeline executable **may** have one or more text or binary internal representations associated with it which are generated as part of the compile process. These **may** include the final shader assembly, a binary form of the compiled shader, or the shader compiler's internal representation at any number of intermediate compile steps. To query the internal representations associated with a pipeline executable, call:

```
// Provided by VK_KHR_pipeline_executable_properties
VkResult vkGetPipelineExecutableInternalRepresentationsKHR(
    VkDevice                                    device,
    const VkPipelineExecutableInfoKHR*          pExecutableInfo,
    uint32_t*                                   pInternalRepresentationCount,
    VkPipelineExecutableInternalRepresentationKHR* pInternalRepresentations);
```

- `device` is the device that created the pipeline.
- `pExecutableInfo` describes the pipeline executable being queried.
- `pInternalRepresentationCount` is a pointer to an integer related to the number of internal representations available or queried, as described below.
- `pInternalRepresentations` is either `NULL` or a pointer to an array of VkPipelineExecutableInternalRepresentationKHR structures.

If `pInternalRepresentations` is `NULL`, then the number of internal representations associated with the pipeline executable is returned in `pInternalRepresentationCount`. Otherwise, `pInternalRepresentationCount` **must** point to a variable set by the application to the number of elements in the `pInternalRepresentations` array, and on return the variable is overwritten with the number of structures actually written to `pInternalRepresentations`. If `pInternalRepresentationCount` is less than the number of internal representations associated with the pipeline executable, at most `pInternalRepresentationCount` structures will be written, and `VK_INCOMPLETE` will be returned instead of `VK_SUCCESS`, to indicate that not all the available representations were returned.

While the details of the internal representations remain implementation-dependent, the implementation **should** order the internal representations in the order in which they occur in the compiled pipeline with the final shader assembly (if any) last.

### Valid Usage

- VUID-vkGetPipelineExecutableInternalRepresentationsKHR-pipelineExecutableInfo-03276
  The `pipelineExecutableInfo` feature **must** be enabled
- VUID-vkGetPipelineExecutableInternalRepresentationsKHR-pipeline-03277
  The `pipeline` member of `pExecutableInfo` **must** have been created with `device`
- VUID-vkGetPipelineExecutableInternalRepresentationsKHR-pipeline-03278
  The `pipeline` member of `pExecutableInfo` **must** have been created with `VK_PIPELINE_CREATE_CAPTURE_INTERNAL_REPRESENTATIONS_BIT_KHR`

### Valid Usage (Implicit)

- VUID-vkGetPipelineExecutableInternalRepresentationsKHR-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkGetPipelineExecutableInternalRepresentationsKHR-pExecutableInfo-parameter
  `pExecutableInfo` **must** be a valid pointer to a valid VkPipelineExecutableInfoKHR structure
- VUID-vkGetPipelineExecutableInternalRepresentationsKHR-pInternalRepresentationCount-parameter
  `pInternalRepresentationCount` **must** be a valid pointer to a `uint32_t` value
- VUID-vkGetPipelineExecutableInternalRepresentationsKHR-pInternalRepresentations-parameter
  If the value referenced by `pInternalRepresentationCount` is not `0`, and `pInternalRepresentations` is not `NULL`, `pInternalRepresentations` **must** be a valid pointer to an array of `pInternalRepresentationCount` VkPipelineExecutableInternalRepresentationKHR structures

### Return Codes

**Success**

- `VK_SUCCESS`
- `VK_INCOMPLETE`

**Failure**

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`

The `VkPipelineExecutableInternalRepresentationKHR` structure is defined as:

```
// Provided by VK_KHR_pipeline_executable_properties
typedef struct VkPipelineExecutableInternalRepresentationKHR {
    VkStructureType    sType;
    void*              pNext;
    char               name[VK_MAX_DESCRIPTION_SIZE];
    char               description[VK_MAX_DESCRIPTION_SIZE];
    VkBool32           isText;
    size_t             dataSize;
    void*              pData;
} VkPipelineExecutableInternalRepresentationKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `name` is an array of `VK_MAX_DESCRIPTION_SIZE` `char` containing a null-terminated UTF-8 string

which is a short human readable name for this internal representation.

- `description` is an array of `VK_MAX_DESCRIPTION_SIZE` `char` containing a null-terminated UTF-8 string which is a human readable description for this internal representation.
- `isText` specifies whether the returned data is text or opaque data. If `isText` is `VK_TRUE` then the data returned in `pData` is text and is guaranteed to be a null-terminated UTF-8 string.
- `dataSize` is an integer related to the size, in bytes, of the internal representation's data, as described below.
- `pData` is either `NULL` or a pointer to a block of data into which the implementation will write the internal representation.

If `pData` is `NULL`, then the size, in bytes, of the internal representation data is returned in `dataSize`. Otherwise, `dataSize` **must** be the size of the buffer, in bytes, pointed to by `pData` and on return `dataSize` is overwritten with the number of bytes of data actually written to `pData` including any trailing null character. If `dataSize` is less than the size, in bytes, of the internal representation's data, at most `dataSize` bytes of data will be written to `pData`, and `VK_INCOMPLETE` will be returned instead of `VK_SUCCESS`, to indicate that not all the available representation was returned.

If `isText` is `VK_TRUE` and `pData` is not `NULL` and `dataSize` is not zero, the last byte written to `pData` will be a null character.

### Valid Usage (Implicit)

- VUID-VkPipelineExecutableInternalRepresentationKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PIPELINE_EXECUTABLE_INTERNAL_REPRESENTATION_KHR`
- VUID-VkPipelineExecutableInternalRepresentationKHR-pNext-pNext
  `pNext` **must** be `NULL`

Information about a particular shader that has been compiled as part of a pipeline object can be extracted by calling:

```
// Provided by VK_AMD_shader_info
VkResult vkGetShaderInfoAMD(
    VkDevice                                    device,
    VkPipeline                                  pipeline,
    VkShaderStageFlagBits                       shaderStage,
    VkShaderInfoTypeAMD                         infoType,
    size_t*                                     pInfoSize,
    void*                                       pInfo);
```

- `device` is the device that created `pipeline`.
- `pipeline` is the target of the query.
- `shaderStage` is a VkShaderStageFlagBits specifying the particular shader within the pipeline about which information is being queried.
- `infoType` describes what kind of information is being queried.

- pInfoSize is a pointer to a value related to the amount of data the query returns, as described below.
- pInfo is either NULL or a pointer to a buffer.

If pInfo is NULL, then the maximum size of the information that **can** be retrieved about the shader, in bytes, is returned in pInfoSize. Otherwise, pInfoSize **must** point to a variable set by the application to the size of the buffer, in bytes, pointed to by pInfo, and on return the variable is overwritten with the amount of data actually written to pInfo. If pInfoSize is less than the maximum size that **can** be retrieved by the pipeline cache, then at most pInfoSize bytes will be written to pInfo, and VK_INCOMPLETE will be returned, instead of VK_SUCCESS, to indicate that not all required of the pipeline cache was returned.

Not all information is available for every shader and implementations may not support all kinds of information for any shader. When a certain type of information is unavailable, the function returns VK_ERROR_FEATURE_NOT_PRESENT.

If information is successfully and fully queried, the function will return VK_SUCCESS.

For infoType VK_SHADER_INFO_TYPE_STATISTICS_AMD, a VkShaderStatisticsInfoAMD structure will be written to the buffer pointed to by pInfo. This structure will be populated with statistics regarding the physical device resources used by that shader along with other miscellaneous information and is described in further detail below.

For infoType VK_SHADER_INFO_TYPE_DISASSEMBLY_AMD, pInfo is a pointer to a null-terminated UTF-8 string containing human-readable disassembly. The exact formatting and contents of the disassembly string are vendor-specific.

The formatting and contents of all other types of information, including infoType VK_SHADER_INFO_TYPE_BINARY_AMD, are left to the vendor and are not further specified by this extension.

### Valid Usage (Implicit)

- VUID-vkGetShaderInfoAMD-device-parameter
  device **must** be a valid VkDevice handle
- VUID-vkGetShaderInfoAMD-pipeline-parameter
  pipeline **must** be a valid VkPipeline handle
- VUID-vkGetShaderInfoAMD-shaderStage-parameter
  shaderStage **must** be a valid VkShaderStageFlagBits value
- VUID-vkGetShaderInfoAMD-infoType-parameter
  infoType **must** be a valid VkShaderInfoTypeAMD value
- VUID-vkGetShaderInfoAMD-pInfoSize-parameter
  pInfoSize **must** be a valid pointer to a size_t value
- VUID-vkGetShaderInfoAMD-pInfo-parameter
  If the value referenced by pInfoSize is not 0, and pInfo is not NULL, pInfo **must** be a valid pointer to an array of pInfoSize bytes

- VUID-vkGetShaderInfoAMD-pipeline-parent
  `pipeline` **must** have been created, allocated, or retrieved from `device`

### Return Codes

**Success**

- `VK_SUCCESS`
- `VK_INCOMPLETE`

**Failure**

- `VK_ERROR_FEATURE_NOT_PRESENT`
- `VK_ERROR_OUT_OF_HOST_MEMORY`

Possible values of vkGetShaderInfoAMD::`infoType`, specifying the information being queried from a shader, are:

```
// Provided by VK_AMD_shader_info
typedef enum VkShaderInfoTypeAMD {
    VK_SHADER_INFO_TYPE_STATISTICS_AMD = 0,
    VK_SHADER_INFO_TYPE_BINARY_AMD = 1,
    VK_SHADER_INFO_TYPE_DISASSEMBLY_AMD = 2,
} VkShaderInfoTypeAMD;
```

- `VK_SHADER_INFO_TYPE_STATISTICS_AMD` specifies that device resources used by a shader will be queried.
- `VK_SHADER_INFO_TYPE_BINARY_AMD` specifies that implementation-specific information will be queried.
- `VK_SHADER_INFO_TYPE_DISASSEMBLY_AMD` specifies that human-readable disassembly of a shader.

The `VkShaderStatisticsInfoAMD` structure is defined as:

```
// Provided by VK_AMD_shader_info
typedef struct VkShaderStatisticsInfoAMD {
    VkShaderStageFlags          shaderStageMask;
    VkShaderResourceUsageAMD    resourceUsage;
    uint32_t                    numPhysicalVgprs;
    uint32_t                    numPhysicalSgprs;
    uint32_t                    numAvailableVgprs;
    uint32_t                    numAvailableSgprs;
    uint32_t                    computeWorkGroupSize[3];
} VkShaderStatisticsInfoAMD;
```

- `shaderStageMask` are the combination of logical shader stages contained within this shader.
- `resourceUsage` is a VkShaderResourceUsageAMD structure describing internal physical device

resources used by this shader.

- numPhysicalVgprs is the maximum number of vector instruction general-purpose registers (VGPRs) available to the physical device.
- numPhysicalSgprs is the maximum number of scalar instruction general-purpose registers (SGPRs) available to the physical device.
- numAvailableVgprs is the maximum limit of VGPRs made available to the shader compiler.
- numAvailableSgprs is the maximum limit of SGPRs made available to the shader compiler.
- computeWorkGroupSize is the local workgroup size of this shader in { X, Y, Z } dimensions.

Some implementations may merge multiple logical shader stages together in a single shader. In such cases, shaderStageMask will contain a bitmask of all of the stages that are active within that shader. Consequently, if specifying those stages as input to vkGetShaderInfoAMD, the same output information **may** be returned for all such shader stage queries.

The number of available VGPRs and SGPRs (numAvailableVgprs and numAvailableSgprs respectively) are the shader-addressable subset of physical registers that is given as a limit to the compiler for register assignment. These values **may** further be limited by implementations due to performance optimizations where register pressure is a bottleneck.

The VkShaderResourceUsageAMD structure is defined as:

```
// Provided by VK_AMD_shader_info
typedef struct VkShaderResourceUsageAMD {
    uint32_t    numUsedVgprs;
    uint32_t    numUsedSgprs;
    uint32_t    ldsSizePerLocalWorkGroup;
    size_t      ldsUsageSizeInBytes;
    size_t      scratchMemUsageInBytes;
} VkShaderResourceUsageAMD;
```

- numUsedVgprs is the number of vector instruction general-purpose registers used by this shader.
- numUsedSgprs is the number of scalar instruction general-purpose registers used by this shader.
- ldsSizePerLocalWorkGroup is the maximum local data store size per work group in bytes.
- ldsUsageSizeInBytes is the LDS usage size in bytes per work group by this shader.
- scratchMemUsageInBytes is the scratch memory usage in bytes by this shader.

## 10.14. Pipeline Compiler Control

The compilation of a pipeline **can** be tuned by adding a VkPipelineCompilerControlCreateInfoAMD structure to the pNext chain of VkGraphicsPipelineCreateInfo or VkComputePipelineCreateInfo.

```
// Provided by VK_AMD_pipeline_compiler_control
typedef struct VkPipelineCompilerControlCreateInfoAMD {
    VkStructureType                     sType;
    const void*                         pNext;
    VkPipelineCompilerControlFlagsAMD   compilerControlFlags;
} VkPipelineCompilerControlCreateInfoAMD;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `compilerControlFlags` is a bitmask of VkPipelineCompilerControlFlagBitsAMD affecting how the pipeline will be compiled.

**Valid Usage (Implicit)**

- VUID-VkPipelineCompilerControlCreateInfoAMD-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PIPELINE_COMPILER_CONTROL_CREATE_INFO_AMD`
- VUID-VkPipelineCompilerControlCreateInfoAMD-compilerControlFlags-zerobitmask
  `compilerControlFlags` **must** be `0`

There are currently no available flags for this extension; flags will be added by future versions of this extension.

```
// Provided by VK_AMD_pipeline_compiler_control
typedef enum VkPipelineCompilerControlFlagBitsAMD {
} VkPipelineCompilerControlFlagBitsAMD;
```

```
// Provided by VK_AMD_pipeline_compiler_control
typedef VkFlags VkPipelineCompilerControlFlagsAMD;
```

`VkPipelineCompilerControlFlagsAMD` is a bitmask type for setting a mask of zero or more VkPipelineCompilerControlFlagBitsAMD.

## 10.15. Pipeline Creation Feedback

Feedback about the creation of a particular pipeline object **can** be obtained by adding a `VkPipelineCreationFeedbackCreateInfo` structure to the `pNext` chain of VkGraphicsPipelineCreateInfo, VkRayTracingPipelineCreateInfoKHR, VkRayTracingPipelineCreateInfoNV, or VkComputePipelineCreateInfo. The `VkPipelineCreationFeedbackCreateInfo` structure is defined as:

```
typedef struct VkPipelineCreationFeedbackCreateInfo {
    VkStructureType                 sType;
    const void*                     pNext;
    VkPipelineCreationFeedback*     pPipelineCreationFeedback;
    uint32_t                        pipelineStageCreationFeedbackCount;
    VkPipelineCreationFeedback*     pPipelineStageCreationFeedbacks;
} VkPipelineCreationFeedbackCreateInfo;
```

or the equivalent

```
// Provided by VK_EXT_pipeline_creation_feedback
typedef VkPipelineCreationFeedbackCreateInfo VkPipelineCreationFeedbackCreateInfoEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `pPipelineCreationFeedback` is a pointer to a VkPipelineCreationFeedback structure.
- `pipelineStageCreationFeedbackCount` is the number of elements in `pPipelineStageCreationFeedbacks`.
- `pPipelineStageCreationFeedbacks` is a pointer to an array of `pipelineStageCreationFeedbackCount` VkPipelineCreationFeedback structures.

An implementation **should** write pipeline creation feedback to `pPipelineCreationFeedback` and **may** write pipeline stage creation feedback to `pPipelineStageCreationFeedbacks`. An implementation **must** set or clear the `VK_PIPELINE_CREATION_FEEDBACK_VALID_BIT` in VkPipelineCreationFeedback::`flags` for `pPipelineCreationFeedback` and every element of `pPipelineStageCreationFeedbacks`.

**NOTE** One common scenario for an implementation to skip per-stage feedback is when `VK_PIPELINE_CREATION_FEEDBACK_APPLICATION_PIPELINE_CACHE_HIT_BIT` is set in `pPipelineCreationFeedback`.

When chained to VkRayTracingPipelineCreateInfoKHR, VkRayTracingPipelineCreateInfoNV, or VkGraphicsPipelineCreateInfo, the `i` element of `pPipelineStageCreationFeedbacks` corresponds to the `i` element of VkRayTracingPipelineCreateInfoKHR::`pStages`, VkRayTracingPipelineCreateInfoNV::`pStages`, or VkGraphicsPipelineCreateInfo::`pStages`. When chained to VkComputePipelineCreateInfo, the first element of `pPipelineStageCreationFeedbacks` corresponds to VkComputePipelineCreateInfo::`stage`.

### Valid Usage (Implicit)

- VUID-VkPipelineCreationFeedbackCreateInfo-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PIPELINE_CREATION_FEEDBACK_CREATE_INFO`
- VUID-VkPipelineCreationFeedbackCreateInfo-pPipelineCreationFeedback-parameter
  `pPipelineCreationFeedback` **must** be a valid pointer to a VkPipelineCreationFeedback structure

- VUID-VkPipelineCreationFeedbackCreateInfo-pPipelineStageCreationFeedbacks-parameter
  If `pipelineStageCreationFeedbackCount` is not `0`, `pPipelineStageCreationFeedbacks` **must** be a valid pointer to an array of `pipelineStageCreationFeedbackCount` VkPipelineCreationFeedback structures

The `VkPipelineCreationFeedback` structure is defined as:

```
typedef struct VkPipelineCreationFeedback {
    VkPipelineCreationFeedbackFlags    flags;
    uint64_t                           duration;
} VkPipelineCreationFeedback;
```

or the equivalent

```
// Provided by VK_EXT_pipeline_creation_feedback
typedef VkPipelineCreationFeedback VkPipelineCreationFeedbackEXT;
```

- `flags` is a bitmask of VkPipelineCreationFeedbackFlagBits providing feedback about the creation of a pipeline or of a pipeline stage.
- `duration` is the duration spent creating a pipeline or pipeline stage in nanoseconds.

If the `VK_PIPELINE_CREATION_FEEDBACK_VALID_BIT` is not set in `flags`, an implementation **must** not set any other bits in `flags`, and the values of all other `VkPipelineCreationFeedback` data members are undefined.

Possible values of the `flags` member of VkPipelineCreationFeedback are:

```
typedef enum VkPipelineCreationFeedbackFlagBits {
    VK_PIPELINE_CREATION_FEEDBACK_VALID_BIT = 0x00000001,
    VK_PIPELINE_CREATION_FEEDBACK_APPLICATION_PIPELINE_CACHE_HIT_BIT = 0x00000002,
    VK_PIPELINE_CREATION_FEEDBACK_BASE_PIPELINE_ACCELERATION_BIT = 0x00000004,
    VK_PIPELINE_CREATION_FEEDBACK_VALID_BIT_EXT =
VK_PIPELINE_CREATION_FEEDBACK_VALID_BIT,
    VK_PIPELINE_CREATION_FEEDBACK_APPLICATION_PIPELINE_CACHE_HIT_BIT_EXT =
VK_PIPELINE_CREATION_FEEDBACK_APPLICATION_PIPELINE_CACHE_HIT_BIT,
    VK_PIPELINE_CREATION_FEEDBACK_BASE_PIPELINE_ACCELERATION_BIT_EXT =
VK_PIPELINE_CREATION_FEEDBACK_BASE_PIPELINE_ACCELERATION_BIT,
} VkPipelineCreationFeedbackFlagBits;
```

or the equivalent

```
// Provided by VK_EXT_pipeline_creation_feedback
typedef VkPipelineCreationFeedbackFlagBits VkPipelineCreationFeedbackFlagBitsEXT;
```

- VK_PIPELINE_CREATION_FEEDBACK_VALID_BIT indicates that the feedback information is valid.
- VK_PIPELINE_CREATION_FEEDBACK_APPLICATION_PIPELINE_CACHE_HIT_BIT indicates that a readily usable pipeline or pipeline stage was found in the pipelineCache specified by the application in the pipeline creation command.

  An implementation **should** set the VK_PIPELINE_CREATION_FEEDBACK_APPLICATION_PIPELINE_CACHE_HIT_BIT bit if it was able to avoid the large majority of pipeline or pipeline stage creation work by using the pipelineCache parameter of vkCreateGraphicsPipelines, vkCreateRayTracingPipelinesKHR, vkCreateRayTracingPipelinesNV, or vkCreateComputePipelines. When an implementation sets this bit for the entire pipeline, it **may** leave it unset for any stage.

  **NOTE** Implementations are encouraged to provide a meaningful signal to applications using this bit. The intention is to communicate to the application that the pipeline or pipeline stage was created "as fast as it gets" using the pipeline cache provided by the application. If an implementation uses an internal cache, it is discouraged from setting this bit as the feedback would be unactionable.

- VK_PIPELINE_CREATION_FEEDBACK_BASE_PIPELINE_ACCELERATION_BIT indicates that the base pipeline specified by the basePipelineHandle or basePipelineIndex member of the Vk*PipelineCreateInfo structure was used to accelerate the creation of the pipeline.

  An implementation **should** set the VK_PIPELINE_CREATION_FEEDBACK_BASE_PIPELINE_ACCELERATION_BIT bit if it was able to avoid a significant amount of work by using the base pipeline.

  **NOTE** While "significant amount of work" is subjective, implementations are encouraged to provide a meaningful signal to applications using this bit. For example, a 1% reduction in duration may not warrant setting this bit, while a 50% reduction would.

```
typedef VkFlags VkPipelineCreationFeedbackFlags;
```

or the equivalent

```
// Provided by VK_EXT_pipeline_creation_feedback
typedef VkPipelineCreationFeedbackFlags VkPipelineCreationFeedbackFlagsEXT;
```

VkPipelineCreationFeedbackFlags is a bitmask type for providing zero or more VkPipelineCreationFeedbackFlagBits.

# Chapter 11. Memory Allocation

Vulkan memory is broken up into two categories, *host memory* and *device memory*.

## 11.1. Host Memory

Host memory is memory needed by the Vulkan implementation for non-device-visible storage.

| | |
|---|---|
| **NOTE** | This memory **may** be used to store the implementation's representation and state of Vulkan objects. |

Vulkan provides applications the opportunity to perform host memory allocations on behalf of the Vulkan implementation. If this feature is not used, the implementation will perform its own memory allocations. Since most memory allocations are off the critical path, this is not meant as a performance feature. Rather, this **can** be useful for certain embedded systems, for debugging purposes (e.g. putting a guard page after all host allocations), or for memory allocation logging.

Allocators are provided by the application as a pointer to a `VkAllocationCallbacks` structure:

```
// Provided by VK_VERSION_1_0
typedef struct VkAllocationCallbacks {
    void*                                   pUserData;
    PFN_vkAllocationFunction                pfnAllocation;
    PFN_vkReallocationFunction              pfnReallocation;
    PFN_vkFreeFunction                      pfnFree;
    PFN_vkInternalAllocationNotification    pfnInternalAllocation;
    PFN_vkInternalFreeNotification          pfnInternalFree;
} VkAllocationCallbacks;
```

- `pUserData` is a value to be interpreted by the implementation of the callbacks. When any of the callbacks in `VkAllocationCallbacks` are called, the Vulkan implementation will pass this value as the first parameter to the callback. This value **can** vary each time an allocator is passed into a command, even when the same object takes an allocator in multiple commands.
- `pfnAllocation` is a PFN_vkAllocationFunction pointer to an application-defined memory allocation function.
- `pfnReallocation` is a PFN_vkReallocationFunction pointer to an application-defined memory reallocation function.
- `pfnFree` is a PFN_vkFreeFunction pointer to an application-defined memory free function.
- `pfnInternalAllocation` is a PFN_vkInternalAllocationNotification pointer to an application-defined function that is called by the implementation when the implementation makes internal allocations.
- `pfnInternalFree` is a PFN_vkInternalFreeNotification pointer to an application-defined function that is called by the implementation when the implementation frees internal allocations.

### Valid Usage

- VUID-VkAllocationCallbacks-pfnAllocation-00632
  `pfnAllocation` **must** be a valid pointer to a valid application-defined PFN_vkAllocationFunction
- VUID-VkAllocationCallbacks-pfnReallocation-00633
  `pfnReallocation` **must** be a valid pointer to a valid application-defined PFN_vkReallocationFunction
- VUID-VkAllocationCallbacks-pfnFree-00634
  `pfnFree` **must** be a valid pointer to a valid application-defined PFN_vkFreeFunction
- VUID-VkAllocationCallbacks-pfnInternalAllocation-00635
  If either of `pfnInternalAllocation` or `pfnInternalFree` is not `NULL`, both **must** be valid callbacks

The type of `pfnAllocation` is:

```
// Provided by VK_VERSION_1_0
typedef void* (VKAPI_PTR *PFN_vkAllocationFunction)(
    void*                                       pUserData,
    size_t                                      size,
    size_t                                      alignment,
    VkSystemAllocationScope                     allocationScope);
```

- `pUserData` is the value specified for VkAllocationCallbacks::`pUserData` in the allocator specified by the application.
- `size` is the size in bytes of the requested allocation.
- `alignment` is the requested alignment of the allocation in bytes and **must** be a power of two.
- `allocationScope` is a VkSystemAllocationScope value specifying the allocation scope of the lifetime of the allocation, as described here.

If `pfnAllocation` is unable to allocate the requested memory, it **must** return `NULL`. If the allocation was successful, it **must** return a valid pointer to memory allocation containing at least `size` bytes, and with the pointer value being a multiple of `alignment`.

**NOTE**

Correct Vulkan operation **cannot** be assumed if the application does not follow these rules.

For example, `pfnAllocation` (or `pfnReallocation`) could cause termination of running Vulkan instance(s) on a failed allocation for debugging purposes, either directly or indirectly. In these circumstances, it **cannot** be assumed that any part of any affected VkInstance objects are going to operate correctly (even vkDestroyInstance), and the application **must** ensure it cleans up properly via other means (e.g. process termination).

If pfnAllocation returns NULL, and if the implementation is unable to continue correct processing of the current command without the requested allocation, it **must** treat this as a runtime error, and generate VK_ERROR_OUT_OF_HOST_MEMORY at the appropriate time for the command in which the condition was detected, as described in Return Codes.

If the implementation is able to continue correct processing of the current command without the requested allocation, then it **may** do so, and **must** not generate VK_ERROR_OUT_OF_HOST_MEMORY as a result of this failed allocation.

The type of pfnReallocation is:

```
// Provided by VK_VERSION_1_0
typedef void* (VKAPI_PTR *PFN_vkReallocationFunction)(
    void*                                       pUserData,
    void*                                       pOriginal,
    size_t                                      size,
    size_t                                      alignment,
    VkSystemAllocationScope                     allocationScope);
```

- pUserData is the value specified for VkAllocationCallbacks::pUserData in the allocator specified by the application.
- pOriginal **must** be either NULL or a pointer previously returned by pfnReallocation or pfnAllocation of a compatible allocator.
- size is the size in bytes of the requested allocation.
- alignment is the requested alignment of the allocation in bytes and **must** be a power of two.
- allocationScope is a VkSystemAllocationScope value specifying the allocation scope of the lifetime of the allocation, as described here.

If the reallocation was successful, pfnReallocation **must** return an allocation with enough space for size bytes, and the contents of the original allocation from bytes zero to min(original size, new size) - 1 **must** be preserved in the returned allocation. If size is larger than the old size, the contents of the additional space are undefined. If satisfying these requirements involves creating a new allocation, then the old allocation **should** be freed.

If pOriginal is NULL, then pfnReallocation **must** behave equivalently to a call to PFN_vkAllocationFunction with the same parameter values (without pOriginal).

If size is zero, then pfnReallocation **must** behave equivalently to a call to PFN_vkFreeFunction with the same pUserData parameter value, and pMemory equal to pOriginal.

If pOriginal is non-NULL, the implementation **must** ensure that alignment is equal to the alignment used to originally allocate pOriginal.

If this function fails and pOriginal is non-NULL the application **must** not free the old allocation.

pfnReallocation **must** follow the same rules for return values as PFN_vkAllocationFunction.

The type of pfnFree is:

```
// Provided by VK_VERSION_1_0
typedef void (VKAPI_PTR *PFN_vkFreeFunction)(
    void*                                       pUserData,
    void*                                       pMemory);
```

- pUserData is the value specified for VkAllocationCallbacks::pUserData in the allocator specified by the application.
- pMemory is the allocation to be freed.

pMemory **may** be NULL, which the callback **must** handle safely. If pMemory is non-NULL, it **must** be a pointer previously allocated by pfnAllocation or pfnReallocation. The application **should** free this memory.

The type of pfnInternalAllocation is:

```
// Provided by VK_VERSION_1_0
typedef void (VKAPI_PTR *PFN_vkInternalAllocationNotification)(
    void*                                       pUserData,
    size_t                                      size,
    VkInternalAllocationType                    allocationType,
    VkSystemAllocationScope                     allocationScope);
```

- pUserData is the value specified for VkAllocationCallbacks::pUserData in the allocator specified by the application.
- size is the requested size of an allocation.
- allocationType is a VkInternalAllocationType value specifying the requested type of an allocation.
- allocationScope is a VkSystemAllocationScope value specifying the allocation scope of the lifetime of the allocation, as described here.

This is a purely informational callback.

The type of pfnInternalFree is:

```
// Provided by VK_VERSION_1_0
typedef void (VKAPI_PTR *PFN_vkInternalFreeNotification)(
    void*                                       pUserData,
    size_t                                      size,
    VkInternalAllocationType                    allocationType,
    VkSystemAllocationScope                     allocationScope);
```

- pUserData is the value specified for VkAllocationCallbacks::pUserData in the allocator specified by the application.
- size is the requested size of an allocation.

- allocationType is a VkInternalAllocationType value specifying the requested type of an allocation.
- allocationScope is a VkSystemAllocationScope value specifying the allocation scope of the lifetime of the allocation, as described here.

Each allocation has an *allocation scope* defining its lifetime and which object it is associated with. Possible values passed to the allocationScope parameter of the callback functions specified by VkAllocationCallbacks, indicating the allocation scope, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkSystemAllocationScope {
    VK_SYSTEM_ALLOCATION_SCOPE_COMMAND = 0,
    VK_SYSTEM_ALLOCATION_SCOPE_OBJECT = 1,
    VK_SYSTEM_ALLOCATION_SCOPE_CACHE = 2,
    VK_SYSTEM_ALLOCATION_SCOPE_DEVICE = 3,
    VK_SYSTEM_ALLOCATION_SCOPE_INSTANCE = 4,
} VkSystemAllocationScope;
```

- VK_SYSTEM_ALLOCATION_SCOPE_COMMAND specifies that the allocation is scoped to the duration of the Vulkan command.
- VK_SYSTEM_ALLOCATION_SCOPE_OBJECT specifies that the allocation is scoped to the lifetime of the Vulkan object that is being created or used.
- VK_SYSTEM_ALLOCATION_SCOPE_CACHE specifies that the allocation is scoped to the lifetime of a VkPipelineCache or VkValidationCacheEXT object.
- VK_SYSTEM_ALLOCATION_SCOPE_DEVICE specifies that the allocation is scoped to the lifetime of the Vulkan device.
- VK_SYSTEM_ALLOCATION_SCOPE_INSTANCE specifies that the allocation is scoped to the lifetime of the Vulkan instance.

Most Vulkan commands operate on a single object, or there is a sole object that is being created or manipulated. When an allocation uses an allocation scope of VK_SYSTEM_ALLOCATION_SCOPE_OBJECT or VK_SYSTEM_ALLOCATION_SCOPE_CACHE, the allocation is scoped to the object being created or manipulated.

When an implementation requires host memory, it will make callbacks to the application using the most specific allocator and allocation scope available:

- If an allocation is scoped to the duration of a command, the allocator will use the VK_SYSTEM_ALLOCATION_SCOPE_COMMAND allocation scope. The most specific allocator available is used: if the object being created or manipulated has an allocator, that object's allocator will be used, else if the parent VkDevice has an allocator it will be used, else if the parent VkInstance has an allocator it will be used. Else,
- If an allocation is associated with a VkValidationCacheEXT or VkPipelineCache object, the allocator will use the VK_SYSTEM_ALLOCATION_SCOPE_CACHE allocation scope. The most specific allocator available is used (cache, else device, else instance). Else,

- If an allocation is scoped to the lifetime of an object, that object is being created or manipulated by the command, and that object's type is not VkDevice or VkInstance, the allocator will use an allocation scope of VK_SYSTEM_ALLOCATION_SCOPE_OBJECT. The most specific allocator available is used (object, else device, else instance). Else,
- If an allocation is scoped to the lifetime of a device, the allocator will use an allocation scope of VK_SYSTEM_ALLOCATION_SCOPE_DEVICE. The most specific allocator available is used (device, else instance). Else,
- If the allocation is scoped to the lifetime of an instance and the instance has an allocator, its allocator will be used with an allocation scope of VK_SYSTEM_ALLOCATION_SCOPE_INSTANCE.
- Otherwise an implementation will allocate memory through an alternative mechanism that is unspecified.

Objects that are allocated from pools do not specify their own allocator. When an implementation requires host memory for such an object, that memory is sourced from the object's parent pool's allocator.

The application is not expected to handle allocating memory that is intended for execution by the host due to the complexities of differing security implementations across multiple platforms. The implementation will allocate such memory internally and invoke an application provided informational callback when these *internal allocations* are allocated and freed. Upon allocation of executable memory, pfnInternalAllocation will be called. Upon freeing executable memory, pfnInternalFree will be called. An implementation will only call an informational callback for executable memory allocations and frees.

The allocationType parameter to the pfnInternalAllocation and pfnInternalFree functions **may** be one of the following values:

```
// Provided by VK_VERSION_1_0
typedef enum VkInternalAllocationType {
    VK_INTERNAL_ALLOCATION_TYPE_EXECUTABLE = 0,
} VkInternalAllocationType;
```

- VK_INTERNAL_ALLOCATION_TYPE_EXECUTABLE specifies that the allocation is intended for execution by the host.

An implementation **must** only make calls into an application-provided allocator during the execution of an API command. An implementation **must** only make calls into an application-provided allocator from the same thread that called the provoking API command. The implementation **should** not synchronize calls to any of the callbacks. If synchronization is needed, the callbacks **must** provide it themselves. The informational callbacks are subject to the same restrictions as the allocation callbacks.

If an implementation intends to make calls through a VkAllocationCallbacks structure between the time a vkCreate* command returns and the time a corresponding vkDestroy* command begins, that implementation **must** save a copy of the allocator before the vkCreate* command returns. The callback functions and any data structures they rely upon **must** remain valid for the lifetime of the object they are associated with.

If an allocator is provided to a vkCreate* command, a *compatible* allocator **must** be provided to the corresponding vkDestroy* command. Two VkAllocationCallbacks structures are compatible if memory allocated with pfnAllocation or pfnReallocation in each **can** be freed with pfnReallocation or pfnFree in the other. An allocator **must** not be provided to a vkDestroy* command if an allocator was not provided to the corresponding vkCreate* command.

If a non-NULL allocator is used, the pfnAllocation, pfnReallocation and pfnFree members **must** be non-NULL and point to valid implementations of the callbacks. An application **can** choose to not provide informational callbacks by setting both pfnInternalAllocation and pfnInternalFree to NULL. pfnInternalAllocation and pfnInternalFree **must** either both be NULL or both be non-NULL.

If pfnAllocation or pfnReallocation fail, the implementation **may** fail object creation and/or generate a VK_ERROR_OUT_OF_HOST_MEMORY error, as appropriate.

Allocation callbacks **must** not call any Vulkan commands.

The following sets of rules define when an implementation is permitted to call the allocator callbacks.

pfnAllocation or pfnReallocation **may** be called in the following situations:

- Allocations scoped to a VkDevice or VkInstance **may** be allocated from any API command.
- Allocations scoped to a command **may** be allocated from any API command.
- Allocations scoped to a VkPipelineCache **may** only be allocated from:
  - vkCreatePipelineCache
  - vkMergePipelineCaches for dstCache
  - vkCreateGraphicsPipelines for pipelineCache
  - vkCreateComputePipelines for pipelineCache
- Allocations scoped to a VkValidationCacheEXT **may** only be allocated from:
  - vkCreateValidationCacheEXT
  - vkMergeValidationCachesEXT for dstCache
  - vkCreateShaderModule for validationCache in VkShaderModuleValidationCacheCreateInfoEXT
- Allocations scoped to a VkDescriptorPool **may** only be allocated from:
  - any command that takes the pool as a direct argument
  - vkAllocateDescriptorSets for the descriptorPool member of its pAllocateInfo parameter
  - vkCreateDescriptorPool
- Allocations scoped to a VkCommandPool **may** only be allocated from:
  - any command that takes the pool as a direct argument
  - vkCreateCommandPool
  - vkAllocateCommandBuffers for the commandPool member of its pAllocateInfo parameter
  - any vkCmd* command whose commandBuffer was allocated from that VkCommandPool

- Allocations scoped to any other object **may** only be allocated in that object's `vkCreate*` command.

`pfnFree`, or `pfnReallocation` with zero `size`, **may** be called in the following situations:

- Allocations scoped to a `VkDevice` or `VkInstance` **may** be freed from any API command.
- Allocations scoped to a command **must** be freed by any API command which allocates such memory.
- Allocations scoped to a `VkPipelineCache` **may** be freed from `vkDestroyPipelineCache`.
- Allocations scoped to a `VkValidationCacheEXT` **may** be freed from `vkDestroyValidationCacheEXT`.
- Allocations scoped to a `VkDescriptorPool` **may** be freed from
  - any command that takes the pool as a direct argument
- Allocations scoped to a `VkCommandPool` **may** be freed from:
  - any command that takes the pool as a direct argument
  - `vkResetCommandBuffer` whose `commandBuffer` was allocated from that `VkCommandPool`
- Allocations scoped to any other object **may** be freed in that object's `vkDestroy*` command.
- Any command that allocates host memory **may** also free host memory of the same scope.

# 11.2. Device Memory

*Device memory* is memory that is visible to the device — for example the contents of the image or buffer objects, which **can** be natively used by the device.

## 11.2.1. Device Memory Properties

Memory properties of a physical device describe the memory heaps and memory types available.

To query memory properties, call:

```
// Provided by VK_VERSION_1_0
void vkGetPhysicalDeviceMemoryProperties(
    VkPhysicalDevice                            physicalDevice,
    VkPhysicalDeviceMemoryProperties*           pMemoryProperties);
```

- `physicalDevice` is the handle to the device to query.
- `pMemoryProperties` is a pointer to a VkPhysicalDeviceMemoryProperties structure in which the properties are returned.

**Valid Usage (Implicit)**

- VUID-vkGetPhysicalDeviceMemoryProperties-physicalDevice-parameter
  `physicalDevice` **must** be a valid VkPhysicalDevice handle
- VUID-vkGetPhysicalDeviceMemoryProperties-pMemoryProperties-parameter

pMemoryProperties **must** be a valid pointer to a VkPhysicalDeviceMemoryProperties structure

The VkPhysicalDeviceMemoryProperties structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkPhysicalDeviceMemoryProperties {
    uint32_t        memoryTypeCount;
    VkMemoryType    memoryTypes[VK_MAX_MEMORY_TYPES];
    uint32_t        memoryHeapCount;
    VkMemoryHeap    memoryHeaps[VK_MAX_MEMORY_HEAPS];
} VkPhysicalDeviceMemoryProperties;
```

- memoryTypeCount is the number of valid elements in the memoryTypes array.
- memoryTypes is an array of VK_MAX_MEMORY_TYPES VkMemoryType structures describing the *memory types* that **can** be used to access memory allocated from the heaps specified by memoryHeaps.
- memoryHeapCount is the number of valid elements in the memoryHeaps array.
- memoryHeaps is an array of VK_MAX_MEMORY_HEAPS VkMemoryHeap structures describing the *memory heaps* from which memory **can** be allocated.

The VkPhysicalDeviceMemoryProperties structure describes a number of *memory heaps* as well as a number of *memory types* that **can** be used to access memory allocated in those heaps. Each heap describes a memory resource of a particular size, and each memory type describes a set of memory properties (e.g. host cached vs. uncached) that **can** be used with a given memory heap. Allocations using a particular memory type will consume resources from the heap indicated by that memory type's heap index. More than one memory type **may** share each heap, and the heaps and memory types provide a mechanism to advertise an accurate size of the physical memory resources while allowing the memory to be used with a variety of different properties.

The number of memory heaps is given by memoryHeapCount and is less than or equal to VK_MAX_MEMORY_HEAPS. Each heap is described by an element of the memoryHeaps array as a VkMemoryHeap structure. The number of memory types available across all memory heaps is given by memoryTypeCount and is less than or equal to VK_MAX_MEMORY_TYPES. Each memory type is described by an element of the memoryTypes array as a VkMemoryType structure.

At least one heap **must** include VK_MEMORY_HEAP_DEVICE_LOCAL_BIT in VkMemoryHeap::flags. If there are multiple heaps that all have similar performance characteristics, they **may** all include VK_MEMORY_HEAP_DEVICE_LOCAL_BIT. In a unified memory architecture (UMA) system there is often only a single memory heap which is considered to be equally "local" to the host and to the device, and such an implementation **must** advertise the heap as device-local.

Each memory type returned by vkGetPhysicalDeviceMemoryProperties **must** have its propertyFlags set to one of the following values:

- 0

- VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT | VK_MEMORY_PROPERTY_HOST_COHERENT_BIT
- VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT | VK_MEMORY_PROPERTY_HOST_CACHED_BIT
- VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT | VK_MEMORY_PROPERTY_HOST_CACHED_BIT | VK_MEMORY_PROPERTY_HOST_COHERENT_BIT
- VK_MEMORY_PROPERTY_DEVICE_LOCAL_BIT
- VK_MEMORY_PROPERTY_DEVICE_LOCAL_BIT | VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT | VK_MEMORY_PROPERTY_HOST_COHERENT_BIT
- VK_MEMORY_PROPERTY_DEVICE_LOCAL_BIT | VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT | VK_MEMORY_PROPERTY_HOST_CACHED_BIT
- VK_MEMORY_PROPERTY_DEVICE_LOCAL_BIT | VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT | VK_MEMORY_PROPERTY_HOST_CACHED_BIT | VK_MEMORY_PROPERTY_HOST_COHERENT_BIT
- VK_MEMORY_PROPERTY_DEVICE_LOCAL_BIT | VK_MEMORY_PROPERTY_LAZILY_ALLOCATED_BIT
- VK_MEMORY_PROPERTY_PROTECTED_BIT
- VK_MEMORY_PROPERTY_PROTECTED_BIT | VK_MEMORY_PROPERTY_DEVICE_LOCAL_BIT
- VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT | VK_MEMORY_PROPERTY_HOST_COHERENT_BIT | VK_MEMORY_PROPERTY_DEVICE_COHERENT_BIT_AMD
- VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT | VK_MEMORY_PROPERTY_HOST_CACHED_BIT | VK_MEMORY_PROPERTY_HOST_COHERENT_BIT | VK_MEMORY_PROPERTY_DEVICE_COHERENT_BIT_AMD
- VK_MEMORY_PROPERTY_DEVICE_LOCAL_BIT | VK_MEMORY_PROPERTY_DEVICE_COHERENT_BIT_AMD
- VK_MEMORY_PROPERTY_DEVICE_LOCAL_BIT | VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT | VK_MEMORY_PROPERTY_HOST_COHERENT_BIT | VK_MEMORY_PROPERTY_DEVICE_COHERENT_BIT_AMD
- VK_MEMORY_PROPERTY_DEVICE_LOCAL_BIT | VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT | VK_MEMORY_PROPERTY_HOST_CACHED_BIT | VK_MEMORY_PROPERTY_HOST_COHERENT_BIT | VK_MEMORY_PROPERTY_DEVICE_COHERENT_BIT_AMD
- VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT | VK_MEMORY_PROPERTY_HOST_COHERENT_BIT |

VK_MEMORY_PROPERTY_DEVICE_COHERENT_BIT_AMD |
VK_MEMORY_PROPERTY_DEVICE_UNCACHED_BIT_AMD

- VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT |
  VK_MEMORY_PROPERTY_HOST_CACHED_BIT |
  VK_MEMORY_PROPERTY_HOST_COHERENT_BIT |
  VK_MEMORY_PROPERTY_DEVICE_COHERENT_BIT_AMD |
  VK_MEMORY_PROPERTY_DEVICE_UNCACHED_BIT_AMD
- VK_MEMORY_PROPERTY_DEVICE_LOCAL_BIT |
  VK_MEMORY_PROPERTY_DEVICE_COHERENT_BIT_AMD |
  VK_MEMORY_PROPERTY_DEVICE_UNCACHED_BIT_AMD
- VK_MEMORY_PROPERTY_DEVICE_LOCAL_BIT |
  VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT |
  VK_MEMORY_PROPERTY_HOST_COHERENT_BIT |
  VK_MEMORY_PROPERTY_DEVICE_COHERENT_BIT_AMD |
  VK_MEMORY_PROPERTY_DEVICE_UNCACHED_BIT_AMD
- VK_MEMORY_PROPERTY_DEVICE_LOCAL_BIT |
  VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT |
  VK_MEMORY_PROPERTY_HOST_CACHED_BIT |
  VK_MEMORY_PROPERTY_HOST_COHERENT_BIT |
  VK_MEMORY_PROPERTY_DEVICE_COHERENT_BIT_AMD |
  VK_MEMORY_PROPERTY_DEVICE_UNCACHED_BIT_AMD
- VK_MEMORY_PROPERTY_DEVICE_LOCAL_BIT |
  VK_MEMORY_PROPERTY_RDMA_CAPABLE_BIT_NV

There **must** be at least one memory type with both the VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT and VK_MEMORY_PROPERTY_HOST_COHERENT_BIT bits set in its propertyFlags. There **must** be at least one memory type with the VK_MEMORY_PROPERTY_DEVICE_LOCAL_BIT bit set in its propertyFlags. If the deviceCoherentMemory feature is enabled, there **must** be at least one memory type with the VK_MEMORY_PROPERTY_DEVICE_COHERENT_BIT_AMD bit set in its propertyFlags.

For each pair of elements **X** and **Y** returned in memoryTypes, **X** **must** be placed at a lower index position than **Y** if:

- the set of bit flags returned in the propertyFlags member of **X** is a strict subset of the set of bit flags returned in the propertyFlags member of **Y**; or
- the propertyFlags members of **X** and **Y** are equal, and **X** belongs to a memory heap with greater performance (as determined in an implementation-specific manner) ; or
- the propertyFlags members of **Y** includes VK_MEMORY_PROPERTY_DEVICE_COHERENT_BIT_AMD or VK_MEMORY_PROPERTY_DEVICE_UNCACHED_BIT_AMD and **X** does not

**NOTE** There is no ordering requirement between **X** and **Y** elements for the case their propertyFlags members are not in a subset relation. That potentially allows more than one possible way to order the same set of memory types. Notice that the list of all allowed memory property flag combinations is written in a valid order. But if instead VK_MEMORY_PROPERTY_DEVICE_LOCAL_BIT was before VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT | VK_MEMORY_PROPERTY_HOST_COHERENT_BIT, the

> list would still be in a valid order.
>
> There may be a performance penalty for using device coherent or uncached device memory types, and using these accidentally is undesirable. In order to avoid this, memory types with these properties always appear at the end of the list; but are subject to the same rules otherwise.

This ordering requirement enables applications to use a simple search loop to select the desired memory type along the lines of:

```
// Find a memory in `memoryTypeBitsRequirement` that includes all of
`requiredProperties`
int32_t findProperties(const VkPhysicalDeviceMemoryProperties* pMemoryProperties,
                       uint32_t memoryTypeBitsRequirement,
                       VkMemoryPropertyFlags requiredProperties) {
    const uint32_t memoryCount = pMemoryProperties->memoryTypeCount;
    for (uint32_t memoryIndex = 0; memoryIndex < memoryCount; ++memoryIndex) {
        const uint32_t memoryTypeBits = (1 << memoryIndex);
        const bool isRequiredMemoryType = memoryTypeBitsRequirement & memoryTypeBits;

        const VkMemoryPropertyFlags properties =
            pMemoryProperties->memoryTypes[memoryIndex].propertyFlags;
        const bool hasRequiredProperties =
            (properties & requiredProperties) == requiredProperties;

        if (isRequiredMemoryType && hasRequiredProperties)
            return static_cast<int32_t>(memoryIndex);
    }

    // failed to find memory type
    return -1;
}

// Try to find an optimal memory type, or if it does not exist try fallback memory
type
// `device` is the VkDevice
// `image` is the VkImage that requires memory to be bound
// `memoryProperties` properties as returned by vkGetPhysicalDeviceMemoryProperties
// `requiredProperties` are the property flags that must be present
// `optimalProperties` are the property flags that are preferred by the application
VkMemoryRequirements memoryRequirements;
vkGetImageMemoryRequirements(device, image, &memoryRequirements);
int32_t memoryType =
    findProperties(&memoryProperties, memoryRequirements.memoryTypeBits,
optimalProperties);
if (memoryType == -1) // not found; try fallback properties
    memoryType =
        findProperties(&memoryProperties, memoryRequirements.memoryTypeBits,
requiredProperties);
```

VK_MAX_MEMORY_TYPES is the length of an array of VkMemoryType structures describing memory types, as returned in VkPhysicalDeviceMemoryProperties::memoryTypes.

```
#define VK_MAX_MEMORY_TYPES              32U
```

VK_MAX_MEMORY_HEAPS is the length of an array of VkMemoryHeap structures describing memory heaps, as returned in VkPhysicalDeviceMemoryProperties::memoryHeaps.

```
#define VK_MAX_MEMORY_HEAPS              16U
```

To query memory properties, call:

```
// Provided by VK_VERSION_1_1
void vkGetPhysicalDeviceMemoryProperties2(
    VkPhysicalDevice                            physicalDevice,
    VkPhysicalDeviceMemoryProperties2*          pMemoryProperties);
```

or the equivalent command

```
// Provided by VK_KHR_get_physical_device_properties2
void vkGetPhysicalDeviceMemoryProperties2KHR(
    VkPhysicalDevice                            physicalDevice,
    VkPhysicalDeviceMemoryProperties2*          pMemoryProperties);
```

- physicalDevice is the handle to the device to query.
- pMemoryProperties is a pointer to a VkPhysicalDeviceMemoryProperties2 structure in which the properties are returned.

vkGetPhysicalDeviceMemoryProperties2 behaves similarly to vkGetPhysicalDeviceMemoryProperties, with the ability to return extended information in a pNext chain of output structures.

### Valid Usage (Implicit)

- VUID-vkGetPhysicalDeviceMemoryProperties2-physicalDevice-parameter physicalDevice **must** be a valid VkPhysicalDevice handle
- VUID-vkGetPhysicalDeviceMemoryProperties2-pMemoryProperties-parameter pMemoryProperties **must** be a valid pointer to a VkPhysicalDeviceMemoryProperties2 structure

The VkPhysicalDeviceMemoryProperties2 structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkPhysicalDeviceMemoryProperties2 {
    VkStructureType                     sType;
    void*                               pNext;
    VkPhysicalDeviceMemoryProperties    memoryProperties;
} VkPhysicalDeviceMemoryProperties2;
```

or the equivalent

```
// Provided by VK_KHR_get_physical_device_properties2
typedef VkPhysicalDeviceMemoryProperties2 VkPhysicalDeviceMemoryProperties2KHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `memoryProperties` is a VkPhysicalDeviceMemoryProperties structure which is populated with the same values as in vkGetPhysicalDeviceMemoryProperties.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceMemoryProperties2-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MEMORY_PROPERTIES_2`
- VUID-VkPhysicalDeviceMemoryProperties2-pNext-pNext
  `pNext` **must** be `NULL` or a pointer to a valid instance of VkPhysicalDeviceMemoryBudgetPropertiesEXT
- VUID-VkPhysicalDeviceMemoryProperties2-sType-unique
  The `sType` value of each struct in the `pNext` chain **must** be unique

The `VkMemoryHeap` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkMemoryHeap {
    VkDeviceSize        size;
    VkMemoryHeapFlags   flags;
} VkMemoryHeap;
```

- `size` is the total memory size in bytes in the heap.
- `flags` is a bitmask of VkMemoryHeapFlagBits specifying attribute flags for the heap.

Bits which **may** be set in VkMemoryHeap::`flags`, indicating attribute flags for the heap, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkMemoryHeapFlagBits {
    VK_MEMORY_HEAP_DEVICE_LOCAL_BIT = 0x00000001,
  // Provided by VK_VERSION_1_1
    VK_MEMORY_HEAP_MULTI_INSTANCE_BIT = 0x00000002,
  // Provided by VK_KHR_device_group_creation
    VK_MEMORY_HEAP_MULTI_INSTANCE_BIT_KHR = VK_MEMORY_HEAP_MULTI_INSTANCE_BIT,
} VkMemoryHeapFlagBits;
```

- `VK_MEMORY_HEAP_DEVICE_LOCAL_BIT` specifies that the heap corresponds to device-local memory. Device-local memory **may** have different performance characteristics than host-local memory, and **may** support different memory property flags.
- `VK_MEMORY_HEAP_MULTI_INSTANCE_BIT` specifies that in a logical device representing more than one physical device, there is a per-physical device instance of the heap memory. By default, an allocation from such a heap will be replicated to each physical device's instance of the heap.

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkMemoryHeapFlags;
```

`VkMemoryHeapFlags` is a bitmask type for setting a mask of zero or more VkMemoryHeapFlagBits.

The `VkMemoryType` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkMemoryType {
    VkMemoryPropertyFlags    propertyFlags;
    uint32_t                 heapIndex;
} VkMemoryType;
```

- `heapIndex` describes which memory heap this memory type corresponds to, and **must** be less than `memoryHeapCount` from the VkPhysicalDeviceMemoryProperties structure.
- `propertyFlags` is a bitmask of VkMemoryPropertyFlagBits of properties for this memory type.

Bits which **may** be set in VkMemoryType::`propertyFlags`, indicating properties of a memory type, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkMemoryPropertyFlagBits {
    VK_MEMORY_PROPERTY_DEVICE_LOCAL_BIT = 0x00000001,
    VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT = 0x00000002,
    VK_MEMORY_PROPERTY_HOST_COHERENT_BIT = 0x00000004,
    VK_MEMORY_PROPERTY_HOST_CACHED_BIT = 0x00000008,
    VK_MEMORY_PROPERTY_LAZILY_ALLOCATED_BIT = 0x00000010,
  // Provided by VK_VERSION_1_1
    VK_MEMORY_PROPERTY_PROTECTED_BIT = 0x00000020,
  // Provided by VK_AMD_device_coherent_memory
    VK_MEMORY_PROPERTY_DEVICE_COHERENT_BIT_AMD = 0x00000040,
  // Provided by VK_AMD_device_coherent_memory
    VK_MEMORY_PROPERTY_DEVICE_UNCACHED_BIT_AMD = 0x00000080,
  // Provided by VK_NV_external_memory_rdma
    VK_MEMORY_PROPERTY_RDMA_CAPABLE_BIT_NV = 0x00000100,
} VkMemoryPropertyFlagBits;
```

- `VK_MEMORY_PROPERTY_DEVICE_LOCAL_BIT` bit specifies that memory allocated with this type is the most efficient for device access. This property will be set if and only if the memory type belongs to a heap with the `VK_MEMORY_HEAP_DEVICE_LOCAL_BIT` set.
- `VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT` bit specifies that memory allocated with this type **can** be mapped for host access using vkMapMemory.
- `VK_MEMORY_PROPERTY_HOST_COHERENT_BIT` bit specifies that the host cache management commands vkFlushMappedMemoryRanges and vkInvalidateMappedMemoryRanges are not needed to flush host writes to the device or make device writes visible to the host, respectively.
- `VK_MEMORY_PROPERTY_HOST_CACHED_BIT` bit specifies that memory allocated with this type is cached on the host. Host memory accesses to uncached memory are slower than to cached memory, however uncached memory is always host coherent.
- `VK_MEMORY_PROPERTY_LAZILY_ALLOCATED_BIT` bit specifies that the memory type only allows device access to the memory. Memory types **must** not have both `VK_MEMORY_PROPERTY_LAZILY_ALLOCATED_BIT` and `VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT` set. Additionally, the object's backing memory **may** be provided by the implementation lazily as specified in Lazily Allocated Memory.
- `VK_MEMORY_PROPERTY_PROTECTED_BIT` bit specifies that the memory type only allows device access to the memory, and allows protected queue operations to access the memory. Memory types **must** not have `VK_MEMORY_PROPERTY_PROTECTED_BIT` set and any of `VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT` set, or `VK_MEMORY_PROPERTY_HOST_COHERENT_BIT` set, or `VK_MEMORY_PROPERTY_HOST_CACHED_BIT` set.
- `VK_MEMORY_PROPERTY_DEVICE_COHERENT_BIT_AMD` bit specifies that device accesses to allocations of this memory type are automatically made available and visible.
- `VK_MEMORY_PROPERTY_DEVICE_UNCACHED_BIT_AMD` bit specifies that memory allocated with this type is not cached on the device. Uncached device memory is always device coherent.
- `VK_MEMORY_PROPERTY_RDMA_CAPABLE_BIT_NV` bit specifies that external devices can access this memory directly.

For any memory allocated with both the VK_MEMORY_PROPERTY_HOST_COHERENT_BIT and the VK_MEMORY_PROPERTY_DEVICE_COHERENT_BIT_AMD, host or device accesses also perform automatic memory domain transfer operations, such that writes are always automatically available and visible to both host and device memory domains.

**NOTE** Device coherence is a useful property for certain debugging use cases (e.g. crash analysis, where performing separate coherence actions could mean values are not reported correctly). However, device coherent accesses may be slower than equivalent accesses without device coherence, particularly if they are also device uncached. For device uncached memory in particular, repeated accesses to the same or neighboring memory locations over a short time period (e.g. within a frame) may be slower than it would be for the equivalent cached memory type. As such, it is generally inadvisable to use device coherent or device uncached memory except when really needed.

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkMemoryPropertyFlags;
```

VkMemoryPropertyFlags is a bitmask type for setting a mask of zero or more VkMemoryPropertyFlagBits.

If the VkPhysicalDeviceMemoryBudgetPropertiesEXT structure is included in the pNext chain of VkPhysicalDeviceMemoryProperties2, it is filled with the current memory budgets and usages.

The VkPhysicalDeviceMemoryBudgetPropertiesEXT structure is defined as:

```
// Provided by VK_EXT_memory_budget
typedef struct VkPhysicalDeviceMemoryBudgetPropertiesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkDeviceSize       heapBudget[VK_MAX_MEMORY_HEAPS];
    VkDeviceSize       heapUsage[VK_MAX_MEMORY_HEAPS];
} VkPhysicalDeviceMemoryBudgetPropertiesEXT;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- heapBudget is an array of VK_MAX_MEMORY_HEAPS VkDeviceSize values in which memory budgets are returned, with one element for each memory heap. A heap's budget is a rough estimate of how much memory the process **can** allocate from that heap before allocations **may** fail or cause performance degradation. The budget includes any currently allocated device memory.
- heapUsage is an array of VK_MAX_MEMORY_HEAPS VkDeviceSize values in which memory usages are returned, with one element for each memory heap. A heap's usage is an estimate of how much memory the process is currently using in that heap.

The values returned in this structure are not invariant. The heapBudget and heapUsage values **must** be zero for array elements greater than or equal to VkPhysicalDeviceMemoryProperties

::memoryHeapCount. The heapBudget value **must** be non-zero for array elements less than VkPhysicalDeviceMemoryProperties::memoryHeapCount. The heapBudget value **must** be less than or equal to VkMemoryHeap::size for each heap.

**Valid Usage (Implicit)**

- VUID-VkPhysicalDeviceMemoryBudgetPropertiesEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MEMORY_BUDGET_PROPERTIES_EXT

### 11.2.2. Device Memory Objects

A Vulkan device operates on data in device memory via memory objects that are represented in the API by a VkDeviceMemory handle:

```
// Provided by VK_VERSION_1_0
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkDeviceMemory)
```

### 11.2.3. Device Memory Allocation

To allocate memory objects, call:

```
// Provided by VK_VERSION_1_0
VkResult vkAllocateMemory(
    VkDevice                                    device,
    const VkMemoryAllocateInfo*                 pAllocateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkDeviceMemory*                             pMemory);
```

- device is the logical device that owns the memory.
- pAllocateInfo is a pointer to a VkMemoryAllocateInfo structure describing parameters of the allocation. A successfully returned allocation **must** use the requested parameters — no substitution is permitted by the implementation.
- pAllocator controls host memory allocation as described in the Memory Allocation chapter.
- pMemory is a pointer to a VkDeviceMemory handle in which information about the allocated memory is returned.

Allocations returned by vkAllocateMemory are guaranteed to meet any alignment requirement of the implementation. For example, if an implementation requires 128 byte alignment for images and 64 byte alignment for buffers, the device memory returned through this mechanism would be 128-byte aligned. This ensures that applications **can** correctly suballocate objects of different types (with potentially different alignment requirements) in the same memory object.

When memory is allocated, its contents are undefined with the following constraint:

- The contents of unprotected memory **must** not be a function of the contents of data protected

memory objects, even if those memory objects were previously freed.

**NOTE** The contents of memory allocated by one application **should** not be a function of data from protected memory objects of another application, even if those memory objects were previously freed.

The maximum number of valid memory allocations that **can** exist simultaneously within a VkDevice **may** be restricted by implementation- or platform-dependent limits. The `maxMemoryAllocationCount` feature describes the number of allocations that **can** exist simultaneously before encountering these internal limits.

**NOTE** For historical reasons, if `maxMemoryAllocationCount` is exceeded, some implementations may return `VK_ERROR_TOO_MANY_OBJECTS`. Exceeding this limit will result in undefined behavior, and an application should not rely on the use of the returned error code in order to identify when the limit is reached.

**NOTE** Many protected memory implementations involve complex hardware and system software support, and often have additional and much lower limits on the number of simultaneous protected memory allocations (from memory types with the `VK_MEMORY_PROPERTY_PROTECTED_BIT` property) than for non-protected memory allocations. These limits can be system-wide, and depend on a variety of factors outside of the Vulkan implementation, so they cannot be queried in Vulkan. Applications **should** use as few allocations as possible from such memory types by suballocating aggressively, and be prepared for allocation failure even when there is apparently plenty of capacity remaining in the memory heap. As a guideline, the Vulkan conformance test suite requires that at least 80 minimum-size allocations can exist concurrently when no other uses of protected memory are active in the system.

Some platforms **may** have a limit on the maximum size of a single allocation. For example, certain systems **may** fail to create allocations with a size greater than or equal to 4GB. Such a limit is implementation-dependent, and if such a failure occurs then the error `VK_ERROR_OUT_OF_DEVICE_MEMORY` **must** be returned. This limit is advertised in VkPhysicalDeviceMaintenance3Properties::`maxMemoryAllocationSize`.

The cumulative memory size allocated to a heap **can** be limited by the size of the specified heap. In such cases, allocated memory is tracked on a per-device and per-heap basis. Some platforms allow overallocation into other heaps. The overallocation behavior **can** be specified through the `VK_AMD_memory_overallocation_behavior` extension.

If the VkPhysicalDevicePageableDeviceLocalMemoryFeaturesEXT::`pageableDeviceLocalMemory` feature is enabled, memory allocations made from a heap that includes `VK_MEMORY_HEAP_DEVICE_LOCAL_BIT` in VkMemoryHeap::`flags` **may** be transparently moved to host-local memory allowing multiple applications to share device-local memory. If there is no space left in device-local memory when this new allocation is made, other allocations **may** be moved out transparently to make room. The operating system will determine which allocations to move to device-local memory or host-local memory based on platform-specific criteria. To help the operating system make good choices, the application **should** set the appropriate memory priority

with VkMemoryPriorityAllocateInfoEXT and adjust it as necessary with vkSetDeviceMemoryPriorityEXT. Higher priority allocations will moved to device-local memory first.

Memory allocations made on heaps without the `VK_MEMORY_HEAP_DEVICE_LOCAL_BIT` property will not be transparently promoted to device-local memory by the operating system.

## Valid Usage

- VUID-vkAllocateMemory-pAllocateInfo-01713
  `pAllocateInfo->allocationSize` **must** be less than or equal to VkPhysicalDeviceMemoryProperties::`memoryHeaps`[memindex].`size` where `memindex` = VkPhysicalDeviceMemoryProperties::`memoryTypes`[`pAllocateInfo->memoryTypeIndex`].`heapIndex` as returned by vkGetPhysicalDeviceMemoryProperties for the VkPhysicalDevice that `device` was created from
- VUID-vkAllocateMemory-pAllocateInfo-01714
  `pAllocateInfo->memoryTypeIndex` **must** be less than VkPhysicalDeviceMemoryProperties::`memoryTypeCount` as returned by vkGetPhysicalDeviceMemoryProperties for the VkPhysicalDevice that `device` was created from
- VUID-vkAllocateMemory-deviceCoherentMemory-02790
  If the `deviceCoherentMemory` feature is not enabled, `pAllocateInfo->memoryTypeIndex` **must** not identify a memory type supporting `VK_MEMORY_PROPERTY_DEVICE_COHERENT_BIT_AMD`
- VUID-vkAllocateMemory-maxMemoryAllocationCount-04101
  There **must** be less than `VkPhysicalDeviceLimits`::`maxMemoryAllocationCount` device memory allocations currently allocated on the device

## Valid Usage (Implicit)

- VUID-vkAllocateMemory-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkAllocateMemory-pAllocateInfo-parameter
  `pAllocateInfo` **must** be a valid pointer to a valid VkMemoryAllocateInfo structure
- VUID-vkAllocateMemory-pAllocator-parameter
  If `pAllocator` is not `NULL`, `pAllocator` **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkAllocateMemory-pMemory-parameter
  `pMemory` **must** be a valid pointer to a VkDeviceMemory handle

## Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY
- VK_ERROR_INVALID_EXTERNAL_HANDLE
- VK_ERROR_INVALID_OPAQUE_CAPTURE_ADDRESS_KHR

The VkMemoryAllocateInfo structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkMemoryAllocateInfo {
    VkStructureType    sType;
    const void*        pNext;
    VkDeviceSize       allocationSize;
    uint32_t           memoryTypeIndex;
} VkMemoryAllocateInfo;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- allocationSize is the size of the allocation in bytes.
- memoryTypeIndex is an index identifying a memory type from the memoryTypes array of the VkPhysicalDeviceMemoryProperties structure.

The internal data of an allocated device memory object **must** include a reference to implementation-specific resources, referred to as the memory object's *payload*. Applications **can** also import and export that internal data to and from device memory objects to share data between Vulkan instances and other compatible APIs. A VkMemoryAllocateInfo structure defines a memory import operation if its pNext chain includes one of the following structures:

- VkImportMemoryWin32HandleInfoKHR with a non-zero handleType value
- VkImportMemoryFdInfoKHR with a non-zero handleType value
- VkImportMemoryHostPointerInfoEXT with a non-zero handleType value
- VkImportAndroidHardwareBufferInfoANDROID with a non-NULL buffer value
- VkImportMemoryZirconHandleInfoFUCHSIA with a non-zero handleType value
- VkImportMemoryBufferCollectionFUCHSIA
- VkImportScreenBufferInfoQNX with a non-NULL buffer value

If the parameters define an import operation and the external handle type is VK_EXTERNAL_MEMORY_HANDLE_TYPE_D3D11_TEXTURE_BIT, VK_EXTERNAL_MEMORY_HANDLE_TYPE_D3D11_TEXTURE_KMT_BIT, or VK_EXTERNAL_MEMORY_HANDLE_TYPE_D3D12_RESOURCE_BIT, allocationSize is ignored. The implementation **must** query the size of these allocations from the OS.

Whether device memory objects constructed via a memory import operation hold a reference to

their payload depends on the properties of the handle type used to perform the import, as defined below for each valid handle type. Importing memory **must** not modify the content of the memory. Implementations **must** ensure that importing memory does not enable the importing Vulkan instance to access any memory or resources in other Vulkan instances other than that corresponding to the memory object imported. Implementations **must** also ensure accessing imported memory which has not been initialized does not allow the importing Vulkan instance to obtain data from the exporting Vulkan instance or vice-versa.

| | |
|---|---|
| **NOTE** | How exported and imported memory is isolated is left to the implementation, but applications should be aware that such isolation **may** prevent implementations from placing multiple exportable memory objects in the same physical or virtual page. Hence, applications **should** avoid creating many small external memory objects whenever possible. |

Importing memory **must** not increase overall heap usage within a system. However, it **must** affect the following per-process values:

- VkPhysicalDeviceLimits::`maxMemoryAllocationCount`
- VkPhysicalDeviceMemoryBudgetPropertiesEXT::`heapUsage`

When performing a memory import operation, it is the responsibility of the application to ensure the external handles and their associated payloads meet all valid usage requirements. However, implementations **must** perform sufficient validation of external handles and payloads to ensure that the operation results in a valid memory object which will not cause program termination, device loss, queue stalls, or corruption of other resources when used as allowed according to its allocation parameters. If the external handle provided does not meet these requirements, the implementation **must** fail the memory import operation with the error code `VK_ERROR_INVALID_EXTERNAL_HANDLE`. If the parameters define an export operation and the external handle type is `VK_EXTERNAL_MEMORY_HANDLE_TYPE_ANDROID_HARDWARE_BUFFER_BIT_ANDROID`, implementations **should** not strictly follow `memoryTypeIndex`. Instead, they **should** modify the allocation internally to use the required memory type for the application's given usage. This is because for an export operation, there is currently no way for the application to know the memory type index before allocating.

### Valid Usage

- VUID-VkMemoryAllocateInfo-allocationSize-07897
  If the parameters do not define an import or export operation, `allocationSize` **must** be greater than `0`
- VUID-VkMemoryAllocateInfo-None-06657
  The parameters **must** not define more than one import operation
- VUID-VkMemoryAllocateInfo-allocationSize-07899
  If the parameters define an export operation and the handle type is not `VK_EXTERNAL_MEMORY_HANDLE_TYPE_ANDROID_HARDWARE_BUFFER_BIT_ANDROID`, `allocationSize` **must** be greater than `0`
- VUID-VkMemoryAllocateInfo-buffer-06380

If the parameters define an import operation from an VkBufferCollectionFUCHSIA, and VkMemoryDedicatedAllocateInfo::buffer is present and non-NULL, VkImportMemoryBufferCollectionFUCHSIA::collection and VkImportMemoryBufferCollectionFUCHSIA::index **must** match VkBufferCollectionBufferCreateInfoFUCHSIA::collection and VkBufferCollectionBufferCreateInfoFUCHSIA::index, respectively, of the VkBufferCollectionBufferCreateInfoFUCHSIA structure used to create the VkMemoryDedicatedAllocateInfo::buffer

- VUID-VkMemoryAllocateInfo-image-06381
  If the parameters define an import operation from an VkBufferCollectionFUCHSIA, and VkMemoryDedicatedAllocateInfo::image is present and non-NULL, VkImportMemoryBufferCollectionFUCHSIA::collection and VkImportMemoryBufferCollectionFUCHSIA::index **must** match VkBufferCollectionImageCreateInfoFUCHSIA::collection and VkBufferCollectionImageCreateInfoFUCHSIA::index, respectively, of the VkBufferCollectionImageCreateInfoFUCHSIA structure used to create the VkMemoryDedicatedAllocateInfo::image
- VUID-VkMemoryAllocateInfo-allocationSize-06382
  If the parameters define an import operation from an VkBufferCollectionFUCHSIA, allocationSize **must** match VkMemoryRequirements::size value retrieved by vkGetImageMemoryRequirements or vkGetBufferMemoryRequirements for image-based or buffer-based collections respectively
- VUID-VkMemoryAllocateInfo-pNext-06383
  If the parameters define an import operation from an VkBufferCollectionFUCHSIA, the pNext chain **must** include a VkMemoryDedicatedAllocateInfo structure with either its image or buffer field set to a value other than VK_NULL_HANDLE
- VUID-VkMemoryAllocateInfo-image-06384
  If the parameters define an import operation from an VkBufferCollectionFUCHSIA and VkMemoryDedicatedAllocateInfo::image is not VK_NULL_HANDLE, the image **must** be created with a VkBufferCollectionImageCreateInfoFUCHSIA structure chained to its VkImageCreateInfo::pNext pointer
- VUID-VkMemoryAllocateInfo-buffer-06385
  If the parameters define an import operation from an VkBufferCollectionFUCHSIA and VkMemoryDedicatedAllocateInfo::buffer is not VK_NULL_HANDLE, the buffer **must** be created with a VkBufferCollectionBufferCreateInfoFUCHSIA structure chained to its VkBufferCreateInfo::pNext pointer
- VUID-VkMemoryAllocateInfo-memoryTypeIndex-06386
  If the parameters define an import operation from an VkBufferCollectionFUCHSIA, memoryTypeIndex **must** be from VkBufferCollectionPropertiesFUCHSIA as retrieved by vkGetBufferCollectionPropertiesFUCHSIA
- VUID-VkMemoryAllocateInfo-pNext-00639
  If the pNext chain includes a VkExportMemoryAllocateInfo structure, and any of the handle types specified in VkExportMemoryAllocateInfo::handleTypes require a dedicated allocation, as reported by vkGetPhysicalDeviceImageFormatProperties2 in VkExternalImageFormatProperties::externalMemoryProperties.externalMemoryFeatures, or by

vkGetPhysicalDeviceExternalBufferProperties in VkExternalBufferProperties::externalMemoryProperties.externalMemoryFeatures, the pNext chain **must** include a VkMemoryDedicatedAllocateInfo or VkDedicatedAllocationMemoryAllocateInfoNV structure with either its image or buffer member set to a value other than VK_NULL_HANDLE

- VUID-VkMemoryAllocateInfo-pNext-00640
  If the pNext chain includes a VkExportMemoryAllocateInfo structure, it **must** not include a VkExportMemoryAllocateInfoNV or VkExportMemoryWin32HandleInfoNV structure
- VUID-VkMemoryAllocateInfo-pNext-00641
  If the pNext chain includes a VkImportMemoryWin32HandleInfoKHR structure, it **must** not include a VkImportMemoryWin32HandleInfoNV structure
- VUID-VkMemoryAllocateInfo-allocationSize-01742
  If the parameters define an import operation, the external handle specified was created by the Vulkan API, and the external handle type is VK_EXTERNAL_MEMORY_HANDLE_TYPE_OPAQUE_FD_BIT, then the values of allocationSize and memoryTypeIndex **must** match those specified when the payload being imported was created
- VUID-VkMemoryAllocateInfo-None-00643
  If the parameters define an import operation and the external handle specified was created by the Vulkan API, the device mask specified by VkMemoryAllocateFlagsInfo **must** match the mask specified when the payload being imported was allocated
- VUID-VkMemoryAllocateInfo-None-00644
  If the parameters define an import operation and the external handle specified was created by the Vulkan API, the list of physical devices that comprise the logical device passed to vkAllocateMemory **must** match the list of physical devices that comprise the logical device on which the payload was originally allocated
- VUID-VkMemoryAllocateInfo-memoryTypeIndex-00645
  If the parameters define an import operation and the external handle is an NT handle or a global share handle created outside of the Vulkan API, the value of memoryTypeIndex **must** be one of those returned by vkGetMemoryWin32HandlePropertiesKHR
- VUID-VkMemoryAllocateInfo-allocationSize-01743
  If the parameters define an import operation, the external handle was created by the Vulkan API, and the external handle type is VK_EXTERNAL_MEMORY_HANDLE_TYPE_OPAQUE_WIN32_BIT or VK_EXTERNAL_MEMORY_HANDLE_TYPE_OPAQUE_WIN32_KMT_BIT, then the values of allocationSize and memoryTypeIndex **must** match those specified when the payload being imported was created
- VUID-VkMemoryAllocateInfo-allocationSize-00647
  If the parameters define an import operation and the external handle type is VK_EXTERNAL_MEMORY_HANDLE_TYPE_D3D12_HEAP_BIT, allocationSize **must** match the size specified when creating the Direct3D 12 heap from which the payload was extracted
- VUID-VkMemoryAllocateInfo-memoryTypeIndex-00648
  If the parameters define an import operation and the external handle is a POSIX file descriptor created outside of the Vulkan API, the value of memoryTypeIndex **must** be one of those returned by vkGetMemoryFdPropertiesKHR

- VUID-VkMemoryAllocateInfo-memoryTypeIndex-01872
  If the `protectedMemory` feature is not enabled, the `VkMemoryAllocateInfo::memoryTypeIndex` **must** not indicate a memory type that reports `VK_MEMORY_PROPERTY_PROTECTED_BIT`
- VUID-VkMemoryAllocateInfo-memoryTypeIndex-01744
  If the parameters define an import operation and the external handle is a host pointer, the value of `memoryTypeIndex` **must** be one of those returned by vkGetMemoryHostPointerPropertiesEXT
- VUID-VkMemoryAllocateInfo-allocationSize-01745
  If the parameters define an import operation and the external handle is a host pointer, `allocationSize` **must** be an integer multiple of `VkPhysicalDeviceExternalMemoryHostPropertiesEXT::minImportedHostPointerAlignment`
- VUID-VkMemoryAllocateInfo-pNext-02805
  If the parameters define an import operation and the external handle is a host pointer, the `pNext` chain **must** not include a VkDedicatedAllocationMemoryAllocateInfoNV structure with either its `image` or `buffer` field set to a value other than VK_NULL_HANDLE
- VUID-VkMemoryAllocateInfo-pNext-02806
  If the parameters define an import operation and the external handle is a host pointer, the `pNext` chain **must** not include a VkMemoryDedicatedAllocateInfo structure with either its `image` or `buffer` field set to a value other than VK_NULL_HANDLE
- VUID-VkMemoryAllocateInfo-allocationSize-02383
  If the parameters define an import operation and the external handle type is `VK_EXTERNAL_MEMORY_HANDLE_TYPE_ANDROID_HARDWARE_BUFFER_BIT_ANDROID`, `allocationSize` **must** be the size returned by vkGetAndroidHardwareBufferPropertiesANDROID for the Android hardware buffer
- VUID-VkMemoryAllocateInfo-pNext-02384
  If the parameters define an import operation and the external handle type is `VK_EXTERNAL_MEMORY_HANDLE_TYPE_ANDROID_HARDWARE_BUFFER_BIT_ANDROID`, and the `pNext` chain does not include a VkMemoryDedicatedAllocateInfo structure or VkMemoryDedicatedAllocateInfo::`image` is VK_NULL_HANDLE, the Android hardware buffer **must** have a `AHardwareBuffer_Desc::format` of `AHARDWAREBUFFER_FORMAT_BLOB` and a `AHardwareBuffer_Desc::usage` that includes `AHARDWAREBUFFER_USAGE_GPU_DATA_BUFFER`
- VUID-VkMemoryAllocateInfo-memoryTypeIndex-02385
  If the parameters define an import operation and the external handle type is `VK_EXTERNAL_MEMORY_HANDLE_TYPE_ANDROID_HARDWARE_BUFFER_BIT_ANDROID`, `memoryTypeIndex` **must** be one of those returned by vkGetAndroidHardwareBufferPropertiesANDROID for the Android hardware buffer
- VUID-VkMemoryAllocateInfo-pNext-01874
  If the parameters do not define an import operation, and the `pNext` chain includes a `VkExportMemoryAllocateInfo` structure with `VK_EXTERNAL_MEMORY_HANDLE_TYPE_ANDROID_HARDWARE_BUFFER_BIT_ANDROID` included in its `handleTypes` member, and the `pNext` chain includes a VkMemoryDedicatedAllocateInfo structure with `image` not equal to VK_NULL_HANDLE, then `allocationSize` **must** be `0`
- VUID-VkMemoryAllocateInfo-pNext-07900
  If the parameters define an export operation, the handle type is

VK_EXTERNAL_MEMORY_HANDLE_TYPE_ANDROID_HARDWARE_BUFFER_BIT_ANDROID, and the pNext does not include a VkMemoryDedicatedAllocateInfo structure, allocationSize **must** be greater than 0

- VUID-VkMemoryAllocateInfo-pNext-07901
  If the parameters define an export operation, the handle type is VK_EXTERNAL_MEMORY_HANDLE_TYPE_ANDROID_HARDWARE_BUFFER_BIT_ANDROID, and the pNext chain includes a VkMemoryDedicatedAllocateInfo structure with buffer set to a valid VkBuffer object, allocationSize **must** be greater than 0
- VUID-VkMemoryAllocateInfo-pNext-02386
  If the parameters define an import operation, the external handle is an Android hardware buffer, and the pNext chain includes a VkMemoryDedicatedAllocateInfo with image that is not VK_NULL_HANDLE, the Android hardware buffer's AHardwareBuffer::usage **must** include at least one of AHARDWAREBUFFER_USAGE_GPU_FRAMEBUFFER, AHARDWAREBUFFER_USAGE_GPU_SAMPLED_IMAGE or AHARDWAREBUFFER_USAGE_GPU_DATA_BUFFER
- VUID-VkMemoryAllocateInfo-pNext-02387
  If the parameters define an import operation, the external handle is an Android hardware buffer, and the pNext chain includes a VkMemoryDedicatedAllocateInfo with image that is not VK_NULL_HANDLE, the format of image **must** be VK_FORMAT_UNDEFINED or the format returned by vkGetAndroidHardwareBufferPropertiesANDROID in VkAndroidHardwareBufferFormatPropertiesANDROID::format for the Android hardware buffer
- VUID-VkMemoryAllocateInfo-pNext-02388
  If the parameters define an import operation, the external handle is an Android hardware buffer, and the pNext chain includes a VkMemoryDedicatedAllocateInfo structure with image that is not VK_NULL_HANDLE, the width, height, and array layer dimensions of image and the Android hardware buffer's AHardwareBuffer_Desc **must** be identical
- VUID-VkMemoryAllocateInfo-pNext-02389
  If the parameters define an import operation, the external handle is an Android hardware buffer, and the pNext chain includes a VkMemoryDedicatedAllocateInfo structure with image that is not VK_NULL_HANDLE, and the Android hardware buffer's AHardwareBuffer::usage includes AHARDWAREBUFFER_USAGE_GPU_MIPMAP_COMPLETE, the image **must** have a complete mipmap chain
- VUID-VkMemoryAllocateInfo-pNext-02586
  If the parameters define an import operation, the external handle is an Android hardware buffer, and the pNext chain includes a VkMemoryDedicatedAllocateInfo structure with image that is not VK_NULL_HANDLE, and the Android hardware buffer's AHardwareBuffer::usage does not include AHARDWAREBUFFER_USAGE_GPU_MIPMAP_COMPLETE, the image **must** have exactly one mipmap level
- VUID-VkMemoryAllocateInfo-pNext-02390
  If the parameters define an import operation, the external handle is an Android hardware buffer, and the pNext chain includes a VkMemoryDedicatedAllocateInfo structure with image that is not VK_NULL_HANDLE, each bit set in the usage of image **must** be listed in AHardwareBuffer Usage Equivalence, and if there is a corresponding AHARDWAREBUFFER_USAGE bit listed that bit **must** be included in the Android hardware buffer's AHardwareBuffer_Desc::usage

- VUID-VkMemoryAllocateInfo-screenBufferImport-08941
  If the parameters define an import operation and the external handle type is `VK_EXTERNAL_MEMORY_HANDLE_TYPE_SCREEN_BUFFER_BIT_QNX`, VkPhysicalDeviceExternalMemoryScreenBufferFeaturesQNX::`screenBufferImport` **must** be enabled
- VUID-VkMemoryAllocateInfo-allocationSize-08942
  If the parameters define an import operation and the external handle type is `VK_EXTERNAL_MEMORY_HANDLE_TYPE_SCREEN_BUFFER_BIT_QNX`, `allocationSize` **must** be the size returned by vkGetScreenBufferPropertiesQNX for the QNX Screen buffer
- VUID-VkMemoryAllocateInfo-memoryTypeIndex-08943
  If the parameters define an import operation and the external handle type is `VK_EXTERNAL_MEMORY_HANDLE_TYPE_SCREEN_BUFFER_BIT_QNX`, `memoryTypeIndex` **must** be one of those returned by vkGetScreenBufferPropertiesQNX for the QNX Screen buffer
- VUID-VkMemoryAllocateInfo-pNext-08944
  If the parameters define an import operation, the external handle is a QNX Screen buffer, and the `pNext` chain includes a VkMemoryDedicatedAllocateInfo with `image` that is not VK_NULL_HANDLE, the QNX Screen's buffer **must** be a valid QNX Screen buffer
- VUID-VkMemoryAllocateInfo-pNext-08945
  If the parameters define an import operation, the external handle is an QNX Screen buffer, and the `pNext` chain includes a VkMemoryDedicatedAllocateInfo with `image` that is not VK_NULL_HANDLE, the format of `image` **must** be `VK_FORMAT_UNDEFINED` or the format returned by vkGetScreenBufferPropertiesQNX in VkScreenBufferFormatPropertiesQNX::`format` for the QNX Screen buffer
- VUID-VkMemoryAllocateInfo-pNext-08946
  If the parameters define an import operation, the external handle is a QNX Screen buffer, and the `pNext` chain includes a VkMemoryDedicatedAllocateInfo structure with `image` that is not VK_NULL_HANDLE, the width, height, and array layer dimensions of `image` and the QNX Screen buffer's `_screen_buffer` **must** be identical
- VUID-VkMemoryAllocateInfo-opaqueCaptureAddress-03329
  If VkMemoryOpaqueCaptureAddressAllocateInfo::`opaqueCaptureAddress` is not zero, `VkMemoryAllocateFlagsInfo`::`flags` **must** include `VK_MEMORY_ALLOCATE_DEVICE_ADDRESS_CAPTURE_REPLAY_BIT`
- VUID-VkMemoryAllocateInfo-flags-03330
  If `VkMemoryAllocateFlagsInfo`::`flags` includes `VK_MEMORY_ALLOCATE_DEVICE_ADDRESS_CAPTURE_REPLAY_BIT`, the `bufferDeviceAddressCaptureReplay` feature **must** be enabled
- VUID-VkMemoryAllocateInfo-flags-03331
  If `VkMemoryAllocateFlagsInfo`::`flags` includes `VK_MEMORY_ALLOCATE_DEVICE_ADDRESS_BIT`, the `bufferDeviceAddress` feature **must** be enabled
- VUID-VkMemoryAllocateInfo-pNext-03332
  If the `pNext` chain includes a `VkImportMemoryHostPointerInfoEXT` structure, VkMemoryOpaqueCaptureAddressAllocateInfo::`opaqueCaptureAddress` **must** be zero
- VUID-VkMemoryAllocateInfo-opaqueCaptureAddress-03333
  If the parameters define an import operation,

VkMemoryOpaqueCaptureAddressAllocateInfo::opaqueCaptureAddress **must** be zero

- VUID-VkMemoryAllocateInfo-None-04749
  If the parameters define an import operation and the external handle type is VK_EXTERNAL_MEMORY_HANDLE_TYPE_ZIRCON_VMO_BIT_FUCHSIA, the value of memoryTypeIndex **must** be an index identifying a memory type from the memoryTypeBits field of the VkMemoryZirconHandlePropertiesFUCHSIA structure populated by a call to vkGetMemoryZirconHandlePropertiesFUCHSIA
- VUID-VkMemoryAllocateInfo-allocationSize-07902
  If the parameters define an import operation and the external handle type is VK_EXTERNAL_MEMORY_HANDLE_TYPE_ZIRCON_VMO_BIT_FUCHSIA, the value of allocationSize **must** be greater than 0
- VUID-VkMemoryAllocateInfo-allocationSize-07903
  If the parameters define an import operation and the external handle type is VK_EXTERNAL_MEMORY_HANDLE_TYPE_ZIRCON_VMO_BIT_FUCHSIA, the value of allocationSize **must** be less than or equal to the size of the VMO as determined by zx_vmo_get_size(handle) where handle is the VMO handle to the imported external memory
- VUID-VkMemoryAllocateInfo-pNext-06780
  If the pNext chain includes a VkExportMetalObjectCreateInfoEXT structure, its exportObjectType member **must** be VK_EXPORT_METAL_OBJECT_TYPE_METAL_BUFFER_BIT_EXT

## Valid Usage (Implicit)

- VUID-VkMemoryAllocateInfo-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_MEMORY_ALLOCATE_INFO
- VUID-VkMemoryAllocateInfo-pNext-pNext
  Each pNext member of any structure (including this one) in the pNext chain **must** be either NULL or a pointer to a valid instance of VkDedicatedAllocationMemoryAllocateInfoNV, VkExportMemoryAllocateInfo, VkExportMemoryAllocateInfoNV, VkExportMemoryWin32HandleInfoKHR, VkExportMemoryWin32HandleInfoNV, VkExportMetalObjectCreateInfoEXT, VkImportAndroidHardwareBufferInfoANDROID, VkImportMemoryBufferCollectionFUCHSIA, VkImportMemoryFdInfoKHR, VkImportMemoryHostPointerInfoEXT, VkImportMemoryWin32HandleInfoKHR, VkImportMemoryWin32HandleInfoNV, VkImportMemoryZirconHandleInfoFUCHSIA, VkImportMetalBufferInfoEXT, VkImportScreenBufferInfoQNX, VkMemoryAllocateFlagsInfo, VkMemoryDedicatedAllocateInfo, VkMemoryOpaqueCaptureAddressAllocateInfo, or VkMemoryPriorityAllocateInfoEXT
- VUID-VkMemoryAllocateInfo-sType-unique
  The sType value of each struct in the pNext chain **must** be unique, with the exception of structures of type VkExportMetalObjectCreateInfoEXT

If the pNext chain includes a VkMemoryDedicatedAllocateInfo structure, then that structure includes a handle of the sole buffer or image resource that the memory **can** be bound to.

The VkMemoryDedicatedAllocateInfo structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkMemoryDedicatedAllocateInfo {
    VkStructureType    sType;
    const void*        pNext;
    VkImage            image;
    VkBuffer           buffer;
} VkMemoryDedicatedAllocateInfo;
```

or the equivalent

```
// Provided by VK_KHR_dedicated_allocation
typedef VkMemoryDedicatedAllocateInfo VkMemoryDedicatedAllocateInfoKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `image` is VK_NULL_HANDLE or a handle of an image which this memory will be bound to.
- `buffer` is VK_NULL_HANDLE or a handle of a buffer which this memory will be bound to.

## Valid Usage

- VUID-VkMemoryDedicatedAllocateInfo-image-01432
  At least one of `image` and `buffer` **must** be VK_NULL_HANDLE
- VUID-VkMemoryDedicatedAllocateInfo-image-02964
  If `image` is not VK_NULL_HANDLE and the memory is not an imported Android Hardware Buffer or an imported QNX Screen buffer , `VkMemoryAllocateInfo::allocationSize` **must** equal the `VkMemoryRequirements::size` of the image
- VUID-VkMemoryDedicatedAllocateInfo-image-01434
  If `image` is not VK_NULL_HANDLE, `image` **must** have been created without `VK_IMAGE_CREATE_SPARSE_BINDING_BIT` set in VkImageCreateInfo::`flags`
- VUID-VkMemoryDedicatedAllocateInfo-buffer-02965
  If `buffer` is not VK_NULL_HANDLE and the memory is not an imported Android Hardware Buffer or an imported QNX Screen buffer , `VkMemoryAllocateInfo::allocationSize` **must** equal the `VkMemoryRequirements::size` of the buffer
- VUID-VkMemoryDedicatedAllocateInfo-buffer-01436
  If `buffer` is not VK_NULL_HANDLE, `buffer` **must** have been created without `VK_BUFFER_CREATE_SPARSE_BINDING_BIT` set in VkBufferCreateInfo::`flags`
- VUID-VkMemoryDedicatedAllocateInfo-image-01876
  If `image` is not VK_NULL_HANDLE and VkMemoryAllocateInfo defines a memory import operation with handle type `VK_EXTERNAL_MEMORY_HANDLE_TYPE_OPAQUE_WIN32_BIT`, `VK_EXTERNAL_MEMORY_HANDLE_TYPE_OPAQUE_WIN32_KMT_BIT`, `VK_EXTERNAL_MEMORY_HANDLE_TYPE_D3D11_TEXTURE_BIT`, `VK_EXTERNAL_MEMORY_HANDLE_TYPE_D3D11_TEXTURE_KMT_BIT`, `VK_EXTERNAL_MEMORY_HANDLE_TYPE_D3D12_HEAP_BIT`, or

VK_EXTERNAL_MEMORY_HANDLE_TYPE_D3D12_RESOURCE_BIT, and the external handle was created by the Vulkan API, then the memory being imported **must** also be a dedicated image allocation and image **must** be identical to the image associated with the imported memory

- VUID-VkMemoryDedicatedAllocateInfo-buffer-01877
  If buffer is not VK_NULL_HANDLE and VkMemoryAllocateInfo defines a memory import operation with handle type VK_EXTERNAL_MEMORY_HANDLE_TYPE_OPAQUE_WIN32_BIT, VK_EXTERNAL_MEMORY_HANDLE_TYPE_OPAQUE_WIN32_KMT_BIT, VK_EXTERNAL_MEMORY_HANDLE_TYPE_D3D11_TEXTURE_BIT, VK_EXTERNAL_MEMORY_HANDLE_TYPE_D3D11_TEXTURE_KMT_BIT, VK_EXTERNAL_MEMORY_HANDLE_TYPE_D3D12_HEAP_BIT, or VK_EXTERNAL_MEMORY_HANDLE_TYPE_D3D12_RESOURCE_BIT, and the external handle was created by the Vulkan API, then the memory being imported **must** also be a dedicated buffer allocation and buffer **must** be identical to the buffer associated with the imported memory
- VUID-VkMemoryDedicatedAllocateInfo-image-01878
  If image is not VK_NULL_HANDLE and VkMemoryAllocateInfo defines a memory import operation with handle type VK_EXTERNAL_MEMORY_HANDLE_TYPE_OPAQUE_FD_BIT, the memory being imported **must** also be a dedicated image allocation and image **must** be identical to the image associated with the imported memory
- VUID-VkMemoryDedicatedAllocateInfo-buffer-01879
  If buffer is not VK_NULL_HANDLE and VkMemoryAllocateInfo defines a memory import operation with handle type VK_EXTERNAL_MEMORY_HANDLE_TYPE_OPAQUE_FD_BIT, the memory being imported **must** also be a dedicated buffer allocation and buffer **must** be identical to the buffer associated with the imported memory
- VUID-VkMemoryDedicatedAllocateInfo-image-01797
  If image is not VK_NULL_HANDLE, image **must** not have been created with VK_IMAGE_CREATE_DISJOINT_BIT set in VkImageCreateInfo::flags
- VUID-VkMemoryDedicatedAllocateInfo-image-04751
  If image is not VK_NULL_HANDLE and VkMemoryAllocateInfo defines a memory import operation with handle type VK_EXTERNAL_MEMORY_HANDLE_TYPE_ZIRCON_VMO_BIT_FUCHSIA, the memory being imported **must** also be a dedicated image allocation and image **must** be identical to the image associated with the imported memory
- VUID-VkMemoryDedicatedAllocateInfo-buffer-04752
  If buffer is not VK_NULL_HANDLE and VkMemoryAllocateInfo defines a memory import operation with handle type VK_EXTERNAL_MEMORY_HANDLE_TYPE_ZIRCON_VMO_BIT_FUCHSIA, the memory being imported **must** also be a dedicated buffer allocation and buffer **must** be identical to the buffer associated with the imported memory

## Valid Usage (Implicit)

- VUID-VkMemoryDedicatedAllocateInfo-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_MEMORY_DEDICATED_ALLOCATE_INFO
- VUID-VkMemoryDedicatedAllocateInfo-image-parameter
  If image is not VK_NULL_HANDLE, image **must** be a valid VkImage handle

- VUID-VkMemoryDedicatedAllocateInfo-buffer-parameter
  If `buffer` is not VK_NULL_HANDLE, `buffer` **must** be a valid VkBuffer handle
- VUID-VkMemoryDedicatedAllocateInfo-commonparent
  Both of `buffer`, and `image` that are valid handles of non-ignored parameters **must** have been created, allocated, or retrieved from the same VkDevice

If the `pNext` chain includes a `VkDedicatedAllocationMemoryAllocateInfoNV` structure, then that structure includes a handle of the sole buffer or image resource that the memory **can** be bound to.

The `VkDedicatedAllocationMemoryAllocateInfoNV` structure is defined as:

```
// Provided by VK_NV_dedicated_allocation
typedef struct VkDedicatedAllocationMemoryAllocateInfoNV {
    VkStructureType    sType;
    const void*        pNext;
    VkImage            image;
    VkBuffer           buffer;
} VkDedicatedAllocationMemoryAllocateInfoNV;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `image` is VK_NULL_HANDLE or a handle of an image which this memory will be bound to.
- `buffer` is VK_NULL_HANDLE or a handle of a buffer which this memory will be bound to.

### Valid Usage

- VUID-VkDedicatedAllocationMemoryAllocateInfoNV-image-00649
  At least one of `image` and `buffer` **must** be VK_NULL_HANDLE
- VUID-VkDedicatedAllocationMemoryAllocateInfoNV-image-00650
  If `image` is not VK_NULL_HANDLE, the image **must** have been created with VkDedicatedAllocationImageCreateInfoNV::`dedicatedAllocation` equal to `VK_TRUE`
- VUID-VkDedicatedAllocationMemoryAllocateInfoNV-buffer-00651
  If `buffer` is not VK_NULL_HANDLE, the buffer **must** have been created with VkDedicatedAllocationBufferCreateInfoNV::`dedicatedAllocation` equal to `VK_TRUE`
- VUID-VkDedicatedAllocationMemoryAllocateInfoNV-image-00652
  If `image` is not VK_NULL_HANDLE, `VkMemoryAllocateInfo`::`allocationSize` **must** equal the `VkMemoryRequirements`::`size` of the image
- VUID-VkDedicatedAllocationMemoryAllocateInfoNV-buffer-00653
  If `buffer` is not VK_NULL_HANDLE, `VkMemoryAllocateInfo`::`allocationSize` **must** equal the `VkMemoryRequirements`::`size` of the buffer
- VUID-VkDedicatedAllocationMemoryAllocateInfoNV-image-00654
  If `image` is not VK_NULL_HANDLE and VkMemoryAllocateInfo defines a memory import operation, the memory being imported **must** also be a dedicated image allocation and

image **must** be identical to the image associated with the imported memory

- VUID-VkDedicatedAllocationMemoryAllocateInfoNV-buffer-00655
  If buffer is not VK_NULL_HANDLE and VkMemoryAllocateInfo defines a memory import operation, the memory being imported **must** also be a dedicated buffer allocation and buffer **must** be identical to the buffer associated with the imported memory

### Valid Usage (Implicit)

- VUID-VkDedicatedAllocationMemoryAllocateInfoNV-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_DEDICATED_ALLOCATION_MEMORY_ALLOCATE_INFO_NV
- VUID-VkDedicatedAllocationMemoryAllocateInfoNV-image-parameter
  If image is not VK_NULL_HANDLE, image **must** be a valid VkImage handle
- VUID-VkDedicatedAllocationMemoryAllocateInfoNV-buffer-parameter
  If buffer is not VK_NULL_HANDLE, buffer **must** be a valid VkBuffer handle
- VUID-VkDedicatedAllocationMemoryAllocateInfoNV-commonparent
  Both of buffer, and image that are valid handles of non-ignored parameters **must** have been created, allocated, or retrieved from the same VkDevice

If the pNext chain includes a VkMemoryPriorityAllocateInfoEXT structure, then that structure includes a priority for the memory.

The VkMemoryPriorityAllocateInfoEXT structure is defined as:

```
// Provided by VK_EXT_memory_priority
typedef struct VkMemoryPriorityAllocateInfoEXT {
    VkStructureType    sType;
    const void*        pNext;
    float              priority;
} VkMemoryPriorityAllocateInfoEXT;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- priority is a floating-point value between 0 and 1, indicating the priority of the allocation relative to other memory allocations. Larger values are higher priority. The granularity of the priorities is implementation-dependent.

Memory allocations with higher priority **may** be more likely to stay in device-local memory when the system is under memory pressure.

If this structure is not included, it is as if the priority value were 0.5.

### Valid Usage

- VUID-VkMemoryPriorityAllocateInfoEXT-priority-02602

priority **must** be between 0 and 1, inclusive

### Valid Usage (Implicit)

- VUID-VkMemoryPriorityAllocateInfoEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_MEMORY_PRIORITY_ALLOCATE_INFO_EXT

To modify the priority of an existing memory allocation, call:

```
// Provided by VK_EXT_pageable_device_local_memory
void vkSetDeviceMemoryPriorityEXT(
    VkDevice                                    device,
    VkDeviceMemory                              memory,
    float                                       priority);
```

- device is the logical device that owns the memory.
- memory is the VkDeviceMemory object to which the new priority will be applied.
- priority is a floating-point value between 0 and 1, indicating the priority of the allocation relative to other memory allocations. Larger values are higher priority. The granularity of the priorities is implementation-dependent.

Memory allocations with higher priority **may** be more likely to stay in device-local memory when the system is under memory pressure.

### Valid Usage

- VUID-vkSetDeviceMemoryPriorityEXT-priority-06258
  priority **must** be between 0 and 1, inclusive

### Valid Usage (Implicit)

- VUID-vkSetDeviceMemoryPriorityEXT-device-parameter
  device **must** be a valid VkDevice handle
- VUID-vkSetDeviceMemoryPriorityEXT-memory-parameter
  memory **must** be a valid VkDeviceMemory handle
- VUID-vkSetDeviceMemoryPriorityEXT-memory-parent
  memory **must** have been created, allocated, or retrieved from device

When allocating memory whose payload **may** be exported to another process or Vulkan instance, add a VkExportMemoryAllocateInfo structure to the pNext chain of the VkMemoryAllocateInfo structure, specifying the handle types that **may** be exported.

The VkExportMemoryAllocateInfo structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkExportMemoryAllocateInfo {
    VkStructureType                     sType;
    const void*                         pNext;
    VkExternalMemoryHandleTypeFlags     handleTypes;
} VkExportMemoryAllocateInfo;
```

or the equivalent

```
// Provided by VK_KHR_external_memory
typedef VkExportMemoryAllocateInfo VkExportMemoryAllocateInfoKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `handleTypes` is zero or a bitmask of VkExternalMemoryHandleTypeFlagBits specifying one or more memory handle types the application **can** export from the resulting allocation. The application **can** request multiple handle types for the same allocation.

### Valid Usage

- VUID-VkExportMemoryAllocateInfo-handleTypes-00656
  The bits in `handleTypes` **must** be supported and compatible, as reported by VkExternalImageFormatProperties or VkExternalBufferProperties

### Valid Usage (Implicit)

- VUID-VkExportMemoryAllocateInfo-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_EXPORT_MEMORY_ALLOCATE_INFO`
- VUID-VkExportMemoryAllocateInfo-handleTypes-parameter
  `handleTypes` **must** be a valid combination of VkExternalMemoryHandleTypeFlagBits values

When allocating memory that **may** be exported to another process or Vulkan instance, add a VkExportMemoryAllocateInfoNV structure to the `pNext` chain of the VkMemoryAllocateInfo structure, specifying the handle types that **may** be exported.

The VkExportMemoryAllocateInfoNV structure is defined as:

```
// Provided by VK_NV_external_memory
typedef struct VkExportMemoryAllocateInfoNV {
    VkStructureType                     sType;
    const void*                         pNext;
    VkExternalMemoryHandleTypeFlagsNV   handleTypes;
} VkExportMemoryAllocateInfoNV;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `handleTypes` is a bitmask of VkExternalMemoryHandleTypeFlagBitsNV specifying one or more memory handle types that **may** be exported. Multiple handle types **may** be requested for the same allocation as long as they are compatible, as reported by vkGetPhysicalDeviceExternalImageFormatPropertiesNV.

**Valid Usage (Implicit)**

- VUID-VkExportMemoryAllocateInfoNV-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_EXPORT_MEMORY_ALLOCATE_INFO_NV`
- VUID-VkExportMemoryAllocateInfoNV-handleTypes-parameter
  `handleTypes` **must** be a valid combination of VkExternalMemoryHandleTypeFlagBitsNV values

### 11.2.4. Win32 External Memory

To specify additional attributes of NT handles exported from a memory object, add a VkExportMemoryWin32HandleInfoKHR structure to the `pNext` chain of the VkMemoryAllocateInfo structure. The `VkExportMemoryWin32HandleInfoKHR` structure is defined as:

```
// Provided by VK_KHR_external_memory_win32
typedef struct VkExportMemoryWin32HandleInfoKHR {
    VkStructureType               sType;
    const void*                   pNext;
    const SECURITY_ATTRIBUTES*    pAttributes;
    DWORD                         dwAccess;
    LPCWSTR                       name;
} VkExportMemoryWin32HandleInfoKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `pAttributes` is a pointer to a Windows `SECURITY_ATTRIBUTES` structure specifying security attributes of the handle.
- `dwAccess` is a `DWORD` specifying access rights of the handle.
- `name` is a null-terminated UTF-16 string to associate with the payload referenced by NT handles

exported from the created memory.

If VkExportMemoryAllocateInfo is not included in the same pNext chain, this structure is ignored.

If VkExportMemoryAllocateInfo is included in the pNext chain of VkMemoryAllocateInfo with a Windows handleType, but either VkExportMemoryWin32HandleInfoKHR is not included in the pNext chain, or it is included but pAttributes is set to NULL, default security descriptor values will be used, and child processes created by the application will not inherit the handle, as described in the MSDN documentation for "Synchronization Object Security and Access Rights"[1]. Further, if the structure is not present, the access rights used depend on the handle type.

For handles of the following types:

- VK_EXTERNAL_MEMORY_HANDLE_TYPE_OPAQUE_WIN32_BIT

The implementation **must** ensure the access rights allow read and write access to the memory.

**1**

https://docs.microsoft.com/en-us/windows/win32/sync/synchronization-object-security-and-access-rights

### Valid Usage

- VUID-VkExportMemoryWin32HandleInfoKHR-handleTypes-00657
  If VkExportMemoryAllocateInfo::handleTypes does not include VK_EXTERNAL_MEMORY_HANDLE_TYPE_OPAQUE_WIN32_BIT, a VkExportMemoryWin32HandleInfoKHR structure **must** not be included in the pNext chain of VkMemoryAllocateInfo

### Valid Usage (Implicit)

- VUID-VkExportMemoryWin32HandleInfoKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_EXPORT_MEMORY_WIN32_HANDLE_INFO_KHR
- VUID-VkExportMemoryWin32HandleInfoKHR-pAttributes-parameter
  If pAttributes is not NULL, pAttributes **must** be a valid pointer to a valid SECURITY_ATTRIBUTES value

To import memory from a Windows handle, add a VkImportMemoryWin32HandleInfoKHR structure to the pNext chain of the VkMemoryAllocateInfo structure.

The VkImportMemoryWin32HandleInfoKHR structure is defined as:

```
// Provided by VK_KHR_external_memory_win32
typedef struct VkImportMemoryWin32HandleInfoKHR {
    VkStructureType                       sType;
    const void*                           pNext;
    VkExternalMemoryHandleTypeFlagBits    handleType;
    HANDLE                                handle;
    LPCWSTR                               name;
} VkImportMemoryWin32HandleInfoKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `handleType` is a VkExternalMemoryHandleTypeFlagBits value specifying the type of `handle` or `name`.
- `handle` is `NULL` or the external handle to import.
- `name` is `NULL` or a null-terminated UTF-16 string naming the payload to import.

Importing memory object payloads from Windows handles does not transfer ownership of the handle to the Vulkan implementation. For handle types defined as NT handles, the application **must** release handle ownership using the `CloseHandle` system call when the handle is no longer needed. For handle types defined as NT handles, the imported memory object holds a reference to its payload.

| | |
|---|---|
| **NOTE** | Non-NT handle import operations do not add a reference to their associated payload. If the original object owning the payload is destroyed, all resources and handles sharing that payload will become invalid. |

Applications **can** import the same payload into multiple instances of Vulkan, into the same instance from which it was exported, and multiple times into a given Vulkan instance. In all cases, each import operation **must** create a distinct `VkDeviceMemory` object.

### Valid Usage

- VUID-VkImportMemoryWin32HandleInfoKHR-handleType-00658
  If `handleType` is not `0`, it **must** be supported for import, as reported by VkExternalImageFormatProperties or VkExternalBufferProperties
- VUID-VkImportMemoryWin32HandleInfoKHR-handle-00659
  The memory from which `handle` was exported, or the memory named by `name` **must** have been created on the same underlying physical device as `device`
- VUID-VkImportMemoryWin32HandleInfoKHR-handleType-00660
  If `handleType` is not `0`, it **must** be defined as an NT handle or a global share handle
- VUID-VkImportMemoryWin32HandleInfoKHR-handleType-01439
  If `handleType` is not `VK_EXTERNAL_MEMORY_HANDLE_TYPE_OPAQUE_WIN32_BIT`, `VK_EXTERNAL_MEMORY_HANDLE_TYPE_D3D11_TEXTURE_BIT`, `VK_EXTERNAL_MEMORY_HANDLE_TYPE_D3D12_HEAP_BIT`, or

VK_EXTERNAL_MEMORY_HANDLE_TYPE_D3D12_RESOURCE_BIT, name **must** be NULL

- VUID-VkImportMemoryWin32HandleInfoKHR-handleType-01440
  If handleType is not 0 and handle is NULL, name **must** name a valid memory resource of the type specified by handleType
- VUID-VkImportMemoryWin32HandleInfoKHR-handleType-00661
  If handleType is not 0 and name is NULL, handle **must** be a valid handle of the type specified by handleType
- VUID-VkImportMemoryWin32HandleInfoKHR-handle-01441
  If handle is not NULL, name **must** be NULL
- VUID-VkImportMemoryWin32HandleInfoKHR-handle-01518
  If handle is not NULL, it **must** obey any requirements listed for handleType in external memory handle types compatibility
- VUID-VkImportMemoryWin32HandleInfoKHR-name-01519
  If name is not NULL, it **must** obey any requirements listed for handleType in external memory handle types compatibility

### Valid Usage (Implicit)

- VUID-VkImportMemoryWin32HandleInfoKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_IMPORT_MEMORY_WIN32_HANDLE_INFO_KHR
- VUID-VkImportMemoryWin32HandleInfoKHR-handleType-parameter
  If handleType is not 0, handleType **must** be a valid VkExternalMemoryHandleTypeFlagBits value

To export a Windows handle representing the payload of a Vulkan device memory object, call:

```
// Provided by VK_KHR_external_memory_win32
VkResult vkGetMemoryWin32HandleKHR(
    VkDevice                                    device,
    const VkMemoryGetWin32HandleInfoKHR*        pGetWin32HandleInfo,
    HANDLE*                                     pHandle);
```

- device is the logical device that created the device memory being exported.
- pGetWin32HandleInfo is a pointer to a VkMemoryGetWin32HandleInfoKHR structure containing parameters of the export operation.
- pHandle will return the Windows handle representing the payload of the device memory object.

For handle types defined as NT handles, the handles returned by vkGetMemoryWin32HandleKHR are owned by the application and hold a reference to their payload. To avoid leaking resources, the application **must** release ownership of them using the CloseHandle system call when they are no longer needed.

**NOTE** Non-NT handle types do not add a reference to their associated payload. If the

> original object owning the payload is destroyed, all resources and handles sharing that payload will become invalid.

### Valid Usage (Implicit)

- VUID-vkGetMemoryWin32HandleKHR-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkGetMemoryWin32HandleKHR-pGetWin32HandleInfo-parameter
  `pGetWin32HandleInfo` **must** be a valid pointer to a valid VkMemoryGetWin32HandleInfoKHR structure
- VUID-vkGetMemoryWin32HandleKHR-pHandle-parameter
  `pHandle` **must** be a valid pointer to a `HANDLE` value

### Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- `VK_ERROR_TOO_MANY_OBJECTS`
- `VK_ERROR_OUT_OF_HOST_MEMORY`

The `VkMemoryGetWin32HandleInfoKHR` structure is defined as:

```
// Provided by VK_KHR_external_memory_win32
typedef struct VkMemoryGetWin32HandleInfoKHR {
    VkStructureType                       sType;
    const void*                           pNext;
    VkDeviceMemory                        memory;
    VkExternalMemoryHandleTypeFlagBits    handleType;
} VkMemoryGetWin32HandleInfoKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `memory` is the memory object from which the handle will be exported.
- `handleType` is a VkExternalMemoryHandleTypeFlagBits value specifying the type of handle requested.

The properties of the handle returned depend on the value of `handleType`. See VkExternalMemoryHandleTypeFlagBits for a description of the properties of the defined external memory handle types.

### Valid Usage

- VUID-VkMemoryGetWin32HandleInfoKHR-handleType-00662
  `handleType` **must** have been included in VkExportMemoryAllocateInfo::`handleTypes` when `memory` was created
- VUID-VkMemoryGetWin32HandleInfoKHR-handleType-00663
  If `handleType` is defined as an NT handle, vkGetMemoryWin32HandleKHR **must** be called no more than once for each valid unique combination of `memory` and `handleType`
- VUID-VkMemoryGetWin32HandleInfoKHR-handleType-00664
  `handleType` **must** be defined as an NT handle or a global share handle

### Valid Usage (Implicit)

- VUID-VkMemoryGetWin32HandleInfoKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_MEMORY_GET_WIN32_HANDLE_INFO_KHR`
- VUID-VkMemoryGetWin32HandleInfoKHR-pNext-pNext
  `pNext` **must** be `NULL`
- VUID-VkMemoryGetWin32HandleInfoKHR-memory-parameter
  `memory` **must** be a valid VkDeviceMemory handle
- VUID-VkMemoryGetWin32HandleInfoKHR-handleType-parameter
  `handleType` **must** be a valid VkExternalMemoryHandleTypeFlagBits value

Windows memory handles compatible with Vulkan **may** also be created by non-Vulkan APIs using methods beyond the scope of this specification. To determine the correct parameters to use when importing such handles, call:

```
// Provided by VK_KHR_external_memory_win32
VkResult vkGetMemoryWin32HandlePropertiesKHR(
    VkDevice                                    device,
    VkExternalMemoryHandleTypeFlagBits          handleType,
    HANDLE                                      handle,
    VkMemoryWin32HandlePropertiesKHR*           pMemoryWin32HandleProperties);
```

- `device` is the logical device that will be importing `handle`.
- `handleType` is a VkExternalMemoryHandleTypeFlagBits value specifying the type of the handle `handle`.
- `handle` is the handle which will be imported.
- `pMemoryWin32HandleProperties` is a pointer to a VkMemoryWin32HandlePropertiesKHR structure in which properties of `handle` are returned.

### Valid Usage

- VUID-vkGetMemoryWin32HandlePropertiesKHR-handle-00665
  `handle` **must** point to a valid Windows memory handle
- VUID-vkGetMemoryWin32HandlePropertiesKHR-handleType-00666
  `handleType` **must** not be one of the handle types defined as opaque

### Valid Usage (Implicit)

- VUID-vkGetMemoryWin32HandlePropertiesKHR-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkGetMemoryWin32HandlePropertiesKHR-handleType-parameter
  `handleType` **must** be a valid VkExternalMemoryHandleTypeFlagBits value
- VUID-vkGetMemoryWin32HandlePropertiesKHR-pMemoryWin32HandleProperties-parameter
  `pMemoryWin32HandleProperties` **must** be a valid pointer to a VkMemoryWin32HandlePropertiesKHR structure

### Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_INVALID_EXTERNAL_HANDLE`

The `VkMemoryWin32HandlePropertiesKHR` structure returned is defined as:

```
// Provided by VK_KHR_external_memory_win32
typedef struct VkMemoryWin32HandlePropertiesKHR {
    VkStructureType    sType;
    void*              pNext;
    uint32_t           memoryTypeBits;
} VkMemoryWin32HandlePropertiesKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `memoryTypeBits` is a bitmask containing one bit set for every memory type which the specified windows handle **can** be imported as.

### Valid Usage (Implicit)

- VUID-VkMemoryWin32HandlePropertiesKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_MEMORY_WIN32_HANDLE_PROPERTIES_KHR`
- VUID-VkMemoryWin32HandlePropertiesKHR-pNext-pNext
  `pNext` **must** be `NULL`

When VkExportMemoryAllocateInfoNV::`handleTypes` includes `VK_EXTERNAL_MEMORY_HANDLE_TYPE_OPAQUE_WIN32_BIT_NV`, add a `VkExportMemoryWin32HandleInfoNV` structure to the `pNext` chain of the VkExportMemoryAllocateInfoNV structure to specify security attributes and access rights for the memory object's external handle.

The `VkExportMemoryWin32HandleInfoNV` structure is defined as:

```
// Provided by VK_NV_external_memory_win32
typedef struct VkExportMemoryWin32HandleInfoNV {
    VkStructureType                 sType;
    const void*                     pNext;
    const SECURITY_ATTRIBUTES*      pAttributes;
    DWORD                           dwAccess;
} VkExportMemoryWin32HandleInfoNV;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `pAttributes` is a pointer to a Windows `SECURITY_ATTRIBUTES` structure specifying security attributes of the handle.
- `dwAccess` is a `DWORD` specifying access rights of the handle.

If this structure is not present, or if `pAttributes` is set to `NULL`, default security descriptor values will be used, and child processes created by the application will not inherit the handle, as described in the MSDN documentation for "Synchronization Object Security and Access Rights"[1]. Further, if the structure is not present, the access rights will be

`DXGI_SHARED_RESOURCE_READ` | `DXGI_SHARED_RESOURCE_WRITE`

**1**

https://docs.microsoft.com/en-us/windows/win32/sync/synchronization-object-security-and-access-rights

### Valid Usage (Implicit)

- VUID-VkExportMemoryWin32HandleInfoNV-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_EXPORT_MEMORY_WIN32_HANDLE_INFO_NV`
- VUID-VkExportMemoryWin32HandleInfoNV-pAttributes-parameter
  If `pAttributes` is not `NULL`, `pAttributes` **must** be a valid pointer to a valid

```
    SECURITY_ATTRIBUTES value
```

To import memory created on the same physical device but outside of the current Vulkan instance, add a VkImportMemoryWin32HandleInfoNV structure to the pNext chain of the VkMemoryAllocateInfo structure, specifying a handle to and the type of the memory.

The VkImportMemoryWin32HandleInfoNV structure is defined as:

```
// Provided by VK_NV_external_memory_win32
typedef struct VkImportMemoryWin32HandleInfoNV {
    VkStructureType                     sType;
    const void*                         pNext;
    VkExternalMemoryHandleTypeFlagsNV   handleType;
    HANDLE                              handle;
} VkImportMemoryWin32HandleInfoNV;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- handleType is 0 or a VkExternalMemoryHandleTypeFlagBitsNV value specifying the type of memory handle in handle.
- handle is a Windows HANDLE referring to the memory.

If handleType is 0, this structure is ignored by consumers of the VkMemoryAllocateInfo structure it is chained from.

### Valid Usage

- VUID-VkImportMemoryWin32HandleInfoNV-handleType-01327
  handleType **must** not have more than one bit set
- VUID-VkImportMemoryWin32HandleInfoNV-handle-01328
  handle **must** be a valid handle to memory, obtained as specified by handleType

### Valid Usage (Implicit)

- VUID-VkImportMemoryWin32HandleInfoNV-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_IMPORT_MEMORY_WIN32_HANDLE_INFO_NV
- VUID-VkImportMemoryWin32HandleInfoNV-handleType-parameter
  handleType **must** be a valid combination of VkExternalMemoryHandleTypeFlagBitsNV values

Bits which **can** be set in handleType are:

Possible values of VkImportMemoryWin32HandleInfoNV::handleType, specifying the type of an

external memory handle, are:

```
// Provided by VK_NV_external_memory_capabilities
typedef enum VkExternalMemoryHandleTypeFlagBitsNV {
    VK_EXTERNAL_MEMORY_HANDLE_TYPE_OPAQUE_WIN32_BIT_NV = 0x00000001,
    VK_EXTERNAL_MEMORY_HANDLE_TYPE_OPAQUE_WIN32_KMT_BIT_NV = 0x00000002,
    VK_EXTERNAL_MEMORY_HANDLE_TYPE_D3D11_IMAGE_BIT_NV = 0x00000004,
    VK_EXTERNAL_MEMORY_HANDLE_TYPE_D3D11_IMAGE_KMT_BIT_NV = 0x00000008,
} VkExternalMemoryHandleTypeFlagBitsNV;
```

- `VK_EXTERNAL_MEMORY_HANDLE_TYPE_OPAQUE_WIN32_KMT_BIT_NV` specifies a handle to memory returned by vkGetMemoryWin32HandleNV.
- `VK_EXTERNAL_MEMORY_HANDLE_TYPE_OPAQUE_WIN32_BIT_NV` specifies a handle to memory returned by vkGetMemoryWin32HandleNV, or one duplicated from such a handle using `DuplicateHandle()`.
- `VK_EXTERNAL_MEMORY_HANDLE_TYPE_D3D11_IMAGE_BIT_NV` specifies a valid NT handle to memory returned by `IDXGIResource1::CreateSharedHandle`, or a handle duplicated from such a handle using `DuplicateHandle()`.
- `VK_EXTERNAL_MEMORY_HANDLE_TYPE_D3D11_IMAGE_KMT_BIT_NV` specifies a handle to memory returned by `IDXGIResource::GetSharedHandle()`.

```
// Provided by VK_NV_external_memory_capabilities
typedef VkFlags VkExternalMemoryHandleTypeFlagsNV;
```

`VkExternalMemoryHandleTypeFlagsNV` is a bitmask type for setting a mask of zero or more VkExternalMemoryHandleTypeFlagBitsNV.

To retrieve the handle corresponding to a device memory object created with VkExportMemoryAllocateInfoNV::`handleTypes` set to include `VK_EXTERNAL_MEMORY_HANDLE_TYPE_OPAQUE_WIN32_BIT_NV` or `VK_EXTERNAL_MEMORY_HANDLE_TYPE_OPAQUE_WIN32_KMT_BIT_NV`, call:

```
// Provided by VK_NV_external_memory_win32
VkResult vkGetMemoryWin32HandleNV(
    VkDevice                                    device,
    VkDeviceMemory                              memory,
    VkExternalMemoryHandleTypeFlagsNV           handleType,
    HANDLE*                                     pHandle);
```

- `device` is the logical device that owns the memory.
- `memory` is the VkDeviceMemory object.
- `handleType` is a bitmask of VkExternalMemoryHandleTypeFlagBitsNV containing a single bit specifying the type of handle requested.
- `handle` is a pointer to a Windows `HANDLE` in which the handle is returned.

### Valid Usage

- VUID-vkGetMemoryWin32HandleNV-handleType-01326
  `handleType` **must** be a flag specified in VkExportMemoryAllocateInfoNV::`handleTypes` when allocating `memory`

### Valid Usage (Implicit)

- VUID-vkGetMemoryWin32HandleNV-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkGetMemoryWin32HandleNV-memory-parameter
  `memory` **must** be a valid VkDeviceMemory handle
- VUID-vkGetMemoryWin32HandleNV-handleType-parameter
  `handleType` **must** be a valid combination of VkExternalMemoryHandleTypeFlagBitsNV values
- VUID-vkGetMemoryWin32HandleNV-handleType-requiredbitmask
  `handleType` **must** not be `0`
- VUID-vkGetMemoryWin32HandleNV-pHandle-parameter
  `pHandle` **must** be a valid pointer to a `HANDLE` value
- VUID-vkGetMemoryWin32HandleNV-memory-parent
  `memory` **must** have been created, allocated, or retrieved from `device`

### Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- `VK_ERROR_TOO_MANY_OBJECTS`
- `VK_ERROR_OUT_OF_HOST_MEMORY`

## 11.2.5. File Descriptor External Memory

To import memory from a POSIX file descriptor handle, add a VkImportMemoryFdInfoKHR structure to the `pNext` chain of the VkMemoryAllocateInfo structure. The `VkImportMemoryFdInfoKHR` structure is defined as:

```
// Provided by VK_KHR_external_memory_fd
typedef struct VkImportMemoryFdInfoKHR {
    VkStructureType                       sType;
    const void*                           pNext;
    VkExternalMemoryHandleTypeFlagBits    handleType;
    int                                   fd;
} VkImportMemoryFdInfoKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `handleType` is a VkExternalMemoryHandleTypeFlagBits value specifying the handle type of `fd`.
- `fd` is the external handle to import.

Importing memory from a file descriptor transfers ownership of the file descriptor from the application to the Vulkan implementation. The application **must** not perform any operations on the file descriptor after a successful import. The imported memory object holds a reference to its payload.

Applications **can** import the same payload into multiple instances of Vulkan, into the same instance from which it was exported, and multiple times into a given Vulkan instance. In all cases, each import operation **must** create a distinct `VkDeviceMemory` object.

### Valid Usage

- VUID-VkImportMemoryFdInfoKHR-handleType-00667
  If `handleType` is not `0`, it **must** be supported for import, as reported by VkExternalImageFormatProperties or VkExternalBufferProperties
- VUID-VkImportMemoryFdInfoKHR-fd-00668
  The memory from which `fd` was exported **must** have been created on the same underlying physical device as `device`
- VUID-VkImportMemoryFdInfoKHR-handleType-00669
  If `handleType` is not `0`, it **must** be `VK_EXTERNAL_MEMORY_HANDLE_TYPE_OPAQUE_FD_BIT` or `VK_EXTERNAL_MEMORY_HANDLE_TYPE_DMA_BUF_BIT_EXT`
- VUID-VkImportMemoryFdInfoKHR-handleType-00670
  If `handleType` is not `0`, `fd` **must** be a valid handle of the type specified by `handleType`
- VUID-VkImportMemoryFdInfoKHR-fd-01746
  The memory represented by `fd` **must** have been created from a physical device and driver that is compatible with `device` and `handleType`, as described in External memory handle types compatibility
- VUID-VkImportMemoryFdInfoKHR-fd-01520
  `fd` **must** obey any requirements listed for `handleType` in external memory handle types compatibility

### Valid Usage (Implicit)

- VUID-VkImportMemoryFdInfoKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_IMPORT_MEMORY_FD_INFO_KHR`
- VUID-VkImportMemoryFdInfoKHR-handleType-parameter
  If `handleType` is not `0`, `handleType` **must** be a valid VkExternalMemoryHandleTypeFlagBits value

To export a POSIX file descriptor referencing the payload of a Vulkan device memory object, call:

```
// Provided by VK_KHR_external_memory_fd
VkResult vkGetMemoryFdKHR(
    VkDevice                                    device,
    const VkMemoryGetFdInfoKHR*                 pGetFdInfo,
    int*                                        pFd);
```

- `device` is the logical device that created the device memory being exported.
- `pGetFdInfo` is a pointer to a VkMemoryGetFdInfoKHR structure containing parameters of the export operation.
- `pFd` will return a file descriptor referencing the payload of the device memory object.

Each call to `vkGetMemoryFdKHR` **must** create a new file descriptor holding a reference to the memory object's payload and transfer ownership of the file descriptor to the application. To avoid leaking resources, the application **must** release ownership of the file descriptor using the `close` system call when it is no longer needed, or by importing a Vulkan memory object from it. Where supported by the operating system, the implementation **must** set the file descriptor to be closed automatically when an `execve` system call is made.

### Valid Usage (Implicit)

- VUID-vkGetMemoryFdKHR-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkGetMemoryFdKHR-pGetFdInfo-parameter
  `pGetFdInfo` **must** be a valid pointer to a valid VkMemoryGetFdInfoKHR structure
- VUID-vkGetMemoryFdKHR-pFd-parameter
  `pFd` **must** be a valid pointer to an `int` value

### Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- VK_ERROR_TOO_MANY_OBJECTS
- VK_ERROR_OUT_OF_HOST_MEMORY

The VkMemoryGetFdInfoKHR structure is defined as:

```
// Provided by VK_KHR_external_memory_fd
typedef struct VkMemoryGetFdInfoKHR {
    VkStructureType                       sType;
    const void*                           pNext;
    VkDeviceMemory                        memory;
    VkExternalMemoryHandleTypeFlagBits    handleType;
} VkMemoryGetFdInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- memory is the memory object from which the handle will be exported.
- handleType is a VkExternalMemoryHandleTypeFlagBits value specifying the type of handle requested.

The properties of the file descriptor exported depend on the value of handleType. See VkExternalMemoryHandleTypeFlagBits for a description of the properties of the defined external memory handle types.

**NOTE** The size of the exported file **may** be larger than the size requested by VkMemoryAllocateInfo::allocationSize. If handleType is VK_EXTERNAL_MEMORY_HANDLE_TYPE_DMA_BUF_BIT_EXT, then the application **can** query the file's actual size with lseek.

## Valid Usage

- VUID-VkMemoryGetFdInfoKHR-handleType-00671
  handleType **must** have been included in VkExportMemoryAllocateInfo::handleTypes when memory was created
- VUID-VkMemoryGetFdInfoKHR-handleType-00672
  handleType **must** be VK_EXTERNAL_MEMORY_HANDLE_TYPE_OPAQUE_FD_BIT or VK_EXTERNAL_MEMORY_HANDLE_TYPE_DMA_BUF_BIT_EXT

## Valid Usage (Implicit)

- VUID-VkMemoryGetFdInfoKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_MEMORY_GET_FD_INFO_KHR
- VUID-VkMemoryGetFdInfoKHR-pNext-pNext

pNext **must** be NULL

- VUID-VkMemoryGetFdInfoKHR-memory-parameter
  memory **must** be a valid VkDeviceMemory handle
- VUID-VkMemoryGetFdInfoKHR-handleType-parameter
  handleType **must** be a valid VkExternalMemoryHandleTypeFlagBits value

POSIX file descriptor memory handles compatible with Vulkan **may** also be created by non-Vulkan APIs using methods beyond the scope of this specification. To determine the correct parameters to use when importing such handles, call:

```
// Provided by VK_KHR_external_memory_fd
VkResult vkGetMemoryFdPropertiesKHR(
    VkDevice                                    device,
    VkExternalMemoryHandleTypeFlagBits          handleType,
    int                                         fd,
    VkMemoryFdPropertiesKHR*                    pMemoryFdProperties);
```

- device is the logical device that will be importing fd.
- handleType is a VkExternalMemoryHandleTypeFlagBits value specifying the type of the handle fd.
- fd is the handle which will be imported.
- pMemoryFdProperties is a pointer to a VkMemoryFdPropertiesKHR structure in which the properties of the handle fd are returned.

## Valid Usage

- VUID-vkGetMemoryFdPropertiesKHR-fd-00673
  fd **must** point to a valid POSIX file descriptor memory handle
- VUID-vkGetMemoryFdPropertiesKHR-handleType-00674
  handleType **must** not be VK_EXTERNAL_MEMORY_HANDLE_TYPE_OPAQUE_FD_BIT

## Valid Usage (Implicit)

- VUID-vkGetMemoryFdPropertiesKHR-device-parameter
  device **must** be a valid VkDevice handle
- VUID-vkGetMemoryFdPropertiesKHR-handleType-parameter
  handleType **must** be a valid VkExternalMemoryHandleTypeFlagBits value
- VUID-vkGetMemoryFdPropertiesKHR-pMemoryFdProperties-parameter
  pMemoryFdProperties **must** be a valid pointer to a VkMemoryFdPropertiesKHR structure

### Return Codes

**Success**

- VK_SUCCESS

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_INVALID_EXTERNAL_HANDLE

The VkMemoryFdPropertiesKHR structure returned is defined as:

```
// Provided by VK_KHR_external_memory_fd
typedef struct VkMemoryFdPropertiesKHR {
    VkStructureType    sType;
    void*              pNext;
    uint32_t           memoryTypeBits;
} VkMemoryFdPropertiesKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- memoryTypeBits is a bitmask containing one bit set for every memory type which the specified file descriptor **can** be imported as.

### Valid Usage (Implicit)

- VUID-VkMemoryFdPropertiesKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_MEMORY_FD_PROPERTIES_KHR
- VUID-VkMemoryFdPropertiesKHR-pNext-pNext
  pNext **must** be NULL

## 11.2.6. Host External Memory

To import memory from a host pointer, add a VkImportMemoryHostPointerInfoEXT structure to the pNext chain of the VkMemoryAllocateInfo structure. The VkImportMemoryHostPointerInfoEXT structure is defined as:

```
// Provided by VK_EXT_external_memory_host
typedef struct VkImportMemoryHostPointerInfoEXT {
    VkStructureType                       sType;
    const void*                           pNext;
    VkExternalMemoryHandleTypeFlagBits    handleType;
    void*                                 pHostPointer;
} VkImportMemoryHostPointerInfoEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `handleType` is a VkExternalMemoryHandleTypeFlagBits value specifying the handle type.
- `pHostPointer` is the host pointer to import from.

Importing memory from a host pointer shares ownership of the memory between the host and the Vulkan implementation. The application **can** continue to access the memory through the host pointer but it is the application's responsibility to synchronize device and non-device access to the payload as defined in Host Access to Device Memory Objects.

Applications **can** import the same payload into multiple instances of Vulkan and multiple times into a given Vulkan instance. However, implementations **may** fail to import the same payload multiple times into a given physical device due to platform constraints.

Importing memory from a particular host pointer **may** not be possible due to additional platform-specific restrictions beyond the scope of this specification in which case the implementation **must** fail the memory import operation with the error code `VK_ERROR_INVALID_EXTERNAL_HANDLE_KHR`.

Whether device memory objects imported from a host pointer hold a reference to their payload is undefined. As such, the application **must** ensure that the imported memory range remains valid and accessible for the lifetime of the imported memory object.

Implementations **may** support importing host pointers for memory types which are not host-visible. In this case, after a successful call to vkAllocateMemory, the memory range imported from `pHostPointer` **must** not be accessed by the application until the VkDeviceMemory has been destroyed. Memory contents for the host memory becomes undefined on import, and is left undefined after the VkDeviceMemory has been destroyed. Applications **must** also not access host memory which is mapped to the same physical memory as `pHostPointer`, but mapped to a different host pointer while the VkDeviceMemory handle is valid. Implementations running on general-purpose operating systems **should** not support importing host pointers for memory types which are not host-visible.

| | |
|---|---|
| **NOTE** | Using host pointers to back non-host visible allocations is a platform-specific use case, and applications should not attempt to do this unless instructed by the platform. |

### Valid Usage

- VUID-VkImportMemoryHostPointerInfoEXT-handleType-01747
  If `handleType` is not `0`, it **must** be supported for import, as reported in VkExternalMemoryProperties
- VUID-VkImportMemoryHostPointerInfoEXT-handleType-01748
  If `handleType` is not `0`, it **must** be `VK_EXTERNAL_MEMORY_HANDLE_TYPE_HOST_ALLOCATION_BIT_EXT` or `VK_EXTERNAL_MEMORY_HANDLE_TYPE_HOST_MAPPED_FOREIGN_MEMORY_BIT_EXT`
- VUID-VkImportMemoryHostPointerInfoEXT-pHostPointer-01749
  `pHostPointer` **must** be a pointer aligned to an integer multiple of

VkPhysicalDeviceExternalMemoryHostPropertiesEXT::minImportedHostPointerAlignment

- VUID-VkImportMemoryHostPointerInfoEXT-handleType-01750
  If handleType is VK_EXTERNAL_MEMORY_HANDLE_TYPE_HOST_ALLOCATION_BIT_EXT, pHostPointer **must** be a pointer to allocationSize number of bytes of host memory, where allocationSize is the member of the VkMemoryAllocateInfo structure this structure is chained to
- VUID-VkImportMemoryHostPointerInfoEXT-handleType-01751
  If handleType is VK_EXTERNAL_MEMORY_HANDLE_TYPE_HOST_MAPPED_FOREIGN_MEMORY_BIT_EXT, pHostPointer **must** be a pointer to allocationSize number of bytes of host mapped foreign memory, where allocationSize is the member of the VkMemoryAllocateInfo structure this structure is chained to

### Valid Usage (Implicit)

- VUID-VkImportMemoryHostPointerInfoEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_IMPORT_MEMORY_HOST_POINTER_INFO_EXT
- VUID-VkImportMemoryHostPointerInfoEXT-handleType-parameter
  handleType **must** be a valid VkExternalMemoryHandleTypeFlagBits value
- VUID-VkImportMemoryHostPointerInfoEXT-pHostPointer-parameter
  pHostPointer **must** be a pointer value

To determine the correct parameters to use when importing host pointers, call:

```
// Provided by VK_EXT_external_memory_host
VkResult vkGetMemoryHostPointerPropertiesEXT(
    VkDevice                                    device,
    VkExternalMemoryHandleTypeFlagBits          handleType,
    const void*                                 pHostPointer,
    VkMemoryHostPointerPropertiesEXT*           pMemoryHostPointerProperties);
```

- device is the logical device that will be importing pHostPointer.
- handleType is a VkExternalMemoryHandleTypeFlagBits value specifying the type of the handle pHostPointer.
- pHostPointer is the host pointer to import from.
- pMemoryHostPointerProperties is a pointer to a VkMemoryHostPointerPropertiesEXT structure in which the host pointer properties are returned.

### Valid Usage

- VUID-vkGetMemoryHostPointerPropertiesEXT-handleType-01752
  handleType **must** be VK_EXTERNAL_MEMORY_HANDLE_TYPE_HOST_ALLOCATION_BIT_EXT or VK_EXTERNAL_MEMORY_HANDLE_TYPE_HOST_MAPPED_FOREIGN_MEMORY_BIT_EXT

- VUID-vkGetMemoryHostPointerPropertiesEXT-pHostPointer-01753
  `pHostPointer` **must** be a pointer aligned to an integer multiple of `VkPhysicalDeviceExternalMemoryHostPropertiesEXT::minImportedHostPointerAlignment`
- VUID-vkGetMemoryHostPointerPropertiesEXT-handleType-01754
  If `handleType` is `VK_EXTERNAL_MEMORY_HANDLE_TYPE_HOST_ALLOCATION_BIT_EXT`, `pHostPointer` **must** be a pointer to host memory
- VUID-vkGetMemoryHostPointerPropertiesEXT-handleType-01755
  If `handleType` is `VK_EXTERNAL_MEMORY_HANDLE_TYPE_HOST_MAPPED_FOREIGN_MEMORY_BIT_EXT`, `pHostPointer` **must** be a pointer to host mapped foreign memory

## Valid Usage (Implicit)

- VUID-vkGetMemoryHostPointerPropertiesEXT-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkGetMemoryHostPointerPropertiesEXT-handleType-parameter
  `handleType` **must** be a valid VkExternalMemoryHandleTypeFlagBits value
- VUID-vkGetMemoryHostPointerPropertiesEXT-pHostPointer-parameter
  `pHostPointer` **must** be a pointer value
- VUID-vkGetMemoryHostPointerPropertiesEXT-pMemoryHostPointerProperties-parameter
  `pMemoryHostPointerProperties` **must** be a valid pointer to a VkMemoryHostPointerPropertiesEXT structure

## Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_INVALID_EXTERNAL_HANDLE`

The `VkMemoryHostPointerPropertiesEXT` structure is defined as:

```
// Provided by VK_EXT_external_memory_host
typedef struct VkMemoryHostPointerPropertiesEXT {
    VkStructureType    sType;
    void*              pNext;
    uint32_t           memoryTypeBits;
} VkMemoryHostPointerPropertiesEXT;
```

- `sType` is a VkStructureType value identifying this structure.

- pNext is NULL or a pointer to a structure extending this structure.
- memoryTypeBits is a bitmask containing one bit set for every memory type which the specified host pointer **can** be imported as.

The value returned by memoryTypeBits **should** only include bits that identify memory types which are host visible. Implementations **may** include bits that identify memory types which are not host visible. Behavior for imported pointers of such types is defined by VkImportMemoryHostPointerInfoEXT.

### Valid Usage (Implicit)

- VUID-VkMemoryHostPointerPropertiesEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_MEMORY_HOST_POINTER_PROPERTIES_EXT
- VUID-VkMemoryHostPointerPropertiesEXT-pNext-pNext
  pNext **must** be NULL

## 11.2.7. Android Hardware Buffer External Memory

To import memory created outside of the current Vulkan instance from an Android hardware buffer, add a VkImportAndroidHardwareBufferInfoANDROID structure to the pNext chain of the VkMemoryAllocateInfo structure. The VkImportAndroidHardwareBufferInfoANDROID structure is defined as:

```
// Provided by VK_ANDROID_external_memory_android_hardware_buffer
typedef struct VkImportAndroidHardwareBufferInfoANDROID {
    VkStructureType            sType;
    const void*                pNext;
    struct AHardwareBuffer*    buffer;
} VkImportAndroidHardwareBufferInfoANDROID;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- buffer is the Android hardware buffer to import.

If the vkAllocateMemory command succeeds, the implementation **must** acquire a reference to the imported hardware buffer, which it **must** release when the device memory object is freed. If the command fails, the implementation **must** not retain a reference.

### Valid Usage

- VUID-VkImportAndroidHardwareBufferInfoANDROID-buffer-01880
  If buffer is not NULL, Android hardware buffers **must** be supported for import, as reported by VkExternalImageFormatProperties or VkExternalBufferProperties
- VUID-VkImportAndroidHardwareBufferInfoANDROID-buffer-01881

If `buffer` is not `NULL`, it **must** be a valid Android hardware buffer object with `AHardwareBuffer_Desc::usage` compatible with Vulkan as described in Android Hardware Buffers

### Valid Usage (Implicit)

- VUID-VkImportAndroidHardwareBufferInfoANDROID-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_IMPORT_ANDROID_HARDWARE_BUFFER_INFO_ANDROID`
- VUID-VkImportAndroidHardwareBufferInfoANDROID-buffer-parameter
  `buffer` **must** be a valid pointer to an AHardwareBuffer value

To export an Android hardware buffer referencing the payload of a Vulkan device memory object, call:

```
// Provided by VK_ANDROID_external_memory_android_hardware_buffer
VkResult vkGetMemoryAndroidHardwareBufferANDROID(
    VkDevice                                    device,
    const VkMemoryGetAndroidHardwareBufferInfoANDROID* pInfo,
    struct AHardwareBuffer**                    pBuffer);
```

- `device` is the logical device that created the device memory being exported.
- `pInfo` is a pointer to a VkMemoryGetAndroidHardwareBufferInfoANDROID structure containing parameters of the export operation.
- `pBuffer` will return an Android hardware buffer referencing the payload of the device memory object.

Each call to `vkGetMemoryAndroidHardwareBufferANDROID` **must** return an Android hardware buffer with a new reference acquired in addition to the reference held by the VkDeviceMemory. To avoid leaking resources, the application **must** release the reference by calling `AHardwareBuffer_release` when it is no longer needed. When called with the same handle in VkMemoryGetAndroidHardwareBufferInfoANDROID::`memory`, `vkGetMemoryAndroidHardwareBufferANDROID` **must** return the same Android hardware buffer object. If the device memory was created by importing an Android hardware buffer, `vkGetMemoryAndroidHardwareBufferANDROID` **must** return that same Android hardware buffer object.

### Valid Usage (Implicit)

- VUID-vkGetMemoryAndroidHardwareBufferANDROID-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkGetMemoryAndroidHardwareBufferANDROID-pInfo-parameter
  `pInfo` **must** be a valid pointer to a valid VkMemoryGetAndroidHardwareBufferInfoANDROID structure
- VUID-vkGetMemoryAndroidHardwareBufferANDROID-pBuffer-parameter

pBuffer **must** be a valid pointer to a valid pointer to an AHardwareBuffer value

## Return Codes

**Success**

- VK_SUCCESS

**Failure**

- VK_ERROR_TOO_MANY_OBJECTS
- VK_ERROR_OUT_OF_HOST_MEMORY

The VkMemoryGetAndroidHardwareBufferInfoANDROID structure is defined as:

```
// Provided by VK_ANDROID_external_memory_android_hardware_buffer
typedef struct VkMemoryGetAndroidHardwareBufferInfoANDROID {
    VkStructureType    sType;
    const void*        pNext;
    VkDeviceMemory     memory;
} VkMemoryGetAndroidHardwareBufferInfoANDROID;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- memory is the memory object from which the Android hardware buffer will be exported.

## Valid Usage

- VUID-VkMemoryGetAndroidHardwareBufferInfoANDROID-handleTypes-01882
  VK_EXTERNAL_MEMORY_HANDLE_TYPE_ANDROID_HARDWARE_BUFFER_BIT_ANDROID **must** have been included in VkExportMemoryAllocateInfo::handleTypes when memory was created
- VUID-VkMemoryGetAndroidHardwareBufferInfoANDROID-pNext-01883
  If the pNext chain of the VkMemoryAllocateInfo used to allocate memory included a VkMemoryDedicatedAllocateInfo with non-NULL image member, then that image **must** already be bound to memory

## Valid Usage (Implicit)

- VUID-VkMemoryGetAndroidHardwareBufferInfoANDROID-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_MEMORY_GET_ANDROID_HARDWARE_BUFFER_INFO_ANDROID
- VUID-VkMemoryGetAndroidHardwareBufferInfoANDROID-pNext-pNext
  pNext **must** be NULL
- VUID-VkMemoryGetAndroidHardwareBufferInfoANDROID-memory-parameter
  memory **must** be a valid VkDeviceMemory handle

To determine the memory parameters to use when importing an Android hardware buffer, call:

```
// Provided by VK_ANDROID_external_memory_android_hardware_buffer
VkResult vkGetAndroidHardwareBufferPropertiesANDROID(
    VkDevice                                    device,
    const struct AHardwareBuffer*               buffer,
    VkAndroidHardwareBufferPropertiesANDROID*   pProperties);
```

- `device` is the logical device that will be importing `buffer`.
- `buffer` is the Android hardware buffer which will be imported.
- `pProperties` is a pointer to a VkAndroidHardwareBufferPropertiesANDROID structure in which the properties of `buffer` are returned.

## Valid Usage

- VUID-vkGetAndroidHardwareBufferPropertiesANDROID-buffer-01884
  `buffer` **must** be a valid Android hardware buffer object with at least one of the `AHARDWAREBUFFER_USAGE_GPU_*` flags in its `AHardwareBuffer_Desc::usage`

## Valid Usage (Implicit)

- VUID-vkGetAndroidHardwareBufferPropertiesANDROID-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkGetAndroidHardwareBufferPropertiesANDROID-buffer-parameter
  `buffer` **must** be a valid pointer to a valid AHardwareBuffer value
- VUID-vkGetAndroidHardwareBufferPropertiesANDROID-pProperties-parameter
  `pProperties` **must** be a valid pointer to a VkAndroidHardwareBufferPropertiesANDROID structure

## Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_INVALID_EXTERNAL_HANDLE_KHR`

The `VkAndroidHardwareBufferPropertiesANDROID` structure returned is defined as:

```
// Provided by VK_ANDROID_external_memory_android_hardware_buffer
typedef struct VkAndroidHardwareBufferPropertiesANDROID {
    VkStructureType    sType;
    void*              pNext;
    VkDeviceSize       allocationSize;
    uint32_t           memoryTypeBits;
} VkAndroidHardwareBufferPropertiesANDROID;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `allocationSize` is the size of the external memory
- `memoryTypeBits` is a bitmask containing one bit set for every memory type which the specified Android hardware buffer **can** be imported as.

### Valid Usage (Implicit)

- VUID-VkAndroidHardwareBufferPropertiesANDROID-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_ANDROID_HARDWARE_BUFFER_PROPERTIES_ANDROID`
- VUID-VkAndroidHardwareBufferPropertiesANDROID-pNext-pNext
  Each `pNext` member of any structure (including this one) in the `pNext` chain **must** be either `NULL` or a pointer to a valid instance of VkAndroidHardwareBufferFormatProperties2ANDROID, VkAndroidHardwareBufferFormatPropertiesANDROID, or VkAndroidHardwareBufferFormatResolvePropertiesANDROID
- VUID-VkAndroidHardwareBufferPropertiesANDROID-sType-unique
  The `sType` value of each struct in the `pNext` chain **must** be unique

To obtain format properties of an Android hardware buffer, include a `VkAndroidHardwareBufferFormatPropertiesANDROID` structure in the `pNext` chain of the VkAndroidHardwareBufferPropertiesANDROID structure passed to vkGetAndroidHardwareBufferPropertiesANDROID. This structure is defined as:

```
// Provided by VK_ANDROID_external_memory_android_hardware_buffer
typedef struct VkAndroidHardwareBufferFormatPropertiesANDROID {
    VkStructureType                    sType;
    void*                              pNext;
    VkFormat                           format;
    uint64_t                           externalFormat;
    VkFormatFeatureFlags               formatFeatures;
    VkComponentMapping                 samplerYcbcrConversionComponents;
    VkSamplerYcbcrModelConversion      suggestedYcbcrModel;
    VkSamplerYcbcrRange                suggestedYcbcrRange;
    VkChromaLocation                   suggestedXChromaOffset;
    VkChromaLocation                   suggestedYChromaOffset;
} VkAndroidHardwareBufferFormatPropertiesANDROID;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `format` is the Vulkan format corresponding to the Android hardware buffer's format, or `VK_FORMAT_UNDEFINED` if there is not an equivalent Vulkan format.
- `externalFormat` is an implementation-defined external format identifier for use with VkExternalFormatANDROID. It **must** not be zero.
- `formatFeatures` describes the capabilities of this external format when used with an image bound to memory imported from `buffer`.
- `samplerYcbcrConversionComponents` is the component swizzle that **should** be used in VkSamplerYcbcrConversionCreateInfo.
- `suggestedYcbcrModel` is a suggested color model to use in the VkSamplerYcbcrConversionCreateInfo.
- `suggestedYcbcrRange` is a suggested numerical value range to use in VkSamplerYcbcrConversionCreateInfo.
- `suggestedXChromaOffset` is a suggested X chroma offset to use in VkSamplerYcbcrConversionCreateInfo.
- `suggestedYChromaOffset` is a suggested Y chroma offset to use in VkSamplerYcbcrConversionCreateInfo.

If the Android hardware buffer has one of the formats listed in the Format Equivalence table, then `format` **must** have the equivalent Vulkan format listed in the table. Otherwise, `format` **may** be `VK_FORMAT_UNDEFINED`, indicating the Android hardware buffer **can** only be used with an external format.

The `formatFeatures` member **must** include `VK_FORMAT_FEATURE_SAMPLED_IMAGE_BIT` and at least one of `VK_FORMAT_FEATURE_MIDPOINT_CHROMA_SAMPLES_BIT` or `VK_FORMAT_FEATURE_COSITED_CHROMA_SAMPLES_BIT`, and **should** include `VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT` and `VK_FORMAT_FEATURE_SAMPLED_IMAGE_YCBCR_CONVERSION_LINEAR_FILTER_BIT`.

**NOTE** The `formatFeatures` member only indicates the features available when using an

> external-format image created from the Android hardware buffer. Images from Android hardware buffers with a format other than VK_FORMAT_UNDEFINED are subject to the format capabilities obtained from vkGetPhysicalDeviceFormatProperties2, and vkGetPhysicalDeviceImageFormatProperties2 with appropriate parameters. These sets of features are independent of each other, e.g. the external format will support sampler $Y'C_BC_R$ conversion even if the non-external format does not, and rendering directly to the external format will not be supported even if the non-external format does support this.

Android hardware buffers with the same external format **must** have the same support for VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT, VK_FORMAT_FEATURE_MIDPOINT_CHROMA_SAMPLES_BIT, VK_FORMAT_FEATURE_COSITED_CHROMA_SAMPLES_BIT, VK_FORMAT_FEATURE_SAMPLED_IMAGE_YCBCR_CONVERSION_LINEAR_FILTER_BIT, VK_FORMAT_FEATURE_SAMPLED_IMAGE_YCBCR_CONVERSION_SEPARATE_RECONSTRUCTION_FILTER_BIT, and VK_FORMAT_FEATURE_SAMPLED_IMAGE_YCBCR_CONVERSION_CHROMA_RECONSTRUCTION_EXPLICIT_FORCEABLE_BIT. in formatFeatures. Other format features **may** differ between Android hardware buffers that have the same external format. This allows applications to use the same VkSamplerYcbcrConversion object (and samplers and pipelines created from them) for any Android hardware buffers that have the same external format.

If format is not VK_FORMAT_UNDEFINED, then the value of samplerYcbcrConversionComponents **must** be valid when used as the components member of VkSamplerYcbcrConversionCreateInfo with that format. If format is VK_FORMAT_UNDEFINED, all members of samplerYcbcrConversionComponents **must** be the identity swizzle.

Implementations **may** not always be able to determine the color model, numerical range, or chroma offsets of the image contents, so the values in VkAndroidHardwareBufferFormatPropertiesANDROID are only suggestions. Applications **should** treat these values as sensible defaults to use in the absence of more reliable information obtained through some other means. If the underlying physical device is also usable via OpenGL ES with the GL_OES_EGL_image_external extension, the implementation **should** suggest values that will produce similar sampled values as would be obtained by sampling the same external image via samplerExternalOES in OpenGL ES using equivalent sampler parameters.

> **NOTE** Since GL_OES_EGL_image_external does not require the same sampling and conversion calculations as Vulkan does, achieving identical results between APIs **may** not be possible on some implementations.

### Valid Usage (Implicit)

- VUID-VkAndroidHardwareBufferFormatPropertiesANDROID-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_ANDROID_HARDWARE_BUFFER_FORMAT_PROPERTIES_ANDROID

The format properties of an Android hardware buffer **can** be obtained by including a VkAndroidHardwareBufferFormatProperties2ANDROID structure in the pNext chain of the VkAndroidHardwareBufferPropertiesANDROID structure passed to vkGetAndroidHardwareBufferPropertiesANDROID. This structure is defined as:

```
// Provided by VK_ANDROID_external_memory_android_hardware_buffer with
VK_KHR_format_feature_flags2 or VK_VERSION_1_3
typedef struct VkAndroidHardwareBufferFormatProperties2ANDROID {
    VkStructureType                 sType;
    void*                           pNext;
    VkFormat                        format;
    uint64_t                        externalFormat;
    VkFormatFeatureFlags2           formatFeatures;
    VkComponentMapping              samplerYcbcrConversionComponents;
    VkSamplerYcbcrModelConversion   suggestedYcbcrModel;
    VkSamplerYcbcrRange             suggestedYcbcrRange;
    VkChromaLocation                suggestedXChromaOffset;
    VkChromaLocation                suggestedYChromaOffset;
} VkAndroidHardwareBufferFormatProperties2ANDROID;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `format` is the Vulkan format corresponding to the Android hardware buffer's format, or `VK_FORMAT_UNDEFINED` if there is not an equivalent Vulkan format.
- `externalFormat` is an implementation-defined external format identifier for use with VkExternalFormatANDROID. It **must** not be zero.
- `formatFeatures` describes the capabilities of this external format when used with an image bound to memory imported from `buffer`.
- `samplerYcbcrConversionComponents` is the component swizzle that **should** be used in VkSamplerYcbcrConversionCreateInfo.
- `suggestedYcbcrModel` is a suggested color model to use in the VkSamplerYcbcrConversionCreateInfo.
- `suggestedYcbcrRange` is a suggested numerical value range to use in VkSamplerYcbcrConversionCreateInfo.
- `suggestedXChromaOffset` is a suggested X chroma offset to use in VkSamplerYcbcrConversionCreateInfo.
- `suggestedYChromaOffset` is a suggested Y chroma offset to use in VkSamplerYcbcrConversionCreateInfo.

The bits reported in `formatFeatures` **must** include the bits reported in the corresponding fields of `VkAndroidHardwareBufferFormatPropertiesANDROID::formatFeatures`.

### Valid Usage (Implicit)

- VUID-VkAndroidHardwareBufferFormatProperties2ANDROID-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_ANDROID_HARDWARE_BUFFER_FORMAT_PROPERTIES_2_ANDROID`

The VkAndroidHardwareBufferFormatResolvePropertiesANDROID structure is defined as:

```
// Provided by VK_ANDROID_external_format_resolve
typedef struct VkAndroidHardwareBufferFormatResolvePropertiesANDROID {
    VkStructureType    sType;
    void*              pNext;
    VkFormat           colorAttachmentFormat;
} VkAndroidHardwareBufferFormatResolvePropertiesANDROID;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `colorAttachmentFormat` is a VkFormat specifying the format of color attachment images that **must** be used for color attachments when resolving to the specified external format. If the implementation supports external format resolves for the specified external format, this value will be set to a color format supporting the `VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BIT` in VkFormatProperties::`optimalTilingFeatures` as returned by vkGetPhysicalDeviceFormatProperties with `format` equal to `colorAttachmentFormat` If external format resolves are not supported, this value will be set to `VK_FORMAT_UNDEFINED`.

Any Android hardware buffer created with the `GRALLOC_USAGE_HW_RENDER` flag **must** be renderable in some way in Vulkan, either:

- VkAndroidHardwareBufferFormatPropertiesANDROID::`format` **must** be a format that supports `VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BIT` or `VK_FORMAT_FEATURE_DEPTH_STENCIL_ATTACHMENT_BIT` in VkFormatProperties::`optimalTilingFeatures`; or
- `colorAttachmentFormat` **must** be a format that supports `VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BIT` in VkFormatProperties::`optimalTilingFeatures`.

**Valid Usage (Implicit)**

- VUID-VkAndroidHardwareBufferFormatResolvePropertiesANDROID-sType-sType `sType` **must** be `VK_STRUCTURE_TYPE_ANDROID_HARDWARE_BUFFER_FORMAT_RESOLVE_PROPERTIES_ANDROID`

### 11.2.8. Remote Device External Memory

To export an address representing the payload of a Vulkan device memory object accessible by remote devices, call:

```
// Provided by VK_NV_external_memory_rdma
VkResult vkGetMemoryRemoteAddressNV(
    VkDevice                                    device,
    const VkMemoryGetRemoteAddressInfoNV*       pMemoryGetRemoteAddressInfo,
    VkRemoteAddressNV*                          pAddress);
```

- `device` is the logical device that created the device memory being exported.

- pMemoryGetRemoteAddressInfo is a pointer to a VkMemoryGetRemoteAddressInfoNV structure containing parameters of the export operation.
- pAddress is a pointer to a VkRemoteAddressNV value in which an address representing the payload of the device memory object is returned.

More communication may be required between the kernel-mode drivers of the devices involved. This information is out of scope of this documentation and should be requested from the vendors of the devices.

### Valid Usage (Implicit)

- VUID-vkGetMemoryRemoteAddressNV-device-parameter
  device **must** be a valid VkDevice handle
- VUID-vkGetMemoryRemoteAddressNV-pMemoryGetRemoteAddressInfo-parameter
  pMemoryGetRemoteAddressInfo **must** be a valid pointer to a valid VkMemoryGetRemoteAddressInfoNV structure
- VUID-vkGetMemoryRemoteAddressNV-pAddress-parameter
  pAddress **must** be a valid pointer to a VkRemoteAddressNV value

### Return Codes

**Success**

- VK_SUCCESS

**Failure**

- VK_ERROR_INVALID_EXTERNAL_HANDLE

The VkMemoryGetRemoteAddressInfoNV structure is defined as:

```
// Provided by VK_NV_external_memory_rdma
typedef struct VkMemoryGetRemoteAddressInfoNV {
    VkStructureType                         sType;
    const void*                             pNext;
    VkDeviceMemory                          memory;
    VkExternalMemoryHandleTypeFlagBits      handleType;
} VkMemoryGetRemoteAddressInfoNV;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- memory is the memory object from which the remote accessible address will be exported.
- handleType is the type of handle requested.

### Valid Usage

- VUID-VkMemoryGetRemoteAddressInfoNV-handleType-04966
  `handleType` **must** have been included in VkExportMemoryAllocateInfo::`handleTypes` when `memory` was created

### Valid Usage (Implicit)

- VUID-VkMemoryGetRemoteAddressInfoNV-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_MEMORY_GET_REMOTE_ADDRESS_INFO_NV`
- VUID-VkMemoryGetRemoteAddressInfoNV-pNext-pNext
  `pNext` **must** be `NULL`
- VUID-VkMemoryGetRemoteAddressInfoNV-memory-parameter
  `memory` **must** be a valid VkDeviceMemory handle
- VUID-VkMemoryGetRemoteAddressInfoNV-handleType-parameter
  `handleType` **must** be a valid VkExternalMemoryHandleTypeFlagBits value

VkRemoteAddressNV represents an address of a memory object accessible by remote devices, as returned in vkGetMemoryRemoteAddressNV::`pAddress`.

```
// Provided by VK_NV_external_memory_rdma
typedef void* VkRemoteAddressNV;
```

## 11.2.9. Fuchsia External Memory

On Fuchsia, when allocating memory that **may** be imported from another device, process or Vulkan instance, add a VkImportMemoryZirconHandleInfoFUCHSIA structure to the `pNext` chain of the VkMemoryAllocateInfo structure.

External memory on Fuchsia is imported and exported using VMO handles of type `zx_handle_t`. VMO handles to external memory are canonically obtained from Fuchsia's Sysmem service or from syscalls such as `zx_vmo_create`(). VMO handles for import can also be obtained by exporting them from another Vulkan instance as described in exporting fuchsia device memory.

Importing VMO handles to the Vulkan instance transfers ownership of the handle to the instance from the application. The application **must** not perform any operations on the handle after successful import.

Applications **can** import the same underlying memory into multiple instances of Vulkan, into the same instance from which it was exported, and multiple times into a given Vulkan instance. In all cases, each import operation **must** create a distinct `VkDeviceMemory` object.

### Importing Fuchsia External Memory

The VkImportMemoryZirconHandleInfoFUCHSIA structure is defined as:

```
// Provided by VK_FUCHSIA_external_memory
typedef struct VkImportMemoryZirconHandleInfoFUCHSIA {
    VkStructureType                         sType;
    const void*                             pNext;
    VkExternalMemoryHandleTypeFlagBits      handleType;
    zx_handle_t                             handle;
} VkImportMemoryZirconHandleInfoFUCHSIA;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- handleType is a VkExternalMemoryHandleTypeFlagBits value specifying the type of handle.
- handle is a zx_handle_t (Zircon) handle to the external memory.

#### Valid Usage

- VUID-VkImportMemoryZirconHandleInfoFUCHSIA-handleType-04771
  handleType **must** be VK_EXTERNAL_MEMORY_HANDLE_TYPE_ZIRCON_VMO_BIT_FUCHSIA
- VUID-VkImportMemoryZirconHandleInfoFUCHSIA-handle-04772
  handle **must** be a valid VMO handle

#### Valid Usage (Implicit)

- VUID-VkImportMemoryZirconHandleInfoFUCHSIA-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_IMPORT_MEMORY_ZIRCON_HANDLE_INFO_FUCHSIA
- VUID-VkImportMemoryZirconHandleInfoFUCHSIA-handleType-parameter
  If handleType is not 0, handleType **must** be a valid VkExternalMemoryHandleTypeFlagBits value

To obtain the memoryTypeIndex for the VkMemoryAllocateInfo structure, call vkGetMemoryZirconHandlePropertiesFUCHSIA:

```
// Provided by VK_FUCHSIA_external_memory
VkResult vkGetMemoryZirconHandlePropertiesFUCHSIA(
    VkDevice                                    device,
    VkExternalMemoryHandleTypeFlagBits          handleType,
    zx_handle_t                                 zirconHandle,
    VkMemoryZirconHandlePropertiesFUCHSIA*      pMemoryZirconHandleProperties);
```

- device is the VkDevice.

- handleType is a VkExternalMemoryHandleTypeFlagBits value specifying the type of zirconHandle
- zirconHandle is a zx_handle_t (Zircon) handle to the external resource.
- pMemoryZirconHandleProperties is a pointer to a VkMemoryZirconHandlePropertiesFUCHSIA structure in which the result will be stored.

### Valid Usage

- VUID-vkGetMemoryZirconHandlePropertiesFUCHSIA-handleType-04773
  handleType **must** be VK_EXTERNAL_MEMORY_HANDLE_TYPE_ZIRCON_VMO_BIT_FUCHSIA
- VUID-vkGetMemoryZirconHandlePropertiesFUCHSIA-zirconHandle-04774
  zirconHandle **must** reference a valid VMO

### Valid Usage (Implicit)

- VUID-vkGetMemoryZirconHandlePropertiesFUCHSIA-device-parameter
  device **must** be a valid VkDevice handle
- VUID-vkGetMemoryZirconHandlePropertiesFUCHSIA-handleType-parameter
  handleType **must** be a valid VkExternalMemoryHandleTypeFlagBits value
- VUID-vkGetMemoryZirconHandlePropertiesFUCHSIA-pMemoryZirconHandleProperties-parameter
  pMemoryZirconHandleProperties **must** be a valid pointer to a VkMemoryZirconHandlePropertiesFUCHSIA structure

### Return Codes

**Success**

- VK_SUCCESS

**Failure**

- VK_ERROR_INVALID_EXTERNAL_HANDLE

The VkMemoryZirconHandlePropertiesFUCHSIA structure is defined as:

```
// Provided by VK_FUCHSIA_external_memory
typedef struct VkMemoryZirconHandlePropertiesFUCHSIA {
    VkStructureType    sType;
    void*              pNext;
    uint32_t           memoryTypeBits;
} VkMemoryZirconHandlePropertiesFUCHSIA;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.

- memoryTypeBits a bitmask containing one bit set for every memory type which the specified handle can be imported as.

### Valid Usage (Implicit)

- VUID-VkMemoryZirconHandlePropertiesFUCHSIA-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_MEMORY_ZIRCON_HANDLE_PROPERTIES_FUCHSIA
- VUID-VkMemoryZirconHandlePropertiesFUCHSIA-pNext-pNext
  pNext **must** be NULL

With pMemoryZirconHandleProperties now successfully populated by vkGetMemoryZirconHandlePropertiesFUCHSIA, assign the VkMemoryAllocateInfo memoryTypeIndex field to a memory type which has a bit set in the VkMemoryZirconHandlePropertiesFUCHSIA memoryTypeBits field.

#### Exporting Fuchsia Device Memory

Similar to importing, exporting a VMO handle from Vulkan transfers ownership of the handle from the Vulkan instance to the application. The application is responsible for closing the handle with zx_handle_close() when it is no longer in use.

To export device memory as a Zircon handle that can be used by another instance, device, or process, retrieve the handle to the VkDeviceMemory using the command:

```
// Provided by VK_FUCHSIA_external_memory
VkResult vkGetMemoryZirconHandleFUCHSIA(
    VkDevice                                    device,
    const VkMemoryGetZirconHandleInfoFUCHSIA*   pGetZirconHandleInfo,
    zx_handle_t*                                pZirconHandle);
```

- device is the VkDevice.
- pGetZirconHandleInfo is a pointer to a VkMemoryGetZirconHandleInfoFUCHSIA structure.
- pZirconHandle is a pointer to a zx_handle_t which holds the resulting Zircon handle.

### Valid Usage (Implicit)

- VUID-vkGetMemoryZirconHandleFUCHSIA-device-parameter
  device **must** be a valid VkDevice handle
- VUID-vkGetMemoryZirconHandleFUCHSIA-pGetZirconHandleInfo-parameter
  pGetZirconHandleInfo **must** be a valid pointer to a valid VkMemoryGetZirconHandleInfoFUCHSIA structure
- VUID-vkGetMemoryZirconHandleFUCHSIA-pZirconHandle-parameter
  pZirconHandle **must** be a valid pointer to a zx_handle_t value

## Return Codes

**Success**

- VK_SUCCESS

**Failure**

- VK_ERROR_TOO_MANY_OBJECTS
- VK_ERROR_OUT_OF_HOST_MEMORY

VkMemoryGetZirconHandleInfoFUCHSIA is defined as:

```
// Provided by VK_FUCHSIA_external_memory
typedef struct VkMemoryGetZirconHandleInfoFUCHSIA {
    VkStructureType                         sType;
    const void*                             pNext;
    VkDeviceMemory                          memory;
    VkExternalMemoryHandleTypeFlagBits      handleType;
} VkMemoryGetZirconHandleInfoFUCHSIA;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- memory the VkDeviceMemory being exported.
- handleType is a VkExternalMemoryHandleTypeFlagBits value specifying the type of the handle pointed to by vkGetMemoryZirconHandleFUCHSIA::pZirconHandle.

## Valid Usage

- VUID-VkMemoryGetZirconHandleInfoFUCHSIA-handleType-04775
  handleType **must** be VK_EXTERNAL_MEMORY_HANDLE_TYPE_ZIRCON_VMO_BIT_FUCHSIA
- VUID-VkMemoryGetZirconHandleInfoFUCHSIA-handleType-04776
  handleType **must** have been included in the handleTypes field of the VkExportMemoryAllocateInfo structure when the external memory was allocated

## Valid Usage (Implicit)

- VUID-VkMemoryGetZirconHandleInfoFUCHSIA-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_MEMORY_GET_ZIRCON_HANDLE_INFO_FUCHSIA
- VUID-VkMemoryGetZirconHandleInfoFUCHSIA-pNext-pNext
  pNext **must** be NULL
- VUID-VkMemoryGetZirconHandleInfoFUCHSIA-memory-parameter
  memory **must** be a valid VkDeviceMemory handle
- VUID-VkMemoryGetZirconHandleInfoFUCHSIA-handleType-parameter

handleType **must** be a valid VkExternalMemoryHandleTypeFlagBits value

With the result pZirconHandle now obtained, the memory properties for the handle can be retrieved using vkGetMemoryZirconHandlePropertiesFUCHSIA as documented above substituting the dereferenced, retrieved pZirconHandle in for the zirconHandle argument.

### 11.2.10. Metal Objects

A Vulkan implementation that is layered on top of Metal on Apple device platform, and implements the VK_EXT_metal_objects extension, supports the ability to import and export the underlying Metal objects associated with specific Vulkan objects.

The underlying Metal objects associated with certain Vulkan objects can be exported from those Vulkan objects using the pNext chain of the VkExportMetalObjectsInfoEXT parameter of the vkExportMetalObjectsEXT command.

An VkDeviceMemory object can be allocated on an existing MTLBuffer object, by including the MTLBuffer object in a VkImportMetalBufferInfoEXT structure in the pNext chain of the VkMemoryAllocateInfo structure in the vkAllocateMemory command.

A new VkImage object can be created on an existing IOSurface object, or one or more existing Metal MTLTexture objects, by including those Metal objects in either VkImportMetalIOSurfaceInfoEXT or VkImportMetalTextureInfoEXT structures in the pNext chain of the VkImageCreateInfo structure in the vkCreateImage command.

To export Metal objects from Vulkan objects, the application **must** first indicate the intention to do so during the creation of the Vulkan object, by including one or more VkExportMetalObjectCreateInfoEXT structures in the pNext chain of the VkInstanceCreateInfo, VkMemoryAllocateInfo, VkImageCreateInfo, VkImageViewCreateInfo, VkBufferViewCreateInfo, VkSemaphoreCreateInfo, or VkEventCreateInfo, in the corresponding Vulkan object creation command.

The VkExportMetalObjectCreateInfoEXT structure is defined as:

```
// Provided by VK_EXT_metal_objects
typedef struct VkExportMetalObjectCreateInfoEXT {
    VkStructureType                         sType;
    const void*                             pNext;
    VkExportMetalObjectTypeFlagBitsEXT      exportObjectType;
} VkExportMetalObjectCreateInfoEXT;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- exportObjectType is a VkExportMetalObjectTypeFlagBitsEXT indicating the type of Metal object that the application may request to be exported from the Vulkan object.

### Valid Usage (Implicit)

- VUID-VkExportMetalObjectCreateInfoEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_EXPORT_METAL_OBJECT_CREATE_INFO_EXT`
- VUID-VkExportMetalObjectCreateInfoEXT-exportObjectType-parameter
  If `exportObjectType` is not `0`, `exportObjectType` **must** be a valid VkExportMetalObjectTypeFlagBitsEXT value

Bits which indicate the types of Metal objects that may be exported from a corresponding Vulkan object are:

```
// Provided by VK_EXT_metal_objects
typedef enum VkExportMetalObjectTypeFlagBitsEXT {
    VK_EXPORT_METAL_OBJECT_TYPE_METAL_DEVICE_BIT_EXT = 0x00000001,
    VK_EXPORT_METAL_OBJECT_TYPE_METAL_COMMAND_QUEUE_BIT_EXT = 0x00000002,
    VK_EXPORT_METAL_OBJECT_TYPE_METAL_BUFFER_BIT_EXT = 0x00000004,
    VK_EXPORT_METAL_OBJECT_TYPE_METAL_TEXTURE_BIT_EXT = 0x00000008,
    VK_EXPORT_METAL_OBJECT_TYPE_METAL_IOSURFACE_BIT_EXT = 0x00000010,
    VK_EXPORT_METAL_OBJECT_TYPE_METAL_SHARED_EVENT_BIT_EXT = 0x00000020,
} VkExportMetalObjectTypeFlagBitsEXT;
```

- `VK_EXPORT_METAL_OBJECT_TYPE_METAL_DEVICE_BIT_EXT` indicates a Metal `MTLDevice` may be exported.
- `VK_EXPORT_METAL_OBJECT_TYPE_METAL_COMMAND_QUEUE_BIT_EXT` indicates a Metal `MTLCommandQueue` may be exported.
- `VK_EXPORT_METAL_OBJECT_TYPE_METAL_BUFFER_BIT_EXT` indicates a Metal `MTLBuffer` may be exported.
- `VK_EXPORT_METAL_OBJECT_TYPE_METAL_TEXTURE_BIT_EXT` indicates a Metal `MTLTexture` may be exported.
- `VK_EXPORT_METAL_OBJECT_TYPE_METAL_IOSURFACE_BIT_EXT` indicates a Metal `IOSurface` may be exported.
- `VK_EXPORT_METAL_OBJECT_TYPE_METAL_SHARED_EVENT_BIT_EXT` indicates a Metal `MTLSharedEvent` may be exported.

```
// Provided by VK_EXT_metal_objects
typedef VkFlags VkExportMetalObjectTypeFlagsEXT;
```

`VkExportMetalObjectTypeFlagsEXT` is a bitmask type for setting a mask of zero or more VkExportMetalObjectTypeFlagBitsEXT.

To export Metal objects that underlie Vulkan objects, call:

```
// Provided by VK_EXT_metal_objects
void vkExportMetalObjectsEXT(
    VkDevice                                    device,
    VkExportMetalObjectsInfoEXT*                pMetalObjectsInfo);
```

- device is the device that created the Vulkan objects.
- pMetalObjectsInfo is a pointer to a VkExportMetalObjectsInfoEXT structure whose pNext chain contains structures, each identifying a Vulkan object and providing a pointer through which the Metal object will be returned.

### Valid Usage (Implicit)

- VUID-vkExportMetalObjectsEXT-device-parameter
  device **must** be a valid VkDevice handle
- VUID-vkExportMetalObjectsEXT-pMetalObjectsInfo-parameter
  pMetalObjectsInfo **must** be a valid pointer to a VkExportMetalObjectsInfoEXT structure

The VkExportMetalObjectsInfoEXT structure is defined as:

```
// Provided by VK_EXT_metal_objects
typedef struct VkExportMetalObjectsInfoEXT {
    VkStructureType    sType;
    const void*        pNext;
} VkExportMetalObjectsInfoEXT;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.

### Valid Usage

- VUID-VkExportMetalObjectsInfoEXT-pNext-06791
  If the pNext chain includes a VkExportMetalDeviceInfoEXT structure, the VkInstance **must** have been created with VK_EXPORT_METAL_OBJECT_TYPE_METAL_DEVICE_BIT_EXT in the exportObjectType member of a VkExportMetalObjectCreateInfoEXT structure in the pNext chain of the VkInstanceCreateInfo structure in the vkCreateInstance command
- VUID-VkExportMetalObjectsInfoEXT-pNext-06792
  If the pNext chain includes a VkExportMetalCommandQueueInfoEXT structure, the VkInstance **must** have been created with VK_EXPORT_METAL_OBJECT_TYPE_METAL_COMMAND_QUEUE_BIT_EXT in the exportObjectType member of a VkExportMetalObjectCreateInfoEXT structure in the pNext chain of the VkInstanceCreateInfo structure in the vkCreateInstance command
- VUID-VkExportMetalObjectsInfoEXT-pNext-06793
  If the pNext chain includes a VkExportMetalBufferInfoEXT structure, the

VkDeviceMemory in its memory member **must** have been allocated with VK_EXPORT_METAL_OBJECT_TYPE_METAL_BUFFER_BIT_EXT in the exportObjectType member of a VkExportMetalObjectCreateInfoEXT structure in the pNext chain of the VkMemoryAllocateInfo structure in the vkAllocateMemory command

- VUID-VkExportMetalObjectsInfoEXT-pNext-06794
  If the pNext chain includes a VkExportMetalTextureInfoEXT structure, exactly one of its image, imageView, or bufferView members **must** not be VK_NULL_HANDLE
- VUID-VkExportMetalObjectsInfoEXT-pNext-06795
  If the pNext chain includes a VkExportMetalTextureInfoEXT structure, and its image member is not VK_NULL_HANDLE, the VkImage in its image member **must** have been created with VK_EXPORT_METAL_OBJECT_TYPE_METAL_TEXTURE_BIT_EXT in the exportObjectType member of a VkExportMetalObjectCreateInfoEXT structure in the pNext chain of the VkImageCreateInfo structure in the vkCreateImage command
- VUID-VkExportMetalObjectsInfoEXT-pNext-06796
  If the pNext chain includes a VkExportMetalTextureInfoEXT structure, and its imageView member is not VK_NULL_HANDLE, the VkImageView in its imageView member **must** have been created with VK_EXPORT_METAL_OBJECT_TYPE_METAL_TEXTURE_BIT_EXT in the exportObjectType member of a VkExportMetalObjectCreateInfoEXT structure in the pNext chain of the VkImageViewCreateInfo structure in the vkCreateImageView command
- VUID-VkExportMetalObjectsInfoEXT-pNext-06797
  If the pNext chain includes a VkExportMetalTextureInfoEXT structure, and its bufferView member is not VK_NULL_HANDLE, the VkBufferView in its bufferView member **must** have been created with VK_EXPORT_METAL_OBJECT_TYPE_METAL_TEXTURE_BIT_EXT in the exportObjectType member of a VkExportMetalObjectCreateInfoEXT structure in the pNext chain of the VkBufferViewCreateInfo structure in the vkCreateBufferView command
- VUID-VkExportMetalObjectsInfoEXT-pNext-06798
  If the pNext chain includes a VkExportMetalTextureInfoEXT structure, and if either its image or imageView member is not VK_NULL_HANDLE, then plane **must** be VK_IMAGE_ASPECT_PLANE_0_BIT, VK_IMAGE_ASPECT_PLANE_1_BIT, or VK_IMAGE_ASPECT_PLANE_2_BIT
- VUID-VkExportMetalObjectsInfoEXT-pNext-06799
  If the pNext chain includes a VkExportMetalTextureInfoEXT structure, and if the VkImage in its image member does not have a multi-planar format, then its plane member **must** be VK_IMAGE_ASPECT_PLANE_0_BIT
- VUID-VkExportMetalObjectsInfoEXT-pNext-06800
  If the pNext chain includes a VkExportMetalTextureInfoEXT structure, and if the VkImage in its image member has a multi-planar format with only two planes, then its plane member **must** not be VK_IMAGE_ASPECT_PLANE_2_BIT
- VUID-VkExportMetalObjectsInfoEXT-pNext-06801
  If the pNext chain includes a VkExportMetalTextureInfoEXT structure, and if the VkImageView in its imageView member does not have a multi-planar format, then its plane member **must** be VK_IMAGE_ASPECT_PLANE_0_BIT
- VUID-VkExportMetalObjectsInfoEXT-pNext-06802
  If the pNext chain includes a VkExportMetalTextureInfoEXT structure, and if the VkImageView in its imageView member has a multi-planar format with only two planes,

then its plane member **must** not be VK_IMAGE_ASPECT_PLANE_2_BIT

- VUID-VkExportMetalObjectsInfoEXT-pNext-06803
  If the pNext chain includes a VkExportMetalIOSurfaceInfoEXT structure, the VkImage in its image member **must** have been created with VK_EXPORT_METAL_OBJECT_TYPE_METAL_IOSURFACE_BIT_EXT in the exportObjectType member of a VkExportMetalObjectCreateInfoEXT structure in the pNext chain of the VkImageCreateInfo structure in the vkCreateImage command
- VUID-VkExportMetalObjectsInfoEXT-pNext-06804
  If the pNext chain includes a VkExportMetalSharedEventInfoEXT structure, exactly one of its semaphore or event members **must** not be VK_NULL_HANDLE
- VUID-VkExportMetalObjectsInfoEXT-pNext-06805
  If the pNext chain includes a VkExportMetalSharedEventInfoEXT structure, and its semaphore member is not VK_NULL_HANDLE, the VkSemaphore in its semaphore member **must** have been created with VK_EXPORT_METAL_OBJECT_TYPE_METAL_SHARED_EVENT_BIT_EXT in the exportObjectType member of a VkExportMetalObjectCreateInfoEXT structure in the pNext chain of the VkSemaphoreCreateInfo structure in the vkCreateSemaphore command
- VUID-VkExportMetalObjectsInfoEXT-pNext-06806
  If the pNext chain includes a VkExportMetalSharedEventInfoEXT structure, and its event member is not VK_NULL_HANDLE, the VkEvent in its event member **must** have been created with VK_EXPORT_METAL_OBJECT_TYPE_METAL_SHARED_EVENT_BIT_EXT in the exportObjectType member of a VkExportMetalObjectCreateInfoEXT structure in the pNext chain of the VkEventCreateInfo structure in the vkCreateEvent command

### Valid Usage (Implicit)

- VUID-VkExportMetalObjectsInfoEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_EXPORT_METAL_OBJECTS_INFO_EXT
- VUID-VkExportMetalObjectsInfoEXT-pNext-pNext
  Each pNext member of any structure (including this one) in the pNext chain **must** be either NULL or a pointer to a valid instance of VkExportMetalBufferInfoEXT, VkExportMetalCommandQueueInfoEXT, VkExportMetalDeviceInfoEXT, VkExportMetalIOSurfaceInfoEXT, VkExportMetalSharedEventInfoEXT, or VkExportMetalTextureInfoEXT
- VUID-VkExportMetalObjectsInfoEXT-sType-unique
  The sType value of each struct in the pNext chain **must** be unique, with the exception of structures of type VkExportMetalBufferInfoEXT, VkExportMetalCommandQueueInfoEXT, VkExportMetalIOSurfaceInfoEXT, VkExportMetalSharedEventInfoEXT, or VkExportMetalTextureInfoEXT

To export the Metal MTLDevice object underlying the VkPhysicalDevice associated with a VkDevice object, include a VkExportMetalDeviceInfoEXT structure in the pNext chain of the pMetalObjectsInfo parameter of a vkExportMetalObjectsEXT call.

The VkExportMetalDeviceInfoEXT structure is defined as:

```
// Provided by VK_EXT_metal_objects
typedef struct VkExportMetalDeviceInfoEXT {
    VkStructureType    sType;
    const void*        pNext;
    MTLDevice_id       mtlDevice;
} VkExportMetalDeviceInfoEXT;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- mtlDevice is the Metal id<MTLDevice> object underlying the VkPhysicalDevice associated with the VkDevice object identified in the call. The implementation will return the MTLDevice in this member, or it will return NULL if no MTLDevice could be found underlying the VkPhysicalDevice object.

### Valid Usage (Implicit)

- VUID-VkExportMetalDeviceInfoEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_EXPORT_METAL_DEVICE_INFO_EXT

The type id<MTLDevice> is defined in Apple's Metal framework, but to remove an unnecessary compile time dependency, an incomplete type definition of MTLDevice_id is provided in the Vulkan headers:

```
// Provided by VK_EXT_metal_objects
#ifdef __OBJC__
@protocol MTLDevice;
typedef __unsafe_unretained id<MTLDevice> MTLDevice_id;
#else
typedef void* MTLDevice_id;
#endif
```

To export the Metal MTLCommandQueue object underlying a VkQueue object, include a VkExportMetalCommandQueueInfoEXT structure in the pNext chain of the pMetalObjectsInfo parameter of a vkExportMetalObjectsEXT call.

The VkExportMetalCommandQueueInfoEXT structure is defined as:

```
// Provided by VK_EXT_metal_objects
typedef struct VkExportMetalCommandQueueInfoEXT {
    VkStructureType       sType;
    const void*           pNext;
    VkQueue               queue;
    MTLCommandQueue_id    mtlCommandQueue;
} VkExportMetalCommandQueueInfoEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `queue` is a VkQueue.
- `mtlCommandQueue` is the Metal `id<MTLCommandQueue>` object underlying the VkQueue object in `queue`. The implementation will return the `MTLCommandQueue` in this member, or it will return `NULL` if no `MTLCommandQueue` could be found underlying the VkQueue object.

### Valid Usage (Implicit)

- VUID-VkExportMetalCommandQueueInfoEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_EXPORT_METAL_COMMAND_QUEUE_INFO_EXT`
- VUID-VkExportMetalCommandQueueInfoEXT-queue-parameter
  `queue` **must** be a valid VkQueue handle

The type `id<MTLCommandQueue>` is defined in Apple's Metal framework, but to remove an unnecessary compile time dependency, an incomplete type definition of MTLCommandQueue_id is provided in the Vulkan headers:

```
// Provided by VK_EXT_metal_objects
#ifdef __OBJC__
@protocol MTLCommandQueue;
typedef __unsafe_unretained id<MTLCommandQueue> MTLCommandQueue_id;
#else
typedef void* MTLCommandQueue_id;
#endif
```

To export the Metal `MTLBuffer` object underlying a VkDeviceMemory object, include a `VkExportMetalBufferInfoEXT` structure in the `pNext` chain of the `pMetalObjectsInfo` parameter of a vkExportMetalObjectsEXT call.

The `VkExportMetalBufferInfoEXT` structure is defined as:

```
// Provided by VK_EXT_metal_objects
typedef struct VkExportMetalBufferInfoEXT {
    VkStructureType    sType;
    const void*        pNext;
    VkDeviceMemory     memory;
    MTLBuffer_id       mtlBuffer;
} VkExportMetalBufferInfoEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `memory` is a VkDeviceMemory.
- `mtlBuffer` is the Metal `id<MTLBuffer>` object underlying the VkDeviceMemory object in `memory`. The implementation will return the `MTLBuffer` in this member, or it will return `NULL` if no `MTLBuffer` could be found underlying the VkDeviceMemory object.

### Valid Usage (Implicit)

- VUID-VkExportMetalBufferInfoEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_EXPORT_METAL_BUFFER_INFO_EXT`
- VUID-VkExportMetalBufferInfoEXT-memory-parameter
  `memory` **must** be a valid VkDeviceMemory handle

To import a Metal `MTLBuffer` object to underlie a VkDeviceMemory object, include a `VkImportMetalBufferInfoEXT` structure in the `pNext` chain of the VkMemoryAllocateInfo structure in a vkAllocateMemory command.

The `VkImportMetalBufferInfoEXT` structure is defined as:

```
// Provided by VK_EXT_metal_objects
typedef struct VkImportMetalBufferInfoEXT {
    VkStructureType    sType;
    const void*        pNext;
    MTLBuffer_id       mtlBuffer;
} VkImportMetalBufferInfoEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `mtlBuffer` is the Metal `id<MTLBuffer>` object that is to underlie the VkDeviceMemory.

The application **must** ensure that the configuration of the `id<MTLBuffer>` object is compatible with the configuration of the VkDeviceMemory. Failure to do so results in undefined behavior.

### Valid Usage (Implicit)

- VUID-VkImportMetalBufferInfoEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_IMPORT_METAL_BUFFER_INFO_EXT`

The type `id<MTLBuffer>` is defined in Apple's Metal framework, but to remove an unnecessary compile time dependency, an incomplete type definition of MTLBuffer_id is provided in the Vulkan headers:

```
// Provided by VK_EXT_metal_objects
#ifdef __OBJC__
@protocol MTLBuffer;
typedef __unsafe_unretained id<MTLBuffer> MTLBuffer_id;
#else
typedef void* MTLBuffer_id;
#endif
```

To export a Metal `MTLTexture` object underlying a VkImage, VkImageView, or VkBufferView object, include a `VkExportMetalTextureInfoEXT` structure in the `pNext` chain of the `pMetalObjectsInfo` parameter of a vkExportMetalObjectsEXT call.

The `VkExportMetalTextureInfoEXT` structure is defined as:

```
// Provided by VK_EXT_metal_objects
typedef struct VkExportMetalTextureInfoEXT {
    VkStructureType          sType;
    const void*              pNext;
    VkImage                  image;
    VkImageView              imageView;
    VkBufferView             bufferView;
    VkImageAspectFlagBits    plane;
    MTLTexture_id            mtlTexture;
} VkExportMetalTextureInfoEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `image` is VK_NULL_HANDLE or a VkImage.
- `imageView` is VK_NULL_HANDLE or a VkImageView.
- `bufferView` is VK_NULL_HANDLE or a VkBufferView.
- `plane` indicates the plane of a multi-planar VkImage or VkImageView.
- `mtlTexture` is the Metal `id<MTLTexture>` object underlying the VkImage, VkImageView, or VkBufferView object in `image`, `imageView`, or `bufferView`, respectively, at the plane indicated in `aspectMask`. The implementation will return the `MTLTexture` in this member, or it will return `NULL` if no `MTLTexture` could be found underlying the VkImage, VkImageView, or VkBufferView object,

at the plane indicated in aspectMask.

### Valid Usage (Implicit)

- VUID-VkExportMetalTextureInfoEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_EXPORT_METAL_TEXTURE_INFO_EXT
- VUID-VkExportMetalTextureInfoEXT-image-parameter
  If image is not VK_NULL_HANDLE, image **must** be a valid VkImage handle
- VUID-VkExportMetalTextureInfoEXT-imageView-parameter
  If imageView is not VK_NULL_HANDLE, imageView **must** be a valid VkImageView handle
- VUID-VkExportMetalTextureInfoEXT-bufferView-parameter
  If bufferView is not VK_NULL_HANDLE, bufferView **must** be a valid VkBufferView handle
- VUID-VkExportMetalTextureInfoEXT-plane-parameter
  plane **must** be a valid VkImageAspectFlagBits value
- VUID-VkExportMetalTextureInfoEXT-commonparent
  Each of bufferView, image, and imageView that are valid handles of non-ignored parameters **must** have been created, allocated, or retrieved from the same VkDevice

To import one or more existing Metal MTLTexture objects to underlie a VkImage object, include one or more VkImportMetalTextureInfoEXT structures in the pNext chain of the VkImageCreateInfo structure in a vkCreateImage command.

The VkImportMetalTextureInfoEXT structure is defined as:

```
// Provided by VK_EXT_metal_objects
typedef struct VkImportMetalTextureInfoEXT {
    VkStructureType          sType;
    const void*              pNext;
    VkImageAspectFlagBits    plane;
    MTLTexture_id            mtlTexture;
} VkImportMetalTextureInfoEXT;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- plane indicates the plane of the VkImage that the id<MTLTexture> object should be attached to.
- mtlTexture is a the Metal id<MTLTexture> object that is to underlie the VkImage plane.

The pNext chain **must** include one VkImportMetalTextureInfoEXT structure for each plane in the VkImage. The application **must** ensure that the configuration of the Metal id<MTLTexture> objects are compatible with the configuration of the VkImage. Failure to do so results in undefined behavior.

### Valid Usage (Implicit)

- VUID-VkImportMetalTextureInfoEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_IMPORT_METAL_TEXTURE_INFO_EXT`
- VUID-VkImportMetalTextureInfoEXT-plane-parameter
  `plane` **must** be a valid VkImageAspectFlagBits value

The type `id<MTLTexture>` is defined in Apple's Metal framework, but to remove an unnecessary compile time dependency, an incomplete type definition of MTLTexture_id is provided in the Vulkan headers:

```
// Provided by VK_EXT_metal_objects
#ifdef __OBJC__
@protocol MTLTexture;
typedef __unsafe_unretained id<MTLTexture> MTLTexture_id;
#else
typedef void* MTLTexture_id;
#endif
```

To export the Metal IOSurfaceRef object underlying a VkImage object, include a `VkExportMetalIOSurfaceInfoEXT` structure in the `pNext` chain of the `pMetalObjectsInfo` parameter of a vkExportMetalObjectsEXT call.

The `VkExportMetalIOSurfaceInfoEXT` structure is defined as:

```
// Provided by VK_EXT_metal_objects
typedef struct VkExportMetalIOSurfaceInfoEXT {
    VkStructureType    sType;
    const void*        pNext;
    VkImage            image;
    IOSurfaceRef       ioSurface;
} VkExportMetalIOSurfaceInfoEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `image` is a VkImage.
- `ioSurface` is the Metal IOSurfaceRef object underlying the VkImage object in `image`. The implementation will return the IOSurfaceRef in this member, or it will return `NULL` if no IOSurfaceRef could be found underlying the VkImage object.

### Valid Usage (Implicit)

- VUID-VkExportMetalIOSurfaceInfoEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_EXPORT_METAL_IO_SURFACE_INFO_EXT`

- VUID-VkExportMetalIOSurfaceInfoEXT-image-parameter
  `image` **must** be a valid VkImage handle

To import, or create, a Metal IOSurfaceRef object to underlie a VkImage object, include a `VkImportMetalIOSurfaceInfoEXT` structure in the `pNext` chain of the VkImageCreateInfo structure in a vkCreateImage command.

The `VkImportMetalIOSurfaceInfoEXT` structure is defined as:

```
// Provided by VK_EXT_metal_objects
typedef struct VkImportMetalIOSurfaceInfoEXT {
    VkStructureType    sType;
    const void*        pNext;
    IOSurfaceRef       ioSurface;
} VkImportMetalIOSurfaceInfoEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `ioSurface` is VK_NULL_HANDLE or the Metal IOSurfaceRef object that is to underlie the VkImage.

If `ioSurface` is not VK_NULL_HANDLE, it will be used to underlie the VkImage. If `ioSurface` is VK_NULL_HANDLE, the implementation will create a new `IOSurface` to underlie the VkImage.

If provided, the application **must** ensure that the configuration of the IOSurfaceRef object is compatible with the configuration of the VkImage. Failure to do so results in undefined behavior.

### Valid Usage (Implicit)

- VUID-VkImportMetalIOSurfaceInfoEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_IMPORT_METAL_IO_SURFACE_INFO_EXT`

The type IOSurfaceRef is defined in Apple's CoreGraphics framework, but to remove an unnecessary compile time dependency, an incomplete type definition of IOSurfaceRef is provided in the Vulkan headers:

```
// Provided by VK_EXT_metal_objects
typedef struct __IOSurface* IOSurfaceRef;
```

To export the Metal `MTLSharedEvent` object underlying a VkSemaphore or VkEvent object, include a `VkExportMetalSharedEventInfoEXT` structure in the `pNext` chain of the `pMetalObjectsInfo` parameter of a vkExportMetalObjectsEXT call.

The `VkExportMetalSharedEventInfoEXT` structure is defined as:

```
// Provided by VK_EXT_metal_objects
typedef struct VkExportMetalSharedEventInfoEXT {
    VkStructureType      sType;
    const void*          pNext;
    VkSemaphore          semaphore;
    VkEvent              event;
    MTLSharedEvent_id    mtlSharedEvent;
} VkExportMetalSharedEventInfoEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `semaphore` is VK_NULL_HANDLE or a VkSemaphore.
- `event` is VK_NULL_HANDLE or a VkEvent.
- `mtlSharedEvent` is the Metal `id<MTLSharedEvent>` object underlying the VkSemaphore or VkEvent object in `semaphore` or `event`, respectively. The implementation will return the `MTLSharedEvent` in this member, or it will return `NULL` if no `MTLSharedEvent` could be found underlying the VkSemaphore or VkEvent object.

### Valid Usage (Implicit)

- VUID-VkExportMetalSharedEventInfoEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_EXPORT_METAL_SHARED_EVENT_INFO_EXT`
- VUID-VkExportMetalSharedEventInfoEXT-semaphore-parameter
  If `semaphore` is not VK_NULL_HANDLE, `semaphore` **must** be a valid VkSemaphore handle
- VUID-VkExportMetalSharedEventInfoEXT-event-parameter
  If `event` is not VK_NULL_HANDLE, `event` **must** be a valid VkEvent handle
- VUID-VkExportMetalSharedEventInfoEXT-commonparent
  Both of `event`, and `semaphore` that are valid handles of non-ignored parameters **must** have been created, allocated, or retrieved from the same VkDevice

To import a Metal `id<MTLSharedEvent>` object to underlie a VkSemaphore or VkEvent object, include a `VkImportMetalSharedEventInfoEXT` structure in the `pNext` chain of the VkSemaphoreCreateInfo or VkEventCreateInfo structure in a vkCreateSemaphore or vkCreateEvent command, respectively.

The `VkImportMetalSharedEventInfoEXT` structure is defined as:

```
// Provided by VK_EXT_metal_objects
typedef struct VkImportMetalSharedEventInfoEXT {
    VkStructureType      sType;
    const void*          pNext;
    MTLSharedEvent_id    mtlSharedEvent;
} VkImportMetalSharedEventInfoEXT;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- mtlSharedEvent is the Metal id<MTLSharedEvent> object that is to underlie the VkSemaphore or VkEvent.

If the pNext chain of the VkSemaphoreCreateInfo structure includes both VkImportMetalSharedEventInfoEXT and VkSemaphoreTypeCreateInfo, the signaledValue property of the imported id<MTLSharedEvent> object will be set to initialValue of VkSemaphoreTypeCreateInfo.

**Valid Usage (Implicit)**

- VUID-VkImportMetalSharedEventInfoEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_IMPORT_METAL_SHARED_EVENT_INFO_EXT

The type id<MTLSharedEvent> is defined in Apple's Metal framework, but to remove an unnecessary compile time dependency, an incomplete type definition of MTLSharedEvent_id is provided in the Vulkan headers:

```
// Provided by VK_EXT_metal_objects
#ifdef __OBJC__
@protocol MTLSharedEvent;
typedef __unsafe_unretained id<MTLSharedEvent> MTLSharedEvent_id;
#else
typedef void* MTLSharedEvent_id;
#endif
```

### 11.2.11. QNX Screen Buffer External Memory

To import memory created outside of the current Vulkan instance from a QNX Screen buffer, add a VkImportScreenBufferInfoQNX structure to the pNext chain of the VkMemoryAllocateInfo structure. The VkImportScreenBufferInfoQNX structure is defined as:

```
// Provided by VK_QNX_external_memory_screen_buffer
typedef struct VkImportScreenBufferInfoQNX {
    VkStructureType           sType;
    const void*               pNext;
    struct _screen_buffer*    buffer;
} VkImportScreenBufferInfoQNX;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- buffer is a pointer to a struct _screen_buffer, the QNX Screen buffer to import

The implementation **may** not acquire a reference to the imported Screen buffer. Therefore, the application **must** ensure that the object referred to by buffer stays valid as long as the device

memory to which it is imported is being used.

### Valid Usage

- VUID-VkImportScreenBufferInfoQNX-buffer-08966
  If `buffer` is not `NULL`, QNX Screen Buffers **must** be supported for import, as reported by VkExternalImageFormatProperties or VkExternalBufferProperties
- VUID-VkImportScreenBufferInfoQNX-buffer-08967
  `buffer` is not `NULL`, it **must** be a pointer to valid QNX Screen buffer

### Valid Usage (Implicit)

- VUID-VkImportScreenBufferInfoQNX-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_IMPORT_SCREEN_BUFFER_INFO_QNX`

To determine the memory parameters to use when importing a QNX Screen buffer, call:

```
// Provided by VK_QNX_external_memory_screen_buffer
VkResult vkGetScreenBufferPropertiesQNX(
    VkDevice                                    device,
    const struct _screen_buffer*                buffer,
    VkScreenBufferPropertiesQNX*                pProperties);
```

- `device` is the logical device that will be importing `buffer`.
- `buffer` is the QNX Screen buffer which will be imported.
- `pProperties` is a pointer to a VkScreenBufferPropertiesQNX structure in which the properties of `buffer` are returned.

### Valid Usage

- VUID-vkGetScreenBufferPropertiesQNX-buffer-08968
  `buffer` **must** be a valid QNX Screen buffer

### Valid Usage (Implicit)

- VUID-vkGetScreenBufferPropertiesQNX-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkGetScreenBufferPropertiesQNX-buffer-parameter
  `buffer` **must** be a valid pointer to a valid `_screen_buffer` value
- VUID-vkGetScreenBufferPropertiesQNX-pProperties-parameter
  `pProperties` **must** be a valid pointer to a VkScreenBufferPropertiesQNX structure

### Return Codes

**Success**

- VK_SUCCESS

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_INVALID_EXTERNAL_HANDLE_KHR

The VkScreenBufferPropertiesQNX structure returned is defined as:

```
// Provided by VK_QNX_external_memory_screen_buffer
typedef struct VkScreenBufferPropertiesQNX {
    VkStructureType    sType;
    void*              pNext;
    VkDeviceSize       allocationSize;
    uint32_t           memoryTypeBits;
} VkScreenBufferPropertiesQNX;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- allocationSize is the size of the external memory.
- memoryTypeBits is a bitmask containing one bit set for every memory type which the specified Screen buffer **can** be imported as.

### Valid Usage (Implicit)

- VUID-VkScreenBufferPropertiesQNX-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_SCREEN_BUFFER_PROPERTIES_QNX
- VUID-VkScreenBufferPropertiesQNX-pNext-pNext
  pNext **must** be NULL or a pointer to a valid instance of VkScreenBufferFormatPropertiesQNX
- VUID-VkScreenBufferPropertiesQNX-sType-unique
  The sType value of each struct in the pNext chain **must** be unique

To obtain format properties of a QNX Screen buffer, include a VkScreenBufferFormatPropertiesQNX structure in the pNext chain of the VkScreenBufferPropertiesQNX structure passed to vkGetScreenBufferPropertiesQNX. This structure is defined as:

```
// Provided by VK_QNX_external_memory_screen_buffer
typedef struct VkScreenBufferFormatPropertiesQNX {
    VkStructureType                 sType;
    void*                           pNext;
    VkFormat                        format;
    uint64_t                        externalFormat;
    uint64_t                        screenUsage;
    VkFormatFeatureFlags            formatFeatures;
    VkComponentMapping              samplerYcbcrConversionComponents;
    VkSamplerYcbcrModelConversion   suggestedYcbcrModel;
    VkSamplerYcbcrRange             suggestedYcbcrRange;
    VkChromaLocation                suggestedXChromaOffset;
    VkChromaLocation                suggestedYChromaOffset;
} VkScreenBufferFormatPropertiesQNX;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `format` is the Vulkan format corresponding to the Screen buffer's format or `VK_FORMAT_UNDEFINED` if there is not an equivalent Vulkan format.
- `externalFormat` is an implementation-defined external format identifier for use with VkExternalFormatQNX. It **must** not be zero.
- `screenUsage` is an implementation-defined external usage identifier for the QNX Screen buffer.
- `formatFeatures` describes the capabilities of this external format when used with an image bound to memory imported from `buffer`.
- `samplerYcbcrConversionComponents` is the component swizzle that **should** be used in VkSamplerYcbcrConversionCreateInfo.
- `suggestedYcbcrModel` is a suggested color model to use in the VkSamplerYcbcrConversionCreateInfo.
- `suggestedYcbcrRange` is a suggested numerical value range to use in VkSamplerYcbcrConversionCreateInfo.
- `suggestedXChromaOffset` is a suggested X chroma offset to use in VkSamplerYcbcrConversionCreateInfo.
- `suggestedYChromaOffset` is a suggested Y chroma offset to use in VkSamplerYcbcrConversionCreateInfo.

If the QNX Screen buffer has one of the formats listed in the QNX Screen Format Equivalence table, then `format` **must** have the equivalent Vulkan format listed in the table. Otherwise, `format` **may** be `VK_FORMAT_UNDEFINED`, indicating the QNX Screen buffer **can** only be used with an external format. The `formatFeatures` member **must** include `VK_FORMAT_FEATURE_SAMPLED_IMAGE_BIT` and **should** include `VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT` and `VK_FORMAT_FEATURE_SAMPLED_IMAGE_YCBCR_CONVERSION_LINEAR_FILTER_BIT`.

### Valid Usage (Implicit)

- VUID-VkScreenBufferFormatPropertiesQNX-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_SCREEN_BUFFER_FORMAT_PROPERTIES_QNX`

## 11.2.12. Device Group Memory Allocations

If the `pNext` chain of VkMemoryAllocateInfo includes a `VkMemoryAllocateFlagsInfo` structure, then that structure includes flags and a device mask controlling how many instances of the memory will be allocated.

The `VkMemoryAllocateFlagsInfo` structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkMemoryAllocateFlagsInfo {
    VkStructureType          sType;
    const void*              pNext;
    VkMemoryAllocateFlags    flags;
    uint32_t                 deviceMask;
} VkMemoryAllocateFlagsInfo;
```

or the equivalent

```
// Provided by VK_KHR_device_group
typedef VkMemoryAllocateFlagsInfo VkMemoryAllocateFlagsInfoKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is a bitmask of VkMemoryAllocateFlagBits controlling the allocation.
- `deviceMask` is a mask of physical devices in the logical device, indicating that memory **must** be allocated on each device in the mask, if `VK_MEMORY_ALLOCATE_DEVICE_MASK_BIT` is set in `flags`.

If `VK_MEMORY_ALLOCATE_DEVICE_MASK_BIT` is not set, the number of instances allocated depends on whether `VK_MEMORY_HEAP_MULTI_INSTANCE_BIT` is set in the memory heap. If `VK_MEMORY_HEAP_MULTI_INSTANCE_BIT` is set, then memory is allocated for every physical device in the logical device (as if `deviceMask` has bits set for all device indices). If `VK_MEMORY_HEAP_MULTI_INSTANCE_BIT` is not set, then a single instance of memory is allocated (as if `deviceMask` is set to one).

On some implementations, allocations from a multi-instance heap **may** consume memory on all physical devices even if the `deviceMask` excludes some devices. If VkPhysicalDeviceGroupProperties::`subsetAllocation` is `VK_TRUE`, then memory is only consumed for the devices in the device mask.

**NOTE** In practice, most allocations on a multi-instance heap will be allocated across all

> physical devices. Unicast allocation support is an optional optimization for a minority of allocations.

### Valid Usage

- VUID-VkMemoryAllocateFlagsInfo-deviceMask-00675
  If `VK_MEMORY_ALLOCATE_DEVICE_MASK_BIT` is set, `deviceMask` **must** be a valid device mask
- VUID-VkMemoryAllocateFlagsInfo-deviceMask-00676
  If `VK_MEMORY_ALLOCATE_DEVICE_MASK_BIT` is set, `deviceMask` **must** not be zero

### Valid Usage (Implicit)

- VUID-VkMemoryAllocateFlagsInfo-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_MEMORY_ALLOCATE_FLAGS_INFO`
- VUID-VkMemoryAllocateFlagsInfo-flags-parameter
  `flags` **must** be a valid combination of VkMemoryAllocateFlagBits values

Bits which **can** be set in VkMemoryAllocateFlagsInfo::`flags`, controlling device memory allocation, are:

```
// Provided by VK_VERSION_1_1
typedef enum VkMemoryAllocateFlagBits {
    VK_MEMORY_ALLOCATE_DEVICE_MASK_BIT = 0x00000001,
    VK_MEMORY_ALLOCATE_DEVICE_ADDRESS_BIT = 0x00000002,
    VK_MEMORY_ALLOCATE_DEVICE_ADDRESS_CAPTURE_REPLAY_BIT = 0x00000004,
  // Provided by VK_KHR_device_group
    VK_MEMORY_ALLOCATE_DEVICE_MASK_BIT_KHR = VK_MEMORY_ALLOCATE_DEVICE_MASK_BIT,
  // Provided by VK_KHR_buffer_device_address
    VK_MEMORY_ALLOCATE_DEVICE_ADDRESS_BIT_KHR = VK_MEMORY_ALLOCATE_DEVICE_ADDRESS_BIT,
  // Provided by VK_KHR_buffer_device_address
    VK_MEMORY_ALLOCATE_DEVICE_ADDRESS_CAPTURE_REPLAY_BIT_KHR =
VK_MEMORY_ALLOCATE_DEVICE_ADDRESS_CAPTURE_REPLAY_BIT,
} VkMemoryAllocateFlagBits;
```

or the equivalent

```
// Provided by VK_KHR_device_group
typedef VkMemoryAllocateFlagBits VkMemoryAllocateFlagBitsKHR;
```

- `VK_MEMORY_ALLOCATE_DEVICE_MASK_BIT` specifies that memory will be allocated for the devices in VkMemoryAllocateFlagsInfo::`deviceMask`.
- `VK_MEMORY_ALLOCATE_DEVICE_ADDRESS_BIT` specifies that the memory **can** be attached to a buffer object created with the `VK_BUFFER_USAGE_SHADER_DEVICE_ADDRESS_BIT` bit set in `usage`, and that the

memory handle **can** be used to retrieve an opaque address via vkGetDeviceMemoryOpaqueCaptureAddressKHR.

- VK_MEMORY_ALLOCATE_DEVICE_ADDRESS_CAPTURE_REPLAY_BIT specifies that the memory's address **can** be saved and reused on a subsequent run (e.g. for trace capture and replay), see VkBufferOpaqueCaptureAddressCreateInfo for more detail.

```
// Provided by VK_VERSION_1_1
typedef VkFlags VkMemoryAllocateFlags;
```

or the equivalent

```
// Provided by VK_KHR_device_group
typedef VkMemoryAllocateFlags VkMemoryAllocateFlagsKHR;
```

VkMemoryAllocateFlags is a bitmask type for setting a mask of zero or more VkMemoryAllocateFlagBits.

### 11.2.13. Opaque Capture Address Allocation

To request a specific device address for a memory allocation, add a VkMemoryOpaqueCaptureAddressAllocateInfo structure to the pNext chain of the VkMemoryAllocateInfo structure. The VkMemoryOpaqueCaptureAddressAllocateInfo structure is defined as:

```
typedef struct VkMemoryOpaqueCaptureAddressAllocateInfo {
    VkStructureType    sType;
    const void*        pNext;
    uint64_t           opaqueCaptureAddress;
} VkMemoryOpaqueCaptureAddressAllocateInfo;
```

or the equivalent

```
// Provided by VK_KHR_buffer_device_address
typedef VkMemoryOpaqueCaptureAddressAllocateInfo
VkMemoryOpaqueCaptureAddressAllocateInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- opaqueCaptureAddress is the opaque capture address requested for the memory allocation.

If opaqueCaptureAddress is zero, no specific address is requested.

If opaqueCaptureAddress is not zero, it **should** be an address retrieved from vkGetDeviceMemoryOpaqueCaptureAddressKHR on an identically created memory allocation on

the same implementation.

**NOTE**

In most cases, it is expected that a non-zero `opaqueAddress` is an address retrieved from vkGetDeviceMemoryOpaqueCaptureAddressKHR on an identically created memory allocation. If this is not the case, it is likely that `VK_ERROR_INVALID_OPAQUE_CAPTURE_ADDRESS` errors will occur.

This is, however, not a strict requirement because trace capture/replay tools may need to adjust memory allocation parameters for imported memory.

If this structure is not present, it is as if `opaqueCaptureAddress` is zero.

### Valid Usage (Implicit)

- VUID-VkMemoryOpaqueCaptureAddressAllocateInfo-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_MEMORY_OPAQUE_CAPTURE_ADDRESS_ALLOCATE_INFO`

## 11.2.14. Freeing Device Memory

To free a memory object, call:

```
// Provided by VK_VERSION_1_0
void vkFreeMemory(
    VkDevice                                    device,
    VkDeviceMemory                              memory,
    const VkAllocationCallbacks*                pAllocator);
```

- `device` is the logical device that owns the memory.
- `memory` is the VkDeviceMemory object to be freed.
- `pAllocator` controls host memory allocation as described in the Memory Allocation chapter.

Before freeing a memory object, an application **must** ensure the memory object is no longer in use by the device — for example by command buffers in the *pending state*. Memory **can** be freed whilst still bound to resources, but those resources **must** not be used afterwards. Freeing a memory object releases the reference it held, if any, to its payload. If there are still any bound images or buffers, the memory object's payload **may** not be immediately released by the implementation, but **must** be released by the time all bound images and buffers have been destroyed. Once all references to a payload are released, it is returned to the heap from which it was allocated.

How memory objects are bound to Images and Buffers is described in detail in the Resource Memory Association section.

If a memory object is mapped at the time it is freed, it is implicitly unmapped.

**NOTE**

As described below, host writes are not implicitly flushed when the memory object is unmapped, but the implementation **must** guarantee that writes that have not

been flushed do not affect any other memory.

### Valid Usage

- VUID-vkFreeMemory-memory-00677
  All submitted commands that refer to `memory` (via images or buffers) **must** have completed execution

### Valid Usage (Implicit)

- VUID-vkFreeMemory-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkFreeMemory-memory-parameter
  If `memory` is not VK_NULL_HANDLE, `memory` **must** be a valid VkDeviceMemory handle
- VUID-vkFreeMemory-pAllocator-parameter
  If `pAllocator` is not `NULL`, `pAllocator` **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkFreeMemory-memory-parent
  If `memory` is a valid handle, it **must** have been created, allocated, or retrieved from `device`

### Host Synchronization

- Host access to `memory` **must** be externally synchronized

## 11.2.15. Host Access to Device Memory Objects

Memory objects created with vkAllocateMemory are not directly host accessible.

Memory objects created with the memory property `VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT` are considered *mappable*. Memory objects **must** be mappable in order to be successfully mapped on the host.

To retrieve a host virtual address pointer to a region of a mappable memory object, call:

```
// Provided by VK_VERSION_1_0
VkResult vkMapMemory(
    VkDevice                                    device,
    VkDeviceMemory                              memory,
    VkDeviceSize                                offset,
    VkDeviceSize                                size,
    VkMemoryMapFlags                            flags,
    void**                                      ppData);
```

- `device` is the logical device that owns the memory.
- `memory` is the VkDeviceMemory object to be mapped.
- `offset` is a zero-based byte offset from the beginning of the memory object.
- `size` is the size of the memory range to map, or `VK_WHOLE_SIZE` to map from `offset` to the end of the allocation.
- `flags` is a bitmask of VkMemoryMapFlagBits specifying additional parameters of the memory map operation.
- `ppData` is a pointer to a `void*` variable in which a host-accessible pointer to the beginning of the mapped range is returned. This pointer minus `offset` **must** be aligned to at least VkPhysicalDeviceLimits::`minMemoryMapAlignment`.

After a successful call to `vkMapMemory` the memory object `memory` is considered to be currently *host mapped*.

| | |
|---|---|
| **NOTE** | It is an application error to call `vkMapMemory` on a memory object that is already *host mapped*. |
| **NOTE** | `vkMapMemory` will fail if the implementation is unable to allocate an appropriately sized contiguous virtual address range, e.g. due to virtual address space fragmentation or platform limits. In such cases, `vkMapMemory` **must** return `VK_ERROR_MEMORY_MAP_FAILED`. The application **can** improve the likelihood of success by reducing the size of the mapped range and/or removing unneeded mappings using vkUnmapMemory. |

`vkMapMemory` does not check whether the device memory is currently in use before returning the host-accessible pointer. The application **must** guarantee that any previously submitted command that writes to this range has completed before the host reads from or writes to that range, and that any previously submitted command that reads from that range has completed before the host writes to that region (see here for details on fulfilling such a guarantee). If the device memory was allocated without the `VK_MEMORY_PROPERTY_HOST_COHERENT_BIT` set, these guarantees **must** be made for an extended range: the application **must** round down the start of the range to the nearest multiple of VkPhysicalDeviceLimits::`nonCoherentAtomSize`, and round the end of the range up to the nearest multiple of VkPhysicalDeviceLimits::`nonCoherentAtomSize`.

While a range of device memory is host mapped, the application is responsible for synchronizing both device and host access to that memory range.

| | |
|---|---|
| **NOTE** | It is important for the application developer to become meticulously familiar with all of the mechanisms described in the chapter on Synchronization and Cache Control as they are crucial to maintaining memory access ordering. |

Calling `vkMapMemory` is equivalent to calling vkMapMemory2KHR with an empty `pNext` chain.

## Valid Usage

- VUID-vkMapMemory-memory-00678
  `memory` **must** not be currently host mapped
- VUID-vkMapMemory-offset-00679
  `offset` **must** be less than the size of `memory`
- VUID-vkMapMemory-size-00680
  If `size` is not equal to `VK_WHOLE_SIZE`, `size` **must** be greater than `0`
- VUID-vkMapMemory-size-00681
  If `size` is not equal to `VK_WHOLE_SIZE`, `size` **must** be less than or equal to the size of the `memory` minus `offset`
- VUID-vkMapMemory-memory-00682
  `memory` **must** have been created with a memory type that reports `VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT`
- VUID-vkMapMemory-memory-00683
  `memory` **must** not have been allocated with multiple instances
- VUID-vkMapMemory-flags-09568
  `VK_MEMORY_MAP_PLACED_BIT_EXT` **must** not be set in `flags`

## Valid Usage (Implicit)

- VUID-vkMapMemory-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkMapMemory-memory-parameter
  `memory` **must** be a valid VkDeviceMemory handle
- VUID-vkMapMemory-flags-parameter
  `flags` **must** be a valid combination of VkMemoryMapFlagBits values
- VUID-vkMapMemory-ppData-parameter
  `ppData` **must** be a valid pointer to a pointer value
- VUID-vkMapMemory-memory-parent
  `memory` **must** have been created, allocated, or retrieved from `device`

## Host Synchronization

- Host access to `memory` **must** be externally synchronized

## Return Codes

### Success

- `VK_SUCCESS`

### Failure

- `VK_ERROR_OUT_OF_HOST_MEMORY`

- VK_ERROR_OUT_OF_DEVICE_MEMORY
- VK_ERROR_MEMORY_MAP_FAILED

Bits which **can** be set in vkMapMemory::flags and VkMemoryMapInfoKHR::flags, specifying additional properties of a memory map, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkMemoryMapFlagBits {
  // Provided by VK_EXT_map_memory_placed
    VK_MEMORY_MAP_PLACED_BIT_EXT = 0x00000001,
} VkMemoryMapFlagBits;
```

- VK_MEMORY_MAP_PLACED_BIT_EXT requests that the implementation place the memory map at the virtual address specified by the application via VkMemoryMapPlacedInfoEXT::pPlacedAddress, replacing any existing mapping at that address. This flag **must** not be used with vkMapMemory as there is no way to specify the placement address.

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkMemoryMapFlags;
```

VkMemoryMapFlags is a bitmask type for setting a mask of zero or more VkMemoryMapFlagBits.

Alternatively, to retrieve a host virtual address pointer to a region of a mappable memory object, call:

```
// Provided by VK_KHR_map_memory2
VkResult vkMapMemory2KHR(
    VkDevice                                    device,
    const VkMemoryMapInfoKHR*                   pMemoryMapInfo,
    void**                                      ppData);
```

- device is the logical device that owns the memory.
- pMemoryMapInfo is a pointer to a VkMemoryMapInfoKHR structure describing parameters of the map.
- ppData is a pointer to a void * variable in which is returned a host-accessible pointer to the beginning of the mapped range. This pointer minus VkMemoryMapInfoKHR::offset **must** be aligned to at least VkPhysicalDeviceLimits::minMemoryMapAlignment.

This function behaves identically to vkMapMemory except that it gets its parameters via an extensible structure pointer rather than directly as function arguments.

### Valid Usage (Implicit)

- VUID-vkMapMemory2KHR-device-parameter

device **must** be a valid VkDevice handle

- VUID-vkMapMemory2KHR-pMemoryMapInfo-parameter
  pMemoryMapInfo **must** be a valid pointer to a valid VkMemoryMapInfoKHR structure
- VUID-vkMapMemory2KHR-ppData-parameter
  ppData **must** be a valid pointer to a pointer value

## Return Codes

**Success**

- VK_SUCCESS

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY
- VK_ERROR_MEMORY_MAP_FAILED

The VkMemoryMapInfoKHR structure is defined as:

```
// Provided by VK_KHR_map_memory2
typedef struct VkMemoryMapInfoKHR {
    VkStructureType     sType;
    const void*         pNext;
    VkMemoryMapFlags    flags;
    VkDeviceMemory      memory;
    VkDeviceSize        offset;
    VkDeviceSize        size;
} VkMemoryMapInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- flags is a bitmask of VkMemoryMapFlagBits specifying additional parameters of the memory map operation.
- memory is the VkDeviceMemory object to be mapped.
- offset is a zero-based byte offset from the beginning of the memory object.
- size is the size of the memory range to map, or VK_WHOLE_SIZE to map from offset to the end of the allocation.

## Valid Usage

- VUID-VkMemoryMapInfoKHR-memory-07958
  memory **must** not be currently host mapped
- VUID-VkMemoryMapInfoKHR-offset-07959

offset **must** be less than the size of memory

- VUID-VkMemoryMapInfoKHR-size-07960
  If size is not equal to VK_WHOLE_SIZE, size **must** be greater than 0
- VUID-VkMemoryMapInfoKHR-size-07961
  If size is not equal to VK_WHOLE_SIZE, size **must** be less than or equal to the size of the memory minus offset
- VUID-VkMemoryMapInfoKHR-memory-07962
  memory **must** have been created with a memory type that reports VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT
- VUID-VkMemoryMapInfoKHR-memory-07963
  memory **must** not have been allocated with multiple instances
- VUID-VkMemoryMapInfoKHR-flags-09569
  If VK_MEMORY_MAP_PLACED_BIT_EXT is set in flags, the memoryMapPlaced feature **must** be enabled
- VUID-VkMemoryMapInfoKHR-flags-09570
  If VK_MEMORY_MAP_PLACED_BIT_EXT is set in flags, the pNext chain **must** include a VkMemoryMapPlacedInfoEXT structure and VkMemoryMapPlacedInfoEXT::pPlacedAddress **must** not be NULL
- VUID-VkMemoryMapInfoKHR-flags-09571
  If VK_MEMORY_MAP_PLACED_BIT_EXT is set in flags and the memoryMapRangePlaced feature is not enabled, offset **must** be zero
- VUID-VkMemoryMapInfoKHR-flags-09572
  If VK_MEMORY_MAP_PLACED_BIT_EXT is set in flags and the memoryMapRangePlaced feature is not enabled, size **must** be VK_WHOLE_SIZE or VkMemoryAllocateInfo::allocationSize
- VUID-VkMemoryMapInfoKHR-flags-09573
  If VK_MEMORY_MAP_PLACED_BIT_EXT is set in flags and the memoryMapRangePlaced feature is enabled, offset **must** be aligned to an integer multiple of VkPhysicalDeviceMapMemoryPlacedPropertiesEXT::minPlacedMemoryMapAlignment
- VUID-VkMemoryMapInfoKHR-flags-09574
  If VK_MEMORY_MAP_PLACED_BIT_EXT is set in flags and size is not VK_WHOLE_SIZE, size **must** be aligned to an integer multiple of VkPhysicalDeviceMapMemoryPlacedPropertiesEXT::minPlacedMemoryMapAlignment
- VUID-VkMemoryMapInfoKHR-flags-09651
  If VK_MEMORY_MAP_PLACED_BIT_EXT is set in flags and size is VK_WHOLE_SIZE, VkMemoryAllocateInfo::allocationSize **must** be aligned to an integer multiple of VkPhysicalDeviceMapMemoryPlacedPropertiesEXT::minPlacedMemoryMapAlignment
- VUID-VkMemoryMapInfoKHR-flags-09575
  If VK_MEMORY_MAP_PLACED_BIT_EXT is set in flags, the memory object **must** not have been imported from a handle type of VK_EXTERNAL_MEMORY_HANDLE_TYPE_HOST_ALLOCATION_BIT_EXT or VK_EXTERNAL_MEMORY_HANDLE_TYPE_HOST_MAPPED_FOREIGN_MEMORY_BIT_EXT

### Valid Usage (Implicit)

- VUID-VkMemoryMapInfoKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_MEMORY_MAP_INFO_KHR`
- VUID-VkMemoryMapInfoKHR-pNext-pNext
  `pNext` **must** be `NULL` or a pointer to a valid instance of VkMemoryMapPlacedInfoEXT
- VUID-VkMemoryMapInfoKHR-sType-unique
  The `sType` value of each struct in the `pNext` chain **must** be unique
- VUID-VkMemoryMapInfoKHR-flags-parameter
  `flags` **must** be a valid combination of VkMemoryMapFlagBits values
- VUID-VkMemoryMapInfoKHR-memory-parameter
  `memory` **must** be a valid VkDeviceMemory handle

### Host Synchronization

- Host access to `memory` **must** be externally synchronized

If `VK_MEMORY_MAP_PLACED_BIT_EXT` is set in `VkMemoryMapInfoKHR::flags` and the `pNext` chain of VkMemoryMapInfoKHR includes a `VkMemoryMapPlacedInfoEXT` structure, then that structure specifies the placement address of the memory map. The implementation will place the memory map at the specified address, replacing any existing maps in the specified memory range. Replacing memory maps in this way does not implicitly unmap Vulkan memory objects. Instead, the application **must** ensure no other Vulkan memory objects are mapped anywhere in the specified virtual address range. If successful, `ppData` will be set to the same value as `VkMemoryMapPlacedInfoEXT::pPlacedAddress` and `vkMapMemory2KHR` will return `VK_SUCCESS`. If it cannot place the map at the requested address for any reason, the memory object is left unmapped and `vkMapMemory2KHR` will return `VK_ERROR_MEMORY_MAP_FAILED`.

The `VkMemoryMapPlacedInfoEXT` structure is defined as:

```
// Provided by VK_EXT_map_memory_placed
typedef struct VkMemoryMapPlacedInfoEXT {
    VkStructureType    sType;
    const void*        pNext;
    void*              pPlacedAddress;
} VkMemoryMapPlacedInfoEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `pPlacedAddress` is the virtual address at which to place the address. If `VkMemoryMapInfoKHR::flags` does not contain `VK_MEMORY_MAP_PLACED_BIT_EXT`, this value is ignored.

### Valid Usage

- VUID-VkMemoryMapPlacedInfoEXT-flags-09576
  If `VkMemoryMapInfoKHR::flags` contains `VK_MEMORY_MAP_PLACED_BIT_EXT`, `pPlacedAddress` **must** not be `NULL`
- VUID-VkMemoryMapPlacedInfoEXT-pPlacedAddress-09577
  `pPlacedAddress` **must** be aligned to an integer multiple of `VkPhysicalDeviceMapMemoryPlacedPropertiesEXT::minPlacedMemoryMapAlignment`
- VUID-VkMemoryMapPlacedInfoEXT-pPlacedAddress-09578
  The address range specified by `pPlacedAddress` and `VkMemoryMapInfoKHR::size` **must** not overlap any existing Vulkan memory object mapping

### Valid Usage (Implicit)

- VUID-VkMemoryMapPlacedInfoEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_MEMORY_MAP_PLACED_INFO_EXT`

Two commands are provided to enable applications to work with non-coherent memory allocations: `vkFlushMappedMemoryRanges` and `vkInvalidateMappedMemoryRanges`.

**NOTE** If the memory object was created with the `VK_MEMORY_PROPERTY_HOST_COHERENT_BIT` set, `vkFlushMappedMemoryRanges` and `vkInvalidateMappedMemoryRanges` are unnecessary and **may** have a performance cost. However, availability and visibility operations still need to be managed on the device. See the description of host access types for more information.

**NOTE** While memory objects imported from a handle type of `VK_EXTERNAL_MEMORY_HANDLE_TYPE_HOST_ALLOCATION_BIT_EXT` or `VK_EXTERNAL_MEMORY_HANDLE_TYPE_HOST_MAPPED_FOREIGN_MEMORY_BIT_EXT` are inherently mapped to host address space, they are not considered to be host mapped device memory unless they are explicitly host mapped using vkMapMemory. That means flushing or invalidating host caches with respect to host accesses performed on such memory through the original host pointer specified at import time is the responsibility of the application and **must** be performed with appropriate synchronization primitives provided by the platform which are outside the scope of Vulkan. `vkFlushMappedMemoryRanges` and `vkInvalidateMappedMemoryRanges`, however, **can** still be used on such memory objects to synchronize host accesses performed through the host pointer of the host mapped device memory range returned by vkMapMemory.

After a successful call to `vkMapMemory` or `vkMapMemory2KHR` the memory object `memory` is considered to be currently *host mapped*.

To flush ranges of non-coherent memory from the host caches, call:

```
// Provided by VK_VERSION_1_0
VkResult vkFlushMappedMemoryRanges(
    VkDevice                                    device,
    uint32_t                                    memoryRangeCount,
    const VkMappedMemoryRange*                  pMemoryRanges);
```

- `device` is the logical device that owns the memory ranges.
- `memoryRangeCount` is the length of the `pMemoryRanges` array.
- `pMemoryRanges` is a pointer to an array of VkMappedMemoryRange structures describing the memory ranges to flush.

`vkFlushMappedMemoryRanges` guarantees that host writes to the memory ranges described by `pMemoryRanges` are made available to the host memory domain, such that they **can** be made available to the device memory domain via memory domain operations using the `VK_ACCESS_HOST_WRITE_BIT` access type.

Within each range described by `pMemoryRanges`, each set of `nonCoherentAtomSize` bytes in that range is flushed if any byte in that set has been written by the host since it was first host mapped, or the last time it was flushed. If `pMemoryRanges` includes sets of `nonCoherentAtomSize` bytes where no bytes have been written by the host, those bytes **must** not be flushed.

Unmapping non-coherent memory does not implicitly flush the host mapped memory, and host writes that have not been flushed **may** not ever be visible to the device. However, implementations **must** ensure that writes that have not been flushed do not become visible to any other memory.

| | |
|---|---|
| **NOTE** | The above guarantee avoids a potential memory corruption in scenarios where host writes to a mapped memory object have not been flushed before the memory is unmapped (or freed), and the virtual address range is subsequently reused for a different mapping (or memory allocation). |

## Valid Usage (Implicit)

- VUID-vkFlushMappedMemoryRanges-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkFlushMappedMemoryRanges-pMemoryRanges-parameter
  `pMemoryRanges` **must** be a valid pointer to an array of `memoryRangeCount` valid VkMappedMemoryRange structures
- VUID-vkFlushMappedMemoryRanges-memoryRangeCount-arraylength
  `memoryRangeCount` **must** be greater than `0`

## Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY

To invalidate ranges of non-coherent memory from the host caches, call:

```
// Provided by VK_VERSION_1_0
VkResult vkInvalidateMappedMemoryRanges(
    VkDevice                                    device,
    uint32_t                                    memoryRangeCount,
    const VkMappedMemoryRange*                  pMemoryRanges);
```

- device is the logical device that owns the memory ranges.
- memoryRangeCount is the length of the pMemoryRanges array.
- pMemoryRanges is a pointer to an array of VkMappedMemoryRange structures describing the memory ranges to invalidate.

vkInvalidateMappedMemoryRanges guarantees that device writes to the memory ranges described by pMemoryRanges, which have been made available to the host memory domain using the VK_ACCESS_HOST_WRITE_BIT and VK_ACCESS_HOST_READ_BIT access types, are made visible to the host. If a range of non-coherent memory is written by the host and then invalidated without first being flushed, its contents are undefined.

Within each range described by pMemoryRanges, each set of nonCoherentAtomSize bytes in that range is invalidated if any byte in that set has been written by the device since it was first host mapped, or the last time it was invalidated.

**NOTE** Mapping non-coherent memory does not implicitly invalidate that memory.

## Valid Usage (Implicit)

- VUID-vkInvalidateMappedMemoryRanges-device-parameter
  device **must** be a valid VkDevice handle
- VUID-vkInvalidateMappedMemoryRanges-pMemoryRanges-parameter
  pMemoryRanges **must** be a valid pointer to an array of memoryRangeCount valid VkMappedMemoryRange structures
- VUID-vkInvalidateMappedMemoryRanges-memoryRangeCount-arraylength
  memoryRangeCount **must** be greater than 0

## Return Codes

**Success**

- VK_SUCCESS

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY

The VkMappedMemoryRange structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkMappedMemoryRange {
    VkStructureType    sType;
    const void*        pNext;
    VkDeviceMemory     memory;
    VkDeviceSize       offset;
    VkDeviceSize       size;
} VkMappedMemoryRange;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- memory is the memory object to which this range belongs.
- offset is the zero-based byte offset from the beginning of the memory object.
- size is either the size of range, or VK_WHOLE_SIZE to affect the range from offset to the end of the current mapping of the allocation.

## Valid Usage

- VUID-VkMappedMemoryRange-memory-00684
  memory **must** be currently host mapped
- VUID-VkMappedMemoryRange-size-00685
  If size is not equal to VK_WHOLE_SIZE, offset and size **must** specify a range contained within the currently mapped range of memory
- VUID-VkMappedMemoryRange-size-00686
  If size is equal to VK_WHOLE_SIZE, offset **must** be within the currently mapped range of memory
- VUID-VkMappedMemoryRange-offset-00687
  offset **must** be a multiple of VkPhysicalDeviceLimits::nonCoherentAtomSize
- VUID-VkMappedMemoryRange-size-01389
  If size is equal to VK_WHOLE_SIZE, the end of the current mapping of memory **must** either be a multiple of VkPhysicalDeviceLimits::nonCoherentAtomSize bytes from the beginning of the memory object, or be equal to the end of the memory object
- VUID-VkMappedMemoryRange-size-01390
  If size is not equal to VK_WHOLE_SIZE, size **must** either be a multiple of VkPhysicalDeviceLimits::nonCoherentAtomSize, or offset plus size **must** equal the size of memory

### Valid Usage (Implicit)

- VUID-VkMappedMemoryRange-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_MAPPED_MEMORY_RANGE`
- VUID-VkMappedMemoryRange-pNext-pNext
  `pNext` **must** be `NULL`
- VUID-VkMappedMemoryRange-memory-parameter
  `memory` **must** be a valid VkDeviceMemory handle

To unmap a memory object once host access to it is no longer needed by the application, call:

```
// Provided by VK_VERSION_1_0
void vkUnmapMemory(
    VkDevice                                    device,
    VkDeviceMemory                              memory);
```

- `device` is the logical device that owns the memory.
- `memory` is the memory object to be unmapped.

Calling `vkUnmapMemory` is equivalent to calling vkUnmapMemory2KHR with an empty `pNext` chain and the flags parameter set to zero.

### Valid Usage

- VUID-vkUnmapMemory-memory-00689
  `memory` **must** be currently host mapped

### Valid Usage (Implicit)

- VUID-vkUnmapMemory-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkUnmapMemory-memory-parameter
  `memory` **must** be a valid VkDeviceMemory handle
- VUID-vkUnmapMemory-memory-parent
  `memory` **must** have been created, allocated, or retrieved from `device`

### Host Synchronization

- Host access to `memory` **must** be externally synchronized

Alternatively, to unmap a memory object once host access to it is no longer needed by the

application, call:

```
// Provided by VK_KHR_map_memory2
VkResult vkUnmapMemory2KHR(
    VkDevice                                    device,
    const VkMemoryUnmapInfoKHR*                 pMemoryUnmapInfo);
```

- `device` is the logical device that owns the memory.
- `pMemoryUnmapInfo` is a pointer to a VkMemoryUnmapInfoKHR structure describing parameters of the unmap.

This function behaves identically to vkUnmapMemory except that it gets its parameters via an extensible structure pointer rather than directly as function arguments.

### Valid Usage (Implicit)

- VUID-vkUnmapMemory2KHR-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkUnmapMemory2KHR-pMemoryUnmapInfo-parameter
  `pMemoryUnmapInfo` **must** be a valid pointer to a valid VkMemoryUnmapInfoKHR structure

### Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- `VK_ERROR_MEMORY_MAP_FAILED`

The `VkMemoryUnmapInfoKHR` structure is defined as:

```
// Provided by VK_KHR_map_memory2
typedef struct VkMemoryUnmapInfoKHR {
    VkStructureType          sType;
    const void*              pNext;
    VkMemoryUnmapFlagsKHR    flags;
    VkDeviceMemory           memory;
} VkMemoryUnmapInfoKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is a bitmask of VkMemoryUnmapFlagBitsKHR specifying additional parameters of the memory map operation.

- memory is the VkDeviceMemory object to be unmapped.

### Valid Usage

- VUID-VkMemoryUnmapInfoKHR-memory-07964
  memory **must** be currently host mapped
- VUID-VkMemoryUnmapInfoKHR-flags-09579
  If VK_MEMORY_UNMAP_RESERVE_BIT_EXT is set in flags, the memoryUnmapReserve **must** be enabled
- VUID-VkMemoryUnmapInfoKHR-flags-09580
  If VK_MEMORY_UNMAP_RESERVE_BIT_EXT is set in flags, the memory object **must** not have been imported from a handle type of VK_EXTERNAL_MEMORY_HANDLE_TYPE_HOST_ALLOCATION_BIT_EXT or VK_EXTERNAL_MEMORY_HANDLE_TYPE_HOST_MAPPED_FOREIGN_MEMORY_BIT_EXT

### Valid Usage (Implicit)

- VUID-VkMemoryUnmapInfoKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_MEMORY_UNMAP_INFO_KHR
- VUID-VkMemoryUnmapInfoKHR-pNext-pNext
  pNext **must** be NULL
- VUID-VkMemoryUnmapInfoKHR-flags-parameter
  flags **must** be a valid combination of VkMemoryUnmapFlagBitsKHR values
- VUID-VkMemoryUnmapInfoKHR-memory-parameter
  memory **must** be a valid VkDeviceMemory handle

### Host Synchronization

- Host access to memory **must** be externally synchronized

Bits which **can** be set in VkMemoryUnmapInfoKHR::flags, specifying additional properties of a memory unmap, are:

```
// Provided by VK_KHR_map_memory2
typedef enum VkMemoryUnmapFlagBitsKHR {
  // Provided by VK_EXT_map_memory_placed
    VK_MEMORY_UNMAP_RESERVE_BIT_EXT = 0x00000001,
} VkMemoryUnmapFlagBitsKHR;
```

- VK_MEMORY_UNMAP_RESERVE_BIT_EXT requests that virtual address range currently occupied by the memory map remain reserved after the vkUnmapMemory2KHR call completes. Future system memory map operations or calls to vkMapMemory or vkMapMemory2KHR will not return addresses in that range unless the range has since been unreserved by the application or the mapping is explicitly placed in that range by calling vkMapMemory2KHR with

VK_MEMORY_MAP_PLACED_BIT_EXT, or doing the system memory map equivalent. When VK_MEMORY_UNMAP_RESERVE_BIT_EXT is set, the memory unmap operation **may** fail, in which case the memory object will remain host mapped and vkUnmapMemory2KHR will return VK_ERROR_MEMORY_MAP_FAILED.

```
// Provided by VK_KHR_map_memory2
typedef VkFlags VkMemoryUnmapFlagsKHR;
```

VkMemoryUnmapFlagsKHR is a bitmask type for setting a mask of zero or more VkMemoryUnmapFlagBitsKHR.

### 11.2.16. Lazily Allocated Memory

If the memory object is allocated from a heap with the VK_MEMORY_PROPERTY_LAZILY_ALLOCATED_BIT bit set, that object's backing memory **may** be provided by the implementation lazily. The actual committed size of the memory **may** initially be as small as zero (or as large as the requested size), and monotonically increases as additional memory is needed.

A memory type with this flag set is only allowed to be bound to a VkImage whose usage flags include VK_IMAGE_USAGE_TRANSIENT_ATTACHMENT_BIT.

| | |
|---|---|
| **NOTE** | Using lazily allocated memory objects for framebuffer attachments that are not needed once a render pass instance has completed **may** allow some implementations to never allocate memory for such attachments. |

To determine the amount of lazily-allocated memory that is currently committed for a memory object, call:

```
// Provided by VK_VERSION_1_0
void vkGetDeviceMemoryCommitment(
    VkDevice                                    device,
    VkDeviceMemory                              memory,
    VkDeviceSize*                               pCommittedMemoryInBytes);
```

- device is the logical device that owns the memory.
- memory is the memory object being queried.
- pCommittedMemoryInBytes is a pointer to a VkDeviceSize value in which the number of bytes currently committed is returned, on success.

The implementation **may** update the commitment at any time, and the value returned by this query **may** be out of date.

The implementation guarantees to allocate any committed memory from the heapIndex indicated by the memory type that the memory object was created with.

### Valid Usage

- VUID-vkGetDeviceMemoryCommitment-memory-00690
  `memory` **must** have been created with a memory type that reports `VK_MEMORY_PROPERTY_LAZILY_ALLOCATED_BIT`

### Valid Usage (Implicit)

- VUID-vkGetDeviceMemoryCommitment-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkGetDeviceMemoryCommitment-memory-parameter
  `memory` **must** be a valid VkDeviceMemory handle
- VUID-vkGetDeviceMemoryCommitment-pCommittedMemoryInBytes-parameter
  `pCommittedMemoryInBytes` **must** be a valid pointer to a VkDeviceSize value
- VUID-vkGetDeviceMemoryCommitment-memory-parent
  `memory` **must** have been created, allocated, or retrieved from `device`

## 11.2.17. Protected Memory

*Protected memory* divides device memory into protected device memory and unprotected device memory.

Protected memory adds the following concepts:

- Memory:
  - Unprotected device memory, which **can** be visible to the device and **can** be visible to the host
  - Protected device memory, which **can** be visible to the device but **must** not be visible to the host
- Resources:
  - Unprotected images and unprotected buffers, to which unprotected memory **can** be bound
  - Protected images and protected buffers, to which protected memory **can** be bound
- Command buffers:
  - Unprotected command buffers, which **can** be submitted to a device queue to execute unprotected queue operations
  - Protected command buffers, which **can** be submitted to a protected-capable device queue to execute protected queue operations
- Device queues:
  - Unprotected device queues, to which unprotected command buffers **can** be submitted
  - Protected-capable device queues, to which unprotected command buffers or protected

command buffers **can** be submitted

- Queue submissions
  - Unprotected queue submissions, through which unprotected command buffers **can** be submitted
  - Protected queue submissions, through which protected command buffers **can** be submitted
- Queue operations
  - Unprotected queue operations
  - Protected queue operations

**NOTE**

When the `protectedMemory` feature is enabled, all pipelines **may** be recorded in either protected or unprotected command buffers (or both), which may incur an extra cost on some implementations. This **can** be mitigated by enabling the `pipelineProtectedAccess` feature, in which case pipelines created with `VK_PIPELINE_CREATE_PROTECTED_ACCESS_ONLY_BIT_EXT` may only be recorded in protected command buffers, and pipelines created with `VK_PIPELINE_CREATE_NO_PROTECTED_ACCESS_BIT_EXT` may only be recorded in unprotected command buffers.

**Protected Memory Access Rules**

If VkPhysicalDeviceProtectedMemoryProperties::`protectedNoFault` is `VK_FALSE`, applications **must** not perform any of the following operations:

- Write to unprotected memory within protected queue operations.
- Access protected memory within protected queue operations other than in framebuffer-space pipeline stages, the compute shader stage, or the transfer stage.
- Perform a query within protected queue operations.

If VkPhysicalDeviceProtectedMemoryProperties::`protectedNoFault` is `VK_TRUE`, these operations are valid, but reads will return undefined values, and writes will either be dropped or store undefined values.

Additionally, indirect operations **must** not be performed within protected queue operations.

Whether these operations are valid or not, or if any other invalid usage is performed, the implementation **must** guarantee that:

- Protected device memory **must** never be visible to the host.
- Values written to unprotected device memory **must** not be a function of values from protected memory.

## 11.2.18. External Memory Handle Types

### Android Hardware Buffer

Android's NDK defines AHardwareBuffer objects, which represent device memory that is shareable across processes and that **can** be accessed by a variety of media APIs and the hardware used to implement them. These Android hardware buffer objects **may** be imported into VkDeviceMemory objects for access via Vulkan, or exported from Vulkan. An VkImage or VkBuffer **can** be bound to the imported or exported VkDeviceMemory object if it is created with `VK_EXTERNAL_MEMORY_HANDLE_TYPE_ANDROID_HARDWARE_BUFFER_BIT_ANDROID`.

To remove an unnecessary compile time dependency, an incomplete type definition of AHardwareBuffer is provided in the Vulkan headers:

```
// Provided by VK_ANDROID_external_memory_android_hardware_buffer
struct AHardwareBuffer;
```

The actual AHardwareBuffer type is defined in Android NDK headers.

| | |
|---|---|
| **NOTE** | The NDK format, usage, and size/dimensions of an AHardwareBuffer object can be obtained with the `AHardwareBuffer_describe` function. While Android hardware buffers can be imported to or exported from Vulkan without using that function, valid usage and implementation behavior is defined in terms of the `AHardwareBuffer_Desc` properties it returns. |

Android hardware buffer objects are reference-counted using Android NDK functions outside of the scope of this specification. A VkDeviceMemory imported from an Android hardware buffer or that **can** be exported to an Android hardware buffer **must** acquire a reference to its AHardwareBuffer object, and **must** release this reference when the device memory is freed. During the host execution of a Vulkan command that has an Android hardware buffer as a parameter (including indirect parameters via `pNext` chains), the application **must** not decrement the Android hardware buffer's reference count to zero.

Android hardware buffers **can** be mapped and unmapped for CPU access using the NDK functions. These lock and unlock APIs are considered to acquire and release ownership of the Android hardware buffer, and applications **must** follow the rules described in External Resource Sharing to transfer ownership between the Vulkan instance and these native APIs.

Android hardware buffers **can** be shared with external APIs and Vulkan instances on the same device, and also with foreign devices. When transferring ownership of the Android hardware buffer, the external and foreign special queue families described in Queue Family Ownership Transfer are not identical. All APIs which produce or consume Android hardware buffers are considered to use foreign devices, except OpenGL ES contexts and Vulkan logical devices that have matching device and driver UUIDs. Implementations **may** treat a transfer to or from the foreign queue family as if it were a transfer to or from the external queue family when the Android hardware buffer's usage only permits it to be used on the same physical device.

#### Android Hardware Buffer Optimal Usages

Vulkan buffer and image usage flags do not correspond exactly to Android hardware buffer usage

flags. When allocating Android hardware buffers with non-Vulkan APIs, if any AHARDWAREBUFFER_USAGE_GPU_* usage bits are included, by default the allocator **must** allocate the memory in such a way that it supports Vulkan usages and creation flags in the usage equivalence table which do not have Android hardware buffer equivalents.

An VkAndroidHardwareBufferUsageANDROID structure **can** be included in the pNext chain of a VkImageFormatProperties2 structure passed to vkGetPhysicalDeviceImageFormatProperties2 to obtain optimal Android hardware buffer usage flags for specific Vulkan resource creation parameters. Some usage flags returned by these commands are **required** based on the input parameters, but additional vendor-specific usage flags (AHARDWAREBUFFER_USAGE_VENDOR_*) **may** also be returned. Any Android hardware buffer allocated with these vendor-specific usage flags and imported to Vulkan **must** only be bound to resources created with parameters that are a subset of the parameters used to obtain the Android hardware buffer usage, since the memory **may** have been allocated in a way incompatible with other parameters. If an Android hardware buffer is successfully allocated with additional non-vendor-specific usage flags in addition to the recommended usage, it **must** support being used in the same ways as an Android hardware buffer allocated with only the recommended usage, and also in ways indicated by the additional usage.

#### Android Hardware Buffer External Formats

Android hardware buffers **may** represent images using implementation-specific formats, layouts, color models, etc., which do not have Vulkan equivalents. Such *external formats* are commonly used by external image sources such as video decoders or cameras. Vulkan **can** import Android hardware buffers that have external formats, but since the image contents are in a possibly proprietary representation, images with external formats **must** have optimal tiling, and their use is restricted. Images with external formats **must** only be sampled with a sampler that has $Y'C_BC_R$ conversion enabled.

Images that will be backed by an Android hardware buffer **can** use an external format by setting VkImageCreateInfo::format to VK_FORMAT_UNDEFINED and including a VkExternalFormatANDROID structure in the pNext chain. Images **can** be created with an external format even if the Android hardware buffer has a format which has an equivalent Vulkan format to enable consistent handling of images from sources that might use either category of format. However, all images created with an external format are subject to the valid usage requirements associated with external formats, even if the Android hardware buffer's format has a Vulkan equivalent. The external format of an Android hardware buffer **can** be obtained by passing a VkAndroidHardwareBufferFormatPropertiesANDROID structure to vkGetAndroidHardwareBufferPropertiesANDROID.

#### Android Hardware Buffer Image Resources

Android hardware buffers have intrinsic width, height, format, and usage properties, so Vulkan images bound to memory imported from an Android hardware buffer **must** use dedicated allocations: VkMemoryDedicatedRequirements::requiresDedicatedAllocation **must** be VK_TRUE for images created with VkExternalMemoryImageCreateInfo::handleTypes that includes VK_EXTERNAL_MEMORY_HANDLE_TYPE_ANDROID_HARDWARE_BUFFER_BIT_ANDROID. When creating an image that will be bound to an imported Android hardware buffer, the image creation parameters **must** be equivalent to the AHardwareBuffer properties as described by the valid usage of VkMemoryAllocateInfo. Similarly, device memory allocated for a dedicated image **must** not be

exported to an Android hardware buffer until it has been bound to that image, and the implementation **must** return an Android hardware buffer with properties derived from the image:

- The `width` and `height` members of `AHardwareBuffer_Desc` **must** be the same as the `width` and `height` members of VkImageCreateInfo::`extent`, respectively.
- The `layers` member of `AHardwareBuffer_Desc` **must** be the same as the `arrayLayers` member of VkImageCreateInfo.
- The `format` member of `AHardwareBuffer_Desc` **must** be equivalent to VkImageCreateInfo::`format` as defined by AHardwareBuffer Format Equivalence.
- The `usage` member of `AHardwareBuffer_Desc` **must** include bits corresponding to bits included in VkImageCreateInfo::`usage` and VkImageCreateInfo::`flags` where such a correspondence exists according to AHardwareBuffer Usage Equivalence. It **may** also include additional usage bits, including vendor-specific usages. Presence of vendor usage bits **may** make the Android hardware buffer only usable in ways indicated by the image creation parameters, even when used outside Vulkan, in a similar way that allocating the Android hardware buffer with usage returned in VkAndroidHardwareBufferUsageANDROID does.

Implementations **may** support fewer combinations of image creation parameters for images with Android hardware buffer external handle type than for non-external images. Support for a given set of parameters **can** be determined by passing VkExternalImageFormatProperties to vkGetPhysicalDeviceImageFormatProperties2 with `handleType` set to `VK_EXTERNAL_MEMORY_HANDLE_TYPE_ANDROID_HARDWARE_BUFFER_BIT_ANDROID`. Any Android hardware buffer successfully allocated outside Vulkan with usage that includes `AHARDWAREBUFFER_USAGE_GPU_*` **must** be supported when using equivalent Vulkan image parameters. If a given choice of image parameters are supported for import, they **can** also be used to create an image and memory that will be exported to an Android hardware buffer.

*Table 13. AHardwareBuffer Format Equivalence*

| AHardwareBuffer Format | Vulkan Format |
|---|---|
| `AHARDWAREBUFFER_FORMAT_R8G8B8A8_UNORM` | `VK_FORMAT_R8G8B8A8_UNORM` |
| `AHARDWAREBUFFER_FORMAT_R8G8B8X8_UNORM` [1] | `VK_FORMAT_R8G8B8A8_UNORM` |
| `AHARDWAREBUFFER_FORMAT_R8G8B8_UNORM` | `VK_FORMAT_R8G8B8_UNORM` |
| `AHARDWAREBUFFER_FORMAT_R5G6B5_UNORM` | `VK_FORMAT_R5G6B5_UNORM_PACK16` |
| `AHARDWAREBUFFER_FORMAT_R16G16B16A16_FLOAT` | `VK_FORMAT_R16G16B16A16_SFLOAT` |
| `AHARDWAREBUFFER_FORMAT_R10G10B10A2_UNORM` | `VK_FORMAT_A2B10G10R10_UNORM_PACK32` |
| `AHARDWAREBUFFER_FORMAT_D16_UNORM` | `VK_FORMAT_D16_UNORM` |
| `AHARDWAREBUFFER_FORMAT_D24_UNORM` | `VK_FORMAT_X8_D24_UNORM_PACK32` |
| `AHARDWAREBUFFER_FORMAT_D24_UNORM_S8_UINT` | `VK_FORMAT_D24_UNORM_S8_UINT` |
| `AHARDWAREBUFFER_FORMAT_D32_FLOAT` | `VK_FORMAT_D32_SFLOAT` |
| `AHARDWAREBUFFER_FORMAT_D32_FLOAT_S8_UINT` | `VK_FORMAT_D32_SFLOAT_S8_UINT` |
| `AHARDWAREBUFFER_FORMAT_S8_UINT` | `VK_FORMAT_S8_UINT` |

*Table 14. AHardwareBuffer Usage Equivalence*

| AHardwareBuffer Usage | Vulkan Usage or Creation Flag |
|---|---|
| None | VK_IMAGE_USAGE_TRANSFER_SRC_BIT |
| None | VK_IMAGE_USAGE_TRANSFER_DST_BIT |
| AHARDWAREBUFFER_USAGE_GPU_SAMPLED_IMAGE | VK_IMAGE_USAGE_SAMPLED_BIT |
| AHARDWAREBUFFER_USAGE_GPU_SAMPLED_IMAGE | VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT |
| AHARDWAREBUFFER_USAGE_GPU_FRAMEBUFFER [3] | VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT |
| AHARDWAREBUFFER_USAGE_GPU_FRAMEBUFFER [3] | VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT |
| AHARDWAREBUFFER_USAGE_GPU_CUBE_MAP | VK_IMAGE_CREATE_CUBE_COMPATIBLE_BIT |
| AHARDWAREBUFFER_USAGE_GPU_MIPMAP_COMPLETE | None [2] |
| AHARDWAREBUFFER_USAGE_PROTECTED_CONTENT | VK_IMAGE_CREATE_PROTECTED_BIT |
| None | VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT |
| None | VK_IMAGE_CREATE_EXTENDED_USAGE_BIT |
| AHARDWAREBUFFER_USAGE_GPU_DATA_BUFFER [4] | VK_IMAGE_USAGE_STORAGE_BIT |

**1**

Vulkan does not differentiate between AHARDWAREBUFFER_FORMAT_R8G8B8A8_UNORM and AHARDWAREBUFFER_FORMAT_R8G8B8X8_UNORM: they both behave as VK_FORMAT_R8G8B8A8_UNORM. After an external entity writes to a AHARDWAREBUFFER_FORMAT_R8G8B8X8_UNORM Android hardware buffer, the values read by Vulkan from the X/A component are undefined. To emulate the traditional behavior of the X component during sampling or blending, applications **should** use VK_COMPONENT_SWIZZLE_ONE in image view component mappings and VK_BLEND_FACTOR_ONE in color blend factors. There is no way to avoid copying these undefined values when copying from such an image to another image or buffer.

**2**

The AHARDWAREBUFFER_USAGE_GPU_MIPMAP_COMPLETE flag does not correspond to a Vulkan image usage or creation flag. Instead, its presence indicates that the Android hardware buffer contains a complete mipmap chain, and its absence indicates that the Android hardware buffer contains only a single mip level.

**3**

Only image usages valid for the format are valid. It would be invalid to take a Android Hardware Buffer with a format of AHARDWAREBUFFER_FORMAT_R8G8B8A8_UNORM that has a AHARDWAREBUFFER_USAGE_GPU_FRAMEBUFFER usage and try to create an image with VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT.

**4**

In combination with a hardware buffer format other than BLOB.

**NOTE**

When using VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT with Android hardware buffer images, applications **should** use VkImageFormatListCreateInfo to inform the implementation which view formats will be used with the image. For some common sets of format, this allows some implementations to provide significantly better

performance when accessing the image via Vulkan.

#### Android Hardware Buffer Resources

Android hardware buffers with a format of `AHARDWAREBUFFER_FORMAT_BLOB` and usage that includes `AHARDWAREBUFFER_USAGE_GPU_DATA_BUFFER` **can** be used as the backing store for VkBuffer objects. Such Android hardware buffers have a size in bytes specified by their `width`; `height` and `layers` are both `1`.

Unlike images, buffer resources backed by Android hardware buffers do not require dedicated allocations.

Exported AHardwareBuffer objects that do not have dedicated images **must** have a format of `AHARDWAREBUFFER_FORMAT_BLOB`, usage **must** include `AHARDWAREBUFFER_USAGE_GPU_DATA_BUFFER`, `width` **must** equal the device memory allocation size, and `height` and `layers` **must** be `1`.

### QNX Screen Buffer

The QNX SDP defines `_screen_buffer` objects, which represent a buffer that the QNX Screen graphics subsystem can use directly in its windowing system APIs. More specifically, a Screen buffer is an area of memory that stores pixel data. It can be attached to Screen windows, streams, or pixmaps. These QNX Screen buffer objects **may** be imported into VkDeviceMemory objects for access via Vulkan. An VkImage or VkBuffer **can** be bound to the imported VkDeviceMemory object if it is created with `VK_EXTERNAL_MEMORY_HANDLE_TYPE_SCREEN_BUFFER_BIT_QNX`.

`struct _screen_buffer` is strongly typed, so naming the handle type is redundant. The internal layout and therefore size of a `struct _screen_buffer` image may depend on native usage flags that do not have corresponding Vulkan counterparts.

#### QNX Screen Buffer Validity

The design of Screen in the QNX SDP makes it difficult to determine the validity of objects from outside of Screen. Therefore, applications **must** ensure that QNX Screen buffer objects provided used in various Vulkan interfaces are ones created explicitly with QNX Screen APIs. See QNX SDP documentation for more information.

A VkDeviceMemory imported from a QNX Screen buffer has no way to acquire a reference to its `_screen_buffer` object. Therefore, during the host execution of a Vulkan command that has a QNX Screen buffer as a parameter (including indirect parameters via `pNext` chains), the application **must** ensure that the QNX Screen buffer resource remains valid.

Generally, for a `_screen_buffer` object to be valid for use within a Vulkan implementation, the buffer object **should** have a `_screen_buffer`::`SCREEN_PROPERTY_USAGE` that includes at least one of: `SCREEN_USAGE_VULKAN`, `SCREEN_USAGE_OPENGL_ES2`, `SCREEN_USAGE_OPENGL_ES3`, or `SCREEN_USAGE_NATIVE`. The exact Screen-native usage flags required depends on the Vulkan implementation, and QNX Screen itself will not necessarily enforce these requirements. Note that Screen-native usage flags are in no way related to usage flags in the Vulkan specification.

#### QNX Screen Buffer External Formats

QNX Screen buffers **may** represent images using implementation-specific formats, layouts, color

models, etc., which do not have Vulkan equivalents. Such *external formats* are commonly used by external image sources such as video decoders or cameras. Vulkan **can** import QNX Screen buffers that have external formats, but since the image contents are in an undiscoverable and possibly proprietary representation, images with external formats **must** only be used as sampled images, **must** only be sampled with a sampler that has $Y'C_BC_R$ conversion enabled, and **must** have optimal tiling.

Images that will be backed by a QNX Screen buffer **can** use an external format by setting VkImageCreateInfo::`format` to `VK_FORMAT_UNDEFINED` and including a VkExternalFormatQNX structure in the `pNext` chain. Images **can** be created with an external format even if the QNX Screen buffer has a format which has an equivalent Vulkan format to enable consistent handling of images from sources that might use either category of format. The external format of a QNX Screen buffer **can** be obtained by passing a VkScreenBufferFormatPropertiesQNX structure to vkGetScreenBufferPropertiesQNX.

#### QNX Screen Buffer Image Resources

QNX Screen buffers have intrinsic width, height, format, and usage properties, so Vulkan images bound to memory imported from a QNX Screen buffer **must** use dedicated allocations: `VkMemoryDedicatedRequirements`::`requiresDedicatedAllocation` **must** be `VK_TRUE` for images created with VkExternalMemoryImageCreateInfo::`handleTypes` that includes `VK_EXTERNAL_MEMORY_HANDLE_TYPE_SCREEN_BUFFER_BIT_QNX`. When creating an image that will be bound to an imported QNX Screen buffer, the image creation parameters **must** be equivalent to the `_screen_buffer` properties as described by the valid usage of VkMemoryAllocateInfo.

*Table 15. QNX Screen Buffer Format Equivalence*

| QNX Screen Format | Vulkan Format |
|---|---|
| `SCREEN_FORMAT_RGBA8888` | `VK_FORMAT_B8G8R8A8_UNORM` |
| `SCREEN_FORMAT_RGBX8888` [1] | `VK_FORMAT_B8G8R8A8_UNORM` |
| `SCREEN_FORMAT_BGRA8888` | `VK_FORMAT_R8G8B8A8_UNORM` |
| `SCREEN_FORMAT_BGRX8888` [1] | `VK_FORMAT_R8G8B8A8_UNORM` |
| `SCREEN_FORMAT_RGBA1010102` | `VK_FORMAT_A2R10G10B10_UNORM_PACK32` |
| `SCREEN_FORMAT_RGBX1010102` [1] | `VK_FORMAT_A2R10G10B10_UNORM_PACK32` |
| `SCREEN_FORMAT_BGRA1010102` | `VK_FORMAT_A2B10G10R10_UNORM_PACK32` |
| `SCREEN_FORMAT_BGRX1010102` [1] | `VK_FORMAT_A2B10G10R10_UNORM_PACK32` |
| `SCREEN_FORMAT_RGBA5551` | `VK_FORMAT_A1R5G5B5_UNORM_PACK16` |
| `SCREEN_FORMAT_RGBX5551` [1] | `VK_FORMAT_A1R5G5B5_UNORM_PACK16` |
| `SCREEN_FORMAT_RGB565` | `VK_FORMAT_R5G6B5_UNORM_PACK16` |
| `SCREEN_FORMAT_RGB888` | `VK_FORMAT_R8G8B8_UNORM` |

**1**

Vulkan does not differentiate between `SCREEN_FORMAT_RGBA8888` and `SCREEN_FORMAT_RGBX8888`: they both behave as `VK_FORMAT_R8G8B8A8_UNORM`. After an external entity writes to a `SCREEN_FORMAT_RGBX8888` QNX Screen buffer, the values read by Vulkan from the X/A component are undefined. To emulate the traditional behavior of the X component during sampling or

blending, applications **should** use VK_COMPONENT_SWIZZLE_ONE in image view component mappings and VK_BLEND_FACTOR_ONE in color blend factors. There is no way to avoid copying these undefined values when copying from such an image to another image or buffer. The same behavior applies to the following pairs: SCREEN_FORMAT_BGRA8888 and SCREEN_FORMAT_BGRX8888, SCREEN_FORMAT_RGBA1010102 and SCREEN_FORMAT_RGBX1010102, SCREEN_FORMAT_BGRA1010102 and SCREEN_FORMAT_BGRX1010102, SCREEN_FORMAT_RGBA5551 and SCREEN_FORMAT_RGBX5551

### 11.2.19. Peer Memory Features

*Peer memory* is memory that is allocated for a given physical device and then bound to a resource and accessed by a different physical device, in a logical device that represents multiple physical devices. Some ways of reading and writing peer memory **may** not be supported by a device.

To determine how peer memory **can** be accessed, call:

```
// Provided by VK_VERSION_1_1
void vkGetDeviceGroupPeerMemoryFeatures(
    VkDevice                                    device,
    uint32_t                                    heapIndex,
    uint32_t                                    localDeviceIndex,
    uint32_t                                    remoteDeviceIndex,
    VkPeerMemoryFeatureFlags*                   pPeerMemoryFeatures);
```

or the equivalent command

```
// Provided by VK_KHR_device_group
void vkGetDeviceGroupPeerMemoryFeaturesKHR(
    VkDevice                                    device,
    uint32_t                                    heapIndex,
    uint32_t                                    localDeviceIndex,
    uint32_t                                    remoteDeviceIndex,
    VkPeerMemoryFeatureFlags*                   pPeerMemoryFeatures);
```

- device is the logical device that owns the memory.
- heapIndex is the index of the memory heap from which the memory is allocated.
- localDeviceIndex is the device index of the physical device that performs the memory access.
- remoteDeviceIndex is the device index of the physical device that the memory is allocated for.
- pPeerMemoryFeatures is a pointer to a VkPeerMemoryFeatureFlags bitmask indicating which types of memory accesses are supported for the combination of heap, local, and remote devices.

#### Valid Usage

- VUID-vkGetDeviceGroupPeerMemoryFeatures-heapIndex-00691
  heapIndex **must** be less than memoryHeapCount
- VUID-vkGetDeviceGroupPeerMemoryFeatures-localDeviceIndex-00692

localDeviceIndex **must** be a valid device index

- VUID-vkGetDeviceGroupPeerMemoryFeatures-remoteDeviceIndex-00693
  remoteDeviceIndex **must** be a valid device index
- VUID-vkGetDeviceGroupPeerMemoryFeatures-localDeviceIndex-00694
  localDeviceIndex **must** not equal remoteDeviceIndex

### Valid Usage (Implicit)

- VUID-vkGetDeviceGroupPeerMemoryFeatures-device-parameter
  device **must** be a valid VkDevice handle
- VUID-vkGetDeviceGroupPeerMemoryFeatures-pPeerMemoryFeatures-parameter
  pPeerMemoryFeatures **must** be a valid pointer to a VkPeerMemoryFeatureFlags value

Bits which **may** be set in vkGetDeviceGroupPeerMemoryFeatures::pPeerMemoryFeatures, indicating supported peer memory features, are:

```
// Provided by VK_VERSION_1_1
typedef enum VkPeerMemoryFeatureFlagBits {
    VK_PEER_MEMORY_FEATURE_COPY_SRC_BIT = 0x00000001,
    VK_PEER_MEMORY_FEATURE_COPY_DST_BIT = 0x00000002,
    VK_PEER_MEMORY_FEATURE_GENERIC_SRC_BIT = 0x00000004,
    VK_PEER_MEMORY_FEATURE_GENERIC_DST_BIT = 0x00000008,
  // Provided by VK_KHR_device_group
    VK_PEER_MEMORY_FEATURE_COPY_SRC_BIT_KHR = VK_PEER_MEMORY_FEATURE_COPY_SRC_BIT,
  // Provided by VK_KHR_device_group
    VK_PEER_MEMORY_FEATURE_COPY_DST_BIT_KHR = VK_PEER_MEMORY_FEATURE_COPY_DST_BIT,
  // Provided by VK_KHR_device_group
    VK_PEER_MEMORY_FEATURE_GENERIC_SRC_BIT_KHR =
VK_PEER_MEMORY_FEATURE_GENERIC_SRC_BIT,
  // Provided by VK_KHR_device_group
    VK_PEER_MEMORY_FEATURE_GENERIC_DST_BIT_KHR =
VK_PEER_MEMORY_FEATURE_GENERIC_DST_BIT,
} VkPeerMemoryFeatureFlagBits;
```

or the equivalent

```
// Provided by VK_KHR_device_group
typedef VkPeerMemoryFeatureFlagBits VkPeerMemoryFeatureFlagBitsKHR;
```

- VK_PEER_MEMORY_FEATURE_COPY_SRC_BIT specifies that the memory **can** be accessed as the source of any vkCmdCopy* command.
- VK_PEER_MEMORY_FEATURE_COPY_DST_BIT specifies that the memory **can** be accessed as the destination of any vkCmdCopy* command.

- VK_PEER_MEMORY_FEATURE_GENERIC_SRC_BIT specifies that the memory **can** be read as any memory access type.
- VK_PEER_MEMORY_FEATURE_GENERIC_DST_BIT specifies that the memory **can** be written as any memory access type. Shader atomics are considered to be writes.

| | |
|---|---|
| **NOTE** | The peer memory features of a memory heap also apply to any accesses that **may** be performed during image layout transitions. |

VK_PEER_MEMORY_FEATURE_COPY_DST_BIT **must** be supported for all host local heaps and for at least one device-local memory heap.

If a device does not support a peer memory feature, it is still valid to use a resource that includes both local and peer memory bindings with the corresponding access type as long as only the local bindings are actually accessed. For example, an application doing split-frame rendering would use framebuffer attachments that include both local and peer memory bindings, but would scissor the rendering to only update local memory.

```
// Provided by VK_VERSION_1_1
typedef VkFlags VkPeerMemoryFeatureFlags;
```

or the equivalent

```
// Provided by VK_KHR_device_group
typedef VkPeerMemoryFeatureFlags VkPeerMemoryFeatureFlagsKHR;
```

VkPeerMemoryFeatureFlags is a bitmask type for setting a mask of zero or more VkPeerMemoryFeatureFlagBits.

### 11.2.20. Opaque Capture Address Query

To query a 64-bit opaque capture address value from a memory object, call:

```
// Provided by VK_KHR_buffer_device_address
uint64_t vkGetDeviceMemoryOpaqueCaptureAddressKHR(
    VkDevice                                    device,
    const VkDeviceMemoryOpaqueCaptureAddressInfo* pInfo);
```

- device is the logical device that the memory object was allocated on.
- pInfo is a pointer to a VkDeviceMemoryOpaqueCaptureAddressInfo structure specifying the memory object to retrieve an address for.

The 64-bit return value is an opaque address representing the start of pInfo->memory.

If the memory object was allocated with a non-zero value of VkMemoryOpaqueCaptureAddressAllocateInfo::opaqueCaptureAddress, the return value **must** be the

same address.

| NOTE | The expected usage for these opaque addresses is only for trace capture/replay tools to store these addresses in a trace and subsequently specify them during replay. |
|---|---|

### Valid Usage

- VUID-vkGetDeviceMemoryOpaqueCaptureAddress-None-03334
  The `bufferDeviceAddress` feature **must** be enabled
- VUID-vkGetDeviceMemoryOpaqueCaptureAddress-device-03335
  If `device` was created with multiple physical devices, then the `bufferDeviceAddressMultiDevice` feature **must** be enabled

### Valid Usage (Implicit)

- VUID-vkGetDeviceMemoryOpaqueCaptureAddress-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkGetDeviceMemoryOpaqueCaptureAddress-pInfo-parameter
  `pInfo` **must** be a valid pointer to a valid VkDeviceMemoryOpaqueCaptureAddressInfo structure

The `VkDeviceMemoryOpaqueCaptureAddressInfo` structure is defined as:

```
typedef struct VkDeviceMemoryOpaqueCaptureAddressInfo {
    VkStructureType    sType;
    const void*        pNext;
    VkDeviceMemory     memory;
} VkDeviceMemoryOpaqueCaptureAddressInfo;
```

or the equivalent

```
// Provided by VK_KHR_buffer_device_address
typedef VkDeviceMemoryOpaqueCaptureAddressInfo
VkDeviceMemoryOpaqueCaptureAddressInfoKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `memory` specifies the memory whose address is being queried.

### Valid Usage

- VUID-VkDeviceMemoryOpaqueCaptureAddressInfo-memory-03336

`memory` **must** have been allocated with `VK_MEMORY_ALLOCATE_DEVICE_ADDRESS_BIT`

### Valid Usage (Implicit)

- VUID-VkDeviceMemoryOpaqueCaptureAddressInfo-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_DEVICE_MEMORY_OPAQUE_CAPTURE_ADDRESS_INFO`
- VUID-VkDeviceMemoryOpaqueCaptureAddressInfo-pNext-pNext
  `pNext` **must** be `NULL`
- VUID-VkDeviceMemoryOpaqueCaptureAddressInfo-memory-parameter
  `memory` **must** be a valid VkDeviceMemory handle

# Chapter 12. Resource Creation

Vulkan supports two primary resource types: *buffers* and *images*. Resources are views of memory with associated formatting and dimensionality. Buffers provide access to raw arrays of bytes, whereas images **can** be multidimensional and **may** have associated metadata.

Other resource types, such as acceleration structures and micromaps use buffers as the backing store for opaque data structures.

## 12.1. Buffers

Buffers represent linear arrays of data which are used for various purposes by binding them to a graphics or compute pipeline via descriptor sets or certain commands, or by directly specifying them as parameters to certain commands.

Buffers are represented by `VkBuffer` handles:

```
// Provided by VK_VERSION_1_0
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkBuffer)
```

To create buffers, call:

```
// Provided by VK_VERSION_1_0
VkResult vkCreateBuffer(
    VkDevice                                    device,
    const VkBufferCreateInfo*                   pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkBuffer*                                   pBuffer);
```

- `device` is the logical device that creates the buffer object.
- `pCreateInfo` is a pointer to a VkBufferCreateInfo structure containing parameters affecting creation of the buffer.
- `pAllocator` controls host memory allocation as described in the Memory Allocation chapter.
- `pBuffer` is a pointer to a VkBuffer handle in which the resulting buffer object is returned.

### Valid Usage

- VUID-vkCreateBuffer-device-09664
  `device` **must** support at least one queue family with one of the `VK_QUEUE_VIDEO_ENCODE_BIT_KHR`, `VK_QUEUE_VIDEO_DECODE_BIT_KHR`, `VK_QUEUE_SPARSE_BINDING_BIT`, `VK_QUEUE_TRANSFER_BIT`, `VK_QUEUE_COMPUTE_BIT`, or `VK_QUEUE_GRAPHICS_BIT` capabilities
- VUID-vkCreateBuffer-flags-00911
  If the `flags` member of `pCreateInfo` includes `VK_BUFFER_CREATE_SPARSE_BINDING_BIT`, and the

extendedSparseAddressSpace feature is not enabled, creating this VkBuffer **must** not cause the total required sparse memory for all currently valid sparse resources on the device to exceed VkPhysicalDeviceLimits::sparseAddressSpaceSize

- VUID-vkCreateBuffer-flags-09383
  If the flags member of pCreateInfo includes VK_BUFFER_CREATE_SPARSE_BINDING_BIT, the extendedSparseAddressSpace feature is enabled, and the usage member of pCreateInfo contains bits not in VkPhysicalDeviceExtendedSparseAddressSpacePropertiesNV::extendedSparseBufferUsageFlags, creating this VkBuffer **must** not cause the total required sparse memory for all currently valid sparse resources on the device, excluding VkBuffer created with usage member of pCreateInfo containing bits in VkPhysicalDeviceExtendedSparseAddressSpacePropertiesNV::extendedSparseBufferUsageFlags and VkImage created with usage member of pCreateInfo containing bits in VkPhysicalDeviceExtendedSparseAddressSpacePropertiesNV::extendedSparseImageUsageFlags, to exceed VkPhysicalDeviceLimits::sparseAddressSpaceSize
- VUID-vkCreateBuffer-flags-09384
  If the flags member of pCreateInfo includes VK_BUFFER_CREATE_SPARSE_BINDING_BIT and the extendedSparseAddressSpace feature is enabled, creating this VkBuffer **must** not cause the total required sparse memory for all currently valid sparse resources on the device to exceed VkPhysicalDeviceExtendedSparseAddressSpacePropertiesNV::extendedSparseAddressSpaceSize
- VUID-vkCreateBuffer-pNext-06387
  If using the VkBuffer for an import operation from a VkBufferCollectionFUCHSIA where a VkBufferCollectionBufferCreateInfoFUCHSIA has been chained to pNext, pCreateInfo **must** match the VkBufferConstraintsInfoFUCHSIA::createInfo used when setting the constraints on the buffer collection with vkSetBufferCollectionBufferConstraintsFUCHSIA

## Valid Usage (Implicit)

- VUID-vkCreateBuffer-device-parameter
  device **must** be a valid VkDevice handle
- VUID-vkCreateBuffer-pCreateInfo-parameter
  pCreateInfo **must** be a valid pointer to a valid VkBufferCreateInfo structure
- VUID-vkCreateBuffer-pAllocator-parameter
  If pAllocator is not NULL, pAllocator **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkCreateBuffer-pBuffer-parameter
  pBuffer **must** be a valid pointer to a VkBuffer handle

## Return Codes

**Success**

- VK_SUCCESS

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY
- VK_ERROR_INVALID_OPAQUE_CAPTURE_ADDRESS_KHR

The VkBufferCreateInfo structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkBufferCreateInfo {
    VkStructureType        sType;
    const void*            pNext;
    VkBufferCreateFlags    flags;
    VkDeviceSize           size;
    VkBufferUsageFlags     usage;
    VkSharingMode          sharingMode;
    uint32_t               queueFamilyIndexCount;
    const uint32_t*        pQueueFamilyIndices;
} VkBufferCreateInfo;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- flags is a bitmask of VkBufferCreateFlagBits specifying additional parameters of the buffer.
- size is the size in bytes of the buffer to be created.
- usage is a bitmask of VkBufferUsageFlagBits specifying allowed usages of the buffer.
- sharingMode is a VkSharingMode value specifying the sharing mode of the buffer when it will be accessed by multiple queue families.
- queueFamilyIndexCount is the number of entries in the pQueueFamilyIndices array.
- pQueueFamilyIndices is a pointer to an array of queue families that will access this buffer. It is ignored if sharingMode is not VK_SHARING_MODE_CONCURRENT.

If the pNext chain includes a VkBufferUsageFlags2CreateInfoKHR structure, VkBufferUsageFlags2CreateInfoKHR::usage from that structure is used instead of usage from this structure.

## Valid Usage

- VUID-VkBufferCreateInfo-None-09499
  If the pNext chain does not include a VkBufferUsageFlags2CreateInfoKHR structure, usage **must** be a valid combination of VkBufferUsageFlagBits values
- VUID-VkBufferCreateInfo-None-09500
  If the pNext chain does not include a VkBufferUsageFlags2CreateInfoKHR structure, usage **must** not be 0

- VUID-VkBufferCreateInfo-size-00912
  `size` **must** be greater than `0`
- VUID-VkBufferCreateInfo-sharingMode-00913
  If `sharingMode` is `VK_SHARING_MODE_CONCURRENT`, `pQueueFamilyIndices` **must** be a valid pointer to an array of `queueFamilyIndexCount` `uint32_t` values
- VUID-VkBufferCreateInfo-sharingMode-00914
  If `sharingMode` is `VK_SHARING_MODE_CONCURRENT`, `queueFamilyIndexCount` **must** be greater than `1`
- VUID-VkBufferCreateInfo-sharingMode-01419
  If `sharingMode` is `VK_SHARING_MODE_CONCURRENT`, each element of `pQueueFamilyIndices` **must** be unique and **must** be less than `pQueueFamilyPropertyCount` returned by either vkGetPhysicalDeviceQueueFamilyProperties2 or vkGetPhysicalDeviceQueueFamilyProperties for the `physicalDevice` that was used to create `device`
- VUID-VkBufferCreateInfo-flags-00915
  If the `sparseBinding` feature is not enabled, `flags` **must** not contain `VK_BUFFER_CREATE_SPARSE_BINDING_BIT`
- VUID-VkBufferCreateInfo-flags-00916
  If the `sparseResidencyBuffer` feature is not enabled, `flags` **must** not contain `VK_BUFFER_CREATE_SPARSE_RESIDENCY_BIT`
- VUID-VkBufferCreateInfo-flags-00917
  If the `sparseResidencyAliased` feature is not enabled, `flags` **must** not contain `VK_BUFFER_CREATE_SPARSE_ALIASED_BIT`
- VUID-VkBufferCreateInfo-flags-00918
  If `flags` contains `VK_BUFFER_CREATE_SPARSE_RESIDENCY_BIT` or `VK_BUFFER_CREATE_SPARSE_ALIASED_BIT`, it **must** also contain `VK_BUFFER_CREATE_SPARSE_BINDING_BIT`
- VUID-VkBufferCreateInfo-pNext-00920
  If the `pNext` chain includes a VkExternalMemoryBufferCreateInfo structure, its `handleTypes` member **must** only contain bits that are also in VkExternalBufferProperties::`externalMemoryProperties.compatibleHandleTypes`, as returned by vkGetPhysicalDeviceExternalBufferProperties with `pExternalBufferInfo->handleType` equal to any one of the handle types specified in VkExternalMemoryBufferCreateInfo::`handleTypes`
- VUID-VkBufferCreateInfo-flags-01887
  If the `protectedMemory` feature is not enabled, `flags` **must** not contain `VK_BUFFER_CREATE_PROTECTED_BIT`
- VUID-VkBufferCreateInfo-None-01888
  If any of the bits `VK_BUFFER_CREATE_SPARSE_BINDING_BIT`, `VK_BUFFER_CREATE_SPARSE_RESIDENCY_BIT`, or `VK_BUFFER_CREATE_SPARSE_ALIASED_BIT` are set, `VK_BUFFER_CREATE_PROTECTED_BIT` **must** not also be set
- VUID-VkBufferCreateInfo-pNext-01571
  If the `pNext` chain includes a VkDedicatedAllocationBufferCreateInfoNV structure, and the

dedicatedAllocation member of the chained structure is VK_TRUE, then flags **must** not include VK_BUFFER_CREATE_SPARSE_BINDING_BIT, VK_BUFFER_CREATE_SPARSE_RESIDENCY_BIT, or VK_BUFFER_CREATE_SPARSE_ALIASED_BIT

- VUID-VkBufferCreateInfo-deviceAddress-02604
  If VkBufferDeviceAddressCreateInfoEXT::deviceAddress is not zero, flags **must** include VK_BUFFER_CREATE_DEVICE_ADDRESS_CAPTURE_REPLAY_BIT
- VUID-VkBufferCreateInfo-opaqueCaptureAddress-03337
  If VkBufferOpaqueCaptureAddressCreateInfo::opaqueCaptureAddress is not zero, flags **must** include VK_BUFFER_CREATE_DEVICE_ADDRESS_CAPTURE_REPLAY_BIT
- VUID-VkBufferCreateInfo-flags-03338
  If flags includes VK_BUFFER_CREATE_DEVICE_ADDRESS_CAPTURE_REPLAY_BIT, the bufferDeviceAddressCaptureReplay feature **must** be enabled
- VUID-VkBufferCreateInfo-usage-04813
  If usage includes VK_BUFFER_USAGE_VIDEO_DECODE_SRC_BIT_KHR or VK_BUFFER_USAGE_VIDEO_DECODE_DST_BIT_KHR, and flags does not include VK_BUFFER_CREATE_VIDEO_PROFILE_INDEPENDENT_BIT_KHR, then the pNext chain **must** include a VkVideoProfileListInfoKHR structure with profileCount greater than 0 and pProfiles including at least one VkVideoProfileInfoKHR structure with a videoCodecOperation member specifying a decode operation
- VUID-VkBufferCreateInfo-usage-04814
  If usage includes VK_BUFFER_USAGE_VIDEO_ENCODE_SRC_BIT_KHR or VK_BUFFER_USAGE_VIDEO_ENCODE_DST_BIT_KHR, and flags does not include VK_BUFFER_CREATE_VIDEO_PROFILE_INDEPENDENT_BIT_KHR, then the pNext chain **must** include a VkVideoProfileListInfoKHR structure with profileCount greater than 0 and pProfiles including at least one VkVideoProfileInfoKHR structure with a videoCodecOperation member specifying an encode operation
- VUID-VkBufferCreateInfo-flags-08325
  If flags includes VK_BUFFER_CREATE_VIDEO_PROFILE_INDEPENDENT_BIT_KHR, then videoMaintenance1 **must** be enabled
- VUID-VkBufferCreateInfo-size-06409
  size **must** be less than or equal to VkPhysicalDeviceMaintenance4Properties::maxBufferSize
- VUID-VkBufferCreateInfo-usage-08097
  If usage includes VK_BUFFER_USAGE_SAMPLER_DESCRIPTOR_BUFFER_BIT_EXT, creating this VkBuffer **must** not cause the total required space for all currently valid buffers using this flag on the device to exceed VkPhysicalDeviceDescriptorBufferPropertiesEXT::samplerDescriptorBufferAddressSpaceSize or VkPhysicalDeviceDescriptorBufferPropertiesEXT::descriptorBufferAddressSpaceSize
- VUID-VkBufferCreateInfo-usage-08098
  If usage includes VK_BUFFER_USAGE_RESOURCE_DESCRIPTOR_BUFFER_BIT_EXT, creating this VkBuffer **must** not cause the total required space for all currently valid buffers using this flag on the device to exceed VkPhysicalDeviceDescriptorBufferPropertiesEXT::resourceDescriptorBufferAddressSpaceSize or VkPhysicalDeviceDescriptorBufferPropertiesEXT::descriptorBufferAddressSpaceSize

- VUID-VkBufferCreateInfo-flags-08099
  If flags includes VK_BUFFER_CREATE_DESCRIPTOR_BUFFER_CAPTURE_REPLAY_BIT_EXT, the descriptorBufferCaptureReplay feature **must** be enabled
- VUID-VkBufferCreateInfo-pNext-08100
  If the pNext chain includes a VkOpaqueCaptureDescriptorDataCreateInfoEXT structure, flags **must** contain VK_BUFFER_CREATE_DESCRIPTOR_BUFFER_CAPTURE_REPLAY_BIT_EXT
- VUID-VkBufferCreateInfo-usage-08101
  If usage includes VK_BUFFER_USAGE_PUSH_DESCRIPTORS_DESCRIPTOR_BUFFER_BIT_EXT, the descriptorBufferPushDescriptors feature **must** be enabled
- VUID-VkBufferCreateInfo-usage-08102
  If usage includes VK_BUFFER_USAGE_PUSH_DESCRIPTORS_DESCRIPTOR_BUFFER_BIT_EXT VkPhysicalDeviceDescriptorBufferPropertiesEXT::bufferlessPushDescriptors **must** be VK_FALSE
- VUID-VkBufferCreateInfo-usage-08103
  If usage includes VK_BUFFER_USAGE_PUSH_DESCRIPTORS_DESCRIPTOR_BUFFER_BIT_EXT, usage **must** contain at least one of VK_BUFFER_USAGE_RESOURCE_DESCRIPTOR_BUFFER_BIT_EXT or VK_BUFFER_USAGE_SAMPLER_DESCRIPTOR_BUFFER_BIT_EXT
- VUID-VkBufferCreateInfo-flags-09641
  If flags includes VK_BUFFER_CREATE_PROTECTED_BIT, then usage **must** not contain any of the following bits
  - VK_BUFFER_USAGE_TRANSFORM_FEEDBACK_BUFFER_BIT_EXT
  - VK_BUFFER_USAGE_TRANSFORM_FEEDBACK_COUNTER_BUFFER_BIT_EXT
  - VK_BUFFER_USAGE_CONDITIONAL_RENDERING_BIT_EXT
  - VK_BUFFER_USAGE_ACCELERATION_STRUCTURE_BUILD_INPUT_READ_ONLY_BIT_KHR
  - VK_BUFFER_USAGE_ACCELERATION_STRUCTURE_STORAGE_BIT_KHR
  - VK_BUFFER_USAGE_SHADER_BINDING_TABLE_BIT_KHR
  - VK_BUFFER_USAGE_SAMPLER_DESCRIPTOR_BUFFER_BIT_EXT
  - VK_BUFFER_USAGE_RESOURCE_DESCRIPTOR_BUFFER_BIT_EXT
  - VK_BUFFER_USAGE_PUSH_DESCRIPTORS_DESCRIPTOR_BUFFER_BIT_EXT
  - VK_BUFFER_USAGE_MICROMAP_BUILD_INPUT_READ_ONLY_BIT_EXT
  - VK_BUFFER_USAGE_MICROMAP_STORAGE_BIT_EXT

## Valid Usage (Implicit)

- VUID-VkBufferCreateInfo-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_BUFFER_CREATE_INFO
- VUID-VkBufferCreateInfo-pNext-pNext
  Each pNext member of any structure (including this one) in the pNext chain **must** be either NULL or a pointer to a valid instance of VkBufferCollectionBufferCreateInfoFUCHSIA, VkBufferDeviceAddressCreateInfoEXT, VkBufferOpaqueCaptureAddressCreateInfo,

VkBufferUsageFlags2CreateInfoKHR, VkDedicatedAllocationBufferCreateInfoNV, VkExternalMemoryBufferCreateInfo, VkOpaqueCaptureDescriptorDataCreateInfoEXT, or VkVideoProfileListInfoKHR

- VUID-VkBufferCreateInfo-sType-unique
  The `sType` value of each struct in the `pNext` chain **must** be unique
- VUID-VkBufferCreateInfo-flags-parameter
  `flags` **must** be a valid combination of VkBufferCreateFlagBits values
- VUID-VkBufferCreateInfo-sharingMode-parameter
  `sharingMode` **must** be a valid VkSharingMode value

The `VkBufferUsageFlags2CreateInfoKHR` structure is defined as:

```
// Provided by VK_KHR_maintenance5
typedef struct VkBufferUsageFlags2CreateInfoKHR {
    VkStructureType           sType;
    const void*               pNext;
    VkBufferUsageFlags2KHR    usage;
} VkBufferUsageFlags2CreateInfoKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `usage` is a bitmask of VkBufferUsageFlagBits2KHR specifying allowed usages of the buffer.

If this structure is included in the `pNext` chain of a buffer creation structure, `usage` is used instead of the corresponding `usage` value passed in that creation structure, allowing additional usage flags to be specified. If this structure is included in the `pNext` chain of a buffer query structure, the usage flags of the buffer are returned in `usage` of this structure, and the usage flags representable in `usage` of the buffer query structure are also returned in that field.

### Valid Usage (Implicit)

- VUID-VkBufferUsageFlags2CreateInfoKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_BUFFER_USAGE_FLAGS_2_CREATE_INFO_KHR`
- VUID-VkBufferUsageFlags2CreateInfoKHR-usage-parameter
  `usage` **must** be a valid combination of VkBufferUsageFlagBits2KHR values
- VUID-VkBufferUsageFlags2CreateInfoKHR-usage-requiredbitmask
  `usage` **must** not be `0`

Bits which **can** be set in VkBufferUsageFlags2CreateInfoKHR::`usage`, specifying usage behavior of a buffer, are:

```
// Provided by VK_KHR_maintenance5
// Flag bits for VkBufferUsageFlagBits2KHR
```

```
typedef VkFlags64 VkBufferUsageFlagBits2KHR;
static const VkBufferUsageFlagBits2KHR VK_BUFFER_USAGE_2_TRANSFER_SRC_BIT_KHR =
0x00000001ULL;
static const VkBufferUsageFlagBits2KHR VK_BUFFER_USAGE_2_TRANSFER_DST_BIT_KHR =
0x00000002ULL;
static const VkBufferUsageFlagBits2KHR VK_BUFFER_USAGE_2_UNIFORM_TEXEL_BUFFER_BIT_KHR
= 0x00000004ULL;
static const VkBufferUsageFlagBits2KHR VK_BUFFER_USAGE_2_STORAGE_TEXEL_BUFFER_BIT_KHR
= 0x00000008ULL;
static const VkBufferUsageFlagBits2KHR VK_BUFFER_USAGE_2_UNIFORM_BUFFER_BIT_KHR =
0x00000010ULL;
static const VkBufferUsageFlagBits2KHR VK_BUFFER_USAGE_2_STORAGE_BUFFER_BIT_KHR =
0x00000020ULL;
static const VkBufferUsageFlagBits2KHR VK_BUFFER_USAGE_2_INDEX_BUFFER_BIT_KHR =
0x00000040ULL;
static const VkBufferUsageFlagBits2KHR VK_BUFFER_USAGE_2_VERTEX_BUFFER_BIT_KHR =
0x00000080ULL;
static const VkBufferUsageFlagBits2KHR VK_BUFFER_USAGE_2_INDIRECT_BUFFER_BIT_KHR =
0x00000100ULL;
// Provided by VK_KHR_maintenance5 with VK_AMDX_shader_enqueue
static const VkBufferUsageFlagBits2KHR
VK_BUFFER_USAGE_2_EXECUTION_GRAPH_SCRATCH_BIT_AMDX = 0x02000000ULL;
// Provided by VK_KHR_maintenance5 with VK_EXT_conditional_rendering
static const VkBufferUsageFlagBits2KHR VK_BUFFER_USAGE_2_CONDITIONAL_RENDERING_BIT_EXT
= 0x00000200ULL;
// Provided by VK_KHR_maintenance5 with VK_KHR_ray_tracing_pipeline
static const VkBufferUsageFlagBits2KHR VK_BUFFER_USAGE_2_SHADER_BINDING_TABLE_BIT_KHR
= 0x00000400ULL;
// Provided by VK_KHR_maintenance5 with VK_NV_ray_tracing
static const VkBufferUsageFlagBits2KHR VK_BUFFER_USAGE_2_RAY_TRACING_BIT_NV =
0x00000400ULL;
// Provided by VK_KHR_maintenance5 with VK_EXT_transform_feedback
static const VkBufferUsageFlagBits2KHR
VK_BUFFER_USAGE_2_TRANSFORM_FEEDBACK_BUFFER_BIT_EXT = 0x00000800ULL;
// Provided by VK_KHR_maintenance5 with VK_EXT_transform_feedback
static const VkBufferUsageFlagBits2KHR
VK_BUFFER_USAGE_2_TRANSFORM_FEEDBACK_COUNTER_BUFFER_BIT_EXT = 0x00001000ULL;
// Provided by VK_KHR_maintenance5 with VK_KHR_video_decode_queue
static const VkBufferUsageFlagBits2KHR VK_BUFFER_USAGE_2_VIDEO_DECODE_SRC_BIT_KHR =
0x00002000ULL;
// Provided by VK_KHR_maintenance5 with VK_KHR_video_decode_queue
static const VkBufferUsageFlagBits2KHR VK_BUFFER_USAGE_2_VIDEO_DECODE_DST_BIT_KHR =
0x00004000ULL;
// Provided by VK_KHR_maintenance5 with VK_KHR_video_encode_queue
static const VkBufferUsageFlagBits2KHR VK_BUFFER_USAGE_2_VIDEO_ENCODE_DST_BIT_KHR =
0x00008000ULL;
// Provided by VK_KHR_maintenance5 with VK_KHR_video_encode_queue
static const VkBufferUsageFlagBits2KHR VK_BUFFER_USAGE_2_VIDEO_ENCODE_SRC_BIT_KHR =
0x00010000ULL;
// Provided by VK_KHR_maintenance5 with (VK_VERSION_1_2 or
VK_KHR_buffer_device_address) or VK_EXT_buffer_device_address
```

```
static const VkBufferUsageFlagBits2KHR VK_BUFFER_USAGE_2_SHADER_DEVICE_ADDRESS_BIT_KHR
= 0x00020000ULL;
// Provided by VK_KHR_acceleration_structure with VK_KHR_maintenance5
static const VkBufferUsageFlagBits2KHR
VK_BUFFER_USAGE_2_ACCELERATION_STRUCTURE_BUILD_INPUT_READ_ONLY_BIT_KHR =
0x00080000ULL;
// Provided by VK_KHR_acceleration_structure with VK_KHR_maintenance5
static const VkBufferUsageFlagBits2KHR
VK_BUFFER_USAGE_2_ACCELERATION_STRUCTURE_STORAGE_BIT_KHR = 0x00100000ULL;
// Provided by VK_KHR_maintenance5 with VK_EXT_descriptor_buffer
static const VkBufferUsageFlagBits2KHR
VK_BUFFER_USAGE_2_SAMPLER_DESCRIPTOR_BUFFER_BIT_EXT = 0x00200000ULL;
// Provided by VK_KHR_maintenance5 with VK_EXT_descriptor_buffer
static const VkBufferUsageFlagBits2KHR
VK_BUFFER_USAGE_2_RESOURCE_DESCRIPTOR_BUFFER_BIT_EXT = 0x00400000ULL;
// Provided by VK_KHR_maintenance5 with VK_EXT_descriptor_buffer
static const VkBufferUsageFlagBits2KHR
VK_BUFFER_USAGE_2_PUSH_DESCRIPTORS_DESCRIPTOR_BUFFER_BIT_EXT = 0x04000000ULL;
// Provided by VK_KHR_maintenance5 with VK_EXT_opacity_micromap
static const VkBufferUsageFlagBits2KHR
VK_BUFFER_USAGE_2_MICROMAP_BUILD_INPUT_READ_ONLY_BIT_EXT = 0x00800000ULL;
// Provided by VK_KHR_maintenance5 with VK_EXT_opacity_micromap
static const VkBufferUsageFlagBits2KHR VK_BUFFER_USAGE_2_MICROMAP_STORAGE_BIT_EXT =
0x01000000ULL;
```

- `VK_BUFFER_USAGE_2_TRANSFER_SRC_BIT_KHR` specifies that the buffer **can** be used as the source of a *transfer command* (see the definition of `VK_PIPELINE_STAGE_TRANSFER_BIT`).
- `VK_BUFFER_USAGE_2_TRANSFER_DST_BIT_KHR` specifies that the buffer **can** be used as the destination of a transfer command.
- `VK_BUFFER_USAGE_2_UNIFORM_TEXEL_BUFFER_BIT_KHR` specifies that the buffer **can** be used to create a `VkBufferView` suitable for occupying a `VkDescriptorSet` slot of type `VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER`.
- `VK_BUFFER_USAGE_2_STORAGE_TEXEL_BUFFER_BIT_KHR` specifies that the buffer **can** be used to create a `VkBufferView` suitable for occupying a `VkDescriptorSet` slot of type `VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER`.
- `VK_BUFFER_USAGE_2_UNIFORM_BUFFER_BIT_KHR` specifies that the buffer **can** be used in a `VkDescriptorBufferInfo` suitable for occupying a `VkDescriptorSet` slot either of type `VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER` or `VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC`.
- `VK_BUFFER_USAGE_2_STORAGE_BUFFER_BIT_KHR` specifies that the buffer **can** be used in a `VkDescriptorBufferInfo` suitable for occupying a `VkDescriptorSet` slot either of type `VK_DESCRIPTOR_TYPE_STORAGE_BUFFER` or `VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC`.
- `VK_BUFFER_USAGE_2_INDEX_BUFFER_BIT_KHR` specifies that the buffer is suitable for passing as the `buffer` parameter to vkCmdBindIndexBuffer2KHR and vkCmdBindIndexBuffer.
- `VK_BUFFER_USAGE_2_VERTEX_BUFFER_BIT_KHR` specifies that the buffer is suitable for passing as an element of the `pBuffers` array to vkCmdBindVertexBuffers.
- `VK_BUFFER_USAGE_2_INDIRECT_BUFFER_BIT_KHR` specifies that the buffer is suitable for passing as the

buffer parameter to vkCmdDrawIndirect, vkCmdDrawIndexedIndirect, vkCmdDrawMeshTasksIndirectNV, vkCmdDrawMeshTasksIndirectCountNV, vkCmdDrawMeshTasksIndirectEXT, vkCmdDrawMeshTasksIndirectCountEXT, vkCmdDrawClusterIndirectHUAWEI, or vkCmdDispatchIndirect. It is also suitable for passing as the buffer member of VkIndirectCommandsStreamNV, or sequencesCountBuffer or sequencesIndexBuffer or preprocessedBuffer member of VkGeneratedCommandsInfoNV

- VK_BUFFER_USAGE_2_CONDITIONAL_RENDERING_BIT_EXT specifies that the buffer is suitable for passing as the buffer parameter to vkCmdBeginConditionalRenderingEXT.
- VK_BUFFER_USAGE_2_TRANSFORM_FEEDBACK_BUFFER_BIT_EXT specifies that the buffer is suitable for using for binding as a transform feedback buffer with vkCmdBindTransformFeedbackBuffersEXT.
- VK_BUFFER_USAGE_2_TRANSFORM_FEEDBACK_COUNTER_BUFFER_BIT_EXT specifies that the buffer is suitable for using as a counter buffer with vkCmdBeginTransformFeedbackEXT and vkCmdEndTransformFeedbackEXT.
- VK_BUFFER_USAGE_2_SAMPLER_DESCRIPTOR_BUFFER_BIT_EXT specifies that the buffer is suitable to contain sampler and combined image sampler descriptors when bound as a descriptor buffer. Buffers containing combined image sampler descriptors **must** also specify VK_BUFFER_USAGE_2_RESOURCE_DESCRIPTOR_BUFFER_BIT_EXT.
- VK_BUFFER_USAGE_2_RESOURCE_DESCRIPTOR_BUFFER_BIT_EXT specifies that the buffer is suitable to contain resource descriptors when bound as a descriptor buffer.
- VK_BUFFER_USAGE_2_PUSH_DESCRIPTORS_DESCRIPTOR_BUFFER_BIT_EXT specifies that the buffer, when bound, **can** be used by the implementation to support push descriptors when using descriptor buffers.
- VK_BUFFER_USAGE_2_RAY_TRACING_BIT_NV specifies that the buffer is suitable for use in vkCmdTraceRaysNV.
- VK_BUFFER_USAGE_2_SHADER_BINDING_TABLE_BIT_KHR specifies that the buffer is suitable for use as a Shader Binding Table.
- VK_BUFFER_USAGE_2_ACCELERATION_STRUCTURE_BUILD_INPUT_READ_ONLY_BIT_KHR specifies that the buffer is suitable for use as a read-only input to an acceleration structure build.
- VK_BUFFER_USAGE_2_ACCELERATION_STRUCTURE_STORAGE_BIT_KHR specifies that the buffer is suitable for storage space for a VkAccelerationStructureKHR.
- VK_BUFFER_USAGE_2_SHADER_DEVICE_ADDRESS_BIT_KHR specifies that the buffer **can** be used to retrieve a buffer device address via vkGetBufferDeviceAddressKHR and use that address to access the buffer's memory from a shader.
- VK_BUFFER_USAGE_2_VIDEO_DECODE_SRC_BIT_KHR specifies that the buffer **can** be used as the source video bitstream buffer in a video decode operation.
- VK_BUFFER_USAGE_2_VIDEO_DECODE_DST_BIT_KHR is reserved for future use.
- VK_BUFFER_USAGE_2_VIDEO_ENCODE_DST_BIT_KHR specifies that the buffer **can** be used as the destination video bitstream buffer in a video encode operation.
- VK_BUFFER_USAGE_2_VIDEO_ENCODE_SRC_BIT_KHR is reserved for future use.
- VK_BUFFER_USAGE_2_EXECUTION_GRAPH_SCRATCH_BIT_AMDX specifies that the buffer **can** be used for as

scratch memory for execution graph dispatch.

```
// Provided by VK_KHR_maintenance5
typedef VkFlags64 VkBufferUsageFlags2KHR;
```

VkBufferUsageFlags2KHR is a bitmask type for setting a mask of zero or more VkBufferUsageFlagBits2KHR.

Bits which **can** be set in VkBufferCreateInfo::usage, specifying usage behavior of a buffer, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkBufferUsageFlagBits {
    VK_BUFFER_USAGE_TRANSFER_SRC_BIT = 0x00000001,
    VK_BUFFER_USAGE_TRANSFER_DST_BIT = 0x00000002,
    VK_BUFFER_USAGE_UNIFORM_TEXEL_BUFFER_BIT = 0x00000004,
    VK_BUFFER_USAGE_STORAGE_TEXEL_BUFFER_BIT = 0x00000008,
    VK_BUFFER_USAGE_UNIFORM_BUFFER_BIT = 0x00000010,
    VK_BUFFER_USAGE_STORAGE_BUFFER_BIT = 0x00000020,
    VK_BUFFER_USAGE_INDEX_BUFFER_BIT = 0x00000040,
    VK_BUFFER_USAGE_VERTEX_BUFFER_BIT = 0x00000080,
    VK_BUFFER_USAGE_INDIRECT_BUFFER_BIT = 0x00000100,
    VK_BUFFER_USAGE_SHADER_DEVICE_ADDRESS_BIT = 0x00020000,
  // Provided by VK_KHR_video_decode_queue
    VK_BUFFER_USAGE_VIDEO_DECODE_SRC_BIT_KHR = 0x00002000,
  // Provided by VK_KHR_video_decode_queue
    VK_BUFFER_USAGE_VIDEO_DECODE_DST_BIT_KHR = 0x00004000,
  // Provided by VK_EXT_transform_feedback
    VK_BUFFER_USAGE_TRANSFORM_FEEDBACK_BUFFER_BIT_EXT = 0x00000800,
  // Provided by VK_EXT_transform_feedback
    VK_BUFFER_USAGE_TRANSFORM_FEEDBACK_COUNTER_BUFFER_BIT_EXT = 0x00001000,
  // Provided by VK_EXT_conditional_rendering
    VK_BUFFER_USAGE_CONDITIONAL_RENDERING_BIT_EXT = 0x00000200,
#ifdef VK_ENABLE_BETA_EXTENSIONS
  // Provided by VK_AMDX_shader_enqueue
    VK_BUFFER_USAGE_EXECUTION_GRAPH_SCRATCH_BIT_AMDX = 0x02000000,
#endif
  // Provided by VK_KHR_acceleration_structure
    VK_BUFFER_USAGE_ACCELERATION_STRUCTURE_BUILD_INPUT_READ_ONLY_BIT_KHR = 0x00080000,
  // Provided by VK_KHR_acceleration_structure
    VK_BUFFER_USAGE_ACCELERATION_STRUCTURE_STORAGE_BIT_KHR = 0x00100000,
  // Provided by VK_KHR_ray_tracing_pipeline
    VK_BUFFER_USAGE_SHADER_BINDING_TABLE_BIT_KHR = 0x00000400,
  // Provided by VK_KHR_video_encode_queue
    VK_BUFFER_USAGE_VIDEO_ENCODE_DST_BIT_KHR = 0x00008000,
  // Provided by VK_KHR_video_encode_queue
    VK_BUFFER_USAGE_VIDEO_ENCODE_SRC_BIT_KHR = 0x00010000,
  // Provided by VK_EXT_descriptor_buffer
    VK_BUFFER_USAGE_SAMPLER_DESCRIPTOR_BUFFER_BIT_EXT = 0x00200000,
  // Provided by VK_EXT_descriptor_buffer
```

```
    VK_BUFFER_USAGE_RESOURCE_DESCRIPTOR_BUFFER_BIT_EXT = 0x00400000,
  // Provided by VK_EXT_descriptor_buffer
    VK_BUFFER_USAGE_PUSH_DESCRIPTORS_DESCRIPTOR_BUFFER_BIT_EXT = 0x04000000,
  // Provided by VK_EXT_opacity_micromap
    VK_BUFFER_USAGE_MICROMAP_BUILD_INPUT_READ_ONLY_BIT_EXT = 0x00800000,
  // Provided by VK_EXT_opacity_micromap
    VK_BUFFER_USAGE_MICROMAP_STORAGE_BIT_EXT = 0x01000000,
  // Provided by VK_NV_ray_tracing
    VK_BUFFER_USAGE_RAY_TRACING_BIT_NV = VK_BUFFER_USAGE_SHADER_BINDING_TABLE_BIT_KHR,
  // Provided by VK_EXT_buffer_device_address
    VK_BUFFER_USAGE_SHADER_DEVICE_ADDRESS_BIT_EXT =
VK_BUFFER_USAGE_SHADER_DEVICE_ADDRESS_BIT,
  // Provided by VK_KHR_buffer_device_address
    VK_BUFFER_USAGE_SHADER_DEVICE_ADDRESS_BIT_KHR =
VK_BUFFER_USAGE_SHADER_DEVICE_ADDRESS_BIT,
} VkBufferUsageFlagBits;
```

- `VK_BUFFER_USAGE_TRANSFER_SRC_BIT` specifies that the buffer **can** be used as the source of a *transfer command* (see the definition of `VK_PIPELINE_STAGE_TRANSFER_BIT`).
- `VK_BUFFER_USAGE_TRANSFER_DST_BIT` specifies that the buffer **can** be used as the destination of a transfer command.
- `VK_BUFFER_USAGE_UNIFORM_TEXEL_BUFFER_BIT` specifies that the buffer **can** be used to create a `VkBufferView` suitable for occupying a `VkDescriptorSet` slot of type `VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER`.
- `VK_BUFFER_USAGE_STORAGE_TEXEL_BUFFER_BIT` specifies that the buffer **can** be used to create a `VkBufferView` suitable for occupying a `VkDescriptorSet` slot of type `VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER`.
- `VK_BUFFER_USAGE_UNIFORM_BUFFER_BIT` specifies that the buffer **can** be used in a `VkDescriptorBufferInfo` suitable for occupying a `VkDescriptorSet` slot either of type `VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER` or `VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC`.
- `VK_BUFFER_USAGE_STORAGE_BUFFER_BIT` specifies that the buffer **can** be used in a `VkDescriptorBufferInfo` suitable for occupying a `VkDescriptorSet` slot either of type `VK_DESCRIPTOR_TYPE_STORAGE_BUFFER` or `VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC`.
- `VK_BUFFER_USAGE_INDEX_BUFFER_BIT` specifies that the buffer is suitable for passing as the `buffer` parameter to vkCmdBindIndexBuffer2KHR and vkCmdBindIndexBuffer.
- `VK_BUFFER_USAGE_VERTEX_BUFFER_BIT` specifies that the buffer is suitable for passing as an element of the `pBuffers` array to vkCmdBindVertexBuffers.
- `VK_BUFFER_USAGE_INDIRECT_BUFFER_BIT` specifies that the buffer is suitable for passing as the `buffer` parameter to vkCmdDrawIndirect, vkCmdDrawIndexedIndirect, vkCmdDrawMeshTasksIndirectNV, vkCmdDrawMeshTasksIndirectCountNV, `vkCmdDrawMeshTasksIndirectEXT`, `vkCmdDrawMeshTasksIndirectCountEXT`, vkCmdDrawClusterIndirectHUAWEI, or vkCmdDispatchIndirect. It is also suitable for passing as the `buffer` member of `VkIndirectCommandsStreamNV`, or `sequencesCountBuffer` or `sequencesIndexBuffer` or `preprocessedBuffer` member of `VkGeneratedCommandsInfoNV`
- `VK_BUFFER_USAGE_CONDITIONAL_RENDERING_BIT_EXT` specifies that the buffer is suitable for passing

as the buffer parameter to vkCmdBeginConditionalRenderingEXT.

- VK_BUFFER_USAGE_TRANSFORM_FEEDBACK_BUFFER_BIT_EXT specifies that the buffer is suitable for using for binding as a transform feedback buffer with vkCmdBindTransformFeedbackBuffersEXT.
- VK_BUFFER_USAGE_TRANSFORM_FEEDBACK_COUNTER_BUFFER_BIT_EXT specifies that the buffer is suitable for using as a counter buffer with vkCmdBeginTransformFeedbackEXT and vkCmdEndTransformFeedbackEXT.
- VK_BUFFER_USAGE_SAMPLER_DESCRIPTOR_BUFFER_BIT_EXT specifies that the buffer is suitable to contain sampler and combined image sampler descriptors when bound as a descriptor buffer. Buffers containing combined image sampler descriptors **must** also specify VK_BUFFER_USAGE_RESOURCE_DESCRIPTOR_BUFFER_BIT_EXT.
- VK_BUFFER_USAGE_RESOURCE_DESCRIPTOR_BUFFER_BIT_EXT specifies that the buffer is suitable to contain resource descriptors when bound as a descriptor buffer.
- VK_BUFFER_USAGE_PUSH_DESCRIPTORS_DESCRIPTOR_BUFFER_BIT_EXT specifies that the buffer, when bound, **can** be used by the implementation to support push descriptors when using descriptor buffers.
- VK_BUFFER_USAGE_RAY_TRACING_BIT_NV specifies that the buffer is suitable for use in vkCmdTraceRaysNV.
- VK_BUFFER_USAGE_SHADER_BINDING_TABLE_BIT_KHR specifies that the buffer is suitable for use as a Shader Binding Table.
- VK_BUFFER_USAGE_ACCELERATION_STRUCTURE_BUILD_INPUT_READ_ONLY_BIT_KHR specifies that the buffer is suitable for use as a read-only input to an acceleration structure build.
- VK_BUFFER_USAGE_ACCELERATION_STRUCTURE_STORAGE_BIT_KHR specifies that the buffer is suitable for storage space for a VkAccelerationStructureKHR.
- VK_BUFFER_USAGE_SHADER_DEVICE_ADDRESS_BIT specifies that the buffer **can** be used to retrieve a buffer device address via vkGetBufferDeviceAddressKHR and use that address to access the buffer's memory from a shader.
- VK_BUFFER_USAGE_VIDEO_DECODE_SRC_BIT_KHR specifies that the buffer **can** be used as the source video bitstream buffer in a video decode operation.
- VK_BUFFER_USAGE_VIDEO_DECODE_DST_BIT_KHR is reserved for future use.
- VK_BUFFER_USAGE_VIDEO_ENCODE_DST_BIT_KHR specifies that the buffer **can** be used as the destination video bitstream buffer in a video encode operation.
- VK_BUFFER_USAGE_VIDEO_ENCODE_SRC_BIT_KHR is reserved for future use.
- VK_BUFFER_USAGE_EXECUTION_GRAPH_SCRATCH_BIT_AMDX specifies that the buffer **can** be used for as scratch memory for execution graph dispatch.

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkBufferUsageFlags;
```

VkBufferUsageFlags is a bitmask type for setting a mask of zero or more VkBufferUsageFlagBits.

Bits which **can** be set in VkBufferCreateInfo::flags, specifying additional parameters of a buffer,

are:

```
// Provided by VK_VERSION_1_0
typedef enum VkBufferCreateFlagBits {
    VK_BUFFER_CREATE_SPARSE_BINDING_BIT = 0x00000001,
    VK_BUFFER_CREATE_SPARSE_RESIDENCY_BIT = 0x00000002,
    VK_BUFFER_CREATE_SPARSE_ALIASED_BIT = 0x00000004,
  // Provided by VK_VERSION_1_1
    VK_BUFFER_CREATE_PROTECTED_BIT = 0x00000008,
    VK_BUFFER_CREATE_DEVICE_ADDRESS_CAPTURE_REPLAY_BIT = 0x00000010,
  // Provided by VK_EXT_descriptor_buffer
    VK_BUFFER_CREATE_DESCRIPTOR_BUFFER_CAPTURE_REPLAY_BIT_EXT = 0x00000020,
  // Provided by VK_KHR_video_maintenance1
    VK_BUFFER_CREATE_VIDEO_PROFILE_INDEPENDENT_BIT_KHR = 0x00000040,
  // Provided by VK_EXT_buffer_device_address
    VK_BUFFER_CREATE_DEVICE_ADDRESS_CAPTURE_REPLAY_BIT_EXT =
VK_BUFFER_CREATE_DEVICE_ADDRESS_CAPTURE_REPLAY_BIT,
  // Provided by VK_KHR_buffer_device_address
    VK_BUFFER_CREATE_DEVICE_ADDRESS_CAPTURE_REPLAY_BIT_KHR =
VK_BUFFER_CREATE_DEVICE_ADDRESS_CAPTURE_REPLAY_BIT,
} VkBufferCreateFlagBits;
```

- `VK_BUFFER_CREATE_SPARSE_BINDING_BIT` specifies that the buffer will be backed using sparse memory binding.
- `VK_BUFFER_CREATE_SPARSE_RESIDENCY_BIT` specifies that the buffer **can** be partially backed using sparse memory binding. Buffers created with this flag **must** also be created with the `VK_BUFFER_CREATE_SPARSE_BINDING_BIT` flag.
- `VK_BUFFER_CREATE_SPARSE_ALIASED_BIT` specifies that the buffer will be backed using sparse memory binding with memory ranges that might also simultaneously be backing another buffer (or another portion of the same buffer). Buffers created with this flag **must** also be created with the `VK_BUFFER_CREATE_SPARSE_BINDING_BIT` flag.
- `VK_BUFFER_CREATE_PROTECTED_BIT` specifies that the buffer is a protected buffer.
- `VK_BUFFER_CREATE_DEVICE_ADDRESS_CAPTURE_REPLAY_BIT` specifies that the buffer's address **can** be saved and reused on a subsequent run (e.g. for trace capture and replay), see VkBufferOpaqueCaptureAddressCreateInfo for more detail.
- `VK_BUFFER_CREATE_DESCRIPTOR_BUFFER_CAPTURE_REPLAY_BIT_EXT` specifies that the buffer **can** be used with descriptor buffers when capturing and replaying (e.g. for trace capture and replay), see VkOpaqueCaptureDescriptorDataCreateInfoEXT for more detail.
- `VK_BUFFER_CREATE_VIDEO_PROFILE_INDEPENDENT_BIT_KHR` specifies that the buffer **can** be used in video coding operations without having to specify at buffer creation time the set of video profiles the buffer will be used with.

See Sparse Resource Features and Physical Device Features for details of the sparse memory features supported on a device.

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkBufferCreateFlags;
```

VkBufferCreateFlags is a bitmask type for setting a mask of zero or more VkBufferCreateFlagBits.

If the pNext chain includes a VkDedicatedAllocationBufferCreateInfoNV structure, then that structure includes an enable controlling whether the buffer will have a dedicated memory allocation bound to it.

The VkDedicatedAllocationBufferCreateInfoNV structure is defined as:

```
// Provided by VK_NV_dedicated_allocation
typedef struct VkDedicatedAllocationBufferCreateInfoNV {
    VkStructureType    sType;
    const void*        pNext;
    VkBool32           dedicatedAllocation;
} VkDedicatedAllocationBufferCreateInfoNV;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- dedicatedAllocation specifies whether the buffer will have a dedicated allocation bound to it.

### Valid Usage (Implicit)

- VUID-VkDedicatedAllocationBufferCreateInfoNV-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_DEDICATED_ALLOCATION_BUFFER_CREATE_INFO_NV

To define a set of external memory handle types that **may** be used as backing store for a buffer, add a VkExternalMemoryBufferCreateInfo structure to the pNext chain of the VkBufferCreateInfo structure. The VkExternalMemoryBufferCreateInfo structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkExternalMemoryBufferCreateInfo {
    VkStructureType                    sType;
    const void*                        pNext;
    VkExternalMemoryHandleTypeFlags    handleTypes;
} VkExternalMemoryBufferCreateInfo;
```

or the equivalent

```
// Provided by VK_KHR_external_memory
typedef VkExternalMemoryBufferCreateInfo VkExternalMemoryBufferCreateInfoKHR;
```

| | |
|---|---|
| **NOTE** | A `VkExternalMemoryBufferCreateInfo` structure with a non-zero `handleTypes` field must be included in the creation parameters for a buffer that will be bound to memory that is either exported or imported. |

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `handleTypes` is zero or a bitmask of VkExternalMemoryHandleTypeFlagBits specifying one or more external memory handle types.

### Valid Usage (Implicit)

- VUID-VkExternalMemoryBufferCreateInfo-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_EXTERNAL_MEMORY_BUFFER_CREATE_INFO`
- VUID-VkExternalMemoryBufferCreateInfo-handleTypes-parameter
  `handleTypes` **must** be a valid combination of VkExternalMemoryHandleTypeFlagBits values

To request a specific device address for a buffer, add a VkBufferOpaqueCaptureAddressCreateInfo structure to the `pNext` chain of the VkBufferCreateInfo structure. The `VkBufferOpaqueCaptureAddressCreateInfo` structure is defined as:

```
typedef struct VkBufferOpaqueCaptureAddressCreateInfo {
    VkStructureType    sType;
    const void*        pNext;
    uint64_t           opaqueCaptureAddress;
} VkBufferOpaqueCaptureAddressCreateInfo;
```

or the equivalent

```
// Provided by VK_KHR_buffer_device_address
typedef VkBufferOpaqueCaptureAddressCreateInfo
VkBufferOpaqueCaptureAddressCreateInfoKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `opaqueCaptureAddress` is the opaque capture address requested for the buffer.

If `opaqueCaptureAddress` is zero, no specific address is requested.

If `opaqueCaptureAddress` is not zero, then it **should** be an address retrieved from vkGetBufferOpaqueCaptureAddressKHR for an identically created buffer on the same implementation.

If this structure is not present, it is as if `opaqueCaptureAddress` is zero.

Apps **should** avoid creating buffers with app-provided addresses and implementation-provided addresses in the same process, to reduce the likelihood of VK_ERROR_INVALID_OPAQUE_CAPTURE_ADDRESS errors.

**NOTE**

The expected usage for this is that a trace capture/replay tool will add the VK_BUFFER_CREATE_DEVICE_ADDRESS_CAPTURE_REPLAY_BIT flag to all buffers that use VK_BUFFER_USAGE_SHADER_DEVICE_ADDRESS_BIT, and during capture will save the queried opaque device addresses in the trace. During replay, the buffers will be created specifying the original address so any address values stored in the trace data will remain valid.

Implementations are expected to separate such buffers in the GPU address space so normal allocations will avoid using these addresses. Apps/tools should avoid mixing app-provided and implementation-provided addresses for buffers created with VK_BUFFER_CREATE_DEVICE_ADDRESS_CAPTURE_REPLAY_BIT, to avoid address space allocation conflicts.

### Valid Usage (Implicit)

- VUID-VkBufferOpaqueCaptureAddressCreateInfo-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_BUFFER_OPAQUE_CAPTURE_ADDRESS_CREATE_INFO

Alternatively, to request a specific device address for a buffer, add a VkBufferDeviceAddressCreateInfoEXT structure to the pNext chain of the VkBufferCreateInfo structure. The VkBufferDeviceAddressCreateInfoEXT structure is defined as:

```
// Provided by VK_EXT_buffer_device_address
typedef struct VkBufferDeviceAddressCreateInfoEXT {
    VkStructureType    sType;
    const void*        pNext;
    VkDeviceAddress    deviceAddress;
} VkBufferDeviceAddressCreateInfoEXT;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- deviceAddress is the device address requested for the buffer.

If deviceAddress is zero, no specific address is requested.

If deviceAddress is not zero, then it **must** be an address retrieved from an identically created buffer on the same implementation. The buffer **must** also be bound to an identically created VkDeviceMemory object.

If this structure is not present, it is as if deviceAddress is zero.

Apps **should** avoid creating buffers with app-provided addresses and implementation-provided

addresses in the same process, to reduce the likelihood of VK_ERROR_INVALID_DEVICE_ADDRESS_EXT errors.

### Valid Usage (Implicit)

- VUID-VkBufferDeviceAddressCreateInfoEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_BUFFER_DEVICE_ADDRESS_CREATE_INFO_EXT

The VkBufferCollectionBufferCreateInfoFUCHSIA structure is defined as:

```
// Provided by VK_FUCHSIA_buffer_collection
typedef struct VkBufferCollectionBufferCreateInfoFUCHSIA {
    VkStructureType              sType;
    const void*                  pNext;
    VkBufferCollectionFUCHSIA    collection;
    uint32_t                     index;
} VkBufferCollectionBufferCreateInfoFUCHSIA;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure
- collection is the VkBufferCollectionFUCHSIA handle
- index is the index of the buffer in the buffer collection from which the memory will be imported

### Valid Usage

- VUID-VkBufferCollectionBufferCreateInfoFUCHSIA-index-06388
  index **must** be less than VkBufferCollectionPropertiesFUCHSIA::bufferCount

### Valid Usage (Implicit)

- VUID-VkBufferCollectionBufferCreateInfoFUCHSIA-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_BUFFER_COLLECTION_BUFFER_CREATE_INFO_FUCHSIA
- VUID-VkBufferCollectionBufferCreateInfoFUCHSIA-collection-parameter
  collection **must** be a valid VkBufferCollectionFUCHSIA handle

To destroy a buffer, call:

```
// Provided by VK_VERSION_1_0
void vkDestroyBuffer(
    VkDevice                                    device,
    VkBuffer                                    buffer,
    const VkAllocationCallbacks*                pAllocator);
```

- `device` is the logical device that destroys the buffer.
- `buffer` is the buffer to destroy.
- `pAllocator` controls host memory allocation as described in the Memory Allocation chapter.

### Valid Usage

- VUID-vkDestroyBuffer-buffer-00922
  All submitted commands that refer to `buffer`, either directly or via a `VkBufferView`, **must** have completed execution
- VUID-vkDestroyBuffer-buffer-00923
  If `VkAllocationCallbacks` were provided when `buffer` was created, a compatible set of callbacks **must** be provided here
- VUID-vkDestroyBuffer-buffer-00924
  If no `VkAllocationCallbacks` were provided when `buffer` was created, `pAllocator` **must** be `NULL`

### Valid Usage (Implicit)

- VUID-vkDestroyBuffer-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkDestroyBuffer-buffer-parameter
  If `buffer` is not VK_NULL_HANDLE, `buffer` **must** be a valid VkBuffer handle
- VUID-vkDestroyBuffer-pAllocator-parameter
  If `pAllocator` is not `NULL`, `pAllocator` **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkDestroyBuffer-buffer-parent
  If `buffer` is a valid handle, it **must** have been created, allocated, or retrieved from `device`

### Host Synchronization

- Host access to `buffer` **must** be externally synchronized

## 12.2. Buffer Views

A *buffer view* represents a contiguous range of a buffer and a specific format to be used to interpret the data. Buffer views are used to enable shaders to access buffer contents using image operations. In order to create a valid buffer view, the buffer **must** have been created with at least one of the following usage flags:

- `VK_BUFFER_USAGE_UNIFORM_TEXEL_BUFFER_BIT`
- `VK_BUFFER_USAGE_STORAGE_TEXEL_BUFFER_BIT`

Buffer views are represented by VkBufferView handles:

```
// Provided by VK_VERSION_1_0
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkBufferView)
```

To create a buffer view, call:

```
// Provided by VK_VERSION_1_0
VkResult vkCreateBufferView(
    VkDevice                                    device,
    const VkBufferViewCreateInfo*               pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkBufferView*                               pView);
```

- device is the logical device that creates the buffer view.
- pCreateInfo is a pointer to a VkBufferViewCreateInfo structure containing parameters to be used to create the buffer view.
- pAllocator controls host memory allocation as described in the Memory Allocation chapter.
- pView is a pointer to a VkBufferView handle in which the resulting buffer view object is returned.

## Valid Usage

- VUID-vkCreateBufferView-device-09665
  device **must** support at least one queue family with one of the VK_QUEUE_COMPUTE_BIT or VK_QUEUE_GRAPHICS_BIT capabilities

## Valid Usage (Implicit)

- VUID-vkCreateBufferView-device-parameter
  device **must** be a valid VkDevice handle
- VUID-vkCreateBufferView-pCreateInfo-parameter
  pCreateInfo **must** be a valid pointer to a valid VkBufferViewCreateInfo structure
- VUID-vkCreateBufferView-pAllocator-parameter
  If pAllocator is not NULL, pAllocator **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkCreateBufferView-pView-parameter
  pView **must** be a valid pointer to a VkBufferView handle

## Return Codes

**Success**

- VK_SUCCESS

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY

The VkBufferViewCreateInfo structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkBufferViewCreateInfo {
    VkStructureType            sType;
    const void*                pNext;
    VkBufferViewCreateFlags    flags;
    VkBuffer                   buffer;
    VkFormat                   format;
    VkDeviceSize               offset;
    VkDeviceSize               range;
} VkBufferViewCreateInfo;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- flags is reserved for future use.
- buffer is a VkBuffer on which the view will be created.
- format is a VkFormat describing the format of the data elements in the buffer.
- offset is an offset in bytes from the base address of the buffer. Accesses to the buffer view from shaders use addressing that is relative to this starting offset.
- range is a size in bytes of the buffer view. If range is equal to VK_WHOLE_SIZE, the range from offset to the end of the buffer is used. If VK_WHOLE_SIZE is used and the remaining size of the buffer is not a multiple of the texel block size of format, the nearest smaller multiple is used.

The buffer view has a *buffer view usage* identifying which descriptor types can be created from it. This usage **can** be defined by including the VkBufferUsageFlags2CreateInfoKHR structure in the pNext chain, and specifying the usage value there. If this structure is not included, it is equal to the VkBufferCreateInfo::usage value used to create buffer.

## Valid Usage

- VUID-VkBufferViewCreateInfo-offset-00925
  offset **must** be less than the size of buffer
- VUID-VkBufferViewCreateInfo-range-00928

If range is not equal to VK_WHOLE_SIZE, range **must** be greater than 0

- VUID-VkBufferViewCreateInfo-range-00929
  If range is not equal to VK_WHOLE_SIZE, range **must** be an integer multiple of the texel block size of format
- VUID-VkBufferViewCreateInfo-range-00930
  If range is not equal to VK_WHOLE_SIZE, the number of texel buffer elements given by (⌊range / (texel block size)⌋ × (texels per block)) where texel block size and texels per block are as defined in the Compatible Formats table for format, **must** be less than or equal to VkPhysicalDeviceLimits::maxTexelBufferElements
- VUID-VkBufferViewCreateInfo-offset-00931
  If range is not equal to VK_WHOLE_SIZE, the sum of offset and range **must** be less than or equal to the size of buffer
- VUID-VkBufferViewCreateInfo-range-04059
  If range is equal to VK_WHOLE_SIZE, the number of texel buffer elements given by (⌊(size - offset) / (texel block size)⌋ × (texels per block)) where size is the size of buffer, and texel block size and texels per block are as defined in the Compatible Formats table for format, **must** be less than or equal to VkPhysicalDeviceLimits::maxTexelBufferElements
- VUID-VkBufferViewCreateInfo-buffer-00932
  buffer **must** have been created with a usage value containing at least one of VK_BUFFER_USAGE_UNIFORM_TEXEL_BUFFER_BIT or VK_BUFFER_USAGE_STORAGE_TEXEL_BUFFER_BIT
- VUID-VkBufferViewCreateInfo-format-08778
  If the buffer view usage contains VK_BUFFER_USAGE_UNIFORM_TEXEL_BUFFER_BIT, then format features of format **must** contain VK_FORMAT_FEATURE_UNIFORM_TEXEL_BUFFER_BIT
- VUID-VkBufferViewCreateInfo-format-08779
  If the buffer view usage contains VK_BUFFER_USAGE_STORAGE_TEXEL_BUFFER_BIT, then format features of format **must** contain VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_BIT
- VUID-VkBufferViewCreateInfo-buffer-00935
  If buffer is non-sparse then it **must** be bound completely and contiguously to a single VkDeviceMemory object
- VUID-VkBufferViewCreateInfo-offset-02749
  If the texelBufferAlignment feature is not enabled, offset **must** be a multiple of VkPhysicalDeviceLimits::minTexelBufferOffsetAlignment
- VUID-VkBufferViewCreateInfo-buffer-02750
  If the texelBufferAlignment feature is enabled and if buffer was created with usage containing VK_BUFFER_USAGE_STORAGE_TEXEL_BUFFER_BIT, offset **must** be a multiple of the lesser of VkPhysicalDeviceTexelBufferAlignmentProperties::storageTexelBufferOffsetAlignmentBytes or, if VkPhysicalDeviceTexelBufferAlignmentProperties::storageTexelBufferOffsetSingleTexelAlignment is VK_TRUE, the size of a texel of the requested format. If the size of a texel is a multiple of three bytes, then the size of a single component of format is used instead
- VUID-VkBufferViewCreateInfo-buffer-02751
  If the texelBufferAlignment feature is enabled and if buffer was created with usage containing VK_BUFFER_USAGE_UNIFORM_TEXEL_BUFFER_BIT, offset **must** be a multiple of the

lesser of VkPhysicalDeviceTexelBufferAlignmentProperties::uniformTexelBufferOffsetAlignmentBytes or, if VkPhysicalDeviceTexelBufferAlignmentProperties::uniformTexelBufferOffsetSingleTexelAlignment is VK_TRUE, the size of a texel of the requested format. If the size of a texel is a multiple of three bytes, then the size of a single component of format is used instead

- VUID-VkBufferViewCreateInfo-pNext-06782
  If the pNext chain includes a VkExportMetalObjectCreateInfoEXT structure, its exportObjectType member **must** be VK_EXPORT_METAL_OBJECT_TYPE_METAL_TEXTURE_BIT_EXT
- VUID-VkBufferViewCreateInfo-pNext-08780
  If the pNext chain includes a VkBufferUsageFlags2CreateInfoKHR, its usage **must** not contain any other bit than VK_BUFFER_USAGE_2_UNIFORM_TEXEL_BUFFER_BIT_KHR or VK_BUFFER_USAGE_2_STORAGE_TEXEL_BUFFER_BIT_KHR
- VUID-VkBufferViewCreateInfo-pNext-08781
  If the pNext chain includes a VkBufferUsageFlags2CreateInfoKHR, its usage **must** be a subset of the VkBufferCreateInfo::usage specified or VkBufferUsageFlags2CreateInfoKHR::usage from VkBufferCreateInfo::pNext when creating buffer

### Valid Usage (Implicit)

- VUID-VkBufferViewCreateInfo-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_BUFFER_VIEW_CREATE_INFO
- VUID-VkBufferViewCreateInfo-pNext-pNext
  Each pNext member of any structure (including this one) in the pNext chain **must** be either NULL or a pointer to a valid instance of VkBufferUsageFlags2CreateInfoKHR or VkExportMetalObjectCreateInfoEXT
- VUID-VkBufferViewCreateInfo-sType-unique
  The sType value of each struct in the pNext chain **must** be unique, with the exception of structures of type VkExportMetalObjectCreateInfoEXT
- VUID-VkBufferViewCreateInfo-flags-zerobitmask
  flags **must** be 0
- VUID-VkBufferViewCreateInfo-buffer-parameter
  buffer **must** be a valid VkBuffer handle
- VUID-VkBufferViewCreateInfo-format-parameter
  format **must** be a valid VkFormat value

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkBufferViewCreateFlags;
```

VkBufferViewCreateFlags is a bitmask type for setting a mask, but is currently reserved for future use.

To destroy a buffer view, call:

```
// Provided by VK_VERSION_1_0
void vkDestroyBufferView(
    VkDevice                                    device,
    VkBufferView                                bufferView,
    const VkAllocationCallbacks*                pAllocator);
```

- `device` is the logical device that destroys the buffer view.
- `bufferView` is the buffer view to destroy.
- `pAllocator` controls host memory allocation as described in the Memory Allocation chapter.

### Valid Usage

- VUID-vkDestroyBufferView-bufferView-00936
  All submitted commands that refer to `bufferView` **must** have completed execution
- VUID-vkDestroyBufferView-bufferView-00937
  If `VkAllocationCallbacks` were provided when `bufferView` was created, a compatible set of callbacks **must** be provided here
- VUID-vkDestroyBufferView-bufferView-00938
  If no `VkAllocationCallbacks` were provided when `bufferView` was created, `pAllocator` **must** be `NULL`

### Valid Usage (Implicit)

- VUID-vkDestroyBufferView-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkDestroyBufferView-bufferView-parameter
  If `bufferView` is not VK_NULL_HANDLE, `bufferView` **must** be a valid VkBufferView handle
- VUID-vkDestroyBufferView-pAllocator-parameter
  If `pAllocator` is not `NULL`, `pAllocator` **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkDestroyBufferView-bufferView-parent
  If `bufferView` is a valid handle, it **must** have been created, allocated, or retrieved from `device`

### Host Synchronization

- Host access to `bufferView` **must** be externally synchronized

## 12.2.1. Buffer View Format Features

Valid uses of a VkBufferView **may** depend on the buffer view's *format features*, defined below. Such

constraints are documented in the affected valid usage statement.

- If Vulkan 1.3 is supported or the VK_KHR_format_feature_flags2 extension is supported, then the buffer view's set of *format features* is the value of VkFormatProperties3::bufferFeatures found by calling vkGetPhysicalDeviceFormatProperties2 on the same format as VkBufferViewCreateInfo::format.

## 12.3. Images

Images represent multidimensional - up to 3 - arrays of data which **can** be used for various purposes (e.g. attachments, textures), by binding them to a graphics or compute pipeline via descriptor sets, or by directly specifying them as parameters to certain commands.

Images are represented by VkImage handles:

```
// Provided by VK_VERSION_1_0
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkImage)
```

To create images, call:

```
// Provided by VK_VERSION_1_0
VkResult vkCreateImage(
    VkDevice                                    device,
    const VkImageCreateInfo*                    pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkImage*                                    pImage);
```

- device is the logical device that creates the image.
- pCreateInfo is a pointer to a VkImageCreateInfo structure containing parameters to be used to create the image.
- pAllocator controls host memory allocation as described in the Memory Allocation chapter.
- pImage is a pointer to a VkImage handle in which the resulting image object is returned.

### Valid Usage

- VUID-vkCreateImage-device-09666
  device **must** support at least one queue family with one of the VK_QUEUE_VIDEO_ENCODE_BIT_KHR, VK_QUEUE_VIDEO_DECODE_BIT_KHR, VK_QUEUE_OPTICAL_FLOW_BIT_NV, VK_QUEUE_SPARSE_BINDING_BIT, VK_QUEUE_TRANSFER_BIT, VK_QUEUE_COMPUTE_BIT, or VK_QUEUE_GRAPHICS_BIT capabilities
- VUID-vkCreateImage-flags-00939
  If the flags member of pCreateInfo includes VK_IMAGE_CREATE_SPARSE_BINDING_BIT, and the extendedSparseAddressSpace feature is not enabled, creating this VkImage **must** not cause the total required sparse memory for all currently valid sparse resources on the device to exceed VkPhysicalDeviceLimits::sparseAddressSpaceSize

- VUID-vkCreateImage-flags-09385
  If the flags member of pCreateInfo includes VK_IMAGE_CREATE_SPARSE_BINDING_BIT, the extendedSparseAddressSpace feature is enabled, and the usage member of pCreateInfo contains bits not in VkPhysicalDeviceExtendedSparseAddressSpacePropertiesNV::extendedSparseImageUsageFlags, creating this VkImage **must** not cause the total required sparse memory for all currently valid sparse resources on the device, excluding VkBuffer created with usage member of pCreateInfo containing bits in VkPhysicalDeviceExtendedSparseAddressSpacePropertiesNV::extendedSparseBufferUsageFlags and VkImage created with usage member of pCreateInfo containing bits in VkPhysicalDeviceExtendedSparseAddressSpacePropertiesNV::extendedSparseImageUsageFlags, to exceed VkPhysicalDeviceLimits::sparseAddressSpaceSize
- VUID-vkCreateImage-flags-09386
  If the flags member of pCreateInfo includes VK_IMAGE_CREATE_SPARSE_BINDING_BIT and the extendedSparseAddressSpace feature is enabled, creating this VkImage **must** not cause the total required sparse memory for all currently valid sparse resources on the device to exceed VkPhysicalDeviceExtendedSparseAddressSpacePropertiesNV::extendedSparseAddressSpaceSize
- VUID-vkCreateImage-pNext-06389
  If a VkBufferCollectionImageCreateInfoFUCHSIA has been chained to pNext, pCreateInfo **must** match the Sysmem chosen VkImageCreateInfo excepting members VkImageCreateInfo::extent and VkImageCreateInfo::usage in the match criteria

## Valid Usage (Implicit)

- VUID-vkCreateImage-device-parameter
  device **must** be a valid VkDevice handle
- VUID-vkCreateImage-pCreateInfo-parameter
  pCreateInfo **must** be a valid pointer to a valid VkImageCreateInfo structure
- VUID-vkCreateImage-pAllocator-parameter
  If pAllocator is not NULL, pAllocator **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkCreateImage-pImage-parameter
  pImage **must** be a valid pointer to a VkImage handle

## Return Codes

### Success

- VK_SUCCESS

### Failure

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY
- VK_ERROR_COMPRESSION_EXHAUSTED_EXT

- VK_ERROR_INVALID_OPAQUE_CAPTURE_ADDRESS_KHR

The VkImageCreateInfo structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkImageCreateInfo {
    VkStructureType          sType;
    const void*              pNext;
    VkImageCreateFlags       flags;
    VkImageType              imageType;
    VkFormat                 format;
    VkExtent3D               extent;
    uint32_t                 mipLevels;
    uint32_t                 arrayLayers;
    VkSampleCountFlagBits    samples;
    VkImageTiling            tiling;
    VkImageUsageFlags        usage;
    VkSharingMode            sharingMode;
    uint32_t                 queueFamilyIndexCount;
    const uint32_t*          pQueueFamilyIndices;
    VkImageLayout            initialLayout;
} VkImageCreateInfo;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- flags is a bitmask of VkImageCreateFlagBits describing additional parameters of the image.
- imageType is a VkImageType value specifying the basic dimensionality of the image. Layers in array textures do not count as a dimension for the purposes of the image type.
- format is a VkFormat describing the format and type of the texel blocks that will be contained in the image.
- extent is a VkExtent3D describing the number of data elements in each dimension of the base level.
- mipLevels describes the number of levels of detail available for minified sampling of the image.
- arrayLayers is the number of layers in the image.
- samples is a VkSampleCountFlagBits value specifying the number of samples per texel.
- tiling is a VkImageTiling value specifying the tiling arrangement of the texel blocks in memory.
- usage is a bitmask of VkImageUsageFlagBits describing the intended usage of the image.
- sharingMode is a VkSharingMode value specifying the sharing mode of the image when it will be accessed by multiple queue families.
- queueFamilyIndexCount is the number of entries in the pQueueFamilyIndices array.
- pQueueFamilyIndices is a pointer to an array of queue families that will access this image. It is ignored if sharingMode is not VK_SHARING_MODE_CONCURRENT.

- initialLayout is a VkImageLayout value specifying the initial VkImageLayout of all image subresources of the image. See Image Layouts.

Images created with tiling equal to VK_IMAGE_TILING_LINEAR have further restrictions on their limits and capabilities compared to images created with tiling equal to VK_IMAGE_TILING_OPTIMAL. Creation of images with tiling VK_IMAGE_TILING_LINEAR **may** not be supported unless other parameters meet all of the constraints:

- imageType is VK_IMAGE_TYPE_2D
- format is not a depth/stencil format
- mipLevels is 1
- arrayLayers is 1
- samples is VK_SAMPLE_COUNT_1_BIT
- usage only includes VK_IMAGE_USAGE_TRANSFER_SRC_BIT and/or VK_IMAGE_USAGE_TRANSFER_DST_BIT

Images created with one of the formats that require a sampler $Y'C_BC_R$ conversion, have further restrictions on their limits and capabilities compared to images created with other formats. Creation of images with a format requiring $Y'C_BC_R$ conversion **may** not be supported unless other parameters meet all of the constraints:

- imageType is VK_IMAGE_TYPE_2D
- mipLevels is 1
- arrayLayers is 1, unless the ycbcrImageArrays feature is enabled, or otherwise indicated by VkImageFormatProperties::maxArrayLayers, as returned by vkGetPhysicalDeviceImageFormatProperties
- samples is VK_SAMPLE_COUNT_1_BIT

Implementations **may** support additional limits and capabilities beyond those listed above.

To determine the set of valid usage bits for a given format, call vkGetPhysicalDeviceFormatProperties.

If the size of the resultant image would exceed maxResourceSize, then vkCreateImage **must** fail and return VK_ERROR_OUT_OF_DEVICE_MEMORY. This failure **may** occur even when all image creation parameters satisfy their valid usage requirements.

If the implementation reports VK_TRUE in VkPhysicalDeviceHostImageCopyPropertiesEXT::identicalMemoryTypeRequirements, usage of VK_IMAGE_USAGE_HOST_TRANSFER_BIT_EXT **must** not affect the memory type requirements of the image as described in Sparse Resource Memory Requirements and Resource Memory Association.

**NOTE**

For images created without VK_IMAGE_CREATE_EXTENDED_USAGE_BIT a usage bit is valid if it is supported for the format the image is created with.

For images created with VK_IMAGE_CREATE_EXTENDED_USAGE_BIT a usage bit is valid if it is supported for at least one of the formats a VkImageView created from the image

**can** have (see Image Views for more detail).

## Image Creation Limits

Valid values for some image creation parameters are limited by a numerical upper bound or by inclusion in a bitset. For example, VkImageCreateInfo::`arrayLayers` is limited by `imageCreateMaxArrayLayers`, defined below; and VkImageCreateInfo::`samples` is limited by `imageCreateSampleCounts`, also defined below.

Several limiting values are defined below, as well as assisting values from which the limiting values are derived. The limiting values are referenced by the relevant valid usage statements of VkImageCreateInfo.

- Let `uint64_t imageCreateDrmFormatModifiers[]` be the set of Linux DRM format modifiers that the resultant image **may** have.
  - If `tiling` is not `VK_IMAGE_TILING_DRM_FORMAT_MODIFIER_EXT`, then `imageCreateDrmFormatModifiers` is empty.
  - If VkImageCreateInfo::`pNext` contains VkImageDrmFormatModifierExplicitCreateInfoEXT, then `imageCreateDrmFormatModifiers` contains exactly one modifier, VkImageDrmFormatModifierExplicitCreateInfoEXT::`drmFormatModifier`.
  - If VkImageCreateInfo::`pNext` contains VkImageDrmFormatModifierListCreateInfoEXT, then `imageCreateDrmFormatModifiers` contains the entire array VkImageDrmFormatModifierListCreateInfoEXT::`pDrmFormatModifiers`.
- Let `VkBool32 imageCreateMaybeLinear` indicate if the resultant image may be linear.
  - If `tiling` is `VK_IMAGE_TILING_LINEAR`, then `imageCreateMaybeLinear` is `VK_TRUE`.
  - If `tiling` is `VK_IMAGE_TILING_OPTIMAL`, then `imageCreateMaybeLinear` is `VK_FALSE`.
  - If `tiling` is `VK_IMAGE_TILING_DRM_FORMAT_MODIFIER_EXT`, then `imageCreateMaybeLinear` is `VK_TRUE` if and only if `imageCreateDrmFormatModifiers` contains `DRM_FORMAT_MOD_LINEAR`.
- Let `VkFormatFeatureFlags imageCreateFormatFeatures` be the set of valid *format features* available during image creation.
  - If `tiling` is `VK_IMAGE_TILING_LINEAR`, then `imageCreateFormatFeatures` is the value of VkFormatProperties::`linearTilingFeatures` found by calling vkGetPhysicalDeviceFormatProperties with parameter `format` equal to VkImageCreateInfo::`format`.
  - If `tiling` is `VK_IMAGE_TILING_OPTIMAL`, and if the `pNext` chain includes no VkExternalFormatANDROID or VkExternalFormatQNX structure with non-zero `externalFormat`, then `imageCreateFormatFeatures` is the value of VkFormatProperties::`optimalTilingFeatures` found by calling vkGetPhysicalDeviceFormatProperties with parameter `format` equal to VkImageCreateInfo::`format`.
  - If `tiling` is `VK_IMAGE_TILING_OPTIMAL`, and if the `pNext` chain includes a VkExternalFormatANDROID structure with non-zero `externalFormat`, then `imageCreateFormatFeatures` is the value of

VkAndroidHardwareBufferFormatPropertiesANDROID::formatFeatures obtained by vkGetAndroidHardwareBufferPropertiesANDROID with a matching externalFormat value.

  - If tiling is VK_IMAGE_TILING_OPTIMAL, and if the pNext chain includes a VkExternalFormatQNX structure with non-zero externalFormat, then imageCreateFormatFeatures is the value of VkScreenBufferFormatPropertiesQNX::formatFeatures obtained by vkGetScreenBufferPropertiesQNX with a matching externalFormat value.
  - If the pNext chain includes a VkBufferCollectionImageCreateInfoFUCHSIA structure, then imageCreateFormatFeatures is the value of VkBufferCollectionPropertiesFUCHSIA::formatFeatures found by calling vkGetBufferCollectionPropertiesFUCHSIA with a parameter collection equal to VkBufferCollectionImageCreateInfoFUCHSIA::collection
  - If tiling is VK_IMAGE_TILING_DRM_FORMAT_MODIFIER_EXT, then the value of imageCreateFormatFeatures is found by calling vkGetPhysicalDeviceFormatProperties2 with VkImageFormatProperties::format equal to VkImageCreateInfo::format and with VkDrmFormatModifierPropertiesListEXT chained into VkFormatProperties2; by collecting all members of the returned array VkDrmFormatModifierPropertiesListEXT::pDrmFormatModifierProperties whose drmFormatModifier belongs to imageCreateDrmFormatModifiers; and by taking the bitwise intersection, over the collected array members, of drmFormatModifierTilingFeatures. (The resultant imageCreateFormatFeatures **may** be empty).

- Let VkImageFormatProperties2 imageCreateImageFormatPropertiesList[] be defined as follows.
  - If VkImageCreateInfo::pNext contains no VkExternalFormatANDROID or VkExternalFormatQNX structure with non-zero externalFormat, then imageCreateImageFormatPropertiesList is the list of structures obtained by calling vkGetPhysicalDeviceImageFormatProperties2, possibly multiple times, as follows:
    - The parameters VkPhysicalDeviceImageFormatInfo2::format, imageType, tiling, usage, and flags **must** be equal to those in VkImageCreateInfo.
    - If VkImageCreateInfo::pNext contains a VkExternalMemoryImageCreateInfo structure whose handleTypes is not 0, then VkPhysicalDeviceImageFormatInfo2::pNext **must** contain a VkPhysicalDeviceExternalImageFormatInfo structure whose handleType is not 0; and vkGetPhysicalDeviceImageFormatProperties2 **must** be called for each handle type in VkExternalMemoryImageCreateInfo::handleTypes, successively setting VkPhysicalDeviceExternalImageFormatInfo::handleType on each call.
    - If VkImageCreateInfo::pNext contains no VkExternalMemoryImageCreateInfo structure, or contains a structure whose handleTypes is 0, then VkPhysicalDeviceImageFormatInfo2::pNext **must** either contain no VkPhysicalDeviceExternalImageFormatInfo structure, or contain a structure whose handleType is 0.
    - If VkImageCreateInfo::pNext contains a VkVideoProfileListInfoKHR structure then VkPhysicalDeviceImageFormatInfo2::pNext **must** also contain the same

VkVideoProfileListInfoKHR structure on each call.

- If `tiling` is `VK_IMAGE_TILING_DRM_FORMAT_MODIFIER_EXT`, then:
  - VkPhysicalDeviceImageFormatInfo2::`pNext` **must** contain a VkPhysicalDeviceImageDrmFormatModifierInfoEXT structure where `sharingMode` is equal to VkImageCreateInfo::`sharingMode`;
  - if `sharingMode` is `VK_SHARING_MODE_CONCURRENT`, then `queueFamilyIndexCount` and `pQueueFamilyIndices` **must** be equal to those in VkImageCreateInfo;
  - if `flags` contains `VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT`, then the VkImageFormatListCreateInfo structure included in the `pNext` chain of VkPhysicalDeviceImageFormatInfo2 **must** be equivalent to the one included in the `pNext` chain of VkImageCreateInfo;
  - if VkImageCreateInfo::`pNext` contains a VkImageCompressionControlEXT structure, then the VkPhysicalDeviceImageFormatInfo2::`pNext` chain **must** contain an equivalent structure;
  - vkGetPhysicalDeviceImageFormatProperties2 **must** be called for each modifier in `imageCreateDrmFormatModifiers`, successively setting VkPhysicalDeviceImageDrmFormatModifierInfoEXT::`drmFormatModifier` on each call.
- If `tiling` is not `VK_IMAGE_TILING_DRM_FORMAT_MODIFIER_EXT`, then VkPhysicalDeviceImageFormatInfo2::`pNext` **must** contain no VkPhysicalDeviceImageDrmFormatModifierInfoEXT structure.
- If any call to vkGetPhysicalDeviceImageFormatProperties2 returns an error, then `imageCreateImageFormatPropertiesList` is defined to be the empty list.

◦ If VkImageCreateInfo::`pNext` contains a VkExternalFormatANDROID structure with non-zero `externalFormat`, then `imageCreateImageFormatPropertiesList` contains a single element where:

- VkImageFormatProperties::`maxMipLevels` is $\lfloor \log_2(\max(\texttt{extent.width}, \texttt{extent.height}, \texttt{extent.depth})) \rfloor + 1$.
- VkImageFormatProperties::`maxArrayLayers` is VkPhysicalDeviceLimits::`maxImageArrayLayers`.
- Each component of VkImageFormatProperties::`maxExtent` is VkPhysicalDeviceLimits::`maxImageDimension2D`.
- VkImageFormatProperties::`sampleCounts` contains exactly `VK_SAMPLE_COUNT_1_BIT`.

- Let `uint32_t` `imageCreateMaxMipLevels` be the minimum value of VkImageFormatProperties::`maxMipLevels` in `imageCreateImageFormatPropertiesList`. The value is undefined if `imageCreateImageFormatPropertiesList` is empty.
- Let `uint32_t` `imageCreateMaxArrayLayers` be the minimum value of VkImageFormatProperties::`maxArrayLayers` in `imageCreateImageFormatPropertiesList`. The value is undefined if `imageCreateImageFormatPropertiesList` is empty.
- Let `VkExtent3D` `imageCreateMaxExtent` be the component-wise minimum over all VkImageFormatProperties::`maxExtent` values in `imageCreateImageFormatPropertiesList`. The

value is undefined if imageCreateImageFormatPropertiesList is empty.

- Let VkSampleCountFlags imageCreateSampleCounts be the intersection of each VkImageFormatProperties::sampleCounts in imageCreateImageFormatPropertiesList. The value is undefined if imageCreateImageFormatPropertiesList is empty.
- Let VkVideoFormatPropertiesKHR videoFormatProperties[] be defined as follows.
  - If VkImageCreateInfo::pNext contains a VkVideoProfileListInfoKHR structure, then videoFormatProperties is the list of structures obtained by calling vkGetPhysicalDeviceVideoFormatPropertiesKHR with VkPhysicalDeviceVideoFormatInfoKHR::imageUsage equal to the usage member of VkImageCreateInfo and VkPhysicalDeviceVideoFormatInfoKHR::pNext containing the same VkVideoProfileListInfoKHR structure chained to VkImageCreateInfo.
  - If VkImageCreateInfo::pNext contains no VkVideoProfileListInfoKHR structure, then videoFormatProperties is an empty list.
- Let VkBool32 supportedVideoFormat indicate if the image parameters are supported by the specified video profiles.
  - supportedVideoFormat is VK_TRUE if there exists an element in the videoFormatProperties list for which all of the following conditions are true:
    - VkImageCreateInfo::format equals VkVideoFormatPropertiesKHR::format.
    - VkImageCreateInfo::flags only contains bits also set in VkVideoFormatPropertiesKHR::imageCreateFlags.
    - VkImageCreateInfo::imageType equals VkVideoFormatPropertiesKHR::imageType.
    - VkImageCreateInfo::tiling equals VkVideoFormatPropertiesKHR::imageTiling.
    - VkImageCreateInfo::usage only contains bits also set in VkVideoFormatPropertiesKHR::imageUsageFlags, or VkImageCreateInfo::flags includes VK_IMAGE_CREATE_EXTENDED_USAGE_BIT.
  - Otherwise supportedVideoFormat is VK_FALSE.

## Valid Usage

- VUID-VkImageCreateInfo-imageCreateMaxMipLevels-02251
  Each of the following values (as described in Image Creation Limits) **must** not be undefined : imageCreateMaxMipLevels, imageCreateMaxArrayLayers, imageCreateMaxExtent, and imageCreateSampleCounts
- VUID-VkImageCreateInfo-sharingMode-00941
  If sharingMode is VK_SHARING_MODE_CONCURRENT, pQueueFamilyIndices **must** be a valid pointer to an array of queueFamilyIndexCount uint32_t values
- VUID-VkImageCreateInfo-sharingMode-00942
  If sharingMode is VK_SHARING_MODE_CONCURRENT, queueFamilyIndexCount **must** be greater than 1
- VUID-VkImageCreateInfo-sharingMode-01420
  If sharingMode is VK_SHARING_MODE_CONCURRENT, each element of pQueueFamilyIndices **must** be

unique and **must** be less than `pQueueFamilyPropertyCount` returned by either vkGetPhysicalDeviceQueueFamilyProperties or vkGetPhysicalDeviceQueueFamilyProperties2 for the `physicalDevice` that was used to create `device`

- VUID-VkImageCreateInfo-pNext-01974
  If the `pNext` chain includes a VkExternalFormatANDROID structure, and its `externalFormat` member is non-zero the `format` **must** be `VK_FORMAT_UNDEFINED`
- VUID-VkImageCreateInfo-pNext-01975
  If the `pNext` chain does not include a VkExternalFormatANDROID structure, or does and its `externalFormat` member is `0`, the `format` **must** not be `VK_FORMAT_UNDEFINED`
- VUID-VkImageCreateInfo-extent-00944
  `extent.width` **must** be greater than `0`
- VUID-VkImageCreateInfo-extent-00945
  `extent.height` **must** be greater than `0`
- VUID-VkImageCreateInfo-extent-00946
  `extent.depth` **must** be greater than `0`
- VUID-VkImageCreateInfo-mipLevels-00947
  `mipLevels` **must** be greater than `0`
- VUID-VkImageCreateInfo-arrayLayers-00948
  `arrayLayers` **must** be greater than `0`
- VUID-VkImageCreateInfo-flags-00949
  If `flags` contains `VK_IMAGE_CREATE_CUBE_COMPATIBLE_BIT`, `imageType` **must** be `VK_IMAGE_TYPE_2D`
- VUID-VkImageCreateInfo-flags-08865
  If `flags` contains `VK_IMAGE_CREATE_CUBE_COMPATIBLE_BIT`, `extent.width` and `extent.height` **must** be equal
- VUID-VkImageCreateInfo-flags-08866
  If `flags` contains `VK_IMAGE_CREATE_CUBE_COMPATIBLE_BIT`, `arrayLayers` **must** be greater than or equal to 6
- VUID-VkImageCreateInfo-flags-02557
  If `flags` contains `VK_IMAGE_USAGE_FRAGMENT_DENSITY_MAP_BIT_EXT`, `imageType` **must** be `VK_IMAGE_TYPE_2D`
- VUID-VkImageCreateInfo-flags-00950
  If `flags` contains `VK_IMAGE_CREATE_2D_ARRAY_COMPATIBLE_BIT`, `imageType` **must** be `VK_IMAGE_TYPE_3D`
- VUID-VkImageCreateInfo-flags-09403
  If `flags` contains `VK_IMAGE_CREATE_2D_ARRAY_COMPATIBLE_BIT`, `flags` **must** not include `VK_IMAGE_CREATE_SPARSE_ALIASED_BIT`, `VK_IMAGE_CREATE_SPARSE_BINDING_BIT`, or `VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT`
- VUID-VkImageCreateInfo-flags-07755
  If `flags` contains `VK_IMAGE_CREATE_2D_VIEW_COMPATIBLE_BIT_EXT`, `imageType` **must** be `VK_IMAGE_TYPE_3D`

- VUID-VkImageCreateInfo-extent-02252
  `extent.width` **must** be less than or equal to `imageCreateMaxExtent.width` (as defined in Image Creation Limits)
- VUID-VkImageCreateInfo-extent-02253
  `extent.height` **must** be less than or equal to `imageCreateMaxExtent.height` (as defined in Image Creation Limits)
- VUID-VkImageCreateInfo-extent-02254
  `extent.depth` **must** be less than or equal to `imageCreateMaxExtent.depth` (as defined in Image Creation Limits)
- VUID-VkImageCreateInfo-imageType-00956
  If `imageType` is `VK_IMAGE_TYPE_1D`, both `extent.height` and `extent.depth` **must** be `1`
- VUID-VkImageCreateInfo-imageType-00957
  If `imageType` is `VK_IMAGE_TYPE_2D`, `extent.depth` **must** be `1`
- VUID-VkImageCreateInfo-mipLevels-00958
  `mipLevels` **must** be less than or equal to the number of levels in the complete mipmap chain based on `extent.width`, `extent.height`, and `extent.depth`
- VUID-VkImageCreateInfo-mipLevels-02255
  `mipLevels` **must** be less than or equal to `imageCreateMaxMipLevels` (as defined in Image Creation Limits)
- VUID-VkImageCreateInfo-arrayLayers-02256
  `arrayLayers` **must** be less than or equal to `imageCreateMaxArrayLayers` (as defined in Image Creation Limits)
- VUID-VkImageCreateInfo-imageType-00961
  If `imageType` is `VK_IMAGE_TYPE_3D`, `arrayLayers` **must** be `1`
- VUID-VkImageCreateInfo-samples-02257
  If `samples` is not `VK_SAMPLE_COUNT_1_BIT`, then `imageType` **must** be `VK_IMAGE_TYPE_2D`, `flags` **must** not contain `VK_IMAGE_CREATE_CUBE_COMPATIBLE_BIT`, `mipLevels` **must** be equal to `1`, and `imageCreateMaybeLinear` (as defined in Image Creation Limits) **must** be `VK_FALSE`,
- VUID-VkImageCreateInfo-samples-02558
  If `samples` is not `VK_SAMPLE_COUNT_1_BIT`, `usage` **must** not contain `VK_IMAGE_USAGE_FRAGMENT_DENSITY_MAP_BIT_EXT`
- VUID-VkImageCreateInfo-usage-00963
  If `usage` includes `VK_IMAGE_USAGE_TRANSIENT_ATTACHMENT_BIT`, then bits other than `VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT`, `VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT`, and `VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT` **must** not be set
- VUID-VkImageCreateInfo-usage-00964
  If `usage` includes `VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT`, `VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT`, `VK_IMAGE_USAGE_TRANSIENT_ATTACHMENT_BIT`, or `VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT`, `extent.width` **must** be less than or equal to `VkPhysicalDeviceLimits::maxFramebufferWidth`
- VUID-VkImageCreateInfo-usage-00965
  If `usage` includes `VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT`, `VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT`, `VK_IMAGE_USAGE_TRANSIENT_ATTACHMENT_BIT`,

or VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT, extent.height **must** be less than or equal to VkPhysicalDeviceLimits::maxFramebufferHeight

- VUID-VkImageCreateInfo-fragmentDensityMapOffset-06514
  If the fragmentDensityMapOffset feature is not enabled and usage includes VK_IMAGE_USAGE_FRAGMENT_DENSITY_MAP_BIT_EXT, extent.width **must** be less than or equal to $\lceil \frac{maxFramebufferWidth}{minFragmentDensityTexelSize_{width}} \rceil$
- VUID-VkImageCreateInfo-fragmentDensityMapOffset-06515
  If the fragmentDensityMapOffset feature is not enabled and usage includes VK_IMAGE_USAGE_FRAGMENT_DENSITY_MAP_BIT_EXT, extent.height **must** be less than or equal to $\lceil \frac{maxFramebufferHeight}{minFragmentDensityTexelSize_{height}} \rceil$
- VUID-VkImageCreateInfo-usage-00966
  If usage includes VK_IMAGE_USAGE_TRANSIENT_ATTACHMENT_BIT, usage **must** also contain at least one of VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT, VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT, or VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT
- VUID-VkImageCreateInfo-samples-02258
  samples **must** be a valid VkSampleCountFlagBits value that is set in imageCreateSampleCounts (as defined in Image Creation Limits)
- VUID-VkImageCreateInfo-usage-00968
  If the shaderStorageImageMultisample feature is not enabled, and usage contains VK_IMAGE_USAGE_STORAGE_BIT, samples **must** be VK_SAMPLE_COUNT_1_BIT
- VUID-VkImageCreateInfo-flags-00969
  If the sparseBinding feature is not enabled, flags **must** not contain VK_IMAGE_CREATE_SPARSE_BINDING_BIT
- VUID-VkImageCreateInfo-flags-01924
  If the sparseResidencyAliased feature is not enabled, flags **must** not contain VK_IMAGE_CREATE_SPARSE_ALIASED_BIT
- VUID-VkImageCreateInfo-tiling-04121
  If tiling is VK_IMAGE_TILING_LINEAR, flags **must** not contain VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT
- VUID-VkImageCreateInfo-imageType-00970
  If imageType is VK_IMAGE_TYPE_1D, flags **must** not contain VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT
- VUID-VkImageCreateInfo-imageType-00971
  If the sparseResidencyImage2D feature is not enabled, and imageType is VK_IMAGE_TYPE_2D, flags **must** not contain VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT
- VUID-VkImageCreateInfo-imageType-00972
  If the sparseResidencyImage3D feature is not enabled, and imageType is VK_IMAGE_TYPE_3D, flags **must** not contain VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT
- VUID-VkImageCreateInfo-imageType-00973
  If the sparseResidency2Samples feature is not enabled, imageType is VK_IMAGE_TYPE_2D, and samples is VK_SAMPLE_COUNT_2_BIT, flags **must** not contain VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT
- VUID-VkImageCreateInfo-imageType-00974

If the sparseResidency4Samples feature is not enabled, imageType is VK_IMAGE_TYPE_2D, and samples is VK_SAMPLE_COUNT_4_BIT, flags **must** not contain VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT

- VUID-VkImageCreateInfo-imageType-00975
  If the sparseResidency8Samples feature is not enabled, imageType is VK_IMAGE_TYPE_2D, and samples is VK_SAMPLE_COUNT_8_BIT, flags **must** not contain VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT
- VUID-VkImageCreateInfo-imageType-00976
  If the sparseResidency16Samples feature is not enabled, imageType is VK_IMAGE_TYPE_2D, and samples is VK_SAMPLE_COUNT_16_BIT, flags **must** not contain VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT
- VUID-VkImageCreateInfo-flags-00987
  If flags contains VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT or VK_IMAGE_CREATE_SPARSE_ALIASED_BIT, it **must** also contain VK_IMAGE_CREATE_SPARSE_BINDING_BIT
- VUID-VkImageCreateInfo-None-01925
  If any of the bits VK_IMAGE_CREATE_SPARSE_BINDING_BIT, VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT, or VK_IMAGE_CREATE_SPARSE_ALIASED_BIT are set, VK_IMAGE_USAGE_TRANSIENT_ATTACHMENT_BIT **must** not also be set
- VUID-VkImageCreateInfo-flags-01890
  If the protectedMemory feature is not enabled, flags **must** not contain VK_IMAGE_CREATE_PROTECTED_BIT
- VUID-VkImageCreateInfo-None-01891
  If any of the bits VK_IMAGE_CREATE_SPARSE_BINDING_BIT, VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT, or VK_IMAGE_CREATE_SPARSE_ALIASED_BIT are set, VK_IMAGE_CREATE_PROTECTED_BIT **must** not also be set
- VUID-VkImageCreateInfo-pNext-00988
  If the pNext chain includes a VkExternalMemoryImageCreateInfoNV structure, it **must** not contain a VkExternalMemoryImageCreateInfo structure
- VUID-VkImageCreateInfo-pNext-00990
  If the pNext chain includes a VkExternalMemoryImageCreateInfo structure, its handleTypes member **must** only contain bits that are also in VkExternalImageFormatProperties::externalMemoryProperties.compatibleHandleTypes, as returned by vkGetPhysicalDeviceImageFormatProperties2 with format, imageType, tiling, usage, and flags equal to those in this structure, and with a VkPhysicalDeviceExternalImageFormatInfo structure included in the pNext chain, with a handleType equal to any one of the handle types specified in VkExternalMemoryImageCreateInfo::handleTypes
- VUID-VkImageCreateInfo-pNext-00991
  If the pNext chain includes a VkExternalMemoryImageCreateInfoNV structure, its handleTypes member **must** only contain bits that are also in VkExternalImageFormatPropertiesNV::externalMemoryProperties.compatibleHandleTypes, as returned by vkGetPhysicalDeviceExternalImageFormatPropertiesNV with format, imageType, tiling, usage, and flags equal to those in this structure, and with

externalHandleType equal to any one of the handle types specified in VkExternalMemoryImageCreateInfoNV::handleTypes

- VUID-VkImageCreateInfo-physicalDeviceCount-01421
  If the logical device was created with VkDeviceGroupDeviceCreateInfo::physicalDeviceCount equal to 1, flags **must** not contain VK_IMAGE_CREATE_SPLIT_INSTANCE_BIND_REGIONS_BIT
- VUID-VkImageCreateInfo-flags-02259
  If flags contains VK_IMAGE_CREATE_SPLIT_INSTANCE_BIND_REGIONS_BIT, then mipLevels **must** be one, arrayLayers **must** be one, imageType **must** be VK_IMAGE_TYPE_2D. and imageCreateMaybeLinear (as defined in Image Creation Limits) **must** be VK_FALSE
- VUID-VkImageCreateInfo-flags-01572
  If flags contains VK_IMAGE_CREATE_BLOCK_TEXEL_VIEW_COMPATIBLE_BIT, then format **must** be a compressed image format
- VUID-VkImageCreateInfo-flags-01573
  If flags contains VK_IMAGE_CREATE_BLOCK_TEXEL_VIEW_COMPATIBLE_BIT, then flags **must** also contain VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT
- VUID-VkImageCreateInfo-initialLayout-00993
  initialLayout **must** be VK_IMAGE_LAYOUT_UNDEFINED or VK_IMAGE_LAYOUT_PREINITIALIZED
- VUID-VkImageCreateInfo-pNext-01443
  If the pNext chain includes a VkExternalMemoryImageCreateInfo or VkExternalMemoryImageCreateInfoNV structure whose handleTypes member is not 0, initialLayout **must** be VK_IMAGE_LAYOUT_UNDEFINED
- VUID-VkImageCreateInfo-format-06410
  If the image format is one of the formats that require a sampler $Y'C_BC_R$ conversion, mipLevels **must** be 1
- VUID-VkImageCreateInfo-format-06411
  If the image format is one of the formats that require a sampler $Y'C_BC_R$ conversion, samples **must** be VK_SAMPLE_COUNT_1_BIT
- VUID-VkImageCreateInfo-format-06412
  If the image format is one of the formats that require a sampler $Y'C_BC_R$ conversion, imageType **must** be VK_IMAGE_TYPE_2D
- VUID-VkImageCreateInfo-imageCreateFormatFeatures-02260
  If format is a *multi-planar* format, and if imageCreateFormatFeatures (as defined in Image Creation Limits) does not contain VK_FORMAT_FEATURE_DISJOINT_BIT, then flags **must** not contain VK_IMAGE_CREATE_DISJOINT_BIT
- VUID-VkImageCreateInfo-format-01577
  If format is not a *multi-planar* format, and flags does not include VK_IMAGE_CREATE_ALIAS_BIT, flags **must** not contain VK_IMAGE_CREATE_DISJOINT_BIT
- VUID-VkImageCreateInfo-format-04712
  If format has a _422 or _420 suffix, extent.width **must** be a multiple of 2
- VUID-VkImageCreateInfo-format-04713
  If format has a _420 suffix, extent.height **must** be a multiple of 2

- VUID-VkImageCreateInfo-format-09583
  If `format` is one of the `VK_FORMAT_PVTRC1_*_IMG` formats, `extent.width` **must** be a power of 2
- VUID-VkImageCreateInfo-format-09584
  If `format` is one of the `VK_FORMAT_PVTRC1_*_IMG` formats, `extent.height` **must** be a power of 2
- VUID-VkImageCreateInfo-tiling-02261
  If `tiling` is `VK_IMAGE_TILING_DRM_FORMAT_MODIFIER_EXT`, then the `pNext` chain **must** include exactly one of VkImageDrmFormatModifierListCreateInfoEXT or VkImageDrmFormatModifierExplicitCreateInfoEXT structures
- VUID-VkImageCreateInfo-pNext-02262
  If the `pNext` chain includes a VkImageDrmFormatModifierListCreateInfoEXT or VkImageDrmFormatModifierExplicitCreateInfoEXT structure, then `tiling` **must** be `VK_IMAGE_TILING_DRM_FORMAT_MODIFIER_EXT`
- VUID-VkImageCreateInfo-tiling-02353
  If `tiling` is `VK_IMAGE_TILING_DRM_FORMAT_MODIFIER_EXT` and `flags` contains `VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT`, then the `pNext` chain **must** include a VkImageFormatListCreateInfo structure with non-zero `viewFormatCount`
- VUID-VkImageCreateInfo-flags-01533
  If `flags` contains `VK_IMAGE_CREATE_SAMPLE_LOCATIONS_COMPATIBLE_DEPTH_BIT_EXT` `format` **must** be a depth or depth/stencil format
- VUID-VkImageCreateInfo-pNext-02393
  If the `pNext` chain includes a VkExternalMemoryImageCreateInfo structure whose `handleTypes` member includes `VK_EXTERNAL_MEMORY_HANDLE_TYPE_ANDROID_HARDWARE_BUFFER_BIT_ANDROID`, `imageType` **must** be `VK_IMAGE_TYPE_2D`
- VUID-VkImageCreateInfo-pNext-02394
  If the `pNext` chain includes a VkExternalMemoryImageCreateInfo structure whose `handleTypes` member includes `VK_EXTERNAL_MEMORY_HANDLE_TYPE_ANDROID_HARDWARE_BUFFER_BIT_ANDROID`, `mipLevels` **must** either be `1` or equal to the number of levels in the complete mipmap chain based on `extent.width`, `extent.height`, and `extent.depth`
- VUID-VkImageCreateInfo-pNext-02396
  If the `pNext` chain includes a VkExternalFormatANDROID structure whose `externalFormat` member is not `0`, `flags` **must** not include `VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT`
- VUID-VkImageCreateInfo-pNext-02397
  If the `pNext` chain includes a VkExternalFormatANDROID structure whose `externalFormat` member is not `0`, `usage` **must** not include any usages except `VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT`, `VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT`, or `VK_IMAGE_USAGE_SAMPLED_BIT`
- VUID-VkImageCreateInfo-pNext-09457
  If the `pNext` chain includes a VkExternalFormatANDROID structure whose `externalFormat` member is not `0`, and `externalFormatResolve` feature is not enabled, `usage` **must** not include `VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT` or `VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT`
- VUID-VkImageCreateInfo-pNext-02398

If the pNext chain includes a VkExternalFormatANDROID structure whose externalFormat member is not 0, tiling **must** be VK_IMAGE_TILING_OPTIMAL

- VUID-VkImageCreateInfo-pNext-08951
  If the pNext chain includes a VkExternalMemoryImageCreateInfo structure whose handleTypes member includes VK_EXTERNAL_MEMORY_HANDLE_TYPE_SCREEN_BUFFER_BIT_QNX, imageType **must** be VK_IMAGE_TYPE_2D
- VUID-VkImageCreateInfo-pNext-08952
  If the pNext chain includes a VkExternalMemoryImageCreateInfo structure whose handleTypes member includes VK_EXTERNAL_MEMORY_HANDLE_TYPE_SCREEN_BUFFER_BIT_QNX, mipLevels **must** either be 1 or equal to the number of levels in the complete mipmap chain based on extent.width, extent.height, and extent.depth
- VUID-VkImageCreateInfo-pNext-08953
  If the pNext chain includes a VkExternalFormatQNX structure whose externalFormat member is not 0, flags **must** not include VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT
- VUID-VkImageCreateInfo-pNext-08954
  If the pNext chain includes a VkExternalFormatQNX structure whose externalFormat member is not 0, usage **must** not include any usages except VK_IMAGE_USAGE_SAMPLED_BIT
- VUID-VkImageCreateInfo-pNext-08955
  If the pNext chain includes a VkExternalFormatQNX structure whose externalFormat member is not 0, tiling **must** be VK_IMAGE_TILING_OPTIMAL
- VUID-VkImageCreateInfo-format-02795
  If format is a depth-stencil format, usage includes VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT, and the pNext chain includes a VkImageStencilUsageCreateInfo structure, then its VkImageStencilUsageCreateInfo::stencilUsage member **must** also include VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT
- VUID-VkImageCreateInfo-format-02796
  If format is a depth-stencil format, usage does not include VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT, and the pNext chain includes a VkImageStencilUsageCreateInfo structure, then its VkImageStencilUsageCreateInfo::stencilUsage member **must** also not include VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT
- VUID-VkImageCreateInfo-format-02797
  If format is a depth-stencil format, usage includes VK_IMAGE_USAGE_TRANSIENT_ATTACHMENT_BIT, and the pNext chain includes a VkImageStencilUsageCreateInfo structure, then its VkImageStencilUsageCreateInfo::stencilUsage member **must** also include VK_IMAGE_USAGE_TRANSIENT_ATTACHMENT_BIT
- VUID-VkImageCreateInfo-format-02798
  If format is a depth-stencil format, usage does not include VK_IMAGE_USAGE_TRANSIENT_ATTACHMENT_BIT, and the pNext chain includes a VkImageStencilUsageCreateInfo structure, then its VkImageStencilUsageCreateInfo::stencilUsage member **must** also not include VK_IMAGE_USAGE_TRANSIENT_ATTACHMENT_BIT
- VUID-VkImageCreateInfo-Format-02536
  If Format is a depth-stencil format and the pNext chain includes a VkImageStencilUsageCreateInfo structure with its stencilUsage member including

VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT, extent.width **must** be less than or equal to VkPhysicalDeviceLimits::maxFramebufferWidth

- VUID-VkImageCreateInfo-format-02537
  If format is a depth-stencil format and the pNext chain includes a VkImageStencilUsageCreateInfo structure with its stencilUsage member including VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT, extent.height **must** be less than or equal to VkPhysicalDeviceLimits::maxFramebufferHeight
- VUID-VkImageCreateInfo-format-02538
  If the shaderStorageImageMultisample feature is not enabled, format is a depth-stencil format and the pNext chain includes a VkImageStencilUsageCreateInfo structure with its stencilUsage including VK_IMAGE_USAGE_STORAGE_BIT, samples **must** be VK_SAMPLE_COUNT_1_BIT
- VUID-VkImageCreateInfo-flags-02050
  If flags contains VK_IMAGE_CREATE_CORNER_SAMPLED_BIT_NV, imageType **must** be VK_IMAGE_TYPE_2D or VK_IMAGE_TYPE_3D
- VUID-VkImageCreateInfo-flags-02051
  If flags contains VK_IMAGE_CREATE_CORNER_SAMPLED_BIT_NV, it **must** not contain VK_IMAGE_CREATE_CUBE_COMPATIBLE_BIT and the format **must** not be a depth/stencil format
- VUID-VkImageCreateInfo-flags-02052
  If flags contains VK_IMAGE_CREATE_CORNER_SAMPLED_BIT_NV and imageType is VK_IMAGE_TYPE_2D, extent.width and extent.height **must** be greater than 1
- VUID-VkImageCreateInfo-flags-02053
  If flags contains VK_IMAGE_CREATE_CORNER_SAMPLED_BIT_NV and imageType is VK_IMAGE_TYPE_3D, extent.width, extent.height, and extent.depth **must** be greater than 1
- VUID-VkImageCreateInfo-imageType-02082
  If usage includes VK_IMAGE_USAGE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR, imageType **must** be VK_IMAGE_TYPE_2D
- VUID-VkImageCreateInfo-samples-02083
  If usage includes VK_IMAGE_USAGE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR, samples **must** be VK_SAMPLE_COUNT_1_BIT
- VUID-VkImageCreateInfo-shadingRateImage-07727
  If the shadingRateImage feature is enabled and usage includes VK_IMAGE_USAGE_SHADING_RATE_IMAGE_BIT_NV, tiling **must** be VK_IMAGE_TILING_OPTIMAL
- VUID-VkImageCreateInfo-flags-02565
  If flags contains VK_IMAGE_CREATE_SUBSAMPLED_BIT_EXT, tiling **must** be VK_IMAGE_TILING_OPTIMAL
- VUID-VkImageCreateInfo-flags-02566
  If flags contains VK_IMAGE_CREATE_SUBSAMPLED_BIT_EXT, imageType **must** be VK_IMAGE_TYPE_2D
- VUID-VkImageCreateInfo-flags-02567
  If flags contains VK_IMAGE_CREATE_SUBSAMPLED_BIT_EXT, flags **must** not contain VK_IMAGE_CREATE_CUBE_COMPATIBLE_BIT
- VUID-VkImageCreateInfo-flags-02568
  If flags contains VK_IMAGE_CREATE_SUBSAMPLED_BIT_EXT, mipLevels **must** be 1

- VUID-VkImageCreateInfo-usage-04992
  If usage includes VK_IMAGE_USAGE_INVOCATION_MASK_BIT_HUAWEI, tiling **must** be VK_IMAGE_TILING_LINEAR
- VUID-VkImageCreateInfo-imageView2DOn3DImage-04459
  If the VK_KHR_portability_subset extension is enabled, and VkPhysicalDevicePortabilitySubsetFeaturesKHR::imageView2DOn3DImage is VK_FALSE, flags **must** not contain VK_IMAGE_CREATE_2D_ARRAY_COMPATIBLE_BIT
- VUID-VkImageCreateInfo-multisampleArrayImage-04460
  If the VK_KHR_portability_subset extension is enabled, and VkPhysicalDevicePortabilitySubsetFeaturesKHR::multisampleArrayImage is VK_FALSE, and samples is not VK_SAMPLE_COUNT_1_BIT, then arrayLayers **must** be 1
- VUID-VkImageCreateInfo-pNext-06722
  If a VkImageFormatListCreateInfo structure was included in the pNext chain and format is not a multi-planar format and VkImageFormatListCreateInfo::viewFormatCount is not zero, then each format in VkImageFormatListCreateInfo::pViewFormats **must** either be compatible with the format as described in the compatibility table or, if flags contains VK_IMAGE_CREATE_BLOCK_TEXEL_VIEW_COMPATIBLE_BIT, be an uncompressed format that is size-compatible with format
- VUID-VkImageCreateInfo-pNext-10062
  If a VkImageFormatListCreateInfo structure was included in the pNext chain and format is a multi-planar format and flags contains VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT and VkImageFormatListCreateInfo::viewFormatCount is not zero, then each format in VkImageFormatListCreateInfo::pViewFormats **must** be compatible with the VkFormat for the plane of the image format
- VUID-VkImageCreateInfo-flags-04738
  If flags does not contain VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT and the pNext chain includes a VkImageFormatListCreateInfo structure, then VkImageFormatListCreateInfo::viewFormatCount **must** be 0 or 1
- VUID-VkImageCreateInfo-usage-04815
  If usage includes VK_IMAGE_USAGE_VIDEO_DECODE_SRC_BIT_KHR, VK_IMAGE_USAGE_VIDEO_DECODE_DST_BIT_KHR, or VK_IMAGE_USAGE_VIDEO_DECODE_DPB_BIT_KHR, and flags does not include VK_IMAGE_CREATE_VIDEO_PROFILE_INDEPENDENT_BIT_KHR, then the pNext chain **must** include a VkVideoProfileListInfoKHR structure with profileCount greater than 0 and pProfiles including at least one VkVideoProfileInfoKHR structure with a videoCodecOperation member specifying a decode operation
- VUID-VkImageCreateInfo-usage-04816
  If usage includes VK_IMAGE_USAGE_VIDEO_ENCODE_SRC_BIT_KHR, VK_IMAGE_USAGE_VIDEO_ENCODE_DST_BIT_KHR, or VK_IMAGE_USAGE_VIDEO_ENCODE_DPB_BIT_KHR, and flags does not include VK_IMAGE_CREATE_VIDEO_PROFILE_INDEPENDENT_BIT_KHR, then the pNext chain **must** include a VkVideoProfileListInfoKHR structure with profileCount greater than 0 and pProfiles including at least one VkVideoProfileInfoKHR structure with a videoCodecOperation member specifying an encode operation
- VUID-VkImageCreateInfo-flags-08328
  If flags includes VK_IMAGE_CREATE_VIDEO_PROFILE_INDEPENDENT_BIT_KHR, then videoMaintenance1 **must** be enabled

- VUID-VkImageCreateInfo-flags-08329
  If `flags` includes `VK_IMAGE_CREATE_VIDEO_PROFILE_INDEPENDENT_BIT_KHR` and `usage` does not include `VK_IMAGE_USAGE_VIDEO_DECODE_DST_BIT_KHR`, then `usage` **must** not include `VK_IMAGE_USAGE_VIDEO_DECODE_DPB_BIT_KHR`
- VUID-VkImageCreateInfo-flags-08331
  If `flags` includes `VK_IMAGE_CREATE_VIDEO_PROFILE_INDEPENDENT_BIT_KHR`, then `usage` **must** not include `VK_IMAGE_USAGE_VIDEO_ENCODE_DPB_BIT_KHR`
- VUID-VkImageCreateInfo-pNext-06811
  If the `pNext` chain includes a VkVideoProfileListInfoKHR structure with `profileCount` greater than `0`, then `supportedVideoFormat` **must** be `VK_TRUE`
- VUID-VkImageCreateInfo-pNext-06390
  If the VkImage is to be used to import memory from a VkBufferCollectionFUCHSIA, a VkBufferCollectionImageCreateInfoFUCHSIA structure **must** be chained to `pNext`
- VUID-VkImageCreateInfo-multisampledRenderToSingleSampled-06882
  If the `multisampledRenderToSingleSampled` feature is not enabled, `flags` **must** not contain `VK_IMAGE_CREATE_MULTISAMPLED_RENDER_TO_SINGLE_SAMPLED_BIT_EXT`
- VUID-VkImageCreateInfo-flags-06883
  If `flags` contains `VK_IMAGE_CREATE_MULTISAMPLED_RENDER_TO_SINGLE_SAMPLED_BIT_EXT`, `samples` **must** be `VK_SAMPLE_COUNT_1_BIT`
- VUID-VkImageCreateInfo-pNext-06743
  If the `pNext` chain includes a VkImageCompressionControlEXT structure, `format` is a multi-planar format, and VkImageCompressionControlEXT::`flags` includes `VK_IMAGE_COMPRESSION_FIXED_RATE_EXPLICIT_EXT`, then VkImageCompressionControlEXT::`compressionControlPlaneCount` **must** be equal to the number of planes in `format`
- VUID-VkImageCreateInfo-pNext-06744
  If the `pNext` chain includes a VkImageCompressionControlEXT structure, `format` is not a multi-planar format, and VkImageCompressionControlEXT::`flags` includes `VK_IMAGE_COMPRESSION_FIXED_RATE_EXPLICIT_EXT`, then VkImageCompressionControlEXT::`compressionControlPlaneCount` **must** be 1
- VUID-VkImageCreateInfo-pNext-06746
  If the `pNext` chain includes a VkImageCompressionControlEXT structure, it **must** not contain a VkImageDrmFormatModifierExplicitCreateInfoEXT structure
- VUID-VkImageCreateInfo-flags-08104
  If `flags` includes `VK_IMAGE_CREATE_DESCRIPTOR_BUFFER_CAPTURE_REPLAY_BIT_EXT`, the `descriptorBufferCaptureReplay` feature **must** be enabled
- VUID-VkImageCreateInfo-pNext-08105
  If the `pNext` chain includes a VkOpaqueCaptureDescriptorDataCreateInfoEXT structure, `flags` **must** contain `VK_IMAGE_CREATE_DESCRIPTOR_BUFFER_CAPTURE_REPLAY_BIT_EXT`
- VUID-VkImageCreateInfo-pNext-06783
  If the `pNext` chain includes a VkExportMetalObjectCreateInfoEXT structure, its `exportObjectType` member **must** be either `VK_EXPORT_METAL_OBJECT_TYPE_METAL_TEXTURE_BIT_EXT` or `VK_EXPORT_METAL_OBJECT_TYPE_METAL_IOSURFACE_BIT_EXT`

- VUID-VkImageCreateInfo-pNext-06784
  If the pNext chain includes a VkImportMetalTextureInfoEXT structure its plane member **must** be VK_IMAGE_ASPECT_PLANE_0_BIT, VK_IMAGE_ASPECT_PLANE_1_BIT, or VK_IMAGE_ASPECT_PLANE_2_BIT
- VUID-VkImageCreateInfo-pNext-06785
  If the pNext chain includes a VkImportMetalTextureInfoEXT structure and the image does not have a multi-planar format, then VkImportMetalTextureInfoEXT::plane **must** be VK_IMAGE_ASPECT_PLANE_0_BIT
- VUID-VkImageCreateInfo-pNext-06786
  If the pNext chain includes a VkImportMetalTextureInfoEXT structure and the image has a multi-planar format with only two planes, then VkImportMetalTextureInfoEXT::plane **must** not be VK_IMAGE_ASPECT_PLANE_2_BIT
- VUID-VkImageCreateInfo-imageCreateFormatFeatures-09048
  If imageCreateFormatFeatures (as defined in Image Creation Limits) does not contain VK_FORMAT_FEATURE_2_HOST_IMAGE_TRANSFER_BIT_EXT, then usage **must** not contain VK_IMAGE_USAGE_HOST_TRANSFER_BIT_EXT
- VUID-VkImageCreateInfo-pNext-09653
  If the pNext chain contains a VkImageAlignmentControlCreateInfoMESA structure, tiling **must** be VK_IMAGE_TILING_OPTIMAL
- VUID-VkImageCreateInfo-pNext-09654
  If the pNext chain contains a VkImageAlignmentControlCreateInfoMESA structure, it **must** not contain a VkExternalMemoryImageCreateInfo structure

## Valid Usage (Implicit)

- VUID-VkImageCreateInfo-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_IMAGE_CREATE_INFO
- VUID-VkImageCreateInfo-pNext-pNext
  Each pNext member of any structure (including this one) in the pNext chain **must** be either NULL or a pointer to a valid instance of VkBufferCollectionImageCreateInfoFUCHSIA, VkDedicatedAllocationImageCreateInfoNV, VkExportMetalObjectCreateInfoEXT, VkExternalFormatANDROID, VkExternalFormatQNX, VkExternalMemoryImageCreateInfo, VkExternalMemoryImageCreateInfoNV, VkImageAlignmentControlCreateInfoMESA, VkImageCompressionControlEXT, VkImageDrmFormatModifierExplicitCreateInfoEXT, VkImageDrmFormatModifierListCreateInfoEXT, VkImageFormatListCreateInfo, VkImageStencilUsageCreateInfo, VkImageSwapchainCreateInfoKHR, VkImportMetalIOSurfaceInfoEXT, VkImportMetalTextureInfoEXT, VkOpaqueCaptureDescriptorDataCreateInfoEXT, VkOpticalFlowImageFormatInfoNV, or VkVideoProfileListInfoKHR
- VUID-VkImageCreateInfo-sType-unique
  The sType value of each struct in the pNext chain **must** be unique, with the exception of structures of type VkExportMetalObjectCreateInfoEXT or VkImportMetalTextureInfoEXT

- VUID-VkImageCreateInfo-flags-parameter
  `flags` **must** be a valid combination of VkImageCreateFlagBits values
- VUID-VkImageCreateInfo-imageType-parameter
  `imageType` **must** be a valid VkImageType value
- VUID-VkImageCreateInfo-format-parameter
  `format` **must** be a valid VkFormat value
- VUID-VkImageCreateInfo-samples-parameter
  `samples` **must** be a valid VkSampleCountFlagBits value
- VUID-VkImageCreateInfo-tiling-parameter
  `tiling` **must** be a valid VkImageTiling value
- VUID-VkImageCreateInfo-usage-parameter
  `usage` **must** be a valid combination of VkImageUsageFlagBits values
- VUID-VkImageCreateInfo-usage-requiredbitmask
  `usage` **must** not be `0`
- VUID-VkImageCreateInfo-sharingMode-parameter
  `sharingMode` **must** be a valid VkSharingMode value
- VUID-VkImageCreateInfo-initialLayout-parameter
  `initialLayout` **must** be a valid VkImageLayout value

The `VkBufferCollectionImageCreateInfoFUCHSIA` structure is defined as:

```
// Provided by VK_FUCHSIA_buffer_collection
typedef struct VkBufferCollectionImageCreateInfoFUCHSIA {
    VkStructureType              sType;
    const void*                  pNext;
    VkBufferCollectionFUCHSIA    collection;
    uint32_t                     index;
} VkBufferCollectionImageCreateInfoFUCHSIA;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure
- `collection` is the VkBufferCollectionFUCHSIA handle
- `index` is the index of the buffer in the buffer collection from which the memory will be imported

### Valid Usage

- VUID-VkBufferCollectionImageCreateInfoFUCHSIA-index-06391
  `index` **must** be less than VkBufferCollectionPropertiesFUCHSIA::`bufferCount`

### Valid Usage (Implicit)

- VUID-VkBufferCollectionImageCreateInfoFUCHSIA-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_BUFFER_COLLECTION_IMAGE_CREATE_INFO_FUCHSIA`
- VUID-VkBufferCollectionImageCreateInfoFUCHSIA-collection-parameter
  `collection` **must** be a valid VkBufferCollectionFUCHSIA handle

The `VkImageStencilUsageCreateInfo` structure is defined as:

```
typedef struct VkImageStencilUsageCreateInfo {
    VkStructureType      sType;
    const void*          pNext;
    VkImageUsageFlags    stencilUsage;
} VkImageStencilUsageCreateInfo;
```

or the equivalent

```
// Provided by VK_EXT_separate_stencil_usage
typedef VkImageStencilUsageCreateInfo VkImageStencilUsageCreateInfoEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `stencilUsage` is a bitmask of VkImageUsageFlagBits describing the intended usage of the stencil aspect of the image.

If the `pNext` chain of VkImageCreateInfo includes a `VkImageStencilUsageCreateInfo` structure, then that structure includes the usage flags specific to the stencil aspect of the image for an image with a depth-stencil format.

This structure specifies image usages which only apply to the stencil aspect of a depth/stencil format image. When this structure is included in the `pNext` chain of VkImageCreateInfo, the stencil aspect of the image **must** only be used as specified by `stencilUsage`. When this structure is not included in the `pNext` chain of VkImageCreateInfo, the stencil aspect of an image **must** only be used as specified by VkImageCreateInfo::`usage`. Use of other aspects of an image are unaffected by this structure.

This structure **can** also be included in the `pNext` chain of VkPhysicalDeviceImageFormatInfo2 to query additional capabilities specific to image creation parameter combinations including a separate set of usage flags for the stencil aspect of the image using vkGetPhysicalDeviceImageFormatProperties2. When this structure is not included in the `pNext` chain of `VkPhysicalDeviceImageFormatInfo2` then the implicit value of `stencilUsage` matches that of `VkPhysicalDeviceImageFormatInfo2`::`usage`.

### Valid Usage

- VUID-VkImageStencilUsageCreateInfo-stencilUsage-02539
  If `stencilUsage` includes `VK_IMAGE_USAGE_TRANSIENT_ATTACHMENT_BIT`, it **must** not include bits other than `VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT` or `VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT`

### Valid Usage (Implicit)

- VUID-VkImageStencilUsageCreateInfo-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_IMAGE_STENCIL_USAGE_CREATE_INFO`
- VUID-VkImageStencilUsageCreateInfo-stencilUsage-parameter
  `stencilUsage` **must** be a valid combination of VkImageUsageFlagBits values
- VUID-VkImageStencilUsageCreateInfo-stencilUsage-requiredbitmask
  `stencilUsage` **must** not be `0`

If the `pNext` chain includes a `VkDedicatedAllocationImageCreateInfoNV` structure, then that structure includes an enable controlling whether the image will have a dedicated memory allocation bound to it.

The `VkDedicatedAllocationImageCreateInfoNV` structure is defined as:

```
// Provided by VK_NV_dedicated_allocation
typedef struct VkDedicatedAllocationImageCreateInfoNV {
    VkStructureType    sType;
    const void*        pNext;
    VkBool32           dedicatedAllocation;
} VkDedicatedAllocationImageCreateInfoNV;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `dedicatedAllocation` specifies whether the image will have a dedicated allocation bound to it.

**NOTE** Using a dedicated allocation for color and depth/stencil attachments or other large images **may** improve performance on some devices.

### Valid Usage

- VUID-VkDedicatedAllocationImageCreateInfoNV-dedicatedAllocation-00994
  If `dedicatedAllocation` is `VK_TRUE`, VkImageCreateInfo::`flags` **must** not include `VK_IMAGE_CREATE_SPARSE_BINDING_BIT`, `VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT`, or `VK_IMAGE_CREATE_SPARSE_ALIASED_BIT`

### Valid Usage (Implicit)

- VUID-VkDedicatedAllocationImageCreateInfoNV-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_DEDICATED_ALLOCATION_IMAGE_CREATE_INFO_NV`

To define a set of external memory handle types that **may** be used as backing store for an image, add a VkExternalMemoryImageCreateInfo structure to the `pNext` chain of the VkImageCreateInfo structure. The `VkExternalMemoryImageCreateInfo` structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkExternalMemoryImageCreateInfo {
    VkStructureType                     sType;
    const void*                         pNext;
    VkExternalMemoryHandleTypeFlags     handleTypes;
} VkExternalMemoryImageCreateInfo;
```

or the equivalent

```
// Provided by VK_KHR_external_memory
typedef VkExternalMemoryImageCreateInfo VkExternalMemoryImageCreateInfoKHR;
```

| | |
|---|---|
| **NOTE** | A `VkExternalMemoryImageCreateInfo` structure with a non-zero `handleTypes` field must be included in the creation parameters for an image that will be bound to memory that is either exported or imported. |

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `handleTypes` is zero or a bitmask of VkExternalMemoryHandleTypeFlagBits specifying one or more external memory handle types.

### Valid Usage (Implicit)

- VUID-VkExternalMemoryImageCreateInfo-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_EXTERNAL_MEMORY_IMAGE_CREATE_INFO`
- VUID-VkExternalMemoryImageCreateInfo-handleTypes-parameter
  `handleTypes` **must** be a valid combination of VkExternalMemoryHandleTypeFlagBits values

If the `pNext` chain includes a `VkExternalMemoryImageCreateInfoNV` structure, then that structure defines a set of external memory handle types that **may** be used as backing store for the image.

The `VkExternalMemoryImageCreateInfoNV` structure is defined as:

```
// Provided by VK_NV_external_memory
typedef struct VkExternalMemoryImageCreateInfoNV {
    VkStructureType                     sType;
    const void*                         pNext;
    VkExternalMemoryHandleTypeFlagsNV   handleTypes;
} VkExternalMemoryImageCreateInfoNV;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- handleTypes is zero or a bitmask of VkExternalMemoryHandleTypeFlagBitsNV specifying one or more external memory handle types.

### Valid Usage (Implicit)

- VUID-VkExternalMemoryImageCreateInfoNV-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_EXTERNAL_MEMORY_IMAGE_CREATE_INFO_NV
- VUID-VkExternalMemoryImageCreateInfoNV-handleTypes-parameter
  handleTypes **must** be a valid combination of VkExternalMemoryHandleTypeFlagBitsNV values

VkExternalFormatANDROID is defined as:

```
// Provided by VK_ANDROID_external_memory_android_hardware_buffer
typedef struct VkExternalFormatANDROID {
    VkStructureType    sType;
    void*              pNext;
    uint64_t           externalFormat;
} VkExternalFormatANDROID;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- externalFormat is an implementation-defined identifier for the external format

When included in the pNext chain of another structure, it indicates additional format information beyond what is provided by VkFormat values for an Android hardware buffer. If externalFormat is zero, it indicates that no external format is used, and implementations should rely only on other format information. If this structure is not present, it is equivalent to setting externalFormat to zero.

### Valid Usage

- VUID-VkExternalFormatANDROID-externalFormat-01894
  externalFormat **must** be 0 or a value returned in the externalFormat member of VkAndroidHardwareBufferFormatPropertiesANDROID by an earlier call to

vkGetAndroidHardwareBufferPropertiesANDROID

### Valid Usage (Implicit)

- VUID-VkExternalFormatANDROID-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_EXTERNAL_FORMAT_ANDROID`

To create an image with an QNX Screen external format, add a `VkExternalFormatQNX` structure in the `pNext` chain of VkImageCreateInfo. `VkExternalFormatQNX` is defined as:

```
// Provided by VK_QNX_external_memory_screen_buffer
typedef struct VkExternalFormatQNX {
    VkStructureType    sType;
    void*              pNext;
    uint64_t           externalFormat;
} VkExternalFormatQNX;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `externalFormat` is an implementation-defined identifier for the external format

If `externalFormat` is zero, the effect is as if the `VkExternalFormatQNX` structure was not present. Otherwise, the `image` will have the specified external format.

### Valid Usage

- VUID-VkExternalFormatQNX-externalFormat-08956
  `externalFormat` **must** be `0` or a value returned in the `externalFormat` member of VkScreenBufferFormatPropertiesQNX by an earlier call to vkGetScreenBufferPropertiesQNX

### Valid Usage (Implicit)

- VUID-VkExternalFormatQNX-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_EXTERNAL_FORMAT_QNX`

If the `pNext` chain of VkImageCreateInfo includes a `VkImageSwapchainCreateInfoKHR` structure, then that structure includes a swapchain handle indicating that the image will be bound to memory from that swapchain.

The `VkImageSwapchainCreateInfoKHR` structure is defined as:

```
// Provided by VK_VERSION_1_1 with VK_KHR_swapchain, VK_KHR_device_group with
VK_KHR_swapchain
typedef struct VkImageSwapchainCreateInfoKHR {
    VkStructureType    sType;
    const void*        pNext;
    VkSwapchainKHR     swapchain;
} VkImageSwapchainCreateInfoKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `swapchain` is VK_NULL_HANDLE or a handle of a swapchain that the image will be bound to.

### Valid Usage

- VUID-VkImageSwapchainCreateInfoKHR-swapchain-00995
  If `swapchain` is not VK_NULL_HANDLE, the fields of VkImageCreateInfo **must** match the implied image creation parameters of the swapchain

### Valid Usage (Implicit)

- VUID-VkImageSwapchainCreateInfoKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_IMAGE_SWAPCHAIN_CREATE_INFO_KHR`
- VUID-VkImageSwapchainCreateInfoKHR-swapchain-parameter
  If `swapchain` is not VK_NULL_HANDLE, `swapchain` **must** be a valid VkSwapchainKHR handle

If the `pNext` chain of VkImageCreateInfo includes a `VkImageFormatListCreateInfo` structure, then that structure contains a list of all formats that **can** be used when creating views of this image.

The `VkImageFormatListCreateInfo` structure is defined as:

```
typedef struct VkImageFormatListCreateInfo {
    VkStructureType    sType;
    const void*        pNext;
    uint32_t           viewFormatCount;
    const VkFormat*    pViewFormats;
} VkImageFormatListCreateInfo;
```

or the equivalent

```
// Provided by VK_KHR_image_format_list
typedef VkImageFormatListCreateInfo VkImageFormatListCreateInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- viewFormatCount is the number of entries in the pViewFormats array.
- pViewFormats is a pointer to an array of VkFormat values specifying all formats which **can** be used when creating views of this image.

If viewFormatCount is zero, pViewFormats is ignored and the image is created as if the VkImageFormatListCreateInfo structure were not included in the pNext chain of VkImageCreateInfo.

### Valid Usage

- VUID-VkImageFormatListCreateInfo-viewFormatCount-09540
  If viewFormatCount is not 0, each element of pViewFormats **must** not be VK_FORMAT_UNDEFINED

### Valid Usage (Implicit)

- VUID-VkImageFormatListCreateInfo-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_IMAGE_FORMAT_LIST_CREATE_INFO
- VUID-VkImageFormatListCreateInfo-pViewFormats-parameter
  If viewFormatCount is not 0, pViewFormats **must** be a valid pointer to an array of viewFormatCount valid VkFormat values

If the pNext chain of VkImageCreateInfo includes a VkImageDrmFormatModifierListCreateInfoEXT structure, then the image will be created with one of the Linux DRM format modifiers listed in the structure. The choice of modifier is implementation-dependent.

The VkImageDrmFormatModifierListCreateInfoEXT structure is defined as:

```
// Provided by VK_EXT_image_drm_format_modifier
typedef struct VkImageDrmFormatModifierListCreateInfoEXT {
    VkStructureType    sType;
    const void*        pNext;
    uint32_t           drmFormatModifierCount;
    const uint64_t*    pDrmFormatModifiers;
} VkImageDrmFormatModifierListCreateInfoEXT;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- drmFormatModifierCount is the length of the pDrmFormatModifiers array.
- pDrmFormatModifiers is a pointer to an array of *Linux DRM format modifiers*.

### Valid Usage

- VUID-VkImageDrmFormatModifierListCreateInfoEXT-pDrmFormatModifiers-02263
  Each *modifier* in `pDrmFormatModifiers` **must** be compatible with the parameters in VkImageCreateInfo and its `pNext` chain, as determined by querying VkPhysicalDeviceImageFormatInfo2 extended with VkPhysicalDeviceImageDrmFormatModifierInfoEXT

### Valid Usage (Implicit)

- VUID-VkImageDrmFormatModifierListCreateInfoEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_IMAGE_DRM_FORMAT_MODIFIER_LIST_CREATE_INFO_EXT`
- VUID-VkImageDrmFormatModifierListCreateInfoEXT-pDrmFormatModifiers-parameter
  `pDrmFormatModifiers` **must** be a valid pointer to an array of `drmFormatModifierCount` `uint64_t` values
- VUID-VkImageDrmFormatModifierListCreateInfoEXT-drmFormatModifierCount-arraylength
  `drmFormatModifierCount` **must** be greater than `0`

If the `pNext` chain of VkImageCreateInfo includes a VkImageDrmFormatModifierExplicitCreateInfoEXT structure, then the image will be created with the Linux DRM format modifier and memory layout defined by the structure.

The VkImageDrmFormatModifierExplicitCreateInfoEXT structure is defined as:

```
// Provided by VK_EXT_image_drm_format_modifier
typedef struct VkImageDrmFormatModifierExplicitCreateInfoEXT {
    VkStructureType                 sType;
    const void*                     pNext;
    uint64_t                        drmFormatModifier;
    uint32_t                        drmFormatModifierPlaneCount;
    const VkSubresourceLayout*      pPlaneLayouts;
} VkImageDrmFormatModifierExplicitCreateInfoEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `drmFormatModifier` is the *Linux DRM format modifier* with which the image will be created.
- `drmFormatModifierPlaneCount` is the number of *memory planes* in the image (as reported by VkDrmFormatModifierPropertiesEXT) as well as the length of the `pPlaneLayouts` array.
- `pPlaneLayouts` is a pointer to an array of VkSubresourceLayout structures describing the image's *memory planes*.

The $i^{th}$ member of `pPlaneLayouts` describes the layout of the image's $i^{th}$ *memory plane* (that is,

VK_IMAGE_ASPECT_MEMORY_PLANE_i_BIT_EXT). In each element of pPlaneLayouts, the implementation **must** ignore size. The implementation calculates the size of each plane, which the application **can** query with vkGetImageSubresourceLayout.

When creating an image with VkImageDrmFormatModifierExplicitCreateInfoEXT, it is the application's responsibility to satisfy all valid usage requirements. However, the implementation **must** validate that the provided pPlaneLayouts, when combined with the provided drmFormatModifier and other creation parameters in VkImageCreateInfo and its pNext chain, produce a valid image. (This validation is necessarily implementation-dependent and outside the scope of Vulkan, and therefore not described by valid usage requirements). If this validation fails, then vkCreateImage returns VK_ERROR_INVALID_DRM_FORMAT_MODIFIER_PLANE_LAYOUT_EXT.

### Valid Usage

- VUID-VkImageDrmFormatModifierExplicitCreateInfoEXT-drmFormatModifier-02264
  drmFormatModifier **must** be compatible with the parameters in VkImageCreateInfo and its pNext chain, as determined by querying VkPhysicalDeviceImageFormatInfo2 extended with VkPhysicalDeviceImageDrmFormatModifierInfoEXT
- VUID-VkImageDrmFormatModifierExplicitCreateInfoEXT-drmFormatModifierPlaneCount-02265
  drmFormatModifierPlaneCount **must** be equal to the VkDrmFormatModifierPropertiesEXT::drmFormatModifierPlaneCount associated with VkImageCreateInfo::format and drmFormatModifier, as found by querying VkDrmFormatModifierPropertiesListEXT
- VUID-VkImageDrmFormatModifierExplicitCreateInfoEXT-size-02267
  For each element of pPlaneLayouts, size **must** be 0
- VUID-VkImageDrmFormatModifierExplicitCreateInfoEXT-arrayPitch-02268
  For each element of pPlaneLayouts, arrayPitch **must** be 0 if VkImageCreateInfo::arrayLayers is 1
- VUID-VkImageDrmFormatModifierExplicitCreateInfoEXT-depthPitch-02269
  For each element of pPlaneLayouts, depthPitch **must** be 0 if VkImageCreateInfo::extent.depth is 1

### Valid Usage (Implicit)

- VUID-VkImageDrmFormatModifierExplicitCreateInfoEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_IMAGE_DRM_FORMAT_MODIFIER_EXPLICIT_CREATE_INFO_EXT
- VUID-VkImageDrmFormatModifierExplicitCreateInfoEXT-pPlaneLayouts-parameter
  pPlaneLayouts **must** be a valid pointer to an array of drmFormatModifierPlaneCount VkSubresourceLayout structures
- VUID-VkImageDrmFormatModifierExplicitCreateInfoEXT-drmFormatModifierPlaneCount-arraylength
  drmFormatModifierPlaneCount **must** be greater than 0

If the pNext list of VkImageCreateInfo includes a VkImageCompressionControlEXT structure, then that

structure describes compression controls for this image.

The VkImageCompressionControlEXT structure is defined as:

```
// Provided by VK_EXT_image_compression_control
typedef struct VkImageCompressionControlEXT {
    VkStructureType                         sType;
    const void*                             pNext;
    VkImageCompressionFlagsEXT              flags;
    uint32_t                                compressionControlPlaneCount;
    VkImageCompressionFixedRateFlagsEXT*    pFixedRateFlags;
} VkImageCompressionControlEXT;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- flags is a bitmask of VkImageCompressionFlagBitsEXT describing compression controls for the image.
- compressionControlPlaneCount is the number of entries in the pFixedRateFlags array.
- pFixedRateFlags is NULL or a pointer to an array of VkImageCompressionFixedRateFlagsEXT bitfields describing allowed fixed-rate compression rates of each image plane. It is ignored if flags does not include VK_IMAGE_COMPRESSION_FIXED_RATE_EXPLICIT_EXT.

If enabled, fixed-rate compression is done in an implementation-defined manner and **may** be applied at block granularity. In that case, a write to an individual texel **may** modify the value of other texels in the same block.

### Valid Usage

- VUID-VkImageCompressionControlEXT-flags-06747
  flags **must** be one of VK_IMAGE_COMPRESSION_DEFAULT_EXT, VK_IMAGE_COMPRESSION_FIXED_RATE_DEFAULT_EXT, VK_IMAGE_COMPRESSION_FIXED_RATE_EXPLICIT_EXT, or VK_IMAGE_COMPRESSION_DISABLED_EXT
- VUID-VkImageCompressionControlEXT-flags-06748
  If flags includes VK_IMAGE_COMPRESSION_FIXED_RATE_EXPLICIT_EXT, pFixedRateFlags **must** not be NULL

### Valid Usage (Implicit)

- VUID-VkImageCompressionControlEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_IMAGE_COMPRESSION_CONTROL_EXT

**NOTE** Some combinations of compression properties may not be supported. For example, some implementations may not support different fixed-rate compression rates per plane of a multi-planar format and will not be able to enable fixed-rate compression

for any plane if the requested rates differ.

Possible values of VkImageCompressionControlEXT::flags, specifying compression controls for an image, are:

```
// Provided by VK_EXT_image_compression_control
typedef enum VkImageCompressionFlagBitsEXT {
    VK_IMAGE_COMPRESSION_DEFAULT_EXT = 0,
    VK_IMAGE_COMPRESSION_FIXED_RATE_DEFAULT_EXT = 0x00000001,
    VK_IMAGE_COMPRESSION_FIXED_RATE_EXPLICIT_EXT = 0x00000002,
    VK_IMAGE_COMPRESSION_DISABLED_EXT = 0x00000004,
} VkImageCompressionFlagBitsEXT;
```

- VK_IMAGE_COMPRESSION_DEFAULT_EXT specifies that the default image compression setting is used. Implementations **must** not apply fixed-rate compression.
- VK_IMAGE_COMPRESSION_FIXED_RATE_DEFAULT_EXT specifies that the implementation **may** choose any supported fixed-rate compression setting in an implementation-defined manner based on the properties of the image.
- VK_IMAGE_COMPRESSION_FIXED_RATE_EXPLICIT_EXT specifies that fixed-rate compression **may** be used and that the allowed compression rates are specified by VkImageCompressionControlEXT::pFixedRateFlags.
- VK_IMAGE_COMPRESSION_DISABLED_EXT specifies that all lossless and fixed-rate compression **should** be disabled.

If VkImageCompressionControlEXT::flags is VK_IMAGE_COMPRESSION_FIXED_RATE_EXPLICIT_EXT, then the $i^{th}$ member of the pFixedRateFlags array specifies the allowed compression rates for the image's $i^{th}$ plane.

**NOTE** If VK_IMAGE_COMPRESSION_DISABLED_EXT is included in VkImageCompressionControlEXT::flags, both lossless and fixed-rate compression will be disabled. This is likely to have a negative impact on performance and is only intended to be used for debugging purposes.

```
// Provided by VK_EXT_image_compression_control
typedef VkFlags VkImageCompressionFlagsEXT;
```

VkImageCompressionFlagsEXT is a bitmask type for setting a mask of zero or more VkImageCompressionFlagBitsEXT.

```
// Provided by VK_EXT_image_compression_control
typedef VkFlags VkImageCompressionFixedRateFlagsEXT;
```

VkImageCompressionFixedRateFlagsEXT is a bitmask type for setting a mask of zero or more VkImageCompressionFixedRateFlagBitsEXT.

Bits which **can** be set in VkImageCompressionControlEXT::pFixedRateFlags, specifying allowed compression rates for an image plane, are:

```
// Provided by VK_EXT_image_compression_control
typedef enum VkImageCompressionFixedRateFlagBitsEXT {
    VK_IMAGE_COMPRESSION_FIXED_RATE_NONE_EXT = 0,
    VK_IMAGE_COMPRESSION_FIXED_RATE_1BPC_BIT_EXT = 0x00000001,
    VK_IMAGE_COMPRESSION_FIXED_RATE_2BPC_BIT_EXT = 0x00000002,
    VK_IMAGE_COMPRESSION_FIXED_RATE_3BPC_BIT_EXT = 0x00000004,
    VK_IMAGE_COMPRESSION_FIXED_RATE_4BPC_BIT_EXT = 0x00000008,
    VK_IMAGE_COMPRESSION_FIXED_RATE_5BPC_BIT_EXT = 0x00000010,
    VK_IMAGE_COMPRESSION_FIXED_RATE_6BPC_BIT_EXT = 0x00000020,
    VK_IMAGE_COMPRESSION_FIXED_RATE_7BPC_BIT_EXT = 0x00000040,
    VK_IMAGE_COMPRESSION_FIXED_RATE_8BPC_BIT_EXT = 0x00000080,
    VK_IMAGE_COMPRESSION_FIXED_RATE_9BPC_BIT_EXT = 0x00000100,
    VK_IMAGE_COMPRESSION_FIXED_RATE_10BPC_BIT_EXT = 0x00000200,
    VK_IMAGE_COMPRESSION_FIXED_RATE_11BPC_BIT_EXT = 0x00000400,
    VK_IMAGE_COMPRESSION_FIXED_RATE_12BPC_BIT_EXT = 0x00000800,
    VK_IMAGE_COMPRESSION_FIXED_RATE_13BPC_BIT_EXT = 0x00001000,
    VK_IMAGE_COMPRESSION_FIXED_RATE_14BPC_BIT_EXT = 0x00002000,
    VK_IMAGE_COMPRESSION_FIXED_RATE_15BPC_BIT_EXT = 0x00004000,
    VK_IMAGE_COMPRESSION_FIXED_RATE_16BPC_BIT_EXT = 0x00008000,
    VK_IMAGE_COMPRESSION_FIXED_RATE_17BPC_BIT_EXT = 0x00010000,
    VK_IMAGE_COMPRESSION_FIXED_RATE_18BPC_BIT_EXT = 0x00020000,
    VK_IMAGE_COMPRESSION_FIXED_RATE_19BPC_BIT_EXT = 0x00040000,
    VK_IMAGE_COMPRESSION_FIXED_RATE_20BPC_BIT_EXT = 0x00080000,
    VK_IMAGE_COMPRESSION_FIXED_RATE_21BPC_BIT_EXT = 0x00100000,
    VK_IMAGE_COMPRESSION_FIXED_RATE_22BPC_BIT_EXT = 0x00200000,
    VK_IMAGE_COMPRESSION_FIXED_RATE_23BPC_BIT_EXT = 0x00400000,
    VK_IMAGE_COMPRESSION_FIXED_RATE_24BPC_BIT_EXT = 0x00800000,
} VkImageCompressionFixedRateFlagBitsEXT;
```

- VK_IMAGE_COMPRESSION_FIXED_RATE_NONE_EXT specifies that fixed-rate compression **must** not be used.
- VK_IMAGE_COMPRESSION_FIXED_RATE_1BPC_BIT_EXT specifies that fixed-rate compression with a bitrate of [1,2) bits per component **may** be used.
- VK_IMAGE_COMPRESSION_FIXED_RATE_2BPC_BIT_EXT specifies that fixed-rate compression with a bitrate of [2,3) bits per component **may** be used.
- VK_IMAGE_COMPRESSION_FIXED_RATE_3BPC_BIT_EXT specifies that fixed-rate compression with a bitrate of [3,4) bits per component **may** be used.
- VK_IMAGE_COMPRESSION_FIXED_RATE_4BPC_BIT_EXT specifies that fixed-rate compression with a bitrate of [4,5) bits per component **may** be used.
- VK_IMAGE_COMPRESSION_FIXED_RATE_5BPC_BIT_EXT specifies that fixed-rate compression with a bitrate of [5,6) bits per component **may** be used.
- VK_IMAGE_COMPRESSION_FIXED_RATE_6BPC_BIT_EXT specifies that fixed-rate compression with a bitrate of [6,7) bits per component **may** be used.

- VK_IMAGE_COMPRESSION_FIXED_RATE_7BPC_BIT_EXT specifies that fixed-rate compression with a bitrate of [7,8) bits per component **may** be used.
- VK_IMAGE_COMPRESSION_FIXED_RATE_8BPC_BIT_EXT specifies that fixed-rate compression with a bitrate of [8,9) bits per component **may** be used.
- VK_IMAGE_COMPRESSION_FIXED_RATE_9BPC_BIT_EXT specifies that fixed-rate compression with a bitrate of [9,10) bits per component **may** be used.
- VK_IMAGE_COMPRESSION_FIXED_RATE_10BPC_BIT_EXT specifies that fixed-rate compression with a bitrate of [10,11) bits per component **may** be used.
- VK_IMAGE_COMPRESSION_FIXED_RATE_11BPC_BIT_EXT specifies that fixed-rate compression with a bitrate of [11,12) bits per component **may** be used.
- VK_IMAGE_COMPRESSION_FIXED_RATE_12BPC_BIT_EXT specifies that fixed-rate compression with a bitrate of at least 12 bits per component **may** be used.

If the format has a different bit rate for different components, VkImageCompressionControlEXT::pFixedRateFlags describes the rate of the component with the largest number of bits assigned to it, scaled pro rata. For example, to request that a VK_FORMAT_A2R10G10B10_UNORM_PACK32 format be stored at a rate of 8 bits per pixel, use VK_IMAGE_COMPRESSION_FIXED_RATE_2BPC_BIT_EXT (10 bits for the largest component, stored at quarter the original size, 2.5 bits, rounded down).

If flags includes VK_IMAGE_COMPRESSION_FIXED_RATE_EXPLICIT_EXT, and multiple bits are set in VkImageCompressionControlEXT::pFixedRateFlags for a plane, implementations **should** apply the lowest allowed bitrate that is supported.

**NOTE** The choice of "bits per component" terminology was chosen so that the same compression rate describes the same degree of compression applied to formats that differ only in the number of components. For example, VK_FORMAT_R8G8_UNORM compressed to half its original size is a rate of 4 bits per component, 8 bits per pixel. VK_FORMAT_R8G8B8A8_UNORM compressed to half *its* original size is 4 bits per component, 16 bits per pixel. Both of these cases can be requested with VK_IMAGE_COMPRESSION_FIXED_RATE_4BPC_BIT_EXT.

To query the compression properties of an image, add a VkImageCompressionPropertiesEXT structure to the pNext chain of the VkSubresourceLayout2EXT structure in a call to vkGetImageSubresourceLayout2KHR or vkGetImageSubresourceLayout2EXT.

To determine the compression rates that are supported for a given image format, add a VkImageCompressionPropertiesEXT structure to the pNext chain of the VkImageFormatProperties2 structure in a call to vkGetPhysicalDeviceImageFormatProperties2.

**NOTE** Since fixed-rate compression is disabled by default, the VkImageCompressionPropertiesEXT structure passed to vkGetPhysicalDeviceImageFormatProperties2 will not indicate any fixed-rate compression support unless a VkImageCompressionControlEXT structure is also included in the pNext chain of the VkPhysicalDeviceImageFormatInfo2 structure passed to the same command.

The VkImageCompressionPropertiesEXT structure is defined as:

```
// Provided by VK_EXT_image_compression_control
typedef struct VkImageCompressionPropertiesEXT {
    VkStructureType                         sType;
    void*                                   pNext;
    VkImageCompressionFlagsEXT              imageCompressionFlags;
    VkImageCompressionFixedRateFlagsEXT     imageCompressionFixedRateFlags;
} VkImageCompressionPropertiesEXT;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- imageCompressionFlags returns a value describing the compression controls that apply to the image. The value will be either VK_IMAGE_COMPRESSION_DEFAULT_EXT to indicate no fixed-rate compression, VK_IMAGE_COMPRESSION_FIXED_RATE_EXPLICIT_EXT to indicate fixed-rate compression, or VK_IMAGE_COMPRESSION_DISABLED_EXT to indicate no compression.
- imageCompressionFixedRateFlags returns a VkImageCompressionFixedRateFlagsEXT value describing the compression rates that apply to the specified aspect of the image.

### Valid Usage (Implicit)

- VUID-VkImageCompressionPropertiesEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_IMAGE_COMPRESSION_PROPERTIES_EXT

If the pNext list of VkImageCreateInfo includes a VkImageAlignmentControlCreateInfoMESA structure, then that structure describes desired alignment for this image.

The VkImageAlignmentControlCreateInfoMESA structure is defined as:

```
// Provided by VK_MESA_image_alignment_control
typedef struct VkImageAlignmentControlCreateInfoMESA {
    VkStructureType    sType;
    const void*        pNext;
    uint32_t           maximumRequestedAlignment;
} VkImageAlignmentControlCreateInfoMESA;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- maximumRequestedAlignment specifies the maximum alignment for the image.

If maximumRequestedAlignment is not 0, the implementation **should** choose an image memory layout that requires an alignment no larger than maximumRequestedAlignment as reported in VkMemoryRequirements::alignment. If such a layout does not exist for the given image creation parameters, the implementation **should** return the smallest alignment which is supported in

VkMemoryRequirements.

If an implementation needs to disable image compression for `maximumRequestedAlignment` to be honored - where a larger alignment would enable image compression - the implementation **should** not use `maximumRequestedAlignment`, and **should** return the smallest alignment which does not compromise compression. If `imageCompressionControl` is enabled, the application **can** chain a VkImageCompressionControlEXT with `VK_IMAGE_COMPRESSION_DISABLED_EXT`. In this case, image compression considerations **should** not apply when implementation decides alignment.

### Valid Usage

- VUID-VkImageAlignmentControlCreateInfoMESA-maximumRequestedAlignment-09655
  If `maximumRequestedAlignment` is not 0, `maximumRequestedAlignment` **must** be a power of two
- VUID-VkImageAlignmentControlCreateInfoMESA-maximumRequestedAlignment-09656
  If `maximumRequestedAlignment` is not 0, the bitwise-and of `maximumRequestedAlignment` and `supportedImageAlignmentMask` **must** be non-zero
- VUID-VkImageAlignmentControlCreateInfoMESA-imageAlignmentControl-09657
  `imageAlignmentControl` **must** be enabled on the device

### Valid Usage (Implicit)

- VUID-VkImageAlignmentControlCreateInfoMESA-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_IMAGE_ALIGNMENT_CONTROL_CREATE_INFO_MESA`

Bits which **can** be set in

- VkImageViewUsageCreateInfo::`usage`
- VkImageStencilUsageCreateInfo::`stencilUsage`
- VkImageCreateInfo::`usage`

specify intended usage of an image, and are:

```
// Provided by VK_VERSION_1_0
typedef enum VkImageUsageFlagBits {
    VK_IMAGE_USAGE_TRANSFER_SRC_BIT = 0x00000001,
    VK_IMAGE_USAGE_TRANSFER_DST_BIT = 0x00000002,
    VK_IMAGE_USAGE_SAMPLED_BIT = 0x00000004,
    VK_IMAGE_USAGE_STORAGE_BIT = 0x00000008,
    VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT = 0x00000010,
    VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT = 0x00000020,
    VK_IMAGE_USAGE_TRANSIENT_ATTACHMENT_BIT = 0x00000040,
    VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT = 0x00000080,
  // Provided by VK_KHR_video_decode_queue
    VK_IMAGE_USAGE_VIDEO_DECODE_DST_BIT_KHR = 0x00000400,
  // Provided by VK_KHR_video_decode_queue
    VK_IMAGE_USAGE_VIDEO_DECODE_SRC_BIT_KHR = 0x00000800,
  // Provided by VK_KHR_video_decode_queue
    VK_IMAGE_USAGE_VIDEO_DECODE_DPB_BIT_KHR = 0x00001000,
  // Provided by VK_EXT_fragment_density_map
    VK_IMAGE_USAGE_FRAGMENT_DENSITY_MAP_BIT_EXT = 0x00000200,
  // Provided by VK_KHR_fragment_shading_rate
    VK_IMAGE_USAGE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR = 0x00000100,
  // Provided by VK_EXT_host_image_copy
    VK_IMAGE_USAGE_HOST_TRANSFER_BIT_EXT = 0x00400000,
  // Provided by VK_KHR_video_encode_queue
    VK_IMAGE_USAGE_VIDEO_ENCODE_DST_BIT_KHR = 0x00002000,
  // Provided by VK_KHR_video_encode_queue
    VK_IMAGE_USAGE_VIDEO_ENCODE_SRC_BIT_KHR = 0x00004000,
  // Provided by VK_KHR_video_encode_queue
    VK_IMAGE_USAGE_VIDEO_ENCODE_DPB_BIT_KHR = 0x00008000,
  // Provided by VK_EXT_attachment_feedback_loop_layout
    VK_IMAGE_USAGE_ATTACHMENT_FEEDBACK_LOOP_BIT_EXT = 0x00080000,
  // Provided by VK_HUAWEI_invocation_mask
    VK_IMAGE_USAGE_INVOCATION_MASK_BIT_HUAWEI = 0x00040000,
  // Provided by VK_QCOM_image_processing
    VK_IMAGE_USAGE_SAMPLE_WEIGHT_BIT_QCOM = 0x00100000,
  // Provided by VK_QCOM_image_processing
    VK_IMAGE_USAGE_SAMPLE_BLOCK_MATCH_BIT_QCOM = 0x00200000,
  // Provided by VK_NV_shading_rate_image
    VK_IMAGE_USAGE_SHADING_RATE_IMAGE_BIT_NV =
VK_IMAGE_USAGE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR,
} VkImageUsageFlagBits;
```

- `VK_IMAGE_USAGE_TRANSFER_SRC_BIT` specifies that the image **can** be used as the source of a transfer command.
- `VK_IMAGE_USAGE_TRANSFER_DST_BIT` specifies that the image **can** be used as the destination of a transfer command.
- `VK_IMAGE_USAGE_SAMPLED_BIT` specifies that the image **can** be used to create a `VkImageView` suitable for occupying a `VkDescriptorSet` slot either of type `VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE` or `VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER`, and be sampled by a shader.

- VK_IMAGE_USAGE_STORAGE_BIT specifies that the image **can** be used to create a VkImageView suitable for occupying a VkDescriptorSet slot of type VK_DESCRIPTOR_TYPE_STORAGE_IMAGE.
- VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT specifies that the image **can** be used to create a VkImageView suitable for use as a color or resolve attachment in a VkFramebuffer.
- VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT specifies that the image **can** be used to create a VkImageView suitable for use as a depth/stencil or depth/stencil resolve attachment in a VkFramebuffer.
- VK_IMAGE_USAGE_TRANSIENT_ATTACHMENT_BIT specifies that implementations **may** support using memory allocations with the VK_MEMORY_PROPERTY_LAZILY_ALLOCATED_BIT to back an image with this usage. This bit **can** be set for any image that **can** be used to create a VkImageView suitable for use as a color, resolve, depth/stencil, or input attachment.
- VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT specifies that the image **can** be used to create a VkImageView suitable for occupying VkDescriptorSet slot of type VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT; be read from a shader as an input attachment; and be used as an input attachment in a framebuffer.
- VK_IMAGE_USAGE_FRAGMENT_DENSITY_MAP_BIT_EXT specifies that the image **can** be used to create a VkImageView suitable for use as a fragment density map image.
- VK_IMAGE_USAGE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR specifies that the image **can** be used to create a VkImageView suitable for use as a fragment shading rate attachment or shading rate image
- VK_IMAGE_USAGE_VIDEO_DECODE_DST_BIT_KHR specifies that the image **can** be used as a decode output picture in a video decode operation.
- VK_IMAGE_USAGE_VIDEO_DECODE_SRC_BIT_KHR is reserved for future use.
- VK_IMAGE_USAGE_VIDEO_DECODE_DPB_BIT_KHR specifies that the image **can** be used as an output reconstructed picture or an input reference picture in a video decode operation.
- VK_IMAGE_USAGE_VIDEO_ENCODE_DST_BIT_KHR is reserved for future use.
- VK_IMAGE_USAGE_VIDEO_ENCODE_SRC_BIT_KHR specifies that the image **can** be used as an encode input picture in a video encode operation.
- VK_IMAGE_USAGE_VIDEO_ENCODE_DPB_BIT_KHR specifies that the image **can** be used as an output reconstructed picture or an input reference picture in a video encode operation.
- VK_IMAGE_USAGE_ATTACHMENT_FEEDBACK_LOOP_BIT_EXT specifies that the image **can** be transitioned to the VK_IMAGE_LAYOUT_ATTACHMENT_FEEDBACK_LOOP_OPTIMAL_EXT layout to be used as a color or depth/stencil attachment in a VkFramebuffer and/or as a read-only input resource in a shader (sampled image, combined image sampler or input attachment) in the same render pass.
- VK_IMAGE_USAGE_HOST_TRANSFER_BIT_EXT specifies that the image **can** be used with host copy commands and host layout transitions.

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkImageUsageFlags;
```

VkImageUsageFlags is a bitmask type for setting a mask of zero or more VkImageUsageFlagBits.

When creating a VkImageView one of the following VkImageUsageFlagBits **must** be set:

- VK_IMAGE_USAGE_SAMPLED_BIT
- VK_IMAGE_USAGE_STORAGE_BIT
- VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT
- VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT
- VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT
- VK_IMAGE_USAGE_TRANSIENT_ATTACHMENT_BIT
- VK_IMAGE_USAGE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR
- VK_IMAGE_USAGE_FRAGMENT_DENSITY_MAP_BIT_EXT
- VK_IMAGE_USAGE_VIDEO_DECODE_DST_BIT_KHR
- VK_IMAGE_USAGE_VIDEO_DECODE_DPB_BIT_KHR
- VK_IMAGE_USAGE_VIDEO_ENCODE_SRC_BIT_KHR
- VK_IMAGE_USAGE_VIDEO_ENCODE_DPB_BIT_KHR
- VK_IMAGE_USAGE_SAMPLE_WEIGHT_BIT_QCOM
- VK_IMAGE_USAGE_SAMPLE_BLOCK_MATCH_BIT_QCOM

Bits which **can** be set in VkImageCreateInfo::flags, specifying additional parameters of an image, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkImageCreateFlagBits {
    VK_IMAGE_CREATE_SPARSE_BINDING_BIT = 0x00000001,
    VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT = 0x00000002,
    VK_IMAGE_CREATE_SPARSE_ALIASED_BIT = 0x00000004,
    VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT = 0x00000008,
    VK_IMAGE_CREATE_CUBE_COMPATIBLE_BIT = 0x00000010,
  // Provided by VK_VERSION_1_1
    VK_IMAGE_CREATE_ALIAS_BIT = 0x00000400,
  // Provided by VK_VERSION_1_1
    VK_IMAGE_CREATE_SPLIT_INSTANCE_BIND_REGIONS_BIT = 0x00000040,
  // Provided by VK_VERSION_1_1
    VK_IMAGE_CREATE_2D_ARRAY_COMPATIBLE_BIT = 0x00000020,
  // Provided by VK_VERSION_1_1
    VK_IMAGE_CREATE_BLOCK_TEXEL_VIEW_COMPATIBLE_BIT = 0x00000080,
  // Provided by VK_VERSION_1_1
    VK_IMAGE_CREATE_EXTENDED_USAGE_BIT = 0x00000100,
  // Provided by VK_VERSION_1_1
    VK_IMAGE_CREATE_PROTECTED_BIT = 0x00000800,
  // Provided by VK_VERSION_1_1
    VK_IMAGE_CREATE_DISJOINT_BIT = 0x00000200,
  // Provided by VK_NV_corner_sampled_image
    VK_IMAGE_CREATE_CORNER_SAMPLED_BIT_NV = 0x00002000,
  // Provided by VK_EXT_sample_locations
```

```
    VK_IMAGE_CREATE_SAMPLE_LOCATIONS_COMPATIBLE_DEPTH_BIT_EXT = 0x00001000,
  // Provided by VK_EXT_fragment_density_map
    VK_IMAGE_CREATE_SUBSAMPLED_BIT_EXT = 0x00004000,
  // Provided by VK_EXT_descriptor_buffer
    VK_IMAGE_CREATE_DESCRIPTOR_BUFFER_CAPTURE_REPLAY_BIT_EXT = 0x00010000,
  // Provided by VK_EXT_multisampled_render_to_single_sampled
    VK_IMAGE_CREATE_MULTISAMPLED_RENDER_TO_SINGLE_SAMPLED_BIT_EXT = 0x00040000,
  // Provided by VK_EXT_image_2d_view_of_3d
    VK_IMAGE_CREATE_2D_VIEW_COMPATIBLE_BIT_EXT = 0x00020000,
  // Provided by VK_QCOM_fragment_density_map_offset
    VK_IMAGE_CREATE_FRAGMENT_DENSITY_MAP_OFFSET_BIT_QCOM = 0x00008000,
  // Provided by VK_KHR_video_maintenance1
    VK_IMAGE_CREATE_VIDEO_PROFILE_INDEPENDENT_BIT_KHR = 0x00100000,
  // Provided by VK_KHR_bind_memory2 with VK_KHR_device_group
    VK_IMAGE_CREATE_SPLIT_INSTANCE_BIND_REGIONS_BIT_KHR =
VK_IMAGE_CREATE_SPLIT_INSTANCE_BIND_REGIONS_BIT,
  // Provided by VK_KHR_maintenance1
    VK_IMAGE_CREATE_2D_ARRAY_COMPATIBLE_BIT_KHR =
VK_IMAGE_CREATE_2D_ARRAY_COMPATIBLE_BIT,
  // Provided by VK_KHR_maintenance2
    VK_IMAGE_CREATE_BLOCK_TEXEL_VIEW_COMPATIBLE_BIT_KHR =
VK_IMAGE_CREATE_BLOCK_TEXEL_VIEW_COMPATIBLE_BIT,
  // Provided by VK_KHR_maintenance2
    VK_IMAGE_CREATE_EXTENDED_USAGE_BIT_KHR = VK_IMAGE_CREATE_EXTENDED_USAGE_BIT,
  // Provided by VK_KHR_sampler_ycbcr_conversion
    VK_IMAGE_CREATE_DISJOINT_BIT_KHR = VK_IMAGE_CREATE_DISJOINT_BIT,
  // Provided by VK_KHR_bind_memory2
    VK_IMAGE_CREATE_ALIAS_BIT_KHR = VK_IMAGE_CREATE_ALIAS_BIT,
} VkImageCreateFlagBits;
```

- `VK_IMAGE_CREATE_SPARSE_BINDING_BIT` specifies that the image will be backed using sparse memory binding.
- `VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT` specifies that the image **can** be partially backed using sparse memory binding. Images created with this flag **must** also be created with the `VK_IMAGE_CREATE_SPARSE_BINDING_BIT` flag.
- `VK_IMAGE_CREATE_SPARSE_ALIASED_BIT` specifies that the image will be backed using sparse memory binding with memory ranges that might also simultaneously be backing another image (or another portion of the same image). Images created with this flag **must** also be created with the `VK_IMAGE_CREATE_SPARSE_BINDING_BIT` flag.
- `VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT` specifies that the image **can** be used to create a `VkImageView` with a different format from the image. For multi-planar formats, `VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT` specifies that a `VkImageView` can be created of a *plane* of the image.
- `VK_IMAGE_CREATE_CUBE_COMPATIBLE_BIT` specifies that the image **can** be used to create a `VkImageView` of type `VK_IMAGE_VIEW_TYPE_CUBE` or `VK_IMAGE_VIEW_TYPE_CUBE_ARRAY`.
- `VK_IMAGE_CREATE_2D_ARRAY_COMPATIBLE_BIT` specifies that the image **can** be used to create a `VkImageView` of type `VK_IMAGE_VIEW_TYPE_2D` or `VK_IMAGE_VIEW_TYPE_2D_ARRAY`.

- VK_IMAGE_CREATE_2D_VIEW_COMPATIBLE_BIT_EXT specifies that the image **can** be used to create a VkImageView of type VK_IMAGE_VIEW_TYPE_2D.
- VK_IMAGE_CREATE_PROTECTED_BIT specifies that the image is a protected image.
- VK_IMAGE_CREATE_SPLIT_INSTANCE_BIND_REGIONS_BIT specifies that the image **can** be used with a non-zero value of the splitInstanceBindRegionCount member of a VkBindImageMemoryDeviceGroupInfo structure passed into vkBindImageMemory2. This flag also has the effect of making the image use the standard sparse image block dimensions.
- VK_IMAGE_CREATE_BLOCK_TEXEL_VIEW_COMPATIBLE_BIT specifies that the image having a compressed format **can** be used to create a VkImageView with an uncompressed format where each texel in the image view corresponds to a compressed texel block of the image.
- VK_IMAGE_CREATE_EXTENDED_USAGE_BIT specifies that the image **can** be created with usage flags that are not supported for the format the image is created with but are supported for at least one format a VkImageView created from the image **can** have.
- VK_IMAGE_CREATE_DISJOINT_BIT specifies that an image with a multi-planar format **must** have each plane separately bound to memory, rather than having a single memory binding for the whole image; the presence of this bit distinguishes a *disjoint image* from an image without this bit set.
- VK_IMAGE_CREATE_ALIAS_BIT specifies that two images created with the same creation parameters and aliased to the same memory **can** interpret the contents of the memory consistently with each other, subject to the rules described in the Memory Aliasing section. This flag further specifies that each plane of a *disjoint* image **can** share an in-memory non-linear representation with single-plane images, and that a single-plane image **can** share an in-memory non-linear representation with a plane of a multi-planar disjoint image, according to the rules in Compatible Formats of Planes of Multi-Planar Formats. If the pNext chain includes a VkExternalMemoryImageCreateInfo or VkExternalMemoryImageCreateInfoNV structure whose handleTypes member is not 0, it is as if VK_IMAGE_CREATE_ALIAS_BIT is set.
- VK_IMAGE_CREATE_SAMPLE_LOCATIONS_COMPATIBLE_DEPTH_BIT_EXT specifies that an image with a depth or depth/stencil format **can** be used with custom sample locations when used as a depth/stencil attachment.
- VK_IMAGE_CREATE_CORNER_SAMPLED_BIT_NV specifies that the image is a corner-sampled image.
- VK_IMAGE_CREATE_SUBSAMPLED_BIT_EXT specifies that an image **can** be in a subsampled format which **may** be more optimal when written as an attachment by a render pass that has a fragment density map attachment. Accessing a subsampled image has additional considerations:
  - Image data read as an image sampler will have undefined values if the sampler was not created with flags containing VK_SAMPLER_CREATE_SUBSAMPLED_BIT_EXT or was not sampled through the use of a combined image sampler with an immutable sampler in VkDescriptorSetLayoutBinding.
  - Image data read with an input attachment will have undefined values if the contents were not written as an attachment in an earlier subpass of the same render pass.
  - Image data read as an image sampler in the fragment shader will be additionally be read by the device during VK_PIPELINE_STAGE_VERTEX_SHADER_BIT if VkPhysicalDeviceFragmentDensityMap2PropertiesEXT::subsampledCoarseReconstructionEarlyAcce

ss is VK_TRUE and the sampler was created with flags containing VK_SAMPLER_CREATE_SUBSAMPLED_COARSE_RECONSTRUCTION_BIT_EXT.

  - Image data read with load operations are resampled to the fragment density of the render pass if VkPhysicalDeviceFragmentDensityMap2PropertiesEXT::subsampledLoads is VK_TRUE. Otherwise, values of image data are undefined.
  - Image contents outside of the render area take on undefined values if the image is stored as a render pass attachment.

- VK_IMAGE_CREATE_FRAGMENT_DENSITY_MAP_OFFSET_BIT_QCOM specifies that an image **can** be used in a render pass with non-zero fragment density map offsets. In a render pass with non-zero offsets, fragment density map attachments, input attachments, color attachments, depth/stencil attachment, resolve attachments, and preserve attachments **must** be created with VK_IMAGE_CREATE_FRAGMENT_DENSITY_MAP_OFFSET_BIT_QCOM.
- VK_IMAGE_CREATE_DESCRIPTOR_BUFFER_CAPTURE_REPLAY_BIT_EXT specifies that the image **can** be used with descriptor buffers when capturing and replaying (e.g. for trace capture and replay), see VkOpaqueCaptureDescriptorDataCreateInfoEXT for more detail.
- VK_IMAGE_CREATE_MULTISAMPLED_RENDER_TO_SINGLE_SAMPLED_BIT_EXT specifies that an image **can** be used with multisampled rendering as a single-sampled framebuffer attachment
- VK_IMAGE_CREATE_VIDEO_PROFILE_INDEPENDENT_BIT_KHR specifies that the image **can** be used in video coding operations without having to specify at image creation time the set of video profiles the image will be used with, except for images used only as DPB pictures, as long as the image is otherwise compatible with the video profile in question.

**NOTE**

This enables exchanging video picture data without additional copies or conversions when used as:

  - Decode output pictures, indifferent of the video profile used to produce them.
  - Encode input pictures, indifferent of the video profile used to consume them.

This includes images created with both VK_IMAGE_USAGE_VIDEO_DECODE_DST_BIT_KHR and VK_IMAGE_USAGE_VIDEO_DECODE_DPB_BIT_KHR, which is necessary to use the same video picture as the reconstructed picture and decode output picture in a video decode operation on implementations supporting VK_VIDEO_DECODE_CAPABILITY_DPB_AND_OUTPUT_COINCIDE_BIT_KHR.

However, images with only DPB usage remain tied to the video profiles the image was created with, as the data layout of such DPB-only images **may** be implementation- and codec-dependent.

If an application would like to share or reuse the device memory backing such images (e.g. for the purposes of temporal aliasing), then it **should** create separate image objects for each video profile and bind them to the same underlying device memory range, similar to how memory resources **can** be shared across separate video sessions or any other memory-backed resource.

See Sparse Resource Features and Sparse Physical Device Features for more details.

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkImageCreateFlags;
```

VkImageCreateFlags is a bitmask type for setting a mask of zero or more VkImageCreateFlagBits.

Possible values of VkImageCreateInfo::imageType, specifying the basic dimensionality of an image, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkImageType {
    VK_IMAGE_TYPE_1D = 0,
    VK_IMAGE_TYPE_2D = 1,
    VK_IMAGE_TYPE_3D = 2,
} VkImageType;
```

- VK_IMAGE_TYPE_1D specifies a one-dimensional image.
- VK_IMAGE_TYPE_2D specifies a two-dimensional image.
- VK_IMAGE_TYPE_3D specifies a three-dimensional image.

Possible values of VkImageCreateInfo::tiling, specifying the tiling arrangement of texel blocks in an image, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkImageTiling {
    VK_IMAGE_TILING_OPTIMAL = 0,
    VK_IMAGE_TILING_LINEAR = 1,
  // Provided by VK_EXT_image_drm_format_modifier
    VK_IMAGE_TILING_DRM_FORMAT_MODIFIER_EXT = 1000158000,
} VkImageTiling;
```

- VK_IMAGE_TILING_OPTIMAL specifies optimal tiling (texels are laid out in an implementation-dependent arrangement, for more efficient memory access).
- VK_IMAGE_TILING_LINEAR specifies linear tiling (texels are laid out in memory in row-major order, possibly with some padding on each row).
- VK_IMAGE_TILING_DRM_FORMAT_MODIFIER_EXT indicates that the image's tiling is defined by a Linux DRM format modifier. The modifier is specified at image creation with VkImageDrmFormatModifierListCreateInfoEXT or VkImageDrmFormatModifierExplicitCreateInfoEXT, and **can** be queried with vkGetImageDrmFormatModifierPropertiesEXT.

To query the memory layout of an image subresource, call:

```
// Provided by VK_VERSION_1_0
void vkGetImageSubresourceLayout(
    VkDevice                                    device,
    VkImage                                     image,
    const VkImageSubresource*                   pSubresource,
    VkSubresourceLayout*                        pLayout);
```

- `device` is the logical device that owns the image.
- `image` is the image whose layout is being queried.
- `pSubresource` is a pointer to a VkImageSubresource structure selecting a specific image subresource from the image.
- `pLayout` is a pointer to a VkSubresourceLayout structure in which the layout is returned.

If the image is linear, then the returned layout is valid for host access.

If the image's tiling is `VK_IMAGE_TILING_LINEAR` and its format is a multi-planar format, then `vkGetImageSubresourceLayout` describes one *format plane* of the image. If the image's tiling is `VK_IMAGE_TILING_DRM_FORMAT_MODIFIER_EXT`, then `vkGetImageSubresourceLayout` describes one *memory plane* of the image. If the image's tiling is `VK_IMAGE_TILING_DRM_FORMAT_MODIFIER_EXT` and the image is non-linear, then the returned layout has an implementation-dependent meaning; the vendor of the image's DRM format modifier **may** provide documentation that explains how to interpret the returned layout.

`vkGetImageSubresourceLayout` is invariant for the lifetime of a single image. However, the subresource layout of images in Android hardware buffer or QNX Screen buffer external memory is not known until the image has been bound to memory, so applications **must** not call vkGetImageSubresourceLayout for such an image before it has been bound.

### Valid Usage

- VUID-vkGetImageSubresourceLayout-image-07790
  `image` **must** have been created with `tiling` equal to `VK_IMAGE_TILING_LINEAR` or `VK_IMAGE_TILING_DRM_FORMAT_MODIFIER_EXT`
- VUID-vkGetImageSubresourceLayout-aspectMask-00997
  The `aspectMask` member of `pSubresource` **must** only have a single bit set
- VUID-vkGetImageSubresourceLayout-mipLevel-01716
  The `mipLevel` member of `pSubresource` **must** be less than the `mipLevels` specified in `image`
- VUID-vkGetImageSubresourceLayout-arrayLayer-01717
  The `arrayLayer` member of `pSubresource` **must** be less than the `arrayLayers` specified in `image`
- VUID-vkGetImageSubresourceLayout-format-08886
  If `format` of the `image` is a color format that is not a multi-planar image format, and `tiling` of the `image` is `VK_IMAGE_TILING_LINEAR` or `VK_IMAGE_TILING_OPTIMAL`, the `aspectMask` member of `pSubresource` **must** be `VK_IMAGE_ASPECT_COLOR_BIT`

- VUID-vkGetImageSubresourceLayout-format-04462
  If `format` of the `image` has a depth component, the `aspectMask` member of `pSubresource` **must** contain `VK_IMAGE_ASPECT_DEPTH_BIT`
- VUID-vkGetImageSubresourceLayout-format-04463
  If `format` of the `image` has a stencil component, the `aspectMask` member of `pSubresource` **must** contain `VK_IMAGE_ASPECT_STENCIL_BIT`
- VUID-vkGetImageSubresourceLayout-format-04464
  If `format` of the `image` does not contain a stencil or depth component, the `aspectMask` member of `pSubresource` **must** not contain `VK_IMAGE_ASPECT_DEPTH_BIT` or `VK_IMAGE_ASPECT_STENCIL_BIT`
- VUID-vkGetImageSubresourceLayout-tiling-08717
  If the `tiling` of the `image` is `VK_IMAGE_TILING_LINEAR` and has a multi-planar image format, then the `aspectMask` member of `pSubresource` **must** be a single valid multi-planar aspect mask bit
- VUID-vkGetImageSubresourceLayout-image-09432
  If `image` was created with the `VK_EXTERNAL_MEMORY_HANDLE_TYPE_ANDROID_HARDWARE_BUFFER_BIT_ANDROID` external memory handle type, then `image` **must** be bound to memory
- VUID-vkGetImageSubresourceLayout-tiling-09433
  If the `tiling` of the `image` is `VK_IMAGE_TILING_DRM_FORMAT_MODIFIER_EXT`, then the `aspectMask` member of `pSubresource` **must** be `VK_IMAGE_ASPECT_MEMORY_PLANE_i_BIT_EXT` and the index *i* **must** be less than the VkDrmFormatModifierPropertiesEXT::`drmFormatModifierPlaneCount` associated with the image's `format` and VkImageDrmFormatModifierPropertiesEXT::`drmFormatModifier`

## Valid Usage (Implicit)

- VUID-vkGetImageSubresourceLayout-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkGetImageSubresourceLayout-image-parameter
  `image` **must** be a valid VkImage handle
- VUID-vkGetImageSubresourceLayout-pSubresource-parameter
  `pSubresource` **must** be a valid pointer to a valid VkImageSubresource structure
- VUID-vkGetImageSubresourceLayout-pLayout-parameter
  `pLayout` **must** be a valid pointer to a VkSubresourceLayout structure
- VUID-vkGetImageSubresourceLayout-image-parent
  `image` **must** have been created, allocated, or retrieved from `device`

The `VkImageSubresource` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkImageSubresource {
    VkImageAspectFlags    aspectMask;
    uint32_t              mipLevel;
    uint32_t              arrayLayer;
} VkImageSubresource;
```

- `aspectMask` is a VkImageAspectFlags value selecting the image *aspect*.
- `mipLevel` selects the mipmap level.
- `arrayLayer` selects the array layer.

### Valid Usage (Implicit)

- VUID-VkImageSubresource-aspectMask-parameter
  `aspectMask` **must** be a valid combination of VkImageAspectFlagBits values
- VUID-VkImageSubresource-aspectMask-requiredbitmask
  `aspectMask` **must** not be `0`

Information about the layout of the image subresource is returned in a `VkSubresourceLayout` structure:

```
// Provided by VK_VERSION_1_0
typedef struct VkSubresourceLayout {
    VkDeviceSize    offset;
    VkDeviceSize    size;
    VkDeviceSize    rowPitch;
    VkDeviceSize    arrayPitch;
    VkDeviceSize    depthPitch;
} VkSubresourceLayout;
```

- `offset` is the byte offset from the start of the image or the plane where the image subresource begins.
- `size` is the size in bytes of the image subresource. `size` includes any extra memory that is required based on `rowPitch`.
- `rowPitch` describes the number of bytes between each row of texels in an image.
- `arrayPitch` describes the number of bytes between each array layer of an image.
- `depthPitch` describes the number of bytes between each slice of 3D image.

If the image is linear, then `rowPitch`, `arrayPitch` and `depthPitch` describe the layout of the image subresource in linear memory. For uncompressed formats, `rowPitch` is the number of bytes between texels with the same x coordinate in adjacent rows (y coordinates differ by one). `arrayPitch` is the number of bytes between texels with the same x and y coordinate in adjacent array layers of the image (array layer values differ by one). `depthPitch` is the number of bytes between texels with the

same x and y coordinate in adjacent slices of a 3D image (z coordinates differ by one). Expressed as an addressing formula, the starting byte of a texel in the image subresource has address:

```
// (x,y,z,layer) are in texel coordinates
address(x,y,z,layer) = layer*arrayPitch + z*depthPitch + y*rowPitch + x*elementSize +
offset
```

For compressed formats, the `rowPitch` is the number of bytes between compressed texel blocks in adjacent rows. `arrayPitch` is the number of bytes between compressed texel blocks in adjacent array layers. `depthPitch` is the number of bytes between compressed texel blocks in adjacent slices of a 3D image.

```
// (x,y,z,layer) are in compressed texel block coordinates
address(x,y,z,layer) = layer*arrayPitch + z*depthPitch + y*rowPitch + x
*compressedTexelBlockByteSize + offset;
```

The value of `arrayPitch` is undefined for images that were not created as arrays. `depthPitch` is defined only for 3D images.

If the image has a *single-plane* color format and its tiling is `VK_IMAGE_TILING_LINEAR` , then the `aspectMask` member of `VkImageSubresource` **must** be `VK_IMAGE_ASPECT_COLOR_BIT`.

If the image has a depth/stencil format and its tiling is `VK_IMAGE_TILING_LINEAR` , then `aspectMask` **must** be either `VK_IMAGE_ASPECT_DEPTH_BIT` or `VK_IMAGE_ASPECT_STENCIL_BIT`. On implementations that store depth and stencil aspects separately, querying each of these image subresource layouts will return a different `offset` and `size` representing the region of memory used for that aspect. On implementations that store depth and stencil aspects interleaved, the same `offset` and `size` are returned and represent the interleaved memory allocation.

If the image has a multi-planar format and its tiling is `VK_IMAGE_TILING_LINEAR` , then the `aspectMask` member of `VkImageSubresource` **must** be `VK_IMAGE_ASPECT_PLANE_0_BIT`, `VK_IMAGE_ASPECT_PLANE_1_BIT`, or (for 3-plane formats only) `VK_IMAGE_ASPECT_PLANE_2_BIT`. Querying each of these image subresource layouts will return a different `offset` and `size` representing the region of memory used for that plane. If the image is *disjoint*, then the `offset` is relative to the base address of the plane. If the image is *non-disjoint*, then the `offset` is relative to the base address of the image.

If the image's tiling is `VK_IMAGE_TILING_DRM_FORMAT_MODIFIER_EXT`, then the `aspectMask` member of `VkImageSubresource` **must** be one of `VK_IMAGE_ASPECT_MEMORY_PLANE_i_BIT_EXT`, where the maximum allowed plane index *i* is defined by the VkDrmFormatModifierPropertiesEXT ::`drmFormatModifierPlaneCount` associated with the image's VkImageCreateInfo::`format` and modifier. The memory range used by the subresource is described by `offset` and `size`. If the image is *disjoint*, then the `offset` is relative to the base address of the *memory plane*. If the image is *non-disjoint*, then the `offset` is relative to the base address of the image. If the image is non-linear, then `rowPitch`, `arrayPitch`, and `depthPitch` have an implementation-dependent meaning.

To query the memory layout of an image subresource, call:

```
// Provided by VK_KHR_maintenance5
void vkGetImageSubresourceLayout2KHR(
    VkDevice                                    device,
    VkImage                                     image,
    const VkImageSubresource2KHR*               pSubresource,
    VkSubresourceLayout2KHR*                    pLayout);
```

or the equivalent command

```
// Provided by VK_EXT_host_image_copy, VK_EXT_image_compression_control
void vkGetImageSubresourceLayout2EXT(
    VkDevice                                    device,
    VkImage                                     image,
    const VkImageSubresource2KHR*               pSubresource,
    VkSubresourceLayout2KHR*                    pLayout);
```

- `device` is the logical device that owns the image.
- `image` is the image whose layout is being queried.
- `pSubresource` is a pointer to a VkImageSubresource2KHR structure selecting a specific image for the image subresource.
- `pLayout` is a pointer to a VkSubresourceLayout2KHR structure in which the layout is returned.

`vkGetImageSubresourceLayout2KHR` behaves similarly to vkGetImageSubresourceLayout, with the ability to specify extended inputs via chained input structures, and to return extended information via chained output structures.

It is legal to call `vkGetImageSubresourceLayout2KHR` with an `image` created with `tiling` equal to `VK_IMAGE_TILING_OPTIMAL`, but the members of VkSubresourceLayout2KHR::`subresourceLayout` will have undefined values in this case.

| | |
|---|---|
| **NOTE** | Structures chained from VkImageSubresource2KHR::`pNext` will also be updated when `tiling` is equal to `VK_IMAGE_TILING_OPTIMAL`. |

### Valid Usage

- VUID-vkGetImageSubresourceLayout2KHR-aspectMask-00997
  The `aspectMask` member of `pSubresource` **must** only have a single bit set
- VUID-vkGetImageSubresourceLayout2KHR-mipLevel-01716
  The `mipLevel` member of `pSubresource` **must** be less than the `mipLevels` specified in `image`
- VUID-vkGetImageSubresourceLayout2KHR-arrayLayer-01717
  The `arrayLayer` member of `pSubresource` **must** be less than the `arrayLayers` specified in `image`
- VUID-vkGetImageSubresourceLayout2KHR-format-08886
  If `format` of the `image` is a color format that is not a multi-planar image format, and `tiling`

of the image is VK_IMAGE_TILING_LINEAR or VK_IMAGE_TILING_OPTIMAL, the aspectMask member of pSubresource **must** be VK_IMAGE_ASPECT_COLOR_BIT

- VUID-vkGetImageSubresourceLayout2KHR-format-04462
  If format of the image has a depth component, the aspectMask member of pSubresource **must** contain VK_IMAGE_ASPECT_DEPTH_BIT
- VUID-vkGetImageSubresourceLayout2KHR-format-04463
  If format of the image has a stencil component, the aspectMask member of pSubresource **must** contain VK_IMAGE_ASPECT_STENCIL_BIT
- VUID-vkGetImageSubresourceLayout2KHR-format-04464
  If format of the image does not contain a stencil or depth component, the aspectMask member of pSubresource **must** not contain VK_IMAGE_ASPECT_DEPTH_BIT or VK_IMAGE_ASPECT_STENCIL_BIT
- VUID-vkGetImageSubresourceLayout2KHR-tiling-08717
  If the tiling of the image is VK_IMAGE_TILING_LINEAR and has a multi-planar image format, then the aspectMask member of pSubresource **must** be a single valid multi-planar aspect mask bit
- VUID-vkGetImageSubresourceLayout2KHR-image-09434
  If image was created with the VK_EXTERNAL_MEMORY_HANDLE_TYPE_ANDROID_HARDWARE_BUFFER_BIT_ANDROID external memory handle type, then image **must** be bound to memory
- VUID-vkGetImageSubresourceLayout2KHR-tiling-09435
  If the tiling of the image is VK_IMAGE_TILING_DRM_FORMAT_MODIFIER_EXT, then the aspectMask member of pSubresource **must** be VK_IMAGE_ASPECT_MEMORY_PLANE_*i*_BIT_EXT and the index *i* **must** be less than the VkDrmFormatModifierPropertiesEXT::drmFormatModifierPlaneCount associated with the image's format and VkImageDrmFormatModifierPropertiesEXT::drmFormatModifier

## Valid Usage (Implicit)

- VUID-vkGetImageSubresourceLayout2KHR-device-parameter
  device **must** be a valid VkDevice handle
- VUID-vkGetImageSubresourceLayout2KHR-image-parameter
  image **must** be a valid VkImage handle
- VUID-vkGetImageSubresourceLayout2KHR-pSubresource-parameter
  pSubresource **must** be a valid pointer to a valid VkImageSubresource2KHR structure
- VUID-vkGetImageSubresourceLayout2KHR-pLayout-parameter
  pLayout **must** be a valid pointer to a VkSubresourceLayout2KHR structure
- VUID-vkGetImageSubresourceLayout2KHR-image-parent
  image **must** have been created, allocated, or retrieved from device

The VkImageSubresource2KHR structure is defined as:

```
// Provided by VK_KHR_maintenance5
typedef struct VkImageSubresource2KHR {
    VkStructureType       sType;
    void*                 pNext;
    VkImageSubresource    imageSubresource;
} VkImageSubresource2KHR;
```

or the equivalent

```
// Provided by VK_EXT_host_image_copy, VK_EXT_image_compression_control
typedef VkImageSubresource2KHR VkImageSubresource2EXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `imageSubresource` is a VkImageSubresource structure.

### Valid Usage (Implicit)

- VUID-VkImageSubresource2KHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_IMAGE_SUBRESOURCE_2_KHR`
- VUID-VkImageSubresource2KHR-pNext-pNext
  `pNext` **must** be `NULL`
- VUID-VkImageSubresource2KHR-imageSubresource-parameter
  `imageSubresource` **must** be a valid VkImageSubresource structure

Information about the layout of the image subresource is returned in a `VkSubresourceLayout2KHR` structure:

```
// Provided by VK_KHR_maintenance5
typedef struct VkSubresourceLayout2KHR {
    VkStructureType        sType;
    void*                  pNext;
    VkSubresourceLayout    subresourceLayout;
} VkSubresourceLayout2KHR;
```

or the equivalent

```
// Provided by VK_EXT_host_image_copy, VK_EXT_image_compression_control
typedef VkSubresourceLayout2KHR VkSubresourceLayout2EXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.

- subresourceLayout is a VkSubresourceLayout structure.

### Valid Usage (Implicit)

- VUID-VkSubresourceLayout2KHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_SUBRESOURCE_LAYOUT_2_KHR
- VUID-VkSubresourceLayout2KHR-pNext-pNext
  Each pNext member of any structure (including this one) in the pNext chain **must** be either NULL or a pointer to a valid instance of VkImageCompressionPropertiesEXT or VkSubresourceHostMemcpySizeEXT
- VUID-VkSubresourceLayout2KHR-sType-unique
  The sType value of each struct in the pNext chain **must** be unique

To query the memory size needed to copy to or from an image using vkCopyMemoryToImageEXT or vkCopyImageToMemoryEXT when the VK_HOST_IMAGE_COPY_MEMCPY_EXT flag is specified, add a VkSubresourceHostMemcpySizeEXT structure to the pNext chain of the VkSubresourceLayout2EXT structure in a call to vkGetImageSubresourceLayout2EXT.

The VkSubresourceHostMemcpySizeEXT structure is defined as:

```
// Provided by VK_EXT_host_image_copy
typedef struct VkSubresourceHostMemcpySizeEXT {
    VkStructureType    sType;
    void*              pNext;
    VkDeviceSize       size;
} VkSubresourceHostMemcpySizeEXT;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- size is the size in bytes of the image subresource.

### Valid Usage (Implicit)

- VUID-VkSubresourceHostMemcpySizeEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_SUBRESOURCE_HOST_MEMCPY_SIZE_EXT

To query the memory layout of an image subresource, without an image object, call:

```
// Provided by VK_KHR_maintenance5
void vkGetDeviceImageSubresourceLayoutKHR(
    VkDevice                                    device,
    const VkDeviceImageSubresourceInfoKHR*      pInfo,
    VkSubresourceLayout2KHR*                    pLayout);
```

- device is the logical device that owns the image.
- pInfo is a pointer to a VkDeviceImageSubresourceInfoKHR structure containing parameters required for the subresource layout query.
- pLayout is a pointer to a VkSubresourceLayout2KHR structure in which the layout is returned.

vkGetDeviceImageSubresourceLayoutKHR behaves similarly to vkGetImageSubresourceLayout2KHR, but uses a VkImageCreateInfo structure to specify the image rather than a VkImage object.

### Valid Usage (Implicit)

- VUID-vkGetDeviceImageSubresourceLayoutKHR-device-parameter
  device **must** be a valid VkDevice handle
- VUID-vkGetDeviceImageSubresourceLayoutKHR-pInfo-parameter
  pInfo **must** be a valid pointer to a valid VkDeviceImageSubresourceInfoKHR structure
- VUID-vkGetDeviceImageSubresourceLayoutKHR-pLayout-parameter
  pLayout **must** be a valid pointer to a VkSubresourceLayout2KHR structure

The VkDeviceImageSubresourceInfoKHR structure is defined as:

```
// Provided by VK_KHR_maintenance5
typedef struct VkDeviceImageSubresourceInfoKHR {
    VkStructureType                 sType;
    const void*                     pNext;
    const VkImageCreateInfo*        pCreateInfo;
    const VkImageSubresource2KHR*   pSubresource;
} VkDeviceImageSubresourceInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- pCreateInfo is a pointer to a VkImageCreateInfo structure containing parameters affecting creation of the image to query.
- pSubresource pSubresource is a pointer to a VkImageSubresource2KHR structure selecting a specific image subresource for the query.

### Valid Usage

- VUID-VkDeviceImageSubresourceInfoKHR-aspectMask-00997
  The aspectMask member of pSubresource **must** only have a single bit set
- VUID-VkDeviceImageSubresourceInfoKHR-mipLevel-01716
  The mipLevel member of pSubresource **must** be less than the mipLevels specified in pCreateInfo
- VUID-VkDeviceImageSubresourceInfoKHR-arrayLayer-01717
  The arrayLayer member of pSubresource **must** be less than the arrayLayers specified in

pCreateInfo

- VUID-VkDeviceImageSubresourceInfoKHR-format-08886
  If format of the image is a color format that is not a multi-planar image format, and tiling of the pCreateInfo is VK_IMAGE_TILING_LINEAR or VK_IMAGE_TILING_OPTIMAL, the aspectMask member of pSubresource **must** be VK_IMAGE_ASPECT_COLOR_BIT
- VUID-VkDeviceImageSubresourceInfoKHR-format-04462
  If format of the pCreateInfo has a depth component, the aspectMask member of pSubresource **must** contain VK_IMAGE_ASPECT_DEPTH_BIT
- VUID-VkDeviceImageSubresourceInfoKHR-format-04463
  If format of the pCreateInfo has a stencil component, the aspectMask member of pSubresource **must** contain VK_IMAGE_ASPECT_STENCIL_BIT
- VUID-VkDeviceImageSubresourceInfoKHR-format-04464
  If format of the pCreateInfo does not contain a stencil or depth component, the aspectMask member of pSubresource **must** not contain VK_IMAGE_ASPECT_DEPTH_BIT or VK_IMAGE_ASPECT_STENCIL_BIT
- VUID-VkDeviceImageSubresourceInfoKHR-tiling-08717
  If the tiling of the pCreateInfo is VK_IMAGE_TILING_LINEAR and has a multi-planar image format, then the aspectMask member of pSubresource **must** be a single valid multi-planar aspect mask bit

### Valid Usage (Implicit)

- VUID-VkDeviceImageSubresourceInfoKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_DEVICE_IMAGE_SUBRESOURCE_INFO_KHR
- VUID-VkDeviceImageSubresourceInfoKHR-pNext-pNext
  pNext **must** be NULL
- VUID-VkDeviceImageSubresourceInfoKHR-pCreateInfo-parameter
  pCreateInfo **must** be a valid pointer to a valid VkImageCreateInfo structure
- VUID-VkDeviceImageSubresourceInfoKHR-pSubresource-parameter
  pSubresource **must** be a valid pointer to a valid VkImageSubresource2KHR structure

If an image was created with VK_IMAGE_TILING_DRM_FORMAT_MODIFIER_EXT, then the image has a Linux DRM format modifier. To query the *modifier*, call:

```
// Provided by VK_EXT_image_drm_format_modifier
VkResult vkGetImageDrmFormatModifierPropertiesEXT(
    VkDevice                                    device,
    VkImage                                     image,
    VkImageDrmFormatModifierPropertiesEXT*      pProperties);
```

- device is the logical device that owns the image.
- image is the queried image.

- `pProperties` is a pointer to a VkImageDrmFormatModifierPropertiesEXT structure in which properties of the image's *DRM format modifier* are returned.

### Valid Usage

- VUID-vkGetImageDrmFormatModifierPropertiesEXT-image-02272
  `image` **must** have been created with `tiling` equal to `VK_IMAGE_TILING_DRM_FORMAT_MODIFIER_EXT`

### Valid Usage (Implicit)

- VUID-vkGetImageDrmFormatModifierPropertiesEXT-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkGetImageDrmFormatModifierPropertiesEXT-image-parameter
  `image` **must** be a valid VkImage handle
- VUID-vkGetImageDrmFormatModifierPropertiesEXT-pProperties-parameter
  `pProperties` **must** be a valid pointer to a VkImageDrmFormatModifierPropertiesEXT structure
- VUID-vkGetImageDrmFormatModifierPropertiesEXT-image-parent
  `image` **must** have been created, allocated, or retrieved from `device`

### Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- `VK_ERROR_OUT_OF_HOST_MEMORY`

The VkImageDrmFormatModifierPropertiesEXT structure is defined as:

```
// Provided by VK_EXT_image_drm_format_modifier
typedef struct VkImageDrmFormatModifierPropertiesEXT {
    VkStructureType    sType;
    void*              pNext;
    uint64_t           drmFormatModifier;
} VkImageDrmFormatModifierPropertiesEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `drmFormatModifier` returns the image's Linux DRM format modifier.

If the `image` was created with VkImageDrmFormatModifierListCreateInfoEXT, then the returned

drmFormatModifier **must** belong to the list of modifiers provided at time of image creation in VkImageDrmFormatModifierListCreateInfoEXT::pDrmFormatModifiers. If the image was created with VkImageDrmFormatModifierExplicitCreateInfoEXT, then the returned drmFormatModifier **must** be the modifier provided at time of image creation in VkImageDrmFormatModifierExplicitCreateInfoEXT::drmFormatModifier.

### Valid Usage (Implicit)

- VUID-VkImageDrmFormatModifierPropertiesEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_IMAGE_DRM_FORMAT_MODIFIER_PROPERTIES_EXT
- VUID-VkImageDrmFormatModifierPropertiesEXT-pNext-pNext
  pNext **must** be NULL

To destroy an image, call:

```
// Provided by VK_VERSION_1_0
void vkDestroyImage(
    VkDevice                                    device,
    VkImage                                     image,
    const VkAllocationCallbacks*                pAllocator);
```

- device is the logical device that destroys the image.
- image is the image to destroy.
- pAllocator controls host memory allocation as described in the Memory Allocation chapter.

### Valid Usage

- VUID-vkDestroyImage-image-01000
  All submitted commands that refer to image, either directly or via a VkImageView, **must** have completed execution
- VUID-vkDestroyImage-image-01001
  If VkAllocationCallbacks were provided when image was created, a compatible set of callbacks **must** be provided here
- VUID-vkDestroyImage-image-01002
  If no VkAllocationCallbacks were provided when image was created, pAllocator **must** be NULL
- VUID-vkDestroyImage-image-04882
  image **must** not have been acquired from vkGetSwapchainImagesKHR

### Valid Usage (Implicit)

- VUID-vkDestroyImage-device-parameter

device **must** be a valid VkDevice handle

- VUID-vkDestroyImage-image-parameter
  If image is not VK_NULL_HANDLE, image **must** be a valid VkImage handle
- VUID-vkDestroyImage-pAllocator-parameter
  If pAllocator is not NULL, pAllocator **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkDestroyImage-image-parent
  If image is a valid handle, it **must** have been created, allocated, or retrieved from device

**Host Synchronization**

- Host access to image **must** be externally synchronized

### 12.3.1. Image Format Features

Valid uses of a VkImage **may** depend on the image's *format features*, defined below. Such constraints are documented in the affected valid usage statement.

- If the image was created with VK_IMAGE_TILING_LINEAR, then its set of *format features* is the value of VkFormatProperties::linearTilingFeatures found by calling vkGetPhysicalDeviceFormatProperties on the same format as VkImageCreateInfo::format.
- If the image was created with VK_IMAGE_TILING_OPTIMAL, but without an Android hardware buffer external format, or a QNX Screen Buffer external format or an VkBufferCollectionImageCreateInfoFUCHSIA, then its set of *format features* is the value of VkFormatProperties::optimalTilingFeatures found by calling vkGetPhysicalDeviceFormatProperties on the same format as VkImageCreateInfo::format.
- If the image was created with an Android hardware buffer external format, then its set of *format features* is the value of VkAndroidHardwareBufferFormatPropertiesANDROID::formatFeatures found by calling vkGetAndroidHardwareBufferPropertiesANDROID on the Android hardware buffer that was imported to the VkDeviceMemory to which the image is bound.
- If the image was created with an QNX Screen buffer external format, then its set of *format features* is the value of VkScreenBufferFormatPropertiesQNX::formatFeatures found by calling vkGetScreenBufferPropertiesQNX on the QNX Screen buffer that was imported to the VkDeviceMemory to which the image is bound.
- If the image was created with VK_IMAGE_TILING_DRM_FORMAT_MODIFIER_EXT, then:
  - The image's DRM format modifier is the value of VkImageDrmFormatModifierPropertiesEXT::drmFormatModifier found by calling vkGetImageDrmFormatModifierPropertiesEXT.
  - Let VkDrmFormatModifierPropertiesListEXT::pDrmFormatModifierProperties be the array found by calling vkGetPhysicalDeviceFormatProperties2 on the same format as VkImageCreateInfo::format.

- Let `VkDrmFormatModifierPropertiesEXT prop` be the array element whose `drmFormatModifier` member is the value of the image's DRM format modifier.
- Then the image's set of *format features* is the value of `prop`::`drmFormatModifierTilingFeatures`.

### 12.3.2. Corner-Sampled Images

A *corner-sampled image* is an image where unnormalized texel coordinates are centered on integer values rather than half-integer values.

A corner-sampled image has a number of differences compared to conventional texture image:

- Texels are centered on integer coordinates. See Unnormalized Texel Coordinate Operations
- Normalized coordinates are scaled using coord × (dim - 1) rather than coord × dim, where dim is the size of one dimension of the image. See normalized texel coordinate transform.
- Partial derivatives are scaled using coord × (dim - 1) rather than coord × dim. See Scale Factor Operation.
- Calculation of the next higher LOD size goes according to ⌊dim / 2⌋ rather than ⌈dim / 2⌉. See Image Mip Level Sizing.
- The minimum level size is 2x2 for 2D images and 2x2x2 for 3D images. See Image Mip Level Sizing.

Corner-sampling is only supported for 2D and 3D images. When sampling a corner-sampled image, the sampler addressing mode **must** be `VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_EDGE`. Corner-sampled images are not supported as cube maps or depth/stencil images.

### 12.3.3. Image Mip Level Sizing

A *complete mipmap chain* is the full set of mip levels, from the largest mip level provided, down to the *minimum mip level size*.

#### Conventional Images

For conventional images, the dimensions of each successive mip level, n+1, are:

$$\texttt{width}_{n+1} = \max(\lfloor \texttt{width}_n/2 \rfloor, 1)$$

$$\texttt{height}_{n+1} = \max(\lfloor \texttt{height}_n/2 \rfloor, 1)$$

$$\texttt{depth}_{n+1} = \max(\lfloor \texttt{depth}_n/2 \rfloor, 1)$$

where $\texttt{width}_n$, $\texttt{height}_n$, and $\texttt{depth}_n$ are the dimensions of the next larger mip level, n.

The minimum mip level size is:

- 1 for one-dimensional images,

- 1x1 for two-dimensional images, and
- 1x1x1 for three-dimensional images.

The number of levels in a complete mipmap chain is:

$$\lfloor \log_2(\max(\texttt{width}_0, \texttt{height}_0, \texttt{depth}_0)) \rfloor + 1$$

where $\texttt{width}_0$, $\texttt{height}_0$, and $\texttt{depth}_0$ are the dimensions of the largest (most detailed) mip level, `0`.

**Corner-Sampled Images**

For corner-sampled images, the dimensions of each successive mip level, n+1, are:

$$\texttt{width}_{n+1} = \max(\lfloor \texttt{width}_n/2 \rfloor, 2)$$

$$\texttt{height}_{n+1} = \max(\lfloor \texttt{height}_n/2 \rfloor, 2)$$

$$\texttt{depth}_{n+1} = \max(\lfloor \texttt{depth}_n/2 \rfloor, 2)$$

where $\texttt{width}_n$, $\texttt{height}_n$, and $\texttt{depth}_n$ are the dimensions of the next larger mip level, n.

The minimum mip level size is:

- 2x2 for two-dimensional images, and
- 2x2x2 for three-dimensional images.

The number of levels in a complete mipmap chain is:

$$\lfloor \log_2(\max(\texttt{width}_0, \texttt{height}_0, \texttt{depth}_0)) \rfloor$$

where $\texttt{width}_0$, $\texttt{height}_0$, and $\texttt{depth}_0$ are the dimensions of the largest (most detailed) mip level, `0`.

## 12.4. Image Layouts

Images are stored in implementation-dependent opaque layouts in memory. Each layout has limitations on what kinds of operations are supported for image subresources using the layout. At any given time, the data representing an image subresource in memory exists in a particular layout which is determined by the most recent layout transition that was performed on that image subresource. Applications have control over which layout each image subresource uses, and **can** transition an image subresource from one layout to another. Transitions **can** happen with an image memory barrier, included as part of a vkCmdPipelineBarrier or a vkCmdWaitEvents command buffer command (see Image Memory Barriers), or as part of a subpass dependency within a render pass (see `VkSubpassDependency`).

Image layout is per-image subresource. Separate image subresources of the same image **can** be in different layouts at the same time, with the exception that depth and stencil aspects of a given image subresource **can** only be in different layouts if the `separateDepthStencilLayouts` feature is enabled.

**NOTE**

Each layout **may** offer optimal performance for a specific usage of image memory. For example, an image with a layout of `VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL` **may** provide optimal performance for use as a color attachment, but be unsupported for use in transfer commands. Applications **can** transition an image subresource from one layout to another in order to achieve optimal performance when the image subresource is used for multiple kinds of operations. After initialization, applications need not use any layout other than the general layout, though this **may** produce suboptimal performance on some implementations.

Upon creation, all image subresources of an image are initially in the same layout, where that layout is selected by the `VkImageCreateInfo::initialLayout` member. The `initialLayout` **must** be either `VK_IMAGE_LAYOUT_UNDEFINED` or `VK_IMAGE_LAYOUT_PREINITIALIZED`. If it is `VK_IMAGE_LAYOUT_PREINITIALIZED`, then the image data **can** be preinitialized by the host while using this layout, and the transition away from this layout will preserve that data. If it is `VK_IMAGE_LAYOUT_UNDEFINED`, then the contents of the data are considered to be undefined, and the transition away from this layout is not guaranteed to preserve that data. For either of these initial layouts, any image subresources **must** be transitioned to another layout before they are accessed by the device.

Host access to image memory is only well-defined for linear images and for image subresources of those images which are currently in either the `VK_IMAGE_LAYOUT_PREINITIALIZED` or `VK_IMAGE_LAYOUT_GENERAL` layout. Calling vkGetImageSubresourceLayout for a linear image returns a subresource layout mapping that is valid for either of those image layouts.

The set of image layouts consists of:

```
// Provided by VK_VERSION_1_0
typedef enum VkImageLayout {
    VK_IMAGE_LAYOUT_UNDEFINED = 0,
    VK_IMAGE_LAYOUT_GENERAL = 1,
    VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL = 2,
    VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL = 3,
    VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL = 4,
    VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL = 5,
    VK_IMAGE_LAYOUT_TRANSFER_SRC_OPTIMAL = 6,
    VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL = 7,
    VK_IMAGE_LAYOUT_PREINITIALIZED = 8,
  // Provided by VK_VERSION_1_1
    VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_STENCIL_ATTACHMENT_OPTIMAL = 1000117000,
  // Provided by VK_VERSION_1_1
    VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_STENCIL_READ_ONLY_OPTIMAL = 1000117001,
    VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL = 1000241000,
    VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL = 1000241001,
    VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL = 1000241002,
```

```
    VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL = 1000241003,
    VK_IMAGE_LAYOUT_READ_ONLY_OPTIMAL = 1000314000,
    VK_IMAGE_LAYOUT_ATTACHMENT_OPTIMAL = 1000314001,
  // Provided by VK_KHR_swapchain
    VK_IMAGE_LAYOUT_PRESENT_SRC_KHR = 1000001002,
  // Provided by VK_KHR_video_decode_queue
    VK_IMAGE_LAYOUT_VIDEO_DECODE_DST_KHR = 1000024000,
  // Provided by VK_KHR_video_decode_queue
    VK_IMAGE_LAYOUT_VIDEO_DECODE_SRC_KHR = 1000024001,
  // Provided by VK_KHR_video_decode_queue
    VK_IMAGE_LAYOUT_VIDEO_DECODE_DPB_KHR = 1000024002,
  // Provided by VK_KHR_shared_presentable_image
    VK_IMAGE_LAYOUT_SHARED_PRESENT_KHR = 1000111000,
  // Provided by VK_EXT_fragment_density_map
    VK_IMAGE_LAYOUT_FRAGMENT_DENSITY_MAP_OPTIMAL_EXT = 1000218000,
  // Provided by VK_KHR_fragment_shading_rate
    VK_IMAGE_LAYOUT_FRAGMENT_SHADING_RATE_ATTACHMENT_OPTIMAL_KHR = 1000164003,
  // Provided by VK_KHR_dynamic_rendering_local_read
    VK_IMAGE_LAYOUT_RENDERING_LOCAL_READ_KHR = 1000232000,
  // Provided by VK_KHR_video_encode_queue
    VK_IMAGE_LAYOUT_VIDEO_ENCODE_DST_KHR = 1000299000,
  // Provided by VK_KHR_video_encode_queue
    VK_IMAGE_LAYOUT_VIDEO_ENCODE_SRC_KHR = 1000299001,
  // Provided by VK_KHR_video_encode_queue
    VK_IMAGE_LAYOUT_VIDEO_ENCODE_DPB_KHR = 1000299002,
  // Provided by VK_EXT_attachment_feedback_loop_layout
    VK_IMAGE_LAYOUT_ATTACHMENT_FEEDBACK_LOOP_OPTIMAL_EXT = 1000339000,
  // Provided by VK_KHR_maintenance2
    VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_STENCIL_ATTACHMENT_OPTIMAL_KHR =
VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_STENCIL_ATTACHMENT_OPTIMAL,
  // Provided by VK_KHR_maintenance2
    VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_STENCIL_READ_ONLY_OPTIMAL_KHR =
VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_STENCIL_READ_ONLY_OPTIMAL,
  // Provided by VK_NV_shading_rate_image
    VK_IMAGE_LAYOUT_SHADING_RATE_OPTIMAL_NV =
VK_IMAGE_LAYOUT_FRAGMENT_SHADING_RATE_ATTACHMENT_OPTIMAL_KHR,
  // Provided by VK_KHR_separate_depth_stencil_layouts
    VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL_KHR =
VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL,
  // Provided by VK_KHR_separate_depth_stencil_layouts
    VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL_KHR =
VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL,
  // Provided by VK_KHR_separate_depth_stencil_layouts
    VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL_KHR =
VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL,
  // Provided by VK_KHR_separate_depth_stencil_layouts
    VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL_KHR =
VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL,
  // Provided by VK_KHR_synchronization2
    VK_IMAGE_LAYOUT_READ_ONLY_OPTIMAL_KHR = VK_IMAGE_LAYOUT_READ_ONLY_OPTIMAL,
  // Provided by VK_KHR_synchronization2
```

```
    VK_IMAGE_LAYOUT_ATTACHMENT_OPTIMAL_KHR = VK_IMAGE_LAYOUT_ATTACHMENT_OPTIMAL,
} VkImageLayout;
```

The type(s) of device access supported by each layout are:

- VK_IMAGE_LAYOUT_UNDEFINED specifies that the layout is unknown. Image memory **cannot** be transitioned into this layout. This layout **can** be used as the initialLayout member of VkImageCreateInfo. This layout **can** be used in place of the current image layout in a layout transition, but doing so will cause the contents of the image's memory to be undefined.
- VK_IMAGE_LAYOUT_PREINITIALIZED specifies that an image's memory is in a defined layout and **can** be populated by data, but that it has not yet been initialized by the driver. Image memory **cannot** be transitioned into this layout. This layout **can** be used as the initialLayout member of VkImageCreateInfo. This layout is intended to be used as the initial layout for an image whose contents are written by the host, and hence the data **can** be written to memory immediately, without first executing a layout transition. Currently, VK_IMAGE_LAYOUT_PREINITIALIZED is only useful with linear images because there is not a standard layout defined for VK_IMAGE_TILING_OPTIMAL images.
- VK_IMAGE_LAYOUT_GENERAL supports all types of device access.
- VK_IMAGE_LAYOUT_ATTACHMENT_OPTIMAL specifies a layout that **must** only be used with attachment accesses in the graphics pipeline.
- VK_IMAGE_LAYOUT_READ_ONLY_OPTIMAL specifies a layout allowing read only access as an attachment, or in shaders as a sampled image, combined image/sampler, or input attachment.
- VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL **must** only be used as a color or resolve attachment in a VkFramebuffer. This layout is valid only for image subresources of images created with the VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT usage bit enabled.
- VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL specifies a layout for both the depth and stencil aspects of a depth/stencil format image allowing read and write access as a depth/stencil attachment. It is equivalent to VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL and VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL.
- VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL specifies a layout for both the depth and stencil aspects of a depth/stencil format image allowing read only access as a depth/stencil attachment or in shaders as a sampled image, combined image/sampler, or input attachment. It is equivalent to VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL and VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL.
- VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_STENCIL_ATTACHMENT_OPTIMAL specifies a layout for depth/stencil format images allowing read and write access to the stencil aspect as a stencil attachment, and read only access to the depth aspect as a depth attachment or in shaders as a sampled image, combined image/sampler, or input attachment. It is equivalent to VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL and VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL.
- VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_STENCIL_READ_ONLY_OPTIMAL specifies a layout for depth/stencil format images allowing read and write access to the depth aspect as a depth attachment, and read only access to the stencil aspect as a stencil attachment or in shaders as a sampled image, combined image/sampler, or input attachment. It is equivalent to VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL and VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL.

- VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL specifies a layout for the depth aspect of a depth/stencil format image allowing read and write access as a depth attachment.
- VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL specifies a layout for the depth aspect of a depth/stencil format image allowing read-only access as a depth attachment or in shaders as a sampled image, combined image/sampler, or input attachment.
- VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL specifies a layout for the stencil aspect of a depth/stencil format image allowing read and write access as a stencil attachment.
- VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL specifies a layout for the stencil aspect of a depth/stencil format image allowing read-only access as a stencil attachment or in shaders as a sampled image, combined image/sampler, or input attachment.
- VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL specifies a layout allowing read-only access in a shader as a sampled image, combined image/sampler, or input attachment. This layout is valid only for image subresources of images created with the VK_IMAGE_USAGE_SAMPLED_BIT or VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT usage bits enabled.
- VK_IMAGE_LAYOUT_TRANSFER_SRC_OPTIMAL **must** only be used as a source image of a transfer command (see the definition of VK_PIPELINE_STAGE_TRANSFER_BIT). This layout is valid only for image subresources of images created with the VK_IMAGE_USAGE_TRANSFER_SRC_BIT usage bit enabled.
- VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL **must** only be used as a destination image of a transfer command. This layout is valid only for image subresources of images created with the VK_IMAGE_USAGE_TRANSFER_DST_BIT usage bit enabled.
- VK_IMAGE_LAYOUT_PRESENT_SRC_KHR **must** only be used for presenting a presentable image for display.
- VK_IMAGE_LAYOUT_SHARED_PRESENT_KHR is valid only for shared presentable images, and **must** be used for any usage the image supports.
- VK_IMAGE_LAYOUT_FRAGMENT_SHADING_RATE_ATTACHMENT_OPTIMAL_KHR **must** only be used as a fragment shading rate attachment or shading rate image. This layout is valid only for image subresources of images created with the VK_IMAGE_USAGE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR usage bit enabled.
- VK_IMAGE_LAYOUT_FRAGMENT_DENSITY_MAP_OPTIMAL_EXT **must** only be used as a fragment density map attachment in a VkRenderPass. This layout is valid only for image subresources of images created with the VK_IMAGE_USAGE_FRAGMENT_DENSITY_MAP_BIT_EXT usage bit enabled.
- VK_IMAGE_LAYOUT_VIDEO_DECODE_DST_KHR **must** only be used as a decode output picture in a video decode operation. This layout is valid only for image subresources of images created with the VK_IMAGE_USAGE_VIDEO_DECODE_DST_BIT_KHR usage bit enabled.
- VK_IMAGE_LAYOUT_VIDEO_DECODE_SRC_KHR is reserved for future use.
- VK_IMAGE_LAYOUT_VIDEO_DECODE_DPB_KHR **must** only be used as an output reconstructed picture or an input reference picture in a video decode operation. This layout is valid only for image subresources of images created with the VK_IMAGE_USAGE_VIDEO_DECODE_DPB_BIT_KHR usage bit enabled.
- VK_IMAGE_LAYOUT_VIDEO_ENCODE_DST_KHR is reserved for future use.
- VK_IMAGE_LAYOUT_VIDEO_ENCODE_SRC_KHR **must** only be used as an encode input picture in a video

encode operation. This layout is valid only for image subresources of images created with the VK_IMAGE_USAGE_VIDEO_ENCODE_SRC_BIT_KHR usage bit enabled.

- VK_IMAGE_LAYOUT_VIDEO_ENCODE_DPB_KHR **must** only be used as an output reconstructed picture or an input reference picture in a video encode operation. This layout is valid only for image subresources of images created with the VK_IMAGE_USAGE_VIDEO_ENCODE_DPB_BIT_KHR usage bit enabled.
- VK_IMAGE_LAYOUT_ATTACHMENT_FEEDBACK_LOOP_OPTIMAL_EXT **must** only be used as either a color attachment or depth/stencil attachment in a VkFramebuffer and/or read-only access in a shader as a sampled image, combined image/sampler, or input attachment. This layout is valid only for image subresources of images created with the VK_IMAGE_USAGE_ATTACHMENT_FEEDBACK_LOOP_BIT_EXT usage bit enabled and either the VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT or VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT and either the VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT or VK_IMAGE_USAGE_SAMPLED_BIT usage bits enabled.
- VK_IMAGE_LAYOUT_RENDERING_LOCAL_READ_KHR **must** only be used as either a storage image, or a color or depth/stencil attachment and an input attachment. This layout is valid only for image subresources of images created with either VK_IMAGE_USAGE_STORAGE_BIT, or both VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT and either of VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT or VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT.

The layout of each image subresource is not a state of the image subresource itself, but is rather a property of how the data in memory is organized, and thus for each mechanism of accessing an image in the API the application **must** specify a parameter or structure member that indicates which image layout the image subresource(s) are considered to be in when the image will be accessed. For transfer commands, this is a parameter to the command (see Clear Commands and Copy Commands). For use as a framebuffer attachment, this is a member in the substructures of the VkRenderPassCreateInfo (see Render Pass). For use in a descriptor set, this is a member in the VkDescriptorImageInfo structure (see Descriptor Set Updates).

### 12.4.1. Image Layout Matching Rules

At the time that any command buffer command accessing an image executes on any queue, the layouts of the image subresources that are accessed **must** all match exactly the layout specified via the API controlling those accesses, except in case of accesses to an image with a depth/stencil format performed through descriptors referring to only a single aspect of the image, where the following relaxed matching rules apply:

- Descriptors referring just to the depth aspect of a depth/stencil image only need to match in the image layout of the depth aspect, thus VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL and VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_STENCIL_ATTACHMENT_OPTIMAL are considered to match.
- Descriptors referring just to the stencil aspect of a depth/stencil image only need to match in the image layout of the stencil aspect, thus VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL and VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_STENCIL_READ_ONLY_OPTIMAL are considered to match.

When performing a layout transition on an image subresource, the old layout value **must** either equal the current layout of the image subresource (at the time the transition executes), or else be VK_IMAGE_LAYOUT_UNDEFINED (implying that the contents of the image subresource need not be preserved). The new layout used in a transition **must** not be VK_IMAGE_LAYOUT_UNDEFINED or

VK_IMAGE_LAYOUT_PREINITIALIZED.

The image layout of each image subresource of a depth/stencil image created with VK_IMAGE_CREATE_SAMPLE_LOCATIONS_COMPATIBLE_DEPTH_BIT_EXT is dependent on the last sample locations used to render to the image subresource as a depth/stencil attachment, thus applications **must** provide the same sample locations that were last used to render to the given image subresource whenever a layout transition of the image subresource happens, otherwise the contents of the depth aspect of the image subresource become undefined.

In addition, depth reads from a depth/stencil attachment referring to an image subresource range of a depth/stencil image created with VK_IMAGE_CREATE_SAMPLE_LOCATIONS_COMPATIBLE_DEPTH_BIT_EXT using different sample locations than what have been last used to perform depth writes to the image subresources of the same image subresource range return undefined values.

Similarly, depth writes to a depth/stencil attachment referring to an image subresource range of a depth/stencil image created with VK_IMAGE_CREATE_SAMPLE_LOCATIONS_COMPATIBLE_DEPTH_BIT_EXT using different sample locations than what have been last used to perform depth writes to the image subresources of the same image subresource range make the contents of the depth aspect of those image subresources undefined.

## 12.5. Image Views

Image objects are not directly accessed by pipeline shaders for reading or writing image data. Instead, *image views* representing contiguous ranges of the image subresources and containing additional metadata are used for that purpose. Views **must** be created on images of compatible types, and **must** represent a valid subset of image subresources.

Image views are represented by VkImageView handles:

```
// Provided by VK_VERSION_1_0
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkImageView)
```

VK_REMAINING_ARRAY_LAYERS is a special constant value used for image views to indicate that all remaining array layers in an image after the base layer should be included in the view.

```
#define VK_REMAINING_ARRAY_LAYERS         (~0U)
```

VK_REMAINING_MIP_LEVELS is a special constant value used for image views to indicate that all remaining mipmap levels in an image after the base level should be included in the view.

```
#define VK_REMAINING_MIP_LEVELS           (~0U)
```

The types of image views that **can** be created are:

```
// Provided by VK_VERSION_1_0
typedef enum VkImageViewType {
    VK_IMAGE_VIEW_TYPE_1D = 0,
    VK_IMAGE_VIEW_TYPE_2D = 1,
    VK_IMAGE_VIEW_TYPE_3D = 2,
    VK_IMAGE_VIEW_TYPE_CUBE = 3,
    VK_IMAGE_VIEW_TYPE_1D_ARRAY = 4,
    VK_IMAGE_VIEW_TYPE_2D_ARRAY = 5,
    VK_IMAGE_VIEW_TYPE_CUBE_ARRAY = 6,
} VkImageViewType;
```

To create an image view, call:

```
// Provided by VK_VERSION_1_0
VkResult vkCreateImageView(
    VkDevice                                    device,
    const VkImageViewCreateInfo*                pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkImageView*                                pView);
```

- `device` is the logical device that creates the image view.
- `pCreateInfo` is a pointer to a `VkImageViewCreateInfo` structure containing parameters to be used to create the image view.
- `pAllocator` controls host memory allocation as described in the Memory Allocation chapter.
- `pView` is a pointer to a VkImageView handle in which the resulting image view object is returned.

## Valid Usage

- VUID-vkCreateImageView-device-09667
  `device` **must** support at least one queue family with one of the `VK_QUEUE_VIDEO_ENCODE_BIT_KHR`, `VK_QUEUE_VIDEO_DECODE_BIT_KHR`, `VK_QUEUE_COMPUTE_BIT`, or `VK_QUEUE_GRAPHICS_BIT` capabilities
- VUID-vkCreateImageView-image-09179
  VkImageViewCreateInfo::`image` **must** have been created from `device`

## Valid Usage (Implicit)

- VUID-vkCreateImageView-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkCreateImageView-pCreateInfo-parameter
  `pCreateInfo` **must** be a valid pointer to a valid VkImageViewCreateInfo structure
- VUID-vkCreateImageView-pAllocator-parameter

If pAllocator is not NULL, pAllocator **must** be a valid pointer to a valid VkAllocationCallbacks structure

- VUID-vkCreateImageView-pView-parameter
  pView **must** be a valid pointer to a VkImageView handle

## Return Codes

**Success**

- VK_SUCCESS

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY
- VK_ERROR_INVALID_OPAQUE_CAPTURE_ADDRESS_KHR

The VkImageViewCreateInfo structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkImageViewCreateInfo {
    VkStructureType            sType;
    const void*                pNext;
    VkImageViewCreateFlags     flags;
    VkImage                    image;
    VkImageViewType            viewType;
    VkFormat                   format;
    VkComponentMapping         components;
    VkImageSubresourceRange    subresourceRange;
} VkImageViewCreateInfo;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- flags is a bitmask of VkImageViewCreateFlagBits specifying additional parameters of the image view.
- image is a VkImage on which the view will be created.
- viewType is a VkImageViewType value specifying the type of the image view.
- format is a VkFormat specifying the format and type used to interpret texel blocks of the image.
- components is a VkComponentMapping structure specifying a remapping of color components (or of depth or stencil components after they have been converted into color components).
- subresourceRange is a VkImageSubresourceRange structure selecting the set of mipmap levels and array layers to be accessible to the view.

Some of the image creation parameters are inherited by the view. In particular, image view creation

inherits the implicit parameter usage specifying the allowed usages of the image view that, by default, takes the value of the corresponding usage parameter specified in VkImageCreateInfo at image creation time. The implicit usage **can** be overridden by adding a VkImageViewUsageCreateInfo structure to the pNext chain, but the view usage **must** be a subset of the image usage. If image has a depth-stencil format and was created with a VkImageStencilUsageCreateInfo structure included in the pNext chain of VkImageCreateInfo, the usage is calculated based on the subresource.aspectMask provided:

- If aspectMask includes only VK_IMAGE_ASPECT_STENCIL_BIT, the implicit usage is equal to VkImageStencilUsageCreateInfo::stencilUsage.
- If aspectMask includes only VK_IMAGE_ASPECT_DEPTH_BIT, the implicit usage is equal to VkImageCreateInfo::usage.
- If both aspects are included in aspectMask, the implicit usage is equal to the intersection of VkImageCreateInfo::usage and VkImageStencilUsageCreateInfo::stencilUsage.

If image is a 3D image, its Z range **can** be restricted to a subset by adding a VkImageViewSlicedCreateInfoEXT to the pNext chain.

If image was created with the VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT flag, and if the format of the image is not multi-planar, format **can** be different from the image's format, but if image was created without the VK_IMAGE_CREATE_BLOCK_TEXEL_VIEW_COMPATIBLE_BIT flag and they are not equal they **must** be *compatible*. Image format compatibility is defined in the Format Compatibility Classes section. Views of compatible formats will have the same mapping between texel coordinates and memory locations irrespective of the format, with only the interpretation of the bit pattern changing.

If image was created with a multi-planar format, and the image view's aspectMask is one of VK_IMAGE_ASPECT_PLANE_0_BIT, VK_IMAGE_ASPECT_PLANE_1_BIT or VK_IMAGE_ASPECT_PLANE_2_BIT, the view's aspect mask is considered to be equivalent to VK_IMAGE_ASPECT_COLOR_BIT when used as a framebuffer attachment.

**NOTE** Values intended to be used with one view format **may** not be exactly preserved when written or read through a different format. For example, an integer value that happens to have the bit pattern of a floating-point denorm or NaN **may** be flushed or canonicalized when written or read through a view with a floating-point format. Similarly, a value written through a signed normalized format that has a bit pattern exactly equal to $-2^b$ **may** be changed to $-2^b + 1$ as described in Conversion from Normalized Fixed-Point to Floating-Point.

If image was created with the VK_IMAGE_CREATE_BLOCK_TEXEL_VIEW_COMPATIBLE_BIT flag, format **must** be *compatible* with the image's format as described above; or **must** be an uncompressed format, in which case it **must** be *size-compatible* with the image's format. In this case, the resulting image view's texel dimensions equal the dimensions of the selected mip level divided by the compressed texel block size and rounded up.

The VkComponentMapping components member describes a remapping from components of the image to components of the vector returned by shader image instructions. This remapping **must** be the identity swizzle for storage image descriptors, input attachment descriptors, framebuffer attachments, and any VkImageView used with a combined image sampler that enables sampler $Y'C_BC_R$

conversion.

If the image view is to be used with a sampler which supports sampler $Y'C_BC_R$ conversion, an *identically defined object* of type VkSamplerYcbcrConversion to that used to create the sampler **must** be passed to vkCreateImageView in a VkSamplerYcbcrConversionInfo included in the `pNext` chain of VkImageViewCreateInfo. Conversely, if a VkSamplerYcbcrConversion object is passed to vkCreateImageView, an identically defined VkSamplerYcbcrConversion object **must** be used when sampling the image.

If the image has a multi-planar `format`, `subresourceRange.aspectMask` is `VK_IMAGE_ASPECT_COLOR_BIT`, and `usage` includes `VK_IMAGE_USAGE_SAMPLED_BIT`, then the `format` **must** be identical to the image `format` and the sampler to be used with the image view **must** enable sampler $Y'C_BC_R$ conversion.

When such an image is used in a video coding operation, the sampler $Y'C_BC_R$ conversion has no effect.

If `image` was created with the `VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT` and the image has a multi-planar `format`, and if `subresourceRange.aspectMask` is `VK_IMAGE_ASPECT_PLANE_0_BIT`, `VK_IMAGE_ASPECT_PLANE_1_BIT`, or `VK_IMAGE_ASPECT_PLANE_2_BIT`, `format` **must** be compatible with the corresponding plane of the image, and the sampler to be used with the image view **must** not enable sampler $Y'C_BC_R$ conversion. The `width` and `height` of the single-plane image view **must** be derived from the multi-planar image's dimensions in the manner listed for plane compatibility for the plane.

Any view of an image plane will have the same mapping between texel coordinates and memory locations as used by the components of the color aspect, subject to the formulae relating texel coordinates to lower-resolution planes as described in Chroma Reconstruction. That is, if an R or B plane has a reduced resolution relative to the G plane of the multi-planar image, the image view operates using the ($u_{plane}$, $v_{plane}$) unnormalized coordinates of the reduced-resolution plane, and these coordinates access the same memory locations as the ($u_{color}$, $v_{color}$) unnormalized coordinates of the color aspect for which chroma reconstruction operations operate on the same ($u_{plane}$, $v_{plane}$) or ($i_{plane}$, $j_{plane}$) coordinates.

*Table 16. Image type and image view type compatibility requirements*

| Image View Type | Compatible Image Types |
|---|---|
| `VK_IMAGE_VIEW_TYPE_1D` | `VK_IMAGE_TYPE_1D` |
| `VK_IMAGE_VIEW_TYPE_1D_ARRAY` | `VK_IMAGE_TYPE_1D` |
| `VK_IMAGE_VIEW_TYPE_2D` | `VK_IMAGE_TYPE_2D` , `VK_IMAGE_TYPE_3D` |
| `VK_IMAGE_VIEW_TYPE_2D_ARRAY` | `VK_IMAGE_TYPE_2D` , `VK_IMAGE_TYPE_3D` |
| `VK_IMAGE_VIEW_TYPE_CUBE` | `VK_IMAGE_TYPE_2D` |
| `VK_IMAGE_VIEW_TYPE_CUBE_ARRAY` | `VK_IMAGE_TYPE_2D` |
| `VK_IMAGE_VIEW_TYPE_3D` | `VK_IMAGE_TYPE_3D` |

## Valid Usage

- VUID-VkImageViewCreateInfo-image-01003

If image was not created with VK_IMAGE_CREATE_CUBE_COMPATIBLE_BIT then viewType **must** not be VK_IMAGE_VIEW_TYPE_CUBE or VK_IMAGE_VIEW_TYPE_CUBE_ARRAY

- VUID-VkImageViewCreateInfo-viewType-01004
  If the imageCubeArray feature is not enabled, viewType **must** not be VK_IMAGE_VIEW_TYPE_CUBE_ARRAY
- VUID-VkImageViewCreateInfo-image-06723
  If image was created with VK_IMAGE_TYPE_3D but without VK_IMAGE_CREATE_2D_ARRAY_COMPATIBLE_BIT set then viewType **must** not be VK_IMAGE_VIEW_TYPE_2D_ARRAY
- VUID-VkImageViewCreateInfo-image-06728
  If image was created with VK_IMAGE_TYPE_3D but without VK_IMAGE_CREATE_2D_ARRAY_COMPATIBLE_BIT or VK_IMAGE_CREATE_2D_VIEW_COMPATIBLE_BIT_EXT set, then viewType **must** not be VK_IMAGE_VIEW_TYPE_2D
- VUID-VkImageViewCreateInfo-image-04970
  If image was created with VK_IMAGE_TYPE_3D and viewType is VK_IMAGE_VIEW_TYPE_2D or VK_IMAGE_VIEW_TYPE_2D_ARRAY then subresourceRange.levelCount **must** be 1
- VUID-VkImageViewCreateInfo-image-04971
  If image was created with VK_IMAGE_TYPE_3D and viewType is VK_IMAGE_VIEW_TYPE_2D or VK_IMAGE_VIEW_TYPE_2D_ARRAY then VkImageCreateInfo::flags **must** not contain any of VK_IMAGE_CREATE_SPARSE_BINDING_BIT, VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT, and VK_IMAGE_CREATE_SPARSE_ALIASED_BIT
- VUID-VkImageViewCreateInfo-image-04972
  If image was created with a samples value not equal to VK_SAMPLE_COUNT_1_BIT then viewType **must** be either VK_IMAGE_VIEW_TYPE_2D or VK_IMAGE_VIEW_TYPE_2D_ARRAY
- VUID-VkImageViewCreateInfo-image-04441
  image **must** have been created with a usage value containing at least one of the usages defined in the valid image usage list for image views
- VUID-VkImageViewCreateInfo-None-02273
  The format features of the resultant image view **must** contain at least one bit
- VUID-VkImageViewCreateInfo-usage-02274
  If usage contains VK_IMAGE_USAGE_SAMPLED_BIT, then the format features of the resultant image view **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_BIT
- VUID-VkImageViewCreateInfo-usage-02275
  If usage contains VK_IMAGE_USAGE_STORAGE_BIT, then the image view's format features **must** contain VK_FORMAT_FEATURE_STORAGE_IMAGE_BIT
- VUID-VkImageViewCreateInfo-usage-08931
  If usage contains VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT, then the image view's format features **must** contain VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BIT or VK_FORMAT_FEATURE_2_LINEAR_COLOR_ATTACHMENT_BIT_NV
- VUID-VkImageViewCreateInfo-usage-02277
  If usage contains VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT, then the image view's format features **must** contain VK_FORMAT_FEATURE_DEPTH_STENCIL_ATTACHMENT_BIT
- VUID-VkImageViewCreateInfo-image-08333

If image was created with VK_IMAGE_CREATE_VIDEO_PROFILE_INDEPENDENT_BIT_KHR and usage contains VK_IMAGE_USAGE_VIDEO_DECODE_DST_BIT_KHR, then the image view's format features **must** contain VK_FORMAT_FEATURE_VIDEO_DECODE_OUTPUT_BIT_KHR

- VUID-VkImageViewCreateInfo-image-08334
  If image was created with VK_IMAGE_CREATE_VIDEO_PROFILE_INDEPENDENT_BIT_KHR and usage contains VK_IMAGE_USAGE_VIDEO_DECODE_DPB_BIT_KHR, then the image view's format features **must** contain VK_FORMAT_FEATURE_VIDEO_DECODE_DPB_BIT_KHR
- VUID-VkImageViewCreateInfo-image-08335
  If image was created with VK_IMAGE_CREATE_VIDEO_PROFILE_INDEPENDENT_BIT_KHR, then usage **must** not include VK_IMAGE_USAGE_VIDEO_DECODE_SRC_BIT_KHR
- VUID-VkImageViewCreateInfo-image-08336
  If image was created with VK_IMAGE_CREATE_VIDEO_PROFILE_INDEPENDENT_BIT_KHR and usage contains VK_IMAGE_USAGE_VIDEO_ENCODE_SRC_BIT_KHR, then the image view's format features **must** contain VK_FORMAT_FEATURE_VIDEO_ENCODE_INPUT_BIT_KHR
- VUID-VkImageViewCreateInfo-image-08337
  If image was created with VK_IMAGE_CREATE_VIDEO_PROFILE_INDEPENDENT_BIT_KHR and usage contains VK_IMAGE_USAGE_VIDEO_ENCODE_DPB_BIT_KHR, then the image view's format features **must** contain VK_FORMAT_FEATURE_VIDEO_ENCODE_DPB_BIT_KHR
- VUID-VkImageViewCreateInfo-image-08338
  If image was created with VK_IMAGE_CREATE_VIDEO_PROFILE_INDEPENDENT_BIT_KHR, then usage **must** not include VK_IMAGE_USAGE_VIDEO_ENCODE_DST_BIT_KHR
- VUID-VkImageViewCreateInfo-usage-08932
  If usage contains VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT, and any of the following is true:
  - the externalFormatResolve feature is not enabled
  - the nullColorAttachmentWithExternalFormatResolve property is VK_FALSE
  - image was created with an VkExternalFormatANDROID::externalFormat value of 0

  then the image view's format features **must** contain at least one of VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BIT or VK_FORMAT_FEATURE_DEPTH_STENCIL_ATTACHMENT_BIT or VK_FORMAT_FEATURE_2_LINEAR_COLOR_ATTACHMENT_BIT_NV
- VUID-VkImageViewCreateInfo-subresourceRange-01478
  subresourceRange.baseMipLevel **must** be less than the mipLevels specified in VkImageCreateInfo when image was created
- VUID-VkImageViewCreateInfo-subresourceRange-01718
  If subresourceRange.levelCount is not VK_REMAINING_MIP_LEVELS, subresourceRange.baseMipLevel + subresourceRange.levelCount **must** be less than or equal to the mipLevels specified in VkImageCreateInfo when image was created
- VUID-VkImageViewCreateInfo-image-02571
  If image was created with usage containing VK_IMAGE_USAGE_FRAGMENT_DENSITY_MAP_BIT_EXT, subresourceRange.levelCount **must** be 1
- VUID-VkImageViewCreateInfo-image-06724
  If image is not a 3D image created with VK_IMAGE_CREATE_2D_ARRAY_COMPATIBLE_BIT or

VK_IMAGE_CREATE_2D_VIEW_COMPATIBLE_BIT_EXT set, or viewType is not VK_IMAGE_VIEW_TYPE_2D or VK_IMAGE_VIEW_TYPE_2D_ARRAY, subresourceRange.baseArrayLayer **must** be less than the arrayLayers specified in VkImageCreateInfo when image was created

- VUID-VkImageViewCreateInfo-subresourceRange-06725
  If subresourceRange.layerCount is not VK_REMAINING_ARRAY_LAYERS, image is not a 3D image created with VK_IMAGE_CREATE_2D_ARRAY_COMPATIBLE_BIT or VK_IMAGE_CREATE_2D_VIEW_COMPATIBLE_BIT_EXT set, or viewType is not VK_IMAGE_VIEW_TYPE_2D or VK_IMAGE_VIEW_TYPE_2D_ARRAY, subresourceRange.layerCount **must** be non-zero and subresourceRange.baseArrayLayer + subresourceRange.layerCount **must** be less than or equal to the arrayLayers specified in VkImageCreateInfo when image was created
- VUID-VkImageViewCreateInfo-image-02724
  If image is a 3D image created with VK_IMAGE_CREATE_2D_ARRAY_COMPATIBLE_BIT set, and viewType is VK_IMAGE_VIEW_TYPE_2D or VK_IMAGE_VIEW_TYPE_2D_ARRAY, subresourceRange.baseArrayLayer **must** be less than the depth computed from baseMipLevel and extent.depth specified in VkImageCreateInfo when image was created, according to the formula defined in Image Mip Level Sizing
- VUID-VkImageViewCreateInfo-subresourceRange-02725
  If subresourceRange.layerCount is not VK_REMAINING_ARRAY_LAYERS, image is a 3D image created with VK_IMAGE_CREATE_2D_ARRAY_COMPATIBLE_BIT set, and viewType is VK_IMAGE_VIEW_TYPE_2D or VK_IMAGE_VIEW_TYPE_2D_ARRAY, subresourceRange.layerCount **must** be non-zero and subresourceRange.baseArrayLayer + subresourceRange.layerCount **must** be less than or equal to the depth computed from baseMipLevel and extent.depth specified in VkImageCreateInfo when image was created, according to the formula defined in Image Mip Level Sizing
- VUID-VkImageViewCreateInfo-image-01761
  If image was created with the VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT flag, but without the VK_IMAGE_CREATE_BLOCK_TEXEL_VIEW_COMPATIBLE_BIT flag, and if the format of the image is not a multi-planar format, format **must** be compatible with the format used to create image, as defined in Format Compatibility Classes
- VUID-VkImageViewCreateInfo-image-01583
  If image was created with the VK_IMAGE_CREATE_BLOCK_TEXEL_VIEW_COMPATIBLE_BIT flag, format **must** be compatible with, or **must** be an uncompressed format that is size-compatible with, the format used to create image
- VUID-VkImageViewCreateInfo-image-07072
  If image was created with the VK_IMAGE_CREATE_BLOCK_TEXEL_VIEW_COMPATIBLE_BIT flag and format is a non-compressed format, the levelCount member of subresourceRange **must** be 1
- VUID-VkImageViewCreateInfo-image-09487
  If image was created with the VK_IMAGE_CREATE_BLOCK_TEXEL_VIEW_COMPATIBLE_BIT flag, the VkPhysicalDeviceMaintenance6PropertiesKHR::blockTexelViewCompatibleMultipleLayers property is not set to VK_TRUE, and format is a non-compressed format, then the layerCount member of subresourceRange **must** be 1
- VUID-VkImageViewCreateInfo-pNext-01585
  If a VkImageFormatListCreateInfo structure was included in the pNext chain of the VkImageCreateInfo structure used when creating image and VkImageFormatListCreateInfo::viewFormatCount is not zero then format **must** be one of the

formats in VkImageFormatListCreateInfo::pViewFormats

- VUID-VkImageViewCreateInfo-image-01586
  If image was created with the VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT flag, if the format of the image is a multi-planar format, and if subresourceRange.aspectMask is one of the multi-planar aspect mask bits, then format **must** be compatible with the VkFormat for the plane of the image format indicated by subresourceRange.aspectMask, as defined in Compatible Formats of Planes of Multi-Planar Formats
- VUID-VkImageViewCreateInfo-subresourceRange-07818
  subresourceRange.aspectMask **must** only have at most 1 valid multi-planar aspect mask bit
- VUID-VkImageViewCreateInfo-image-01762
  If image was not created with the VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT flag, or if the format of the image is a multi-planar format and if subresourceRange.aspectMask is VK_IMAGE_ASPECT_COLOR_BIT, format **must** be identical to the format used to create image
- VUID-VkImageViewCreateInfo-format-06415
  If the image view requires a sampler $Y'C_BC_R$ conversion and usage contains VK_IMAGE_USAGE_SAMPLED_BIT, then the pNext chain **must** include a VkSamplerYcbcrConversionInfo structure with a conversion value other than VK_NULL_HANDLE
- VUID-VkImageViewCreateInfo-format-04714
  If format has a _422 or _420 suffix then image **must** have been created with a width that is a multiple of 2
- VUID-VkImageViewCreateInfo-format-04715
  If format has a _420 suffix then image **must** have been created with a height that is a multiple of 2
- VUID-VkImageViewCreateInfo-pNext-01970
  If the pNext chain includes a VkSamplerYcbcrConversionInfo structure with a conversion value other than VK_NULL_HANDLE, all members of components **must** have the identity swizzle
- VUID-VkImageViewCreateInfo-pNext-06658
  If the pNext chain includes a VkSamplerYcbcrConversionInfo structure with a conversion value other than VK_NULL_HANDLE, format **must** be the same used in VkSamplerYcbcrConversionCreateInfo::format
- VUID-VkImageViewCreateInfo-image-01020
  If image is non-sparse then it **must** be bound completely and contiguously to a single VkDeviceMemory object
- VUID-VkImageViewCreateInfo-subResourceRange-01021
  viewType **must** be compatible with the type of image as shown in the view type compatibility table
- VUID-VkImageViewCreateInfo-image-02399
  If image has an Android external format, format **must** be VK_FORMAT_UNDEFINED
- VUID-VkImageViewCreateInfo-image-02400
  If image has an Android external format, the pNext chain **must** include a VkSamplerYcbcrConversionInfo structure with a conversion object created with the same

external format as image

- VUID-VkImageViewCreateInfo-image-02401
  If image has an Android external format, all members of components **must** be the identity swizzle
- VUID-VkImageViewCreateInfo-image-08957
  If image has an QNX Screen external format, format **must** be VK_FORMAT_UNDEFINED
- VUID-VkImageViewCreateInfo-image-08958
  If image has an QNX Screen external format, the pNext chain **must** include a VkSamplerYcbcrConversionInfo structure with a conversion object created with the same external format as image
- VUID-VkImageViewCreateInfo-image-08959
  If image has an QNX Screen external format, all members of components **must** be the identity swizzle
- VUID-VkImageViewCreateInfo-image-02086
  If image was created with usage containing VK_IMAGE_USAGE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR, viewType **must** be VK_IMAGE_VIEW_TYPE_2D or VK_IMAGE_VIEW_TYPE_2D_ARRAY
- VUID-VkImageViewCreateInfo-image-02087
  If the shadingRateImage feature is enabled, and If image was created with usage containing VK_IMAGE_USAGE_SHADING_RATE_IMAGE_BIT_NV, format **must** be VK_FORMAT_R8_UINT
- VUID-VkImageViewCreateInfo-usage-04550
  If the attachmentFragmentShadingRate feature is enabled, and the usage for the image view includes VK_IMAGE_USAGE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR, then the image view's format features **must** contain VK_FORMAT_FEATURE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR
- VUID-VkImageViewCreateInfo-usage-04551
  If the attachmentFragmentShadingRate feature is enabled, the usage for the image view includes VK_IMAGE_USAGE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR, and layeredShadingRateAttachments is VK_FALSE, subresourceRange.layerCount **must** be 1
- VUID-VkImageViewCreateInfo-flags-02572
  If the fragmentDensityMapDynamic feature is not enabled, flags **must** not contain VK_IMAGE_VIEW_CREATE_FRAGMENT_DENSITY_MAP_DYNAMIC_BIT_EXT
- VUID-VkImageViewCreateInfo-flags-03567
  If the fragmentDensityMapDeferred feature is not enabled, flags **must** not contain VK_IMAGE_VIEW_CREATE_FRAGMENT_DENSITY_MAP_DEFERRED_BIT_EXT
- VUID-VkImageViewCreateInfo-flags-03568
  If flags contains VK_IMAGE_VIEW_CREATE_FRAGMENT_DENSITY_MAP_DEFERRED_BIT_EXT, flags **must** not contain VK_IMAGE_VIEW_CREATE_FRAGMENT_DENSITY_MAP_DYNAMIC_BIT_EXT
- VUID-VkImageViewCreateInfo-image-03569
  If image was created with flags containing VK_IMAGE_CREATE_SUBSAMPLED_BIT_EXT and usage containing VK_IMAGE_USAGE_SAMPLED_BIT, subresourceRange.layerCount **must** be less than or equal to VkPhysicalDeviceFragmentDensityMap2PropertiesEXT::maxSubsampledArrayLayers
- VUID-VkImageViewCreateInfo-invocationMask-04993

If the invocationMask feature is enabled, and if image was created with usage containing VK_IMAGE_USAGE_INVOCATION_MASK_BIT_HUAWEI, format **must** be VK_FORMAT_R8_UINT

- VUID-VkImageViewCreateInfo-flags-04116
  If flags does not contain VK_IMAGE_VIEW_CREATE_FRAGMENT_DENSITY_MAP_DYNAMIC_BIT_EXT and image was created with usage containing VK_IMAGE_USAGE_FRAGMENT_DENSITY_MAP_BIT_EXT, its flags **must** not contain any of VK_IMAGE_CREATE_PROTECTED_BIT, VK_IMAGE_CREATE_SPARSE_BINDING_BIT, VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT, or VK_IMAGE_CREATE_SPARSE_ALIASED_BIT
- VUID-VkImageViewCreateInfo-pNext-02662
  If the pNext chain includes a VkImageViewUsageCreateInfo structure, and image was not created with a VkImageStencilUsageCreateInfo structure included in the pNext chain of VkImageCreateInfo, its usage member **must** not include any bits that were not set in the usage member of the VkImageCreateInfo structure used to create image
- VUID-VkImageViewCreateInfo-pNext-02663
  If the pNext chain includes a VkImageViewUsageCreateInfo structure, image was created with a VkImageStencilUsageCreateInfo structure included in the pNext chain of VkImageCreateInfo, and subresourceRange.aspectMask includes VK_IMAGE_ASPECT_STENCIL_BIT, the usage member of the VkImageViewUsageCreateInfo structure **must** not include any bits that were not set in the usage member of the VkImageStencilUsageCreateInfo structure used to create image
- VUID-VkImageViewCreateInfo-pNext-02664
  If the pNext chain includes a VkImageViewUsageCreateInfo structure, image was created with a VkImageStencilUsageCreateInfo structure included in the pNext chain of VkImageCreateInfo, and subresourceRange.aspectMask includes bits other than VK_IMAGE_ASPECT_STENCIL_BIT, the usage member of the VkImageViewUsageCreateInfo structure **must** not include any bits that were not set in the usage member of the VkImageCreateInfo structure used to create image
- VUID-VkImageViewCreateInfo-imageViewType-04973
  If viewType is VK_IMAGE_VIEW_TYPE_1D, VK_IMAGE_VIEW_TYPE_2D, or VK_IMAGE_VIEW_TYPE_3D; and subresourceRange.layerCount is not VK_REMAINING_ARRAY_LAYERS, then subresourceRange.layerCount **must** be 1
- VUID-VkImageViewCreateInfo-imageViewType-04974
  If viewType is VK_IMAGE_VIEW_TYPE_1D, VK_IMAGE_VIEW_TYPE_2D, or VK_IMAGE_VIEW_TYPE_3D; and subresourceRange.layerCount is VK_REMAINING_ARRAY_LAYERS, then the remaining number of layers **must** be 1
- VUID-VkImageViewCreateInfo-viewType-02960
  If viewType is VK_IMAGE_VIEW_TYPE_CUBE and subresourceRange.layerCount is not VK_REMAINING_ARRAY_LAYERS, subresourceRange.layerCount **must** be 6
- VUID-VkImageViewCreateInfo-viewType-02961
  If viewType is VK_IMAGE_VIEW_TYPE_CUBE_ARRAY and subresourceRange.layerCount is not VK_REMAINING_ARRAY_LAYERS, subresourceRange.layerCount **must** be a multiple of 6
- VUID-VkImageViewCreateInfo-viewType-02962
  If viewType is VK_IMAGE_VIEW_TYPE_CUBE and subresourceRange.layerCount is VK_REMAINING_ARRAY_LAYERS, the remaining number of layers **must** be 6

- VUID-VkImageViewCreateInfo-viewType-02963
  If `viewType` is `VK_IMAGE_VIEW_TYPE_CUBE_ARRAY` and `subresourceRange.layerCount` is `VK_REMAINING_ARRAY_LAYERS`, the remaining number of layers **must** be a multiple of `6`
- VUID-VkImageViewCreateInfo-imageViewFormatSwizzle-04465
  If the `VK_KHR_portability_subset` extension is enabled, and VkPhysicalDevicePortabilitySubsetFeaturesKHR::`imageViewFormatSwizzle` is `VK_FALSE`, all elements of `components` **must** have the identity swizzle
- VUID-VkImageViewCreateInfo-imageViewFormatReinterpretation-04466
  If the `VK_KHR_portability_subset` extension is enabled, and VkPhysicalDevicePortabilitySubsetFeaturesKHR::`imageViewFormatReinterpretation` is `VK_FALSE`, the VkFormat in `format` **must** not contain a different number of components, or a different number of bits in each component, than the format of the `VkImage` in `image`
- VUID-VkImageViewCreateInfo-image-04817
  If `image` was created with `usage` containing `VK_IMAGE_USAGE_VIDEO_DECODE_DST_BIT_KHR`, `VK_IMAGE_USAGE_VIDEO_DECODE_SRC_BIT_KHR`, or `VK_IMAGE_USAGE_VIDEO_DECODE_DPB_BIT_KHR`, then the `viewType` **must** be `VK_IMAGE_VIEW_TYPE_2D` or `VK_IMAGE_VIEW_TYPE_2D_ARRAY`
- VUID-VkImageViewCreateInfo-image-04818
  If `image` was created with `usage` containing `VK_IMAGE_USAGE_VIDEO_ENCODE_DST_BIT_KHR`, `VK_IMAGE_USAGE_VIDEO_ENCODE_SRC_BIT_KHR`, or `VK_IMAGE_USAGE_VIDEO_ENCODE_DPB_BIT_KHR`, then the `viewType` **must** be `VK_IMAGE_VIEW_TYPE_2D` or `VK_IMAGE_VIEW_TYPE_2D_ARRAY`
- VUID-VkImageViewCreateInfo-flags-08106
  If `flags` includes `VK_IMAGE_VIEW_CREATE_DESCRIPTOR_BUFFER_CAPTURE_REPLAY_BIT_EXT`, the `descriptorBufferCaptureReplay` feature **must** be enabled
- VUID-VkImageViewCreateInfo-pNext-08107
  If the `pNext` chain includes a VkOpaqueCaptureDescriptorDataCreateInfoEXT structure, `flags` **must** contain `VK_IMAGE_VIEW_CREATE_DESCRIPTOR_BUFFER_CAPTURE_REPLAY_BIT_EXT`
- VUID-VkImageViewCreateInfo-pNext-06787
  If the `pNext` chain includes a VkExportMetalObjectCreateInfoEXT structure, its `exportObjectType` member **must** be `VK_EXPORT_METAL_OBJECT_TYPE_METAL_TEXTURE_BIT_EXT`
- VUID-VkImageViewCreateInfo-pNext-06944
  If the `pNext` chain includes VkImageViewSampleWeightCreateInfoQCOM structure, then `textureSampleWeighted` feature **must** be enabled
- VUID-VkImageViewCreateInfo-pNext-06945
  If the `pNext` chain includes VkImageViewSampleWeightCreateInfoQCOM structure, then `image` **must** have been created with `usage` containing `VK_IMAGE_USAGE_SAMPLE_WEIGHT_BIT_QCOM`
- VUID-VkImageViewCreateInfo-pNext-06946
  If the `pNext` chain includes VkImageViewSampleWeightCreateInfoQCOM structure, then `components` **must** be `VK_COMPONENT_SWIZZLE_IDENTITY` for all components
- VUID-VkImageViewCreateInfo-pNext-06947
  If the `pNext` chain includes VkImageViewSampleWeightCreateInfoQCOM structure, then `subresourceRange.aspectMask` **must** be `VK_IMAGE_ASPECT_COLOR_BIT`
- VUID-VkImageViewCreateInfo-pNext-06948

If the pNext chain includes VkImageViewSampleWeightCreateInfoQCOM structure, then subresourceRange.levelCount **must** be 1

- VUID-VkImageViewCreateInfo-pNext-06949
  If the pNext chain includes VkImageViewSampleWeightCreateInfoQCOM structure, then viewType **must** be VK_IMAGE_VIEW_TYPE_1D_ARRAY or VK_IMAGE_VIEW_TYPE_2D_ARRAY
- VUID-VkImageViewCreateInfo-pNext-06950
  If the pNext chain includes VkImageViewSampleWeightCreateInfoQCOM structure and if viewType is VK_IMAGE_VIEW_TYPE_1D_ARRAY, then image **must** have been created with imageType VK_IMAGE_TYPE_1D
- VUID-VkImageViewCreateInfo-pNext-06951
  If the pNext chain includes VkImageViewSampleWeightCreateInfoQCOM structure and viewType is VK_IMAGE_VIEW_TYPE_1D_ARRAY, then subresourceRange.layerCount **must** be equal to 2
- VUID-VkImageViewCreateInfo-pNext-06952
  If the pNext chain includes VkImageViewSampleWeightCreateInfoQCOM structure and viewType is VK_IMAGE_VIEW_TYPE_1D_ARRAY, then image **must** have been created with width equal to or greater than $(numPhases \times \max(\text{align}(filterSize.width, 4), filterSize.height))$
- VUID-VkImageViewCreateInfo-pNext-06953
  If the pNext chain includes VkImageViewSampleWeightCreateInfoQCOM structure and if viewType is VK_IMAGE_VIEW_TYPE_2D_ARRAY, then image **must** have been created with imageType VK_IMAGE_TYPE_2D
- VUID-VkImageViewCreateInfo-pNext-06954
  If the pNext chain includes VkImageViewSampleWeightCreateInfoQCOM structure and viewType is VK_IMAGE_VIEW_TYPE_2D_ARRAY, then subresourceRange.layerCount **must** be equal or greater than numPhases
- VUID-VkImageViewCreateInfo-pNext-06955
  If the pNext chain includes VkImageViewSampleWeightCreateInfoQCOM structure and viewType is VK_IMAGE_VIEW_TYPE_2D_ARRAY, then image **must** have been created with width equal to or greater than filterSize.width
- VUID-VkImageViewCreateInfo-pNext-06956
  If the pNext chain includes VkImageViewSampleWeightCreateInfoQCOM structure and viewType is VK_IMAGE_VIEW_TYPE_2D_ARRAY, then image **must** have been created with height equal to or greater than filterSize.height
- VUID-VkImageViewCreateInfo-pNext-06957
  If the pNext chain includes VkImageViewSampleWeightCreateInfoQCOM structure then VkImageViewSampleWeightCreateInfoQCOM::filterSize.height **must** be less than or equal to VkPhysicalDeviceImageProcessingPropertiesQCOM::maxWeightFilterDimension.height
- VUID-VkImageViewCreateInfo-subresourceRange-09594
  subresourceRange.aspectMask **must** be valid for the format the image was created with

## Valid Usage (Implicit)

- VUID-VkImageViewCreateInfo-sType-sType

sType **must** be VK_STRUCTURE_TYPE_IMAGE_VIEW_CREATE_INFO

- VUID-VkImageViewCreateInfo-pNext-pNext
  Each pNext member of any structure (including this one) in the pNext chain **must** be either NULL or a pointer to a valid instance of VkExportMetalObjectCreateInfoEXT, VkImageViewASTCDecodeModeEXT, VkImageViewMinLodCreateInfoEXT, VkImageViewSampleWeightCreateInfoQCOM, VkImageViewSlicedCreateInfoEXT, VkImageViewUsageCreateInfo, VkOpaqueCaptureDescriptorDataCreateInfoEXT, or VkSamplerYcbcrConversionInfo
- VUID-VkImageViewCreateInfo-sType-unique
  The sType value of each struct in the pNext chain **must** be unique, with the exception of structures of type VkExportMetalObjectCreateInfoEXT
- VUID-VkImageViewCreateInfo-flags-parameter
  flags **must** be a valid combination of VkImageViewCreateFlagBits values
- VUID-VkImageViewCreateInfo-image-parameter
  image **must** be a valid VkImage handle
- VUID-VkImageViewCreateInfo-viewType-parameter
  viewType **must** be a valid VkImageViewType value
- VUID-VkImageViewCreateInfo-format-parameter
  format **must** be a valid VkFormat value
- VUID-VkImageViewCreateInfo-components-parameter
  components **must** be a valid VkComponentMapping structure
- VUID-VkImageViewCreateInfo-subresourceRange-parameter
  subresourceRange **must** be a valid VkImageSubresourceRange structure

Bits which **can** be set in VkImageViewCreateInfo::flags, specifying additional parameters of an image view, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkImageViewCreateFlagBits {
  // Provided by VK_EXT_fragment_density_map
    VK_IMAGE_VIEW_CREATE_FRAGMENT_DENSITY_MAP_DYNAMIC_BIT_EXT = 0x00000001,
  // Provided by VK_EXT_descriptor_buffer
    VK_IMAGE_VIEW_CREATE_DESCRIPTOR_BUFFER_CAPTURE_REPLAY_BIT_EXT = 0x00000004,
  // Provided by VK_EXT_fragment_density_map2
    VK_IMAGE_VIEW_CREATE_FRAGMENT_DENSITY_MAP_DEFERRED_BIT_EXT = 0x00000002,
} VkImageViewCreateFlagBits;
```

- VK_IMAGE_VIEW_CREATE_FRAGMENT_DENSITY_MAP_DYNAMIC_BIT_EXT specifies that the fragment density map will be read by device during VK_PIPELINE_STAGE_FRAGMENT_DENSITY_PROCESS_BIT_EXT
- VK_IMAGE_VIEW_CREATE_FRAGMENT_DENSITY_MAP_DEFERRED_BIT_EXT specifies that the fragment density map will be read by the host during vkEndCommandBuffer for the primary command buffer that the render pass is recorded into
- VK_IMAGE_VIEW_CREATE_DESCRIPTOR_BUFFER_CAPTURE_REPLAY_BIT_EXT specifies that the image view

**can** be used with descriptor buffers when capturing and replaying (e.g. for trace capture and replay), see VkOpaqueCaptureDescriptorDataCreateInfoEXT for more detail.

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkImageViewCreateFlags;
```

VkImageViewCreateFlags is a bitmask type for setting a mask of zero or more VkImageViewCreateFlagBits.

The set of usages for the created image view **can** be restricted compared to the parent image's usage flags by adding a VkImageViewUsageCreateInfo structure to the pNext chain of VkImageViewCreateInfo.

The VkImageViewUsageCreateInfo structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkImageViewUsageCreateInfo {
    VkStructureType      sType;
    const void*          pNext;
    VkImageUsageFlags    usage;
} VkImageViewUsageCreateInfo;
```

or the equivalent

```
// Provided by VK_KHR_maintenance2
typedef VkImageViewUsageCreateInfo VkImageViewUsageCreateInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- usage is a bitmask of VkImageUsageFlagBits specifying allowed usages of the image view.

When this structure is chained to VkImageViewCreateInfo the usage field overrides the implicit usage parameter inherited from image creation time and its value is used instead for the purposes of determining the valid usage conditions of VkImageViewCreateInfo.

### Valid Usage (Implicit)

- VUID-VkImageViewUsageCreateInfo-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_IMAGE_VIEW_USAGE_CREATE_INFO
- VUID-VkImageViewUsageCreateInfo-usage-parameter
  usage **must** be a valid combination of VkImageUsageFlagBits values
- VUID-VkImageViewUsageCreateInfo-usage-requiredbitmask
  usage **must** not be 0

The range of 3D slices for the created image view **can** be restricted to a subset of the parent image's Z range by adding a VkImageViewSlicedCreateInfoEXT structure to the pNext chain of VkImageViewCreateInfo.

The VkImageViewSlicedCreateInfoEXT structure is defined as:

```
// Provided by VK_EXT_image_sliced_view_of_3d
typedef struct VkImageViewSlicedCreateInfoEXT {
    VkStructureType    sType;
    const void*        pNext;
    uint32_t           sliceOffset;
    uint32_t           sliceCount;
} VkImageViewSlicedCreateInfoEXT;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- sliceOffset is the Z-offset for the first 3D slice accessible to the image view.
- sliceCount is the number of 3D slices accessible to the image view.

When this structure is chained to VkImageViewCreateInfo the sliceOffset field is treated as a Z-offset for the sliced view and sliceCount specifies the range. Shader accesses using a Z coordinate of 0 will access the depth slice corresponding to sliceOffset in the image, and in a shader, the maximum in-bounds Z coordinate for the view is sliceCount - 1.

A sliced 3D view **must** only be used with a single mip level. The slice coordinates are integer coordinates within the subresourceRange.baseMipLevel used to create the image view.

The effective view depth is equal to extent.depth used to create the image for this view adjusted by subresourceRange.baseMipLevel as specified in Image Mip Level Sizing.

Shader access to this image view is only affected by VkImageViewSlicedCreateInfoEXT if it uses a descriptor of type VK_DESCRIPTOR_TYPE_STORAGE_IMAGE. For access using any other descriptor type, the contents of VkImageViewSlicedCreateInfoEXT are ignored; instead, sliceOffset is treated as being equal to 0, and sliceCount is treated as being equal to VK_REMAINING_3D_SLICES_EXT.

### Valid Usage

- VUID-VkImageViewSlicedCreateInfoEXT-sliceOffset-07867
  sliceOffset **must** be less than the effective view depth as specified in Image Mip Level Sizing
- VUID-VkImageViewSlicedCreateInfoEXT-sliceCount-07868
  If sliceCount is not VK_REMAINING_3D_SLICES_EXT, it **must** be non-zero and sliceOffset + sliceCount **must** be less than or equal to the effective view depth as specified in Image Mip Level Sizing
- VUID-VkImageViewSlicedCreateInfoEXT-image-07869
  image **must** have been created with imageType equal to VK_IMAGE_TYPE_3D

- VUID-VkImageViewSlicedCreateInfoEXT-viewType-07909
  `viewType` **must** be `VK_IMAGE_VIEW_TYPE_3D`
- VUID-VkImageViewSlicedCreateInfoEXT-None-07870
  The image view **must** reference exactly 1 mip level
- VUID-VkImageViewSlicedCreateInfoEXT-None-07871
  The imageSlicedViewOf3D feature **must** be enabled on the device

### Valid Usage (Implicit)

- VUID-VkImageViewSlicedCreateInfoEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_IMAGE_VIEW_SLICED_CREATE_INFO_EXT`

`VK_REMAINING_3D_SLICES_EXT` is a special constant value used for VkImageViewSlicedCreateInfoEXT ::`sliceCount` to indicate that all remaining 3D slices in an image after the first slice offset specified should be included in the view.

```
#define VK_REMAINING_3D_SLICES_EXT        (~0U)
```

The `VkImageSubresourceRange` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkImageSubresourceRange {
    VkImageAspectFlags    aspectMask;
    uint32_t              baseMipLevel;
    uint32_t              levelCount;
    uint32_t              baseArrayLayer;
    uint32_t              layerCount;
} VkImageSubresourceRange;
```

- `aspectMask` is a bitmask of VkImageAspectFlagBits specifying which aspect(s) of the image are included in the view.
- `baseMipLevel` is the first mipmap level accessible to the view.
- `levelCount` is the number of mipmap levels (starting from `baseMipLevel`) accessible to the view.
- `baseArrayLayer` is the first array layer accessible to the view.
- `layerCount` is the number of array layers (starting from `baseArrayLayer`) accessible to the view.

The number of mipmap levels and array layers **must** be a subset of the image subresources in the image. If an application wants to use all mip levels or layers in an image after the `baseMipLevel` or `baseArrayLayer`, it **can** set `levelCount` and `layerCount` to the special values `VK_REMAINING_MIP_LEVELS` and `VK_REMAINING_ARRAY_LAYERS` without knowing the exact number of mip levels or layers.

For cube and cube array image views, the layers of the image view starting at `baseArrayLayer` correspond to faces in the order +X, -X, +Y, -Y, +Z, -Z. For cube arrays, each set of six sequential

layers is a single cube, so the number of cube maps in a cube map array view is `layerCount` / *6*, and image array layer (`baseArrayLayer` + i) is face index (i mod 6) of cube *i* / *6*. If the number of layers in the view, whether set explicitly in `layerCount` or implied by `VK_REMAINING_ARRAY_LAYERS`, is not a multiple of 6, the last cube map in the array **must** not be accessed.

`aspectMask` **must** be only `VK_IMAGE_ASPECT_COLOR_BIT`, `VK_IMAGE_ASPECT_DEPTH_BIT` or `VK_IMAGE_ASPECT_STENCIL_BIT` if `format` is a color, depth-only or stencil-only format, respectively, except if `format` is a multi-planar format. If using a depth/stencil format with both depth and stencil components, `aspectMask` **must** include at least one of `VK_IMAGE_ASPECT_DEPTH_BIT` and `VK_IMAGE_ASPECT_STENCIL_BIT`, and **can** include both.

When the `VkImageSubresourceRange` structure is used to select a subset of the slices of a 3D image's mip level in order to create a 2D or 2D array image view of a 3D image created with `VK_IMAGE_CREATE_2D_ARRAY_COMPATIBLE_BIT`, `baseArrayLayer` and `layerCount` specify the first slice index and the number of slices to include in the created image view. Such an image view **can** be used as a framebuffer attachment that refers only to the specified range of slices of the selected mip level. However, any layout transitions performed on such an attachment view during a render pass instance still apply to the entire subresource referenced which includes all the slices of the selected mip level.

When using an image view of a depth/stencil image to populate a descriptor set (e.g. for sampling in the shader, or for use as an input attachment), the `aspectMask` **must** only include one bit, which selects whether the image view is used for depth reads (i.e. using a floating-point sampler or input attachment in the shader) or stencil reads (i.e. using an unsigned integer sampler or input attachment in the shader). When an image view of a depth/stencil image is used as a depth/stencil framebuffer attachment, the `aspectMask` is ignored and both depth and stencil image subresources are used.

When creating a `VkImageView`, if sampler $Y'C_BC_R$ conversion is enabled in the sampler, the `aspectMask` of a `subresourceRange` used by the `VkImageView` **must** be `VK_IMAGE_ASPECT_COLOR_BIT`.

When creating a `VkImageView`, if sampler $Y'C_BC_R$ conversion is not enabled in the sampler and the image `format` is multi-planar, the image **must** have been created with `VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT`, and the `aspectMask` of the `VkImageView`'s `subresourceRange` **must** be `VK_IMAGE_ASPECT_PLANE_0_BIT`, `VK_IMAGE_ASPECT_PLANE_1_BIT` or `VK_IMAGE_ASPECT_PLANE_2_BIT`.

### Valid Usage

- VUID-VkImageSubresourceRange-levelCount-01720
  If `levelCount` is not `VK_REMAINING_MIP_LEVELS`, it **must** be greater than `0`
- VUID-VkImageSubresourceRange-layerCount-01721
  If `layerCount` is not `VK_REMAINING_ARRAY_LAYERS`, it **must** be greater than `0`
- VUID-VkImageSubresourceRange-aspectMask-01670
  If `aspectMask` includes `VK_IMAGE_ASPECT_COLOR_BIT`, then it **must** not include any of `VK_IMAGE_ASPECT_PLANE_0_BIT`, `VK_IMAGE_ASPECT_PLANE_1_BIT`, or `VK_IMAGE_ASPECT_PLANE_2_BIT`
- VUID-VkImageSubresourceRange-aspectMask-02278
  `aspectMask` **must** not include `VK_IMAGE_ASPECT_MEMORY_PLANE_i_BIT_EXT` for any index *i*

### Valid Usage (Implicit)

- VUID-VkImageSubresourceRange-aspectMask-parameter
  `aspectMask` **must** be a valid combination of VkImageAspectFlagBits values
- VUID-VkImageSubresourceRange-aspectMask-requiredbitmask
  `aspectMask` **must** not be `0`

Bits which **can** be set in an aspect mask to specify aspects of an image for purposes such as identifying a subresource, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkImageAspectFlagBits {
    VK_IMAGE_ASPECT_COLOR_BIT = 0x00000001,
    VK_IMAGE_ASPECT_DEPTH_BIT = 0x00000002,
    VK_IMAGE_ASPECT_STENCIL_BIT = 0x00000004,
    VK_IMAGE_ASPECT_METADATA_BIT = 0x00000008,
  // Provided by VK_VERSION_1_1
    VK_IMAGE_ASPECT_PLANE_0_BIT = 0x00000010,
  // Provided by VK_VERSION_1_1
    VK_IMAGE_ASPECT_PLANE_1_BIT = 0x00000020,
  // Provided by VK_VERSION_1_1
    VK_IMAGE_ASPECT_PLANE_2_BIT = 0x00000040,
    VK_IMAGE_ASPECT_NONE = 0,
  // Provided by VK_EXT_image_drm_format_modifier
    VK_IMAGE_ASPECT_MEMORY_PLANE_0_BIT_EXT = 0x00000080,
  // Provided by VK_EXT_image_drm_format_modifier
    VK_IMAGE_ASPECT_MEMORY_PLANE_1_BIT_EXT = 0x00000100,
  // Provided by VK_EXT_image_drm_format_modifier
    VK_IMAGE_ASPECT_MEMORY_PLANE_2_BIT_EXT = 0x00000200,
  // Provided by VK_EXT_image_drm_format_modifier
    VK_IMAGE_ASPECT_MEMORY_PLANE_3_BIT_EXT = 0x00000400,
  // Provided by VK_KHR_sampler_ycbcr_conversion
    VK_IMAGE_ASPECT_PLANE_0_BIT_KHR = VK_IMAGE_ASPECT_PLANE_0_BIT,
  // Provided by VK_KHR_sampler_ycbcr_conversion
    VK_IMAGE_ASPECT_PLANE_1_BIT_KHR = VK_IMAGE_ASPECT_PLANE_1_BIT,
  // Provided by VK_KHR_sampler_ycbcr_conversion
    VK_IMAGE_ASPECT_PLANE_2_BIT_KHR = VK_IMAGE_ASPECT_PLANE_2_BIT,
  // Provided by VK_KHR_maintenance4
    VK_IMAGE_ASPECT_NONE_KHR = VK_IMAGE_ASPECT_NONE,
} VkImageAspectFlagBits;
```

- `VK_IMAGE_ASPECT_NONE` specifies no image aspect, or the image aspect is not applicable.
- `VK_IMAGE_ASPECT_COLOR_BIT` specifies the color aspect.
- `VK_IMAGE_ASPECT_DEPTH_BIT` specifies the depth aspect.
- `VK_IMAGE_ASPECT_STENCIL_BIT` specifies the stencil aspect.
- `VK_IMAGE_ASPECT_METADATA_BIT` specifies the metadata aspect used for sparse resource operations.

- VK_IMAGE_ASPECT_PLANE_0_BIT specifies plane 0 of a *multi-planar* image format.
- VK_IMAGE_ASPECT_PLANE_1_BIT specifies plane 1 of a *multi-planar* image format.
- VK_IMAGE_ASPECT_PLANE_2_BIT specifies plane 2 of a *multi-planar* image format.
- VK_IMAGE_ASPECT_MEMORY_PLANE_0_BIT_EXT specifies *memory plane* 0.
- VK_IMAGE_ASPECT_MEMORY_PLANE_1_BIT_EXT specifies *memory plane* 1.
- VK_IMAGE_ASPECT_MEMORY_PLANE_2_BIT_EXT specifies *memory plane* 2.
- VK_IMAGE_ASPECT_MEMORY_PLANE_3_BIT_EXT specifies *memory plane* 3.

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkImageAspectFlags;
```

VkImageAspectFlags is a bitmask type for setting a mask of zero or more VkImageAspectFlagBits.

The VkComponentMapping structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkComponentMapping {
    VkComponentSwizzle    r;
    VkComponentSwizzle    g;
    VkComponentSwizzle    b;
    VkComponentSwizzle    a;
} VkComponentMapping;
```

- r is a VkComponentSwizzle specifying the component value placed in the R component of the output vector.
- g is a VkComponentSwizzle specifying the component value placed in the G component of the output vector.
- b is a VkComponentSwizzle specifying the component value placed in the B component of the output vector.
- a is a VkComponentSwizzle specifying the component value placed in the A component of the output vector.

### Valid Usage (Implicit)

- VUID-VkComponentMapping-r-parameter
  r **must** be a valid VkComponentSwizzle value
- VUID-VkComponentMapping-g-parameter
  g **must** be a valid VkComponentSwizzle value
- VUID-VkComponentMapping-b-parameter
  b **must** be a valid VkComponentSwizzle value
- VUID-VkComponentMapping-a-parameter
  a **must** be a valid VkComponentSwizzle value

Possible values of the members of VkComponentMapping, specifying the component values placed in each component of the output vector, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkComponentSwizzle {
    VK_COMPONENT_SWIZZLE_IDENTITY = 0,
    VK_COMPONENT_SWIZZLE_ZERO = 1,
    VK_COMPONENT_SWIZZLE_ONE = 2,
    VK_COMPONENT_SWIZZLE_R = 3,
    VK_COMPONENT_SWIZZLE_G = 4,
    VK_COMPONENT_SWIZZLE_B = 5,
    VK_COMPONENT_SWIZZLE_A = 6,
} VkComponentSwizzle;
```

- VK_COMPONENT_SWIZZLE_IDENTITY specifies that the component is set to the identity swizzle.
- VK_COMPONENT_SWIZZLE_ZERO specifies that the component is set to zero.
- VK_COMPONENT_SWIZZLE_ONE specifies that the component is set to either 1 or 1.0, depending on whether the type of the image view format is integer or floating-point respectively, as determined by the Format Definition section for each VkFormat.
- VK_COMPONENT_SWIZZLE_R specifies that the component is set to the value of the R component of the image.
- VK_COMPONENT_SWIZZLE_G specifies that the component is set to the value of the G component of the image.
- VK_COMPONENT_SWIZZLE_B specifies that the component is set to the value of the B component of the image.
- VK_COMPONENT_SWIZZLE_A specifies that the component is set to the value of the A component of the image.

Setting the identity swizzle on a component is equivalent to setting the identity mapping on that component. That is:

*Table 17. Component Mappings Equivalent To* VK_COMPONENT_SWIZZLE_IDENTITY

| Component | Identity Mapping |
|---|---|
| components.r | VK_COMPONENT_SWIZZLE_R |
| components.g | VK_COMPONENT_SWIZZLE_G |
| components.b | VK_COMPONENT_SWIZZLE_B |
| components.a | VK_COMPONENT_SWIZZLE_A |

If the pNext chain includes a VkImageViewASTCDecodeModeEXT structure, then that structure includes a parameter specifying the decode mode for image views using ASTC compressed formats.

The VkImageViewASTCDecodeModeEXT structure is defined as:

```
// Provided by VK_EXT_astc_decode_mode
typedef struct VkImageViewASTCDecodeModeEXT {
    VkStructureType    sType;
    const void*        pNext;
    VkFormat           decodeMode;
} VkImageViewASTCDecodeModeEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `decodeMode` is the intermediate format used to decode ASTC compressed formats.

### Valid Usage

- VUID-VkImageViewASTCDecodeModeEXT-decodeMode-02230
  `decodeMode` **must** be one of `VK_FORMAT_R16G16B16A16_SFLOAT`, `VK_FORMAT_R8G8B8A8_UNORM`, or `VK_FORMAT_E5B9G9R9_UFLOAT_PACK32`
- VUID-VkImageViewASTCDecodeModeEXT-decodeMode-02231
  If the `decodeModeSharedExponent` feature is not enabled, `decodeMode` **must** not be `VK_FORMAT_E5B9G9R9_UFLOAT_PACK32`
- VUID-VkImageViewASTCDecodeModeEXT-decodeMode-02232
  If `decodeMode` is `VK_FORMAT_R8G8B8A8_UNORM` the image view **must** not include blocks using any of the ASTC HDR modes
- VUID-VkImageViewASTCDecodeModeEXT-format-04084
  `format` of the image view **must** be one of the ASTC Compressed Image Formats

If `format` uses sRGB encoding then the `decodeMode` has no effect.

### Valid Usage (Implicit)

- VUID-VkImageViewASTCDecodeModeEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_IMAGE_VIEW_ASTC_DECODE_MODE_EXT`
- VUID-VkImageViewASTCDecodeModeEXT-decodeMode-parameter
  `decodeMode` **must** be a valid VkFormat value

If the `pNext` chain includes a `VkImageViewSampleWeightCreateInfoQCOM` structure, then that structure includes a parameter specifying the parameters for weight image views used in weight image sampling.

The `VkImageViewSampleWeightCreateInfoQCOM` structure is defined as:

```
// Provided by VK_QCOM_image_processing
typedef struct VkImageViewSampleWeightCreateInfoQCOM {
    VkStructureType    sType;
    const void*        pNext;
    VkOffset2D         filterCenter;
    VkExtent2D         filterSize;
    uint32_t           numPhases;
} VkImageViewSampleWeightCreateInfoQCOM;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `filterCenter` is a VkOffset2D describing the location of the weight filter origin.
- `filterSize` is a VkExtent2D specifying weight filter dimensions.
- `numPhases` is number of sub-pixel filter phases.

The `filterCenter` specifies the origin or center of the filter kernel, as described in Weight Sampling Operation. The `numPhases` describes the number of sub-pixel filter phases as described in Weight Sampling Phases.

## Valid Usage

- VUID-VkImageViewSampleWeightCreateInfoQCOM-filterSize-06958
  `filterSize.width` **must** be less than or equal to `VkPhysicalDeviceImageProcessingPropertiesQCOM::maxWeightFilterDimension.width`
- VUID-VkImageViewSampleWeightCreateInfoQCOM-filterSize-06959
  `filterSize.height` **must** be less than or equal to `VkPhysicalDeviceImageProcessingPropertiesQCOM::maxWeightFilterDimension.height`
- VUID-VkImageViewSampleWeightCreateInfoQCOM-filterCenter-06960
  `filterCenter.x` **must** be less than or equal to (filterSize.width - 1)
- VUID-VkImageViewSampleWeightCreateInfoQCOM-filterCenter-06961
  `filterCenter.y` **must** be less than or equal to (filterSize.height - 1)
- VUID-VkImageViewSampleWeightCreateInfoQCOM-numPhases-06962
  `numPhases` **must** be a power of two squared value (i.e., 1, 4, 16, 64, 256, etc.)
- VUID-VkImageViewSampleWeightCreateInfoQCOM-numPhases-06963
  `numPhases` **must** be less than or equal to `VkPhysicalDeviceImageProcessingPropertiesQCOM::maxWeightFilterPhases`

## Valid Usage (Implicit)

- VUID-VkImageViewSampleWeightCreateInfoQCOM-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_IMAGE_VIEW_SAMPLE_WEIGHT_CREATE_INFO_QCOM`

To destroy an image view, call:

```
// Provided by VK_VERSION_1_0
void vkDestroyImageView(
    VkDevice                                    device,
    VkImageView                                 imageView,
    const VkAllocationCallbacks*                pAllocator);
```

- `device` is the logical device that destroys the image view.
- `imageView` is the image view to destroy.
- `pAllocator` controls host memory allocation as described in the Memory Allocation chapter.

## Valid Usage

- VUID-vkDestroyImageView-imageView-01026
  All submitted commands that refer to `imageView` **must** have completed execution
- VUID-vkDestroyImageView-imageView-01027
  If `VkAllocationCallbacks` were provided when `imageView` was created, a compatible set of callbacks **must** be provided here
- VUID-vkDestroyImageView-imageView-01028
  If no `VkAllocationCallbacks` were provided when `imageView` was created, `pAllocator` **must** be `NULL`

## Valid Usage (Implicit)

- VUID-vkDestroyImageView-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkDestroyImageView-imageView-parameter
  If `imageView` is not VK_NULL_HANDLE, `imageView` **must** be a valid VkImageView handle
- VUID-vkDestroyImageView-pAllocator-parameter
  If `pAllocator` is not `NULL`, `pAllocator` **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkDestroyImageView-imageView-parent
  If `imageView` is a valid handle, it **must** have been created, allocated, or retrieved from `device`

## Host Synchronization

- Host access to `imageView` **must** be externally synchronized

To get the handle for an image view, call:

```
// Provided by VK_NVX_image_view_handle
uint32_t vkGetImageViewHandleNVX(
    VkDevice                                    device,
    const VkImageViewHandleInfoNVX*             pInfo);
```

- device is the logical device that owns the image view.
- pInfo describes the image view to query and type of handle.

### Valid Usage (Implicit)

- VUID-vkGetImageViewHandleNVX-device-parameter
  device **must** be a valid VkDevice handle
- VUID-vkGetImageViewHandleNVX-pInfo-parameter
  pInfo **must** be a valid pointer to a valid VkImageViewHandleInfoNVX structure

The VkImageViewHandleInfoNVX structure is defined as:

```
// Provided by VK_NVX_image_view_handle
typedef struct VkImageViewHandleInfoNVX {
    VkStructureType     sType;
    const void*         pNext;
    VkImageView         imageView;
    VkDescriptorType    descriptorType;
    VkSampler           sampler;
} VkImageViewHandleInfoNVX;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- imageView is the image view to query.
- descriptorType is the type of descriptor for which to query a handle.
- sampler is the sampler to combine with the image view when generating the handle.

### Valid Usage

- VUID-VkImageViewHandleInfoNVX-descriptorType-02654
  descriptorType **must** be VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE, VK_DESCRIPTOR_TYPE_STORAGE_IMAGE, or VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER
- VUID-VkImageViewHandleInfoNVX-sampler-02655
  sampler **must** be a valid VkSampler if descriptorType is VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER
- VUID-VkImageViewHandleInfoNVX-imageView-02656
  If descriptorType is VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE or

VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER, the image that imageView was created from **must** have been created with the VK_IMAGE_USAGE_SAMPLED_BIT usage bit set

- VUID-VkImageViewHandleInfoNVX-imageView-02657
  If descriptorType is VK_DESCRIPTOR_TYPE_STORAGE_IMAGE, the image that imageView was created from **must** have been created with the VK_IMAGE_USAGE_STORAGE_BIT usage bit set

### Valid Usage (Implicit)

- VUID-VkImageViewHandleInfoNVX-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_IMAGE_VIEW_HANDLE_INFO_NVX
- VUID-VkImageViewHandleInfoNVX-pNext-pNext
  pNext **must** be NULL
- VUID-VkImageViewHandleInfoNVX-imageView-parameter
  imageView **must** be a valid VkImageView handle
- VUID-VkImageViewHandleInfoNVX-descriptorType-parameter
  descriptorType **must** be a valid VkDescriptorType value
- VUID-VkImageViewHandleInfoNVX-sampler-parameter
  If sampler is not VK_NULL_HANDLE, sampler **must** be a valid VkSampler handle
- VUID-VkImageViewHandleInfoNVX-commonparent
  Both of imageView, and sampler that are valid handles of non-ignored parameters **must** have been created, allocated, or retrieved from the same VkDevice

To get the device address for an image view, call:

```
// Provided by VK_NVX_image_view_handle
VkResult vkGetImageViewAddressNVX(
    VkDevice                                    device,
    VkImageView                                 imageView,
    VkImageViewAddressPropertiesNVX*            pProperties);
```

- device is the logical device that owns the image view.
- imageView is a handle to the image view.
- pProperties contains the device address and size when the call returns.

### Valid Usage (Implicit)

- VUID-vkGetImageViewAddressNVX-device-parameter
  device **must** be a valid VkDevice handle
- VUID-vkGetImageViewAddressNVX-imageView-parameter
  imageView **must** be a valid VkImageView handle
- VUID-vkGetImageViewAddressNVX-pProperties-parameter

pProperties **must** be a valid pointer to a VkImageViewAddressPropertiesNVX structure

- VUID-vkGetImageViewAddressNVX-imageView-parent
  imageView **must** have been created, allocated, or retrieved from device

### Return Codes

**Success**

- VK_SUCCESS

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY

The VkImageViewAddressPropertiesNVX structure is defined as:

```
// Provided by VK_NVX_image_view_handle
typedef struct VkImageViewAddressPropertiesNVX {
    VkStructureType    sType;
    void*              pNext;
    VkDeviceAddress    deviceAddress;
    VkDeviceSize       size;
} VkImageViewAddressPropertiesNVX;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- deviceAddress is the device address of the image view.
- size is the size in bytes of the image view device memory.

### Valid Usage (Implicit)

- VUID-VkImageViewAddressPropertiesNVX-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_IMAGE_VIEW_ADDRESS_PROPERTIES_NVX
- VUID-VkImageViewAddressPropertiesNVX-pNext-pNext
  pNext **must** be NULL

## 12.5.1. Image View Format Features

Valid uses of a VkImageView **may** depend on the image view's *format features*, defined below. Such constraints are documented in the affected valid usage statement.

- If Vulkan 1.3 is supported or the VK_KHR_format_feature_flags2 extension is supported, and VkImageViewCreateInfo::image was created with VK_IMAGE_TILING_LINEAR, then the image view's set of *format features* is the value of VkFormatProperties3::linearTilingFeatures found by calling vkGetPhysicalDeviceFormatProperties2 on the same format as VkImageViewCreateInfo

::format.

- If Vulkan 1.3 is not supported and the VK_KHR_format_feature_flags2 extension is not supported, and VkImageViewCreateInfo::image was created with VK_IMAGE_TILING_LINEAR, then the image view's set of *format features* is the union of the value of VkFormatProperties::linearTilingFeatures found by calling vkGetPhysicalDeviceFormatProperties on the same format as VkImageViewCreateInfo::format, with:
  - VK_FORMAT_FEATURE_2_SAMPLED_IMAGE_DEPTH_COMPARISON_BIT if the format is a depth/stencil format and the image view features also contain VK_FORMAT_FEATURE_2_SAMPLED_IMAGE_BIT.
  - VK_FORMAT_FEATURE_2_STORAGE_READ_WITHOUT_FORMAT_BIT if the format is one of the extended storage formats and shaderStorageImageReadWithoutFormat is enabled on the device.
  - VK_FORMAT_FEATURE_2_STORAGE_WRITE_WITHOUT_FORMAT_BIT if the format is one of the extended storage formats and shaderStorageImageWriteWithoutFormat is enabled on the device.
- If Vulkan 1.3 is supported or the VK_KHR_format_feature_flags2 extension is supported, and VkImageViewCreateInfo::image was created with VK_IMAGE_TILING_OPTIMAL, but without an Android hardware buffer external format, or a QNX Screen buffer external format, then the image view's set of *format features* is the value of VkFormatProperties::optimalTilingFeatures or VkFormatProperties3::optimalTilingFeatures found by calling vkGetPhysicalDeviceFormatProperties or vkGetPhysicalDeviceImageFormatProperties2 on the same format as VkImageViewCreateInfo::format.
- If Vulkan 1.3 is not supported and the VK_KHR_format_feature_flags2 extension is not supported, and VkImageViewCreateInfo::image was created with VK_IMAGE_TILING_OPTIMAL, but without an Android hardware buffer external format, or a QNX Screen buffer external format, then the image view's set of *format features* is the union of the value of VkFormatProperties::optimalTilingFeatures found by calling vkGetPhysicalDeviceFormatProperties on the same format as VkImageViewCreateInfo::format, with:
  - VK_FORMAT_FEATURE_2_SAMPLED_IMAGE_DEPTH_COMPARISON_BIT if the format is a depth/stencil format and the image view features also contain VK_FORMAT_FEATURE_2_SAMPLED_IMAGE_BIT.
  - VK_FORMAT_FEATURE_2_STORAGE_READ_WITHOUT_FORMAT_BIT if the format is one of the extended storage formats and shaderStorageImageReadWithoutFormat is enabled on the device.
  - VK_FORMAT_FEATURE_2_STORAGE_WRITE_WITHOUT_FORMAT_BIT if the format is one of the extended storage formats and shaderStorageImageWriteWithoutFormat is enabled on the device.
- If VkImageViewCreateInfo::image was created with an Android hardware buffer external format, then the image views's set of *format features* is the value of VkAndroidHardwareBufferFormatPropertiesANDROID::formatFeatures found by calling vkGetAndroidHardwareBufferPropertiesANDROID on the Android hardware buffer that was imported to the VkDeviceMemory to which the VkImageViewCreateInfo::image is bound.
- If VkImageViewCreateInfo::image was created with a QNX Screen buffer external format, then the image views's set of *format features* is the value of VkScreenBufferFormatPropertiesQNX::formatFeatures found by calling vkGetScreenBufferPropertiesQNX on the QNX Screen buffer that was imported to the VkDeviceMemory to which the VkImageViewCreateInfo::image is bound.
- If VkImageViewCreateInfo::image was created with a chained VkBufferCollectionImageCreateInfoFUCHSIA, then the image view's set of *format features* is the

value of VkBufferCollectionPropertiesFUCHSIA::formatFeatures found by calling vkGetBufferCollectionPropertiesFUCHSIA on the buffer collection passed as VkBufferCollectionImageCreateInfoFUCHSIA::collection when the image was created.

- If VkImageViewCreateInfo::image was created with VK_IMAGE_TILING_DRM_FORMAT_MODIFIER_EXT, then:
  - The image's DRM format modifier is the value of VkImageDrmFormatModifierPropertiesEXT::drmFormatModifier found by calling vkGetImageDrmFormatModifierPropertiesEXT.
  - Let VkDrmFormatModifierPropertiesListEXT::pDrmFormatModifierProperties be the array found by calling vkGetPhysicalDeviceFormatProperties2 on the same format as VkImageViewCreateInfo::format.
  - Let VkDrmFormatModifierPropertiesEXT prop be the array element whose drmFormatModifier member is the value of the image's DRM format modifier.
  - Then the image view's set of *format features* is prop::drmFormatModifierTilingFeatures.

The VkImageViewMinLodCreateInfoEXT structure is defined as:

```
// Provided by VK_EXT_image_view_min_lod
typedef struct VkImageViewMinLodCreateInfoEXT {
    VkStructureType    sType;
    const void*        pNext;
    float              minLod;
} VkImageViewMinLodCreateInfoEXT;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- minLod is the value to clamp the minimum LOD accessible by this VkImageView.

If the pNext chain includes a VkImageViewMinLodCreateInfoEXT structure, then that structure includes a parameter specifying a value to clamp the minimum LOD value during Image Level(s) Selection, Texel Gathering and Integer Texel Coordinate Operations.

If the image view contains VkImageViewMinLodCreateInfoEXT and it is used as part of a sampling operation:

$\text{minLodFloat}_{\text{imageView}}$ = minLod

otherwise:

$\text{minLodFloat}_{\text{imageView}} = 0.0$

An integer variant of this parameter is also defined for sampling operations which access integer mipmap levels:

$\text{minLodInteger}_{\text{imageView}} = \lfloor \text{minLodFloat}_{\text{imageView}} \rfloor$

### Valid Usage

- VUID-VkImageViewMinLodCreateInfoEXT-minLod-06455
  If the `minLod` feature is not enabled, `minLod` **must** be `0.0`
- VUID-VkImageViewMinLodCreateInfoEXT-minLod-06456
  `minLod` **must** be less or equal to the index of the last mipmap level accessible to the view

### Valid Usage (Implicit)

- VUID-VkImageViewMinLodCreateInfoEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_IMAGE_VIEW_MIN_LOD_CREATE_INFO_EXT`

## 12.6. Acceleration Structures

Acceleration structures are opaque data structures that are built by the implementation to more efficiently perform spatial queries on the provided geometric data. For this extension, an acceleration structure is either a top-level acceleration structure containing a set of bottom-level acceleration structures or a bottom-level acceleration structure containing either a set of axis-aligned bounding boxes for custom geometry or a set of triangles.

Each instance in the top-level acceleration structure contains a reference to a bottom-level acceleration structure as well as an instance transform plus information required to index into the shader bindings. The top-level acceleration structure is what is bound to the acceleration descriptor, for example to trace inside the shader in the ray tracing pipeline.

Acceleration structures are represented by `VkAccelerationStructureKHR` handles:

```
// Provided by VK_KHR_acceleration_structure
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkAccelerationStructureKHR)
```

Acceleration structures for the `VK_NV_ray_tracing extension` are represented by the similar `VkAccelerationStructureNV` handles:

```
// Provided by VK_NV_ray_tracing
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkAccelerationStructureNV)
```

To create acceleration structures, call:

```
// Provided by VK_NV_ray_tracing
VkResult vkCreateAccelerationStructureNV(
    VkDevice                                    device,
    const VkAccelerationStructureCreateInfoNV*  pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkAccelerationStructureNV*                  pAccelerationStructure);
```

- `device` is the logical device that creates the buffer object.
- `pCreateInfo` is a pointer to a VkAccelerationStructureCreateInfoNV structure containing parameters affecting creation of the acceleration structure.
- `pAllocator` controls host memory allocation as described in the Memory Allocation chapter.
- `pAccelerationStructure` is a pointer to a VkAccelerationStructureNV handle in which the resulting acceleration structure object is returned.

Similarly to other objects in Vulkan, the acceleration structure creation merely creates an object with a specific "shape" as specified by the information in VkAccelerationStructureInfoNV and `compactedSize` in `pCreateInfo`.

Once memory has been bound to the acceleration structure using vkBindAccelerationStructureMemoryNV, that memory is populated by calls to vkCmdBuildAccelerationStructureNV and vkCmdCopyAccelerationStructureNV.

Acceleration structure creation uses the count and type information from the geometries, but does not use the data references in the structures.

## Valid Usage (Implicit)

- VUID-vkCreateAccelerationStructureNV-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkCreateAccelerationStructureNV-pCreateInfo-parameter
  `pCreateInfo` **must** be a valid pointer to a valid VkAccelerationStructureCreateInfoNV structure
- VUID-vkCreateAccelerationStructureNV-pAllocator-parameter
  If `pAllocator` is not `NULL`, `pAllocator` **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkCreateAccelerationStructureNV-pAccelerationStructure-parameter
  `pAccelerationStructure` **must** be a valid pointer to a VkAccelerationStructureNV handle

## Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY

The VkAccelerationStructureCreateInfoNV structure is defined as:

```
// Provided by VK_NV_ray_tracing
typedef struct VkAccelerationStructureCreateInfoNV {
    VkStructureType                 sType;
    const void*                     pNext;
    VkDeviceSize                    compactedSize;
    VkAccelerationStructureInfoNV   info;
} VkAccelerationStructureCreateInfoNV;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- compactedSize is the size from the result of vkCmdWriteAccelerationStructuresPropertiesNV if this acceleration structure is going to be the target of a compacting copy.
- info is the VkAccelerationStructureInfoNV structure specifying further parameters of the created acceleration structure.

## Valid Usage

- VUID-VkAccelerationStructureCreateInfoNV-compactedSize-02421
  If compactedSize is not 0 then both info.geometryCount and info.instanceCount **must** be 0

## Valid Usage (Implicit)

- VUID-VkAccelerationStructureCreateInfoNV-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_ACCELERATION_STRUCTURE_CREATE_INFO_NV
- VUID-VkAccelerationStructureCreateInfoNV-pNext-pNext
  pNext **must** be NULL or a pointer to a valid instance of VkOpaqueCaptureDescriptorDataCreateInfoEXT
- VUID-VkAccelerationStructureCreateInfoNV-sType-unique
  The sType value of each struct in the pNext chain **must** be unique
- VUID-VkAccelerationStructureCreateInfoNV-info-parameter
  info **must** be a valid VkAccelerationStructureInfoNV structure

The VkAccelerationStructureInfoNV structure is defined as:

```
// Provided by VK_NV_ray_tracing
typedef struct VkAccelerationStructureInfoNV {
    VkStructureType                         sType;
    const void*                             pNext;
    VkAccelerationStructureTypeNV           type;
    VkBuildAccelerationStructureFlagsNV     flags;
    uint32_t                                instanceCount;
    uint32_t                                geometryCount;
    const VkGeometryNV*                     pGeometries;
} VkAccelerationStructureInfoNV;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `type` is a VkAccelerationStructureTypeNV value specifying the type of acceleration structure that will be created.
- `flags` is a bitmask of VkBuildAccelerationStructureFlagBitsNV specifying additional parameters of the acceleration structure.
- `instanceCount` specifies the number of instances that will be in the new acceleration structure.
- `geometryCount` specifies the number of geometries that will be in the new acceleration structure.
- `pGeometries` is a pointer to an array of `geometryCount` VkGeometryNV structures containing the scene data being passed into the acceleration structure.

`VkAccelerationStructureInfoNV` contains information that is used both for acceleration structure creation with vkCreateAccelerationStructureNV and in combination with the actual geometric data to build the acceleration structure with vkCmdBuildAccelerationStructureNV.

### Valid Usage

- VUID-VkAccelerationStructureInfoNV-geometryCount-02422
  `geometryCount` **must** be less than or equal to VkPhysicalDeviceRayTracingPropertiesNV::`maxGeometryCount`
- VUID-VkAccelerationStructureInfoNV-instanceCount-02423
  `instanceCount` **must** be less than or equal to VkPhysicalDeviceRayTracingPropertiesNV::`maxInstanceCount`
- VUID-VkAccelerationStructureInfoNV-maxTriangleCount-02424
  The total number of triangles in all geometries **must** be less than or equal to VkPhysicalDeviceRayTracingPropertiesNV::`maxTriangleCount`
- VUID-VkAccelerationStructureInfoNV-type-02425
  If `type` is `VK_ACCELERATION_STRUCTURE_TYPE_TOP_LEVEL_NV` then `geometryCount` **must** be `0`
- VUID-VkAccelerationStructureInfoNV-type-02426
  If `type` is `VK_ACCELERATION_STRUCTURE_TYPE_BOTTOM_LEVEL_NV` then `instanceCount` **must** be `0`
- VUID-VkAccelerationStructureInfoNV-type-02786
  If `type` is `VK_ACCELERATION_STRUCTURE_TYPE_BOTTOM_LEVEL_NV` then the `geometryType` member

of each geometry in pGeometries **must** be the same

- VUID-VkAccelerationStructureInfoNV-type-04623
  type **must** not be VK_ACCELERATION_STRUCTURE_TYPE_GENERIC_KHR
- VUID-VkAccelerationStructureInfoNV-flags-02592
  If flags has the VK_BUILD_ACCELERATION_STRUCTURE_PREFER_FAST_TRACE_BIT_NV bit set, then it **must** not have the VK_BUILD_ACCELERATION_STRUCTURE_PREFER_FAST_BUILD_BIT_NV bit set
- VUID-VkAccelerationStructureInfoNV-scratch-02781
  scratch **must** have been created with VK_BUFFER_USAGE_RAY_TRACING_BIT_NV usage flag
- VUID-VkAccelerationStructureInfoNV-instanceData-02782
  If instanceData is not VK_NULL_HANDLE, instanceData **must** have been created with VK_BUFFER_USAGE_RAY_TRACING_BIT_NV usage flag

### Valid Usage (Implicit)

- VUID-VkAccelerationStructureInfoNV-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_ACCELERATION_STRUCTURE_INFO_NV
- VUID-VkAccelerationStructureInfoNV-pNext-pNext
  pNext **must** be NULL
- VUID-VkAccelerationStructureInfoNV-type-parameter
  type **must** be a valid VkAccelerationStructureTypeNV value
- VUID-VkAccelerationStructureInfoNV-flags-parameter
  flags **must** be a valid combination of VkBuildAccelerationStructureFlagBitsNV values
- VUID-VkAccelerationStructureInfoNV-pGeometries-parameter
  If geometryCount is not 0, pGeometries **must** be a valid pointer to an array of geometryCount valid VkGeometryNV structures

To create an acceleration structure, call:

```
// Provided by VK_KHR_acceleration_structure
VkResult vkCreateAccelerationStructureKHR(
    VkDevice                                    device,
    const VkAccelerationStructureCreateInfoKHR* pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkAccelerationStructureKHR*                 pAccelerationStructure);
```

- device is the logical device that creates the acceleration structure object.
- pCreateInfo is a pointer to a VkAccelerationStructureCreateInfoKHR structure containing parameters affecting creation of the acceleration structure.
- pAllocator controls host memory allocation as described in the Memory Allocation chapter.
- pAccelerationStructure is a pointer to a VkAccelerationStructureKHR handle in which the resulting acceleration structure object is returned.

Similar to other objects in Vulkan, the acceleration structure creation merely creates an object with a specific “shape”. The type and quantity of geometry that can be built into an acceleration structure is determined by the parameters of VkAccelerationStructureCreateInfoKHR.

The acceleration structure data is stored in the object referred to by VkAccelerationStructureCreateInfoKHR::buffer. Once memory has been bound to that buffer, it **must** be populated by acceleration structure build or acceleration structure copy commands such as vkCmdBuildAccelerationStructuresKHR, vkBuildAccelerationStructuresKHR, vkCmdCopyAccelerationStructureKHR, and vkCopyAccelerationStructureKHR.

| | |
|---|---|
| **NOTE** | The expected usage for a trace capture/replay tool is that it will serialize and later deserialize the acceleration structure data using acceleration structure copy commands. During capture the tool will use vkCopyAccelerationStructureToMemoryKHR or vkCmdCopyAccelerationStructureToMemoryKHR with a mode of VK_COPY_ACCELERATION_STRUCTURE_MODE_SERIALIZE_KHR, and vkCopyMemoryToAccelerationStructureKHR or vkCmdCopyMemoryToAccelerationStructureKHR with a mode of VK_COPY_ACCELERATION_STRUCTURE_MODE_DESERIALIZE_KHR during replay. |

| | |
|---|---|
| **NOTE** | Memory does not need to be bound to the underlying buffer when vkCreateAccelerationStructureKHR is called. |

The input buffers passed to acceleration structure build commands will be referenced by the implementation for the duration of the command. After the command completes, the acceleration structure **may** hold a reference to any acceleration structure specified by an active instance contained therein. Apart from this referencing, acceleration structures **must** be fully self-contained. The application **can** reuse or free any memory which was used by the command as an input or as scratch without affecting the results of ray traversal.

### Valid Usage

- VUID-vkCreateAccelerationStructureKHR-accelerationStructure-03611
  The VkPhysicalDeviceAccelerationStructureFeaturesKHR::accelerationStructure feature **must** be enabled
- VUID-vkCreateAccelerationStructureKHR-deviceAddress-03488
  If VkAccelerationStructureCreateInfoKHR::deviceAddress is not zero, the accelerationStructureCaptureReplay feature **must** be enabled
- VUID-vkCreateAccelerationStructureKHR-device-03489
  If device was created with multiple physical devices, then the bufferDeviceAddressMultiDevice feature **must** be enabled

### Valid Usage (Implicit)

- VUID-vkCreateAccelerationStructureKHR-device-parameter

device **must** be a valid VkDevice handle

- VUID-vkCreateAccelerationStructureKHR-pCreateInfo-parameter
  pCreateInfo **must** be a valid pointer to a valid VkAccelerationStructureCreateInfoKHR structure
- VUID-vkCreateAccelerationStructureKHR-pAllocator-parameter
  If pAllocator is not NULL, pAllocator **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkCreateAccelerationStructureKHR-pAccelerationStructure-parameter
  pAccelerationStructure **must** be a valid pointer to a VkAccelerationStructureKHR handle

## Return Codes

**Success**

- VK_SUCCESS

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_INVALID_OPAQUE_CAPTURE_ADDRESS_KHR

The VkAccelerationStructureCreateInfoKHR structure is defined as:

```
// Provided by VK_KHR_acceleration_structure
typedef struct VkAccelerationStructureCreateInfoKHR {
    VkStructureType                         sType;
    const void*                             pNext;
    VkAccelerationStructureCreateFlagsKHR   createFlags;
    VkBuffer                                buffer;
    VkDeviceSize                            offset;
    VkDeviceSize                            size;
    VkAccelerationStructureTypeKHR          type;
    VkDeviceAddress                         deviceAddress;
} VkAccelerationStructureCreateInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- createFlags is a bitmask of VkAccelerationStructureCreateFlagBitsKHR specifying additional creation parameters of the acceleration structure.
- buffer is the buffer on which the acceleration structure will be stored.
- offset is an offset in bytes from the base address of the buffer at which the acceleration structure will be stored, and **must** be a multiple of 256.
- size is the size required for the acceleration structure.
- type is a VkAccelerationStructureTypeKHR value specifying the type of acceleration structure

that will be created.

- `deviceAddress` is the device address requested for the acceleration structure if the `accelerationStructureCaptureReplay` feature is being used. If `deviceAddress` is zero, no specific address is requested.

Applications **should** avoid creating acceleration structures with application-provided addresses and implementation-provided addresses in the same process, to reduce the likelihood of `VK_ERROR_INVALID_OPAQUE_CAPTURE_ADDRESS_KHR` errors.

**NOTE**

The expected usage for this is that a trace capture/replay tool will add the `VK_BUFFER_CREATE_DEVICE_ADDRESS_CAPTURE_REPLAY_BIT` flag to all buffers that use `VK_BUFFER_USAGE_SHADER_DEVICE_ADDRESS_BIT`, and will add `VK_BUFFER_USAGE_SHADER_DEVICE_ADDRESS_BIT` to all buffers used as storage for an acceleration structure where `deviceAddress` is not zero. This also means that the tool will need to add `VK_MEMORY_ALLOCATE_DEVICE_ADDRESS_BIT` to memory allocations to allow the flag to be set where the application may not have otherwise required it. During capture the tool will save the queried opaque device addresses in the trace. During replay, the buffers will be created specifying the original address so any address values stored in the trace data will remain valid.

Implementations are expected to separate such buffers in the GPU address space so normal allocations will avoid using these addresses. Apps/tools should avoid mixing app-provided and implementation-provided addresses for buffers created with `VK_BUFFER_CREATE_DEVICE_ADDRESS_CAPTURE_REPLAY_BIT`, to avoid address space allocation conflicts.

Applications **should** create an acceleration structure with a specific VkAccelerationStructureTypeKHR other than `VK_ACCELERATION_STRUCTURE_TYPE_GENERIC_KHR`.

**NOTE**

`VK_ACCELERATION_STRUCTURE_TYPE_GENERIC_KHR` is intended to be used by API translation layers. This can be used at acceleration structure creation time in cases where the actual acceleration structure type (top or bottom) is not yet known. The actual acceleration structure type must be specified as `VK_ACCELERATION_STRUCTURE_TYPE_TOP_LEVEL_KHR` or `VK_ACCELERATION_STRUCTURE_TYPE_BOTTOM_LEVEL_KHR` when the build is performed.

If the acceleration structure will be the target of a build operation, the required size for an acceleration structure **can** be queried with vkGetAccelerationStructureBuildSizesKHR. If the acceleration structure is going to be the target of a compacting copy, vkCmdWriteAccelerationStructuresPropertiesKHR or vkWriteAccelerationStructuresPropertiesKHR **can** be used to obtain the compacted size required.

If the acceleration structure will be the target of a build operation with `VK_BUILD_ACCELERATION_STRUCTURE_MOTION_BIT_NV` it **must** include `VK_ACCELERATION_STRUCTURE_CREATE_MOTION_BIT_NV` in `createFlags` and include VkAccelerationStructureMotionInfoNV as an extension structure in `pNext` with the number of instances as metadata for the object.

## Valid Usage

- VUID-VkAccelerationStructureCreateInfoKHR-deviceAddress-03612
  If `deviceAddress` is not zero, `createFlags` **must** include `VK_ACCELERATION_STRUCTURE_CREATE_DEVICE_ADDRESS_CAPTURE_REPLAY_BIT_KHR`
- VUID-VkAccelerationStructureCreateInfoKHR-deviceAddress-09488
  If `deviceAddress` is not zero, it **must** have been retrieved from an identically created acceleration structure, except for `buffer` and `deviceAddress`
- VUID-VkAccelerationStructureCreateInfoKHR-deviceAddress-09489
  If `deviceAddress` is not zero, `buffer` **must** have been created identically to the `buffer` used to create the acceleration structure from which `deviceAddress` was retrieved, except for VkBufferOpaqueCaptureAddressCreateInfo::`opaqueCaptureAddress`
- VUID-VkAccelerationStructureCreateInfoKHR-deviceAddress-09490
  If `deviceAddress` is not zero, `buffer` **must** have been created with a VkBufferOpaqueCaptureAddressCreateInfo::`opaqueCaptureAddress` that was retrieved from vkGetBufferOpaqueCaptureAddressKHR for the `buffer` that was used to create the acceleration structure from which `deviceAddress` was retrieved
- VUID-VkAccelerationStructureCreateInfoKHR-createFlags-03613
  If `createFlags` includes `VK_ACCELERATION_STRUCTURE_CREATE_DEVICE_ADDRESS_CAPTURE_REPLAY_BIT_KHR`, VkPhysicalDeviceAccelerationStructureFeaturesKHR::`accelerationStructureCaptureReplay` **must** be `VK_TRUE`
- VUID-VkAccelerationStructureCreateInfoKHR-buffer-03614
  `buffer` **must** have been created with a `usage` value containing `VK_BUFFER_USAGE_ACCELERATION_STRUCTURE_STORAGE_BIT_KHR`
- VUID-VkAccelerationStructureCreateInfoKHR-buffer-03615
  `buffer` **must** not have been created with `VK_BUFFER_CREATE_SPARSE_RESIDENCY_BIT`
- VUID-VkAccelerationStructureCreateInfoKHR-offset-03616
  The sum of `offset` and `size` **must** be less than the size of `buffer`
- VUID-VkAccelerationStructureCreateInfoKHR-offset-03734
  `offset` **must** be a multiple of `256` bytes
- VUID-VkAccelerationStructureCreateInfoKHR-createFlags-04954
  If `VK_ACCELERATION_STRUCTURE_CREATE_MOTION_BIT_NV` is set in `createFlags` and `type` is `VK_ACCELERATION_STRUCTURE_TYPE_TOP_LEVEL_KHR`, one member of the `pNext` chain **must** be a pointer to a valid instance of VkAccelerationStructureMotionInfoNV
- VUID-VkAccelerationStructureCreateInfoKHR-createFlags-04955
  If any geometry includes `VkAccelerationStructureGeometryMotionTrianglesDataNV` then `createFlags` **must** contain `VK_ACCELERATION_STRUCTURE_CREATE_MOTION_BIT_NV`
- VUID-VkAccelerationStructureCreateInfoKHR-createFlags-08108
  If `createFlags` includes `VK_ACCELERATION_STRUCTURE_CREATE_DESCRIPTOR_BUFFER_CAPTURE_REPLAY_BIT_EXT`, the `descriptorBufferCaptureReplay` feature **must** be enabled

- VUID-VkAccelerationStructureCreateInfoKHR-pNext-08109
  If the pNext chain includes a VkOpaqueCaptureDescriptorDataCreateInfoEXT structure, createFlags **must** contain VK_ACCELERATION_STRUCTURE_CREATE_DESCRIPTOR_BUFFER_CAPTURE_REPLAY_BIT_EXT

### Valid Usage (Implicit)

- VUID-VkAccelerationStructureCreateInfoKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_ACCELERATION_STRUCTURE_CREATE_INFO_KHR
- VUID-VkAccelerationStructureCreateInfoKHR-pNext-pNext
  Each pNext member of any structure (including this one) in the pNext chain **must** be either NULL or a pointer to a valid instance of VkAccelerationStructureMotionInfoNV or VkOpaqueCaptureDescriptorDataCreateInfoEXT
- VUID-VkAccelerationStructureCreateInfoKHR-sType-unique
  The sType value of each struct in the pNext chain **must** be unique
- VUID-VkAccelerationStructureCreateInfoKHR-createFlags-parameter
  createFlags **must** be a valid combination of VkAccelerationStructureCreateFlagBitsKHR values
- VUID-VkAccelerationStructureCreateInfoKHR-buffer-parameter
  buffer **must** be a valid VkBuffer handle
- VUID-VkAccelerationStructureCreateInfoKHR-type-parameter
  type **must** be a valid VkAccelerationStructureTypeKHR value

The VkAccelerationStructureMotionInfoNV structure is defined as:

```
// Provided by VK_NV_ray_tracing_motion_blur
typedef struct VkAccelerationStructureMotionInfoNV {
    VkStructureType                             sType;
    const void*                                 pNext;
    uint32_t                                    maxInstances;
    VkAccelerationStructureMotionInfoFlagsNV    flags;
} VkAccelerationStructureMotionInfoNV;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- maxInstances is the maximum number of instances that **may** be used in the motion top-level acceleration structure.
- flags is 0 and reserved for future use.

### Valid Usage (Implicit)

- VUID-VkAccelerationStructureMotionInfoNV-sType-sType

sType **must** be VK_STRUCTURE_TYPE_ACCELERATION_STRUCTURE_MOTION_INFO_NV

- VUID-VkAccelerationStructureMotionInfoNV-flags-zerobitmask
  flags **must** be 0

```
// Provided by VK_NV_ray_tracing_motion_blur
typedef VkFlags VkAccelerationStructureMotionInfoFlagsNV;
```

VkAccelerationStructureMotionInfoFlagsNV is a bitmask type for setting a mask, but is currently reserved for future use.

To get the build sizes for an acceleration structure, call:

```
// Provided by VK_KHR_acceleration_structure
void vkGetAccelerationStructureBuildSizesKHR(
    VkDevice                                    device,
    VkAccelerationStructureBuildTypeKHR         buildType,
    const VkAccelerationStructureBuildGeometryInfoKHR* pBuildInfo,
    const uint32_t*                             pMaxPrimitiveCounts,
    VkAccelerationStructureBuildSizesInfoKHR*   pSizeInfo);
```

- device is the logical device that will be used for creating the acceleration structure.
- buildType defines whether host or device operations (or both) are being queried for.
- pBuildInfo is a pointer to a VkAccelerationStructureBuildGeometryInfoKHR structure describing parameters of a build operation.
- pMaxPrimitiveCounts is a pointer to an array of pBuildInfo->geometryCount uint32_t values defining the number of primitives built into each geometry.
- pSizeInfo is a pointer to a VkAccelerationStructureBuildSizesInfoKHR structure which returns the size required for an acceleration structure and the sizes required for the scratch buffers, given the build parameters.

The srcAccelerationStructure, dstAccelerationStructure, and mode members of pBuildInfo are ignored. Any VkDeviceOrHostAddressKHR or VkDeviceOrHostAddressConstKHR members of pBuildInfo are ignored by this command, except that the hostAddress member of VkAccelerationStructureGeometryTrianglesDataKHR::transformData will be examined to check if it is NULL.

An acceleration structure created with the accelerationStructureSize returned by this command supports any build or update with a VkAccelerationStructureBuildGeometryInfoKHR structure and array of VkAccelerationStructureBuildRangeInfoKHR structures subject to the following properties:

- The build command is a host build command, and buildType is VK_ACCELERATION_STRUCTURE_BUILD_TYPE_HOST_KHR or VK_ACCELERATION_STRUCTURE_BUILD_TYPE_HOST_OR_DEVICE_KHR
- The build command is a device build command, and buildType is

VK_ACCELERATION_STRUCTURE_BUILD_TYPE_DEVICE_KHR or VK_ACCELERATION_STRUCTURE_BUILD_TYPE_HOST_OR_DEVICE_KHR

- For VkAccelerationStructureBuildGeometryInfoKHR:
  - Its type, and flags members are equal to pBuildInfo->type and pBuildInfo->flags, respectively.
  - geometryCount is less than or equal to pBuildInfo->geometryCount.
  - For each element of either pGeometries or ppGeometries at a given index, its geometryType member is equal to pBuildInfo->geometryType.
  - For each element of either pGeometries or ppGeometries at a given index, its flags member is equal to the corresponding member of the same element in pBuildInfo.
  - For each element of either pGeometries or ppGeometries at a given index, with a geometryType member equal to VK_GEOMETRY_TYPE_TRIANGLES_KHR, the vertexFormat and indexType members of geometry.triangles are equal to the corresponding members of the same element in pBuildInfo.
  - For each element of either pGeometries or ppGeometries at a given index, with a geometryType member equal to VK_GEOMETRY_TYPE_TRIANGLES_KHR, the maxVertex member of geometry.triangles is less than or equal to the corresponding member of the same element in pBuildInfo.
  - For each element of either pGeometries or ppGeometries at a given index, with a geometryType member equal to VK_GEOMETRY_TYPE_TRIANGLES_KHR, if the applicable address in the transformData member of geometry.triangles is not NULL, the corresponding transformData.hostAddress parameter in pBuildInfo is not NULL.
- For each VkAccelerationStructureBuildRangeInfoKHR corresponding to the VkAccelerationStructureBuildGeometryInfoKHR:
  - Its primitiveCount member is less than or equal to the corresponding element of pMaxPrimitiveCounts.
  - For each element of either pGeometries or ppGeometries at a given index, with a geometryType member equal to VK_GEOMETRY_TYPE_TRIANGLES_KHR, if the pNext chain contains VkAccelerationStructureTrianglesOpacityMicromapEXT the corresponding member of pBuildInfo also contains VkAccelerationStructureTrianglesOpacityMicromapEXT and with an equivalent micromap.
  - For each element of either pGeometries or ppGeometries at a given index, with a geometryType member equal to VK_GEOMETRY_TYPE_TRIANGLES_KHR, if the pNext chain contains VkAccelerationStructureTrianglesDisplacementMicromapNV the corresponding member of pBuildInfo also contains VkAccelerationStructureTrianglesDisplacementMicromapNV and with an equivalent micromap.

Similarly, the updateScratchSize value will support any build command specifying the VK_BUILD_ACCELERATION_STRUCTURE_MODE_UPDATE_KHR mode under the above conditions, and the buildScratchSize value will support any build command specifying the VK_BUILD_ACCELERATION_STRUCTURE_MODE_BUILD_KHR mode under the above conditions.

## Valid Usage

- VUID-vkGetAccelerationStructureBuildSizesKHR-accelerationStructure-08933
  The `VkPhysicalDeviceAccelerationStructureFeaturesKHR::accelerationStructure` feature **must** be enabled
- VUID-vkGetAccelerationStructureBuildSizesKHR-device-03618
  If `device` was created with multiple physical devices, then the `bufferDeviceAddressMultiDevice` feature **must** be enabled
- VUID-vkGetAccelerationStructureBuildSizesKHR-pBuildInfo-03619
  If `pBuildInfo->geometryCount` is not `0`, `pMaxPrimitiveCounts` **must** be a valid pointer to an array of `pBuildInfo->geometryCount` `uint32_t` values
- VUID-vkGetAccelerationStructureBuildSizesKHR-pBuildInfo-03785
  If `pBuildInfo->pGeometries` or `pBuildInfo->ppGeometries` has a `geometryType` of `VK_GEOMETRY_TYPE_INSTANCES_KHR`, each `pMaxPrimitiveCounts`[i] **must** be less than or equal to VkPhysicalDeviceAccelerationStructurePropertiesKHR::`maxInstanceCount`

## Valid Usage (Implicit)

- VUID-vkGetAccelerationStructureBuildSizesKHR-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkGetAccelerationStructureBuildSizesKHR-buildType-parameter
  `buildType` **must** be a valid VkAccelerationStructureBuildTypeKHR value
- VUID-vkGetAccelerationStructureBuildSizesKHR-pBuildInfo-parameter
  `pBuildInfo` **must** be a valid pointer to a valid VkAccelerationStructureBuildGeometryInfoKHR structure
- VUID-vkGetAccelerationStructureBuildSizesKHR-pMaxPrimitiveCounts-parameter
  If `pMaxPrimitiveCounts` is not `NULL`, `pMaxPrimitiveCounts` **must** be a valid pointer to an array of `pBuildInfo->geometryCount` `uint32_t` values
- VUID-vkGetAccelerationStructureBuildSizesKHR-pSizeInfo-parameter
  `pSizeInfo` **must** be a valid pointer to a VkAccelerationStructureBuildSizesInfoKHR structure

The `VkAccelerationStructureBuildSizesInfoKHR` structure describes the required build sizes for an acceleration structure and scratch buffers and is defined as:

```
// Provided by VK_KHR_acceleration_structure
typedef struct VkAccelerationStructureBuildSizesInfoKHR {
    VkStructureType    sType;
    const void*        pNext;
    VkDeviceSize       accelerationStructureSize;
    VkDeviceSize       updateScratchSize;
    VkDeviceSize       buildScratchSize;
} VkAccelerationStructureBuildSizesInfoKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `accelerationStructureSize` is the size in bytes required in a VkAccelerationStructureKHR for a build or update operation.
- `updateScratchSize` is the size in bytes required in a scratch buffer for an update operation.
- `buildScratchSize` is the size in bytes required in a scratch buffer for a build operation.

### Valid Usage (Implicit)

- VUID-VkAccelerationStructureBuildSizesInfoKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_ACCELERATION_STRUCTURE_BUILD_SIZES_INFO_KHR`
- VUID-VkAccelerationStructureBuildSizesInfoKHR-pNext-pNext
  `pNext` **must** be `NULL`

Values which **can** be set in VkAccelerationStructureCreateInfoKHR::`type` or VkAccelerationStructureInfoNV::`type` specifying the type of acceleration structure, are:

```
// Provided by VK_KHR_acceleration_structure
typedef enum VkAccelerationStructureTypeKHR {
    VK_ACCELERATION_STRUCTURE_TYPE_TOP_LEVEL_KHR = 0,
    VK_ACCELERATION_STRUCTURE_TYPE_BOTTOM_LEVEL_KHR = 1,
    VK_ACCELERATION_STRUCTURE_TYPE_GENERIC_KHR = 2,
  // Provided by VK_NV_ray_tracing
    VK_ACCELERATION_STRUCTURE_TYPE_TOP_LEVEL_NV =
VK_ACCELERATION_STRUCTURE_TYPE_TOP_LEVEL_KHR,
  // Provided by VK_NV_ray_tracing
    VK_ACCELERATION_STRUCTURE_TYPE_BOTTOM_LEVEL_NV =
VK_ACCELERATION_STRUCTURE_TYPE_BOTTOM_LEVEL_KHR,
} VkAccelerationStructureTypeKHR;
```

or the equivalent

```
// Provided by VK_NV_ray_tracing
typedef VkAccelerationStructureTypeKHR VkAccelerationStructureTypeNV;
```

- VK_ACCELERATION_STRUCTURE_TYPE_TOP_LEVEL_KHR is a top-level acceleration structure containing instance data referring to bottom-level acceleration structures.
- VK_ACCELERATION_STRUCTURE_TYPE_BOTTOM_LEVEL_KHR is a bottom-level acceleration structure containing the AABBs or geometry to be intersected.
- VK_ACCELERATION_STRUCTURE_TYPE_GENERIC_KHR is an acceleration structure whose type is determined at build time used for special circumstances. In these cases, the acceleration structure type is not known at creation time, but **must** be specified at build time as either top or bottom.

Bits which **can** be set in VkAccelerationStructureCreateInfoKHR::createFlags, specifying additional creation parameters for acceleration structures, are:

```
// Provided by VK_KHR_acceleration_structure
typedef enum VkAccelerationStructureCreateFlagBitsKHR {
    VK_ACCELERATION_STRUCTURE_CREATE_DEVICE_ADDRESS_CAPTURE_REPLAY_BIT_KHR =
0x00000001,
  // Provided by VK_EXT_descriptor_buffer
    VK_ACCELERATION_STRUCTURE_CREATE_DESCRIPTOR_BUFFER_CAPTURE_REPLAY_BIT_EXT =
0x00000008,
  // Provided by VK_NV_ray_tracing_motion_blur
    VK_ACCELERATION_STRUCTURE_CREATE_MOTION_BIT_NV = 0x00000004,
} VkAccelerationStructureCreateFlagBitsKHR;
```

- VK_ACCELERATION_STRUCTURE_CREATE_DEVICE_ADDRESS_CAPTURE_REPLAY_BIT_KHR specifies that the acceleration structure's address **can** be saved and reused on a subsequent run.
- VK_ACCELERATION_STRUCTURE_CREATE_DESCRIPTOR_BUFFER_CAPTURE_REPLAY_BIT_EXT specifies that the acceleration structure **can** be used with descriptor buffers when capturing and replaying (e.g. for trace capture and replay), see VkOpaqueCaptureDescriptorDataCreateInfoEXT for more detail.

```
// Provided by VK_KHR_acceleration_structure
typedef VkFlags VkAccelerationStructureCreateFlagsKHR;
```

VkAccelerationStructureCreateFlagsKHR is a bitmask type for setting a mask of zero or more VkAccelerationStructureCreateFlagBitsKHR.

Bits which **can** be set in VkAccelerationStructureBuildGeometryInfoKHR::flags or VkAccelerationStructureInfoNV::flags specifying additional parameters for acceleration structure builds, are:

```
// Provided by VK_KHR_acceleration_structure
typedef enum VkBuildAccelerationStructureFlagBitsKHR {
    VK_BUILD_ACCELERATION_STRUCTURE_ALLOW_UPDATE_BIT_KHR = 0x00000001,
    VK_BUILD_ACCELERATION_STRUCTURE_ALLOW_COMPACTION_BIT_KHR = 0x00000002,
    VK_BUILD_ACCELERATION_STRUCTURE_PREFER_FAST_TRACE_BIT_KHR = 0x00000004,
    VK_BUILD_ACCELERATION_STRUCTURE_PREFER_FAST_BUILD_BIT_KHR = 0x00000008,
    VK_BUILD_ACCELERATION_STRUCTURE_LOW_MEMORY_BIT_KHR = 0x00000010,
  // Provided by VK_NV_ray_tracing_motion_blur
    VK_BUILD_ACCELERATION_STRUCTURE_MOTION_BIT_NV = 0x00000020,
  // Provided by VK_EXT_opacity_micromap
    VK_BUILD_ACCELERATION_STRUCTURE_ALLOW_OPACITY_MICROMAP_UPDATE_EXT = 0x00000040,
  // Provided by VK_EXT_opacity_micromap
    VK_BUILD_ACCELERATION_STRUCTURE_ALLOW_DISABLE_OPACITY_MICROMAPS_EXT = 0x00000080,
  // Provided by VK_EXT_opacity_micromap
    VK_BUILD_ACCELERATION_STRUCTURE_ALLOW_OPACITY_MICROMAP_DATA_UPDATE_EXT =
0x00000100,
#ifdef VK_ENABLE_BETA_EXTENSIONS
  // Provided by VK_NV_displacement_micromap
    VK_BUILD_ACCELERATION_STRUCTURE_ALLOW_DISPLACEMENT_MICROMAP_UPDATE_NV =
0x00000200,
#endif
  // Provided by VK_KHR_ray_tracing_position_fetch
    VK_BUILD_ACCELERATION_STRUCTURE_ALLOW_DATA_ACCESS_KHR = 0x00000800,
  // Provided by VK_NV_ray_tracing
    VK_BUILD_ACCELERATION_STRUCTURE_ALLOW_UPDATE_BIT_NV =
VK_BUILD_ACCELERATION_STRUCTURE_ALLOW_UPDATE_BIT_KHR,
  // Provided by VK_NV_ray_tracing
    VK_BUILD_ACCELERATION_STRUCTURE_ALLOW_COMPACTION_BIT_NV =
VK_BUILD_ACCELERATION_STRUCTURE_ALLOW_COMPACTION_BIT_KHR,
  // Provided by VK_NV_ray_tracing
    VK_BUILD_ACCELERATION_STRUCTURE_PREFER_FAST_TRACE_BIT_NV =
VK_BUILD_ACCELERATION_STRUCTURE_PREFER_FAST_TRACE_BIT_KHR,
  // Provided by VK_NV_ray_tracing
    VK_BUILD_ACCELERATION_STRUCTURE_PREFER_FAST_BUILD_BIT_NV =
VK_BUILD_ACCELERATION_STRUCTURE_PREFER_FAST_BUILD_BIT_KHR,
  // Provided by VK_NV_ray_tracing
    VK_BUILD_ACCELERATION_STRUCTURE_LOW_MEMORY_BIT_NV =
VK_BUILD_ACCELERATION_STRUCTURE_LOW_MEMORY_BIT_KHR,
} VkBuildAccelerationStructureFlagBitsKHR;
```

or the equivalent

```
// Provided by VK_NV_ray_tracing
typedef VkBuildAccelerationStructureFlagBitsKHR
VkBuildAccelerationStructureFlagBitsNV;
```

- `VK_BUILD_ACCELERATION_STRUCTURE_ALLOW_UPDATE_BIT_KHR` indicates that the specified acceleration structure **can** be updated with a `mode` of `VK_BUILD_ACCELERATION_STRUCTURE_MODE_UPDATE_KHR` in

VkAccelerationStructureBuildGeometryInfoKHR or an update of VK_TRUE in vkCmdBuildAccelerationStructureNV .

- VK_BUILD_ACCELERATION_STRUCTURE_ALLOW_COMPACTION_BIT_KHR indicates that the specified acceleration structure **can** act as the source for a copy acceleration structure command with mode of VK_COPY_ACCELERATION_STRUCTURE_MODE_COMPACT_KHR to produce a compacted acceleration structure.
- VK_BUILD_ACCELERATION_STRUCTURE_PREFER_FAST_TRACE_BIT_KHR indicates that the given acceleration structure build **should** prioritize trace performance over build time.
- VK_BUILD_ACCELERATION_STRUCTURE_PREFER_FAST_BUILD_BIT_KHR indicates that the given acceleration structure build **should** prioritize build time over trace performance.
- VK_BUILD_ACCELERATION_STRUCTURE_LOW_MEMORY_BIT_KHR indicates that this acceleration structure **should** minimize the size of the scratch memory and the final result acceleration structure, potentially at the expense of build time or trace performance.
- VK_BUILD_ACCELERATION_STRUCTURE_ALLOW_OPACITY_MICROMAP_UPDATE_EXT indicates that the opacity micromaps associated with the specified acceleration structure **may** change with an acceleration structure update.
- VK_BUILD_ACCELERATION_STRUCTURE_ALLOW_OPACITY_MICROMAP_DATA_UPDATE_EXT indicates that the data of the opacity micromaps associated with the specified acceleration structure **may** change with an acceleration structure update.
- VK_BUILD_ACCELERATION_STRUCTURE_ALLOW_DISABLE_OPACITY_MICROMAPS_EXT indicates that the specified acceleration structure **may** be referenced in an instance with VK_GEOMETRY_INSTANCE_DISABLE_OPACITY_MICROMAPS_EXT set.
- VK_BUILD_ACCELERATION_STRUCTURE_ALLOW_DATA_ACCESS_KHR indicates that the specified acceleration structure **can** be used when fetching the vertex positions of a hit triangle.
- VK_BUILD_ACCELERATION_STRUCTURE_ALLOW_DISPLACEMENT_MICROMAP_UPDATE_NV indicates that the displacement micromaps associated with the specified acceleration structure **may** change with an acceleration structure update.

**NOTE** VK_BUILD_ACCELERATION_STRUCTURE_ALLOW_UPDATE_BIT_KHR and VK_BUILD_ACCELERATION_STRUCTURE_ALLOW_COMPACTION_BIT_KHR **may** take more time and memory than a normal build, and so **should** only be used when those features are needed.

**NOTE** VK_BUILD_ACCELERATION_STRUCTURE_ALLOW_UPDATE_BIT_KHR and VK_BUILD_ACCELERATION_STRUCTURE_ALLOW_COMPACTION_BIT_KHR are allowed to be used together. In that case, the result of the compaction copy is used as the source of a build with mode of VK_BUILD_ACCELERATION_STRUCTURE_MODE_UPDATE_KHR to perform the compacted update.

```
// Provided by VK_KHR_acceleration_structure
typedef VkFlags VkBuildAccelerationStructureFlagsKHR;
```

or the equivalent

```
// Provided by VK_NV_ray_tracing
typedef VkBuildAccelerationStructureFlagsKHR VkBuildAccelerationStructureFlagsNV;
```

VkBuildAccelerationStructureFlagsKHR is a bitmask type for setting a mask of zero or more VkBuildAccelerationStructureFlagBitsKHR.

The VkGeometryNV structure describes geometry in a bottom-level acceleration structure and is defined as:

```
// Provided by VK_NV_ray_tracing
typedef struct VkGeometryNV {
    VkStructureType       sType;
    const void*           pNext;
    VkGeometryTypeKHR     geometryType;
    VkGeometryDataNV      geometry;
    VkGeometryFlagsKHR    flags;
} VkGeometryNV;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- geometryType specifies the VkGeometryTypeKHR which this geometry refers to.
- geometry contains the geometry data as described in VkGeometryDataNV.
- flags has VkGeometryFlagBitsKHR describing options for this geometry.

### Valid Usage

- VUID-VkGeometryNV-geometryType-03503
  geometryType **must** be VK_GEOMETRY_TYPE_TRIANGLES_NV or VK_GEOMETRY_TYPE_AABBS_NV

### Valid Usage (Implicit)

- VUID-VkGeometryNV-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_GEOMETRY_NV
- VUID-VkGeometryNV-pNext-pNext
  pNext **must** be NULL
- VUID-VkGeometryNV-geometryType-parameter
  geometryType **must** be a valid VkGeometryTypeKHR value
- VUID-VkGeometryNV-geometry-parameter
  geometry **must** be a valid VkGeometryDataNV structure
- VUID-VkGeometryNV-flags-parameter
  flags **must** be a valid combination of VkGeometryFlagBitsKHR values

Geometry types are specified by VkGeometryTypeKHR, which takes values:

```
// Provided by VK_KHR_acceleration_structure
typedef enum VkGeometryTypeKHR {
    VK_GEOMETRY_TYPE_TRIANGLES_KHR = 0,
    VK_GEOMETRY_TYPE_AABBS_KHR = 1,
    VK_GEOMETRY_TYPE_INSTANCES_KHR = 2,
  // Provided by VK_NV_ray_tracing
    VK_GEOMETRY_TYPE_TRIANGLES_NV = VK_GEOMETRY_TYPE_TRIANGLES_KHR,
  // Provided by VK_NV_ray_tracing
    VK_GEOMETRY_TYPE_AABBS_NV = VK_GEOMETRY_TYPE_AABBS_KHR,
} VkGeometryTypeKHR;
```

or the equivalent

```
// Provided by VK_NV_ray_tracing
typedef VkGeometryTypeKHR VkGeometryTypeNV;
```

- VK_GEOMETRY_TYPE_TRIANGLES_KHR specifies a geometry type consisting of triangles.
- VK_GEOMETRY_TYPE_AABBS_KHR specifies a geometry type consisting of axis-aligned bounding boxes.
- VK_GEOMETRY_TYPE_INSTANCES_KHR specifies a geometry type consisting of acceleration structure instances.

Bits specifying additional parameters for geometries in acceleration structure builds, are:

```
// Provided by VK_KHR_acceleration_structure
typedef enum VkGeometryFlagBitsKHR {
    VK_GEOMETRY_OPAQUE_BIT_KHR = 0x00000001,
    VK_GEOMETRY_NO_DUPLICATE_ANY_HIT_INVOCATION_BIT_KHR = 0x00000002,
  // Provided by VK_NV_ray_tracing
    VK_GEOMETRY_OPAQUE_BIT_NV = VK_GEOMETRY_OPAQUE_BIT_KHR,
  // Provided by VK_NV_ray_tracing
    VK_GEOMETRY_NO_DUPLICATE_ANY_HIT_INVOCATION_BIT_NV =
VK_GEOMETRY_NO_DUPLICATE_ANY_HIT_INVOCATION_BIT_KHR,
} VkGeometryFlagBitsKHR;
```

or the equivalent

```
// Provided by VK_NV_ray_tracing
typedef VkGeometryFlagBitsKHR VkGeometryFlagBitsNV;
```

- VK_GEOMETRY_OPAQUE_BIT_KHR indicates that this geometry does not invoke the any-hit shaders even if present in a hit group.
- VK_GEOMETRY_NO_DUPLICATE_ANY_HIT_INVOCATION_BIT_KHR indicates that the implementation **must** only call the any-hit shader a single time for each primitive in this geometry. If this bit is absent

an implementation **may** invoke the any-hit shader more than once for this geometry.

```
// Provided by VK_KHR_acceleration_structure
typedef VkFlags VkGeometryFlagsKHR;
```

or the equivalent

```
// Provided by VK_NV_ray_tracing
typedef VkGeometryFlagsKHR VkGeometryFlagsNV;
```

`VkGeometryFlagsKHR` is a bitmask type for setting a mask of zero or more VkGeometryFlagBitsKHR.

The `VkGeometryDataNV` structure specifies geometry in a bottom-level acceleration structure and is defined as:

```
// Provided by VK_NV_ray_tracing
typedef struct VkGeometryDataNV {
    VkGeometryTrianglesNV    triangles;
    VkGeometryAABBNV         aabbs;
} VkGeometryDataNV;
```

- `triangles` contains triangle data if VkGeometryNV::`geometryType` is `VK_GEOMETRY_TYPE_TRIANGLES_NV`.
- `aabbs` contains axis-aligned bounding box data if VkGeometryNV::`geometryType` is `VK_GEOMETRY_TYPE_AABBS_NV`.

### Valid Usage (Implicit)

- VUID-VkGeometryDataNV-triangles-parameter
  `triangles` **must** be a valid VkGeometryTrianglesNV structure
- VUID-VkGeometryDataNV-aabbs-parameter
  `aabbs` **must** be a valid VkGeometryAABBNV structure

The `VkGeometryTrianglesNV` structure specifies triangle geometry in a bottom-level acceleration structure and is defined as:

```
// Provided by VK_NV_ray_tracing
typedef struct VkGeometryTrianglesNV {
    VkStructureType    sType;
    const void*        pNext;
    VkBuffer           vertexData;
    VkDeviceSize       vertexOffset;
    uint32_t           vertexCount;
    VkDeviceSize       vertexStride;
    VkFormat           vertexFormat;
    VkBuffer           indexData;
    VkDeviceSize       indexOffset;
    uint32_t           indexCount;
    VkIndexType        indexType;
    VkBuffer           transformData;
    VkDeviceSize       transformOffset;
} VkGeometryTrianglesNV;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `vertexData` is the buffer containing vertex data for this geometry.
- `vertexOffset` is the offset in bytes within `vertexData` containing vertex data for this geometry.
- `vertexCount` is the number of valid vertices.
- `vertexStride` is the stride in bytes between each vertex.
- `vertexFormat` is a VkFormat describing the format of each vertex element.
- `indexData` is the buffer containing index data for this geometry.
- `indexOffset` is the offset in bytes within `indexData` containing index data for this geometry.
- `indexCount` is the number of indices to include in this geometry.
- `indexType` is a VkIndexType describing the format of each index.
- `transformData` is an optional buffer containing an VkTransformMatrixNV structure defining a transformation to be applied to this geometry.
- `transformOffset` is the offset in bytes in `transformData` of the transform information described above.

If `indexType` is `VK_INDEX_TYPE_NONE_NV`, then this structure describes a set of triangles determined by `vertexCount`. Otherwise, this structure describes a set of indexed triangles determined by `indexCount`.

### Valid Usage

- VUID-VkGeometryTrianglesNV-vertexOffset-02428
  `vertexOffset` **must** be less than the size of `vertexData`
- VUID-VkGeometryTrianglesNV-vertexOffset-02429
  `vertexOffset` **must** be a multiple of the component size of `vertexFormat`

- VUID-VkGeometryTrianglesNV-vertexFormat-02430
  `vertexFormat` **must** be one of `VK_FORMAT_R32G32B32_SFLOAT`, `VK_FORMAT_R32G32_SFLOAT`, `VK_FORMAT_R16G16B16_SFLOAT`, `VK_FORMAT_R16G16_SFLOAT`, `VK_FORMAT_R16G16_SNORM`, or `VK_FORMAT_R16G16B16_SNORM`
- VUID-VkGeometryTrianglesNV-vertexStride-03818
  `vertexStride` **must** be less than or equal to $2^{32}$-1
- VUID-VkGeometryTrianglesNV-indexOffset-02431
  `indexOffset` **must** be less than the size of `indexData`
- VUID-VkGeometryTrianglesNV-indexOffset-02432
  `indexOffset` **must** be a multiple of the element size of `indexType`
- VUID-VkGeometryTrianglesNV-indexType-02433
  `indexType` **must** be `VK_INDEX_TYPE_UINT16`, `VK_INDEX_TYPE_UINT32`, or `VK_INDEX_TYPE_NONE_NV`
- VUID-VkGeometryTrianglesNV-indexData-02434
  `indexData` **must** be VK_NULL_HANDLE if `indexType` is `VK_INDEX_TYPE_NONE_NV`
- VUID-VkGeometryTrianglesNV-indexData-02435
  `indexData` **must** be a valid `VkBuffer` handle if `indexType` is not `VK_INDEX_TYPE_NONE_NV`
- VUID-VkGeometryTrianglesNV-indexCount-02436
  `indexCount` **must** be `0` if `indexType` is `VK_INDEX_TYPE_NONE_NV`
- VUID-VkGeometryTrianglesNV-transformOffset-02437
  `transformOffset` **must** be less than the size of `transformData`
- VUID-VkGeometryTrianglesNV-transformOffset-02438
  `transformOffset` **must** be a multiple of `16`

## Valid Usage (Implicit)

- VUID-VkGeometryTrianglesNV-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_GEOMETRY_TRIANGLES_NV`
- VUID-VkGeometryTrianglesNV-pNext-pNext
  `pNext` **must** be `NULL`
- VUID-VkGeometryTrianglesNV-vertexData-parameter
  If `vertexData` is not VK_NULL_HANDLE, `vertexData` **must** be a valid VkBuffer handle
- VUID-VkGeometryTrianglesNV-vertexFormat-parameter
  `vertexFormat` **must** be a valid VkFormat value
- VUID-VkGeometryTrianglesNV-indexData-parameter
  If `indexData` is not VK_NULL_HANDLE, `indexData` **must** be a valid VkBuffer handle
- VUID-VkGeometryTrianglesNV-indexType-parameter
  `indexType` **must** be a valid VkIndexType value
- VUID-VkGeometryTrianglesNV-transformData-parameter
  If `transformData` is not VK_NULL_HANDLE, `transformData` **must** be a valid VkBuffer handle
- VUID-VkGeometryTrianglesNV-commonparent

Each of indexData, transformData, and vertexData that are valid handles of non-ignored parameters **must** have been created, allocated, or retrieved from the same VkDevice

The VkGeometryAABBNV structure specifies axis-aligned bounding box geometry in a bottom-level acceleration structure, and is defined as:

```
// Provided by VK_NV_ray_tracing
typedef struct VkGeometryAABBNV {
    VkStructureType    sType;
    const void*        pNext;
    VkBuffer           aabbData;
    uint32_t           numAABBs;
    uint32_t           stride;
    VkDeviceSize       offset;
} VkGeometryAABBNV;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- aabbData is the buffer containing axis-aligned bounding box data.
- numAABBs is the number of AABBs in this geometry.
- stride is the stride in bytes between AABBs in aabbData.
- offset is the offset in bytes of the first AABB in aabbData.

The AABB data in memory is six 32-bit floats consisting of the minimum x, y, and z values followed by the maximum x, y, and z values.

### Valid Usage

- VUID-VkGeometryAABBNV-offset-02439
  offset **must** be less than the size of aabbData
- VUID-VkGeometryAABBNV-offset-02440
  offset **must** be a multiple of 8
- VUID-VkGeometryAABBNV-stride-02441
  stride **must** be a multiple of 8

### Valid Usage (Implicit)

- VUID-VkGeometryAABBNV-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_GEOMETRY_AABB_NV
- VUID-VkGeometryAABBNV-pNext-pNext
  pNext **must** be NULL
- VUID-VkGeometryAABBNV-aabbData-parameter

If aabbData is not VK_NULL_HANDLE, aabbData **must** be a valid VkBuffer handle

To destroy an acceleration structure, call:

```
// Provided by VK_KHR_acceleration_structure
void vkDestroyAccelerationStructureKHR(
    VkDevice                                    device,
    VkAccelerationStructureKHR                  accelerationStructure,
    const VkAllocationCallbacks*                pAllocator);
```

- device is the logical device that destroys the acceleration structure.
- accelerationStructure is the acceleration structure to destroy.
- pAllocator controls host memory allocation as described in the Memory Allocation chapter.

## Valid Usage

- VUID-vkDestroyAccelerationStructureKHR-accelerationStructure-08934
  The VkPhysicalDeviceAccelerationStructureFeaturesKHR::accelerationStructure feature **must** be enabled
- VUID-vkDestroyAccelerationStructureKHR-accelerationStructure-02442
  All submitted commands that refer to accelerationStructure **must** have completed execution
- VUID-vkDestroyAccelerationStructureKHR-accelerationStructure-02443
  If VkAllocationCallbacks were provided when accelerationStructure was created, a compatible set of callbacks **must** be provided here
- VUID-vkDestroyAccelerationStructureKHR-accelerationStructure-02444
  If no VkAllocationCallbacks were provided when accelerationStructure was created, pAllocator **must** be NULL

## Valid Usage (Implicit)

- VUID-vkDestroyAccelerationStructureKHR-device-parameter
  device **must** be a valid VkDevice handle
- VUID-vkDestroyAccelerationStructureKHR-accelerationStructure-parameter
  If accelerationStructure is not VK_NULL_HANDLE, accelerationStructure **must** be a valid VkAccelerationStructureKHR handle
- VUID-vkDestroyAccelerationStructureKHR-pAllocator-parameter
  If pAllocator is not NULL, pAllocator **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkDestroyAccelerationStructureKHR-accelerationStructure-parent
  If accelerationStructure is a valid handle, it **must** have been created, allocated, or retrieved from device

## Host Synchronization

- Host access to `accelerationStructure` **must** be externally synchronized

To destroy an acceleration structure, call:

```
// Provided by VK_NV_ray_tracing
void vkDestroyAccelerationStructureNV(
    VkDevice                                    device,
    VkAccelerationStructureNV                   accelerationStructure,
    const VkAllocationCallbacks*                pAllocator);
```

- `device` is the logical device that destroys the buffer.
- `accelerationStructure` is the acceleration structure to destroy.
- `pAllocator` controls host memory allocation as described in the Memory Allocation chapter.

## Valid Usage

- VUID-vkDestroyAccelerationStructureNV-accelerationStructure-03752
  All submitted commands that refer to `accelerationStructure` **must** have completed execution
- VUID-vkDestroyAccelerationStructureNV-accelerationStructure-03753
  If `VkAllocationCallbacks` were provided when `accelerationStructure` was created, a compatible set of callbacks **must** be provided here
- VUID-vkDestroyAccelerationStructureNV-accelerationStructure-03754
  If no `VkAllocationCallbacks` were provided when `accelerationStructure` was created, `pAllocator` **must** be `NULL`

## Valid Usage (Implicit)

- VUID-vkDestroyAccelerationStructureNV-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkDestroyAccelerationStructureNV-accelerationStructure-parameter
  If `accelerationStructure` is not VK_NULL_HANDLE, `accelerationStructure` **must** be a valid VkAccelerationStructureNV handle
- VUID-vkDestroyAccelerationStructureNV-pAllocator-parameter
  If `pAllocator` is not `NULL`, `pAllocator` **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkDestroyAccelerationStructureNV-accelerationStructure-parent
  If `accelerationStructure` is a valid handle, it **must** have been created, allocated, or retrieved from `device`

### Host Synchronization

- Host access to `accelerationStructure` **must** be externally synchronized

An acceleration structure has memory requirements for the structure object itself, scratch space for the build, and scratch space for the update.

Scratch space is allocated as a `VkBuffer`, so for `VK_ACCELERATION_STRUCTURE_MEMORY_REQUIREMENTS_TYPE_BUILD_SCRATCH_NV` and `VK_ACCELERATION_STRUCTURE_MEMORY_REQUIREMENTS_TYPE_UPDATE_SCRATCH_NV` the `pMemoryRequirements->alignment` and `pMemoryRequirements->memoryTypeBits` values returned by this call **must** be filled with zero, and **should** be ignored by the application.

To query the memory requirements, call:

```
// Provided by VK_NV_ray_tracing
void vkGetAccelerationStructureMemoryRequirementsNV(
    VkDevice                                    device,
    const VkAccelerationStructureMemoryRequirementsInfoNV* pInfo,
    VkMemoryRequirements2KHR*                   pMemoryRequirements);
```

- `device` is the logical device on which the acceleration structure was created.
- `pInfo` is a pointer to a VkAccelerationStructureMemoryRequirementsInfoNV structure specifying the acceleration structure to get memory requirements for.
- `pMemoryRequirements` is a pointer to a VkMemoryRequirements2KHR structure in which the requested acceleration structure memory requirements are returned.

### Valid Usage (Implicit)

- VUID-vkGetAccelerationStructureMemoryRequirementsNV-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkGetAccelerationStructureMemoryRequirementsNV-pInfo-parameter
  `pInfo` **must** be a valid pointer to a valid VkAccelerationStructureMemoryRequirementsInfoNV structure
- VUID-vkGetAccelerationStructureMemoryRequirementsNV-pMemoryRequirements-parameter
  `pMemoryRequirements` **must** be a valid pointer to a VkMemoryRequirements2KHR structure

The `VkAccelerationStructureMemoryRequirementsInfoNV` structure is defined as:

```
// Provided by VK_NV_ray_tracing
typedef struct VkAccelerationStructureMemoryRequirementsInfoNV {
    VkStructureType                                     sType;
    const void*                                         pNext;
    VkAccelerationStructureMemoryRequirementsTypeNV     type;
    VkAccelerationStructureNV                           accelerationStructure;
} VkAccelerationStructureMemoryRequirementsInfoNV;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `type` selects the type of memory requirement being queried. `VK_ACCELERATION_STRUCTURE_MEMORY_REQUIREMENTS_TYPE_OBJECT_NV` returns the memory requirements for the object itself. `VK_ACCELERATION_STRUCTURE_MEMORY_REQUIREMENTS_TYPE_BUILD_SCRATCH_NV` returns the memory requirements for the scratch memory when doing a build. `VK_ACCELERATION_STRUCTURE_MEMORY_REQUIREMENTS_TYPE_UPDATE_SCRATCH_NV` returns the memory requirements for the scratch memory when doing an update.
- `accelerationStructure` is the acceleration structure to be queried for memory requirements.

### Valid Usage (Implicit)

- VUID-VkAccelerationStructureMemoryRequirementsInfoNV-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_ACCELERATION_STRUCTURE_MEMORY_REQUIREMENTS_INFO_NV`
- VUID-VkAccelerationStructureMemoryRequirementsInfoNV-pNext-pNext
  `pNext` **must** be `NULL`
- VUID-VkAccelerationStructureMemoryRequirementsInfoNV-type-parameter
  `type` **must** be a valid VkAccelerationStructureMemoryRequirementsTypeNV value
- VUID-VkAccelerationStructureMemoryRequirementsInfoNV-accelerationStructure-parameter
  `accelerationStructure` **must** be a valid VkAccelerationStructureNV handle

Possible values of `type` in `VkAccelerationStructureMemoryRequirementsInfoNV` are:,

```
// Provided by VK_NV_ray_tracing
typedef enum VkAccelerationStructureMemoryRequirementsTypeNV {
    VK_ACCELERATION_STRUCTURE_MEMORY_REQUIREMENTS_TYPE_OBJECT_NV = 0,
    VK_ACCELERATION_STRUCTURE_MEMORY_REQUIREMENTS_TYPE_BUILD_SCRATCH_NV = 1,
    VK_ACCELERATION_STRUCTURE_MEMORY_REQUIREMENTS_TYPE_UPDATE_SCRATCH_NV = 2,
} VkAccelerationStructureMemoryRequirementsTypeNV;
```

- `VK_ACCELERATION_STRUCTURE_MEMORY_REQUIREMENTS_TYPE_OBJECT_NV` requests the memory requirement for the `VkAccelerationStructureNV` backing store.
- `VK_ACCELERATION_STRUCTURE_MEMORY_REQUIREMENTS_TYPE_BUILD_SCRATCH_NV` requests the memory

requirement for scratch space during the initial build.

- VK_ACCELERATION_STRUCTURE_MEMORY_REQUIREMENTS_TYPE_UPDATE_SCRATCH_NV requests the memory requirement for scratch space during an update.

Possible values of buildType in vkGetAccelerationStructureBuildSizesKHR are:

```
// Provided by VK_KHR_acceleration_structure
typedef enum VkAccelerationStructureBuildTypeKHR {
    VK_ACCELERATION_STRUCTURE_BUILD_TYPE_HOST_KHR = 0,
    VK_ACCELERATION_STRUCTURE_BUILD_TYPE_DEVICE_KHR = 1,
    VK_ACCELERATION_STRUCTURE_BUILD_TYPE_HOST_OR_DEVICE_KHR = 2,
} VkAccelerationStructureBuildTypeKHR;
```

- VK_ACCELERATION_STRUCTURE_BUILD_TYPE_HOST_KHR requests the memory requirement for operations performed by the host.
- VK_ACCELERATION_STRUCTURE_BUILD_TYPE_DEVICE_KHR requests the memory requirement for operations performed by the device.
- VK_ACCELERATION_STRUCTURE_BUILD_TYPE_HOST_OR_DEVICE_KHR requests the memory requirement for operations performed by either the host, or the device.

To attach memory to one or more acceleration structures at a time, call:

```
// Provided by VK_NV_ray_tracing
VkResult vkBindAccelerationStructureMemoryNV(
    VkDevice                                    device,
    uint32_t                                    bindInfoCount,
    const VkBindAccelerationStructureMemoryInfoNV* pBindInfos);
```

- device is the logical device that owns the acceleration structures and memory.
- bindInfoCount is the number of elements in pBindInfos.
- pBindInfos is a pointer to an array of VkBindAccelerationStructureMemoryInfoNV structures describing acceleration structures and memory to bind.

### Valid Usage (Implicit)

- VUID-vkBindAccelerationStructureMemoryNV-device-parameter
  device **must** be a valid VkDevice handle
- VUID-vkBindAccelerationStructureMemoryNV-pBindInfos-parameter
  pBindInfos **must** be a valid pointer to an array of bindInfoCount valid VkBindAccelerationStructureMemoryInfoNV structures
- VUID-vkBindAccelerationStructureMemoryNV-bindInfoCount-arraylength
  bindInfoCount **must** be greater than 0

## Return Codes

**Success**

- VK_SUCCESS

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY

The VkBindAccelerationStructureMemoryInfoNV structure is defined as:

```
// Provided by VK_NV_ray_tracing
typedef struct VkBindAccelerationStructureMemoryInfoNV {
    VkStructureType              sType;
    const void*                  pNext;
    VkAccelerationStructureNV    accelerationStructure;
    VkDeviceMemory               memory;
    VkDeviceSize                 memoryOffset;
    uint32_t                     deviceIndexCount;
    const uint32_t*              pDeviceIndices;
} VkBindAccelerationStructureMemoryInfoNV;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- accelerationStructure is the acceleration structure to be attached to memory.
- memory is a VkDeviceMemory object describing the device memory to attach.
- memoryOffset is the start offset of the region of memory that is to be bound to the acceleration structure. The number of bytes returned in the VkMemoryRequirements::size member in memory, starting from memoryOffset bytes, will be bound to the specified acceleration structure.
- deviceIndexCount is the number of elements in pDeviceIndices.
- pDeviceIndices is a pointer to an array of device indices.

## Valid Usage

- VUID-VkBindAccelerationStructureMemoryInfoNV-accelerationStructure-03620
  accelerationStructure **must** not already be backed by a memory object
- VUID-VkBindAccelerationStructureMemoryInfoNV-memoryOffset-03621
  memoryOffset **must** be less than the size of memory
- VUID-VkBindAccelerationStructureMemoryInfoNV-memory-03622
  memory **must** have been allocated using one of the memory types allowed in the memoryTypeBits member of the VkMemoryRequirements structure returned from a call to vkGetAccelerationStructureMemoryRequirementsNV with accelerationStructure and type of VK_ACCELERATION_STRUCTURE_MEMORY_REQUIREMENTS_TYPE_OBJECT_NV

- VUID-VkBindAccelerationStructureMemoryInfoNV-memoryOffset-03623
  `memoryOffset` **must** be an integer multiple of the `alignment` member of the VkMemoryRequirements structure returned from a call to vkGetAccelerationStructureMemoryRequirementsNV with `accelerationStructure` and `type` of `VK_ACCELERATION_STRUCTURE_MEMORY_REQUIREMENTS_TYPE_OBJECT_NV`
- VUID-VkBindAccelerationStructureMemoryInfoNV-size-03624
  The `size` member of the `VkMemoryRequirements` structure returned from a call to vkGetAccelerationStructureMemoryRequirementsNV with `accelerationStructure` and `type` of `VK_ACCELERATION_STRUCTURE_MEMORY_REQUIREMENTS_TYPE_OBJECT_NV` **must** be less than or equal to the size of `memory` minus `memoryOffset`

### Valid Usage (Implicit)

- VUID-VkBindAccelerationStructureMemoryInfoNV-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_BIND_ACCELERATION_STRUCTURE_MEMORY_INFO_NV`
- VUID-VkBindAccelerationStructureMemoryInfoNV-pNext-pNext
  `pNext` **must** be `NULL`
- VUID-VkBindAccelerationStructureMemoryInfoNV-accelerationStructure-parameter
  `accelerationStructure` **must** be a valid VkAccelerationStructureNV handle
- VUID-VkBindAccelerationStructureMemoryInfoNV-memory-parameter
  `memory` **must** be a valid VkDeviceMemory handle
- VUID-VkBindAccelerationStructureMemoryInfoNV-pDeviceIndices-parameter
  If `deviceIndexCount` is not `0`, `pDeviceIndices` **must** be a valid pointer to an array of `deviceIndexCount` `uint32_t` values
- VUID-VkBindAccelerationStructureMemoryInfoNV-commonparent
  Both of `accelerationStructure`, and `memory` **must** have been created, allocated, or retrieved from the same VkDevice

To allow constructing geometry instances with device code if desired, we need to be able to query an opaque handle for an acceleration structure. This handle is a value of 8 bytes. To get this handle, call:

```
// Provided by VK_NV_ray_tracing
VkResult vkGetAccelerationStructureHandleNV(
    VkDevice                                    device,
    VkAccelerationStructureNV                   accelerationStructure,
    size_t                                      dataSize,
    void*                                       pData);
```

- `device` is the logical device that owns the acceleration structures.
- `accelerationStructure` is the acceleration structure.
- `dataSize` is the size in bytes of the buffer pointed to by `pData`.

- `pData` is a pointer to an application-allocated buffer where the results will be written.

### Valid Usage

- VUID-vkGetAccelerationStructureHandleNV-dataSize-02240
  `dataSize` **must** be large enough to contain the result of the query, as described above
- VUID-vkGetAccelerationStructureHandleNV-accelerationStructure-02787
  `accelerationStructure` **must** be bound completely and contiguously to a single `VkDeviceMemory` object via vkBindAccelerationStructureMemoryNV

### Valid Usage (Implicit)

- VUID-vkGetAccelerationStructureHandleNV-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkGetAccelerationStructureHandleNV-accelerationStructure-parameter
  `accelerationStructure` **must** be a valid VkAccelerationStructureNV handle
- VUID-vkGetAccelerationStructureHandleNV-pData-parameter
  `pData` **must** be a valid pointer to an array of `dataSize` bytes
- VUID-vkGetAccelerationStructureHandleNV-dataSize-arraylength
  `dataSize` **must** be greater than `0`
- VUID-vkGetAccelerationStructureHandleNV-accelerationStructure-parent
  `accelerationStructure` **must** have been created, allocated, or retrieved from `device`

### Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`

To query the 64-bit device address for an acceleration structure, call:

```
// Provided by VK_KHR_acceleration_structure
VkDeviceAddress vkGetAccelerationStructureDeviceAddressKHR(
    VkDevice                                    device,
    const VkAccelerationStructureDeviceAddressInfoKHR* pInfo);
```

- `device` is the logical device that the acceleration structure was created on.
- `pInfo` is a pointer to a VkAccelerationStructureDeviceAddressInfoKHR structure specifying the

acceleration structure to retrieve an address for.

The 64-bit return value is an address of the acceleration structure, which can be used for device and shader operations that involve acceleration structures, such as ray traversal and acceleration structure building.

If the acceleration structure was created with a non-zero value of VkAccelerationStructureCreateInfoKHR::deviceAddress, the return value will be the same address.

If the acceleration structure was created with a type of VK_ACCELERATION_STRUCTURE_TYPE_GENERIC_KHR, the returned address **must** be consistent with the relative offset to other acceleration structures with type VK_ACCELERATION_STRUCTURE_TYPE_GENERIC_KHR allocated with the same VkBuffer. That is, the difference in returned addresses between the two **must** be the same as the difference in offsets provided at acceleration structure creation.

The returned address **must** be aligned to 256 bytes.

**NOTE** The acceleration structure device address **may** be different from the buffer device address corresponding to the acceleration structure's start offset in its storage buffer for acceleration structure types other than VK_ACCELERATION_STRUCTURE_TYPE_GENERIC_KHR.

## Valid Usage

- VUID-vkGetAccelerationStructureDeviceAddressKHR-accelerationStructure-08935
  The VkPhysicalDeviceAccelerationStructureFeaturesKHR::accelerationStructure feature **must** be enabled
- VUID-vkGetAccelerationStructureDeviceAddressKHR-device-03504
  If device was created with multiple physical devices, then the bufferDeviceAddressMultiDevice feature **must** be enabled
- VUID-vkGetAccelerationStructureDeviceAddressKHR-pInfo-09541
  If the buffer on which pInfo->accelerationStructure was placed is non-sparse then it **must** be bound completely and contiguously to a single VkDeviceMemory object
- VUID-vkGetAccelerationStructureDeviceAddressKHR-pInfo-09542
  The buffer on which pInfo->accelerationStructure was placed **must** have been created with the VK_BUFFER_USAGE_SHADER_DEVICE_ADDRESS_BIT usage flag

## Valid Usage (Implicit)

- VUID-vkGetAccelerationStructureDeviceAddressKHR-device-parameter
  device **must** be a valid VkDevice handle
- VUID-vkGetAccelerationStructureDeviceAddressKHR-pInfo-parameter
  pInfo **must** be a valid pointer to a valid VkAccelerationStructureDeviceAddressInfoKHR structure

The VkAccelerationStructureDeviceAddressInfoKHR structure is defined as:

```
// Provided by VK_KHR_acceleration_structure
typedef struct VkAccelerationStructureDeviceAddressInfoKHR {
    VkStructureType                 sType;
    const void*                     pNext;
    VkAccelerationStructureKHR      accelerationStructure;
} VkAccelerationStructureDeviceAddressInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- accelerationStructure specifies the acceleration structure whose address is being queried.

**Valid Usage (Implicit)**

- VUID-VkAccelerationStructureDeviceAddressInfoKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_ACCELERATION_STRUCTURE_DEVICE_ADDRESS_INFO_KHR
- VUID-VkAccelerationStructureDeviceAddressInfoKHR-pNext-pNext
  pNext **must** be NULL
- VUID-VkAccelerationStructureDeviceAddressInfoKHR-accelerationStructure-parameter
  accelerationStructure **must** be a valid VkAccelerationStructureKHR handle

## 12.7. Micromaps

Micromaps are opaque data structures that are built by the implementation to encode sub-triangle data to be included in an acceleration structure.

Micromaps are represented by VkMicromapEXT handles:

```
// Provided by VK_EXT_opacity_micromap
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkMicromapEXT)
```

To create a micromap, call:

```
// Provided by VK_EXT_opacity_micromap
VkResult vkCreateMicromapEXT(
    VkDevice                                    device,
    const VkMicromapCreateInfoEXT*              pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkMicromapEXT*                              pMicromap);
```

- device is the logical device that creates the acceleration structure object.
- pCreateInfo is a pointer to a VkMicromapCreateInfoEXT structure containing parameters

affecting creation of the micromap.

- `pAllocator` controls host memory allocation as described in the Memory Allocation chapter.
- `pMicromap` is a pointer to a `VkMicromapEXT` handle in which the resulting micromap object is returned.

Similar to other objects in Vulkan, the micromap creation merely creates an object with a specific "shape". The type and quantity of geometry that can be built into a micromap is determined by the parameters of VkMicromapCreateInfoEXT.

The micromap data is stored in the object referred to by `VkMicromapCreateInfoEXT::buffer`. Once memory has been bound to that buffer, it **must** be populated by micromap build or micromap copy commands such as vkCmdBuildMicromapsEXT, vkBuildMicromapsEXT, vkCmdCopyMicromapEXT, and vkCopyMicromapEXT.

**NOTE** The expected usage for a trace capture/replay tool is that it will serialize and later deserialize the micromap data using micromap copy commands. During capture the tool will use vkCopyMicromapToMemoryEXT or vkCmdCopyMicromapToMemoryEXT with a `mode` of `VK_COPY_MICROMAP_MODE_SERIALIZE_EXT`, and vkCopyMemoryToMicromapEXT or vkCmdCopyMemoryToMicromapEXT with a `mode` of `VK_COPY_MICROMAP_MODE_DESERIALIZE_EXT` during replay.

The input buffers passed to micromap build commands will be referenced by the implementation for the duration of the command. Micromaps **must** be fully self-contained. The application **can** reuse or free any memory which was used by the command as an input or as scratch without affecting the results of a subsequent acceleration structure build using the micromap or traversal of that acceleration structure.

### Valid Usage

- VUID-vkCreateMicromapEXT-micromap-07430
  The `micromap` feature **must** be enabled
- VUID-vkCreateMicromapEXT-deviceAddress-07431
  If VkMicromapCreateInfoEXT::`deviceAddress` is not zero, the `micromapCaptureReplay` feature **must** be enabled
- VUID-vkCreateMicromapEXT-device-07432
  If `device` was created with multiple physical devices, then the `bufferDeviceAddressMultiDevice` feature **must** be enabled

### Valid Usage (Implicit)

- VUID-vkCreateMicromapEXT-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkCreateMicromapEXT-pCreateInfo-parameter
  `pCreateInfo` **must** be a valid pointer to a valid VkMicromapCreateInfoEXT structure

- VUID-vkCreateMicromapEXT-pAllocator-parameter
  If `pAllocator` is not `NULL`, `pAllocator` **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkCreateMicromapEXT-pMicromap-parameter
  `pMicromap` **must** be a valid pointer to a VkMicromapEXT handle

## Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_INVALID_OPAQUE_CAPTURE_ADDRESS_KHR`

The `VkMicromapCreateInfoEXT` structure is defined as:

```
// Provided by VK_EXT_opacity_micromap
typedef struct VkMicromapCreateInfoEXT {
    VkStructureType             sType;
    const void*                 pNext;
    VkMicromapCreateFlagsEXT    createFlags;
    VkBuffer                    buffer;
    VkDeviceSize                offset;
    VkDeviceSize                size;
    VkMicromapTypeEXT           type;
    VkDeviceAddress             deviceAddress;
} VkMicromapCreateInfoEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `createFlags` is a bitmask of VkMicromapCreateFlagBitsEXT specifying additional creation parameters of the micromap.
- `buffer` is the buffer on which the micromap will be stored.
- `offset` is an offset in bytes from the base address of the buffer at which the micromap will be stored, and **must** be a multiple of `256`.
- `size` is the size required for the micromap.
- `type` is a VkMicromapTypeEXT value specifying the type of micromap that will be created.
- `deviceAddress` is the device address requested for the micromap if the `micromapCaptureReplay` feature is being used.

If `deviceAddress` is zero, no specific address is requested.

If deviceAddress is not zero, deviceAddress **must** be an address retrieved from an identically created micromap on the same implementation. The micromap **must** also be placed on an identically created buffer and at the same offset.

Applications **should** avoid creating micromaps with application-provided addresses and implementation-provided addresses in the same process, to reduce the likelihood of VK_ERROR_INVALID_OPAQUE_CAPTURE_ADDRESS_KHR errors.

**NOTE**

The expected usage for this is that a trace capture/replay tool will add the VK_BUFFER_CREATE_DEVICE_ADDRESS_CAPTURE_REPLAY_BIT flag to all buffers that use VK_BUFFER_USAGE_SHADER_DEVICE_ADDRESS_BIT, and will add VK_BUFFER_USAGE_SHADER_DEVICE_ADDRESS_BIT to all buffers used as storage for a micromap where deviceAddress is not zero. This also means that the tool will need to add VK_MEMORY_ALLOCATE_DEVICE_ADDRESS_BIT to memory allocations to allow the flag to be set where the application may not have otherwise required it. During capture the tool will save the queried opaque device addresses in the trace. During replay, the buffers will be created specifying the original address so any address values stored in the trace data will remain valid.

Implementations are expected to separate such buffers in the GPU address space so normal allocations will avoid using these addresses. Apps/tools should avoid mixing app-provided and implementation-provided addresses for buffers created with VK_BUFFER_CREATE_DEVICE_ADDRESS_CAPTURE_REPLAY_BIT, to avoid address space allocation conflicts.

If the micromap will be the target of a build operation, the required size for a micromap **can** be queried with vkGetMicromapBuildSizesEXT.

## Valid Usage

- VUID-VkMicromapCreateInfoEXT-deviceAddress-07433
  If deviceAddress is not zero, createFlags **must** include VK_MICROMAP_CREATE_DEVICE_ADDRESS_CAPTURE_REPLAY_BIT_EXT
- VUID-VkMicromapCreateInfoEXT-createFlags-07434
  If createFlags includes VK_MICROMAP_CREATE_DEVICE_ADDRESS_CAPTURE_REPLAY_BIT_EXT, VkPhysicalDeviceOpacityMicromapFeaturesEXT::micromapCaptureReplay **must** be VK_TRUE
- VUID-VkMicromapCreateInfoEXT-buffer-07435
  buffer **must** have been created with a usage value containing VK_BUFFER_USAGE_MICROMAP_STORAGE_BIT_EXT
- VUID-VkMicromapCreateInfoEXT-buffer-07436
  buffer **must** not have been created with VK_BUFFER_CREATE_SPARSE_RESIDENCY_BIT
- VUID-VkMicromapCreateInfoEXT-offset-07437
  The sum of offset and size **must** be less than the size of buffer
- VUID-VkMicromapCreateInfoEXT-offset-07438
  offset **must** be a multiple of 256 bytes

### Valid Usage (Implicit)

- VUID-VkMicromapCreateInfoEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_MICROMAP_CREATE_INFO_EXT`
- VUID-VkMicromapCreateInfoEXT-pNext-pNext
  `pNext` **must** be `NULL`
- VUID-VkMicromapCreateInfoEXT-createFlags-parameter
  `createFlags` **must** be a valid combination of VkMicromapCreateFlagBitsEXT values
- VUID-VkMicromapCreateInfoEXT-buffer-parameter
  `buffer` **must** be a valid VkBuffer handle
- VUID-VkMicromapCreateInfoEXT-type-parameter
  `type` **must** be a valid VkMicromapTypeEXT value

To get the build sizes for a micromap, call:

```
// Provided by VK_EXT_opacity_micromap
void vkGetMicromapBuildSizesEXT(
    VkDevice                                    device,
    VkAccelerationStructureBuildTypeKHR         buildType,
    const VkMicromapBuildInfoEXT*               pBuildInfo,
    VkMicromapBuildSizesInfoEXT*                pSizeInfo);
```

- `device` is the logical device that will be used for creating the micromap.
- `buildType` defines whether host or device operations (or both) are being queried for.
- `pBuildInfo` is a pointer to a VkMicromapBuildInfoEXT structure describing parameters of a build operation.
- `pSizeInfo` is a pointer to a VkMicromapBuildSizesInfoEXT structure which returns the size required for a micromap and the sizes required for the scratch buffers, given the build parameters.

The `dstMicromap` and `mode` members of `pBuildInfo` are ignored. Any VkDeviceOrHostAddressKHR members of `pBuildInfo` are ignored by this command.

A micromap created with the `micromapSize` returned by this command supports any build with a VkMicromapBuildInfoEXT structure subject to the following properties:

- The build command is a host build command, and `buildType` is `VK_ACCELERATION_STRUCTURE_BUILD_TYPE_HOST_KHR` or `VK_ACCELERATION_STRUCTURE_BUILD_TYPE_HOST_OR_DEVICE_KHR`
- The build command is a device build command, and `buildType` is `VK_ACCELERATION_STRUCTURE_BUILD_TYPE_DEVICE_KHR` or `VK_ACCELERATION_STRUCTURE_BUILD_TYPE_HOST_OR_DEVICE_KHR`
- For VkMicromapBuildInfoEXT:

- Its `type`, and `flags` members are equal to `pBuildInfo->type` and `pBuildInfo->flags`, respectively.
- The sum of usage information in either `pUsageCounts` or `ppUsageCounts` is equal to the sum of usage information in either `pBuildInfo->pUsageCounts` or `pBuildInfo->ppUsageCounts`.

Similarly, the `buildScratchSize` value will support any build command specifying the `VK_BUILD_MICROMAP_MODE_BUILD_EXT` `mode` under the above conditions.

### Valid Usage

- VUID-vkGetMicromapBuildSizesEXT-dstMicromap-09180
  VkMicromapBuildInfoEXT::`dstMicromap` **must** have been created from `device`
- VUID-vkGetMicromapBuildSizesEXT-micromap-07439
  The `micromap` feature **must** be enabled
- VUID-vkGetMicromapBuildSizesEXT-device-07440
  If `device` was created with multiple physical devices, then the `bufferDeviceAddressMultiDevice` feature **must** be enabled

### Valid Usage (Implicit)

- VUID-vkGetMicromapBuildSizesEXT-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkGetMicromapBuildSizesEXT-buildType-parameter
  `buildType` **must** be a valid VkAccelerationStructureBuildTypeKHR value
- VUID-vkGetMicromapBuildSizesEXT-pBuildInfo-parameter
  `pBuildInfo` **must** be a valid pointer to a valid VkMicromapBuildInfoEXT structure
- VUID-vkGetMicromapBuildSizesEXT-pSizeInfo-parameter
  `pSizeInfo` **must** be a valid pointer to a VkMicromapBuildSizesInfoEXT structure

The `VkMicromapBuildSizesInfoEXT` structure describes the required build sizes for a micromap and scratch buffers and is defined as:

```
// Provided by VK_EXT_opacity_micromap
typedef struct VkMicromapBuildSizesInfoEXT {
    VkStructureType    sType;
    const void*        pNext;
    VkDeviceSize       micromapSize;
    VkDeviceSize       buildScratchSize;
    VkBool32           discardable;
} VkMicromapBuildSizesInfoEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.

- `micromapSize` is the size in bytes required in a VkMicromapEXT for a build or update operation.
- `buildScratchSize` is the size in bytes required in a scratch buffer for a build operation.
- `discardable` indicates whether or not the micromap object may be destroyed after an acceleration structure build or update. A false value means that acceleration structures built with this micromap **may** contain references to the data contained therein, and the application **must** not destroy the micromap until ray traversal has concluded. A true value means that the information in the micromap will be copied by value into the acceleration structure, and the micromap **may** be destroyed after the acceleration structure build concludes.

### Valid Usage (Implicit)

- VUID-VkMicromapBuildSizesInfoEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_MICROMAP_BUILD_SIZES_INFO_EXT`
- VUID-VkMicromapBuildSizesInfoEXT-pNext-pNext
  `pNext` **must** be `NULL`

Values which **can** be set in VkMicromapCreateInfoEXT::`type` specifying the type of micromap, are:

```
// Provided by VK_EXT_opacity_micromap
typedef enum VkMicromapTypeEXT {
    VK_MICROMAP_TYPE_OPACITY_MICROMAP_EXT = 0,
#ifdef VK_ENABLE_BETA_EXTENSIONS
  // Provided by VK_NV_displacement_micromap
    VK_MICROMAP_TYPE_DISPLACEMENT_MICROMAP_NV = 1000397000,
#endif
} VkMicromapTypeEXT;
```

- `VK_MICROMAP_TYPE_OPACITY_MICROMAP_EXT` is a micromap containing data to control the opacity of a triangle.
- `VK_MICROMAP_TYPE_DISPLACEMENT_MICROMAP_NV` is a micromap containing data to control the displacement of subtriangles within a triangle.

Bits which **can** be set in VkMicromapCreateInfoEXT::`createFlags`, specifying additional creation parameters for micromaps, are:

```
// Provided by VK_EXT_opacity_micromap
typedef enum VkMicromapCreateFlagBitsEXT {
    VK_MICROMAP_CREATE_DEVICE_ADDRESS_CAPTURE_REPLAY_BIT_EXT = 0x00000001,
} VkMicromapCreateFlagBitsEXT;
```

- `VK_MICROMAP_CREATE_DEVICE_ADDRESS_CAPTURE_REPLAY_BIT_EXT` specifies that the micromap's address **can** be saved and reused on a subsequent run.

```
// Provided by VK_EXT_opacity_micromap
typedef VkFlags VkMicromapCreateFlagsEXT;
```

VkMicromapCreateFlagsEXT is a bitmask type for setting a mask of zero or more VkMicromapCreateFlagBitsEXT.

Bits which **can** be set in VkMicromapBuildInfoEXT::flags specifying additional parameters for micromap builds, are:

```
// Provided by VK_EXT_opacity_micromap
typedef enum VkBuildMicromapFlagBitsEXT {
    VK_BUILD_MICROMAP_PREFER_FAST_TRACE_BIT_EXT = 0x00000001,
    VK_BUILD_MICROMAP_PREFER_FAST_BUILD_BIT_EXT = 0x00000002,
    VK_BUILD_MICROMAP_ALLOW_COMPACTION_BIT_EXT = 0x00000004,
} VkBuildMicromapFlagBitsEXT;
```

- VK_BUILD_MICROMAP_PREFER_FAST_TRACE_BIT_EXT indicates that the given micromap build **should** prioritize trace performance over build time.
- VK_BUILD_MICROMAP_PREFER_FAST_BUILD_BIT_EXT indicates that the given micromap build **should** prioritize build time over trace performance.

```
// Provided by VK_EXT_opacity_micromap
typedef VkFlags VkBuildMicromapFlagsEXT;
```

VkBuildMicromapFlagsEXT is a bitmask type for setting a mask of zero or more VkBuildMicromapFlagBitsEXT.

To destroy a micromap, call:

```
// Provided by VK_EXT_opacity_micromap
void vkDestroyMicromapEXT(
    VkDevice                                    device,
    VkMicromapEXT                               micromap,
    const VkAllocationCallbacks*                pAllocator);
```

- device is the logical device that destroys the micromap.
- micromap is the micromap to destroy.
- pAllocator controls host memory allocation as described in the Memory Allocation chapter.

### Valid Usage

- VUID-vkDestroyMicromapEXT-micromap-07441
  All submitted commands that refer to micromap **must** have completed execution

- VUID-vkDestroyMicromapEXT-micromap-07442
  If `VkAllocationCallbacks` were provided when `micromap` was created, a compatible set of callbacks **must** be provided here
- VUID-vkDestroyMicromapEXT-micromap-07443
  If no `VkAllocationCallbacks` were provided when `micromap` was created, `pAllocator` **must** be `NULL`

### Valid Usage (Implicit)

- VUID-vkDestroyMicromapEXT-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkDestroyMicromapEXT-micromap-parameter
  If `micromap` is not VK_NULL_HANDLE, `micromap` **must** be a valid VkMicromapEXT handle
- VUID-vkDestroyMicromapEXT-pAllocator-parameter
  If `pAllocator` is not `NULL`, `pAllocator` **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkDestroyMicromapEXT-micromap-parent
  If `micromap` is a valid handle, it **must** have been created, allocated, or retrieved from `device`

### Host Synchronization

- Host access to `micromap` **must** be externally synchronized

## 12.8. Resource Memory Association

Resources are initially created as *virtual allocations* with no backing memory. Device memory is allocated separately (see Device Memory) and then associated with the resource. This association is done differently for sparse and non-sparse resources.

Resources created with any of the sparse creation flags are considered sparse resources. Resources created without these flags are non-sparse. The details on resource memory association for sparse resources is described in Sparse Resources.

Non-sparse resources **must** be bound completely and contiguously to a single `VkDeviceMemory` object before the resource is passed as a parameter to any of the following operations:

- creating image or buffer views
- updating descriptor sets
- recording commands in a command buffer

Once bound, the memory binding is immutable for the lifetime of the resource.

In a logical device representing more than one physical device, buffer and image resources exist on

all physical devices but **can** be bound to memory differently on each. Each such replicated resource is an *instance* of the resource. For sparse resources, each instance **can** be bound to memory arbitrarily differently. For non-sparse resources, each instance **can** either be bound to the local or a peer instance of the memory, or for images **can** be bound to rectangular regions from the local and/or peer instances. When a resource is used in a descriptor set, each physical device interprets the descriptor according to its own instance's binding to memory.

| | |
|---|---|
| **NOTE** | There are no new copy commands to transfer data between physical devices. Instead, an application **can** create a resource with a peer mapping and use it as the source or destination of a transfer command executed by a single physical device to copy the data from one physical device to another. |

To determine the memory requirements for a buffer resource, call:

```
// Provided by VK_VERSION_1_0
void vkGetBufferMemoryRequirements(
    VkDevice                                    device,
    VkBuffer                                    buffer,
    VkMemoryRequirements*                       pMemoryRequirements);
```

- `device` is the logical device that owns the buffer.
- `buffer` is the buffer to query.
- `pMemoryRequirements` is a pointer to a VkMemoryRequirements structure in which the memory requirements of the buffer object are returned.

### Valid Usage (Implicit)

- VUID-vkGetBufferMemoryRequirements-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkGetBufferMemoryRequirements-buffer-parameter
  `buffer` **must** be a valid VkBuffer handle
- VUID-vkGetBufferMemoryRequirements-pMemoryRequirements-parameter
  `pMemoryRequirements` **must** be a valid pointer to a VkMemoryRequirements structure
- VUID-vkGetBufferMemoryRequirements-buffer-parent
  `buffer` **must** have been created, allocated, or retrieved from `device`

To determine the memory requirements for an image resource which is not created with the `VK_IMAGE_CREATE_DISJOINT_BIT` flag set, call:

```
// Provided by VK_VERSION_1_0
void vkGetImageMemoryRequirements(
    VkDevice                                    device,
    VkImage                                     image,
    VkMemoryRequirements*                       pMemoryRequirements);
```

- `device` is the logical device that owns the image.
- `image` is the image to query.
- `pMemoryRequirements` is a pointer to a VkMemoryRequirements structure in which the memory requirements of the image object are returned.

## Valid Usage

- VUID-vkGetImageMemoryRequirements-image-01588
  `image` **must** not have been created with the `VK_IMAGE_CREATE_DISJOINT_BIT` flag set
- VUID-vkGetImageMemoryRequirements-image-04004
  If `image` was created with the `VK_EXTERNAL_MEMORY_HANDLE_TYPE_ANDROID_HARDWARE_BUFFER_BIT_ANDROID` external memory handle type, then `image` **must** be bound to memory
- VUID-vkGetImageMemoryRequirements-image-08960
  If `image` was created with the `VK_EXTERNAL_MEMORY_HANDLE_TYPE_SCREEN_BUFFER_BIT_QNX` external memory handle type, then `image` **must** be bound to memory

## Valid Usage (Implicit)

- VUID-vkGetImageMemoryRequirements-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkGetImageMemoryRequirements-image-parameter
  `image` **must** be a valid VkImage handle
- VUID-vkGetImageMemoryRequirements-pMemoryRequirements-parameter
  `pMemoryRequirements` **must** be a valid pointer to a VkMemoryRequirements structure
- VUID-vkGetImageMemoryRequirements-image-parent
  `image` **must** have been created, allocated, or retrieved from `device`

The `VkMemoryRequirements` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkMemoryRequirements {
    VkDeviceSize    size;
    VkDeviceSize    alignment;
    uint32_t        memoryTypeBits;
} VkMemoryRequirements;
```

- `size` is the size, in bytes, of the memory allocation **required** for the resource.
- `alignment` is the alignment, in bytes, of the offset within the allocation **required** for the resource.
- `memoryTypeBits` is a bitmask and contains one bit set for every supported memory type for the resource. Bit `i` is set if and only if the memory type `i` in the `VkPhysicalDeviceMemoryProperties` structure for the physical device is supported for the resource.

The precise size of images that will be bound to external Android hardware buffer memory is unknown until the memory has been imported or allocated, so applications **must** not call vkGetImageMemoryRequirements or vkGetImageMemoryRequirements2 with such a VkImage before it has been bound to memory. For this reason, applications also **must** not call vkGetDeviceImageMemoryRequirementsKHR with a VkImageCreateInfo describing an external Android hardware buffer. When importing Android hardware buffer memory, the `allocationSize` **can** be determined by calling vkGetAndroidHardwareBufferPropertiesANDROID. When allocating new memory for a VkImage that **can** be exported to an Android hardware buffer, the memory's `allocationSize` **must** be zero; the actual size will be determined by the dedicated image's parameters. After the memory has been allocated, the amount of space allocated from the memory's heap **can** be obtained by getting the image's memory requirements or by calling vkGetAndroidHardwareBufferPropertiesANDROID with the Android hardware buffer exported from the memory.

When allocating new memory for a VkBuffer that **can** be exported to an Android hardware buffer an application **may** still call vkGetBufferMemoryRequirements or vkGetBufferMemoryRequirements2 with VkBuffer before it has been bound to memory.

If the resource being queried was created with the `VK_EXTERNAL_MEMORY_HANDLE_TYPE_D3D11_TEXTURE_BIT`, `VK_EXTERNAL_MEMORY_HANDLE_TYPE_D3D11_TEXTURE_KMT_BIT`, or `VK_EXTERNAL_MEMORY_HANDLE_TYPE_D3D12_RESOURCE_BIT` external memory handle type, the value of `size` has no meaning and **should** be ignored.

The implementation guarantees certain properties about the memory requirements returned by vkGetBufferMemoryRequirements2, vkGetImageMemoryRequirements2, vkGetDeviceBufferMemoryRequirementsKHR, vkGetDeviceImageMemoryRequirementsKHR, vkGetBufferMemoryRequirements and vkGetImageMemoryRequirements:

- The `memoryTypeBits` member always contains at least one bit set.
- If `buffer` is a `VkBuffer` not created with the `VK_BUFFER_CREATE_SPARSE_BINDING_BIT` or `VK_BUFFER_CREATE_PROTECTED_BIT` bits set, or if `image` is a linear image that was not created with the `VK_IMAGE_CREATE_PROTECTED_BIT` bit set, then the `memoryTypeBits` member always contains at

least one bit set corresponding to a VkMemoryType with a propertyFlags that has both the VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT bit and the VK_MEMORY_PROPERTY_HOST_COHERENT_BIT bit set. In other words, mappable coherent memory **can** always be attached to these objects.

- If buffer was created with VkExternalMemoryBufferCreateInfo::handleTypes set to 0 or image was created with VkExternalMemoryImageCreateInfo::handleTypes set to 0, the memoryTypeBits member always contains at least one bit set corresponding to a VkMemoryType with a propertyFlags that has the VK_MEMORY_PROPERTY_DEVICE_LOCAL_BIT bit set.
- The memoryTypeBits member is identical for all VkBuffer objects created with the same value for the flags and usage members in the VkBufferCreateInfo structure and the handleTypes member of the VkExternalMemoryBufferCreateInfo structure passed to vkCreateBuffer. Further, if usage1 and usage2 of type VkBufferUsageFlags are such that the bits set in usage2 are a subset of the bits set in usage1, and they have the same flags and VkExternalMemoryBufferCreateInfo::handleTypes, then the bits set in memoryTypeBits returned for usage1 **must** be a subset of the bits set in memoryTypeBits returned for usage2, for all values of flags.
- The alignment member is a power of two.
- The alignment member is identical for all VkBuffer objects created with the same combination of values for the usage and flags members in the VkBufferCreateInfo structure passed to vkCreateBuffer.
- If the maintenance4 feature is enabled, then the alignment member is identical for all VkImage objects created with the same combination of values for the flags, imageType, format, extent, mipLevels, arrayLayers, samples, tiling and usage members in the VkImageCreateInfo structure passed to vkCreateImage.
- The alignment member satisfies the buffer descriptor offset alignment requirements associated with the VkBuffer's usage:
  - If usage included VK_BUFFER_USAGE_UNIFORM_TEXEL_BUFFER_BIT or VK_BUFFER_USAGE_STORAGE_TEXEL_BUFFER_BIT, alignment **must** be an integer multiple of VkPhysicalDeviceLimits::minTexelBufferOffsetAlignment.
  - If usage included VK_BUFFER_USAGE_UNIFORM_BUFFER_BIT, alignment **must** be an integer multiple of VkPhysicalDeviceLimits::minUniformBufferOffsetAlignment.
  - If usage included VK_BUFFER_USAGE_STORAGE_BUFFER_BIT, alignment **must** be an integer multiple of VkPhysicalDeviceLimits::minStorageBufferOffsetAlignment.
- For images created with a color format, the memoryTypeBits member is identical for all VkImage objects created with the same combination of values for the tiling member, the VK_IMAGE_CREATE_SPARSE_BINDING_BIT bit and VK_IMAGE_CREATE_PROTECTED_BIT bit of the flags member, the VK_IMAGE_CREATE_SPLIT_INSTANCE_BIND_REGIONS_BIT bit of the flags member, the VK_IMAGE_USAGE_HOST_TRANSFER_BIT_EXT bit of the usage member if the VkPhysicalDeviceHostImageCopyPropertiesEXT::identicalMemoryTypeRequirements property is VK_FALSE, handleTypes member of VkExternalMemoryImageCreateInfo, and the VK_IMAGE_USAGE_TRANSIENT_ATTACHMENT_BIT of the usage member in the VkImageCreateInfo structure passed to vkCreateImage.
- For images created with a depth/stencil format, the memoryTypeBits member is identical for all VkImage objects created with the same combination of values for the format member, the tiling member, the VK_IMAGE_CREATE_SPARSE_BINDING_BIT bit and VK_IMAGE_CREATE_PROTECTED_BIT bit of

the flags member, the VK_IMAGE_CREATE_SPLIT_INSTANCE_BIND_REGIONS_BIT bit of the flags member, the VK_IMAGE_USAGE_HOST_TRANSFER_BIT_EXT bit of the usage member if the VkPhysicalDeviceHostImageCopyPropertiesEXT::identicalMemoryTypeRequirements property is VK_FALSE, handleTypes member of VkExternalMemoryImageCreateInfo, and the VK_IMAGE_USAGE_TRANSIENT_ATTACHMENT_BIT of the usage member in the VkImageCreateInfo structure passed to vkCreateImage.

- If the memory requirements are for a VkImage, the memoryTypeBits member **must** not refer to a VkMemoryType with a propertyFlags that has the VK_MEMORY_PROPERTY_LAZILY_ALLOCATED_BIT bit set if the image did not have VK_IMAGE_USAGE_TRANSIENT_ATTACHMENT_BIT bit set in the usage member of the VkImageCreateInfo structure passed to vkCreateImage.
- If the memory requirements are for a VkBuffer, the memoryTypeBits member **must** not refer to a VkMemoryType with a propertyFlags that has the VK_MEMORY_PROPERTY_LAZILY_ALLOCATED_BIT bit set.

| | |
|---|---|
| **NOTE** | The implication of this requirement is that lazily allocated memory is disallowed for buffers in all cases. |

- The size member is identical for all VkBuffer objects created with the same combination of creation parameters specified in VkBufferCreateInfo and its pNext chain.
- The size member is identical for all VkImage objects created with the same combination of creation parameters specified in VkImageCreateInfo and its pNext chain.

| | |
|---|---|
| **NOTE** | This, however, does not imply that they interpret the contents of the bound memory identically with each other. That additional guarantee, however, **can** be explicitly requested using VK_IMAGE_CREATE_ALIAS_BIT. |

- If the maintenance4 feature is enabled, these additional guarantees apply:
  - For a VkBuffer, the size memory requirement is never greater than that of another VkBuffer created with a greater or equal size specified in VkBufferCreateInfo, all other creation parameters being identical.
  - For a VkBuffer, the size memory requirement is never greater than the result of aligning VkBufferCreateInfo::size with the alignment memory requirement.
  - For a VkImage, the size memory requirement is never greater than that of another VkImage created with a greater or equal value in each of extent.width, extent.height, and extent.depth; all other creation parameters being identical.
  - The memory requirements returned by vkGetDeviceBufferMemoryRequirementsKHR are identical to those that would be returned by vkGetBufferMemoryRequirements2 if it were called with a VkBuffer created with the same VkBufferCreateInfo values.
  - The memory requirements returned by vkGetDeviceImageMemoryRequirementsKHR are identical to those that would be returned by vkGetImageMemoryRequirements2 if it were called with a VkImage created with the same VkImageCreateInfo values.

To determine the memory requirements for a buffer resource, call:

```
// Provided by VK_VERSION_1_1
void vkGetBufferMemoryRequirements2(
    VkDevice                                    device,
    const VkBufferMemoryRequirementsInfo2*      pInfo,
    VkMemoryRequirements2*                      pMemoryRequirements);
```

or the equivalent command

```
// Provided by VK_KHR_get_memory_requirements2
void vkGetBufferMemoryRequirements2KHR(
    VkDevice                                    device,
    const VkBufferMemoryRequirementsInfo2*      pInfo,
    VkMemoryRequirements2*                      pMemoryRequirements);
```

- `device` is the logical device that owns the buffer.
- `pInfo` is a pointer to a VkBufferMemoryRequirementsInfo2 structure containing parameters required for the memory requirements query.
- `pMemoryRequirements` is a pointer to a VkMemoryRequirements2 structure in which the memory requirements of the buffer object are returned.

### Valid Usage (Implicit)

- VUID-vkGetBufferMemoryRequirements2-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkGetBufferMemoryRequirements2-pInfo-parameter
  `pInfo` **must** be a valid pointer to a valid VkBufferMemoryRequirementsInfo2 structure
- VUID-vkGetBufferMemoryRequirements2-pMemoryRequirements-parameter
  `pMemoryRequirements` **must** be a valid pointer to a VkMemoryRequirements2 structure

To determine the memory requirements for a buffer resource without creating an object, call:

```
// Provided by VK_KHR_maintenance4
void vkGetDeviceBufferMemoryRequirementsKHR(
    VkDevice                                    device,
    const VkDeviceBufferMemoryRequirements*     pInfo,
    VkMemoryRequirements2*                      pMemoryRequirements);
```

- `device` is the logical device intended to own the buffer.
- `pInfo` is a pointer to a VkDeviceBufferMemoryRequirements structure containing parameters required for the memory requirements query.
- `pMemoryRequirements` is a pointer to a VkMemoryRequirements2 structure in which the memory requirements of the buffer object are returned.

### Valid Usage (Implicit)

- VUID-vkGetDeviceBufferMemoryRequirements-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkGetDeviceBufferMemoryRequirements-pInfo-parameter
  `pInfo` **must** be a valid pointer to a valid VkDeviceBufferMemoryRequirements structure
- VUID-vkGetDeviceBufferMemoryRequirements-pMemoryRequirements-parameter
  `pMemoryRequirements` **must** be a valid pointer to a VkMemoryRequirements2 structure

The `VkBufferMemoryRequirementsInfo2` structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkBufferMemoryRequirementsInfo2 {
    VkStructureType    sType;
    const void*        pNext;
    VkBuffer           buffer;
} VkBufferMemoryRequirementsInfo2;
```

or the equivalent

```
// Provided by VK_KHR_get_memory_requirements2
typedef VkBufferMemoryRequirementsInfo2 VkBufferMemoryRequirementsInfo2KHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `buffer` is the buffer to query.

### Valid Usage (Implicit)

- VUID-VkBufferMemoryRequirementsInfo2-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_BUFFER_MEMORY_REQUIREMENTS_INFO_2`
- VUID-VkBufferMemoryRequirementsInfo2-pNext-pNext
  `pNext` **must** be `NULL`
- VUID-VkBufferMemoryRequirementsInfo2-buffer-parameter
  `buffer` **must** be a valid VkBuffer handle

The `VkDeviceBufferMemoryRequirements` structure is defined as:

```
typedef struct VkDeviceBufferMemoryRequirements {
    VkStructureType              sType;
    const void*                  pNext;
    const VkBufferCreateInfo*    pCreateInfo;
} VkDeviceBufferMemoryRequirements;
```

or the equivalent

```
// Provided by VK_KHR_maintenance4
typedef VkDeviceBufferMemoryRequirements VkDeviceBufferMemoryRequirementsKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `pCreateInfo` is a pointer to a VkBufferCreateInfo structure containing parameters affecting creation of the buffer to query.

### Valid Usage (Implicit)

- VUID-VkDeviceBufferMemoryRequirements-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_DEVICE_BUFFER_MEMORY_REQUIREMENTS`
- VUID-VkDeviceBufferMemoryRequirements-pNext-pNext
  `pNext` **must** be `NULL`
- VUID-VkDeviceBufferMemoryRequirements-pCreateInfo-parameter
  `pCreateInfo` **must** be a valid pointer to a valid VkBufferCreateInfo structure

To determine the memory requirements for an image resource, call:

```
// Provided by VK_VERSION_1_1
void vkGetImageMemoryRequirements2(
    VkDevice                                    device,
    const VkImageMemoryRequirementsInfo2*       pInfo,
    VkMemoryRequirements2*                      pMemoryRequirements);
```

or the equivalent command

```
// Provided by VK_KHR_get_memory_requirements2
void vkGetImageMemoryRequirements2KHR(
    VkDevice                                    device,
    const VkImageMemoryRequirementsInfo2*       pInfo,
    VkMemoryRequirements2*                      pMemoryRequirements);
```

- `device` is the logical device that owns the image.

- pInfo is a pointer to a VkImageMemoryRequirementsInfo2 structure containing parameters required for the memory requirements query.
- pMemoryRequirements is a pointer to a VkMemoryRequirements2 structure in which the memory requirements of the image object are returned.

### Valid Usage (Implicit)

- VUID-vkGetImageMemoryRequirements2-device-parameter
  device **must** be a valid VkDevice handle
- VUID-vkGetImageMemoryRequirements2-pInfo-parameter
  pInfo **must** be a valid pointer to a valid VkImageMemoryRequirementsInfo2 structure
- VUID-vkGetImageMemoryRequirements2-pMemoryRequirements-parameter
  pMemoryRequirements **must** be a valid pointer to a VkMemoryRequirements2 structure

To determine the memory requirements for an image resource without creating an object, call:

```
// Provided by VK_KHR_maintenance4
void vkGetDeviceImageMemoryRequirementsKHR(
    VkDevice                                    device,
    const VkDeviceImageMemoryRequirements*      pInfo,
    VkMemoryRequirements2*                      pMemoryRequirements);
```

- device is the logical device intended to own the image.
- pInfo is a pointer to a VkDeviceImageMemoryRequirements structure containing parameters required for the memory requirements query.
- pMemoryRequirements is a pointer to a VkMemoryRequirements2 structure in which the memory requirements of the image object are returned.

### Valid Usage (Implicit)

- VUID-vkGetDeviceImageMemoryRequirements-device-parameter
  device **must** be a valid VkDevice handle
- VUID-vkGetDeviceImageMemoryRequirements-pInfo-parameter
  pInfo **must** be a valid pointer to a valid VkDeviceImageMemoryRequirements structure
- VUID-vkGetDeviceImageMemoryRequirements-pMemoryRequirements-parameter
  pMemoryRequirements **must** be a valid pointer to a VkMemoryRequirements2 structure

The VkImageMemoryRequirementsInfo2 structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkImageMemoryRequirementsInfo2 {
    VkStructureType    sType;
    const void*        pNext;
    VkImage            image;
} VkImageMemoryRequirementsInfo2;
```

or the equivalent

```
// Provided by VK_KHR_get_memory_requirements2
typedef VkImageMemoryRequirementsInfo2 VkImageMemoryRequirementsInfo2KHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `image` is the image to query.

## Valid Usage

- VUID-VkImageMemoryRequirementsInfo2-image-01589
  If `image` was created with a *multi-planar* format and the `VK_IMAGE_CREATE_DISJOINT_BIT` flag, there **must** be a VkImagePlaneMemoryRequirementsInfo included in the `pNext` chain of the VkImageMemoryRequirementsInfo2 structure
- VUID-VkImageMemoryRequirementsInfo2-image-02279
  If `image` was created with `VK_IMAGE_CREATE_DISJOINT_BIT` and with `VK_IMAGE_TILING_DRM_FORMAT_MODIFIER_EXT`, then there **must** be a VkImagePlaneMemoryRequirementsInfo included in the `pNext` chain of the VkImageMemoryRequirementsInfo2 structure
- VUID-VkImageMemoryRequirementsInfo2-image-01590
  If `image` was not created with the `VK_IMAGE_CREATE_DISJOINT_BIT` flag, there **must** not be a VkImagePlaneMemoryRequirementsInfo included in the `pNext` chain of the VkImageMemoryRequirementsInfo2 structure
- VUID-VkImageMemoryRequirementsInfo2-image-02280
  If `image` was created with a single-plane format and with any `tiling` other than `VK_IMAGE_TILING_DRM_FORMAT_MODIFIER_EXT`, then there **must** not be a VkImagePlaneMemoryRequirementsInfo included in the `pNext` chain of the VkImageMemoryRequirementsInfo2 structure
- VUID-VkImageMemoryRequirementsInfo2-image-01897
  If `image` was created with the `VK_EXTERNAL_MEMORY_HANDLE_TYPE_ANDROID_HARDWARE_BUFFER_BIT_ANDROID` external memory handle type, then `image` **must** be bound to memory
- VUID-VkImageMemoryRequirementsInfo2-image-08961
  If `image` was created with the `VK_EXTERNAL_MEMORY_HANDLE_TYPE_SCREEN_BUFFER_BIT_QNX` external memory handle type, then `image` **must** be bound to memory

### Valid Usage (Implicit)

- VUID-VkImageMemoryRequirementsInfo2-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_IMAGE_MEMORY_REQUIREMENTS_INFO_2`
- VUID-VkImageMemoryRequirementsInfo2-pNext-pNext
  `pNext` **must** be `NULL` or a pointer to a valid instance of VkImagePlaneMemoryRequirementsInfo
- VUID-VkImageMemoryRequirementsInfo2-sType-unique
  The `sType` value of each struct in the `pNext` chain **must** be unique
- VUID-VkImageMemoryRequirementsInfo2-image-parameter
  `image` **must** be a valid VkImage handle

The `VkDeviceImageMemoryRequirements` structure is defined as:

```
typedef struct VkDeviceImageMemoryRequirements {
    VkStructureType             sType;
    const void*                 pNext;
    const VkImageCreateInfo*    pCreateInfo;
    VkImageAspectFlagBits       planeAspect;
} VkDeviceImageMemoryRequirements;
```

or the equivalent

```
// Provided by VK_KHR_maintenance4
typedef VkDeviceImageMemoryRequirements VkDeviceImageMemoryRequirementsKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `pCreateInfo` is a pointer to a VkImageCreateInfo structure containing parameters affecting creation of the image to query.
- `planeAspect` is a VkImageAspectFlagBits value specifying the aspect corresponding to the image plane to query. This parameter is ignored unless `pCreateInfo->tiling` is `VK_IMAGE_TILING_DRM_FORMAT_MODIFIER_EXT`, or `pCreateInfo->flags` has `VK_IMAGE_CREATE_DISJOINT_BIT` set.

### Valid Usage

- VUID-VkDeviceImageMemoryRequirements-pCreateInfo-06416
  The `pCreateInfo->pNext` chain **must** not contain a VkImageSwapchainCreateInfoKHR structure
- VUID-VkDeviceImageMemoryRequirements-pCreateInfo-06776
  The `pCreateInfo->pNext` chain **must** not contain a

VkImageDrmFormatModifierExplicitCreateInfoEXT structure

- VUID-VkDeviceImageMemoryRequirements-pNext-06996
  Applications also **must** not call vkGetDeviceImageMemoryRequirementsKHR with a VkImageCreateInfo whose `pNext` chain includes a VkExternalFormatANDROID structure with non-zero `externalFormat`
- VUID-VkDeviceImageMemoryRequirements-pNext-08962
  Applications also **must** not call vkGetDeviceImageMemoryRequirementsKHR with a VkImageCreateInfo whose `pNext` chain includes a VkExternalFormatQNX structure with non-zero `externalFormat`
- VUID-VkDeviceImageMemoryRequirements-pCreateInfo-06417
  If `pCreateInfo->format` specifies a *multi-planar* format and `pCreateInfo->flags` has `VK_IMAGE_CREATE_DISJOINT_BIT` set then `planeAspect` **must** not be `VK_IMAGE_ASPECT_NONE_KHR`
- VUID-VkDeviceImageMemoryRequirements-pCreateInfo-06419
  If `pCreateInfo->flags` has `VK_IMAGE_CREATE_DISJOINT_BIT` set and if the `pCreateInfo->tiling` is `VK_IMAGE_TILING_LINEAR` or `VK_IMAGE_TILING_OPTIMAL`, then `planeAspect` **must** be a single valid multi-planar aspect mask bit
- VUID-VkDeviceImageMemoryRequirements-pCreateInfo-06420
  If `pCreateInfo->tiling` is `VK_IMAGE_TILING_DRM_FORMAT_MODIFIER_EXT`, then `planeAspect` **must** be a single valid *memory plane* for the image (that is, `aspectMask` **must** specify a plane index that is less than the VkDrmFormatModifierPropertiesEXT::`drmFormatModifierPlaneCount` associated with the image's `format` and VkImageDrmFormatModifierPropertiesEXT::`drmFormatModifier`)

## Valid Usage (Implicit)

- VUID-VkDeviceImageMemoryRequirements-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_DEVICE_IMAGE_MEMORY_REQUIREMENTS`
- VUID-VkDeviceImageMemoryRequirements-pNext-pNext
  `pNext` **must** be `NULL`
- VUID-VkDeviceImageMemoryRequirements-pCreateInfo-parameter
  `pCreateInfo` **must** be a valid pointer to a valid VkImageCreateInfo structure
- VUID-VkDeviceImageMemoryRequirements-planeAspect-parameter
  If `planeAspect` is not `0`, `planeAspect` **must** be a valid VkImageAspectFlagBits value

To determine the memory requirements for a plane of a disjoint image, add a `VkImagePlaneMemoryRequirementsInfo` structure to the `pNext` chain of the `VkImageMemoryRequirementsInfo2` structure.

The `VkImagePlaneMemoryRequirementsInfo` structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkImagePlaneMemoryRequirementsInfo {
    VkStructureType          sType;
    const void*              pNext;
    VkImageAspectFlagBits    planeAspect;
} VkImagePlaneMemoryRequirementsInfo;
```

or the equivalent

```
// Provided by VK_KHR_sampler_ycbcr_conversion
typedef VkImagePlaneMemoryRequirementsInfo VkImagePlaneMemoryRequirementsInfoKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `planeAspect` is a VkImageAspectFlagBits value specifying the aspect corresponding to the image plane to query.

### Valid Usage

- VUID-VkImagePlaneMemoryRequirementsInfo-planeAspect-02281
  If the image's `tiling` is `VK_IMAGE_TILING_LINEAR` or `VK_IMAGE_TILING_OPTIMAL`, then `planeAspect` **must** be a single valid multi-planar aspect mask bit
- VUID-VkImagePlaneMemoryRequirementsInfo-planeAspect-02282
  If the image's `tiling` is `VK_IMAGE_TILING_DRM_FORMAT_MODIFIER_EXT`, then `planeAspect` **must** be a single valid *memory plane* for the image (that is, `aspectMask` **must** specify a plane index that is less than the VkDrmFormatModifierPropertiesEXT::`drmFormatModifierPlaneCount` associated with the image's `format` and VkImageDrmFormatModifierPropertiesEXT::`drmFormatModifier`)

### Valid Usage (Implicit)

- VUID-VkImagePlaneMemoryRequirementsInfo-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_IMAGE_PLANE_MEMORY_REQUIREMENTS_INFO`
- VUID-VkImagePlaneMemoryRequirementsInfo-planeAspect-parameter
  `planeAspect` **must** be a valid VkImageAspectFlagBits value

The `VkMemoryRequirements2` structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkMemoryRequirements2 {
    VkStructureType         sType;
    void*                   pNext;
    VkMemoryRequirements    memoryRequirements;
} VkMemoryRequirements2;
```

or the equivalent

```
// Provided by VK_KHR_get_memory_requirements2, VK_NV_ray_tracing with
VK_KHR_get_memory_requirements2 or VK_VERSION_1_1
typedef VkMemoryRequirements2 VkMemoryRequirements2KHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `memoryRequirements` is a VkMemoryRequirements structure describing the memory requirements of the resource.

### Valid Usage (Implicit)

- VUID-VkMemoryRequirements2-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_MEMORY_REQUIREMENTS_2`
- VUID-VkMemoryRequirements2-pNext-pNext
  `pNext` **must** be `NULL` or a pointer to a valid instance of VkMemoryDedicatedRequirements
- VUID-VkMemoryRequirements2-sType-unique
  The `sType` value of each struct in the `pNext` chain **must** be unique

The `VkMemoryDedicatedRequirements` structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkMemoryDedicatedRequirements {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           prefersDedicatedAllocation;
    VkBool32           requiresDedicatedAllocation;
} VkMemoryDedicatedRequirements;
```

or the equivalent

```
// Provided by VK_KHR_dedicated_allocation
typedef VkMemoryDedicatedRequirements VkMemoryDedicatedRequirementsKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- prefersDedicatedAllocation specifies that the implementation would prefer a dedicated allocation for this resource. The application is still free to suballocate the resource but it **may** get better performance if a dedicated allocation is used.
- requiresDedicatedAllocation specifies that a dedicated allocation is required for this resource.

To determine the dedicated allocation requirements of a buffer or image resource, add a VkMemoryDedicatedRequirements structure to the pNext chain of the VkMemoryRequirements2 structure passed as the pMemoryRequirements parameter of vkGetBufferMemoryRequirements2 or vkGetImageMemoryRequirements2, respectively.

Constraints on the values returned for buffer resources are:

- requiresDedicatedAllocation **may** be VK_TRUE if the pNext chain of VkBufferCreateInfo for the call to vkCreateBuffer used to create the buffer being queried included a VkExternalMemoryBufferCreateInfo structure, and any of the handle types specified in VkExternalMemoryBufferCreateInfo::handleTypes requires dedicated allocation, as reported by vkGetPhysicalDeviceExternalBufferProperties in VkExternalBufferProperties::externalMemoryProperties.externalMemoryFeatures. Otherwise, requiresDedicatedAllocation will be VK_FALSE.
- When the implementation sets requiresDedicatedAllocation to VK_TRUE, it **must** also set prefersDedicatedAllocation to VK_TRUE.
- If VK_BUFFER_CREATE_SPARSE_BINDING_BIT was set in VkBufferCreateInfo::flags when buffer was created, then both prefersDedicatedAllocation and requiresDedicatedAllocation will be VK_FALSE.

Constraints on the values returned for image resources are:

- requiresDedicatedAllocation **may** be VK_TRUE if the pNext chain of VkImageCreateInfo for the call to vkCreateImage used to create the image being queried included a VkExternalMemoryImageCreateInfo structure, and any of the handle types specified in VkExternalMemoryImageCreateInfo::handleTypes requires dedicated allocation, as reported by vkGetPhysicalDeviceImageFormatProperties2 in VkExternalImageFormatProperties::externalMemoryProperties.externalMemoryFeatures.
- requiresDedicatedAllocation **may** be VK_TRUE if the image's tiling is VK_IMAGE_TILING_DRM_FORMAT_MODIFIER_EXT.
- requiresDedicatedAllocation will otherwise be VK_FALSE
- If VK_IMAGE_CREATE_SPARSE_BINDING_BIT was set in VkImageCreateInfo::flags when image was created, then both prefersDedicatedAllocation and requiresDedicatedAllocation will be VK_FALSE.

### Valid Usage (Implicit)

- VUID-VkMemoryDedicatedRequirements-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_MEMORY_DEDICATED_REQUIREMENTS

To attach memory to a buffer object, call:

```
// Provided by VK_VERSION_1_0
VkResult vkBindBufferMemory(
    VkDevice                                    device,
    VkBuffer                                    buffer,
    VkDeviceMemory                              memory,
    VkDeviceSize                                memoryOffset);
```

- `device` is the logical device that owns the buffer and memory.
- `buffer` is the buffer to be attached to memory.
- `memory` is a VkDeviceMemory object describing the device memory to attach.
- `memoryOffset` is the start offset of the region of `memory` which is to be bound to the buffer. The number of bytes returned in the `VkMemoryRequirements`::`size` member in `memory`, starting from `memoryOffset` bytes, will be bound to the specified buffer.

`vkBindBufferMemory` is equivalent to passing the same parameters through VkBindBufferMemoryInfo to vkBindBufferMemory2.

### Valid Usage

- VUID-vkBindBufferMemory-buffer-07459
  `buffer` **must** not have been bound to a memory object
- VUID-vkBindBufferMemory-buffer-01030
  `buffer` **must** not have been created with any sparse memory binding flags
- VUID-vkBindBufferMemory-memoryOffset-01031
  `memoryOffset` **must** be less than the size of `memory`
- VUID-vkBindBufferMemory-memory-01035
  `memory` **must** have been allocated using one of the memory types allowed in the `memoryTypeBits` member of the `VkMemoryRequirements` structure returned from a call to `vkGetBufferMemoryRequirements` with `buffer`
- VUID-vkBindBufferMemory-memoryOffset-01036
  `memoryOffset` **must** be an integer multiple of the `alignment` member of the `VkMemoryRequirements` structure returned from a call to `vkGetBufferMemoryRequirements` with `buffer`
- VUID-vkBindBufferMemory-size-01037
  The `size` member of the `VkMemoryRequirements` structure returned from a call to `vkGetBufferMemoryRequirements` with `buffer` **must** be less than or equal to the size of `memory` minus `memoryOffset`
- VUID-vkBindBufferMemory-buffer-01444
  If `buffer` requires a dedicated allocation (as reported by vkGetBufferMemoryRequirements2 in VkMemoryDedicatedRequirements::`requiresDedicatedAllocation` for `buffer`), `memory` **must** have been allocated with VkMemoryDedicatedAllocateInfo::`buffer` equal to `buffer`

- VUID-vkBindBufferMemory-memory-01508
  If the VkMemoryAllocateInfo provided when memory was allocated included a VkMemoryDedicatedAllocateInfo structure in its pNext chain, and VkMemoryDedicatedAllocateInfo::buffer was not VK_NULL_HANDLE, then buffer **must** equal VkMemoryDedicatedAllocateInfo::buffer, and memoryOffset **must** be zero
- VUID-vkBindBufferMemory-None-01898
  If buffer was created with the VK_BUFFER_CREATE_PROTECTED_BIT bit set, the buffer **must** be bound to a memory object allocated with a memory type that reports VK_MEMORY_PROPERTY_PROTECTED_BIT
- VUID-vkBindBufferMemory-None-01899
  If buffer was created with the VK_BUFFER_CREATE_PROTECTED_BIT bit not set, the buffer **must** not be bound to a memory object allocated with a memory type that reports VK_MEMORY_PROPERTY_PROTECTED_BIT
- VUID-vkBindBufferMemory-buffer-01038
  If buffer was created with VkDedicatedAllocationBufferCreateInfoNV::dedicatedAllocation equal to VK_TRUE, memory **must** have been allocated with VkDedicatedAllocationMemoryAllocateInfoNV::buffer equal to a buffer handle created with identical creation parameters to buffer and memoryOffset **must** be zero
- VUID-vkBindBufferMemory-apiVersion-07920
  If the VK_KHR_dedicated_allocation extension is not enabled, VkPhysicalDeviceProperties::apiVersion is less than Vulkan 1.1, and buffer was not created with VkDedicatedAllocationBufferCreateInfoNV::dedicatedAllocation equal to VK_TRUE, memory **must** not have been allocated dedicated for a specific buffer or image
- VUID-vkBindBufferMemory-memory-02726
  If the value of VkExportMemoryAllocateInfo::handleTypes used to allocate memory is not 0, it **must** include at least one of the handles set in VkExternalMemoryBufferCreateInfo::handleTypes when buffer was created
- VUID-vkBindBufferMemory-memory-02985
  If memory was allocated by a memory import operation, that is not VkImportAndroidHardwareBufferInfoANDROID with a non-NULL buffer value, the external handle type of the imported memory **must** also have been set in VkExternalMemoryBufferCreateInfo::handleTypes when buffer was created
- VUID-vkBindBufferMemory-memory-02986
  If memory was allocated with the VkImportAndroidHardwareBufferInfoANDROID memory import operation with a non-NULL buffer value, VK_EXTERNAL_MEMORY_HANDLE_TYPE_ANDROID_HARDWARE_BUFFER_BIT_ANDROID **must** also have been set in VkExternalMemoryBufferCreateInfo::handleTypes when buffer was created
- VUID-vkBindBufferMemory-bufferDeviceAddress-03339
  If the VkPhysicalDeviceBufferDeviceAddressFeatures::bufferDeviceAddress feature is enabled and buffer was created with the VK_BUFFER_USAGE_SHADER_DEVICE_ADDRESS_BIT bit set, memory **must** have been allocated with the VK_MEMORY_ALLOCATE_DEVICE_ADDRESS_BIT bit set
- VUID-vkBindBufferMemory-bufferDeviceAddressCaptureReplay-09200
  If the VkPhysicalDeviceBufferDeviceAddressFeatures::bufferDeviceAddressCaptureReplay

feature is enabled and `buffer` was created with the `VK_BUFFER_CREATE_DEVICE_ADDRESS_CAPTURE_REPLAY_BIT` bit set, `memory` **must** have been allocated with the `VK_MEMORY_ALLOCATE_DEVICE_ADDRESS_CAPTURE_REPLAY_BIT` bit set

- VUID-vkBindBufferMemory-buffer-06408
  If `buffer` was created with VkBufferCollectionBufferCreateInfoFUCHSIA chained to VkBufferCreateInfo::`pNext`, `memory` **must** be allocated with a VkImportMemoryBufferCollectionFUCHSIA chained to VkMemoryAllocateInfo::`pNext`
- VUID-vkBindBufferMemory-descriptorBufferCaptureReplay-08112
  If the `buffer` was created with the `VK_BUFFER_CREATE_DESCRIPTOR_BUFFER_CAPTURE_REPLAY_BIT_EXT` bit set, `memory` **must** have been allocated with the `VK_MEMORY_ALLOCATE_DEVICE_ADDRESS_BIT` bit set
- VUID-vkBindBufferMemory-buffer-09201
  If the `buffer` was created with the `VK_BUFFER_CREATE_DESCRIPTOR_BUFFER_CAPTURE_REPLAY_BIT_EXT` bit set, `memory` **must** have been allocated with the `VK_MEMORY_ALLOCATE_DEVICE_ADDRESS_CAPTURE_REPLAY_BIT` bit set

## Valid Usage (Implicit)

- VUID-vkBindBufferMemory-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkBindBufferMemory-buffer-parameter
  `buffer` **must** be a valid VkBuffer handle
- VUID-vkBindBufferMemory-memory-parameter
  `memory` **must** be a valid VkDeviceMemory handle
- VUID-vkBindBufferMemory-buffer-parent
  `buffer` **must** have been created, allocated, or retrieved from `device`
- VUID-vkBindBufferMemory-memory-parent
  `memory` **must** have been created, allocated, or retrieved from `device`

## Host Synchronization

- Host access to `buffer` **must** be externally synchronized

## Return Codes

### Success

- `VK_SUCCESS`

### Failure

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`

- VK_ERROR_INVALID_OPAQUE_CAPTURE_ADDRESS_KHR

To attach memory to buffer objects for one or more buffers at a time, call:

```
// Provided by VK_VERSION_1_1
VkResult vkBindBufferMemory2(
    VkDevice                                    device,
    uint32_t                                    bindInfoCount,
    const VkBindBufferMemoryInfo*               pBindInfos);
```

or the equivalent command

```
// Provided by VK_KHR_bind_memory2
VkResult vkBindBufferMemory2KHR(
    VkDevice                                    device,
    uint32_t                                    bindInfoCount,
    const VkBindBufferMemoryInfo*               pBindInfos);
```

- device is the logical device that owns the buffers and memory.
- bindInfoCount is the number of elements in pBindInfos.
- pBindInfos is a pointer to an array of bindInfoCount VkBindBufferMemoryInfo structures describing buffers and memory to bind.

On some implementations, it **may** be more efficient to batch memory bindings into a single command.

If the maintenance6 feature is enabled, this command **must** attempt to perform all of the memory binding operations described by pBindInfos, and **must** not early exit on the first failure.

If any of the memory binding operations described by pBindInfos fail, the VkResult returned by this command **must** be the return value of any one of the memory binding operations which did not return VK_SUCCESS.

**NOTE**

If the vkBindBufferMemory2 command failed, VkBindMemoryStatusKHR structures were not included in the pNext chains of each element of pBindInfos, and bindInfoCount was greater than one, then the buffers referenced by pBindInfos will be in an indeterminate state, and must not be used.

Applications should destroy these buffers.

### Valid Usage (Implicit)

- VUID-vkBindBufferMemory2-device-parameter
  device **must** be a valid VkDevice handle
- VUID-vkBindBufferMemory2-pBindInfos-parameter

pBindInfos **must** be a valid pointer to an array of bindInfoCount valid VkBindBufferMemoryInfo structures

- VUID-vkBindBufferMemory2-bindInfoCount-arraylength
  bindInfoCount **must** be greater than 0

## Return Codes

**Success**

- VK_SUCCESS

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY
- VK_ERROR_INVALID_OPAQUE_CAPTURE_ADDRESS_KHR

VkBindBufferMemoryInfo contains members corresponding to the parameters of vkBindBufferMemory.

The VkBindBufferMemoryInfo structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkBindBufferMemoryInfo {
    VkStructureType    sType;
    const void*        pNext;
    VkBuffer           buffer;
    VkDeviceMemory     memory;
    VkDeviceSize       memoryOffset;
} VkBindBufferMemoryInfo;
```

or the equivalent

```
// Provided by VK_KHR_bind_memory2
typedef VkBindBufferMemoryInfo VkBindBufferMemoryInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- buffer is the buffer to be attached to memory.
- memory is a VkDeviceMemory object describing the device memory to attach.
- memoryOffset is the start offset of the region of memory which is to be bound to the buffer. The number of bytes returned in the VkMemoryRequirements::size member in memory, starting from memoryOffset bytes, will be bound to the specified buffer.

## Valid Usage

- VUID-VkBindBufferMemoryInfo-buffer-07459
  `buffer` **must** not have been bound to a memory object
- VUID-VkBindBufferMemoryInfo-buffer-01030
  `buffer` **must** not have been created with any sparse memory binding flags
- VUID-VkBindBufferMemoryInfo-memoryOffset-01031
  `memoryOffset` **must** be less than the size of `memory`
- VUID-VkBindBufferMemoryInfo-memory-01035
  `memory` **must** have been allocated using one of the memory types allowed in the `memoryTypeBits` member of the `VkMemoryRequirements` structure returned from a call to `vkGetBufferMemoryRequirements` with `buffer`
- VUID-VkBindBufferMemoryInfo-memoryOffset-01036
  `memoryOffset` **must** be an integer multiple of the `alignment` member of the `VkMemoryRequirements` structure returned from a call to `vkGetBufferMemoryRequirements` with `buffer`
- VUID-VkBindBufferMemoryInfo-size-01037
  The `size` member of the `VkMemoryRequirements` structure returned from a call to `vkGetBufferMemoryRequirements` with `buffer` **must** be less than or equal to the size of `memory` minus `memoryOffset`
- VUID-VkBindBufferMemoryInfo-buffer-01444
  If `buffer` requires a dedicated allocation (as reported by vkGetBufferMemoryRequirements2 in VkMemoryDedicatedRequirements::`requiresDedicatedAllocation` for `buffer`), `memory` **must** have been allocated with VkMemoryDedicatedAllocateInfo::`buffer` equal to `buffer`
- VUID-VkBindBufferMemoryInfo-memory-01508
  If the `VkMemoryAllocateInfo` provided when `memory` was allocated included a VkMemoryDedicatedAllocateInfo structure in its `pNext` chain, and VkMemoryDedicatedAllocateInfo::`buffer` was not VK_NULL_HANDLE, then `buffer` **must** equal VkMemoryDedicatedAllocateInfo::`buffer`, and `memoryOffset` **must** be zero
- VUID-VkBindBufferMemoryInfo-None-01898
  If `buffer` was created with the `VK_BUFFER_CREATE_PROTECTED_BIT` bit set, the buffer **must** be bound to a memory object allocated with a memory type that reports `VK_MEMORY_PROPERTY_PROTECTED_BIT`
- VUID-VkBindBufferMemoryInfo-None-01899
  If `buffer` was created with the `VK_BUFFER_CREATE_PROTECTED_BIT` bit not set, the buffer **must** not be bound to a memory object allocated with a memory type that reports `VK_MEMORY_PROPERTY_PROTECTED_BIT`
- VUID-VkBindBufferMemoryInfo-buffer-01038
  If `buffer` was created with VkDedicatedAllocationBufferCreateInfoNV::`dedicatedAllocation` equal to `VK_TRUE`, `memory` **must** have been allocated with VkDedicatedAllocationMemoryAllocateInfoNV::`buffer` equal to a buffer handle created with identical creation parameters to `buffer` and `memoryOffset` **must** be zero

- VUID-VkBindBufferMemoryInfo-apiVersion-07920
  If the VK_KHR_dedicated_allocation extension is not enabled, VkPhysicalDeviceProperties::apiVersion is less than Vulkan 1.1, and buffer was not created with VkDedicatedAllocationBufferCreateInfoNV::dedicatedAllocation equal to VK_TRUE, memory **must** not have been allocated dedicated for a specific buffer or image
- VUID-VkBindBufferMemoryInfo-memory-02726
  If the value of VkExportMemoryAllocateInfo::handleTypes used to allocate memory is not 0, it **must** include at least one of the handles set in VkExternalMemoryBufferCreateInfo::handleTypes when buffer was created
- VUID-VkBindBufferMemoryInfo-memory-02985
  If memory was allocated by a memory import operation, that is not VkImportAndroidHardwareBufferInfoANDROID with a non-NULL buffer value, the external handle type of the imported memory **must** also have been set in VkExternalMemoryBufferCreateInfo::handleTypes when buffer was created
- VUID-VkBindBufferMemoryInfo-memory-02986
  If memory was allocated with the VkImportAndroidHardwareBufferInfoANDROID memory import operation with a non-NULL buffer value, VK_EXTERNAL_MEMORY_HANDLE_TYPE_ANDROID_HARDWARE_BUFFER_BIT_ANDROID **must** also have been set in VkExternalMemoryBufferCreateInfo::handleTypes when buffer was created
- VUID-VkBindBufferMemoryInfo-bufferDeviceAddress-03339
  If the VkPhysicalDeviceBufferDeviceAddressFeatures::bufferDeviceAddress feature is enabled and buffer was created with the VK_BUFFER_USAGE_SHADER_DEVICE_ADDRESS_BIT bit set, memory **must** have been allocated with the VK_MEMORY_ALLOCATE_DEVICE_ADDRESS_BIT bit set
- VUID-VkBindBufferMemoryInfo-bufferDeviceAddressCaptureReplay-09200
  If the VkPhysicalDeviceBufferDeviceAddressFeatures::bufferDeviceAddressCaptureReplay feature is enabled and buffer was created with the VK_BUFFER_CREATE_DEVICE_ADDRESS_CAPTURE_REPLAY_BIT bit set, memory **must** have been allocated with the VK_MEMORY_ALLOCATE_DEVICE_ADDRESS_CAPTURE_REPLAY_BIT bit set
- VUID-VkBindBufferMemoryInfo-buffer-06408
  If buffer was created with VkBufferCollectionBufferCreateInfoFUCHSIA chained to VkBufferCreateInfo::pNext, memory **must** be allocated with a VkImportMemoryBufferCollectionFUCHSIA chained to VkMemoryAllocateInfo::pNext
- VUID-VkBindBufferMemoryInfo-descriptorBufferCaptureReplay-08112
  If the buffer was created with the VK_BUFFER_CREATE_DESCRIPTOR_BUFFER_CAPTURE_REPLAY_BIT_EXT bit set, memory **must** have been allocated with the VK_MEMORY_ALLOCATE_DEVICE_ADDRESS_BIT bit set
- VUID-VkBindBufferMemoryInfo-buffer-09201
  If the buffer was created with the VK_BUFFER_CREATE_DESCRIPTOR_BUFFER_CAPTURE_REPLAY_BIT_EXT bit set, memory **must** have been allocated with the VK_MEMORY_ALLOCATE_DEVICE_ADDRESS_CAPTURE_REPLAY_BIT bit set
- VUID-VkBindBufferMemoryInfo-pNext-01605
  If the pNext chain includes a VkBindBufferMemoryDeviceGroupInfo structure, all instances of memory specified by VkBindBufferMemoryDeviceGroupInfo::pDeviceIndices

**must** have been allocated

### Valid Usage (Implicit)

- VUID-VkBindBufferMemoryInfo-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_BIND_BUFFER_MEMORY_INFO`
- VUID-VkBindBufferMemoryInfo-pNext-pNext
  Each `pNext` member of any structure (including this one) in the `pNext` chain **must** be either `NULL` or a pointer to a valid instance of VkBindBufferMemoryDeviceGroupInfo or VkBindMemoryStatusKHR
- VUID-VkBindBufferMemoryInfo-sType-unique
  The `sType` value of each struct in the `pNext` chain **must** be unique
- VUID-VkBindBufferMemoryInfo-buffer-parameter
  `buffer` **must** be a valid VkBuffer handle
- VUID-VkBindBufferMemoryInfo-memory-parameter
  `memory` **must** be a valid VkDeviceMemory handle
- VUID-VkBindBufferMemoryInfo-commonparent
  Both of `buffer`, and `memory` **must** have been created, allocated, or retrieved from the same VkDevice

The `VkBindBufferMemoryDeviceGroupInfo` structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkBindBufferMemoryDeviceGroupInfo {
    VkStructureType    sType;
    const void*        pNext;
    uint32_t           deviceIndexCount;
    const uint32_t*    pDeviceIndices;
} VkBindBufferMemoryDeviceGroupInfo;
```

or the equivalent

```
// Provided by VK_KHR_bind_memory2 with VK_KHR_device_group
typedef VkBindBufferMemoryDeviceGroupInfo VkBindBufferMemoryDeviceGroupInfoKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `deviceIndexCount` is the number of elements in `pDeviceIndices`.
- `pDeviceIndices` is a pointer to an array of device indices.

If the `pNext` chain of VkBindBufferMemoryInfo includes a `VkBindBufferMemoryDeviceGroupInfo` structure, then that structure determines how memory is bound to buffers across multiple devices

in a device group.

If deviceIndexCount is greater than zero, then on device index i the buffer is attached to the instance of memory on the physical device with device index pDeviceIndices[i].

If deviceIndexCount is zero and memory comes from a memory heap with the VK_MEMORY_HEAP_MULTI_INSTANCE_BIT bit set, then it is as if pDeviceIndices contains consecutive indices from zero to the number of physical devices in the logical device, minus one. In other words, by default each physical device attaches to its own instance of memory.

If deviceIndexCount is zero and memory comes from a memory heap without the VK_MEMORY_HEAP_MULTI_INSTANCE_BIT bit set, then it is as if pDeviceIndices contains an array of zeros. In other words, by default each physical device attaches to instance zero.

### Valid Usage

- VUID-VkBindBufferMemoryDeviceGroupInfo-deviceIndexCount-01606
  deviceIndexCount **must** either be zero or equal to the number of physical devices in the logical device
- VUID-VkBindBufferMemoryDeviceGroupInfo-pDeviceIndices-01607
  All elements of pDeviceIndices **must** be valid device indices

### Valid Usage (Implicit)

- VUID-VkBindBufferMemoryDeviceGroupInfo-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_BIND_BUFFER_MEMORY_DEVICE_GROUP_INFO
- VUID-VkBindBufferMemoryDeviceGroupInfo-pDeviceIndices-parameter
  If deviceIndexCount is not 0, pDeviceIndices **must** be a valid pointer to an array of deviceIndexCount uint32_t values

The VkBindMemoryStatusKHR structure is defined as:

```
// Provided by VK_KHR_maintenance6
typedef struct VkBindMemoryStatusKHR {
    VkStructureType    sType;
    const void*        pNext;
    VkResult*          pResult;
} VkBindMemoryStatusKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- pResult is a pointer to a VkResult value.

If the pNext chain of VkBindBufferMemoryInfo or VkBindImageMemoryInfo includes a VkBindMemoryStatusKHR structure, then the VkBindMemoryStatusKHR::pResult will be populated with a

value describing the result of the corresponding memory binding operation.

### Valid Usage (Implicit)

- VUID-VkBindMemoryStatusKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_BIND_MEMORY_STATUS_KHR`
- VUID-VkBindMemoryStatusKHR-pResult-parameter
  `pResult` **must** be a valid pointer to a VkResult value

To attach memory to a `VkImage` object created without the `VK_IMAGE_CREATE_DISJOINT_BIT` set, call:

```
// Provided by VK_VERSION_1_0
VkResult vkBindImageMemory(
    VkDevice                                    device,
    VkImage                                     image,
    VkDeviceMemory                              memory,
    VkDeviceSize                                memoryOffset);
```

- `device` is the logical device that owns the image and memory.
- `image` is the image.
- `memory` is the VkDeviceMemory object describing the device memory to attach.
- `memoryOffset` is the start offset of the region of `memory` which is to be bound to the image. The number of bytes returned in the `VkMemoryRequirements`::`size` member in `memory`, starting from `memoryOffset` bytes, will be bound to the specified image.

`vkBindImageMemory` is equivalent to passing the same parameters through VkBindImageMemoryInfo to vkBindImageMemory2.

### Valid Usage

- VUID-vkBindImageMemory-image-07460
  `image` **must** not have been bound to a memory object
- VUID-vkBindImageMemory-image-01045
  `image` **must** not have been created with any sparse memory binding flags
- VUID-vkBindImageMemory-memoryOffset-01046
  `memoryOffset` **must** be less than the size of `memory`
- VUID-vkBindImageMemory-image-01445
  If `image` requires a dedicated allocation (as reported by vkGetImageMemoryRequirements2 in VkMemoryDedicatedRequirements::`requiresDedicatedAllocation` for `image`), `memory` **must** have been created with VkMemoryDedicatedAllocateInfo::`image` equal to `image`
- VUID-vkBindImageMemory-memory-02628
  If the `dedicatedAllocationImageAliasing` feature is not enabled, and the

VkMemoryAllocateInfo provided when memory was allocated included a VkMemoryDedicatedAllocateInfo structure in its pNext chain, and VkMemoryDedicatedAllocateInfo::image was not VK_NULL_HANDLE, then image **must** equal VkMemoryDedicatedAllocateInfo::image and memoryOffset **must** be zero

- VUID-vkBindImageMemory-memory-02629
  If the dedicatedAllocationImageAliasing feature is enabled, and the VkMemoryAllocateInfo provided when memory was allocated included a VkMemoryDedicatedAllocateInfo structure in its pNext chain, and VkMemoryDedicatedAllocateInfo::image was not VK_NULL_HANDLE, then memoryOffset **must** be zero, and image **must** be either equal to VkMemoryDedicatedAllocateInfo::image or an image that was created using the same parameters in VkImageCreateInfo, with the exception that extent and arrayLayers **may** differ subject to the following restrictions: every dimension in the extent parameter of the image being bound **must** be equal to or smaller than the original image for which the allocation was created; and the arrayLayers parameter of the image being bound **must** be equal to or smaller than the original image for which the allocation was created

- VUID-vkBindImageMemory-None-01901
  If image was created with the VK_IMAGE_CREATE_PROTECTED_BIT bit set, the image **must** be bound to a memory object allocated with a memory type that reports VK_MEMORY_PROPERTY_PROTECTED_BIT

- VUID-vkBindImageMemory-None-01902
  If image was created with the VK_IMAGE_CREATE_PROTECTED_BIT bit not set, the image **must** not be bound to a memory object created with a memory type that reports VK_MEMORY_PROPERTY_PROTECTED_BIT

- VUID-vkBindImageMemory-image-01050
  If image was created with VkDedicatedAllocationImageCreateInfoNV::dedicatedAllocation equal to VK_TRUE, memory **must** have been created with VkDedicatedAllocationMemoryAllocateInfoNV::image equal to an image handle created with identical creation parameters to image and memoryOffset **must** be zero

- VUID-vkBindImageMemory-apiVersion-07921
  If the VK_KHR_dedicated_allocation extension is not enabled, VkPhysicalDeviceProperties::apiVersion is less than Vulkan 1.1, and image was not created with VkDedicatedAllocationImageCreateInfoNV::dedicatedAllocation equal to VK_TRUE, memory **must** not have been allocated dedicated for a specific buffer or image

- VUID-vkBindImageMemory-memory-02728
  If the value of VkExportMemoryAllocateInfo::handleTypes used to allocate memory is not 0, it **must** include at least one of the handles set in VkExternalMemoryImageCreateInfo::handleTypes when image was created

- VUID-vkBindImageMemory-memory-02989
  If memory was created by a memory import operation, that is not VkImportAndroidHardwareBufferInfoANDROID with a non-NULL buffer value, the external handle type of the imported memory **must** also have been set in VkExternalMemoryImageCreateInfo::handleTypes when image was created

- VUID-vkBindImageMemory-memory-02990
  If memory was created with the VkImportAndroidHardwareBufferInfoANDROID memory import operation with a non-NULL buffer value,

VK_EXTERNAL_MEMORY_HANDLE_TYPE_ANDROID_HARDWARE_BUFFER_BIT_ANDROID **must** also have been set in VkExternalMemoryImageCreateInfo::handleTypes when image was created

- VUID-vkBindImageMemory-descriptorBufferCaptureReplay-08113
  If the image was created with the VK_IMAGE_CREATE_DESCRIPTOR_BUFFER_CAPTURE_REPLAY_BIT_EXT bit set, memory **must** have been allocated with the VK_MEMORY_ALLOCATE_DEVICE_ADDRESS_BIT bit set
- VUID-vkBindImageMemory-image-09202
  If the image was created with the VK_IMAGE_CREATE_DESCRIPTOR_BUFFER_CAPTURE_REPLAY_BIT_EXT bit set, memory **must** have been allocated with the VK_MEMORY_ALLOCATE_DEVICE_ADDRESS_CAPTURE_REPLAY_BIT bit set
- VUID-vkBindImageMemory-image-01608
  image **must** not have been created with the VK_IMAGE_CREATE_DISJOINT_BIT set
- VUID-vkBindImageMemory-memory-01047
  memory **must** have been allocated using one of the memory types allowed in the memoryTypeBits member of the VkMemoryRequirements structure returned from a call to vkGetImageMemoryRequirements with image
- VUID-vkBindImageMemory-memoryOffset-01048
  memoryOffset **must** be an integer multiple of the alignment member of the VkMemoryRequirements structure returned from a call to vkGetImageMemoryRequirements with image
- VUID-vkBindImageMemory-size-01049
  The difference of the size of memory and memoryOffset **must** be greater than or equal to the size member of the VkMemoryRequirements structure returned from a call to vkGetImageMemoryRequirements with the same image
- VUID-vkBindImageMemory-image-06392
  If image was created with VkBufferCollectionImageCreateInfoFUCHSIA chained to VkImageCreateInfo::pNext, memory **must** be allocated with a VkImportMemoryBufferCollectionFUCHSIA chained to VkMemoryAllocateInfo::pNext

## Valid Usage (Implicit)

- VUID-vkBindImageMemory-device-parameter
  device **must** be a valid VkDevice handle
- VUID-vkBindImageMemory-image-parameter
  image **must** be a valid VkImage handle
- VUID-vkBindImageMemory-memory-parameter
  memory **must** be a valid VkDeviceMemory handle
- VUID-vkBindImageMemory-image-parent
  image **must** have been created, allocated, or retrieved from device
- VUID-vkBindImageMemory-memory-parent
  memory **must** have been created, allocated, or retrieved from device

### Host Synchronization

- Host access to `image` **must** be externally synchronized

### Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`

To attach memory to image objects for one or more images at a time, call:

```
// Provided by VK_VERSION_1_1
VkResult vkBindImageMemory2(
    VkDevice                                    device,
    uint32_t                                    bindInfoCount,
    const VkBindImageMemoryInfo*                pBindInfos);
```

or the equivalent command

```
// Provided by VK_KHR_bind_memory2
VkResult vkBindImageMemory2KHR(
    VkDevice                                    device,
    uint32_t                                    bindInfoCount,
    const VkBindImageMemoryInfo*                pBindInfos);
```

- `device` is the logical device that owns the images and memory.
- `bindInfoCount` is the number of elements in `pBindInfos`.
- `pBindInfos` is a pointer to an array of VkBindImageMemoryInfo structures, describing images and memory to bind.

On some implementations, it **may** be more efficient to batch memory bindings into a single command.

If the `maintenance6` feature is enabled, this command **must** attempt to perform all of the memory binding operations described by `pBindInfos`, and **must** not early exit on the first failure.

If any of the memory binding operations described by `pBindInfos` fail, the VkResult returned by this command **must** be the return value of any one of the memory binding operations which did not return `VK_SUCCESS`.

**NOTE**

If the `vkBindImageMemory2` command failed, VkBindMemoryStatusKHR structures were not included in the `pNext` chains of each element of `pBindInfos`, and `bindInfoCount` was greater than one, then the images referenced by `pBindInfos` will be in an indeterminate state, and must not be used.

Applications should destroy these images.

## Valid Usage

- VUID-vkBindImageMemory2-pBindInfos-02858
  If any VkBindImageMemoryInfo::`image` was created with `VK_IMAGE_CREATE_DISJOINT_BIT` then all planes of VkBindImageMemoryInfo::`image` **must** be bound individually in separate `pBindInfos`
- VUID-vkBindImageMemory2-pBindInfos-04006
  `pBindInfos` **must** not refer to the same image subresource more than once

## Valid Usage (Implicit)

- VUID-vkBindImageMemory2-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkBindImageMemory2-pBindInfos-parameter
  `pBindInfos` **must** be a valid pointer to an array of `bindInfoCount` valid VkBindImageMemoryInfo structures
- VUID-vkBindImageMemory2-bindInfoCount-arraylength
  `bindInfoCount` **must** be greater than `0`

## Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`

`VkBindImageMemoryInfo` contains members corresponding to the parameters of vkBindImageMemory.

The `VkBindImageMemoryInfo` structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkBindImageMemoryInfo {
    VkStructureType    sType;
    const void*        pNext;
    VkImage            image;
    VkDeviceMemory     memory;
    VkDeviceSize       memoryOffset;
} VkBindImageMemoryInfo;
```

or the equivalent

```
// Provided by VK_KHR_bind_memory2
typedef VkBindImageMemoryInfo VkBindImageMemoryInfoKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `image` is the image to be attached to memory.
- `memory` is a VkDeviceMemory object describing the device memory to attach.
- `memoryOffset` is the start offset of the region of `memory` which is to be bound to the image. The number of bytes returned in the `VkMemoryRequirements`::`size` member in `memory`, starting from `memoryOffset` bytes, will be bound to the specified image.

## Valid Usage

- VUID-VkBindImageMemoryInfo-image-07460
  `image` **must** not have been bound to a memory object
- VUID-VkBindImageMemoryInfo-image-01045
  `image` **must** not have been created with any sparse memory binding flags
- VUID-VkBindImageMemoryInfo-memoryOffset-01046
  `memoryOffset` **must** be less than the size of `memory`
- VUID-VkBindImageMemoryInfo-image-01445
  If `image` requires a dedicated allocation (as reported by vkGetImageMemoryRequirements2 in VkMemoryDedicatedRequirements::`requiresDedicatedAllocation` for `image`), `memory` **must** have been created with VkMemoryDedicatedAllocateInfo::`image` equal to `image`
- VUID-VkBindImageMemoryInfo-memory-02628
  If the `dedicatedAllocationImageAliasing` feature is not enabled, and the `VkMemoryAllocateInfo` provided when `memory` was allocated included a VkMemoryDedicatedAllocateInfo structure in its `pNext` chain, and VkMemoryDedicatedAllocateInfo::`image` was not VK_NULL_HANDLE, then `image` **must** equal VkMemoryDedicatedAllocateInfo::`image` and `memoryOffset` **must** be zero
- VUID-VkBindImageMemoryInfo-memory-02629

If the dedicatedAllocationImageAliasing feature is enabled, and the VkMemoryAllocateInfo provided when memory was allocated included a VkMemoryDedicatedAllocateInfo structure in its pNext chain, and VkMemoryDedicatedAllocateInfo::image was not VK_NULL_HANDLE, then memoryOffset **must** be zero, and image **must** be either equal to VkMemoryDedicatedAllocateInfo::image or an image that was created using the same parameters in VkImageCreateInfo, with the exception that extent and arrayLayers **may** differ subject to the following restrictions: every dimension in the extent parameter of the image being bound **must** be equal to or smaller than the original image for which the allocation was created; and the arrayLayers parameter of the image being bound **must** be equal to or smaller than the original image for which the allocation was created

- VUID-VkBindImageMemoryInfo-None-01901
  If image was created with the VK_IMAGE_CREATE_PROTECTED_BIT bit set, the image **must** be bound to a memory object allocated with a memory type that reports VK_MEMORY_PROPERTY_PROTECTED_BIT
- VUID-VkBindImageMemoryInfo-None-01902
  If image was created with the VK_IMAGE_CREATE_PROTECTED_BIT bit not set, the image **must** not be bound to a memory object created with a memory type that reports VK_MEMORY_PROPERTY_PROTECTED_BIT
- VUID-VkBindImageMemoryInfo-image-01050
  If image was created with VkDedicatedAllocationImageCreateInfoNV::dedicatedAllocation equal to VK_TRUE, memory **must** have been created with VkDedicatedAllocationMemoryAllocateInfoNV::image equal to an image handle created with identical creation parameters to image and memoryOffset **must** be zero
- VUID-VkBindImageMemoryInfo-apiVersion-07921
  If the VK_KHR_dedicated_allocation extension is not enabled, VkPhysicalDeviceProperties::apiVersion is less than Vulkan 1.1, and image was not created with VkDedicatedAllocationImageCreateInfoNV::dedicatedAllocation equal to VK_TRUE, memory **must** not have been allocated dedicated for a specific buffer or image
- VUID-VkBindImageMemoryInfo-memory-02728
  If the value of VkExportMemoryAllocateInfo::handleTypes used to allocate memory is not 0, it **must** include at least one of the handles set in VkExternalMemoryImageCreateInfo::handleTypes when image was created
- VUID-VkBindImageMemoryInfo-memory-02989
  If memory was created by a memory import operation, that is not VkImportAndroidHardwareBufferInfoANDROID with a non-NULL buffer value, the external handle type of the imported memory **must** also have been set in VkExternalMemoryImageCreateInfo::handleTypes when image was created
- VUID-VkBindImageMemoryInfo-memory-02990
  If memory was created with the VkImportAndroidHardwareBufferInfoANDROID memory import operation with a non-NULL buffer value, VK_EXTERNAL_MEMORY_HANDLE_TYPE_ANDROID_HARDWARE_BUFFER_BIT_ANDROID **must** also have been set in VkExternalMemoryImageCreateInfo::handleTypes when image was created
- VUID-VkBindImageMemoryInfo-descriptorBufferCaptureReplay-08113
  If the image was created with the VK_IMAGE_CREATE_DESCRIPTOR_BUFFER_CAPTURE_REPLAY_BIT_EXT bit set, memory **must** have been

allocated with the VK_MEMORY_ALLOCATE_DEVICE_ADDRESS_BIT bit set

- VUID-VkBindImageMemoryInfo-image-09202
  If the image was created with the VK_IMAGE_CREATE_DESCRIPTOR_BUFFER_CAPTURE_REPLAY_BIT_EXT bit set, memory **must** have been allocated with the VK_MEMORY_ALLOCATE_DEVICE_ADDRESS_CAPTURE_REPLAY_BIT bit set
- VUID-VkBindImageMemoryInfo-pNext-01615
  If the pNext chain does not include a VkBindImagePlaneMemoryInfo structure, memory **must** have been allocated using one of the memory types allowed in the memoryTypeBits member of the VkMemoryRequirements structure returned from a call to vkGetImageMemoryRequirements2 with image
- VUID-VkBindImageMemoryInfo-pNext-01616
  If the pNext chain does not include a VkBindImagePlaneMemoryInfo structure, memoryOffset **must** be an integer multiple of the alignment member of the VkMemoryRequirements structure returned from a call to vkGetImageMemoryRequirements2 with image
- VUID-VkBindImageMemoryInfo-pNext-01617
  If the pNext chain does not include a VkBindImagePlaneMemoryInfo structure, the difference of the size of memory and memoryOffset **must** be greater than or equal to the size member of the VkMemoryRequirements structure returned from a call to vkGetImageMemoryRequirements2 with the same image
- VUID-VkBindImageMemoryInfo-pNext-01618
  If the pNext chain includes a VkBindImagePlaneMemoryInfo structure, image **must** have been created with the VK_IMAGE_CREATE_DISJOINT_BIT bit set
- VUID-VkBindImageMemoryInfo-image-07736
  If image was created with the VK_IMAGE_CREATE_DISJOINT_BIT bit set, then the pNext chain **must** include a VkBindImagePlaneMemoryInfo structure
- VUID-VkBindImageMemoryInfo-pNext-01619
  If the pNext chain includes a VkBindImagePlaneMemoryInfo structure, memory **must** have been allocated using one of the memory types allowed in the memoryTypeBits member of the VkMemoryRequirements structure returned from a call to vkGetImageMemoryRequirements2 with image and where VkBindImagePlaneMemoryInfo::planeAspect corresponds to the VkImagePlaneMemoryRequirementsInfo::planeAspect in the VkImageMemoryRequirementsInfo2 structure's pNext chain
- VUID-VkBindImageMemoryInfo-pNext-01620
  If the pNext chain includes a VkBindImagePlaneMemoryInfo structure, memoryOffset **must** be an integer multiple of the alignment member of the VkMemoryRequirements structure returned from a call to vkGetImageMemoryRequirements2 with image and where VkBindImagePlaneMemoryInfo::planeAspect corresponds to the VkImagePlaneMemoryRequirementsInfo::planeAspect in the VkImageMemoryRequirementsInfo2 structure's pNext chain
- VUID-VkBindImageMemoryInfo-pNext-01621
  If the pNext chain includes a VkBindImagePlaneMemoryInfo structure, the difference of the size of memory and memoryOffset **must** be greater than or equal to the size member of

the VkMemoryRequirements structure returned from a call to vkGetImageMemoryRequirements2 with the same image and where VkBindImagePlaneMemoryInfo::planeAspect corresponds to the VkImagePlaneMemoryRequirementsInfo::planeAspect in the VkImageMemoryRequirementsInfo2 structure's pNext chain

- VUID-VkBindImageMemoryInfo-pNext-01626
  If the pNext chain includes a VkBindImageMemoryDeviceGroupInfo structure, all instances of memory specified by VkBindImageMemoryDeviceGroupInfo::pDeviceIndices **must** have been allocated
- VUID-VkBindImageMemoryInfo-pNext-01627
  If the pNext chain includes a VkBindImageMemoryDeviceGroupInfo structure, and VkBindImageMemoryDeviceGroupInfo::splitInstanceBindRegionCount is not zero, then image **must** have been created with the VK_IMAGE_CREATE_SPLIT_INSTANCE_BIND_REGIONS_BIT bit set
- VUID-VkBindImageMemoryInfo-pNext-01628
  If the pNext chain includes a VkBindImageMemoryDeviceGroupInfo structure, all elements of VkBindImageMemoryDeviceGroupInfo::pSplitInstanceBindRegions **must** be valid rectangles contained within the dimensions of image
- VUID-VkBindImageMemoryInfo-pNext-01629
  If the pNext chain includes a VkBindImageMemoryDeviceGroupInfo structure, the union of the areas of all elements of VkBindImageMemoryDeviceGroupInfo::pSplitInstanceBindRegions that correspond to the same instance of image **must** cover the entire image
- VUID-VkBindImageMemoryInfo-image-01630
  If image was created with a valid swapchain handle in VkImageSwapchainCreateInfoKHR::swapchain, then the pNext chain **must** include a VkBindImageMemorySwapchainInfoKHR structure containing the same swapchain handle
- VUID-VkBindImageMemoryInfo-pNext-01631
  If the pNext chain includes a VkBindImageMemorySwapchainInfoKHR structure, memory **must** be VK_NULL_HANDLE
- VUID-VkBindImageMemoryInfo-pNext-01632
  If the pNext chain does not include a VkBindImageMemorySwapchainInfoKHR structure, memory **must** be a valid VkDeviceMemory handle

## Valid Usage (Implicit)

- VUID-VkBindImageMemoryInfo-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_BIND_IMAGE_MEMORY_INFO
- VUID-VkBindImageMemoryInfo-pNext-pNext
  Each pNext member of any structure (including this one) in the pNext chain **must** be either NULL or a pointer to a valid instance of VkBindImageMemoryDeviceGroupInfo, VkBindImageMemorySwapchainInfoKHR, VkBindImagePlaneMemoryInfo, or

VkBindMemoryStatusKHR

- VUID-VkBindImageMemoryInfo-sType-unique
  The `sType` value of each struct in the `pNext` chain **must** be unique
- VUID-VkBindImageMemoryInfo-image-parameter
  `image` **must** be a valid VkImage handle
- VUID-VkBindImageMemoryInfo-commonparent
  Both of `image`, and `memory` that are valid handles of non-ignored parameters **must** have been created, allocated, or retrieved from the same VkDevice

The `VkBindImageMemoryDeviceGroupInfo` structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkBindImageMemoryDeviceGroupInfo {
    VkStructureType    sType;
    const void*        pNext;
    uint32_t           deviceIndexCount;
    const uint32_t*    pDeviceIndices;
    uint32_t           splitInstanceBindRegionCount;
    const VkRect2D*    pSplitInstanceBindRegions;
} VkBindImageMemoryDeviceGroupInfo;
```

or the equivalent

```
// Provided by VK_KHR_bind_memory2 with VK_KHR_device_group
typedef VkBindImageMemoryDeviceGroupInfo VkBindImageMemoryDeviceGroupInfoKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `deviceIndexCount` is the number of elements in `pDeviceIndices`.
- `pDeviceIndices` is a pointer to an array of device indices.
- `splitInstanceBindRegionCount` is the number of elements in `pSplitInstanceBindRegions`.
- `pSplitInstanceBindRegions` is a pointer to an array of VkRect2D structures describing which regions of the image are attached to each instance of memory.

If the `pNext` chain of VkBindImageMemoryInfo includes a `VkBindImageMemoryDeviceGroupInfo` structure, then that structure determines how memory is bound to images across multiple devices in a device group.

If `deviceIndexCount` is greater than zero, then on device index i `image` is attached to the instance of the memory on the physical device with device index `pDeviceIndices`[i].

Let N be the number of physical devices in the logical device. If `splitInstanceBindRegionCount` is greater than zero, then `pSplitInstanceBindRegions` is a pointer to an array of $N^2$ rectangles, where the image region specified by the rectangle at element i*N+j in resource instance i is bound to the

memory instance j. The blocks of the memory that are bound to each sparse image block region use an offset in memory, relative to `memoryOffset`, computed as if the whole image was being bound to a contiguous range of memory. In other words, horizontally adjacent image blocks use consecutive blocks of memory, vertically adjacent image blocks are separated by the number of bytes per block multiplied by the width in blocks of `image`, and the block at (0,0) corresponds to memory starting at `memoryOffset`.

If `splitInstanceBindRegionCount` and `deviceIndexCount` are zero and the memory comes from a memory heap with the `VK_MEMORY_HEAP_MULTI_INSTANCE_BIT` bit set, then it is as if `pDeviceIndices` contains consecutive indices from zero to the number of physical devices in the logical device, minus one. In other words, by default each physical device attaches to its own instance of the memory.

If `splitInstanceBindRegionCount` and `deviceIndexCount` are zero and the memory comes from a memory heap without the `VK_MEMORY_HEAP_MULTI_INSTANCE_BIT` bit set, then it is as if `pDeviceIndices` contains an array of zeros. In other words, by default each physical device attaches to instance zero.

## Valid Usage

- VUID-VkBindImageMemoryDeviceGroupInfo-deviceIndexCount-01633
  At least one of `deviceIndexCount` and `splitInstanceBindRegionCount` **must** be zero
- VUID-VkBindImageMemoryDeviceGroupInfo-deviceIndexCount-01634
  `deviceIndexCount` **must** either be zero or equal to the number of physical devices in the logical device
- VUID-VkBindImageMemoryDeviceGroupInfo-pDeviceIndices-01635
  All elements of `pDeviceIndices` **must** be valid device indices
- VUID-VkBindImageMemoryDeviceGroupInfo-splitInstanceBindRegionCount-01636
  `splitInstanceBindRegionCount` **must** either be zero or equal to the number of physical devices in the logical device squared
- VUID-VkBindImageMemoryDeviceGroupInfo-pSplitInstanceBindRegions-01637
  Elements of `pSplitInstanceBindRegions` that correspond to the same instance of an image **must** not overlap
- VUID-VkBindImageMemoryDeviceGroupInfo-offset-01638
  The `offset.x` member of any element of `pSplitInstanceBindRegions` **must** be a multiple of the sparse image block width (`VkSparseImageFormatProperties`::`imageGranularity.width`) of all non-metadata aspects of the image
- VUID-VkBindImageMemoryDeviceGroupInfo-offset-01639
  The `offset.y` member of any element of `pSplitInstanceBindRegions` **must** be a multiple of the sparse image block height (`VkSparseImageFormatProperties`::`imageGranularity.height`) of all non-metadata aspects of the image
- VUID-VkBindImageMemoryDeviceGroupInfo-extent-01640
  The `extent.width` member of any element of `pSplitInstanceBindRegions` **must** either be a multiple of the sparse image block width of all non-metadata aspects of the image, or else `extent.width` + `offset.x` **must** equal the width of the image subresource
- VUID-VkBindImageMemoryDeviceGroupInfo-extent-01641

The extent.height member of any element of pSplitInstanceBindRegions **must** either be a multiple of the sparse image block height of all non-metadata aspects of the image, or else extent.height + offset.y **must** equal the height of the image subresource

### Valid Usage (Implicit)

- VUID-VkBindImageMemoryDeviceGroupInfo-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_BIND_IMAGE_MEMORY_DEVICE_GROUP_INFO
- VUID-VkBindImageMemoryDeviceGroupInfo-pDeviceIndices-parameter
  If deviceIndexCount is not 0, pDeviceIndices **must** be a valid pointer to an array of deviceIndexCount uint32_t values
- VUID-VkBindImageMemoryDeviceGroupInfo-pSplitInstanceBindRegions-parameter
  If splitInstanceBindRegionCount is not 0, pSplitInstanceBindRegions **must** be a valid pointer to an array of splitInstanceBindRegionCount VkRect2D structures

If the pNext chain of VkBindImageMemoryInfo includes a VkBindImageMemorySwapchainInfoKHR structure, then that structure includes a swapchain handle and image index indicating that the image will be bound to memory from that swapchain.

The VkBindImageMemorySwapchainInfoKHR structure is defined as:

```
// Provided by VK_VERSION_1_1 with VK_KHR_swapchain, VK_KHR_device_group with
VK_KHR_swapchain
typedef struct VkBindImageMemorySwapchainInfoKHR {
    VkStructureType    sType;
    const void*        pNext;
    VkSwapchainKHR     swapchain;
    uint32_t           imageIndex;
} VkBindImageMemorySwapchainInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- swapchain is VK_NULL_HANDLE or a swapchain handle.
- imageIndex is an image index within swapchain.

If swapchain is not NULL, the swapchain and imageIndex are used to determine the memory that the image is bound to, instead of memory and memoryOffset.

Memory **can** be bound to a swapchain and use the pDeviceIndices or pSplitInstanceBindRegions members of VkBindImageMemoryDeviceGroupInfo.

### Valid Usage

- VUID-VkBindImageMemorySwapchainInfoKHR-imageIndex-01644

imageIndex **must** be less than the number of images in swapchain

- VUID-VkBindImageMemorySwapchainInfoKHR-swapchain-07756
  If the swapchain has been created with VK_SWAPCHAIN_CREATE_DEFERRED_MEMORY_ALLOCATION_BIT_EXT, imageIndex **must** be one that has previously been returned by vkAcquireNextImageKHR or vkAcquireNextImage2KHR

### Valid Usage (Implicit)

- VUID-VkBindImageMemorySwapchainInfoKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_BIND_IMAGE_MEMORY_SWAPCHAIN_INFO_KHR
- VUID-VkBindImageMemorySwapchainInfoKHR-swapchain-parameter
  swapchain **must** be a valid VkSwapchainKHR handle

### Host Synchronization

- Host access to swapchain **must** be externally synchronized

In order to bind *planes* of a *disjoint image*, add a VkBindImagePlaneMemoryInfo structure to the pNext chain of VkBindImageMemoryInfo.

The VkBindImagePlaneMemoryInfo structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkBindImagePlaneMemoryInfo {
    VkStructureType          sType;
    const void*              pNext;
    VkImageAspectFlagBits    planeAspect;
} VkBindImagePlaneMemoryInfo;
```

or the equivalent

```
// Provided by VK_KHR_sampler_ycbcr_conversion
typedef VkBindImagePlaneMemoryInfo VkBindImagePlaneMemoryInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- planeAspect is a VkImageAspectFlagBits value specifying the aspect of the disjoint image plane to bind.

### Valid Usage

- VUID-VkBindImagePlaneMemoryInfo-planeAspect-02283

If the image's `tiling` is `VK_IMAGE_TILING_LINEAR` or `VK_IMAGE_TILING_OPTIMAL`, then `planeAspect` **must** be a single valid multi-planar aspect mask bit

- VUID-VkBindImagePlaneMemoryInfo-planeAspect-02284
  If the image's `tiling` is `VK_IMAGE_TILING_DRM_FORMAT_MODIFIER_EXT`, then `planeAspect` **must** be a single valid *memory plane* for the image (that is, `aspectMask` **must** specify a plane index that is less than the VkDrmFormatModifierPropertiesEXT::`drmFormatModifierPlaneCount` associated with the image's `format` and VkImageDrmFormatModifierPropertiesEXT::`drmFormatModifier`)

### Valid Usage (Implicit)

- VUID-VkBindImagePlaneMemoryInfo-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_BIND_IMAGE_PLANE_MEMORY_INFO`
- VUID-VkBindImagePlaneMemoryInfo-planeAspect-parameter
  `planeAspect` **must** be a valid VkImageAspectFlagBits value

*Buffer-Image Granularity*

The implementation-dependent limit `bufferImageGranularity` specifies a page-like granularity at which linear and non-linear resources **must** be placed in adjacent memory locations to avoid aliasing. Two resources which do not satisfy this granularity requirement are said to alias. `bufferImageGranularity` is specified in bytes, and **must** be a power of two. Implementations which do not impose a granularity restriction **may** report a `bufferImageGranularity` value of one.

**NOTE** Despite its name, `bufferImageGranularity` is really a granularity between "linear" and "non-linear" resources.

Given resourceA at the lower memory offset and resourceB at the higher memory offset in the same `VkDeviceMemory` object, where one resource is linear and the other is non-linear (as defined in the Glossary), and the following:

```
resourceA.end       = resourceA.memoryOffset + resourceA.size - 1
resourceA.endPage   = resourceA.end & ~(bufferImageGranularity-1)
resourceB.start     = resourceB.memoryOffset
resourceB.startPage = resourceB.start & ~(bufferImageGranularity-1)
```

The following property **must** hold:

```
resourceA.endPage < resourceB.startPage
```

That is, the end of the first resource (A) and the beginning of the second resource (B) **must** be on separate "pages" of size `bufferImageGranularity`. `bufferImageGranularity` **may** be different than the physical page size of the memory heap. This restriction is only needed when a linear resource and a non-linear resource are adjacent in memory and will be used simultaneously. The memory ranges

of adjacent resources **can** be closer than bufferImageGranularity, provided they meet the alignment requirement for the objects in question.

Sparse block size in bytes and sparse image and buffer memory alignments **must** all be multiples of the bufferImageGranularity. Therefore, memory bound to sparse resources naturally satisfies the bufferImageGranularity.

## 12.9. Resource Sharing Mode

Buffer and image objects are created with a *sharing mode* controlling how they **can** be accessed from queues. The supported sharing modes are:

```
// Provided by VK_VERSION_1_0
typedef enum VkSharingMode {
    VK_SHARING_MODE_EXCLUSIVE = 0,
    VK_SHARING_MODE_CONCURRENT = 1,
} VkSharingMode;
```

- VK_SHARING_MODE_EXCLUSIVE specifies that access to any range or image subresource of the object will be exclusive to a single queue family at a time.
- VK_SHARING_MODE_CONCURRENT specifies that concurrent access to any range or image subresource of the object from multiple queue families is supported.

**NOTE** VK_SHARING_MODE_CONCURRENT **may** result in lower performance access to the buffer or image than VK_SHARING_MODE_EXCLUSIVE.

Ranges of buffers and image subresources of image objects created using VK_SHARING_MODE_EXCLUSIVE **must** only be accessed by queues in the queue family that has *ownership* of the resource. Upon creation, such resources are not owned by any queue family; ownership is implicitly acquired upon first use within a queue. Once a resource using VK_SHARING_MODE_EXCLUSIVE is owned by some queue family, the application **must** perform a queue family ownership transfer to make the memory contents of a range or image subresource accessible to a different queue family.

**NOTE** Images still require a layout transition from VK_IMAGE_LAYOUT_UNDEFINED or VK_IMAGE_LAYOUT_PREINITIALIZED before being used on the first queue.

A queue family **can** take ownership of an image subresource or buffer range of a resource created with VK_SHARING_MODE_EXCLUSIVE, without an ownership transfer, in the same way as for a resource that was just created; however, taking ownership in this way has the effect that the contents of the image subresource or buffer range are undefined.

Ranges of buffers and image subresources of image objects created using VK_SHARING_MODE_CONCURRENT **must** only be accessed by queues from the queue families specified through the queueFamilyIndexCount and pQueueFamilyIndices members of the corresponding create info structures.

### 12.9.1. External Resource Sharing

Resources **should** only be accessed in the Vulkan instance that has exclusive ownership of their underlying memory. Only one Vulkan instance has exclusive ownership of a resource's underlying memory at a given time, regardless of whether the resource was created using `VK_SHARING_MODE_EXCLUSIVE` or `VK_SHARING_MODE_CONCURRENT`. Applications can transfer ownership of a resource's underlying memory only if the memory has been imported from or exported to another instance or external API using external memory handles. The semantics for transferring ownership outside of the instance are similar to those used for transferring ownership of `VK_SHARING_MODE_EXCLUSIVE` resources between queues, and is also accomplished using VkBufferMemoryBarrier or VkImageMemoryBarrier operations. To make the contents of the underlying memory accessible in the destination instance or API, applications **must**

1. Release exclusive ownership from the source instance or API.
2. Ensure the release operation has completed using semaphores or fences.
3. Acquire exclusive ownership in the destination instance or API

Unlike queue family ownership transfers, the destination instance or API is not specified explicitly when releasing ownership, nor is the source instance or API specified when acquiring ownership. Instead, the image or memory barrier's `dstQueueFamilyIndex` or `srcQueueFamilyIndex` parameters are set to the reserved queue family index `VK_QUEUE_FAMILY_EXTERNAL` or `VK_QUEUE_FAMILY_FOREIGN_EXT` to represent the external destination or source respectively.

Binding a resource to a memory object shared between multiple Vulkan instances or other APIs does not change the ownership of the underlying memory. The first entity to access the resource implicitly acquires ownership. An entity **can** also implicitly take ownership from another entity in the same way without an explicit ownership transfer. However, taking ownership in this way has the effect that the contents of the underlying memory are undefined.

Accessing a resource backed by memory that is owned by a particular instance or API has the same semantics as accessing a `VK_SHARING_MODE_EXCLUSIVE` resource, with one exception: Implementations **must** ensure layout transitions performed on one member of a set of identical subresources of identical images that alias the same range of an underlying memory object affect the layout of all the subresources in the set.

As a corollary, writes to any image subresources in such a set **must** not make the contents of memory used by other subresources in the set undefined. An application **can** define the content of a subresource of one image by performing device writes to an identical subresource of another image provided both images are bound to the same region of external memory. Applications **may** also add resources to such a set after the content of the existing set members has been defined without making the content undefined by creating a new image with the initial layout `VK_IMAGE_LAYOUT_UNDEFINED` and binding it to the same region of external memory as the existing images.

| | |
|---|---|
| **NOTE** | Because layout transitions apply to all identical images aliasing the same region of external memory, the actual layout of the memory backing a new image as well as an existing image with defined content will not be undefined. Such an image is not usable until it acquires ownership of its memory from the existing owner. |

> Therefore, the layout specified as part of this transition will be the true initial layout of the image. The undefined layout specified when creating it is a placeholder to simplify valid usage requirements.

## 12.10. Memory Aliasing

A range of a `VkDeviceMemory` allocation is *aliased* if it is bound to multiple resources simultaneously, as described below, via vkBindImageMemory, vkBindBufferMemory, vkBindAccelerationStructureMemoryNV, via sparse memory bindings, or by binding the memory to resources in multiple Vulkan instances or external APIs using external memory handle export and import mechanisms.

Consider two resources, $resource_A$ and $resource_B$, bound respectively to memory $range_A$ and $range_B$. Let $paddedRange_A$ and $paddedRange_B$ be, respectively, $range_A$ and $range_B$ aligned to `bufferImageGranularity`. If the resources are both linear or both non-linear (as defined in the Glossary), then the resources *alias* the memory in the intersection of $range_A$ and $range_B$. If one resource is linear and the other is non-linear, then the resources *alias* the memory in the intersection of $paddedRange_A$ and $paddedRange_B$.

Applications **can** alias memory, but use of multiple aliases is subject to several constraints.

> **NOTE** Memory aliasing **can** be useful to reduce the total device memory footprint of an application, if some large resources are used for disjoint periods of time.

When a non-linear, non-`VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT` image is bound to an aliased range, all image subresources of the image *overlap* the range. When a linear image is bound to an aliased range, the image subresources that (according to the image's advertised layout) include bytes from the aliased range overlap the range. When a `VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT` image has sparse image blocks bound to an aliased range, only image subresources including those sparse image blocks overlap the range, and when the memory bound to the image's mip tail overlaps an aliased range all image subresources in the mip tail overlap the range.

Buffers, and linear image subresources in either the `VK_IMAGE_LAYOUT_PREINITIALIZED` or `VK_IMAGE_LAYOUT_GENERAL` layouts, are *host-accessible subresources*. That is, the host has a well-defined addressing scheme to interpret the contents, and thus the layout of the data in memory **can** be consistently interpreted across aliases if each of those aliases is a host-accessible subresource. Non-linear images, and linear image subresources in other layouts, are not host-accessible.

If two aliases are both host-accessible, then they interpret the contents of the memory in consistent ways, and data written to one alias **can** be read by the other alias.

If two aliases are both images that were created with identical creation parameters, both were created with the `VK_IMAGE_CREATE_ALIAS_BIT` flag set, and both are bound identically to memory except for VkBindImageMemoryDeviceGroupInfo::`pDeviceIndices` and VkBindImageMemoryDeviceGroupInfo::`pSplitInstanceBindRegions`, then they interpret the contents of the memory in consistent ways, and data written to one alias **can** be read by the other alias.

Additionally, if an individual plane of a multi-planar image and a single-plane image alias the same

memory, then they also interpret the contents of the memory in consistent ways under the same conditions, but with the following modifications:

- Both **must** have been created with the `VK_IMAGE_CREATE_DISJOINT_BIT` flag.
- The single-plane image **must** have a VkFormat that is equivalent to that of the multi-planar image's individual plane.
- The single-plane image and the individual plane of the multi-planar image **must** be bound identically to memory except for VkBindImageMemoryDeviceGroupInfo::`pDeviceIndices` and VkBindImageMemoryDeviceGroupInfo::`pSplitInstanceBindRegions`.
- The `width` and `height` of the single-plane image are derived from the multi-planar image's dimensions in the manner listed for plane compatibility for the aliased plane.
- If either image's `tiling` is `VK_IMAGE_TILING_DRM_FORMAT_MODIFIER_EXT`, then both images **must** be linear.
- All other creation parameters **must** be identical

Aliases created by binding the same memory to resources in multiple Vulkan instances or external APIs using external memory handle export and import mechanisms interpret the contents of the memory in consistent ways, and data written to one alias **can** be read by the other alias.

Otherwise, the aliases interpret the contents of the memory differently, and writes via one alias make the contents of memory partially or completely undefined to the other alias. If the first alias is a host-accessible subresource, then the bytes affected are those written by the memory operations according to its addressing scheme. If the first alias is not host-accessible, then the bytes affected are those overlapped by the image subresources that were written. If the second alias is a host-accessible subresource, the affected bytes become undefined. If the second alias is not host-accessible, all sparse image blocks (for sparse partially-resident images) or all image subresources (for non-sparse image and fully resident sparse images) that overlap the affected bytes become undefined.

If any image subresources are made undefined due to writes to an alias, then each of those image subresources **must** have its layout transitioned from `VK_IMAGE_LAYOUT_UNDEFINED` to a valid layout before it is used, or from `VK_IMAGE_LAYOUT_PREINITIALIZED` if the memory has been written by the host. If any sparse blocks of a sparse image have been made undefined, then only the image subresources containing them **must** be transitioned.

Use of an overlapping range by two aliases **must** be separated by a memory dependency using the appropriate access types if at least one of those uses performs writes, whether the aliases interpret memory consistently or not. If buffer or image memory barriers are used, the scope of the barrier **must** contain the entire range and/or set of image subresources that overlap.

If two aliasing image views are used in the same framebuffer, then the render pass **must** declare the attachments using the `VK_ATTACHMENT_DESCRIPTION_MAY_ALIAS_BIT`, and follow the other rules listed in that section.

| | |
|---|---|
| **NOTE** | Memory recycled via an application suballocator (i.e. without freeing and reallocating the memory objects) is not substantially different from memory aliasing. However, a suballocator usually waits on a fence before recycling a region |

of memory, and signaling a fence involves sufficient implicit dependencies to satisfy all the above requirements.

### 12.10.1. Resource Memory Overlap

Applications **can** safely access a resource concurrently as long as the memory locations do not overlap as defined in Memory Location. This includes aliased resources if such aliasing is well-defined. It also includes access from different queues and/or queue families if such concurrent access is supported by the resource. Transfer commands only access memory locations specified by the range of the transfer command.

**NOTE** The intent is that buffers (or linear images) can be accessed concurrently, even when they share cache lines, but otherwise do not access the same memory range. The concept of a device cache line size is not exposed in the memory model.

## 12.11. Buffer Collections

Fuchsia's FIDL-based Sysmem service interoperates with Vulkan via the `VK_FUCHSIA_buffer_collection` extension.

A buffer collection is a set of one or more buffers which were allocated together as a group and which all have the same properties. These properties describe the buffers' internal representation, such as its dimensions and memory layout. This ensures that all of the buffers can be used interchangeably by tasks that require swapping among multiple buffers, such as double-buffered graphics rendering.

On Fuchsia, the Sysmem service uses buffer collections as a core construct in its design.

Buffer collections are represented by `VkBufferCollectionFUCHSIA` handles:

```
// Provided by VK_FUCHSIA_buffer_collection
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkBufferCollectionFUCHSIA)
```

### 12.11.1. Definitions

- FIDL - Fuchsia Interface Definition Language. The declarative language used to define FIDL interprocess communication interfaces on Fuchsia. FIDL files use the `fidl` extension. FIDL is also used to refer to the services defined by interfaces declared in the FIDL language
- Sysmem - The FIDL service that facilitates optimal buffer sharing and reuse on Fuchsia
- client - Any participant of the buffer collection e.g. the Vulkan application
- token - A `zx_handle_t` Zircon channel object that allows participation in the buffer collection

### 12.11.2. Platform Initialization for Buffer Collections

To initialize a buffer collection on Fuchsia:

- Connect to the Sysmem service to initialize a Sysmem allocator
- Create an initial buffer collection token using the Sysmem allocator
- Duplicate the token for each participant beyond the initiator
- See the Sysmem Overview and fuchsia.sysmem FIDL documentation on fuchsia.dev for more detailed information

### 12.11.3. Create the Buffer Collection

To create a VkBufferCollectionFUCHSIA for Vulkan to participate in the buffer collection:

```
// Provided by VK_FUCHSIA_buffer_collection
VkResult vkCreateBufferCollectionFUCHSIA(
    VkDevice                                    device,
    const VkBufferCollectionCreateInfoFUCHSIA*  pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkBufferCollectionFUCHSIA*                  pCollection);
```

- `device` is the logical device that creates the `VkBufferCollectionFUCHSIA`
- `pCreateInfo` is a pointer to a VkBufferCollectionCreateInfoFUCHSIA structure containing parameters affecting creation of the buffer collection
- `pAllocator` is a pointer to a VkAllocationCallbacks structure controlling host memory allocation as described in the Memory Allocation chapter
- `pBufferCollection` is a pointer to a VkBufferCollectionFUCHSIA handle in which the resulting buffer collection object is returned

#### Valid Usage (Implicit)

- VUID-vkCreateBufferCollectionFUCHSIA-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkCreateBufferCollectionFUCHSIA-pCreateInfo-parameter
  `pCreateInfo` **must** be a valid pointer to a valid VkBufferCollectionCreateInfoFUCHSIA structure
- VUID-vkCreateBufferCollectionFUCHSIA-pAllocator-parameter
  If `pAllocator` is not `NULL`, `pAllocator` **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkCreateBufferCollectionFUCHSIA-pCollection-parameter
  `pCollection` **must** be a valid pointer to a VkBufferCollectionFUCHSIA handle

#### Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_INVALID_EXTERNAL_HANDLE
- VK_ERROR_INITIALIZATION_FAILED

### Host Access

All functions referencing a VkBufferCollectionFUCHSIA **must** be externally synchronized with the exception of vkCreateBufferCollectionFUCHSIA.

The VkBufferCollectionCreateInfoFUCHSIA structure is defined as:

```
// Provided by VK_FUCHSIA_buffer_collection
typedef struct VkBufferCollectionCreateInfoFUCHSIA {
    VkStructureType    sType;
    const void*        pNext;
    zx_handle_t        collectionToken;
} VkBufferCollectionCreateInfoFUCHSIA;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure
- collectionToken is a zx_handle_t containing the Sysmem client's buffer collection token

### Valid Usage

- VUID-VkBufferCollectionCreateInfoFUCHSIA-collectionToken-06393
  collectionToken **must** be a valid zx_handle_t to a Zircon channel allocated from Sysmem (fuchsia.sysmem.Allocator/AllocateSharedCollection) with ZX_DEFAULT_CHANNEL_RIGHTS rights

### Valid Usage (Implicit)

- VUID-VkBufferCollectionCreateInfoFUCHSIA-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_BUFFER_COLLECTION_CREATE_INFO_FUCHSIA
- VUID-VkBufferCollectionCreateInfoFUCHSIA-pNext-pNext
  pNext **must** be NULL

## 12.11.4. Set the Constraints

Buffer collections can be established for VkImage allocations or VkBuffer allocations.

### Set Image-Based Buffer Collection Constraints

Setting the constraints on the buffer collection initiates the format negotiation and allocation of the buffer collection. To set the constraints on a VkImage buffer collection, call:

```
// Provided by VK_FUCHSIA_buffer_collection
VkResult vkSetBufferCollectionImageConstraintsFUCHSIA(
    VkDevice                                    device,
    VkBufferCollectionFUCHSIA                   collection,
    const VkImageConstraintsInfoFUCHSIA*        pImageConstraintsInfo);
```

- `device` is the logical device
- `collection` is the VkBufferCollectionFUCHSIA handle
- `pImageConstraintsInfo` is a pointer to a VkImageConstraintsInfoFUCHSIA structure

`vkSetBufferCollectionImageConstraintsFUCHSIA` **may** fail if `pImageConstraintsInfo->formatConstraintsCount` is larger than the implementation-defined limit. If that occurs, vkSetBufferCollectionImageConstraintsFUCHSIA will return `VK_ERROR_INITIALIZATION_FAILED`.

`vkSetBufferCollectionImageConstraintsFUCHSIA` **may** fail if the implementation does not support any of the formats described by the `pImageConstraintsInfo` structure. If that occurs, vkSetBufferCollectionImageConstraintsFUCHSIA will return `VK_ERROR_FORMAT_NOT_SUPPORTED`.

#### Valid Usage

- VUID-vkSetBufferCollectionImageConstraintsFUCHSIA-collection-06394
  `vkSetBufferCollectionImageConstraintsFUCHSIA` or `vkSetBufferCollectionBufferConstraintsFUCHSIA` **must** not have already been called on `collection`

#### Valid Usage (Implicit)

- VUID-vkSetBufferCollectionImageConstraintsFUCHSIA-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkSetBufferCollectionImageConstraintsFUCHSIA-collection-parameter
  `collection` **must** be a valid VkBufferCollectionFUCHSIA handle
- VUID-vkSetBufferCollectionImageConstraintsFUCHSIA-pImageConstraintsInfo-parameter
  `pImageConstraintsInfo` **must** be a valid pointer to a valid VkImageConstraintsInfoFUCHSIA structure
- VUID-vkSetBufferCollectionImageConstraintsFUCHSIA-collection-parent
  `collection` **must** have been created, allocated, or retrieved from `device`

## Return Codes

**Success**

- VK_SUCCESS

**Failure**

- VK_ERROR_INITIALIZATION_FAILED
- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_FORMAT_NOT_SUPPORTED

The VkImageConstraintsInfoFUCHSIA structure is defined as:

```
// Provided by VK_FUCHSIA_buffer_collection
typedef struct VkImageConstraintsInfoFUCHSIA {
    VkStructureType                                 sType;
    const void*                                     pNext;
    uint32_t                                        formatConstraintsCount;
    const VkImageFormatConstraintsInfoFUCHSIA*      pFormatConstraints;
    VkBufferCollectionConstraintsInfoFUCHSIA        bufferCollectionConstraints;
    VkImageConstraintsInfoFlagsFUCHSIA              flags;
} VkImageConstraints InfoFUCHSIA;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- formatConstraintsCount is the number of elements in pFormatConstraints.
- pFormatConstraints is a pointer to an array of VkImageFormatConstraintsInfoFUCHSIA structures of size formatConstraintsCount that is used to further constrain buffer collection format selection for image-based buffer collections.
- bufferCollectionConstraints is a VkBufferCollectionConstraintsInfoFUCHSIA structure used to supply parameters for the negotiation and allocation for buffer-based buffer collections.
- flags is a VkImageConstraintsInfoFlagBitsFUCHSIA value specifying hints about the type of memory Sysmem should allocate for the buffer collection.

## Valid Usage

- VUID-VkImageConstraintsInfoFUCHSIA-pFormatConstraints-06395
  All elements of pFormatConstraints **must** have at least one bit set in its VkImageFormatConstraintsInfoFUCHSIA::requiredFormatFeatures
- VUID-VkImageConstraintsInfoFUCHSIA-pFormatConstraints-06396
  If pFormatConstraints->imageCreateInfo->usage contains VK_IMAGE_USAGE_SAMPLED_BIT, then pFormatConstraints->requiredFormatFeatures **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_BIT
- VUID-VkImageConstraintsInfoFUCHSIA-pFormatConstraints-06397

If pFormatConstraints->imageCreateInfo->usage contains VK_IMAGE_USAGE_STORAGE_BIT, then pFormatConstraints->requiredFormatFeatures **must** contain VK_FORMAT_FEATURE_STORAGE_IMAGE_BIT

- VUID-VkImageConstraintsInfoFUCHSIA-pFormatConstraints-06398
  If pFormatConstraints->imageCreateInfo->usage contains VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT, then pFormatConstraints->requiredFormatFeatures **must** contain VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BIT
- VUID-VkImageConstraintsInfoFUCHSIA-pFormatConstraints-06399
  If pFormatConstraints->imageCreateInfo->usage contains VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT, then pFormatConstraints->requiredFormatFeatures **must** contain VK_FORMAT_FEATURE_DEPTH_STENCIL_ATTACHMENT_BIT
- VUID-VkImageConstraintsInfoFUCHSIA-pFormatConstraints-06400
  If pFormatConstraints->imageCreateInfo->usage contains VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT, then pFormatConstraints->requiredFormatFeatures **must** contain at least one of VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BIT or VK_FORMAT_FEATURE_DEPTH_STENCIL_ATTACHMENT_BIT
- VUID-VkImageConstraintsInfoFUCHSIA-attachmentFragmentShadingRate-06401
  If the attachmentFragmentShadingRate feature is enabled, and pFormatConstraints->imageCreateInfo->usage contains VK_IMAGE_USAGE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR, then pFormatConstraints->requiredFormatFeatures **must** contain VK_FORMAT_FEATURE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR

## Valid Usage (Implicit)

- VUID-VkImageConstraintsInfoFUCHSIA-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_IMAGE_CONSTRAINTS_INFO_FUCHSIA
- VUID-VkImageConstraintsInfoFUCHSIA-pNext-pNext
  pNext **must** be NULL
- VUID-VkImageConstraintsInfoFUCHSIA-pFormatConstraints-parameter
  pFormatConstraints **must** be a valid pointer to an array of formatConstraintsCount valid VkImageFormatConstraintsInfoFUCHSIA structures
- VUID-VkImageConstraintsInfoFUCHSIA-bufferCollectionConstraints-parameter
  bufferCollectionConstraints **must** be a valid VkBufferCollectionConstraintsInfoFUCHSIA structure
- VUID-VkImageConstraintsInfoFUCHSIA-flags-parameter
  flags **must** be a valid combination of VkImageConstraintsInfoFlagBitsFUCHSIA values
- VUID-VkImageConstraintsInfoFUCHSIA-formatConstraintsCount-arraylength
  formatConstraintsCount **must** be greater than 0

```
// Provided by VK_FUCHSIA_buffer_collection
typedef VkFlags VkImageConstraintsInfoFlagsFUCHSIA;
```

VkImageConstraintsInfoFlagsFUCHSIA is a bitmask type for setting a mask of zero or more VkImageConstraintsInfoFlagBitsFUCHSIA bits.

Bits which **can** be set in VkImageConstraintsInfoFlagBitsFUCHSIA::flags include:

```
// Provided by VK_FUCHSIA_buffer_collection
typedef enum VkImageConstraintsInfoFlagBitsFUCHSIA {
    VK_IMAGE_CONSTRAINTS_INFO_CPU_READ_RARELY_FUCHSIA = 0x00000001,
    VK_IMAGE_CONSTRAINTS_INFO_CPU_READ_OFTEN_FUCHSIA = 0x00000002,
    VK_IMAGE_CONSTRAINTS_INFO_CPU_WRITE_RARELY_FUCHSIA = 0x00000004,
    VK_IMAGE_CONSTRAINTS_INFO_CPU_WRITE_OFTEN_FUCHSIA = 0x00000008,
    VK_IMAGE_CONSTRAINTS_INFO_PROTECTED_OPTIONAL_FUCHSIA = 0x00000010,
} VkImageConstraintsInfoFlagBitsFUCHSIA;
```

General hints about the type of memory that should be allocated by Sysmem based on the expected usage of the images in the buffer collection include:

- VK_IMAGE_CONSTRAINTS_INFO_CPU_READ_RARELY_FUCHSIA
- VK_IMAGE_CONSTRAINTS_INFO_CPU_READ_OFTEN_FUCHSIA
- VK_IMAGE_CONSTRAINTS_INFO_CPU_WRITE_RARELY_FUCHSIA
- VK_IMAGE_CONSTRAINTS_INFO_CPU_WRITE_OFTEN_FUCHSIA

For protected memory:

- VK_IMAGE_CONSTRAINTS_INFO_PROTECTED_OPTIONAL_FUCHSIA specifies that protected memory is optional for the buffer collection.

Note that if all participants in the buffer collection (Vulkan or otherwise) specify that protected memory is optional, Sysmem will not allocate protected memory.

The VkImageFormatConstraintsInfoFUCHSIA structure is defined as:

```
// Provided by VK_FUCHSIA_buffer_collection
typedef struct VkImageFormatConstraintsInfoFUCHSIA {
    VkStructureType                         sType;
    const void*                             pNext;
    VkImageCreateInfo                       imageCreateInfo;
    VkFormatFeatureFlags                    requiredFormatFeatures;
    VkImageFormatConstraintsFlagsFUCHSIA    flags;
    uint64_t                                sysmemPixelFormat;
    uint32_t                                colorSpaceCount;
    const VkSysmemColorSpaceFUCHSIA*        pColorSpaces;
} VkImageFormatConstraintsInfoFUCHSIA;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure
- `imageCreateInfo` is the VkImageCreateInfo used to create a VkImage that is to use memory from the VkBufferCollectionFUCHSIA
- `requiredFormatFeatures` is a bitmask of `VkFormatFeatureFlagBits` specifying required features of the buffers in the buffer collection
- `flags` is reserved for future use
- `sysmemPixelFormat` is a `PixelFormatType` value from the `fuchsia.sysmem/image_formats.fidl` FIDL interface
- `colorSpaceCount` the element count of `pColorSpaces`
- `pColorSpaces` is a pointer to an array of VkSysmemColorSpaceFUCHSIA structs of size `colorSpaceCount`

### Valid Usage (Implicit)

- VUID-VkImageFormatConstraintsInfoFUCHSIA-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_IMAGE_FORMAT_CONSTRAINTS_INFO_FUCHSIA`
- VUID-VkImageFormatConstraintsInfoFUCHSIA-pNext-pNext
  `pNext` **must** be `NULL`
- VUID-VkImageFormatConstraintsInfoFUCHSIA-imageCreateInfo-parameter
  `imageCreateInfo` **must** be a valid VkImageCreateInfo structure
- VUID-VkImageFormatConstraintsInfoFUCHSIA-requiredFormatFeatures-parameter
  `requiredFormatFeatures` **must** be a valid combination of VkFormatFeatureFlagBits values
- VUID-VkImageFormatConstraintsInfoFUCHSIA-requiredFormatFeatures-requiredbitmask
  `requiredFormatFeatures` **must** not be `0`
- VUID-VkImageFormatConstraintsInfoFUCHSIA-flags-zerobitmask
  `flags` **must** be `0`
- VUID-VkImageFormatConstraintsInfoFUCHSIA-pColorSpaces-parameter
  `pColorSpaces` **must** be a valid pointer to an array of `colorSpaceCount` valid VkSysmemColorSpaceFUCHSIA structures
- VUID-VkImageFormatConstraintsInfoFUCHSIA-colorSpaceCount-arraylength
  `colorSpaceCount` **must** be greater than `0`

```
// Provided by VK_FUCHSIA_buffer_collection
typedef VkFlags VkImageFormatConstraintsFlagsFUCHSIA;
```

`VkImageFormatConstraintsFlagsFUCHSIA` is a bitmask type for setting a mask, but is currently reserved for future use.

The `VkBufferCollectionConstraintsInfoFUCHSIA` structure is defined as:

```
// Provided by VK_FUCHSIA_buffer_collection
typedef struct VkBufferCollectionConstraintsInfoFUCHSIA {
    VkStructureType    sType;
    const void*        pNext;
    uint32_t           minBufferCount;
    uint32_t           maxBufferCount;
    uint32_t           minBufferCountForCamping;
    uint32_t           minBufferCountForDedicatedSlack;
    uint32_t           minBufferCountForSharedSlack;
} VkBufferCollectionConstraintsInfoFUCHSIA;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure
- `minBufferCount` is the minimum number of buffers available in the collection
- `maxBufferCount` is the maximum number of buffers allowed in the collection
- `minBufferCountForCamping` is the per-participant minimum buffers for camping
- `minBufferCountForDedicatedSlack` is the per-participant minimum buffers for dedicated slack
- `minBufferCountForSharedSlack` is the per-participant minimum buffers for shared slack

Sysmem uses all buffer count parameters in combination to determine the number of buffers it will allocate. Sysmem defines buffer count constraints in `fuchsia.sysmem/constraints.fidl`.

*Camping* as referred to by `minBufferCountForCamping`, is the number of buffers that should be available for the participant that are not for transient use. This number of buffers is required for the participant to logically operate.

*Slack* as referred to by `minBufferCountForDedicatedSlack` and `minBufferCountForSharedSlack`, refers to the number of buffers desired by participants for optimal performance. `minBufferCountForDedicatedSlack` refers to the current participant. `minBufferCountForSharedSlack` refers to buffer slack for all participants in the collection.

### Valid Usage (Implicit)

- VUID-VkBufferCollectionConstraintsInfoFUCHSIA-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_BUFFER_COLLECTION_CONSTRAINTS_INFO_FUCHSIA`
- VUID-VkBufferCollectionConstraintsInfoFUCHSIA-pNext-pNext
  `pNext` **must** be `NULL`

The `VkSysmemColorSpaceFUCHSIA` structure is defined as:

```
// Provided by VK_FUCHSIA_buffer_collection
typedef struct VkSysmemColorSpaceFUCHSIA {
    VkStructureType    sType;
    const void*        pNext;
    uint32_t           colorSpace;
} VkSysmemColorSpaceFUCHSIA;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure
- `colorSpace` value of the Sysmem `ColorSpaceType`

### Valid Usage

- VUID-VkSysmemColorSpaceFUCHSIA-colorSpace-06402
  `colorSpace` **must** be a `ColorSpaceType` as defined in `fuchsia.sysmem/image_formats.fidl`

### Valid Usage (Implicit)

- VUID-VkSysmemColorSpaceFUCHSIA-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_SYSMEM_COLOR_SPACE_FUCHSIA`
- VUID-VkSysmemColorSpaceFUCHSIA-pNext-pNext
  `pNext` **must** be `NULL`

#### Set Buffer-Based Buffer Collection Constraints

To set the constraints on a VkBuffer buffer collection, call:

```
// Provided by VK_FUCHSIA_buffer_collection
VkResult vkSetBufferCollectionBufferConstraintsFUCHSIA(
    VkDevice                                    device,
    VkBufferCollectionFUCHSIA                   collection,
    const VkBufferConstraintsInfoFUCHSIA*       pBufferConstraintsInfo);
```

- `device` is the logical device
- `collection` is the VkBufferCollectionFUCHSIA handle
- `pBufferConstraintsInfo` is a pointer to a VkBufferConstraintsInfoFUCHSIA structure

`vkSetBufferCollectionBufferConstraintsFUCHSIA` **may** fail if the implementation does not support the constraints specified in the `bufferCollectionConstraints` structure. If that occurs, vkSetBufferCollectionBufferConstraintsFUCHSIA will return `VK_ERROR_FORMAT_NOT_SUPPORTED`.

## Valid Usage

- VUID-vkSetBufferCollectionBufferConstraintsFUCHSIA-collection-06403 `vkSetBufferCollectionImageConstraintsFUCHSIA` or `vkSetBufferCollectionBufferConstraintsFUCHSIA` **must** not have already been called on `collection`

## Valid Usage (Implicit)

- VUID-vkSetBufferCollectionBufferConstraintsFUCHSIA-device-parameter `device` **must** be a valid VkDevice handle
- VUID-vkSetBufferCollectionBufferConstraintsFUCHSIA-collection-parameter `collection` **must** be a valid VkBufferCollectionFUCHSIA handle
- VUID-vkSetBufferCollectionBufferConstraintsFUCHSIA-pBufferConstraintsInfo-parameter `pBufferConstraintsInfo` **must** be a valid pointer to a valid VkBufferConstraintsInfoFUCHSIA structure
- VUID-vkSetBufferCollectionBufferConstraintsFUCHSIA-collection-parent `collection` **must** have been created, allocated, or retrieved from `device`

## Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- `VK_ERROR_INITIALIZATION_FAILED`
- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_FORMAT_NOT_SUPPORTED`

The `VkBufferConstraintsInfoFUCHSIA` structure is defined as:

```
// Provided by VK_FUCHSIA_buffer_collection
typedef struct VkBufferConstraintsInfoFUCHSIA {
    VkStructureType                             sType;
    const void*                                 pNext;
    VkBufferCreateInfo                          createInfo;
    VkFormatFeatureFlags                        requiredFormatFeatures;
    VkBufferCollectionConstraintsInfoFUCHSIA    bufferCollectionConstraints;
} VkBufferConstraintsInfoFUCHSIA;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure

- `pBufferCreateInfo` a pointer to a VkBufferCreateInfo struct describing the buffer attributes for the buffer collection
- `requiredFormatFeatures` bitmask of `VkFormatFeatureFlagBits` required features of the buffers in the buffer collection
- `bufferCollectionConstraints` is used to supply parameters for the negotiation and allocation of the buffer collection

### Valid Usage

- VUID-VkBufferConstraintsInfoFUCHSIA-requiredFormatFeatures-06404
  The `requiredFormatFeatures` bitmask of `VkFormatFeatureFlagBits` **must** be chosen from among the buffer compatible format features listed in buffer compatible format features

### Valid Usage (Implicit)

- VUID-VkBufferConstraintsInfoFUCHSIA-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_BUFFER_CONSTRAINTS_INFO_FUCHSIA`
- VUID-VkBufferConstraintsInfoFUCHSIA-pNext-pNext
  `pNext` **must** be `NULL`
- VUID-VkBufferConstraintsInfoFUCHSIA-createInfo-parameter
  `createInfo` **must** be a valid VkBufferCreateInfo structure
- VUID-VkBufferConstraintsInfoFUCHSIA-requiredFormatFeatures-parameter
  `requiredFormatFeatures` **must** be a valid combination of VkFormatFeatureFlagBits values
- VUID-VkBufferConstraintsInfoFUCHSIA-bufferCollectionConstraints-parameter
  `bufferCollectionConstraints` **must** be a valid VkBufferCollectionConstraintsInfoFUCHSIA structure

## 12.11.5. Retrieve Buffer Collection Properties

After constraints have been set on the buffer collection by calling vkSetBufferCollectionImageConstraintsFUCHSIA or vkSetBufferCollectionBufferConstraintsFUCHSIA, call `vkGetBufferCollectionPropertiesFUCHSIA` to retrieve the negotiated and finalized properties of the buffer collection.

The call to `vkGetBufferCollectionPropertiesFUCHSIA` is synchronous. It waits for the Sysmem format negotiation and buffer collection allocation to complete before returning.

```
// Provided by VK_FUCHSIA_buffer_collection
VkResult vkGetBufferCollectionPropertiesFUCHSIA(
    VkDevice                                    device,
    VkBufferCollectionFUCHSIA                   collection,
    VkBufferCollectionPropertiesFUCHSIA*        pProperties);
```

- device is the logical device handle
- collection is the VkBufferCollectionFUCHSIA handle
- pProperties is a pointer to the retrieved VkBufferCollectionPropertiesFUCHSIA struct

For image-based buffer collections, upon calling vkGetBufferCollectionPropertiesFUCHSIA, Sysmem will choose an element of the VkImageConstraintsInfoFUCHSIA::pImageCreateInfos established by the preceding call to vkSetBufferCollectionImageConstraintsFUCHSIA. The index of the element chosen is stored in and can be retrieved from VkBufferCollectionPropertiesFUCHSIA::createInfoIndex.

For buffer-based buffer collections, a single VkBufferCreateInfo is specified as VkBufferConstraintsInfoFUCHSIA::createInfo. VkBufferCollectionPropertiesFUCHSIA::createInfoIndex will therefore always be zero.

vkGetBufferCollectionPropertiesFUCHSIA **may** fail if Sysmem is unable to resolve the constraints of all of the participants in the buffer collection. If that occurs, vkGetBufferCollectionPropertiesFUCHSIA will return VK_ERROR_INITIALIZATION_FAILED.

## Valid Usage

- VUID-vkGetBufferCollectionPropertiesFUCHSIA-None-06405
  Prior to calling vkGetBufferCollectionPropertiesFUCHSIA, the constraints on the buffer collection **must** have been set by either vkSetBufferCollectionImageConstraintsFUCHSIA or vkSetBufferCollectionBufferConstraintsFUCHSIA

## Valid Usage (Implicit)

- VUID-vkGetBufferCollectionPropertiesFUCHSIA-device-parameter
  device **must** be a valid VkDevice handle
- VUID-vkGetBufferCollectionPropertiesFUCHSIA-collection-parameter
  collection **must** be a valid VkBufferCollectionFUCHSIA handle
- VUID-vkGetBufferCollectionPropertiesFUCHSIA-pProperties-parameter
  pProperties **must** be a valid pointer to a VkBufferCollectionPropertiesFUCHSIA structure
- VUID-vkGetBufferCollectionPropertiesFUCHSIA-collection-parent
  collection **must** have been created, allocated, or retrieved from device

## Return Codes

**Success**

- VK_SUCCESS

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_INITIALIZATION_FAILED

The VkBufferCollectionPropertiesFUCHSIA structure is defined as:

```
// Provided by VK_FUCHSIA_buffer_collection
typedef struct VkBufferCollectionPropertiesFUCHSIA {
    VkStructureType                 sType;
    void*                           pNext;
    uint32_t                        memoryTypeBits;
    uint32_t                        bufferCount;
    uint32_t                        createInfoIndex;
    uint64_t                        sysmemPixelFormat;
    VkFormatFeatureFlags            formatFeatures;
    VkSysmemColorSpaceFUCHSIA       sysmemColorSpaceIndex;
    VkComponentMapping              samplerYcbcrConversionComponents;
    VkSamplerYcbcrModelConversion   suggestedYcbcrModel;
    VkSamplerYcbcrRange             suggestedYcbcrRange;
    VkChromaLocation                suggestedXChromaOffset;
    VkChromaLocation                suggestedYChromaOffset;
} VkBufferCollectionPropertiesFUCHSIA;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure
- memoryTypeBits is a bitmask containing one bit set for every memory type which the buffer collection can be imported as buffer collection
- bufferCount is the number of buffers in the collection
- createInfoIndex as described in Sysmem chosen create infos
- sysmemPixelFormat is the Sysmem PixelFormatType as defined in fuchsia.sysmem/image_formats.fidl
- formatFeatures is a bitmask of VkFormatFeatureFlagBits shared by the buffer collection
- sysmemColorSpaceIndex is a VkSysmemColorSpaceFUCHSIA struct specifying the color space
- samplerYcbcrConversionComponents is a VkComponentMapping struct specifying the component mapping
- suggestedYcbcrModel is a VkSamplerYcbcrModelConversion value specifying the suggested $Y'C_BC_R$ model
- suggestedYcbcrRange is a VkSamplerYcbcrRange value specifying the suggested $Y'C_BC_R$ range
- suggestedXChromaOffset is a VkChromaLocation value specifying the suggested X chroma offset
- suggestedYChromaOffset is a VkChromaLocation value specifying the suggested Y chroma offset

sysmemColorSpace is only set for image-based buffer collections where the constraints were specified using VkImageConstraintsInfoFUCHSIA in a call to vkSetBufferCollectionImageConstraintsFUCHSIA.

For image-based buffer collections, createInfoIndex will identify both the VkImageConstraintsInfoFUCHSIA::pImageCreateInfos element and the VkImageConstraintsInfoFUCHSIA::pFormatConstraints element chosen by Sysmem when vkSetBufferCollectionImageConstraintsFUCHSIA was called. The value of sysmemColorSpaceIndex will

be an index to one of the color spaces provided in the VkImageFormatConstraintsInfoFUCHSIA::pColorSpaces array.

The implementation **must** have formatFeatures with all bits set that were set in VkImageFormatConstraintsInfoFUCHSIA::requiredFormatFeatures, by the call to vkSetBufferCollectionImageConstraintsFUCHSIA, at createInfoIndex (other bits could be set as well).

### Valid Usage (Implicit)

- VUID-VkBufferCollectionPropertiesFUCHSIA-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_BUFFER_COLLECTION_PROPERTIES_FUCHSIA
- VUID-VkBufferCollectionPropertiesFUCHSIA-pNext-pNext
  pNext **must** be NULL

## 12.11.6. Memory Allocation

To import memory from a buffer collection into a VkImage or a VkBuffer, chain a VkImportMemoryBufferCollectionFUCHSIA structure to the pNext member of the VkMemoryAllocateInfo in the call to vkAllocateMemory.

The VkImportMemoryBufferCollectionFUCHSIA structure is defined as:

```
// Provided by VK_FUCHSIA_buffer_collection
typedef struct VkImportMemoryBufferCollectionFUCHSIA {
    VkStructureType              sType;
    const void*                  pNext;
    VkBufferCollectionFUCHSIA    collection;
    uint32_t                     index;
} VkImportMemoryBufferCollectionFUCHSIA;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure
- collection is the VkBufferCollectionFUCHSIA handle
- index the index of the buffer to import from collection

### Valid Usage

- VUID-VkImportMemoryBufferCollectionFUCHSIA-index-06406
  index **must** be less than the value retrieved as VkBufferCollectionPropertiesFUCHSIA:bufferCount

### Valid Usage (Implicit)

- VUID-VkImportMemoryBufferCollectionFUCHSIA-sType-sType

sType **must** be VK_STRUCTURE_TYPE_IMPORT_MEMORY_BUFFER_COLLECTION_FUCHSIA

- VUID-VkImportMemoryBufferCollectionFUCHSIA-collection-parameter
  collection **must** be a valid VkBufferCollectionFUCHSIA handle

To release a VkBufferCollectionFUCHSIA:

```
// Provided by VK_FUCHSIA_buffer_collection
void vkDestroyBufferCollectionFUCHSIA(
    VkDevice                                    device,
    VkBufferCollectionFUCHSIA                   collection,
    const VkAllocationCallbacks*                pAllocator);
```

- device is the logical device that creates the VkBufferCollectionFUCHSIA
- collection is the VkBufferCollectionFUCHSIA handle
- pAllocator is a pointer to a VkAllocationCallbacks structure controlling host memory allocation as described in the Memory Allocation chapter

## Valid Usage

- VUID-vkDestroyBufferCollectionFUCHSIA-collection-06407
  VkImage and VkBuffer objects that referenced collection upon creation by inclusion of a VkBufferCollectionImageCreateInfoFUCHSIA or VkBufferCollectionBufferCreateInfoFUCHSIA chained to their VkImageCreateInfo or VkBufferCreateInfo structures respectively, **may** outlive collection

## Valid Usage (Implicit)

- VUID-vkDestroyBufferCollectionFUCHSIA-device-parameter
  device **must** be a valid VkDevice handle
- VUID-vkDestroyBufferCollectionFUCHSIA-collection-parameter
  collection **must** be a valid VkBufferCollectionFUCHSIA handle
- VUID-vkDestroyBufferCollectionFUCHSIA-pAllocator-parameter
  If pAllocator is not NULL, pAllocator **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkDestroyBufferCollectionFUCHSIA-collection-parent
  collection **must** have been created, allocated, or retrieved from device

# Chapter 13. Samplers

VkSampler objects represent the state of an image sampler which is used by the implementation to read image data and apply filtering and other transformations for the shader.

Samplers are represented by VkSampler handles:

```
// Provided by VK_VERSION_1_0
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkSampler)
```

To create a sampler object, call:

```
// Provided by VK_VERSION_1_0
VkResult vkCreateSampler(
    VkDevice                                    device,
    const VkSamplerCreateInfo*                  pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkSampler*                                  pSampler);
```

- device is the logical device that creates the sampler.
- pCreateInfo is a pointer to a VkSamplerCreateInfo structure specifying the state of the sampler object.
- pAllocator controls host memory allocation as described in the Memory Allocation chapter.
- pSampler is a pointer to a VkSampler handle in which the resulting sampler object is returned.

### Valid Usage

- VUID-vkCreateSampler-device-09668
  device **must** support at least one queue family with one of the VK_QUEUE_COMPUTE_BIT or VK_QUEUE_GRAPHICS_BIT capabilities
- VUID-vkCreateSampler-maxSamplerAllocationCount-04110
  There **must** be less than VkPhysicalDeviceLimits::maxSamplerAllocationCount VkSampler objects currently created on the device

### Valid Usage (Implicit)

- VUID-vkCreateSampler-device-parameter
  device **must** be a valid VkDevice handle
- VUID-vkCreateSampler-pCreateInfo-parameter
  pCreateInfo **must** be a valid pointer to a valid VkSamplerCreateInfo structure
- VUID-vkCreateSampler-pAllocator-parameter
  If pAllocator is not NULL, pAllocator **must** be a valid pointer to a valid

VkAllocationCallbacks structure

- VUID-vkCreateSampler-pSampler-parameter
  `pSampler` **must** be a valid pointer to a VkSampler handle

## Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_INVALID_OPAQUE_CAPTURE_ADDRESS_KHR`

The `VkSamplerCreateInfo` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkSamplerCreateInfo {
    VkStructureType         sType;
    const void*             pNext;
    VkSamplerCreateFlags    flags;
    VkFilter                magFilter;
    VkFilter                minFilter;
    VkSamplerMipmapMode     mipmapMode;
    VkSamplerAddressMode    addressModeU;
    VkSamplerAddressMode    addressModeV;
    VkSamplerAddressMode    addressModeW;
    float                   mipLodBias;
    VkBool32                anisotropyEnable;
    float                   maxAnisotropy;
    VkBool32                compareEnable;
    VkCompareOp             compareOp;
    float                   minLod;
    float                   maxLod;
    VkBorderColor           borderColor;
    VkBool32                unnormalizedCoordinates;
} VkSamplerCreateInfo;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is a bitmask of VkSamplerCreateFlagBits describing additional parameters of the sampler.
- `magFilter` is a VkFilter value specifying the magnification filter to apply to lookups.
- `minFilter` is a VkFilter value specifying the minification filter to apply to lookups.
- `mipmapMode` is a VkSamplerMipmapMode value specifying the mipmap filter to apply to lookups.

- `addressModeU` is a VkSamplerAddressMode value specifying the addressing mode for U coordinates outside [0,1).
- `addressModeV` is a VkSamplerAddressMode value specifying the addressing mode for V coordinates outside [0,1).
- `addressModeW` is a VkSamplerAddressMode value specifying the addressing mode for W coordinates outside [0,1).
- `mipLodBias` is the bias to be added to mipmap LOD calculation and bias provided by image sampling functions in SPIR-V, as described in the LOD Operation section.
- `anisotropyEnable` is `VK_TRUE` to enable anisotropic filtering, as described in the Texel Anisotropic Filtering section, or `VK_FALSE` otherwise.
- `maxAnisotropy` is the anisotropy value clamp used by the sampler when `anisotropyEnable` is `VK_TRUE`. If `anisotropyEnable` is `VK_FALSE`, `maxAnisotropy` is ignored.
- `compareEnable` is `VK_TRUE` to enable comparison against a reference value during lookups, or `VK_FALSE` otherwise.
  - Note: Some implementations will default to shader state if this member does not match.
- `compareOp` is a VkCompareOp value specifying the comparison operator to apply to fetched data before filtering as described in the Depth Compare Operation section.
- `minLod` is used to clamp the minimum of the computed LOD value.
- `maxLod` is used to clamp the maximum of the computed LOD value. To avoid clamping the maximum value, set `maxLod` to the constant `VK_LOD_CLAMP_NONE`.
- `borderColor` is a VkBorderColor value specifying the predefined border color to use.
- `unnormalizedCoordinates` controls whether to use unnormalized or normalized texel coordinates to address texels of the image. When set to `VK_TRUE`, the range of the image coordinates used to lookup the texel is in the range of zero to the image size in each dimension. When set to `VK_FALSE` the range of image coordinates is zero to one.

  When `unnormalizedCoordinates` is `VK_TRUE`, images the sampler is used with in the shader have the following requirements:

  - The `viewType` **must** be either `VK_IMAGE_VIEW_TYPE_1D` or `VK_IMAGE_VIEW_TYPE_2D`.
  - The image view **must** have a single layer and a single mip level.

  When `unnormalizedCoordinates` is `VK_TRUE`, image built-in functions in the shader that use the sampler have the following requirements:

  - The functions **must** not use projection.
  - The functions **must** not use offsets.

**NOTE**

*Mapping of OpenGL to Vulkan filter modes*

`magFilter` values of `VK_FILTER_NEAREST` and `VK_FILTER_LINEAR` directly correspond to `GL_NEAREST` and `GL_LINEAR` magnification filters. `minFilter` and `mipmapMode` combine to correspond to the similarly named OpenGL minification filter of `GL_minFilter_MIPMAP_mipmapMode` (e.g. `minFilter` of `VK_FILTER_LINEAR` and `mipmapMode` of

VK_SAMPLER_MIPMAP_MODE_NEAREST correspond to GL_LINEAR_MIPMAP_NEAREST).

There are no Vulkan filter modes that directly correspond to OpenGL minification filters of GL_LINEAR or GL_NEAREST, but they **can** be emulated using VK_SAMPLER_MIPMAP_MODE_NEAREST, minLod = 0, and maxLod = 0.25, and using minFilter = VK_FILTER_LINEAR or minFilter = VK_FILTER_NEAREST, respectively.

Note that using a maxLod of zero would cause magnification to always be performed, and the magFilter to always be used. This is valid, just not an exact match for OpenGL behavior. Clamping the maximum LOD to 0.25 allows the λ value to be non-zero and minification to be performed, while still always rounding down to the base level. If the minFilter and magFilter are equal, then using a maxLod of zero also works.

The maximum number of sampler objects which **can** be simultaneously created on a device is implementation-dependent and specified by the maxSamplerAllocationCount member of the VkPhysicalDeviceLimits structure.

**NOTE** For historical reasons, if maxSamplerAllocationCount is exceeded, some implementations may return VK_ERROR_TOO_MANY_OBJECTS. Exceeding this limit will result in undefined behavior, and an application should not rely on the use of the returned error code in order to identify when the limit is reached.

Since VkSampler is a non-dispatchable handle type, implementations **may** return the same handle for sampler state vectors that are identical. In such cases, all such objects would only count once against the maxSamplerAllocationCount limit.

### Valid Usage

- VUID-VkSamplerCreateInfo-mipLodBias-01069
  The absolute value of mipLodBias **must** be less than or equal to VkPhysicalDeviceLimits::maxSamplerLodBias
- VUID-VkSamplerCreateInfo-samplerMipLodBias-04467
  If the VK_KHR_portability_subset extension is enabled, and VkPhysicalDevicePortabilitySubsetFeaturesKHR::samplerMipLodBias is VK_FALSE, mipLodBias **must** be zero
- VUID-VkSamplerCreateInfo-maxLod-01973
  maxLod **must** be greater than or equal to minLod
- VUID-VkSamplerCreateInfo-anisotropyEnable-01070
  If the samplerAnisotropy feature is not enabled, anisotropyEnable **must** be VK_FALSE
- VUID-VkSamplerCreateInfo-anisotropyEnable-01071
  If anisotropyEnable is VK_TRUE, maxAnisotropy **must** be between 1.0 and VkPhysicalDeviceLimits::maxSamplerAnisotropy, inclusive
- VUID-VkSamplerCreateInfo-minFilter-01645
  If sampler $Y'C_BC_R$ conversion is enabled and the potential format features of the sampler $Y'C_BC_R$ conversion do not support

VK_FORMAT_FEATURE_SAMPLED_IMAGE_YCBCR_CONVERSION_SEPARATE_RECONSTRUCTION_FILTER_BIT, minFilter and magFilter **must** be equal to the sampler $Y'C_BC_R$ conversion's chromaFilter

- VUID-VkSamplerCreateInfo-unnormalizedCoordinates-01072
  If unnormalizedCoordinates is VK_TRUE, minFilter and magFilter **must** be equal
- VUID-VkSamplerCreateInfo-unnormalizedCoordinates-01073
  If unnormalizedCoordinates is VK_TRUE, mipmapMode **must** be VK_SAMPLER_MIPMAP_MODE_NEAREST
- VUID-VkSamplerCreateInfo-unnormalizedCoordinates-01074
  If unnormalizedCoordinates is VK_TRUE, minLod and maxLod **must** be zero
- VUID-VkSamplerCreateInfo-unnormalizedCoordinates-01075
  If unnormalizedCoordinates is VK_TRUE, addressModeU and addressModeV **must** each be either VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_EDGE or VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_BORDER
- VUID-VkSamplerCreateInfo-unnormalizedCoordinates-01076
  If unnormalizedCoordinates is VK_TRUE, anisotropyEnable **must** be VK_FALSE
- VUID-VkSamplerCreateInfo-unnormalizedCoordinates-01077
  If unnormalizedCoordinates is VK_TRUE, compareEnable **must** be VK_FALSE
- VUID-VkSamplerCreateInfo-addressModeU-01078
  If any of addressModeU, addressModeV or addressModeW are VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_BORDER, borderColor **must** be a valid VkBorderColor value
- VUID-VkSamplerCreateInfo-addressModeU-01646
  If sampler $Y'C_BC_R$ conversion is enabled, addressModeU, addressModeV, and addressModeW **must** be VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_EDGE, anisotropyEnable **must** be VK_FALSE, and unnormalizedCoordinates **must** be VK_FALSE
- VUID-VkSamplerCreateInfo-None-01647
  If sampler $Y'C_BC_R$ conversion is enabled and the pNext chain includes a VkSamplerReductionModeCreateInfo structure, then the sampler reduction mode **must** be set to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE
- VUID-VkSamplerCreateInfo-addressModeU-01079
  If samplerMirrorClampToEdge is not enabled, and if the VK_KHR_sampler_mirror_clamp_to_edge extension is not enabled, addressModeU, addressModeV and addressModeW **must** not be VK_SAMPLER_ADDRESS_MODE_MIRROR_CLAMP_TO_EDGE
- VUID-VkSamplerCreateInfo-compareEnable-01080
  If compareEnable is VK_TRUE, compareOp **must** be a valid VkCompareOp value
- VUID-VkSamplerCreateInfo-magFilter-01081
  If either magFilter or minFilter is VK_FILTER_CUBIC_EXT, anisotropyEnable **must** be VK_FALSE
- VUID-VkSamplerCreateInfo-magFilter-07911
  If the VK_EXT_filter_cubic extension is not enabled and either magFilter or minFilter is VK_FILTER_CUBIC_IMG, the reductionMode member of VkSamplerReductionModeCreateInfo **must** be VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE
- VUID-VkSamplerCreateInfo-compareEnable-01423
  If compareEnable is VK_TRUE, the reductionMode member of VkSamplerReductionModeCreateInfo **must** be

VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE

- VUID-VkSamplerCreateInfo-flags-02574
  If flags includes VK_SAMPLER_CREATE_SUBSAMPLED_BIT_EXT, then minFilter and magFilter **must** be equal
- VUID-VkSamplerCreateInfo-flags-02575
  If flags includes VK_SAMPLER_CREATE_SUBSAMPLED_BIT_EXT, then mipmapMode **must** be VK_SAMPLER_MIPMAP_MODE_NEAREST
- VUID-VkSamplerCreateInfo-flags-02576
  If flags includes VK_SAMPLER_CREATE_SUBSAMPLED_BIT_EXT, then minLod and maxLod **must** be zero
- VUID-VkSamplerCreateInfo-flags-02577
  If flags includes VK_SAMPLER_CREATE_SUBSAMPLED_BIT_EXT, then addressModeU and addressModeV **must** each be either VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_EDGE or VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_BORDER
- VUID-VkSamplerCreateInfo-flags-02578
  If flags includes VK_SAMPLER_CREATE_SUBSAMPLED_BIT_EXT, then anisotropyEnable **must** be VK_FALSE
- VUID-VkSamplerCreateInfo-flags-02579
  If flags includes VK_SAMPLER_CREATE_SUBSAMPLED_BIT_EXT, then compareEnable **must** be VK_FALSE
- VUID-VkSamplerCreateInfo-flags-02580
  If flags includes VK_SAMPLER_CREATE_SUBSAMPLED_BIT_EXT, then unnormalizedCoordinates **must** be VK_FALSE
- VUID-VkSamplerCreateInfo-nonSeamlessCubeMap-06788
  If the nonSeamlessCubeMap feature is not enabled, flags **must** not include VK_SAMPLER_CREATE_NON_SEAMLESS_CUBE_MAP_BIT_EXT
- VUID-VkSamplerCreateInfo-borderColor-04011
  If borderColor is one of VK_BORDER_COLOR_FLOAT_CUSTOM_EXT or VK_BORDER_COLOR_INT_CUSTOM_EXT, then a VkSamplerCustomBorderColorCreateInfoEXT **must** be included in the pNext chain
- VUID-VkSamplerCreateInfo-customBorderColors-04085
  If the customBorderColors feature is not enabled, borderColor **must** not be VK_BORDER_COLOR_FLOAT_CUSTOM_EXT or VK_BORDER_COLOR_INT_CUSTOM_EXT
- VUID-VkSamplerCreateInfo-borderColor-04442
  If borderColor is one of VK_BORDER_COLOR_FLOAT_CUSTOM_EXT or VK_BORDER_COLOR_INT_CUSTOM_EXT, and VkSamplerCustomBorderColorCreateInfoEXT::format is not VK_FORMAT_UNDEFINED, VkSamplerCustomBorderColorCreateInfoEXT::customBorderColor **must** be within the range of values representable in format
- VUID-VkSamplerCreateInfo-None-04012
  The maximum number of samplers with custom border colors which **can** be simultaneously created on a device is implementation-dependent and specified by the maxCustomBorderColorSamplers member of the VkPhysicalDeviceCustomBorderColorPropertiesEXT structure

- VUID-VkSamplerCreateInfo-flags-08110
  If flags includes VK_SAMPLER_CREATE_DESCRIPTOR_BUFFER_CAPTURE_REPLAY_BIT_EXT, the descriptorBufferCaptureReplay feature **must** be enabled
- VUID-VkSamplerCreateInfo-pNext-08111
  If the pNext chain includes a VkOpaqueCaptureDescriptorDataCreateInfoEXT structure, flags **must** contain VK_SAMPLER_CREATE_DESCRIPTOR_BUFFER_CAPTURE_REPLAY_BIT_EXT
- VUID-VkSamplerCreateInfo-flags-06964
  If flags includes VK_SAMPLER_CREATE_IMAGE_PROCESSING_BIT_QCOM, then minFilter and magFilter **must** be VK_FILTER_NEAREST
- VUID-VkSamplerCreateInfo-flags-06965
  If flags includes VK_SAMPLER_CREATE_IMAGE_PROCESSING_BIT_QCOM, then mipmapMode **must** be VK_SAMPLER_MIPMAP_MODE_NEAREST
- VUID-VkSamplerCreateInfo-flags-06966
  [If flags includes VK_SAMPLER_CREATE_IMAGE_PROCESSING_BIT_QCOM, then minLod and maxLod **must** be zero
- VUID-VkSamplerCreateInfo-flags-06967
  If flags includes VK_SAMPLER_CREATE_IMAGE_PROCESSING_BIT_QCOM, then addressModeU and addressModeV **must** each be either VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_EDGE or VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_BORDER
- VUID-VkSamplerCreateInfo-flags-06968
  If flags includes VK_SAMPLER_CREATE_IMAGE_PROCESSING_BIT_QCOM, and if addressModeU or addressModeV is VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_BORDER, then borderColor **must** be VK_BORDER_COLOR_FLOAT_TRANSPARENT_BLACK
- VUID-VkSamplerCreateInfo-flags-06969
  If flags includes VK_SAMPLER_CREATE_IMAGE_PROCESSING_BIT_QCOM, then anisotropyEnable **must** be VK_FALSE
- VUID-VkSamplerCreateInfo-flags-06970
  If flags includes VK_SAMPLER_CREATE_IMAGE_PROCESSING_BIT_QCOM, then compareEnable **must** be VK_FALSE

## Valid Usage (Implicit)

- VUID-VkSamplerCreateInfo-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_SAMPLER_CREATE_INFO
- VUID-VkSamplerCreateInfo-pNext-pNext
  Each pNext member of any structure (including this one) in the pNext chain **must** be either NULL or a pointer to a valid instance of VkOpaqueCaptureDescriptorDataCreateInfoEXT, VkSamplerBlockMatchWindowCreateInfoQCOM, VkSamplerBorderColorComponentMappingCreateInfoEXT, VkSamplerCubicWeightsCreateInfoQCOM, VkSamplerCustomBorderColorCreateInfoEXT, VkSamplerReductionModeCreateInfo, or VkSamplerYcbcrConversionInfo
- VUID-VkSamplerCreateInfo-sType-unique
  The sType value of each struct in the pNext chain **must** be unique

- VUID-VkSamplerCreateInfo-flags-parameter
  `flags` **must** be a valid combination of VkSamplerCreateFlagBits values
- VUID-VkSamplerCreateInfo-magFilter-parameter
  `magFilter` **must** be a valid VkFilter value
- VUID-VkSamplerCreateInfo-minFilter-parameter
  `minFilter` **must** be a valid VkFilter value
- VUID-VkSamplerCreateInfo-mipmapMode-parameter
  `mipmapMode` **must** be a valid VkSamplerMipmapMode value
- VUID-VkSamplerCreateInfo-addressModeU-parameter
  `addressModeU` **must** be a valid VkSamplerAddressMode value
- VUID-VkSamplerCreateInfo-addressModeV-parameter
  `addressModeV` **must** be a valid VkSamplerAddressMode value
- VUID-VkSamplerCreateInfo-addressModeW-parameter
  `addressModeW` **must** be a valid VkSamplerAddressMode value

`VK_LOD_CLAMP_NONE` is a special constant value used for VkSamplerCreateInfo::`maxLod` to indicate that maximum LOD clamping should not be performed.

```
#define VK_LOD_CLAMP_NONE                 1000.0F
```

Bits which **can** be set in VkSamplerCreateInfo::`flags`, specifying additional parameters of a sampler, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkSamplerCreateFlagBits {
  // Provided by VK_EXT_fragment_density_map
    VK_SAMPLER_CREATE_SUBSAMPLED_BIT_EXT = 0x00000001,
  // Provided by VK_EXT_fragment_density_map
    VK_SAMPLER_CREATE_SUBSAMPLED_COARSE_RECONSTRUCTION_BIT_EXT = 0x00000002,
  // Provided by VK_EXT_descriptor_buffer
    VK_SAMPLER_CREATE_DESCRIPTOR_BUFFER_CAPTURE_REPLAY_BIT_EXT = 0x00000008,
  // Provided by VK_EXT_non_seamless_cube_map
    VK_SAMPLER_CREATE_NON_SEAMLESS_CUBE_MAP_BIT_EXT = 0x00000004,
  // Provided by VK_QCOM_image_processing
    VK_SAMPLER_CREATE_IMAGE_PROCESSING_BIT_QCOM = 0x00000010,
} VkSamplerCreateFlagBits;
```

- `VK_SAMPLER_CREATE_SUBSAMPLED_BIT_EXT` specifies that the sampler will read from an image created with `flags` containing `VK_IMAGE_CREATE_SUBSAMPLED_BIT_EXT`.
- `VK_SAMPLER_CREATE_SUBSAMPLED_COARSE_RECONSTRUCTION_BIT_EXT` specifies that the implementation **may** use approximations when reconstructing a full color value for texture access from a subsampled image.
- `VK_SAMPLER_CREATE_NON_SEAMLESS_CUBE_MAP_BIT_EXT` specifies that cube map edge handling is not

performed.

- VK_SAMPLER_CREATE_IMAGE_PROCESSING_BIT_QCOM specifies that the sampler will read from images using only OpImageWeightedSampleQCOM, OpImageBoxFilterQCOM, OpImageBlockMatchGatherSSDQCOM, OpImageBlockMatchGatherSADQCOM, OpImageBlockMatchWindowSSDQCOM, OpImageBlockMatchWindowSADQCOM, OpImageBlockMatchSSDQCOM, or OpImageBlockMatchSADQCOM.

| | |
|---|---|
| **NOTE** | The approximations used when VK_SAMPLER_CREATE_SUBSAMPLED_COARSE_RECONSTRUCTION_BIT_EXT is specified are implementation defined. Some implementations **may** interpolate between fragment density levels in a subsampled image. In that case, this bit **may** be used to decide whether the interpolation factors are calculated per fragment or at a coarser granularity. |

- VK_SAMPLER_CREATE_DESCRIPTOR_BUFFER_CAPTURE_REPLAY_BIT_EXT specifies that the sampler **can** be used with descriptor buffers when capturing and replaying (e.g. for trace capture and replay), see VkOpaqueCaptureDescriptorDataCreateInfoEXT for more detail.

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkSamplerCreateFlags;
```

VkSamplerCreateFlags is a bitmask type for setting a mask of zero or more VkSamplerCreateFlagBits.

The VkSamplerReductionModeCreateInfo structure is defined as:

```
typedef struct VkSamplerReductionModeCreateInfo {
    VkStructureType           sType;
    const void*               pNext;
    VkSamplerReductionMode    reductionMode;
} VkSamplerReductionModeCreateInfo;
```

or the equivalent

```
// Provided by VK_EXT_sampler_filter_minmax
typedef VkSamplerReductionModeCreateInfo VkSamplerReductionModeCreateInfoEXT;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- reductionMode is a VkSamplerReductionMode value controlling how texture filtering combines texel values.

If the pNext chain of VkSamplerCreateInfo includes a VkSamplerReductionModeCreateInfo structure, then that structure includes a mode controlling how texture filtering combines texel values.

If this structure is not present, reductionMode is considered to be VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE.

### Valid Usage (Implicit)

- VUID-VkSamplerReductionModeCreateInfo-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_SAMPLER_REDUCTION_MODE_CREATE_INFO`
- VUID-VkSamplerReductionModeCreateInfo-reductionMode-parameter
  `reductionMode` **must** be a valid VkSamplerReductionMode value

Reduction modes are specified by VkSamplerReductionMode, which takes values:

```
typedef enum VkSamplerReductionMode {
    VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE = 0,
    VK_SAMPLER_REDUCTION_MODE_MIN = 1,
    VK_SAMPLER_REDUCTION_MODE_MAX = 2,
  // Provided by VK_QCOM_filter_cubic_clamp
    VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE_RANGECLAMP_QCOM = 1000521000,
  // Provided by VK_EXT_sampler_filter_minmax
    VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE_EXT =
VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE,
  // Provided by VK_EXT_sampler_filter_minmax
    VK_SAMPLER_REDUCTION_MODE_MIN_EXT = VK_SAMPLER_REDUCTION_MODE_MIN,
  // Provided by VK_EXT_sampler_filter_minmax
    VK_SAMPLER_REDUCTION_MODE_MAX_EXT = VK_SAMPLER_REDUCTION_MODE_MAX,
} VkSamplerReductionMode;
```

or the equivalent

```
// Provided by VK_EXT_sampler_filter_minmax
typedef VkSamplerReductionMode VkSamplerReductionModeEXT;
```

- `VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE` specifies that texel values are combined by computing a weighted average of values in the footprint, using weights as specified in the image operations chapter.
- `VK_SAMPLER_REDUCTION_MODE_MIN` specifies that texel values are combined by taking the component-wise minimum of values in the footprint with non-zero weights.
- `VK_SAMPLER_REDUCTION_MODE_MAX` specifies that texel values are combined by taking the component-wise maximum of values in the footprint with non-zero weights.
- `VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE_RANGECLAMP_QCOM` specifies values are combined as described by `VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE`, followed by a texel range clamp.

The `VkSamplerCubicWeightsCreateInfoQCOM` structure is defined as:

```
// Provided by VK_QCOM_filter_cubic_weights
typedef struct VkSamplerCubicWeightsCreateInfoQCOM {
    VkStructureType             sType;
    const void*                 pNext;
    VkCubicFilterWeightsQCOM    cubicWeights;
} VkSamplerCubicWeightsCreateInfoQCOM;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `cubicWeights` is a VkCubicFilterWeightsQCOM value controlling which cubic weights are used.

If the `pNext` chain of VkSamplerCreateInfo includes a `VkSamplerCubicWeightsCreateInfoQCOM` structure, then that structure specifies which cubic weights are used.

If that structure is not present, `cubicWeights` is considered to be `VK_CUBIC_FILTER_WEIGHTS_CATMULL_ROM_QCOM`.

### Valid Usage (Implicit)

- VUID-VkSamplerCubicWeightsCreateInfoQCOM-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_SAMPLER_CUBIC_WEIGHTS_CREATE_INFO_QCOM`
- VUID-VkSamplerCubicWeightsCreateInfoQCOM-cubicWeights-parameter
  `cubicWeights` **must** be a valid VkCubicFilterWeightsQCOM value

Possible values of the VkSamplerCubicWeightsCreateInfoQCOM::`cubicWeights`, specifying cubic weights used in Texel Cubic Filtering are:

```
// Provided by VK_QCOM_filter_cubic_weights
typedef enum VkCubicFilterWeightsQCOM {
    VK_CUBIC_FILTER_WEIGHTS_CATMULL_ROM_QCOM = 0,
    VK_CUBIC_FILTER_WEIGHTS_ZERO_TANGENT_CARDINAL_QCOM = 1,
    VK_CUBIC_FILTER_WEIGHTS_B_SPLINE_QCOM = 2,
    VK_CUBIC_FILTER_WEIGHTS_MITCHELL_NETRAVALI_QCOM = 3,
} VkCubicFilterWeightsQCOM;
```

- `VK_CUBIC_FILTER_WEIGHTS_CATMULL_ROM_QCOM` specifies Catmull-Rom weights.
- `VK_CUBIC_FILTER_WEIGHTS_ZERO_TANGENT_CARDINAL_QCOM` specifies Zero Tangent Cardinal weights.
- `VK_CUBIC_FILTER_WEIGHTS_B_SPLINE_QCOM` specifies B-Spline weights.
- `VK_CUBIC_FILTER_WEIGHTS_MITCHELL_NETRAVALI_QCOM` specifies Mitchell-Netravali weights.

Possible values of the VkSamplerCreateInfo::`magFilter` and `minFilter` parameters, specifying filters used for texture lookups, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkFilter {
    VK_FILTER_NEAREST = 0,
    VK_FILTER_LINEAR = 1,
  // Provided by VK_EXT_filter_cubic
    VK_FILTER_CUBIC_EXT = 1000015000,
  // Provided by VK_IMG_filter_cubic
    VK_FILTER_CUBIC_IMG = VK_FILTER_CUBIC_EXT,
} VkFilter;
```

- `VK_FILTER_NEAREST` specifies nearest filtering.
- `VK_FILTER_LINEAR` specifies linear filtering.
- `VK_FILTER_CUBIC_EXT` specifies cubic filtering.

These filters are described in detail in Texel Filtering.

Possible values of the VkSamplerCreateInfo::`mipmapMode`, specifying the mipmap mode used for texture lookups, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkSamplerMipmapMode {
    VK_SAMPLER_MIPMAP_MODE_NEAREST = 0,
    VK_SAMPLER_MIPMAP_MODE_LINEAR = 1,
} VkSamplerMipmapMode;
```

- `VK_SAMPLER_MIPMAP_MODE_NEAREST` specifies nearest filtering.
- `VK_SAMPLER_MIPMAP_MODE_LINEAR` specifies linear filtering.

These modes are described in detail in Texel Filtering.

Possible values of the VkSamplerCreateInfo::`addressMode*` parameters, specifying the behavior of sampling with coordinates outside the range [0,1] for the respective u, v, or w coordinate as defined in the Wrapping Operation section, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkSamplerAddressMode {
    VK_SAMPLER_ADDRESS_MODE_REPEAT = 0,
    VK_SAMPLER_ADDRESS_MODE_MIRRORED_REPEAT = 1,
    VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_EDGE = 2,
    VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_BORDER = 3,
  // Provided by VK_KHR_sampler_mirror_clamp_to_edge
    VK_SAMPLER_ADDRESS_MODE_MIRROR_CLAMP_TO_EDGE = 4,
  // Provided by VK_KHR_sampler_mirror_clamp_to_edge
  // VK_SAMPLER_ADDRESS_MODE_MIRROR_CLAMP_TO_EDGE_KHR is a deprecated alias
    VK_SAMPLER_ADDRESS_MODE_MIRROR_CLAMP_TO_EDGE_KHR =
VK_SAMPLER_ADDRESS_MODE_MIRROR_CLAMP_TO_EDGE,
} VkSamplerAddressMode;
```

- `VK_SAMPLER_ADDRESS_MODE_REPEAT` specifies that the repeat wrap mode will be used.
- `VK_SAMPLER_ADDRESS_MODE_MIRRORED_REPEAT` specifies that the mirrored repeat wrap mode will be used.
- `VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_EDGE` specifies that the clamp to edge wrap mode will be used.
- `VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_BORDER` specifies that the clamp to border wrap mode will be used.
- `VK_SAMPLER_ADDRESS_MODE_MIRROR_CLAMP_TO_EDGE` specifies that the mirror clamp to edge wrap mode will be used. This is only valid if the `VK_KHR_sampler_mirror_clamp_to_edge` extension is enabled.

*Comparison operators* compare a *reference* and a *test* value, and return a true ("passed") or false ("failed") value depending on the comparison operator chosen. The supported operators are:

```
// Provided by VK_VERSION_1_0
typedef enum VkCompareOp {
    VK_COMPARE_OP_NEVER = 0,
    VK_COMPARE_OP_LESS = 1,
    VK_COMPARE_OP_EQUAL = 2,
    VK_COMPARE_OP_LESS_OR_EQUAL = 3,
    VK_COMPARE_OP_GREATER = 4,
    VK_COMPARE_OP_NOT_EQUAL = 5,
    VK_COMPARE_OP_GREATER_OR_EQUAL = 6,
    VK_COMPARE_OP_ALWAYS = 7,
} VkCompareOp;
```

- `VK_COMPARE_OP_NEVER` specifies that the comparison always evaluates false.
- `VK_COMPARE_OP_LESS` specifies that the comparison evaluates *reference* < *test*.
- `VK_COMPARE_OP_EQUAL` specifies that the comparison evaluates *reference* = *test*.
- `VK_COMPARE_OP_LESS_OR_EQUAL` specifies that the comparison evaluates *reference* ≤ *test*.
- `VK_COMPARE_OP_GREATER` specifies that the comparison evaluates *reference* > *test*.

- VK_COMPARE_OP_NOT_EQUAL specifies that the comparison evaluates *reference* ≠ *test*.
- VK_COMPARE_OP_GREATER_OR_EQUAL specifies that the comparison evaluates *reference* ≥ *test*.
- VK_COMPARE_OP_ALWAYS specifies that the comparison always evaluates true.

Comparison operators are used for:

- The Depth Compare Operation operator for a sampler, specified by VkSamplerCreateInfo::compareOp.
- The stencil comparison operator for the stencil test, specified by vkCmdSetStencilOpEXT::compareOp or VkStencilOpState::compareOp.
- The Depth Comparison operator for the depth test, specified by vkCmdSetDepthCompareOpEXT::depthCompareOp or VkPipelineDepthStencilStateCreateInfo::depthCompareOp.

Each such use describes how the *reference* and *test* values for that comparison are determined.

Possible values of VkSamplerCreateInfo::borderColor, specifying the border color used for texture lookups, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkBorderColor {
    VK_BORDER_COLOR_FLOAT_TRANSPARENT_BLACK = 0,
    VK_BORDER_COLOR_INT_TRANSPARENT_BLACK = 1,
    VK_BORDER_COLOR_FLOAT_OPAQUE_BLACK = 2,
    VK_BORDER_COLOR_INT_OPAQUE_BLACK = 3,
    VK_BORDER_COLOR_FLOAT_OPAQUE_WHITE = 4,
    VK_BORDER_COLOR_INT_OPAQUE_WHITE = 5,
  // Provided by VK_EXT_custom_border_color
    VK_BORDER_COLOR_FLOAT_CUSTOM_EXT = 1000287003,
  // Provided by VK_EXT_custom_border_color
    VK_BORDER_COLOR_INT_CUSTOM_EXT = 1000287004,
} VkBorderColor;
```

- VK_BORDER_COLOR_FLOAT_TRANSPARENT_BLACK specifies a transparent, floating-point format, black color.
- VK_BORDER_COLOR_INT_TRANSPARENT_BLACK specifies a transparent, integer format, black color.
- VK_BORDER_COLOR_FLOAT_OPAQUE_BLACK specifies an opaque, floating-point format, black color.
- VK_BORDER_COLOR_INT_OPAQUE_BLACK specifies an opaque, integer format, black color.
- VK_BORDER_COLOR_FLOAT_OPAQUE_WHITE specifies an opaque, floating-point format, white color.
- VK_BORDER_COLOR_INT_OPAQUE_WHITE specifies an opaque, integer format, white color.
- VK_BORDER_COLOR_FLOAT_CUSTOM_EXT indicates that a VkSamplerCustomBorderColorCreateInfoEXT structure is included in the VkSamplerCreateInfo::pNext chain containing the color data in floating-point format.
- VK_BORDER_COLOR_INT_CUSTOM_EXT indicates that a VkSamplerCustomBorderColorCreateInfoEXT

structure is included in the VkSamplerCreateInfo::pNext chain containing the color data in integer format.

These colors are described in detail in Texel Replacement.

To destroy a sampler, call:

```
// Provided by VK_VERSION_1_0
void vkDestroySampler(
    VkDevice                                    device,
    VkSampler                                   sampler,
    const VkAllocationCallbacks*                pAllocator);
```

- device is the logical device that destroys the sampler.
- sampler is the sampler to destroy.
- pAllocator controls host memory allocation as described in the Memory Allocation chapter.

### Valid Usage

- VUID-vkDestroySampler-sampler-01082
  All submitted commands that refer to sampler **must** have completed execution
- VUID-vkDestroySampler-sampler-01083
  If VkAllocationCallbacks were provided when sampler was created, a compatible set of callbacks **must** be provided here
- VUID-vkDestroySampler-sampler-01084
  If no VkAllocationCallbacks were provided when sampler was created, pAllocator **must** be NULL

### Valid Usage (Implicit)

- VUID-vkDestroySampler-device-parameter
  device **must** be a valid VkDevice handle
- VUID-vkDestroySampler-sampler-parameter
  If sampler is not VK_NULL_HANDLE, sampler **must** be a valid VkSampler handle
- VUID-vkDestroySampler-pAllocator-parameter
  If pAllocator is not NULL, pAllocator **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkDestroySampler-sampler-parent
  If sampler is a valid handle, it **must** have been created, allocated, or retrieved from device

### Host Synchronization

- Host access to `sampler` **must** be externally synchronized

## 13.1. Sampler $Y'C_BC_R$ Conversion

To create a sampler with $Y'C_BC_R$ conversion enabled, add a VkSamplerYcbcrConversionInfo structure to the `pNext` chain of the VkSamplerCreateInfo structure. To create a sampler $Y'C_BC_R$ conversion, the `samplerYcbcrConversion` feature **must** be enabled. Conversion **must** be fixed at pipeline creation time, through use of a combined image sampler with an immutable sampler in `VkDescriptorSetLayoutBinding`.

A VkSamplerYcbcrConversionInfo **must** be provided for samplers to be used with image views that access `VK_IMAGE_ASPECT_COLOR_BIT` if the format is one of the formats that require a sampler $Y'C_BC_R$ conversion , or if the image view has an external format .

The `VkSamplerYcbcrConversionInfo` structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkSamplerYcbcrConversionInfo {
    VkStructureType             sType;
    const void*                 pNext;
    VkSamplerYcbcrConversion    conversion;
} VkSamplerYcbcrConversionInfo;
```

or the equivalent

```
// Provided by VK_KHR_sampler_ycbcr_conversion
typedef VkSamplerYcbcrConversionInfo VkSamplerYcbcrConversionInfoKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `conversion` is a VkSamplerYcbcrConversion handle created with vkCreateSamplerYcbcrConversion.

### Valid Usage (Implicit)

- VUID-VkSamplerYcbcrConversionInfo-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_SAMPLER_YCBCR_CONVERSION_INFO`
- VUID-VkSamplerYcbcrConversionInfo-conversion-parameter
  `conversion` **must** be a valid VkSamplerYcbcrConversion handle

A sampler $Y'C_BC_R$ conversion is an opaque representation of a device-specific sampler $Y'C_BC_R$

conversion description, represented as a VkSamplerYcbcrConversion handle:

```
// Provided by VK_VERSION_1_1
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkSamplerYcbcrConversion)
```

or the equivalent

```
// Provided by VK_KHR_sampler_ycbcr_conversion
typedef VkSamplerYcbcrConversion VkSamplerYcbcrConversionKHR;
```

To create a VkSamplerYcbcrConversion, call:

```
// Provided by VK_VERSION_1_1
VkResult vkCreateSamplerYcbcrConversion(
    VkDevice                                    device,
    const VkSamplerYcbcrConversionCreateInfo*   pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkSamplerYcbcrConversion*                   pYcbcrConversion);
```

or the equivalent command

```
// Provided by VK_KHR_sampler_ycbcr_conversion
VkResult vkCreateSamplerYcbcrConversionKHR(
    VkDevice                                    device,
    const VkSamplerYcbcrConversionCreateInfo*   pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkSamplerYcbcrConversion*                   pYcbcrConversion);
```

- device is the logical device that creates the sampler $Y'C_BC_R$ conversion.
- pCreateInfo is a pointer to a VkSamplerYcbcrConversionCreateInfo structure specifying the requested sampler $Y'C_BC_R$ conversion.
- pAllocator controls host memory allocation as described in the Memory Allocation chapter.
- pYcbcrConversion is a pointer to a VkSamplerYcbcrConversion handle in which the resulting sampler $Y'C_BC_R$ conversion is returned.

The interpretation of the configured sampler $Y'C_BC_R$ conversion is described in more detail in the description of sampler $Y'C_BC_R$ conversion in the Image Operations chapter.

### Valid Usage

- VUID-vkCreateSamplerYcbcrConversion-None-01648
  The samplerYcbcrConversion feature **must** be enabled

## Valid Usage (Implicit)

- VUID-vkCreateSamplerYcbcrConversion-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkCreateSamplerYcbcrConversion-pCreateInfo-parameter
  `pCreateInfo` **must** be a valid pointer to a valid VkSamplerYcbcrConversionCreateInfo structure
- VUID-vkCreateSamplerYcbcrConversion-pAllocator-parameter
  If `pAllocator` is not `NULL`, `pAllocator` **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkCreateSamplerYcbcrConversion-pYcbcrConversion-parameter
  `pYcbcrConversion` **must** be a valid pointer to a VkSamplerYcbcrConversion handle

## Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`

The `VkSamplerYcbcrConversionCreateInfo` structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkSamplerYcbcrConversionCreateInfo {
    VkStructureType                 sType;
    const void*                     pNext;
    VkFormat                        format;
    VkSamplerYcbcrModelConversion   ycbcrModel;
    VkSamplerYcbcrRange             ycbcrRange;
    VkComponentMapping              components;
    VkChromaLocation                xChromaOffset;
    VkChromaLocation                yChromaOffset;
    VkFilter                        chromaFilter;
    VkBool32                        forceExplicitReconstruction;
} VkSamplerYcbcrConversionCreateInfo;
```

or the equivalent

```
// Provided by VK_KHR_sampler_ycbcr_conversion
typedef VkSamplerYcbcrConversionCreateInfo VkSamplerYcbcrConversionCreateInfoKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `format` is the format of the image from which color information will be retrieved.
- `ycbcrModel` describes the color matrix for conversion between color models.
- `ycbcrRange` describes whether the encoded values have headroom and foot room, or whether the encoding uses the full numerical range.
- `components` applies a *swizzle* based on VkComponentSwizzle enums prior to range expansion and color model conversion.
- `xChromaOffset` describes the sample location associated with downsampled chroma components in the x dimension. `xChromaOffset` has no effect for formats in which chroma components are not downsampled horizontally.
- `yChromaOffset` describes the sample location associated with downsampled chroma components in the y dimension. `yChromaOffset` has no effect for formats in which the chroma components are not downsampled vertically.
- `chromaFilter` is the filter for chroma reconstruction.
- `forceExplicitReconstruction` **can** be used to ensure that reconstruction is done explicitly, if supported.

**NOTE**

Setting `forceExplicitReconstruction` to `VK_TRUE` **may** have a performance penalty on implementations where explicit reconstruction is not the default mode of operation.

If `format` supports `VK_FORMAT_FEATURE_SAMPLED_IMAGE_YCBCR_CONVERSION_CHROMA_RECONSTRUCTION_EXPLICIT_BIT` the `forceExplicitReconstruction` value behaves as if it was set to `VK_TRUE`.

If the `pNext` chain includes a VkExternalFormatANDROID structure with non-zero `externalFormat` member, the sampler $Y'C_BC_R$ conversion object represents an *external format conversion*, and `format` **must** be `VK_FORMAT_UNDEFINED`. Such conversions **must** only be used to sample image views with a matching external format. When creating an external format conversion, the value of `components` is ignored.

## Valid Usage

- VUID-VkSamplerYcbcrConversionCreateInfo-format-01904
  If an external format conversion is being created, `format` **must** be `VK_FORMAT_UNDEFINED`
- VUID-VkSamplerYcbcrConversionCreateInfo-format-04061
  If an external format conversion is not being created, `format` **must** represent unsigned normalized values (i.e. the format **must** be a `UNORM` format)
- VUID-VkSamplerYcbcrConversionCreateInfo-format-01650
  The potential format features of the sampler $Y'C_BC_R$ conversion **must** support `VK_FORMAT_FEATURE_MIDPOINT_CHROMA_SAMPLES_BIT` or `VK_FORMAT_FEATURE_COSITED_CHROMA_SAMPLES_BIT`
- VUID-VkSamplerYcbcrConversionCreateInfo-xChromaOffset-01651

If the potential format features of the sampler $Y'C_BC_R$ conversion do not support VK_FORMAT_FEATURE_COSITED_CHROMA_SAMPLES_BIT, xChromaOffset and yChromaOffset **must** not be VK_CHROMA_LOCATION_COSITED_EVEN if the corresponding components are downsampled

- VUID-VkSamplerYcbcrConversionCreateInfo-xChromaOffset-01652
  If the potential format features of the sampler $Y'C_BC_R$ conversion do not support VK_FORMAT_FEATURE_MIDPOINT_CHROMA_SAMPLES_BIT, xChromaOffset and yChromaOffset **must** not be VK_CHROMA_LOCATION_MIDPOINT if the corresponding components are downsampled
- VUID-VkSamplerYcbcrConversionCreateInfo-components-02581
  If the format has a _422 or _420 suffix, then components.g **must** be the identity swizzle
- VUID-VkSamplerYcbcrConversionCreateInfo-components-02582
  If the format has a _422 or _420 suffix, then components.a **must** be the identity swizzle, VK_COMPONENT_SWIZZLE_ONE, or VK_COMPONENT_SWIZZLE_ZERO
- VUID-VkSamplerYcbcrConversionCreateInfo-components-02583
  If the format has a _422 or _420 suffix, then components.r **must** be the identity swizzle or VK_COMPONENT_SWIZZLE_B
- VUID-VkSamplerYcbcrConversionCreateInfo-components-02584
  If the format has a _422 or _420 suffix, then components.b **must** be the identity swizzle or VK_COMPONENT_SWIZZLE_R
- VUID-VkSamplerYcbcrConversionCreateInfo-components-02585
  If the format has a _422 or _420 suffix, and if either components.r or components.b is the identity swizzle, both values **must** be the identity swizzle
- VUID-VkSamplerYcbcrConversionCreateInfo-ycbcrModel-01655
  If ycbcrModel is not VK_SAMPLER_YCBCR_MODEL_CONVERSION_RGB_IDENTITY, then components.r, components.g, and components.b **must** correspond to components of the format; that is, components.r, components.g, and components.b **must** not be VK_COMPONENT_SWIZZLE_ZERO or VK_COMPONENT_SWIZZLE_ONE, and **must** not correspond to a component containing zero or one as a consequence of conversion to RGBA
- VUID-VkSamplerYcbcrConversionCreateInfo-ycbcrRange-02748
  If ycbcrRange is VK_SAMPLER_YCBCR_RANGE_ITU_NARROW then the R, G and B components obtained by applying the component swizzle to format **must** each have a bit-depth greater than or equal to 8
- VUID-VkSamplerYcbcrConversionCreateInfo-forceExplicitReconstruction-01656
  If the potential format features of the sampler $Y'C_BC_R$ conversion do not support VK_FORMAT_FEATURE_SAMPLED_IMAGE_YCBCR_CONVERSION_CHROMA_RECONSTRUCTION_EXPLICIT_FORCEABLE_BIT forceExplicitReconstruction **must** be VK_FALSE
- VUID-VkSamplerYcbcrConversionCreateInfo-chromaFilter-01657
  If the potential format features of the sampler $Y'C_BC_R$ conversion do not support VK_FORMAT_FEATURE_SAMPLED_IMAGE_YCBCR_CONVERSION_LINEAR_FILTER_BIT, chromaFilter **must** not be VK_FILTER_LINEAR
- VUID-VkSamplerYcbcrConversionCreateInfo-pNext-09207
  If the pNext chain includes a VkSamplerYcbcrConversionYcbcrDegammaCreateInfoQCOM structure, and if the ycbcrDegamma feature is not enabled, then VkSamplerYcbcrConversionYcbcrDegammaCreateInfoQCOM::enableYDegamma **must** be

VK_FALSE

- VUID-VkSamplerYcbcrConversionCreateInfo-pNext-09208
  If the pNext chain includes a VkSamplerYcbcrConversionYcbcrDegammaCreateInfoQCOM structure, and if the ycbcrDegamma feature is not enabled, then VkSamplerYcbcrConversionYcbcrDegammaCreateInfoQCOM::enableCbCrDegamma **must** be VK_FALSE
- VUID-VkSamplerYcbcrConversionCreateInfo-pNext-09209
  If the pNext chain includes a VkSamplerYcbcrConversionYcbcrDegammaCreateInfoQCOM structure, format **must** be a format with 8-bit R, G, and B components

## Valid Usage (Implicit)

- VUID-VkSamplerYcbcrConversionCreateInfo-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_SAMPLER_YCBCR_CONVERSION_CREATE_INFO
- VUID-VkSamplerYcbcrConversionCreateInfo-pNext-pNext
  Each pNext member of any structure (including this one) in the pNext chain **must** be either NULL or a pointer to a valid instance of VkExternalFormatANDROID, VkExternalFormatQNX, or VkSamplerYcbcrConversionYcbcrDegammaCreateInfoQCOM
- VUID-VkSamplerYcbcrConversionCreateInfo-sType-unique
  The sType value of each struct in the pNext chain **must** be unique
- VUID-VkSamplerYcbcrConversionCreateInfo-format-parameter
  format **must** be a valid VkFormat value
- VUID-VkSamplerYcbcrConversionCreateInfo-ycbcrModel-parameter
  ycbcrModel **must** be a valid VkSamplerYcbcrModelConversion value
- VUID-VkSamplerYcbcrConversionCreateInfo-ycbcrRange-parameter
  ycbcrRange **must** be a valid VkSamplerYcbcrRange value
- VUID-VkSamplerYcbcrConversionCreateInfo-components-parameter
  components **must** be a valid VkComponentMapping structure
- VUID-VkSamplerYcbcrConversionCreateInfo-xChromaOffset-parameter
  xChromaOffset **must** be a valid VkChromaLocation value
- VUID-VkSamplerYcbcrConversionCreateInfo-yChromaOffset-parameter
  yChromaOffset **must** be a valid VkChromaLocation value
- VUID-VkSamplerYcbcrConversionCreateInfo-chromaFilter-parameter
  chromaFilter **must** be a valid VkFilter value

If chromaFilter is VK_FILTER_NEAREST, chroma samples are reconstructed to luma component resolution using nearest-neighbour sampling. Otherwise, chroma samples are reconstructed using interpolation. More details can be found in the description of sampler $Y'C_BC_R$ conversion in the Image Operations chapter.

VkSamplerYcbcrModelConversion defines the conversion from the source color model to the shader color model. Possible values are:

```
// Provided by VK_VERSION_1_1
typedef enum VkSamplerYcbcrModelConversion {
    VK_SAMPLER_YCBCR_MODEL_CONVERSION_RGB_IDENTITY = 0,
    VK_SAMPLER_YCBCR_MODEL_CONVERSION_YCBCR_IDENTITY = 1,
    VK_SAMPLER_YCBCR_MODEL_CONVERSION_YCBCR_709 = 2,
    VK_SAMPLER_YCBCR_MODEL_CONVERSION_YCBCR_601 = 3,
    VK_SAMPLER_YCBCR_MODEL_CONVERSION_YCBCR_2020 = 4,
  // Provided by VK_KHR_sampler_ycbcr_conversion
    VK_SAMPLER_YCBCR_MODEL_CONVERSION_RGB_IDENTITY_KHR =
VK_SAMPLER_YCBCR_MODEL_CONVERSION_RGB_IDENTITY,
  // Provided by VK_KHR_sampler_ycbcr_conversion
    VK_SAMPLER_YCBCR_MODEL_CONVERSION_YCBCR_IDENTITY_KHR =
VK_SAMPLER_YCBCR_MODEL_CONVERSION_YCBCR_IDENTITY,
  // Provided by VK_KHR_sampler_ycbcr_conversion
    VK_SAMPLER_YCBCR_MODEL_CONVERSION_YCBCR_709_KHR =
VK_SAMPLER_YCBCR_MODEL_CONVERSION_YCBCR_709,
  // Provided by VK_KHR_sampler_ycbcr_conversion
    VK_SAMPLER_YCBCR_MODEL_CONVERSION_YCBCR_601_KHR =
VK_SAMPLER_YCBCR_MODEL_CONVERSION_YCBCR_601,
  // Provided by VK_KHR_sampler_ycbcr_conversion
    VK_SAMPLER_YCBCR_MODEL_CONVERSION_YCBCR_2020_KHR =
VK_SAMPLER_YCBCR_MODEL_CONVERSION_YCBCR_2020,
} VkSamplerYcbcrModelConversion;
```

or the equivalent

```
// Provided by VK_KHR_sampler_ycbcr_conversion
typedef VkSamplerYcbcrModelConversion VkSamplerYcbcrModelConversionKHR;
```

- `VK_SAMPLER_YCBCR_MODEL_CONVERSION_RGB_IDENTITY` specifies that the input values to the conversion are unmodified.
- `VK_SAMPLER_YCBCR_MODEL_CONVERSION_YCBCR_IDENTITY` specifies no model conversion but the inputs are range expanded as for $Y'C_BC_R$.
- `VK_SAMPLER_YCBCR_MODEL_CONVERSION_YCBCR_709` specifies the color model conversion from $Y'C_BC_R$ to R'G'B' defined in BT.709 and described in the “BT.709 $Y'C_BC_R$ conversion” section of the Khronos Data Format Specification.
- `VK_SAMPLER_YCBCR_MODEL_CONVERSION_YCBCR_601` specifies the color model conversion from $Y'C_BC_R$ to R'G'B' defined in BT.601 and described in the “BT.601 $Y'C_BC_R$ conversion” section of the Khronos Data Format Specification.
- `VK_SAMPLER_YCBCR_MODEL_CONVERSION_YCBCR_2020` specifies the color model conversion from $Y'C_BC_R$ to R'G'B' defined in BT.2020 and described in the “BT.2020 $Y'C_BC_R$ conversion” section of the Khronos Data Format Specification.

In the `VK_SAMPLER_YCBCR_MODEL_CONVERSION_YCBCR_*` color models, for the input to the sampler $Y'C_BC_R$ range expansion and model conversion:

- the Y (Y' luma) component corresponds to the G component of an RGB image.
- the CB ($C_B$ or “U” blue color difference) component corresponds to the B component of an RGB image.
- the CR ($C_R$ or “V” red color difference) component corresponds to the R component of an RGB image.
- the alpha component, if present, is not modified by color model conversion.

These rules reflect the mapping of components after the component swizzle operation (controlled by VkSamplerYcbcrConversionCreateInfo::`components`).

**NOTE**

For example, an “YUVA” 32-bit format comprising four 8-bit components can be implemented as `VK_FORMAT_R8G8B8A8_UNORM` with a component mapping:

- `components.a` = `VK_COMPONENT_SWIZZLE_IDENTITY`
- `components.r` = `VK_COMPONENT_SWIZZLE_B`
- `components.g` = `VK_COMPONENT_SWIZZLE_R`
- `components.b` = `VK_COMPONENT_SWIZZLE_G`

The VkSamplerYcbcrRange enum describes whether color components are encoded using the full range of numerical values or whether values are reserved for headroom and foot room. VkSamplerYcbcrRange is defined as:

```
// Provided by VK_VERSION_1_1
typedef enum VkSamplerYcbcrRange {
    VK_SAMPLER_YCBCR_RANGE_ITU_FULL = 0,
    VK_SAMPLER_YCBCR_RANGE_ITU_NARROW = 1,
  // Provided by VK_KHR_sampler_ycbcr_conversion
    VK_SAMPLER_YCBCR_RANGE_ITU_FULL_KHR = VK_SAMPLER_YCBCR_RANGE_ITU_FULL,
  // Provided by VK_KHR_sampler_ycbcr_conversion
    VK_SAMPLER_YCBCR_RANGE_ITU_NARROW_KHR = VK_SAMPLER_YCBCR_RANGE_ITU_NARROW,
} VkSamplerYcbcrRange;
```

or the equivalent

```
// Provided by VK_KHR_sampler_ycbcr_conversion
typedef VkSamplerYcbcrRange VkSamplerYcbcrRangeKHR;
```

- `VK_SAMPLER_YCBCR_RANGE_ITU_FULL` specifies that the full range of the encoded values are valid and interpreted according to the ITU “full range” quantization rules.
- `VK_SAMPLER_YCBCR_RANGE_ITU_NARROW` specifies that headroom and foot room are reserved in the numerical range of encoded values, and the remaining values are expanded according to the ITU “narrow range” quantization rules.

The formulae for these conversions is described in the Sampler $Y'C_BC_R$ Range Expansion section of

the Image Operations chapter.

No range modification takes place if ycbcrModel is VK_SAMPLER_YCBCR_MODEL_CONVERSION_RGB_IDENTITY; the ycbcrRange field of VkSamplerYcbcrConversionCreateInfo is ignored in this case.

The VkChromaLocation enum defines the location of downsampled chroma component samples relative to the luma samples, and is defined as:

```
// Provided by VK_VERSION_1_1
typedef enum VkChromaLocation {
    VK_CHROMA_LOCATION_COSITED_EVEN = 0,
    VK_CHROMA_LOCATION_MIDPOINT = 1,
  // Provided by VK_KHR_sampler_ycbcr_conversion
    VK_CHROMA_LOCATION_COSITED_EVEN_KHR = VK_CHROMA_LOCATION_COSITED_EVEN,
  // Provided by VK_KHR_sampler_ycbcr_conversion
    VK_CHROMA_LOCATION_MIDPOINT_KHR = VK_CHROMA_LOCATION_MIDPOINT,
} VkChromaLocation;
```

or the equivalent

```
// Provided by VK_KHR_sampler_ycbcr_conversion
typedef VkChromaLocation VkChromaLocationKHR;
```

- VK_CHROMA_LOCATION_COSITED_EVEN specifies that downsampled chroma samples are aligned with luma samples with even coordinates.
- VK_CHROMA_LOCATION_MIDPOINT specifies that downsampled chroma samples are located half way between each even luma sample and the nearest higher odd luma sample.

Applications **can** enable sRGB to linear conversion for the R, G, and B components of a $Y'C_BC_R$ image during format conversion by including VkSamplerYcbcrConversionYcbcrDegammaCreateInfoQCOM structure in the pNext chain of VkSamplerYcbcrConversionCreateInfo.

The VkSamplerYcbcrConversionYcbcrDegammaCreateInfoQCOM structure is defined as:

```
// Provided by VK_QCOM_ycbcr_degamma
typedef struct VkSamplerYcbcrConversionYcbcrDegammaCreateInfoQCOM {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           enableYDegamma;
    VkBool32           enableCbCrDegamma;
} VkSamplerYcbcrConversionYcbcrDegammaCreateInfoQCOM;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- enableYDegamma indicates sRGB to linear conversion is enabled for the G component.

- enableCbCrDegamma indicates sRGB to linear conversion is enabled for the R and B components.

### Valid Usage (Implicit)

- VUID-VkSamplerYcbcrConversionYcbcrDegammaCreateInfoQCOM-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_SAMPLER_YCBCR_CONVERSION_YCBCR_DEGAMMA_CREATE_INFO_QCOM

To destroy a sampler $Y'C_BC_R$ conversion, call:

```
// Provided by VK_VERSION_1_1
void vkDestroySamplerYcbcrConversion(
    VkDevice                                    device,
    VkSamplerYcbcrConversion                    ycbcrConversion,
    const VkAllocationCallbacks*                pAllocator);
```

or the equivalent command

```
// Provided by VK_KHR_sampler_ycbcr_conversion
void vkDestroySamplerYcbcrConversionKHR(
    VkDevice                                    device,
    VkSamplerYcbcrConversion                    ycbcrConversion,
    const VkAllocationCallbacks*                pAllocator);
```

- device is the logical device that destroys the $Y'C_BC_R$ conversion.
- ycbcrConversion is the conversion to destroy.
- pAllocator controls host memory allocation as described in the Memory Allocation chapter.

### Valid Usage (Implicit)

- VUID-vkDestroySamplerYcbcrConversion-device-parameter
  device **must** be a valid VkDevice handle
- VUID-vkDestroySamplerYcbcrConversion-ycbcrConversion-parameter
  If ycbcrConversion is not VK_NULL_HANDLE, ycbcrConversion **must** be a valid VkSamplerYcbcrConversion handle
- VUID-vkDestroySamplerYcbcrConversion-pAllocator-parameter
  If pAllocator is not NULL, pAllocator **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkDestroySamplerYcbcrConversion-ycbcrConversion-parent
  If ycbcrConversion is a valid handle, it **must** have been created, allocated, or retrieved from device

## Host Synchronization

- Host access to `ycbcrConversion` **must** be externally synchronized

In addition to the predefined border color values, applications **can** provide a custom border color value by including the `VkSamplerCustomBorderColorCreateInfoEXT` structure in the VkSamplerCreateInfo::`pNext` chain.

The `VkSamplerCustomBorderColorCreateInfoEXT` structure is defined as:

```
// Provided by VK_EXT_custom_border_color
typedef struct VkSamplerCustomBorderColorCreateInfoEXT {
    VkStructureType      sType;
    const void*          pNext;
    VkClearColorValue    customBorderColor;
    VkFormat             format;
} VkSamplerCustomBorderColorCreateInfoEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `customBorderColor` is a VkClearColorValue representing the desired custom sampler border color.
- `format` is a VkFormat representing the format of the sampled image view(s). This field may be `VK_FORMAT_UNDEFINED` if the `customBorderColorWithoutFormat` feature is enabled.

**NOTE**

If `format` is a depth/stencil format, the aspect is determined by the value of VkSamplerCreateInfo::`borderColor`. If VkSamplerCreateInfo::`borderColor` is `VK_BORDER_COLOR_FLOAT_CUSTOM_EXT`, the depth aspect is considered. If VkSamplerCreateInfo::`borderColor` is `VK_BORDER_COLOR_INT_CUSTOM_EXT`, the stencil aspect is considered.

If `format` is `VK_FORMAT_UNDEFINED`, the VkSamplerCreateInfo::`borderColor` is `VK_BORDER_COLOR_INT_CUSTOM_EXT`, and the sampler is used with an image with a stencil format, then the implementation **must** source the custom border color from either the first or second components of VkSamplerCreateInfo::`customBorderColor` and **should** source it from the first component.

## Valid Usage

- VUID-VkSamplerCustomBorderColorCreateInfoEXT-format-07605
  If `format` is not `VK_FORMAT_UNDEFINED` and `format` is not a depth/stencil format then the VkSamplerCreateInfo::`borderColor` type **must** match the sampled type of the provided `format`, as shown in the *SPIR-V Type* column of the Interpretation of Numeric Format table
- VUID-VkSamplerCustomBorderColorCreateInfoEXT-format-04014

If the customBorderColorWithoutFormat feature is not enabled then format **must** not be VK_FORMAT_UNDEFINED

- VUID-VkSamplerCustomBorderColorCreateInfoEXT-format-04015
  If the sampler is used to sample an image view of VK_FORMAT_B4G4R4A4_UNORM_PACK16, VK_FORMAT_B5G6R5_UNORM_PACK16, VK_FORMAT_A1B5G5R5_UNORM_PACK16_KHR, or VK_FORMAT_B5G5R5A1_UNORM_PACK16 format then format **must** not be VK_FORMAT_UNDEFINED

### Valid Usage (Implicit)

- VUID-VkSamplerCustomBorderColorCreateInfoEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_SAMPLER_CUSTOM_BORDER_COLOR_CREATE_INFO_EXT
- VUID-VkSamplerCustomBorderColorCreateInfoEXT-format-parameter
  format **must** be a valid VkFormat value

If the sampler is created with VK_BORDER_COLOR_FLOAT_OPAQUE_BLACK, VK_BORDER_COLOR_INT_OPAQUE_BLACK, VK_BORDER_COLOR_FLOAT_CUSTOM_EXT, or VK_BORDER_COLOR_INT_CUSTOM_EXT borderColor, and that sampler will be combined with an image view that does not have an identity swizzle, and VkPhysicalDeviceBorderColorSwizzleFeaturesEXT::borderColorSwizzleFromImage is not enabled, then it is necessary to specify the component mapping of the border color, by including the VkSamplerBorderColorComponentMappingCreateInfoEXT structure in the VkSamplerCreateInfo::pNext chain, to get defined results.

The VkSamplerBorderColorComponentMappingCreateInfoEXT structure is defined as:

```
// Provided by VK_EXT_border_color_swizzle
typedef struct VkSamplerBorderColorComponentMappingCreateInfoEXT {
    VkStructureType       sType;
    const void*           pNext;
    VkComponentMapping    components;
    VkBool32              srgb;
} VkSamplerBorderColorComponentMappingCreateInfoEXT;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- components is a VkComponentMapping structure specifying a remapping of the border color components.
- srgb indicates that the sampler will be combined with an image view that has an image format which is sRGB encoded.

The VkComponentMapping components member describes a remapping from components of the border color to components of the vector returned by shader image instructions when the border color is used.

### Valid Usage

- VUID-VkSamplerBorderColorComponentMappingCreateInfoEXT-borderColorSwizzle-06437
  The `borderColorSwizzle` feature **must** be enabled

### Valid Usage (Implicit)

- VUID-VkSamplerBorderColorComponentMappingCreateInfoEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_SAMPLER_BORDER_COLOR_COMPONENT_MAPPING_CREATE_INFO_EXT`
- VUID-VkSamplerBorderColorComponentMappingCreateInfoEXT-components-parameter
  `components` **must** be a valid VkComponentMapping structure

The `VkSamplerBlockMatchWindowCreateInfoQCOM` structure is defined as:

```
// Provided by VK_QCOM_image_processing2
typedef struct VkSamplerBlockMatchWindowCreateInfoQCOM {
    VkStructureType                     sType;
    const void*                         pNext;
    VkExtent2D                          windowExtent;
    VkBlockMatchWindowCompareModeQCOM   windowCompareMode;
} VkSamplerBlockMatchWindowCreateInfoQCOM;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `windowExtent` is a VkExtent2D specifying a the width and height of the block match window.
- `windowCompareMode` is a VkBlockMatchWindowCompareModeQCOM specifying the compare mode.

### Valid Usage

- VUID-VkSamplerBlockMatchWindowCreateInfoQCOM-WindowExtent-09210
  `WindowExtent` **must** not be larger than VkPhysicalDeviceImageProcessing2PropertiesQCOM::`maxBlockMatchWindow`

### Valid Usage (Implicit)

- VUID-VkSamplerBlockMatchWindowCreateInfoQCOM-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_SAMPLER_BLOCK_MATCH_WINDOW_CREATE_INFO_QCOM`
- VUID-VkSamplerBlockMatchWindowCreateInfoQCOM-windowCompareMode-parameter
  `windowCompareMode` **must** be a valid VkBlockMatchWindowCompareModeQCOM value

The VkBlockMatchWindowCompareModeQCOM enum describes how block match values within the window are compared. VkBlockMatchWindowCompareModeQCOM is defined as:

```
// Provided by VK_QCOM_image_processing2
typedef enum VkBlockMatchWindowCompareModeQCOM {
    VK_BLOCK_MATCH_WINDOW_COMPARE_MODE_MIN_QCOM = 0,
    VK_BLOCK_MATCH_WINDOW_COMPARE_MODE_MAX_QCOM = 1,
} VkBlockMatchWindowCompareModeQCOM;
```

- VK_BLOCK_MATCH_WINDOW_COMPARE_MODE_MIN_QCOM specifies that windowed block match operations return the minimum error within the window.
- VK_BLOCK_MATCH_WINDOW_COMPARE_MODE_MAX_QCOM specifies that windowed block match operations return the maximum error within the window.

# Chapter 14. Resource Descriptors

A *descriptor* is an opaque data structure representing a shader resource such as a buffer, buffer view, image view, sampler, or combined image sampler. Descriptors are organized into *descriptor sets*, which are bound during command recording for use in subsequent drawing commands. The arrangement of content in each descriptor set is determined by a *descriptor set layout*, which determines what descriptors can be stored within it. The sequence of descriptor set layouts that **can** be used by a pipeline is specified in a *pipeline layout*. Each pipeline object **can** use up to `maxBoundDescriptorSets` (see Limits) descriptor sets.

If the `descriptorBuffer` feature is enabled, the implementation supports placing descriptors into descriptor buffers which are bound during command recording in a similar way to descriptor sets.

Shaders access resources via variables decorated with a descriptor set and binding number that link them to a descriptor in a descriptor set. The shader interface mapping to bound descriptor sets is described in the Shader Resource Interface section.

Shaders **can** also access buffers without going through descriptors by using Physical Storage Buffer Access to access them through 64-bit addresses.

## 14.1. Descriptor Types

There are a number of different types of descriptor supported by Vulkan, corresponding to different resources or usage. The following sections describe the API definitions of each descriptor type. The mapping of each type to SPIR-V is listed in the Shader Resource and Descriptor Type Correspondence and Shader Resource and Storage Class Correspondence tables in the Shader Interfaces chapter.

### 14.1.1. Storage Image

A *storage image* (`VK_DESCRIPTOR_TYPE_STORAGE_IMAGE`) is a descriptor type associated with an image resource via an image view that load, store, and atomic operations **can** be performed on.

Storage image loads are supported in all shader stages for image views whose format features contain `VK_FORMAT_FEATURE_STORAGE_IMAGE_BIT`.

Stores to storage images are supported in task, mesh and compute shaders for image views whose format features contain `VK_FORMAT_FEATURE_STORAGE_IMAGE_BIT`.

Atomic operations on storage images are supported in task, mesh and compute shaders for image views whose format features contain `VK_FORMAT_FEATURE_STORAGE_IMAGE_ATOMIC_BIT`.

When the `fragmentStoresAndAtomics` feature is enabled, stores and atomic operations are also supported for storage images in fragment shaders with the same set of image formats as supported in compute shaders. When the `vertexPipelineStoresAndAtomics` feature is enabled, stores and atomic operations are also supported in vertex, tessellation, and geometry shaders with the same set of image formats as supported in compute shaders.

The image subresources for a storage image **must** be in the `VK_IMAGE_LAYOUT_SHARED_PRESENT_KHR` or

VK_IMAGE_LAYOUT_GENERAL layout in order to access its data in a shader.

### 14.1.2. Sampler

A *sampler descriptor* (VK_DESCRIPTOR_TYPE_SAMPLER) is a descriptor type associated with a sampler object, used to control the behavior of sampling operations performed on a sampled image.

### 14.1.3. Sampled Image

A *sampled image* (VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE) is a descriptor type associated with an image resource via an image view that sampling operations **can** be performed on.

Shaders combine a sampled image variable and a sampler variable to perform sampling operations.

Sampled images are supported in all shader stages for image views whose format features contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_BIT.

An image subresources for a sampled image **must** be in one of the following layouts:

- VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL
- VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL
- VK_IMAGE_LAYOUT_GENERAL
- VK_IMAGE_LAYOUT_SHARED_PRESENT_KHR
- VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_STENCIL_ATTACHMENT_OPTIMAL
- VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_STENCIL_READ_ONLY_OPTIMAL
- VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL
- VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL
- VK_IMAGE_LAYOUT_READ_ONLY_OPTIMAL_KHR
- VK_IMAGE_LAYOUT_ATTACHMENT_FEEDBACK_LOOP_OPTIMAL_EXT

### 14.1.4. Combined Image Sampler

A *combined image sampler* (VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER) is a single descriptor type associated with both a sampler and an image resource, combining both a sampler and sampled image descriptor into a single descriptor.

If the descriptor refers to a sampler that performs $Y'C_BC_R$ conversion or samples a subsampled image, the sampler **must** only be used to sample the image in the same descriptor. Otherwise, the sampler and image in this type of descriptor **can** be used freely with any other samplers and images.

An image subresources for a combined image sampler **must** be in one of the following layouts:

- VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL
- VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL

- VK_IMAGE_LAYOUT_GENERAL
- VK_IMAGE_LAYOUT_SHARED_PRESENT_KHR
- VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_STENCIL_ATTACHMENT_OPTIMAL
- VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_STENCIL_READ_ONLY_OPTIMAL
- VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL
- VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL
- VK_IMAGE_LAYOUT_READ_ONLY_OPTIMAL_KHR
- VK_IMAGE_LAYOUT_ATTACHMENT_FEEDBACK_LOOP_OPTIMAL_EXT

| | |
|---|---|
| **NOTE** | On some implementations, it **may** be more efficient to sample from an image using a combination of sampler and sampled image that are stored together in the descriptor set in a combined descriptor. |

### 14.1.5. Uniform Texel Buffer

A *uniform texel buffer* (VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER) is a descriptor type associated with a buffer resource via a buffer view that image sampling operations **can** be performed on.

Uniform texel buffers define a tightly-packed 1-dimensional linear array of texels, with texels going through format conversion when read in a shader in the same way as they are for an image.

Load operations from uniform texel buffers are supported in all shader stages for buffer view formats which report format features support for VK_FORMAT_FEATURE_UNIFORM_TEXEL_BUFFER_BIT

### 14.1.6. Storage Texel Buffer

A *storage texel buffer* (VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER) is a descriptor type associated with a buffer resource via a buffer view that image load, store, and atomic operations **can** be performed on.

Storage texel buffers define a tightly-packed 1-dimensional linear array of texels, with texels going through format conversion when read in a shader in the same way as they are for an image. Unlike uniform texel buffers, these buffers can also be written to in the same way as for storage images.

Storage texel buffer loads are supported in all shader stages for texel buffer view formats which report format features support for VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_BIT

Stores to storage texel buffers are supported in task, mesh and compute shaders for texel buffer formats which report format features support for VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_BIT

Atomic operations on storage texel buffers are supported in task, mesh and compute shaders for texel buffer formats which report format features support for VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_ATOMIC_BIT

When the fragmentStoresAndAtomics feature is enabled, stores and atomic operations are also supported for storage texel buffers in fragment shaders with the same set of texel buffer formats as supported in compute shaders. When the vertexPipelineStoresAndAtomics feature is enabled, stores

and atomic operations are also supported in vertex, tessellation, and geometry shaders with the same set of texel buffer formats as supported in compute shaders.

### 14.1.7. Storage Buffer

A *storage buffer* (VK_DESCRIPTOR_TYPE_STORAGE_BUFFER) is a descriptor type associated with a buffer resource directly, described in a shader as a structure with various members that load, store, and atomic operations **can** be performed on.

| | |
|---|---|
| **NOTE** | Atomic operations **can** only be performed on members of certain types as defined in the SPIR-V environment appendix. |

### 14.1.8. Uniform Buffer

A *uniform buffer* (VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER) is a descriptor type associated with a buffer resource directly, described in a shader as a structure with various members that load operations **can** be performed on.

### 14.1.9. Dynamic Uniform Buffer

A *dynamic uniform buffer* (VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC) is almost identical to a uniform buffer, and differs only in how the offset into the buffer is specified. The base offset calculated by the VkDescriptorBufferInfo when initially updating the descriptor set is added to a dynamic offset when binding the descriptor set.

### 14.1.10. Dynamic Storage Buffer

A *dynamic storage buffer* (VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC) is almost identical to a storage buffer, and differs only in how the offset into the buffer is specified. The base offset calculated by the VkDescriptorBufferInfo when initially updating the descriptor set is added to a dynamic offset when binding the descriptor set.

### 14.1.11. Inline Uniform Block

An *inline uniform block* (VK_DESCRIPTOR_TYPE_INLINE_UNIFORM_BLOCK) is almost identical to a uniform buffer, and differs only in taking its storage directly from the encompassing descriptor set instead of being backed by buffer memory. It is typically used to access a small set of constant data that does not require the additional flexibility provided by the indirection enabled when using a uniform buffer where the descriptor and the referenced buffer memory are decoupled. Compared to push constants, they allow reusing the same set of constant data across multiple disjoint sets of drawing and dispatching commands.

Inline uniform block descriptors **cannot** be aggregated into arrays. Instead, the array size specified for an inline uniform block descriptor binding specifies the binding's capacity in bytes.

### 14.1.12. Sample Weight Image

A *sample weight image* (VK_DESCRIPTOR_TYPE_SAMPLE_WEIGHT_IMAGE_QCOM) is a descriptor type

associated with an image resource via an image view that **can** be used in weight image sampling. The image view **must** have been created with VkImageViewSampleWeightCreateInfoQCOM.

Shaders **can** combine a weight image variable, a sampled image variable, and a sampler variable to perform weight image sampling.

Weight image sampling is supported in all shader stages if the weight image view specifies a format that supports format feature `VK_FORMAT_FEATURE_2_WEIGHT_IMAGE_BIT_QCOM` and the sampled image view specifies a format that supports format feature `VK_FORMAT_FEATURE_2_WEIGHT_SAMPLED_IMAGE_BIT_QCOM`

The image subresources for the weight image **must** be in the `VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL`, or `VK_IMAGE_LAYOUT_GENERAL` layout in order to access its data in a shader.

### 14.1.13. Block Matching Image

A *block matching image* (`VK_DESCRIPTOR_TYPE_BLOCK_MATCH_IMAGE_QCOM`) is a descriptor type associated with an image resource via an image view that **can** be used in block matching.

Shaders **can** combine a target image variable, a reference image variable, and a sampler variable to perform block matching.

Block matching is supported in all shader stages for if both the target view and reference view specifies a format that supports format feature `VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM`

The image subresources for block matching **must** be in the `VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL`, or `VK_IMAGE_LAYOUT_GENERAL` layout in order to access its data in a shader.

### 14.1.14. Input Attachment

An *input attachment* (`VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT`) is a descriptor type associated with an image resource via an image view that **can** be used for framebuffer local load operations in fragment shaders.

All image formats that are supported for color attachments (`VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BIT` or `VK_FORMAT_FEATURE_2_LINEAR_COLOR_ATTACHMENT_BIT_NV` ) or depth/stencil attachments (`VK_FORMAT_FEATURE_DEPTH_STENCIL_ATTACHMENT_BIT`) for a given image tiling mode are also supported for input attachments.

An image view used as an input attachment **must** be in one of the following layouts:

- `VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL`
- `VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL`
- `VK_IMAGE_LAYOUT_GENERAL`
- `VK_IMAGE_LAYOUT_SHARED_PRESENT_KHR`
- `VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_STENCIL_ATTACHMENT_OPTIMAL`

- VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_STENCIL_READ_ONLY_OPTIMAL
- VK_IMAGE_LAYOUT_READ_ONLY_OPTIMAL_KHR
- VK_IMAGE_LAYOUT_ATTACHMENT_FEEDBACK_LOOP_OPTIMAL_EXT
- VK_IMAGE_LAYOUT_RENDERING_LOCAL_READ_KHR

### 14.1.15. Acceleration Structure

An *acceleration structure* ( VK_DESCRIPTOR_TYPE_ACCELERATION_STRUCTURE_KHR or VK_DESCRIPTOR_TYPE_ACCELERATION_STRUCTURE_NV ) is a descriptor type that is used to retrieve scene geometry from within shaders that are used for ray traversal. Shaders have read-only access to the memory.

### 14.1.16. Mutable

A descriptor of *mutable* (VK_DESCRIPTOR_TYPE_MUTABLE_EXT) type indicates that this descriptor **can** mutate to any of the descriptor types given in the VkMutableDescriptorTypeListEXT::pDescriptorTypes list of descriptor types in the pNext chain of VkDescriptorSetLayoutCreateInfo for this binding. At any point, each individual descriptor of mutable type has an active descriptor type. The active descriptor type **can** be any one of the declared types in pDescriptorTypes. Additionally, a mutable descriptor's active descriptor type **can** be of the VK_DESCRIPTOR_TYPE_MUTABLE_EXT type, which is the initial active descriptor type. The active descriptor type **can** change when the descriptor is updated. When a descriptor is consumed by binding a descriptor set, the active descriptor type is considered, not VK_DESCRIPTOR_TYPE_MUTABLE_EXT.

An active descriptor type of VK_DESCRIPTOR_TYPE_MUTABLE_EXT is considered an undefined descriptor. If a descriptor is consumed where the active descriptor type does not match what the shader expects, the descriptor is considered an undefined descriptor.

**NOTE** To find which descriptor types are supported as VK_DESCRIPTOR_TYPE_MUTABLE_EXT, the application **can** use vkGetDescriptorSetLayoutSupport with a VK_DESCRIPTOR_TYPE_MUTABLE_EXT binding, with the list of descriptor types to query in the VkMutableDescriptorTypeCreateInfoEXT::pDescriptorTypes array for that binding.

**NOTE** The intention of a mutable descriptor type is that implementations allocate N bytes per descriptor, where N is determined by the maximum descriptor size for a given descriptor binding. Implementations are not expected to keep track of the active descriptor type, and it should be considered a C-like union type.

A mutable descriptor type is not considered as efficient in terms of runtime performance as using a non-mutable descriptor type, and applications are not encouraged to use them outside API layering efforts. Mutable descriptor types can be more efficient if the alternative is using many different descriptors to emulate mutable descriptor types.

# 14.2. Descriptor Sets

Descriptors are grouped together into descriptor set objects. A descriptor set object is an opaque object containing storage for a set of descriptors, where the types and number of descriptors is defined by a descriptor set layout. The layout object **may** be used to define the association of each descriptor binding with memory or other implementation resources. The layout is used both for determining the resources that need to be associated with the descriptor set, and determining the interface between shader stages and shader resources.

## 14.2.1. Descriptor Set Layout

A descriptor set layout object is defined by an array of zero or more descriptor bindings. Each individual descriptor binding is specified by a descriptor type, a count (array size) of the number of descriptors in the binding, a set of shader stages that **can** access the binding, and (if using immutable samplers) an array of sampler descriptors.

Descriptor set layout objects are represented by `VkDescriptorSetLayout` handles:

```
// Provided by VK_VERSION_1_0
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkDescriptorSetLayout)
```

To create descriptor set layout objects, call:

```
// Provided by VK_VERSION_1_0
VkResult vkCreateDescriptorSetLayout(
    VkDevice                                    device,
    const VkDescriptorSetLayoutCreateInfo*      pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkDescriptorSetLayout*                      pSetLayout);
```

- `device` is the logical device that creates the descriptor set layout.
- `pCreateInfo` is a pointer to a VkDescriptorSetLayoutCreateInfo structure specifying the state of the descriptor set layout object.
- `pAllocator` controls host memory allocation as described in the Memory Allocation chapter.
- `pSetLayout` is a pointer to a VkDescriptorSetLayout handle in which the resulting descriptor set layout object is returned.

### Valid Usage

- VUID-vkCreateDescriptorSetLayout-support-09582
  If the descriptor layout exceeds the limits reported through the physical device limits, then vkGetDescriptorSetLayoutSupport **must** have returned VkDescriptorSetLayoutSupport with `support` equal to `VK_TRUE` for `pCreateInfo`

## Valid Usage (Implicit)

- VUID-vkCreateDescriptorSetLayout-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkCreateDescriptorSetLayout-pCreateInfo-parameter
  `pCreateInfo` **must** be a valid pointer to a valid VkDescriptorSetLayoutCreateInfo structure
- VUID-vkCreateDescriptorSetLayout-pAllocator-parameter
  If `pAllocator` is not `NULL`, `pAllocator` **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkCreateDescriptorSetLayout-pSetLayout-parameter
  `pSetLayout` **must** be a valid pointer to a VkDescriptorSetLayout handle

## Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`

Information about the descriptor set layout is passed in a `VkDescriptorSetLayoutCreateInfo` structure:

```
// Provided by VK_VERSION_1_0
typedef struct VkDescriptorSetLayoutCreateInfo {
    VkStructureType                         sType;
    const void*                             pNext;
    VkDescriptorSetLayoutCreateFlags        flags;
    uint32_t                                bindingCount;
    const VkDescriptorSetLayoutBinding*     pBindings;
} VkDescriptorSetLayoutCreateInfo;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is a bitmask of VkDescriptorSetLayoutCreateFlagBits specifying options for descriptor set layout creation.
- `bindingCount` is the number of elements in `pBindings`.
- `pBindings` is a pointer to an array of VkDescriptorSetLayoutBinding structures.

## Valid Usage

- VUID-VkDescriptorSetLayoutCreateInfo-binding-00279
  If the `perStageDescriptorSet` feature is not enabled, or `flags` does not contain `VK_DESCRIPTOR_SET_LAYOUT_CREATE_PER_STAGE_BIT_NV`, then the VkDescriptorSetLayoutBinding::`binding` members of the elements of the `pBindings` array **must** each have different values
- VUID-VkDescriptorSetLayoutCreateInfo-flags-00280
  If `flags` contains `VK_DESCRIPTOR_SET_LAYOUT_CREATE_PUSH_DESCRIPTOR_BIT_KHR`, then all elements of `pBindings` **must** not have a `descriptorType` of `VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC` or `VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC`
- VUID-VkDescriptorSetLayoutCreateInfo-flags-02208
  If `flags` contains `VK_DESCRIPTOR_SET_LAYOUT_CREATE_PUSH_DESCRIPTOR_BIT_KHR`, then all elements of `pBindings` **must** not have a `descriptorType` of `VK_DESCRIPTOR_TYPE_INLINE_UNIFORM_BLOCK`
- VUID-VkDescriptorSetLayoutCreateInfo-flags-00281
  If `flags` contains `VK_DESCRIPTOR_SET_LAYOUT_CREATE_PUSH_DESCRIPTOR_BIT_KHR`, then the total number of elements of all bindings **must** be less than or equal to VkPhysicalDevicePushDescriptorPropertiesKHR::`maxPushDescriptors`
- VUID-VkDescriptorSetLayoutCreateInfo-flags-04590
  If `flags` contains `VK_DESCRIPTOR_SET_LAYOUT_CREATE_PUSH_DESCRIPTOR_BIT_KHR`, `flags` **must** not contain `VK_DESCRIPTOR_SET_LAYOUT_CREATE_HOST_ONLY_POOL_BIT_EXT`
- VUID-VkDescriptorSetLayoutCreateInfo-flags-04591
  If `flags` contains `VK_DESCRIPTOR_SET_LAYOUT_CREATE_PUSH_DESCRIPTOR_BIT_KHR`, `pBindings` **must** not have a `descriptorType` of `VK_DESCRIPTOR_TYPE_MUTABLE_EXT`
- VUID-VkDescriptorSetLayoutCreateInfo-flags-03000
  If any binding has the `VK_DESCRIPTOR_BINDING_UPDATE_AFTER_BIND_BIT` bit set, `flags` **must** include `VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT`
- VUID-VkDescriptorSetLayoutCreateInfo-descriptorType-03001
  If any binding has the `VK_DESCRIPTOR_BINDING_UPDATE_AFTER_BIND_BIT` bit set, then all bindings **must** not have `descriptorType` of `VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC` or `VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC`
- VUID-VkDescriptorSetLayoutCreateInfo-flags-04592
  If `flags` contains `VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT`, `flags` **must** not contain `VK_DESCRIPTOR_SET_LAYOUT_CREATE_HOST_ONLY_POOL_BIT_EXT`
- VUID-VkDescriptorSetLayoutCreateInfo-pBindings-07303
  If any element `pBindings`[i] has a `descriptorType` of `VK_DESCRIPTOR_TYPE_MUTABLE_EXT`, then the `pNext` chain **must** include a VkMutableDescriptorTypeCreateInfoEXT structure, and `mutableDescriptorTypeListCount` **must** be greater than i
- VUID-VkDescriptorSetLayoutCreateInfo-descriptorType-04594
  If a binding has a `descriptorType` value of `VK_DESCRIPTOR_TYPE_MUTABLE_EXT`, then `pImmutableSamplers` **must** be `NULL`
- VUID-VkDescriptorSetLayoutCreateInfo-mutableDescriptorType-04595

If VkPhysicalDeviceMutableDescriptorTypeFeaturesEXT::mutableDescriptorType is not enabled, pBindings **must** not contain a descriptorType of VK_DESCRIPTOR_TYPE_MUTABLE_EXT

- VUID-VkDescriptorSetLayoutCreateInfo-flags-04596
  If flags contains VK_DESCRIPTOR_SET_LAYOUT_CREATE_HOST_ONLY_POOL_BIT_EXT, VkPhysicalDeviceMutableDescriptorTypeFeaturesEXT::mutableDescriptorType **must** be enabled
- VUID-VkDescriptorSetLayoutCreateInfo-flags-08000
  If flags contains VK_DESCRIPTOR_SET_LAYOUT_CREATE_DESCRIPTOR_BUFFER_BIT_EXT, then all elements of pBindings **must** not have a descriptorType of VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC or VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC
- VUID-VkDescriptorSetLayoutCreateInfo-flags-08001
  If flags contains VK_DESCRIPTOR_SET_LAYOUT_CREATE_EMBEDDED_IMMUTABLE_SAMPLERS_BIT_EXT, flags **must** also contain VK_DESCRIPTOR_SET_LAYOUT_CREATE_DESCRIPTOR_BUFFER_BIT_EXT
- VUID-VkDescriptorSetLayoutCreateInfo-flags-08002
  If flags contains VK_DESCRIPTOR_SET_LAYOUT_CREATE_DESCRIPTOR_BUFFER_BIT_EXT, then flags **must** not contain VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT
- VUID-VkDescriptorSetLayoutCreateInfo-flags-08003
  If flags contains VK_DESCRIPTOR_SET_LAYOUT_CREATE_DESCRIPTOR_BUFFER_BIT_EXT, then flags **must** not contain VK_DESCRIPTOR_SET_LAYOUT_CREATE_HOST_ONLY_POOL_BIT_VALVE
- VUID-VkDescriptorSetLayoutCreateInfo-flags-09463
  If flags contains VK_DESCRIPTOR_SET_LAYOUT_CREATE_PER_STAGE_BIT_NV, then perStageDescriptorSet **must** be enabled
- VUID-VkDescriptorSetLayoutCreateInfo-flags-09464
  If flags contains VK_DESCRIPTOR_SET_LAYOUT_CREATE_PER_STAGE_BIT_NV, then there **must** not be any two elements of the pBindings array with the same VkDescriptorSetLayoutBinding::binding value and their VkDescriptorSetLayoutBinding::stageFlags containing the same bit

## Valid Usage (Implicit)

- VUID-VkDescriptorSetLayoutCreateInfo-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_DESCRIPTOR_SET_LAYOUT_CREATE_INFO
- VUID-VkDescriptorSetLayoutCreateInfo-pNext-pNext
  Each pNext member of any structure (including this one) in the pNext chain **must** be either NULL or a pointer to a valid instance of VkDescriptorSetLayoutBindingFlagsCreateInfo or VkMutableDescriptorTypeCreateInfoEXT
- VUID-VkDescriptorSetLayoutCreateInfo-sType-unique
  The sType value of each struct in the pNext chain **must** be unique
- VUID-VkDescriptorSetLayoutCreateInfo-flags-parameter
  flags **must** be a valid combination of VkDescriptorSetLayoutCreateFlagBits values
- VUID-VkDescriptorSetLayoutCreateInfo-pBindings-parameter
  If bindingCount is not 0, pBindings **must** be a valid pointer to an array of bindingCount valid

VkDescriptorSetLayoutBinding structures

If the pNext chain of a VkDescriptorSetLayoutCreateInfo or VkDescriptorPoolCreateInfo structure includes a VkMutableDescriptorTypeCreateInfoEXT structure, then that structure specifies Information about the possible descriptor types for mutable descriptor types.

The VkMutableDescriptorTypeCreateInfoEXT structure is defined as:

```
// Provided by VK_EXT_mutable_descriptor_type
typedef struct VkMutableDescriptorTypeCreateInfoEXT {
    VkStructureType                          sType;
    const void*                              pNext;
    uint32_t                                 mutableDescriptorTypeListCount;
    const VkMutableDescriptorTypeListEXT*    pMutableDescriptorTypeLists;
} VkMutableDescriptorTypeCreateInfoEXT;
```

or the equivalent

```
// Provided by VK_VALVE_mutable_descriptor_type
typedef VkMutableDescriptorTypeCreateInfoEXT VkMutableDescriptorTypeCreateInfoVALVE;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- mutableDescriptorTypeListCount is the number of elements in pMutableDescriptorTypeLists.
- pMutableDescriptorTypeLists is a pointer to an array of VkMutableDescriptorTypeListEXT structures.

If mutableDescriptorTypeListCount is zero or if this structure is not included in the pNext chain, the VkMutableDescriptorTypeListEXT for each element is considered to be zero or NULL for each member. Otherwise, the descriptor set layout binding at VkDescriptorSetLayoutCreateInfo::pBindings[i] uses the descriptor type lists in VkMutableDescriptorTypeCreateInfoEXT::pMutableDescriptorTypeLists[i].

### Valid Usage (Implicit)

- VUID-VkMutableDescriptorTypeCreateInfoEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_MUTABLE_DESCRIPTOR_TYPE_CREATE_INFO_EXT
- VUID-VkMutableDescriptorTypeCreateInfoEXT-pMutableDescriptorTypeLists-parameter
  If mutableDescriptorTypeListCount is not 0, pMutableDescriptorTypeLists **must** be a valid pointer to an array of mutableDescriptorTypeListCount valid VkMutableDescriptorTypeListEXT structures

The list of potential descriptor types a given mutable descriptor **can** mutate to is passed in a VkMutableDescriptorTypeListEXT structure.

The VkMutableDescriptorTypeListEXT structure is defined as:

```
// Provided by VK_EXT_mutable_descriptor_type
typedef struct VkMutableDescriptorTypeListEXT {
    uint32_t                    descriptorTypeCount;
    const VkDescriptorType*     pDescriptorTypes;
} VkMutableDescriptorTypeListEXT;
```

or the equivalent

```
// Provided by VK_VALVE_mutable_descriptor_type
typedef VkMutableDescriptorTypeListEXT VkMutableDescriptorTypeListVALVE;
```

- descriptorTypeCount is the number of elements in pDescriptorTypes.
- pDescriptorTypes is NULL or a pointer to an array of descriptorTypeCount VkDescriptorType values defining which descriptor types a given binding may mutate to.

## Valid Usage

- VUID-VkMutableDescriptorTypeListEXT-descriptorTypeCount-04597
  descriptorTypeCount **must** not be 0 if the corresponding binding is of VK_DESCRIPTOR_TYPE_MUTABLE_EXT
- VUID-VkMutableDescriptorTypeListEXT-pDescriptorTypes-04598
  pDescriptorTypes **must** be a valid pointer to an array of descriptorTypeCount valid, unique VkDescriptorType values if the given binding is of VK_DESCRIPTOR_TYPE_MUTABLE_EXT type
- VUID-VkMutableDescriptorTypeListEXT-descriptorTypeCount-04599
  descriptorTypeCount **must** be 0 if the corresponding binding is not of VK_DESCRIPTOR_TYPE_MUTABLE_EXT
- VUID-VkMutableDescriptorTypeListEXT-pDescriptorTypes-04600
  pDescriptorTypes **must** not contain VK_DESCRIPTOR_TYPE_MUTABLE_EXT
- VUID-VkMutableDescriptorTypeListEXT-pDescriptorTypes-04601
  pDescriptorTypes **must** not contain VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC
- VUID-VkMutableDescriptorTypeListEXT-pDescriptorTypes-04602
  pDescriptorTypes **must** not contain VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC
- VUID-VkMutableDescriptorTypeListEXT-pDescriptorTypes-04603
  pDescriptorTypes **must** not contain VK_DESCRIPTOR_TYPE_INLINE_UNIFORM_BLOCK

## Valid Usage (Implicit)

- VUID-VkMutableDescriptorTypeListEXT-pDescriptorTypes-parameter
  If descriptorTypeCount is not 0, pDescriptorTypes **must** be a valid pointer to an array of descriptorTypeCount valid VkDescriptorType values

Bits which **can** be set in VkDescriptorSetLayoutCreateInfo::flags, specifying options for descriptor set layout, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkDescriptorSetLayoutCreateFlagBits {
    VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT = 0x00000002,
  // Provided by VK_KHR_push_descriptor
    VK_DESCRIPTOR_SET_LAYOUT_CREATE_PUSH_DESCRIPTOR_BIT_KHR = 0x00000001,
  // Provided by VK_EXT_descriptor_buffer
    VK_DESCRIPTOR_SET_LAYOUT_CREATE_DESCRIPTOR_BUFFER_BIT_EXT = 0x00000010,
  // Provided by VK_EXT_descriptor_buffer
    VK_DESCRIPTOR_SET_LAYOUT_CREATE_EMBEDDED_IMMUTABLE_SAMPLERS_BIT_EXT = 0x00000020,
  // Provided by VK_NV_device_generated_commands_compute
    VK_DESCRIPTOR_SET_LAYOUT_CREATE_INDIRECT_BINDABLE_BIT_NV = 0x00000080,
  // Provided by VK_EXT_mutable_descriptor_type
    VK_DESCRIPTOR_SET_LAYOUT_CREATE_HOST_ONLY_POOL_BIT_EXT = 0x00000004,
  // Provided by VK_NV_per_stage_descriptor_set
    VK_DESCRIPTOR_SET_LAYOUT_CREATE_PER_STAGE_BIT_NV = 0x00000040,
  // Provided by VK_EXT_descriptor_indexing
    VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT_EXT =
VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT,
  // Provided by VK_VALVE_mutable_descriptor_type
    VK_DESCRIPTOR_SET_LAYOUT_CREATE_HOST_ONLY_POOL_BIT_VALVE =
VK_DESCRIPTOR_SET_LAYOUT_CREATE_HOST_ONLY_POOL_BIT_EXT,
} VkDescriptorSetLayoutCreateFlagBits;
```

- VK_DESCRIPTOR_SET_LAYOUT_CREATE_PUSH_DESCRIPTOR_BIT_KHR specifies that descriptor sets **must** not be allocated using this layout, and descriptors are instead pushed by vkCmdPushDescriptorSetKHR.
- VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT specifies that descriptor sets using this layout **must** be allocated from a descriptor pool created with the VK_DESCRIPTOR_POOL_CREATE_UPDATE_AFTER_BIND_BIT bit set. Descriptor set layouts created with this bit set have alternate limits for the maximum number of descriptors per-stage and per-pipeline layout. The non-UpdateAfterBind limits only count descriptors in sets created without this flag. The UpdateAfterBind limits count all descriptors, but the limits **may** be higher than the non-UpdateAfterBind limits.
- VK_DESCRIPTOR_SET_LAYOUT_CREATE_INDIRECT_BINDABLE_BIT_NV specifies that descriptor sets using this layout allows them to be bound with compute pipelines that are created with VK_PIPELINE_CREATE_INDIRECT_BINDABLE_BIT_NV flag set to be used in Device-Generated Commands.
- VK_DESCRIPTOR_SET_LAYOUT_CREATE_DESCRIPTOR_BUFFER_BIT_EXT specifies that this layout **must** only be used with descriptor buffers.
- VK_DESCRIPTOR_SET_LAYOUT_CREATE_EMBEDDED_IMMUTABLE_SAMPLERS_BIT_EXT specifies that this is a layout only containing immutable samplers that **can** be bound by vkCmdBindDescriptorBufferEmbeddedSamplersEXT. Unlike normal immutable samplers, embedded immutable samplers do not require the application to provide them in a descriptor buffer.

- VK_DESCRIPTOR_SET_LAYOUT_CREATE_HOST_ONLY_POOL_BIT_EXT specifies that descriptor sets using this layout **must** be allocated from a descriptor pool created with the VK_DESCRIPTOR_POOL_CREATE_HOST_ONLY_BIT_EXT bit set. Descriptor set layouts created with this bit have no expressible limit for maximum number of descriptors per-stage. Host descriptor sets are limited only by available host memory, but **may** be limited for implementation specific reasons. Implementations **may** limit the number of supported descriptors to UpdateAfterBind limits or non-UpdateAfterBind limits, whichever is larger.
- VK_DESCRIPTOR_SET_LAYOUT_CREATE_PER_STAGE_BIT_NV specifies that binding numbers in descriptor sets using this layout **may** represent different resources and/or types of resources in each stage.

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkDescriptorSetLayoutCreateFlags;
```

VkDescriptorSetLayoutCreateFlags is a bitmask type for setting a mask of zero or more VkDescriptorSetLayoutCreateFlagBits.

The VkDescriptorSetLayoutBinding structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkDescriptorSetLayoutBinding {
    uint32_t              binding;
    VkDescriptorType      descriptorType;
    uint32_t              descriptorCount;
    VkShaderStageFlags    stageFlags;
    const VkSampler*      pImmutableSamplers;
} VkDescriptorSetLayoutBinding;
```

- binding is the binding number of this entry and corresponds to a resource of the same binding number in the shader stages.
- descriptorType is a VkDescriptorType specifying which type of resource descriptors are used for this binding.
- descriptorCount is the number of descriptors contained in the binding, accessed in a shader as an array, except if descriptorType is VK_DESCRIPTOR_TYPE_INLINE_UNIFORM_BLOCK in which case descriptorCount is the size in bytes of the inline uniform block. If descriptorCount is zero this binding entry is reserved and the resource **must** not be accessed from any stage via this binding within any pipeline using the set layout.
- stageFlags member is a bitmask of VkShaderStageFlagBits specifying which pipeline shader stages **can** access a resource for this binding. VK_SHADER_STAGE_ALL is a shorthand specifying that all defined shader stages, including any additional stages defined by extensions, **can** access the resource.

  If a shader stage is not included in stageFlags, then a resource **must** not be accessed from that stage via this binding within any pipeline using the set layout. Other than input attachments which are limited to the fragment shader, there are no limitations on what combinations of stages **can** use a descriptor binding, and in particular a binding **can** be used by both graphics

stages and the compute stage.

- `pImmutableSamplers` affects initialization of samplers. If `descriptorType` specifies a `VK_DESCRIPTOR_TYPE_SAMPLER` or `VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER` type descriptor, then `pImmutableSamplers` **can** be used to initialize a set of *immutable samplers*. Immutable samplers are permanently bound into the set layout and **must** not be changed; updating a `VK_DESCRIPTOR_TYPE_SAMPLER` descriptor with immutable samplers is not allowed and updates to a `VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER` descriptor with immutable samplers does not modify the samplers (the image views are updated, but the sampler updates are ignored). If `pImmutableSamplers` is not `NULL`, then it is a pointer to an array of sampler handles that will be copied into the set layout and used for the corresponding binding. Only the sampler handles are copied; the sampler objects **must** not be destroyed before the final use of the set layout and any descriptor pools and sets created using it. If `pImmutableSamplers` is `NULL`, then the sampler slots are dynamic and sampler handles **must** be bound into descriptor sets using this layout. If `descriptorType` is not one of these descriptor types, then `pImmutableSamplers` is ignored.

The above layout definition allows the descriptor bindings to be specified sparsely such that not all binding numbers between 0 and the maximum binding number need to be specified in the `pBindings` array. Bindings that are not specified have a `descriptorCount` and `stageFlags` of zero, and the value of `descriptorType` is undefined. However, all binding numbers between 0 and the maximum binding number in the VkDescriptorSetLayoutCreateInfo::`pBindings` array **may** consume memory in the descriptor set layout even if not all descriptor bindings are used, though it **should** not consume additional memory from the descriptor pool.

| | |
|---|---|
| **NOTE** | The maximum binding number specified **should** be as compact as possible to avoid wasted memory. |

## Valid Usage

- VUID-VkDescriptorSetLayoutBinding-descriptorType-00282
  If `descriptorType` is `VK_DESCRIPTOR_TYPE_SAMPLER` or `VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER`, and `descriptorCount` is not `0` and `pImmutableSamplers` is not `NULL`, `pImmutableSamplers` **must** be a valid pointer to an array of `descriptorCount` valid `VkSampler` handles
- VUID-VkDescriptorSetLayoutBinding-descriptorType-04604
  If the `inlineUniformBlock` feature is not enabled, `descriptorType` **must** not be `VK_DESCRIPTOR_TYPE_INLINE_UNIFORM_BLOCK`
- VUID-VkDescriptorSetLayoutBinding-descriptorType-02209
  If `descriptorType` is `VK_DESCRIPTOR_TYPE_INLINE_UNIFORM_BLOCK` then `descriptorCount` **must** be a multiple of `4`
- VUID-VkDescriptorSetLayoutBinding-descriptorType-08004
  If `descriptorType` is `VK_DESCRIPTOR_TYPE_INLINE_UNIFORM_BLOCK` and VkDescriptorSetLayoutCreateInfo::`flags` does not contain `VK_DESCRIPTOR_SET_LAYOUT_CREATE_DESCRIPTOR_BUFFER_BIT_EXT` then `descriptorCount` **must** be less than or equal to `VkPhysicalDeviceInlineUniformBlockProperties::maxInlineUniformBlockSize`

- VUID-VkDescriptorSetLayoutBinding-flags-08005
  If VkDescriptorSetLayoutCreateInfo::flags contains VK_DESCRIPTOR_SET_LAYOUT_CREATE_EMBEDDED_IMMUTABLE_SAMPLERS_BIT_EXT, descriptorType **must** be VK_DESCRIPTOR_TYPE_SAMPLER
- VUID-VkDescriptorSetLayoutBinding-flags-08006
  If VkDescriptorSetLayoutCreateInfo::flags contains VK_DESCRIPTOR_SET_LAYOUT_CREATE_EMBEDDED_IMMUTABLE_SAMPLERS_BIT_EXT, descriptorCount **must** less than or equal to 1
- VUID-VkDescriptorSetLayoutBinding-flags-08007
  If VkDescriptorSetLayoutCreateInfo::flags contains VK_DESCRIPTOR_SET_LAYOUT_CREATE_EMBEDDED_IMMUTABLE_SAMPLERS_BIT_EXT, and descriptorCount is equal to 1, pImmutableSamplers **must** not be NULL
- VUID-VkDescriptorSetLayoutBinding-descriptorCount-09465
  If descriptorCount is not 0, stageFlags **must** be VK_SHADER_STAGE_ALL or a valid combination of other VkShaderStageFlagBits values
- VUID-VkDescriptorSetLayoutBinding-descriptorType-01510
  If descriptorType is VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT and descriptorCount is not 0, then stageFlags **must** be 0 or VK_SHADER_STAGE_FRAGMENT_BIT
- VUID-VkDescriptorSetLayoutBinding-pImmutableSamplers-04009
  The sampler objects indicated by pImmutableSamplers **must** not have a borderColor with one of the values VK_BORDER_COLOR_FLOAT_CUSTOM_EXT or VK_BORDER_COLOR_INT_CUSTOM_EXT
- VUID-VkDescriptorSetLayoutBinding-descriptorType-04605
  If descriptorType is VK_DESCRIPTOR_TYPE_MUTABLE_EXT, then pImmutableSamplers **must** be NULL
- VUID-VkDescriptorSetLayoutBinding-flags-09466
  If VkDescriptorSetLayoutCreateInfo::flags contains VK_DESCRIPTOR_SET_LAYOUT_CREATE_PER_STAGE_BIT_NV, and descriptorCount is not 0, then stageFlags **must** be a valid combination of VK_SHADER_STAGE_VERTEX_BIT, VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT, VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT, VK_SHADER_STAGE_GEOMETRY_BIT, VK_SHADER_STAGE_FRAGMENT_BIT and VK_SHADER_STAGE_COMPUTE_BIT values

### Valid Usage (Implicit)

- VUID-VkDescriptorSetLayoutBinding-descriptorType-parameter
  descriptorType **must** be a valid VkDescriptorType value

If the pNext chain of a VkDescriptorSetLayoutCreateInfo structure includes a VkDescriptorSetLayoutBindingFlagsCreateInfo structure, then that structure includes an array of flags, one for each descriptor set layout binding.

The VkDescriptorSetLayoutBindingFlagsCreateInfo structure is defined as:

```
typedef struct VkDescriptorSetLayoutBindingFlagsCreateInfo {
    VkStructureType                    sType;
    const void*                        pNext;
    uint32_t                           bindingCount;
    const VkDescriptorBindingFlags*    pBindingFlags;
} VkDescriptorSetLayoutBindingFlagsCreateInfo;
```

or the equivalent

```
// Provided by VK_EXT_descriptor_indexing
typedef VkDescriptorSetLayoutBindingFlagsCreateInfo
VkDescriptorSetLayoutBindingFlagsCreateInfoEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `bindingCount` is zero or the number of elements in `pBindingFlags`.
- `pBindingFlags` is a pointer to an array of VkDescriptorBindingFlags bitfields, one for each descriptor set layout binding.

If `bindingCount` is zero or if this structure is not included in the `pNext` chain, the VkDescriptorBindingFlags for each descriptor set layout binding is considered to be zero. Otherwise, the descriptor set layout binding at VkDescriptorSetLayoutCreateInfo::`pBindings`[i] uses the flags in `pBindingFlags`[i].

## Valid Usage

- VUID-VkDescriptorSetLayoutBindingFlagsCreateInfo-bindingCount-03002
  If `bindingCount` is not zero, `bindingCount` **must** equal VkDescriptorSetLayoutCreateInfo::`bindingCount`
- VUID-VkDescriptorSetLayoutBindingFlagsCreateInfo-flags-03003
  If VkDescriptorSetLayoutCreateInfo::`flags` includes `VK_DESCRIPTOR_SET_LAYOUT_CREATE_PUSH_DESCRIPTOR_BIT_KHR`, then all elements of `pBindingFlags` **must** not include `VK_DESCRIPTOR_BINDING_UPDATE_AFTER_BIND_BIT`, `VK_DESCRIPTOR_BINDING_UPDATE_UNUSED_WHILE_PENDING_BIT`, or `VK_DESCRIPTOR_BINDING_VARIABLE_DESCRIPTOR_COUNT_BIT`
- VUID-VkDescriptorSetLayoutBindingFlagsCreateInfo-pBindingFlags-03004
  If an element of `pBindingFlags` includes `VK_DESCRIPTOR_BINDING_VARIABLE_DESCRIPTOR_COUNT_BIT`, then it **must** be the element with the highest `binding` number
- VUID-VkDescriptorSetLayoutBindingFlagsCreateInfo-descriptorBindingUniformBufferUpdateAfterBind-03005
  If VkPhysicalDeviceDescriptorIndexingFeatures::`descriptorBindingUniformBufferUpdateAfterBind` is not enabled, all bindings with descriptor type `VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER` **must** not use

VK_DESCRIPTOR_BINDING_UPDATE_AFTER_BIND_BIT

- VUID-VkDescriptorSetLayoutBindingFlagsCreateInfo-descriptorBindingSampledImageUpdateAfterBind-03006
  If VkPhysicalDeviceDescriptorIndexingFeatures::descriptorBindingSampledImageUpdateAfterBind is not enabled, all bindings with descriptor type VK_DESCRIPTOR_TYPE_SAMPLER, VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER, or VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE **must** not use VK_DESCRIPTOR_BINDING_UPDATE_AFTER_BIND_BIT
- VUID-VkDescriptorSetLayoutBindingFlagsCreateInfo-descriptorBindingStorageImageUpdateAfterBind-03007
  If VkPhysicalDeviceDescriptorIndexingFeatures::descriptorBindingStorageImageUpdateAfterBind is not enabled, all bindings with descriptor type VK_DESCRIPTOR_TYPE_STORAGE_IMAGE **must** not use VK_DESCRIPTOR_BINDING_UPDATE_AFTER_BIND_BIT
- VUID-VkDescriptorSetLayoutBindingFlagsCreateInfo-descriptorBindingStorageBufferUpdateAfterBind-03008
  If VkPhysicalDeviceDescriptorIndexingFeatures::descriptorBindingStorageBufferUpdateAfterBind is not enabled, all bindings with descriptor type VK_DESCRIPTOR_TYPE_STORAGE_BUFFER **must** not use VK_DESCRIPTOR_BINDING_UPDATE_AFTER_BIND_BIT
- VUID-VkDescriptorSetLayoutBindingFlagsCreateInfo-descriptorBindingUniformTexelBufferUpdateAfterBind-03009
  If VkPhysicalDeviceDescriptorIndexingFeatures::descriptorBindingUniformTexelBufferUpdateAfterBind is not enabled, all bindings with descriptor type VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER **must** not use VK_DESCRIPTOR_BINDING_UPDATE_AFTER_BIND_BIT
- VUID-VkDescriptorSetLayoutBindingFlagsCreateInfo-descriptorBindingStorageTexelBufferUpdateAfterBind-03010
  If VkPhysicalDeviceDescriptorIndexingFeatures::descriptorBindingStorageTexelBufferUpdateAfterBind is not enabled, all bindings with descriptor type VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER **must** not use VK_DESCRIPTOR_BINDING_UPDATE_AFTER_BIND_BIT
- VUID-VkDescriptorSetLayoutBindingFlagsCreateInfo-descriptorBindingInlineUniformBlockUpdateAfterBind-02211
  If VkPhysicalDeviceInlineUniformBlockFeatures::descriptorBindingInlineUniformBlockUpdateAfterBind is not enabled, all bindings with descriptor type VK_DESCRIPTOR_TYPE_INLINE_UNIFORM_BLOCK **must** not use VK_DESCRIPTOR_BINDING_UPDATE_AFTER_BIND_BIT
- VUID-VkDescriptorSetLayoutBindingFlagsCreateInfo-descriptorBindingAccelerationStructureUpdateAfterBind-03570
  If VkPhysicalDeviceAccelerationStructureFeaturesKHR::descriptorBindingAccelerationStructureUpdateAfterBind is not enabled, all bindings with descriptor type VK_DESCRIPTOR_TYPE_ACCELERATION_STRUCTURE_KHR or VK_DESCRIPTOR_TYPE_ACCELERATION_STRUCTURE_NV **must** not use VK_DESCRIPTOR_BINDING_UPDATE_AFTER_BIND_BIT

- VUID-VkDescriptorSetLayoutBindingFlagsCreateInfo-None-03011
  All bindings with descriptor type `VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT`, `VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC`, or `VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC` **must** not use `VK_DESCRIPTOR_BINDING_UPDATE_AFTER_BIND_BIT`
- VUID-VkDescriptorSetLayoutBindingFlagsCreateInfo-descriptorBindingUpdateUnusedWhilePending-03012
  If VkPhysicalDeviceDescriptorIndexingFeatures::`descriptorBindingUpdateUnusedWhilePending` is not enabled, all elements of `pBindingFlags` **must** not include `VK_DESCRIPTOR_BINDING_UPDATE_UNUSED_WHILE_PENDING_BIT`
- VUID-VkDescriptorSetLayoutBindingFlagsCreateInfo-descriptorBindingPartiallyBound-03013
  If VkPhysicalDeviceDescriptorIndexingFeatures::`descriptorBindingPartiallyBound` is not enabled, all elements of `pBindingFlags` **must** not include `VK_DESCRIPTOR_BINDING_PARTIALLY_BOUND_BIT`
- VUID-VkDescriptorSetLayoutBindingFlagsCreateInfo-descriptorBindingVariableDescriptorCount-03014
  If VkPhysicalDeviceDescriptorIndexingFeatures::`descriptorBindingVariableDescriptorCount` is not enabled, all elements of `pBindingFlags` **must** not include `VK_DESCRIPTOR_BINDING_VARIABLE_DESCRIPTOR_COUNT_BIT`
- VUID-VkDescriptorSetLayoutBindingFlagsCreateInfo-pBindingFlags-03015
  If an element of `pBindingFlags` includes `VK_DESCRIPTOR_BINDING_VARIABLE_DESCRIPTOR_COUNT_BIT`, that element's `descriptorType` **must** not be `VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC` or `VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC`

## Valid Usage (Implicit)

- VUID-VkDescriptorSetLayoutBindingFlagsCreateInfo-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_DESCRIPTOR_SET_LAYOUT_BINDING_FLAGS_CREATE_INFO`
- VUID-VkDescriptorSetLayoutBindingFlagsCreateInfo-pBindingFlags-parameter
  If `bindingCount` is not `0`, `pBindingFlags` **must** be a valid pointer to an array of `bindingCount` valid combinations of VkDescriptorBindingFlagBits values

Bits which **can** be set in each element of VkDescriptorSetLayoutBindingFlagsCreateInfo::`pBindingFlags`, specifying options for the corresponding descriptor set layout binding, are:

```
typedef enum VkDescriptorBindingFlagBits {
    VK_DESCRIPTOR_BINDING_UPDATE_AFTER_BIND_BIT = 0x00000001,
    VK_DESCRIPTOR_BINDING_UPDATE_UNUSED_WHILE_PENDING_BIT = 0x00000002,
    VK_DESCRIPTOR_BINDING_PARTIALLY_BOUND_BIT = 0x00000004,
    VK_DESCRIPTOR_BINDING_VARIABLE_DESCRIPTOR_COUNT_BIT = 0x00000008,
  // Provided by VK_EXT_descriptor_indexing
    VK_DESCRIPTOR_BINDING_UPDATE_AFTER_BIND_BIT_EXT =
VK_DESCRIPTOR_BINDING_UPDATE_AFTER_BIND_BIT,
  // Provided by VK_EXT_descriptor_indexing
    VK_DESCRIPTOR_BINDING_UPDATE_UNUSED_WHILE_PENDING_BIT_EXT =
VK_DESCRIPTOR_BINDING_UPDATE_UNUSED_WHILE_PENDING_BIT,
  // Provided by VK_EXT_descriptor_indexing
    VK_DESCRIPTOR_BINDING_PARTIALLY_BOUND_BIT_EXT =
VK_DESCRIPTOR_BINDING_PARTIALLY_BOUND_BIT,
  // Provided by VK_EXT_descriptor_indexing
    VK_DESCRIPTOR_BINDING_VARIABLE_DESCRIPTOR_COUNT_BIT_EXT =
VK_DESCRIPTOR_BINDING_VARIABLE_DESCRIPTOR_COUNT_BIT,
} VkDescriptorBindingFlagBits;
```

or the equivalent

```
// Provided by VK_EXT_descriptor_indexing
typedef VkDescriptorBindingFlagBits VkDescriptorBindingFlagBitsEXT;
```

- `VK_DESCRIPTOR_BINDING_UPDATE_AFTER_BIND_BIT` indicates that if descriptors in this binding are updated between when the descriptor set is bound in a command buffer and when that command buffer is submitted to a queue, then the submission will use the most recently set descriptors for this binding and the updates do not invalidate the command buffer. Descriptor bindings created with this flag are also partially exempt from the external synchronization requirement in vkUpdateDescriptorSetWithTemplateKHR and vkUpdateDescriptorSets. Multiple descriptors with this flag set **can** be updated concurrently in different threads, though the same descriptor **must** not be updated concurrently by two threads. Descriptors with this flag set **can** be updated concurrently with the set being bound to a command buffer in another thread, but not concurrently with the set being reset or freed.
- `VK_DESCRIPTOR_BINDING_PARTIALLY_BOUND_BIT` indicates that descriptors in this binding that are not *dynamically used* need not contain valid descriptors at the time the descriptors are consumed. A descriptor is dynamically used if any shader invocation executes an instruction that performs any memory access using the descriptor. If a descriptor is not dynamically used, any resource referenced by the descriptor is not considered to be referenced during command execution.
- `VK_DESCRIPTOR_BINDING_UPDATE_UNUSED_WHILE_PENDING_BIT` indicates that descriptors in this binding **can** be updated after a command buffer has bound this descriptor set, or while a command buffer that uses this descriptor set is pending execution, as long as the descriptors that are updated are not used by those command buffers. Descriptor bindings created with this flag are also partially exempt from the external synchronization requirement in vkUpdateDescriptorSetWithTemplateKHR and vkUpdateDescriptorSets in the same way as for

VK_DESCRIPTOR_BINDING_UPDATE_AFTER_BIND_BIT. If VK_DESCRIPTOR_BINDING_PARTIALLY_BOUND_BIT is also set, then descriptors **can** be updated as long as they are not dynamically used by any shader invocations. If VK_DESCRIPTOR_BINDING_PARTIALLY_BOUND_BIT is not set, then descriptors **can** be updated as long as they are not statically used by any shader invocations.

- VK_DESCRIPTOR_BINDING_VARIABLE_DESCRIPTOR_COUNT_BIT indicates that this is a *variable-sized descriptor binding* whose size will be specified when a descriptor set is allocated using this layout. The value of descriptorCount is treated as an upper bound on the size of the binding. This **must** only be used for the last binding in the descriptor set layout (i.e. the binding with the largest value of binding). For the purposes of counting against limits such as maxDescriptorSet* and maxPerStageDescriptor*, the full value of descriptorCount is counted, except for descriptor bindings with a descriptor type of VK_DESCRIPTOR_TYPE_INLINE_UNIFORM_BLOCK, when VkDescriptorSetLayoutCreateInfo::flags does not contain VK_DESCRIPTOR_SET_LAYOUT_CREATE_DESCRIPTOR_BUFFER_BIT_EXT. In this case, descriptorCount specifies the upper bound on the byte size of the binding; thus it counts against the maxInlineUniformBlockSize limit instead.

**NOTE** Note that while VK_DESCRIPTOR_BINDING_UPDATE_AFTER_BIND_BIT and VK_DESCRIPTOR_BINDING_UPDATE_UNUSED_WHILE_PENDING_BIT both involve updates to descriptor sets after they are bound, VK_DESCRIPTOR_BINDING_UPDATE_UNUSED_WHILE_PENDING_BIT is a weaker requirement since it is only about descriptors that are not used, whereas VK_DESCRIPTOR_BINDING_UPDATE_AFTER_BIND_BIT requires the implementation to observe updates to descriptors that are used.

```
typedef VkFlags VkDescriptorBindingFlags;
```

or the equivalent

```
// Provided by VK_EXT_descriptor_indexing
typedef VkDescriptorBindingFlags VkDescriptorBindingFlagsEXT;
```

VkDescriptorBindingFlags is a bitmask type for setting a mask of zero or more VkDescriptorBindingFlagBits.

To query information about whether a descriptor set layout **can** be created, call:

```
// Provided by VK_VERSION_1_1
void vkGetDescriptorSetLayoutSupport(
    VkDevice                                    device,
    const VkDescriptorSetLayoutCreateInfo*      pCreateInfo,
    VkDescriptorSetLayoutSupport*               pSupport);
```

or the equivalent command

```
// Provided by VK_KHR_maintenance3
void vkGetDescriptorSetLayoutSupportKHR(
    VkDevice                                    device,
    const VkDescriptorSetLayoutCreateInfo*      pCreateInfo,
    VkDescriptorSetLayoutSupport*               pSupport);
```

- `device` is the logical device that would create the descriptor set layout.
- `pCreateInfo` is a pointer to a VkDescriptorSetLayoutCreateInfo structure specifying the state of the descriptor set layout object.
- `pSupport` is a pointer to a VkDescriptorSetLayoutSupport structure, in which information about support for the descriptor set layout object is returned.

Some implementations have limitations on what fits in a descriptor set which are not easily expressible in terms of existing limits like `maxDescriptorSet`*, for example if all descriptor types share a limited space in memory but each descriptor is a different size or alignment. This command returns information about whether a descriptor set satisfies this limit. If the descriptor set layout satisfies the VkPhysicalDeviceMaintenance3Properties::`maxPerSetDescriptors` limit, this command is guaranteed to return `VK_TRUE` in VkDescriptorSetLayoutSupport::`supported`. If the descriptor set layout exceeds the VkPhysicalDeviceMaintenance3Properties::`maxPerSetDescriptors` limit, whether the descriptor set layout is supported is implementation-dependent and **may** depend on whether the descriptor sizes and alignments cause the layout to exceed an internal limit.

This command does not consider other limits such as `maxPerStageDescriptor`*, and so a descriptor set layout that is supported according to this command **must** still satisfy the pipeline layout limits such as `maxPerStageDescriptor`* in order to be used in a pipeline layout.

**NOTE** This is a `VkDevice` query rather than `VkPhysicalDevice` because the answer **may** depend on enabled features.

### Valid Usage (Implicit)

- VUID-vkGetDescriptorSetLayoutSupport-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkGetDescriptorSetLayoutSupport-pCreateInfo-parameter
  `pCreateInfo` **must** be a valid pointer to a valid VkDescriptorSetLayoutCreateInfo structure
- VUID-vkGetDescriptorSetLayoutSupport-pSupport-parameter
  `pSupport` **must** be a valid pointer to a VkDescriptorSetLayoutSupport structure

Information about support for the descriptor set layout is returned in a `VkDescriptorSetLayoutSupport` structure:

```
// Provided by VK_VERSION_1_1
typedef struct VkDescriptorSetLayoutSupport {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           supported;
} VkDescriptorSetLayoutSupport;
```

or the equivalent

```
// Provided by VK_KHR_maintenance3
typedef VkDescriptorSetLayoutSupport VkDescriptorSetLayoutSupportKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `supported` specifies whether the descriptor set layout **can** be created.

`supported` is set to `VK_TRUE` if the descriptor set **can** be created, or else is set to `VK_FALSE`.

### Valid Usage (Implicit)

- VUID-VkDescriptorSetLayoutSupport-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_DESCRIPTOR_SET_LAYOUT_SUPPORT`
- VUID-VkDescriptorSetLayoutSupport-pNext-pNext
  `pNext` **must** be `NULL` or a pointer to a valid instance of VkDescriptorSetVariableDescriptorCountLayoutSupport
- VUID-VkDescriptorSetLayoutSupport-sType-unique
  The `sType` value of each struct in the `pNext` chain **must** be unique

If the `pNext` chain of a VkDescriptorSetLayoutSupport structure includes a `VkDescriptorSetVariableDescriptorCountLayoutSupport` structure, then that structure returns additional information about whether the descriptor set layout is supported.

```
typedef struct VkDescriptorSetVariableDescriptorCountLayoutSupport {
    VkStructureType    sType;
    void*              pNext;
    uint32_t           maxVariableDescriptorCount;
} VkDescriptorSetVariableDescriptorCountLayoutSupport;
```

or the equivalent

```
// Provided by VK_EXT_descriptor_indexing
typedef VkDescriptorSetVariableDescriptorCountLayoutSupport
VkDescriptorSetVariableDescriptorCountLayoutSupportEXT;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- maxVariableDescriptorCount indicates the maximum number of descriptors supported in the highest numbered binding of the layout, if that binding is variable-sized. If the highest numbered binding of the layout has a descriptor type of VK_DESCRIPTOR_TYPE_INLINE_UNIFORM_BLOCK then maxVariableDescriptorCount indicates the maximum byte size supported for the binding, if that binding is variable-sized.

If the VkDescriptorSetLayoutCreateInfo structure specified in vkGetDescriptorSetLayoutSupport::pCreateInfo includes a variable-sized descriptor, then supported is determined assuming the requested size of the variable-sized descriptor, and maxVariableDescriptorCount is set to the maximum size of that descriptor that **can** be successfully created (which is greater than or equal to the requested size passed in). If the VkDescriptorSetLayoutCreateInfo structure does not include a variable-sized descriptor, or if the VkPhysicalDeviceDescriptorIndexingFeatures::descriptorBindingVariableDescriptorCount feature is not enabled, then maxVariableDescriptorCount is set to zero. For the purposes of this command, a variable-sized descriptor binding with a descriptorCount of zero is treated as having a descriptorCount of four if descriptorType is VK_DESCRIPTOR_TYPE_INLINE_UNIFORM_BLOCK, or one otherwise, and thus the binding is not ignored and the maximum descriptor count will be returned. If the layout is not supported, then the value written to maxVariableDescriptorCount is undefined.

### Valid Usage (Implicit)

- VUID-VkDescriptorSetVariableDescriptorCountLayoutSupport-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_DESCRIPTOR_SET_VARIABLE_DESCRIPTOR_COUNT_LAYOUT_SUPPORT

The following examples show a shader snippet using two descriptor sets, and application code that creates corresponding descriptor set layouts.

*GLSL example*

```
//
// binding to a single sampled image descriptor in set 0
//
layout (set=0, binding=0) uniform texture2D mySampledImage;

//
// binding to an array of sampled image descriptors in set 0
//
layout (set=0, binding=1) uniform texture2D myArrayOfSampledImages[12];
```

```
//
// binding to a single uniform buffer descriptor in set 1
//
layout (set=1, binding=0) uniform myUniformBuffer
{
    vec4 myElement[32];
};
```

*SPIR-V example*

```
            ...
      %1 = OpExtInstImport "GLSL.std.450"
            ...
            OpName %9 "mySampledImage"
            OpName %14 "myArrayOfSampledImages"
            OpName %18 "myUniformBuffer"
            OpMemberName %18 0 "myElement"
            OpName %20 ""
            OpDecorate %9 DescriptorSet 0
            OpDecorate %9 Binding 0
            OpDecorate %14 DescriptorSet 0
            OpDecorate %14 Binding 1
            OpDecorate %17 ArrayStride 16
            OpMemberDecorate %18 0 Offset 0
            OpDecorate %18 Block
            OpDecorate %20 DescriptorSet 1
            OpDecorate %20 Binding 0
      %2 = OpTypeVoid
      %3 = OpTypeFunction %2
      %6 = OpTypeFloat 32
      %7 = OpTypeImage %6 2D 0 0 0 1 Unknown
      %8 = OpTypePointer UniformConstant %7
      %9 = OpVariable %8 UniformConstant
     %10 = OpTypeInt 32 0
     %11 = OpConstant %10 12
     %12 = OpTypeArray %7 %11
     %13 = OpTypePointer UniformConstant %12
     %14 = OpVariable %13 UniformConstant
     %15 = OpTypeVector %6 4
     %16 = OpConstant %10 32
     %17 = OpTypeArray %15 %16
     %18 = OpTypeStruct %17
     %19 = OpTypePointer Uniform %18
     %20 = OpVariable %19 Uniform
            ...
```

*API example*

```
VkResult myResult;
```

```
const VkDescriptorSetLayoutBinding myDescriptorSetLayoutBinding[] =
{
    // binding to a single image descriptor
    {
        .binding = 0,
        .descriptorType = VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE,
        .descriptorCount = 1,
        .stageFlags = VK_SHADER_STAGE_FRAGMENT_BIT,
        .pImmutableSamplers = NULL
    },

    // binding to an array of image descriptors
    {
        .binding = 1,
        .descriptorType = VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE,
        .descriptorCount = 12,
        .stageFlags = VK_SHADER_STAGE_FRAGMENT_BIT,
        .pImmutableSamplers = NULL
    },

    // binding to a single uniform buffer descriptor
    {
        .binding = 0,
        .descriptorType = VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER,
        .descriptorCount = 1,
        .stageFlags = VK_SHADER_STAGE_FRAGMENT_BIT,
        .pImmutableSamplers = NULL
    }
};

const VkDescriptorSetLayoutCreateInfo myDescriptorSetLayoutCreateInfo[] =
{
    // Information for first descriptor set with two descriptor bindings
    {
        .sType = VK_STRUCTURE_TYPE_DESCRIPTOR_SET_LAYOUT_CREATE_INFO,
        .pNext = NULL,
        .flags = 0,
        .bindingCount = 2,
        .pBindings = &myDescriptorSetLayoutBinding[0]
    },

    // Information for second descriptor set with one descriptor binding
    {
        .sType = VK_STRUCTURE_TYPE_DESCRIPTOR_SET_LAYOUT_CREATE_INFO,
        .pNext = NULL,
        .flags = 0,
        .bindingCount = 1,
        .pBindings = &myDescriptorSetLayoutBinding[2]
    }
};
```

```
VkDescriptorSetLayout myDescriptorSetLayout[2];

//
// Create first descriptor set layout
//
myResult = vkCreateDescriptorSetLayout(
    myDevice,
    &myDescriptorSetLayoutCreateInfo[0],
    NULL,
    &myDescriptorSetLayout[0]);

//
// Create second descriptor set layout
//
myResult = vkCreateDescriptorSetLayout(
    myDevice,
    &myDescriptorSetLayoutCreateInfo[1],
    NULL,
    &myDescriptorSetLayout[1]);
```

To destroy a descriptor set layout, call:

```
// Provided by VK_VERSION_1_0
void vkDestroyDescriptorSetLayout(
    VkDevice                                    device,
    VkDescriptorSetLayout                       descriptorSetLayout,
    const VkAllocationCallbacks*                pAllocator);
```

- `device` is the logical device that destroys the descriptor set layout.
- `descriptorSetLayout` is the descriptor set layout to destroy.
- `pAllocator` controls host memory allocation as described in the Memory Allocation chapter.

### Valid Usage

- VUID-vkDestroyDescriptorSetLayout-descriptorSetLayout-00284
  If `VkAllocationCallbacks` were provided when `descriptorSetLayout` was created, a compatible set of callbacks **must** be provided here
- VUID-vkDestroyDescriptorSetLayout-descriptorSetLayout-00285
  If no `VkAllocationCallbacks` were provided when `descriptorSetLayout` was created, `pAllocator` **must** be `NULL`

### Valid Usage (Implicit)

- VUID-vkDestroyDescriptorSetLayout-device-parameter
  `device` **must** be a valid VkDevice handle

- VUID-vkDestroyDescriptorSetLayout-descriptorSetLayout-parameter
  If `descriptorSetLayout` is not VK_NULL_HANDLE, `descriptorSetLayout` **must** be a valid VkDescriptorSetLayout handle
- VUID-vkDestroyDescriptorSetLayout-pAllocator-parameter
  If `pAllocator` is not `NULL`, `pAllocator` **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkDestroyDescriptorSetLayout-descriptorSetLayout-parent
  If `descriptorSetLayout` is a valid handle, it **must** have been created, allocated, or retrieved from `device`

**Host Synchronization**

- Host access to `descriptorSetLayout` **must** be externally synchronized

### 14.2.2. Pipeline Layouts

Access to descriptor sets from a pipeline is accomplished through a *pipeline layout*. Zero or more descriptor set layouts and zero or more push constant ranges are combined to form a pipeline layout object describing the complete set of resources that **can** be accessed by a pipeline. The pipeline layout represents a sequence of descriptor sets with each having a specific layout. This sequence of layouts is used to determine the interface between shader stages and shader resources. Each pipeline is created using a pipeline layout.

Pipeline layout objects are represented by `VkPipelineLayout` handles:

```
// Provided by VK_VERSION_1_0
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkPipelineLayout)
```

To create a pipeline layout, call:

```
// Provided by VK_VERSION_1_0
VkResult vkCreatePipelineLayout(
    VkDevice                                    device,
    const VkPipelineLayoutCreateInfo*           pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkPipelineLayout*                           pPipelineLayout);
```

- `device` is the logical device that creates the pipeline layout.
- `pCreateInfo` is a pointer to a VkPipelineLayoutCreateInfo structure specifying the state of the pipeline layout object.
- `pAllocator` controls host memory allocation as described in the Memory Allocation chapter.
- `pPipelineLayout` is a pointer to a VkPipelineLayout handle in which the resulting pipeline layout object is returned.

### Valid Usage (Implicit)

- VUID-vkCreatePipelineLayout-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkCreatePipelineLayout-pCreateInfo-parameter
  `pCreateInfo` **must** be a valid pointer to a valid VkPipelineLayoutCreateInfo structure
- VUID-vkCreatePipelineLayout-pAllocator-parameter
  If `pAllocator` is not `NULL`, `pAllocator` **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkCreatePipelineLayout-pPipelineLayout-parameter
  `pPipelineLayout` **must** be a valid pointer to a VkPipelineLayout handle

### Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`

The VkPipelineLayoutCreateInfo structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkPipelineLayoutCreateInfo {
    VkStructureType                 sType;
    const void*                     pNext;
    VkPipelineLayoutCreateFlags     flags;
    uint32_t                        setLayoutCount;
    const VkDescriptorSetLayout*    pSetLayouts;
    uint32_t                        pushConstantRangeCount;
    const VkPushConstantRange*      pPushConstantRanges;
} VkPipelineLayoutCreateInfo;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is a bitmask of VkPipelineLayoutCreateFlagBits specifying options for pipeline layout creation.
- `setLayoutCount` is the number of descriptor sets included in the pipeline layout.
- `pSetLayouts` is a pointer to an array of `VkDescriptorSetLayout` objects.
- `pushConstantRangeCount` is the number of push constant ranges included in the pipeline layout.
- `pPushConstantRanges` is a pointer to an array of `VkPushConstantRange` structures defining a set of

push constant ranges for use in a single pipeline layout. In addition to descriptor set layouts, a pipeline layout also describes how many push constants **can** be accessed by each stage of the pipeline.

| | |
|---|---|
| **NOTE** | Push constants represent a high speed path to modify constant data in pipelines that is expected to outperform memory-backed resource updates. |

## Valid Usage

- VUID-VkPipelineLayoutCreateInfo-setLayoutCount-00286
  `setLayoutCount` **must** be less than or equal to `VkPhysicalDeviceLimits::maxBoundDescriptorSets`
- VUID-VkPipelineLayoutCreateInfo-descriptorType-03016
  The total number of descriptors in descriptor set layouts created without the `VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT` bit set with a `descriptorType` of `VK_DESCRIPTOR_TYPE_SAMPLER` and `VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER` accessible to any given shader stage across all elements of `pSetLayouts` **must** be less than or equal to `VkPhysicalDeviceLimits::maxPerStageDescriptorSamplers`
- VUID-VkPipelineLayoutCreateInfo-descriptorType-03017
  The total number of descriptors in descriptor set layouts created without the `VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT` bit set with a `descriptorType` of `VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER` and `VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC` accessible to any given shader stage across all elements of `pSetLayouts` **must** be less than or equal to `VkPhysicalDeviceLimits::maxPerStageDescriptorUniformBuffers`
- VUID-VkPipelineLayoutCreateInfo-descriptorType-03018
  The total number of descriptors in descriptor set layouts created without the `VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT` bit set with a `descriptorType` of `VK_DESCRIPTOR_TYPE_STORAGE_BUFFER` and `VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC` accessible to any given shader stage across all elements of `pSetLayouts` **must** be less than or equal to `VkPhysicalDeviceLimits::maxPerStageDescriptorStorageBuffers`
- VUID-VkPipelineLayoutCreateInfo-descriptorType-06939
  The total number of descriptors in descriptor set layouts created without the `VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT` bit set with a `descriptorType` of `VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER`, `VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE`, `VK_DESCRIPTOR_TYPE_SAMPLE_WEIGHT_IMAGE_QCOM`, `VK_DESCRIPTOR_TYPE_BLOCK_MATCH_IMAGE_QCOM`, and `VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER`, accessible to any given shader stage across all elements of `pSetLayouts` **must** be less than or equal to `VkPhysicalDeviceLimits::maxPerStageDescriptorSampledImages`
- VUID-VkPipelineLayoutCreateInfo-descriptorType-03020
  The total number of descriptors in descriptor set layouts created without the `VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT` bit set with a `descriptorType` of `VK_DESCRIPTOR_TYPE_STORAGE_IMAGE`, and `VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER` accessible to any given shader stage across all elements of `pSetLayouts` **must** be less than or equal to `VkPhysicalDeviceLimits::maxPerStageDescriptorStorageImages`
- VUID-VkPipelineLayoutCreateInfo-descriptorType-03021

The total number of descriptors in descriptor set layouts created without the VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT bit set with a descriptorType of VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT accessible to any given shader stage across all elements of pSetLayouts **must** be less than or equal to VkPhysicalDeviceLimits::maxPerStageDescriptorInputAttachments

- VUID-VkPipelineLayoutCreateInfo-descriptorType-02214
  The total number of bindings in descriptor set layouts created without the VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT bit set and with a descriptorType of VK_DESCRIPTOR_TYPE_INLINE_UNIFORM_BLOCK accessible to any given shader stage across all elements of pSetLayouts, **must** be less than or equal to VkPhysicalDeviceInlineUniformBlockProperties::maxPerStageDescriptorInlineUniformBlocks
- VUID-VkPipelineLayoutCreateInfo-descriptorType-03022
  The total number of descriptors with a descriptorType of VK_DESCRIPTOR_TYPE_SAMPLER and VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER accessible to any given shader stage across all elements of pSetLayouts **must** be less than or equal to VkPhysicalDeviceDescriptorIndexingProperties::maxPerStageDescriptorUpdateAfterBindSamplers
- VUID-VkPipelineLayoutCreateInfo-descriptorType-03023
  The total number of descriptors with a descriptorType of VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER and VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC accessible to any given shader stage across all elements of pSetLayouts **must** be less than or equal to VkPhysicalDeviceDescriptorIndexingProperties::maxPerStageDescriptorUpdateAfterBindUniformBuffers
- VUID-VkPipelineLayoutCreateInfo-descriptorType-03024
  The total number of descriptors with a descriptorType of VK_DESCRIPTOR_TYPE_STORAGE_BUFFER and VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC accessible to any given shader stage across all elements of pSetLayouts **must** be less than or equal to VkPhysicalDeviceDescriptorIndexingProperties::maxPerStageDescriptorUpdateAfterBindStorageBuffers
- VUID-VkPipelineLayoutCreateInfo-descriptorType-03025
  The total number of descriptors with a descriptorType of VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER, VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE, and VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER accessible to any given shader stage across all elements of pSetLayouts **must** be less than or equal to VkPhysicalDeviceDescriptorIndexingProperties::maxPerStageDescriptorUpdateAfterBindSampledImages
- VUID-VkPipelineLayoutCreateInfo-descriptorType-03026
  The total number of descriptors with a descriptorType of VK_DESCRIPTOR_TYPE_STORAGE_IMAGE, and VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER accessible to any given shader stage across all elements of pSetLayouts **must** be less than or equal to VkPhysicalDeviceDescriptorIndexingProperties::maxPerStageDescriptorUpdateAfterBindStorageImages
- VUID-VkPipelineLayoutCreateInfo-descriptorType-03027
  The total number of descriptors with a descriptorType of VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT accessible to any given shader stage across all

elements of pSetLayouts **must** be less than or equal to VkPhysicalDeviceDescriptorIndexingProperties::maxPerStageDescriptorUpdateAfterBindInputAttachments

- VUID-VkPipelineLayoutCreateInfo-descriptorType-02215
  The total number of bindings with a descriptorType of VK_DESCRIPTOR_TYPE_INLINE_UNIFORM_BLOCK accessible to any given shader stage across all elements of pSetLayouts **must** be less than or equal to VkPhysicalDeviceInlineUniformBlockProperties::maxPerStageDescriptorUpdateAfterBindInlineUniformBlocks

- VUID-VkPipelineLayoutCreateInfo-descriptorType-03028
  The total number of descriptors in descriptor set layouts created without the VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT bit set with a descriptorType of VK_DESCRIPTOR_TYPE_SAMPLER and VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER accessible across all shader stages and across all elements of pSetLayouts **must** be less than or equal to VkPhysicalDeviceLimits::maxDescriptorSetSamplers

- VUID-VkPipelineLayoutCreateInfo-descriptorType-03029
  The total number of descriptors in descriptor set layouts created without the VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT bit set with a descriptorType of VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER accessible across all shader stages and across all elements of pSetLayouts **must** be less than or equal to VkPhysicalDeviceLimits::maxDescriptorSetUniformBuffers

- VUID-VkPipelineLayoutCreateInfo-descriptorType-03030
  If the maintenance7 feature is not enabled, the total number of descriptors in descriptor set layouts created without the VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT bit set with a descriptorType of VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC accessible across all shader stages and across all elements of pSetLayouts **must** be less than or equal to VkPhysicalDeviceLimits::maxDescriptorSetUniformBuffersDynamic

- VUID-VkPipelineLayoutCreateInfo-maintenance7-10003
  If the maintenance7 feature is enabled, the total number of descriptors in descriptor set layouts created without the VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT bit set with a descriptorType of VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC accessible across all shader stages and across all elements of pSetLayouts **must** be less than or equal to VkPhysicalDeviceMaintenance7PropertiesKHR::maxDescriptorSetTotalUniformBuffersDynamic

- VUID-VkPipelineLayoutCreateInfo-descriptorType-03031
  The total number of descriptors in descriptor set layouts created without the VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT bit set with a descriptorType of VK_DESCRIPTOR_TYPE_STORAGE_BUFFER accessible across all shader stages and across all elements of pSetLayouts **must** be less than or equal to VkPhysicalDeviceLimits::maxDescriptorSetStorageBuffers

- VUID-VkPipelineLayoutCreateInfo-descriptorType-03032
  If the maintenance7 feature is not enabled, the total number of descriptors in descriptor set layouts created without the VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT bit set with a descriptorType of VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC accessible across all shader stages and across all elements of pSetLayouts **must** be less than or equal

to VkPhysicalDeviceLimits::maxDescriptorSetStorageBuffersDynamic

- VUID-VkPipelineLayoutCreateInfo-maintenance7-10004
  If the maintenance7 feature is enabled, the total number of descriptors in descriptor set layouts created without the VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT bit set with a descriptorType of VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC accessible across all shader stages and across all elements of pSetLayouts **must** be less than or equal to VkPhysicalDeviceMaintenance7PropertiesKHR::maxDescriptorSetTotalStorageBuffersDynamic
- VUID-VkPipelineLayoutCreateInfo-None-10005
  The total number of descriptors in descriptor set layouts created without the VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT bit set with a descriptorType of VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC or VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC accessible across all shader stages and across all elements of pSetLayouts **must** be less than or equal to VkPhysicalDeviceMaintenance7PropertiesKHR::maxDescriptorSetTotalBuffersDynamic
- VUID-VkPipelineLayoutCreateInfo-pSetLayouts-10006
  The total number of descriptors of the type VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC or VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC accessible across all shader stages and across all elements of pSetLayouts **must** be less than or equal to VkPhysicalDeviceMaintenance7PropertiesKHR::maxDescriptorSetUpdateAfterBindTotalBuffersDynamic
- VUID-VkPipelineLayoutCreateInfo-descriptorType-03033
  The total number of descriptors in descriptor set layouts created without the VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT bit set with a descriptorType of VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER, VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE, and VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER accessible across all shader stages and across all elements of pSetLayouts **must** be less than or equal to VkPhysicalDeviceLimits::maxDescriptorSetSampledImages
- VUID-VkPipelineLayoutCreateInfo-descriptorType-03034
  The total number of descriptors in descriptor set layouts created without the VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT bit set with a descriptorType of VK_DESCRIPTOR_TYPE_STORAGE_IMAGE, and VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER accessible across all shader stages and across all elements of pSetLayouts **must** be less than or equal to VkPhysicalDeviceLimits::maxDescriptorSetStorageImages
- VUID-VkPipelineLayoutCreateInfo-descriptorType-03035
  The total number of descriptors in descriptor set layouts created without the VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT bit set with a descriptorType of VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT accessible across all shader stages and across all elements of pSetLayouts **must** be less than or equal to VkPhysicalDeviceLimits::maxDescriptorSetInputAttachments
- VUID-VkPipelineLayoutCreateInfo-descriptorType-02216
  The total number of bindings in descriptor set layouts created without the VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT bit set with a descriptorType of VK_DESCRIPTOR_TYPE_INLINE_UNIFORM_BLOCK accessible across all shader stages and across all elements of pSetLayouts **must** be less than or equal to

VkPhysicalDeviceInlineUniformBlockProperties::maxDescriptorSetInlineUniformBlocks

- VUID-VkPipelineLayoutCreateInfo-pSetLayouts-03036
  The total number of descriptors of the type VK_DESCRIPTOR_TYPE_SAMPLER and VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER accessible across all shader stages and across all elements of pSetLayouts **must** be less than or equal to VkPhysicalDeviceDescriptorIndexingProperties::maxDescriptorSetUpdateAfterBindSamplers
- VUID-VkPipelineLayoutCreateInfo-pSetLayouts-03037
  The total number of descriptors of the type VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER accessible across all shader stages and across all elements of pSetLayouts **must** be less than or equal to VkPhysicalDeviceDescriptorIndexingProperties::maxDescriptorSetUpdateAfterBindUniformBuffers
- VUID-VkPipelineLayoutCreateInfo-pSetLayouts-03038
  If the maintenance7 feature is not enabled, the total number of descriptors of the type VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC accessible across all shader stages and across all elements of pSetLayouts **must** be less than or equal to VkPhysicalDeviceLimits::maxDescriptorSetUpdateAfterBindUniformBuffersDynamic
- VUID-VkPipelineLayoutCreateInfo-maintenance7-10007
  If the maintenance7 feature is enabled, the total number of descriptors of the type VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC accessible across all shader stages and across all elements of pSetLayouts **must** be less than or equal to VkPhysicalDeviceMaintenance7PropertiesKHR::maxDescriptorSetUpdateAfterBindTotalUniformBuffersDynamic
- VUID-VkPipelineLayoutCreateInfo-pSetLayouts-03039
  The total number of descriptors of the type VK_DESCRIPTOR_TYPE_STORAGE_BUFFER accessible across all shader stages and across all elements of pSetLayouts **must** be less than or equal to VkPhysicalDeviceDescriptorIndexingProperties::maxDescriptorSetUpdateAfterBindStorageBuffers
- VUID-VkPipelineLayoutCreateInfo-pSetLayouts-03040
  If the maintenance7 feature is not enabled, the total number of descriptors of the type VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC accessible across all shader stages and across all elements of pSetLayouts **must** be less than or equal to VkPhysicalDeviceLimits::maxDescriptorSetUpdateAfterBindStorageBuffersDynamic
- VUID-VkPipelineLayoutCreateInfo-maintenance7-10008
  If the maintenance7 feature is enabled, the total number of descriptors of the type VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC accessible across all shader stages and across all elements of pSetLayouts **must** be less than or equal to VkPhysicalDeviceMaintenance7PropertiesKHR::maxDescriptorSetUpdateAfterBindTotalStorageBuffersDynamic
- VUID-VkPipelineLayoutCreateInfo-pSetLayouts-03041
  The total number of descriptors of the type VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER, VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE, and VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER accessible across all shader stages and across all elements of pSetLayouts **must** be less than or equal to VkPhysicalDeviceDescriptorIndexingProperties::maxDescriptorSetUpdateAfterBindSampledImages

- VUID-VkPipelineLayoutCreateInfo-pSetLayouts-03042
  The total number of descriptors of the type `VK_DESCRIPTOR_TYPE_STORAGE_IMAGE`, and `VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER` accessible across all shader stages and across all elements of `pSetLayouts` **must** be less than or equal to `VkPhysicalDeviceDescriptorIndexingProperties::maxDescriptorSetUpdateAfterBindStorageImages`
- VUID-VkPipelineLayoutCreateInfo-pSetLayouts-03043
  The total number of descriptors of the type `VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT` accessible across all shader stages and across all elements of `pSetLayouts` **must** be less than or equal to `VkPhysicalDeviceDescriptorIndexingProperties::maxDescriptorSetUpdateAfterBindInputAttachments`
- VUID-VkPipelineLayoutCreateInfo-descriptorType-02217
  The total number of bindings with a `descriptorType` of `VK_DESCRIPTOR_TYPE_INLINE_UNIFORM_BLOCK` accessible across all shader stages and across all elements of `pSetLayouts` **must** be less than or equal to `VkPhysicalDeviceInlineUniformBlockProperties::maxDescriptorSetUpdateAfterBindInlineUniformBlocks`
- VUID-VkPipelineLayoutCreateInfo-pPushConstantRanges-00292
  Any two elements of `pPushConstantRanges` **must** not include the same stage in `stageFlags`
- VUID-VkPipelineLayoutCreateInfo-pSetLayouts-00293
  `pSetLayouts` **must** not contain more than one descriptor set layout that was created with `VK_DESCRIPTOR_SET_LAYOUT_CREATE_PUSH_DESCRIPTOR_BIT_KHR` set
- VUID-VkPipelineLayoutCreateInfo-descriptorType-03571
  The total number of bindings in descriptor set layouts created without the `VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT` bit set with a `descriptorType` of `VK_DESCRIPTOR_TYPE_ACCELERATION_STRUCTURE_KHR` accessible to any given shader stage across all elements of `pSetLayouts` **must** be less than or equal to VkPhysicalDeviceAccelerationStructurePropertiesKHR::`maxPerStageDescriptorAccelerationStructures`
- VUID-VkPipelineLayoutCreateInfo-descriptorType-03572
  The total number of bindings with a `descriptorType` of `VK_DESCRIPTOR_TYPE_ACCELERATION_STRUCTURE_KHR` accessible to any given shader stage across all elements of `pSetLayouts` **must** be less than or equal to VkPhysicalDeviceAccelerationStructurePropertiesKHR::`maxPerStageDescriptorUpdateAfterBindAccelerationStructures`
- VUID-VkPipelineLayoutCreateInfo-descriptorType-03573
  The total number of bindings in descriptor set layouts created without the `VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT` bit set with a `descriptorType` of `VK_DESCRIPTOR_TYPE_ACCELERATION_STRUCTURE_KHR` accessible across all shader stages and across all elements of `pSetLayouts` **must** be less than or equal to VkPhysicalDeviceAccelerationStructurePropertiesKHR::`maxDescriptorSetAccelerationStructures`
- VUID-VkPipelineLayoutCreateInfo-descriptorType-03574
  The total number of bindings with a `descriptorType` of `VK_DESCRIPTOR_TYPE_ACCELERATION_STRUCTURE_KHR` accessible across all shader stages and

across all elements of pSetLayouts **must** be less than or equal to VkPhysicalDeviceAccelerationStructurePropertiesKHR::maxDescriptorSetUpdateAfterBindAccelerationStructures

- VUID-VkPipelineLayoutCreateInfo-descriptorType-02381
  The total number of bindings with a descriptorType of VK_DESCRIPTOR_TYPE_ACCELERATION_STRUCTURE_NV accessible across all shader stages and across all elements of pSetLayouts **must** be less than or equal to VkPhysicalDeviceRayTracingPropertiesNV::maxDescriptorSetAccelerationStructures
- VUID-VkPipelineLayoutCreateInfo-pImmutableSamplers-03566
  The total number of pImmutableSamplers created with flags containing VK_SAMPLER_CREATE_SUBSAMPLED_BIT_EXT or VK_SAMPLER_CREATE_SUBSAMPLED_COARSE_RECONSTRUCTION_BIT_EXT across all shader stages and across all elements of pSetLayouts **must** be less than or equal to VkPhysicalDeviceFragmentDensityMap2PropertiesEXT::maxDescriptorSetSubsampledSamplers
- VUID-VkPipelineLayoutCreateInfo-pSetLayouts-04606
  Any element of pSetLayouts **must** not have been created with the VK_DESCRIPTOR_SET_LAYOUT_CREATE_HOST_ONLY_POOL_BIT_EXT bit set
- VUID-VkPipelineLayoutCreateInfo-graphicsPipelineLibrary-06753
  If graphicsPipelineLibrary is not enabled, elements of pSetLayouts **must** be valid VkDescriptorSetLayout objects
- VUID-VkPipelineLayoutCreateInfo-pSetLayouts-08008
  If any element of pSetLayouts was created with the VK_DESCRIPTOR_SET_LAYOUT_CREATE_DESCRIPTOR_BUFFER_BIT_EXT bit set, all elements of pSetLayouts **must** have been created with the VK_DESCRIPTOR_SET_LAYOUT_CREATE_DESCRIPTOR_BUFFER_BIT_EXT bit set

## Valid Usage (Implicit)

- VUID-VkPipelineLayoutCreateInfo-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PIPELINE_LAYOUT_CREATE_INFO
- VUID-VkPipelineLayoutCreateInfo-flags-parameter
  flags **must** be a valid combination of VkPipelineLayoutCreateFlagBits values
- VUID-VkPipelineLayoutCreateInfo-pSetLayouts-parameter
  If setLayoutCount is not 0, pSetLayouts **must** be a valid pointer to an array of setLayoutCount valid or VK_NULL_HANDLE VkDescriptorSetLayout handles
- VUID-VkPipelineLayoutCreateInfo-pPushConstantRanges-parameter
  If pushConstantRangeCount is not 0, pPushConstantRanges **must** be a valid pointer to an array of pushConstantRangeCount valid VkPushConstantRange structures

```
// Provided by VK_EXT_graphics_pipeline_library
typedef enum VkPipelineLayoutCreateFlagBits {
  // Provided by VK_EXT_graphics_pipeline_library
    VK_PIPELINE_LAYOUT_CREATE_INDEPENDENT_SETS_BIT_EXT = 0x00000002,
} VkPipelineLayoutCreateFlagBits;
```

- VK_PIPELINE_LAYOUT_CREATE_INDEPENDENT_SETS_BIT_EXT specifies that implementations **must** ensure that the properties and/or absence of a particular descriptor set do not influence any other properties of the pipeline layout. This allows pipelines libraries linked without VK_PIPELINE_CREATE_LINK_TIME_OPTIMIZATION_BIT_EXT to be created with a subset of the total descriptor sets.

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkPipelineLayoutCreateFlags;
```

VkPipelineLayoutCreateFlags is a bitmask type for setting a mask of VkPipelineLayoutCreateFlagBits.

The VkPushConstantRange structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkPushConstantRange {
    VkShaderStageFlags    stageFlags;
    uint32_t              offset;
    uint32_t              size;
} VkPushConstantRange;
```

- stageFlags is a set of stage flags describing the shader stages that will access a range of push constants. If a particular stage is not included in the range, then accessing members of that range of push constants from the corresponding shader stage will return undefined values.
- offset and size are the start offset and size, respectively, consumed by the range. Both offset and size are in units of bytes and **must** be a multiple of 4. The layout of the push constant variables is specified in the shader.

### Valid Usage

- VUID-VkPushConstantRange-offset-00294
  offset **must** be less than VkPhysicalDeviceLimits::maxPushConstantsSize
- VUID-VkPushConstantRange-offset-00295
  offset **must** be a multiple of 4
- VUID-VkPushConstantRange-size-00296
  size **must** be greater than 0
- VUID-VkPushConstantRange-size-00297
  size **must** be a multiple of 4

- VUID-VkPushConstantRange-size-00298
  `size` **must** be less than or equal to `VkPhysicalDeviceLimits::maxPushConstantsSize` minus `offset`

### Valid Usage (Implicit)

- VUID-VkPushConstantRange-stageFlags-parameter
  `stageFlags` **must** be a valid combination of VkShaderStageFlagBits values
- VUID-VkPushConstantRange-stageFlags-requiredbitmask
  `stageFlags` **must** not be `0`

Once created, pipeline layouts are used as part of pipeline creation (see Pipelines), as part of binding descriptor sets (see Descriptor Set Binding), and as part of setting push constants (see Push Constant Updates). Pipeline creation accepts a pipeline layout as input, and the layout **may** be used to map (set, binding, arrayElement) tuples to implementation resources or memory locations within a descriptor set. The assignment of implementation resources depends only on the bindings defined in the descriptor sets that comprise the pipeline layout, and not on any shader source.

All resource variables statically used in all shaders in a pipeline **must** be declared with a (set, binding, arrayElement) that exists in the corresponding descriptor set layout and is of an appropriate descriptor type and includes the set of shader stages it is used by in `stageFlags`. The pipeline layout **can** include entries that are not used by a particular pipeline. The pipeline layout allows the application to provide a consistent set of bindings across multiple pipeline compiles, which enables those pipelines to be compiled in a way that the implementation **may** cheaply switch pipelines without reprogramming the bindings.

Similarly, the push constant block declared in each shader (if present) **must** only place variables at offsets that are each included in a push constant range with `stageFlags` including the bit corresponding to the shader stage that uses it. The pipeline layout **can** include ranges or portions of ranges that are not used by a particular pipeline.

There is a limit on the total number of resources of each type that **can** be included in bindings in all descriptor set layouts in a pipeline layout as shown in Pipeline Layout Resource Limits. The “Total Resources Available” column gives the limit on the number of each type of resource that **can** be included in bindings in all descriptor sets in the pipeline layout. Some resource types count against multiple limits. Additionally, there are limits on the total number of each type of resource that **can** be used in any pipeline stage as described in Shader Resource Limits.

*Table 18. Pipeline Layout Resource Limits*

| Total Resources Available | Resource Types |
|---|---|
| `maxDescriptorSetSamplers` or `maxDescriptorSetUpdateAfterBindSamplers` | sampler |
| | combined image sampler |

| Total Resources Available | Resource Types |
|---|---|
| maxDescriptorSetSampledImages or maxDescriptorSetUpdateAfterBindSampledImages | sampled image |
| | combined image sampler |
| | uniform texel buffer |
| maxDescriptorSetStorageImages or maxDescriptorSetUpdateAfterBindStorageImages | storage image |
| | storage texel buffer |
| maxDescriptorSetUniformBuffers or maxDescriptorSetUpdateAfterBindUniformBuffers | uniform buffer |
| | uniform buffer dynamic |
| maxDescriptorSetUniformBuffersDynamic or maxDescriptorSetUpdateAfterBindUniformBuffersDynamic or maxDescriptorSetUpdateAfterBindTotalUniformBuffersDynamic | uniform buffer dynamic |
| maxDescriptorSetStorageBuffers or maxDescriptorSetUpdateAfterBindStorageBuffers | storage buffer |
| | storage buffer dynamic |
| maxDescriptorSetStorageBuffersDynamic or maxDescriptorSetUpdateAfterBindStorageBuffersDynamic or maxDescriptorSetUpdateAfterBindTotalStorageBuffersDynamic | storage buffer dynamic |
| maxDescriptorSetInputAttachments or maxDescriptorSetUpdateAfterBindInputAttachments | input attachment |
| maxDescriptorSetInlineUniformBlocks or maxDescriptorSetUpdateAfterBindInlineUniformBlocks | inline uniform block |
| maxDescriptorSetAccelerationStructures or maxDescriptorSetUpdateAfterBindAccelerationStructures | acceleration structure |

To destroy a pipeline layout, call:

```
// Provided by VK_VERSION_1_0
void vkDestroyPipelineLayout(
    VkDevice                                    device,
    VkPipelineLayout                            pipelineLayout,
    const VkAllocationCallbacks*                pAllocator);
```

- device is the logical device that destroys the pipeline layout.
- pipelineLayout is the pipeline layout to destroy.
- pAllocator controls host memory allocation as described in the Memory Allocation chapter.

## Valid Usage

- VUID-vkDestroyPipelineLayout-pipelineLayout-00299
  If `VkAllocationCallbacks` were provided when `pipelineLayout` was created, a compatible set of callbacks **must** be provided here
- VUID-vkDestroyPipelineLayout-pipelineLayout-00300
  If no `VkAllocationCallbacks` were provided when `pipelineLayout` was created, `pAllocator` **must** be `NULL`
- VUID-vkDestroyPipelineLayout-pipelineLayout-02004
  `pipelineLayout` **must** not have been passed to any `vkCmd*` command for any command buffers that are still in the recording state when `vkDestroyPipelineLayout` is called

## Valid Usage (Implicit)

- VUID-vkDestroyPipelineLayout-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkDestroyPipelineLayout-pipelineLayout-parameter
  If `pipelineLayout` is not VK_NULL_HANDLE, `pipelineLayout` **must** be a valid VkPipelineLayout handle
- VUID-vkDestroyPipelineLayout-pAllocator-parameter
  If `pAllocator` is not `NULL`, `pAllocator` **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkDestroyPipelineLayout-pipelineLayout-parent
  If `pipelineLayout` is a valid handle, it **must** have been created, allocated, or retrieved from `device`

## Host Synchronization

- Host access to `pipelineLayout` **must** be externally synchronized

### Pipeline Layout Compatibility

Two pipeline layouts are defined to be "compatible for push constants" if they were created with identical push constant ranges. Two pipeline layouts are defined to be "compatible for set N" if they were created with *identically defined* descriptor set layouts for sets zero through N, if both of them either were or were not created with `VK_PIPELINE_LAYOUT_CREATE_INDEPENDENT_SETS_BIT_EXT`, and if they were created with identical push constant ranges.

When binding a descriptor set (see Descriptor Set Binding) to set number N, a previously bound descriptor set bound with lower index M than N is disturbed if the pipeline layouts for set M and N are not compatible for set M. Otherwise, the bound descriptor set in M is not disturbed.

If, additionally, the previously bound descriptor set for set N was bound using a pipeline layout not

compatible for set N, then all bindings in sets numbered greater than N are disturbed.

When binding a pipeline, the pipeline **can** correctly access any previously bound descriptor set N if it was bound with compatible pipeline layout for set N, and it was not disturbed.

Layout compatibility means that descriptor sets **can** be bound to a command buffer for use by any pipeline created with a compatible pipeline layout, and without having bound a particular pipeline first. It also means that descriptor sets **can** remain valid across a pipeline change, and the same resources will be accessible to the newly bound pipeline.

When a descriptor set is disturbed by binding descriptor sets, the disturbed set is considered to contain undefined descriptors bound with the same pipeline layout as the disturbing descriptor set.

### Implementor's Note

A consequence of layout compatibility is that when the implementation compiles a pipeline layout and maps pipeline resources to implementation resources, the mechanism for set N **should** only be a function of sets [0..N].

| | |
|---|---|
| **NOTE** | Place the least frequently changing descriptor sets near the start of the pipeline layout, and place the descriptor sets representing the most frequently changing resources near the end. When pipelines are switched, only the descriptor set bindings that have been invalidated will need to be updated and the remainder of the descriptor set bindings will remain in place. |

The maximum number of descriptor sets that **can** be bound to a pipeline layout is queried from physical device properties (see `maxBoundDescriptorSets` in Limits).

*API example*

```
const VkDescriptorSetLayout layouts[] = { layout1, layout2 };

const VkPushConstantRange ranges[] =
{
    {
        .stageFlags = VK_SHADER_STAGE_VERTEX_BIT,
        .offset = 0,
        .size = 4
    },
    {
        .stageFlags = VK_SHADER_STAGE_FRAGMENT_BIT,
        .offset = 4,
        .size = 4
    },
};

const VkPipelineLayoutCreateInfo createInfo =
{
    .sType = VK_STRUCTURE_TYPE_PIPELINE_LAYOUT_CREATE_INFO,
```

```
    .pNext = NULL,
    .flags = 0,
    .setLayoutCount = 2,
    .pSetLayouts = layouts,
    .pushConstantRangeCount = 2,
    .pPushConstantRanges = ranges
};

VkPipelineLayout myPipelineLayout;
myResult = vkCreatePipelineLayout(
    myDevice,
    &createInfo,
    NULL,
    &myPipelineLayout);
```

### 14.2.3. Allocation of Descriptor Sets

A *descriptor pool* maintains a pool of descriptors, from which descriptor sets are allocated. Descriptor pools are externally synchronized, meaning that the application **must** not allocate and/or free descriptor sets from the same pool in multiple threads simultaneously.

Descriptor pools are represented by `VkDescriptorPool` handles:

```
// Provided by VK_VERSION_1_0
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkDescriptorPool)
```

To create a descriptor pool object, call:

```
// Provided by VK_VERSION_1_0
VkResult vkCreateDescriptorPool(
    VkDevice                                    device,
    const VkDescriptorPoolCreateInfo*           pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkDescriptorPool*                           pDescriptorPool);
```

- `device` is the logical device that creates the descriptor pool.
- `pCreateInfo` is a pointer to a VkDescriptorPoolCreateInfo structure specifying the state of the descriptor pool object.
- `pAllocator` controls host memory allocation as described in the Memory Allocation chapter.
- `pDescriptorPool` is a pointer to a VkDescriptorPool handle in which the resulting descriptor pool object is returned.

The created descriptor pool is returned in `pDescriptorPool`.

### Valid Usage (Implicit)

- VUID-vkCreateDescriptorPool-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkCreateDescriptorPool-pCreateInfo-parameter
  `pCreateInfo` **must** be a valid pointer to a valid VkDescriptorPoolCreateInfo structure
- VUID-vkCreateDescriptorPool-pAllocator-parameter
  If `pAllocator` is not `NULL`, `pAllocator` **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkCreateDescriptorPool-pDescriptorPool-parameter
  `pDescriptorPool` **must** be a valid pointer to a VkDescriptorPool handle

### Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_FRAGMENTATION_EXT`

Additional information about the pool is passed in a `VkDescriptorPoolCreateInfo` structure:

```
// Provided by VK_VERSION_1_0
typedef struct VkDescriptorPoolCreateInfo {
    VkStructureType                sType;
    const void*                    pNext;
    VkDescriptorPoolCreateFlags    flags;
    uint32_t                       maxSets;
    uint32_t                       poolSizeCount;
    const VkDescriptorPoolSize*    pPoolSizes;
} VkDescriptorPoolCreateInfo;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is a bitmask of VkDescriptorPoolCreateFlagBits specifying certain supported operations on the pool.
- `maxSets` is the maximum number of descriptor sets that **can** be allocated from the pool.
- `poolSizeCount` is the number of elements in `pPoolSizes`.
- `pPoolSizes` is a pointer to an array of VkDescriptorPoolSize structures, each containing a descriptor type and number of descriptors of that type to be allocated in the pool.

If multiple VkDescriptorPoolSize structures containing the same descriptor type appear in the pPoolSizes array then the pool will be created with enough storage for the total number of descriptors of each type.

Fragmentation of a descriptor pool is possible and **may** lead to descriptor set allocation failures. A failure due to fragmentation is defined as failing a descriptor set allocation despite the sum of all outstanding descriptor set allocations from the pool plus the requested allocation requiring no more than the total number of descriptors requested at pool creation. Implementations provide certain guarantees of when fragmentation **must** not cause allocation failure, as described below.

If a descriptor pool has not had any descriptor sets freed since it was created or most recently reset then fragmentation **must** not cause an allocation failure (note that this is always the case for a pool created without the VK_DESCRIPTOR_POOL_CREATE_FREE_DESCRIPTOR_SET_BIT bit set). Additionally, if all sets allocated from the pool since it was created or most recently reset use the same number of descriptors (of each type) and the requested allocation also uses that same number of descriptors (of each type), then fragmentation **must** not cause an allocation failure.

If an allocation failure occurs due to fragmentation, an application **can** create an additional descriptor pool to perform further descriptor set allocations.

If flags has the VK_DESCRIPTOR_POOL_CREATE_UPDATE_AFTER_BIND_BIT bit set, descriptor pool creation **may** fail with the error VK_ERROR_FRAGMENTATION if the total number of descriptors across all pools (including this one) created with this bit set exceeds maxUpdateAfterBindDescriptorsInAllPools, or if fragmentation of the underlying hardware resources occurs.

If a pPoolSizes[i]::type is VK_DESCRIPTOR_TYPE_MUTABLE_EXT, a VkMutableDescriptorTypeCreateInfoEXT struct in the pNext chain **can** be used to specify which mutable descriptor types **can** be allocated from the pool. If included in the pNext chain, VkMutableDescriptorTypeCreateInfoEXT ::pMutableDescriptorTypeLists[i] specifies which kind of VK_DESCRIPTOR_TYPE_MUTABLE_EXT descriptors **can** be allocated from this pool entry. If VkMutableDescriptorTypeCreateInfoEXT does not exist in the pNext chain, or VkMutableDescriptorTypeCreateInfoEXT::pMutableDescriptorTypeLists[i] is out of range, the descriptor pool allocates enough memory to be able to allocate a VK_DESCRIPTOR_TYPE_MUTABLE_EXT descriptor with any supported VkDescriptorType as a mutable descriptor. A mutable descriptor **can** be allocated from a pool entry if the type list in VkDescriptorSetLayoutCreateInfo is a subset of the type list declared in the descriptor pool, or if the pool entry is created without a descriptor type list. Multiple pPoolSizes entries with VK_DESCRIPTOR_TYPE_MUTABLE_EXT **can** be declared. When multiple such pool entries are present in pPoolSizes, they specify sets of supported descriptor types which either fully overlap, partially overlap, or are disjoint. Two sets fully overlap if the sets of supported descriptor types are equal. If the sets are not disjoint they partially overlap. A pool entry without a VkMutableDescriptorTypeListEXT assigned to it is considered to partially overlap any other pool entry which has a VkMutableDescriptorTypeListEXT assigned to it. The application **must** ensure that partial overlap does not exist in pPoolSizes.

| | |
|---|---|
| **NOTE** | The requirement of no partial overlap is intended to resolve ambiguity for validation as there is no confusion which pPoolSizes entries will be allocated from. An implementation is not expected to depend on this requirement. |

## Valid Usage

- VUID-VkDescriptorPoolCreateInfo-descriptorPoolOverallocation-09227
  If the `descriptorPoolOverallocation` feature is not enabled, or `flags` does not have `VK_DESCRIPTOR_POOL_CREATE_ALLOW_OVERALLOCATION_SETS_BIT_NV` set, `maxSets` **must** be greater than `0`
- VUID-VkDescriptorPoolCreateInfo-flags-09228
  If `flags` has the `VK_DESCRIPTOR_POOL_CREATE_ALLOW_OVERALLOCATION_SETS_BIT_NV` or `VK_DESCRIPTOR_POOL_CREATE_ALLOW_OVERALLOCATION_POOLS_BIT_NV` bits set, then `descriptorPoolOverallocation` **must** be enabled
- VUID-VkDescriptorPoolCreateInfo-flags-04607
  If `flags` has the `VK_DESCRIPTOR_POOL_CREATE_HOST_ONLY_BIT_EXT` bit set, then the `VK_DESCRIPTOR_POOL_CREATE_UPDATE_AFTER_BIND_BIT` bit **must** not be set
- VUID-VkDescriptorPoolCreateInfo-mutableDescriptorType-04608
  If VkPhysicalDeviceMutableDescriptorTypeFeaturesEXT::`mutableDescriptorType` is not enabled, `pPoolSizes` **must** not contain a `descriptorType` of `VK_DESCRIPTOR_TYPE_MUTABLE_EXT`
- VUID-VkDescriptorPoolCreateInfo-flags-04609
  If `flags` has the `VK_DESCRIPTOR_POOL_CREATE_HOST_ONLY_BIT_EXT` bit set, VkPhysicalDeviceMutableDescriptorTypeFeaturesEXT::`mutableDescriptorType` **must** be enabled
- VUID-VkDescriptorPoolCreateInfo-pPoolSizes-04787
  If `pPoolSizes` contains a `descriptorType` of `VK_DESCRIPTOR_TYPE_MUTABLE_EXT`, any other `VK_DESCRIPTOR_TYPE_MUTABLE_EXT` element in `pPoolSizes` **must** not have sets of supported descriptor types which partially overlap
- VUID-VkDescriptorPoolCreateInfo-pPoolSizes-09424
  If `pPoolSizes` contains a `descriptorType` of `VK_DESCRIPTOR_TYPE_INLINE_UNIFORM_BLOCK`, the `pNext` chain **must** include a VkDescriptorPoolInlineUniformBlockCreateInfo structure whose `maxInlineUniformBlockBindings` member is not zero

## Valid Usage (Implicit)

- VUID-VkDescriptorPoolCreateInfo-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_DESCRIPTOR_POOL_CREATE_INFO`
- VUID-VkDescriptorPoolCreateInfo-pNext-pNext
  Each `pNext` member of any structure (including this one) in the `pNext` chain **must** be either `NULL` or a pointer to a valid instance of VkDescriptorPoolInlineUniformBlockCreateInfo or VkMutableDescriptorTypeCreateInfoEXT
- VUID-VkDescriptorPoolCreateInfo-sType-unique
  The `sType` value of each struct in the `pNext` chain **must** be unique
- VUID-VkDescriptorPoolCreateInfo-flags-parameter
  `flags` **must** be a valid combination of VkDescriptorPoolCreateFlagBits values
- VUID-VkDescriptorPoolCreateInfo-pPoolSizes-parameter

If poolSizeCount is not 0, pPoolSizes **must** be a valid pointer to an array of poolSizeCount valid VkDescriptorPoolSize structures

In order to be able to allocate descriptor sets having inline uniform block bindings the descriptor pool **must** be created with specifying the inline uniform block binding capacity of the descriptor pool, in addition to the total inline uniform data capacity in bytes which is specified through a VkDescriptorPoolSize structure with a descriptorType value of VK_DESCRIPTOR_TYPE_INLINE_UNIFORM_BLOCK. This **can** be done by adding a VkDescriptorPoolInlineUniformBlockCreateInfo structure to the pNext chain of VkDescriptorPoolCreateInfo.

The VkDescriptorPoolInlineUniformBlockCreateInfo structure is defined as:

```
typedef struct VkDescriptorPoolInlineUniformBlockCreateInfo {
    VkStructureType    sType;
    const void*        pNext;
    uint32_t           maxInlineUniformBlockBindings;
} VkDescriptorPoolInlineUniformBlockCreateInfo;
```

or the equivalent

```
// Provided by VK_EXT_inline_uniform_block
typedef VkDescriptorPoolInlineUniformBlockCreateInfo
VkDescriptorPoolInlineUniformBlockCreateInfoEXT;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- maxInlineUniformBlockBindings is the number of inline uniform block bindings to allocate.

### Valid Usage (Implicit)

- VUID-VkDescriptorPoolInlineUniformBlockCreateInfo-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_DESCRIPTOR_POOL_INLINE_UNIFORM_BLOCK_CREATE_INFO

Bits which **can** be set in VkDescriptorPoolCreateInfo::flags, enabling operations on a descriptor pool, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkDescriptorPoolCreateFlagBits {
    VK_DESCRIPTOR_POOL_CREATE_FREE_DESCRIPTOR_SET_BIT = 0x00000001,
    VK_DESCRIPTOR_POOL_CREATE_UPDATE_AFTER_BIND_BIT = 0x00000002,
  // Provided by VK_EXT_mutable_descriptor_type
    VK_DESCRIPTOR_POOL_CREATE_HOST_ONLY_BIT_EXT = 0x00000004,
  // Provided by VK_NV_descriptor_pool_overallocation
    VK_DESCRIPTOR_POOL_CREATE_ALLOW_OVERALLOCATION_SETS_BIT_NV = 0x00000008,
  // Provided by VK_NV_descriptor_pool_overallocation
    VK_DESCRIPTOR_POOL_CREATE_ALLOW_OVERALLOCATION_POOLS_BIT_NV = 0x00000010,
  // Provided by VK_EXT_descriptor_indexing
    VK_DESCRIPTOR_POOL_CREATE_UPDATE_AFTER_BIND_BIT_EXT =
VK_DESCRIPTOR_POOL_CREATE_UPDATE_AFTER_BIND_BIT,
  // Provided by VK_VALVE_mutable_descriptor_type
    VK_DESCRIPTOR_POOL_CREATE_HOST_ONLY_BIT_VALVE =
VK_DESCRIPTOR_POOL_CREATE_HOST_ONLY_BIT_EXT,
} VkDescriptorPoolCreateFlagBits;
```

- `VK_DESCRIPTOR_POOL_CREATE_FREE_DESCRIPTOR_SET_BIT` specifies that descriptor sets **can** return their individual allocations to the pool, i.e. all of vkAllocateDescriptorSets, vkFreeDescriptorSets, and vkResetDescriptorPool are allowed. Otherwise, descriptor sets allocated from the pool **must** not be individually freed back to the pool, i.e. only vkAllocateDescriptorSets and vkResetDescriptorPool are allowed.
- `VK_DESCRIPTOR_POOL_CREATE_UPDATE_AFTER_BIND_BIT` specifies that descriptor sets allocated from this pool **can** include bindings with the `VK_DESCRIPTOR_BINDING_UPDATE_AFTER_BIND_BIT` bit set. It is valid to allocate descriptor sets that have bindings that do not set the `VK_DESCRIPTOR_BINDING_UPDATE_AFTER_BIND_BIT` bit from a pool that has `VK_DESCRIPTOR_POOL_CREATE_UPDATE_AFTER_BIND_BIT` set.
- `VK_DESCRIPTOR_POOL_CREATE_HOST_ONLY_BIT_EXT` specifies that this descriptor pool and the descriptor sets allocated from it reside entirely in host memory and cannot be bound. Similar to descriptor sets allocated without this flag, applications **can** copy-from and copy-to descriptors sets allocated from this descriptor pool. Descriptor sets allocated from this pool are partially exempt from the external synchronization requirement in vkUpdateDescriptorSetWithTemplateKHR and vkUpdateDescriptorSets. Descriptor sets and their descriptors can be updated concurrently in different threads, though the same descriptor **must** not be updated concurrently by two threads.
- `VK_DESCRIPTOR_POOL_CREATE_ALLOW_OVERALLOCATION_SETS_BIT_NV` specifies that the implementation should allow the application to allocate more than VkDescriptorPoolCreateInfo::`maxSets` descriptor set objects from the descriptor pool as available resources allow. The implementation **may** use the `maxSets` value to allocate the initial available sets, but using zero is permitted.
- `VK_DESCRIPTOR_POOL_CREATE_ALLOW_OVERALLOCATION_POOLS_BIT_NV` specifies that the implementation should allow the application to allocate more descriptors from the pool than was specified by the VkDescriptorPoolSize::`descriptorCount` for any descriptor type as specified by VkDescriptorPoolCreateInfo::`poolSizeCount` and VkDescriptorPoolCreateInfo::`pPoolSizes`, as available resources allow. The implementation **may** use the `descriptorCount` for each descriptor type to allocate the initial pool, but the application is allowed to set the `poolSizeCount` to zero, or

any of the descriptorCount values in the pPoolSizes array to zero.

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkDescriptorPoolCreateFlags;
```

VkDescriptorPoolCreateFlags is a bitmask type for setting a mask of zero or more VkDescriptorPoolCreateFlagBits.

The VkDescriptorPoolSize structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkDescriptorPoolSize {
    VkDescriptorType    type;
    uint32_t            descriptorCount;
} VkDescriptorPoolSize;
```

- type is the type of descriptor.
- descriptorCount is the number of descriptors of that type to allocate. If type is VK_DESCRIPTOR_TYPE_INLINE_UNIFORM_BLOCK then descriptorCount is the number of bytes to allocate for descriptors of this type.

**NOTE**

When creating a descriptor pool that will contain descriptors for combined image samplers of multi-planar formats, an application needs to account for non-trivial descriptor consumption when choosing the descriptorCount value, as indicated by VkSamplerYcbcrConversionImageFormatProperties::combinedImageSamplerDescriptorCount.

For simplicity the application **can** use the VkPhysicalDeviceMaintenance6PropertiesKHR::maxCombinedImageSamplerDescriptorCount property, which is sized to accommodate any and all formats that require a sampler $Y'C_BC_R$ conversion supported by the implementation.

## Valid Usage

- VUID-VkDescriptorPoolSize-descriptorCount-00302
  descriptorCount **must** be greater than 0
- VUID-VkDescriptorPoolSize-type-02218
  If type is VK_DESCRIPTOR_TYPE_INLINE_UNIFORM_BLOCK then descriptorCount **must** be a multiple of 4

## Valid Usage (Implicit)

- VUID-VkDescriptorPoolSize-type-parameter
  type **must** be a valid VkDescriptorType value

To destroy a descriptor pool, call:

```
// Provided by VK_VERSION_1_0
void vkDestroyDescriptorPool(
    VkDevice                                    device,
    VkDescriptorPool                            descriptorPool,
    const VkAllocationCallbacks*                pAllocator);
```

- `device` is the logical device that destroys the descriptor pool.
- `descriptorPool` is the descriptor pool to destroy.
- `pAllocator` controls host memory allocation as described in the Memory Allocation chapter.

When a pool is destroyed, all descriptor sets allocated from the pool are implicitly freed and become invalid. Descriptor sets allocated from a given pool do not need to be freed before destroying that descriptor pool.

### Valid Usage

- VUID-vkDestroyDescriptorPool-descriptorPool-00303
  All submitted commands that refer to `descriptorPool` (via any allocated descriptor sets) **must** have completed execution
- VUID-vkDestroyDescriptorPool-descriptorPool-00304
  If `VkAllocationCallbacks` were provided when `descriptorPool` was created, a compatible set of callbacks **must** be provided here
- VUID-vkDestroyDescriptorPool-descriptorPool-00305
  If no `VkAllocationCallbacks` were provided when `descriptorPool` was created, `pAllocator` **must** be `NULL`

### Valid Usage (Implicit)

- VUID-vkDestroyDescriptorPool-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkDestroyDescriptorPool-descriptorPool-parameter
  If `descriptorPool` is not VK_NULL_HANDLE, `descriptorPool` **must** be a valid VkDescriptorPool handle
- VUID-vkDestroyDescriptorPool-pAllocator-parameter
  If `pAllocator` is not `NULL`, `pAllocator` **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkDestroyDescriptorPool-descriptorPool-parent
  If `descriptorPool` is a valid handle, it **must** have been created, allocated, or retrieved from `device`

### Host Synchronization

- Host access to descriptorPool **must** be externally synchronized

Descriptor sets are allocated from descriptor pool objects, and are represented by VkDescriptorSet handles:

```
// Provided by VK_VERSION_1_0
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkDescriptorSet)
```

To allocate descriptor sets from a descriptor pool, call:

```
// Provided by VK_VERSION_1_0
VkResult vkAllocateDescriptorSets(
    VkDevice                                    device,
    const VkDescriptorSetAllocateInfo*          pAllocateInfo,
    VkDescriptorSet*                            pDescriptorSets);
```

- device is the logical device that owns the descriptor pool.
- pAllocateInfo is a pointer to a VkDescriptorSetAllocateInfo structure describing parameters of the allocation.
- pDescriptorSets is a pointer to an array of VkDescriptorSet handles in which the resulting descriptor set objects are returned.

The allocated descriptor sets are returned in pDescriptorSets.

When a descriptor set is allocated, the initial state is largely uninitialized and all descriptors are undefined, with the exception that samplers with a non-null pImmutableSamplers are initialized on allocation. Descriptors also become undefined if the underlying resource or view object is destroyed. Descriptor sets containing undefined descriptors **can** still be bound and used, subject to the following conditions:

- For descriptor set bindings created with the VK_DESCRIPTOR_BINDING_PARTIALLY_BOUND_BIT bit set, all descriptors in that binding that are dynamically used **must** have been populated before the descriptor set is consumed.
- For descriptor set bindings created without the VK_DESCRIPTOR_BINDING_PARTIALLY_BOUND_BIT bit set, all descriptors in that binding that are statically used **must** have been populated before the descriptor set is consumed.
- Descriptor bindings with descriptor type of VK_DESCRIPTOR_TYPE_INLINE_UNIFORM_BLOCK **can** be undefined when the descriptor set is consumed; though values in that block will be undefined.
- Entries that are not used by a pipeline **can** have undefined descriptors.

If a call to vkAllocateDescriptorSets would cause the total number of descriptor sets allocated from the pool to exceed the value of VkDescriptorPoolCreateInfo::maxSets used to create pAllocateInfo-

>descriptorPool, then the allocation **may** fail due to lack of space in the descriptor pool. Similarly, the allocation **may** fail due to lack of space if the call to vkAllocateDescriptorSets would cause the number of any given descriptor type to exceed the sum of all the descriptorCount members of each element of VkDescriptorPoolCreateInfo::pPoolSizes with a type equal to that type.

Additionally, the allocation **may** also fail if a call to vkAllocateDescriptorSets would cause the total number of inline uniform block bindings allocated from the pool to exceed the value of VkDescriptorPoolInlineUniformBlockCreateInfo::maxInlineUniformBlockBindings used to create the descriptor pool.

If the allocation fails due to no more space in the descriptor pool, and not because of system or device memory exhaustion, then VK_ERROR_OUT_OF_POOL_MEMORY **must** be returned.

vkAllocateDescriptorSets **can** be used to create multiple descriptor sets. If the creation of any of those descriptor sets fails, then the implementation **must** destroy all successfully created descriptor set objects from this command, set all entries of the pDescriptorSets array to VK_NULL_HANDLE and return the error.

## Valid Usage (Implicit)

- VUID-vkAllocateDescriptorSets-device-parameter
  device **must** be a valid VkDevice handle
- VUID-vkAllocateDescriptorSets-pAllocateInfo-parameter
  pAllocateInfo **must** be a valid pointer to a valid VkDescriptorSetAllocateInfo structure
- VUID-vkAllocateDescriptorSets-pDescriptorSets-parameter
  pDescriptorSets **must** be a valid pointer to an array of pAllocateInfo->descriptorSetCount VkDescriptorSet handles
- VUID-vkAllocateDescriptorSets-pAllocateInfo::descriptorSetCount-arraylength
  pAllocateInfo->descriptorSetCount **must** be greater than 0

## Host Synchronization

- Host access to pAllocateInfo->descriptorPool **must** be externally synchronized

## Return Codes

**Success**

- VK_SUCCESS

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY
- VK_ERROR_FRAGMENTED_POOL
- VK_ERROR_OUT_OF_POOL_MEMORY

The VkDescriptorSetAllocateInfo structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkDescriptorSetAllocateInfo {
    VkStructureType                 sType;
    const void*                     pNext;
    VkDescriptorPool                descriptorPool;
    uint32_t                        descriptorSetCount;
    const VkDescriptorSetLayout*    pSetLayouts;
} VkDescriptorSetAllocateInfo;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- descriptorPool is the pool which the sets will be allocated from.
- descriptorSetCount determines the number of descriptor sets to be allocated from the pool.
- pSetLayouts is a pointer to an array of descriptor set layouts, with each member specifying how the corresponding descriptor set is allocated.

## Valid Usage

- VUID-VkDescriptorSetAllocateInfo-apiVersion-07895
  If the VK_KHR_maintenance1 extension is not enabled and VkPhysicalDeviceProperties::apiVersion is less than Vulkan 1.1, descriptorSetCount **must** not be greater than the number of sets that are currently available for allocation in descriptorPool
- VUID-VkDescriptorSetAllocateInfo-apiVersion-07896
  If the VK_KHR_maintenance1 extension is not enabled and VkPhysicalDeviceProperties::apiVersion is less than Vulkan 1.1, descriptorPool **must** have enough free descriptor capacity remaining to allocate the descriptor sets of the specified layouts
- VUID-VkDescriptorSetAllocateInfo-pSetLayouts-00308
  Each element of pSetLayouts **must** not have been created with VK_DESCRIPTOR_SET_LAYOUT_CREATE_PUSH_DESCRIPTOR_BIT_KHR set
- VUID-VkDescriptorSetAllocateInfo-pSetLayouts-03044
  If any element of pSetLayouts was created with the VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT bit set, descriptorPool **must** have been created with the VK_DESCRIPTOR_POOL_CREATE_UPDATE_AFTER_BIND_BIT flag set
- VUID-VkDescriptorSetAllocateInfo-pSetLayouts-09380
  If pSetLayouts[i] was created with an element of pBindingFlags that includes VK_DESCRIPTOR_BINDING_VARIABLE_DESCRIPTOR_COUNT_BIT, and VkDescriptorSetVariableDescriptorCountAllocateInfo is included in the pNext chain, and VkDescriptorSetVariableDescriptorCountAllocateInfo::descriptorSetCount is not zero, then VkDescriptorSetVariableDescriptorCountAllocateInfo::pDescriptorCounts[i] **must** be less than or equal to VkDescriptorSetLayoutBinding::descriptorCount for the corresponding binding used to create pSetLayouts[i]

- VUID-VkDescriptorSetAllocateInfo-pSetLayouts-04610
  If any element of pSetLayouts was created with the VK_DESCRIPTOR_SET_LAYOUT_CREATE_HOST_ONLY_POOL_BIT_EXT bit set, descriptorPool **must** have been created with the VK_DESCRIPTOR_POOL_CREATE_HOST_ONLY_BIT_EXT flag set
- VUID-VkDescriptorSetAllocateInfo-pSetLayouts-08009
  Each element of pSetLayouts **must** not have been created with the VK_DESCRIPTOR_SET_LAYOUT_CREATE_DESCRIPTOR_BUFFER_BIT_EXT bit set

### Valid Usage (Implicit)

- VUID-VkDescriptorSetAllocateInfo-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_DESCRIPTOR_SET_ALLOCATE_INFO
- VUID-VkDescriptorSetAllocateInfo-pNext-pNext
  pNext **must** be NULL or a pointer to a valid instance of VkDescriptorSetVariableDescriptorCountAllocateInfo
- VUID-VkDescriptorSetAllocateInfo-sType-unique
  The sType value of each struct in the pNext chain **must** be unique
- VUID-VkDescriptorSetAllocateInfo-descriptorPool-parameter
  descriptorPool **must** be a valid VkDescriptorPool handle
- VUID-VkDescriptorSetAllocateInfo-pSetLayouts-parameter
  pSetLayouts **must** be a valid pointer to an array of descriptorSetCount valid VkDescriptorSetLayout handles
- VUID-VkDescriptorSetAllocateInfo-descriptorSetCount-arraylength
  descriptorSetCount **must** be greater than 0
- VUID-VkDescriptorSetAllocateInfo-commonparent
  Both of descriptorPool, and the elements of pSetLayouts **must** have been created, allocated, or retrieved from the same VkDevice

If the pNext chain of a VkDescriptorSetAllocateInfo structure includes a VkDescriptorSetVariableDescriptorCountAllocateInfo structure, then that structure includes an array of descriptor counts for variable-sized descriptor bindings, one for each descriptor set being allocated.

The VkDescriptorSetVariableDescriptorCountAllocateInfo structure is defined as:

```
typedef struct VkDescriptorSetVariableDescriptorCountAllocateInfo {
    VkStructureType    sType;
    const void*        pNext;
    uint32_t           descriptorSetCount;
    const uint32_t*    pDescriptorCounts;
} VkDescriptorSetVariableDescriptorCountAllocateInfo;
```

or the equivalent

```
// Provided by VK_EXT_descriptor_indexing
typedef VkDescriptorSetVariableDescriptorCountAllocateInfo
VkDescriptorSetVariableDescriptorCountAllocateInfoEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `descriptorSetCount` is zero or the number of elements in `pDescriptorCounts`.
- `pDescriptorCounts` is a pointer to an array of descriptor counts, with each member specifying the number of descriptors in a variable-sized descriptor binding in the corresponding descriptor set being allocated.

If `descriptorSetCount` is zero or this structure is not included in the `pNext` chain, then the variable lengths are considered to be zero. Otherwise, `pDescriptorCounts`[i] is the number of descriptors in the variable-sized descriptor binding in the corresponding descriptor set layout. If the variable-sized descriptor binding in the corresponding descriptor set layout has a descriptor type of `VK_DESCRIPTOR_TYPE_INLINE_UNIFORM_BLOCK` then `pDescriptorCounts`[i] specifies the binding's capacity in bytes. If VkDescriptorSetAllocateInfo::`pSetLayouts`[i] does not include a variable-sized descriptor binding, then `pDescriptorCounts`[i] is ignored.

### Valid Usage

- VUID-VkDescriptorSetVariableDescriptorCountAllocateInfo-descriptorSetCount-03045
  If `descriptorSetCount` is not zero, `descriptorSetCount` **must** equal VkDescriptorSetAllocateInfo::`descriptorSetCount`

### Valid Usage (Implicit)

- VUID-VkDescriptorSetVariableDescriptorCountAllocateInfo-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_DESCRIPTOR_SET_VARIABLE_DESCRIPTOR_COUNT_ALLOCATE_INFO`
- VUID-VkDescriptorSetVariableDescriptorCountAllocateInfo-pDescriptorCounts-parameter
  If `descriptorSetCount` is not `0`, `pDescriptorCounts` **must** be a valid pointer to an array of `descriptorSetCount` `uint32_t` values

To free allocated descriptor sets, call:

```
// Provided by VK_VERSION_1_0
VkResult vkFreeDescriptorSets(
    VkDevice                                    device,
    VkDescriptorPool                            descriptorPool,
    uint32_t                                    descriptorSetCount,
    const VkDescriptorSet*                      pDescriptorSets);
```

- `device` is the logical device that owns the descriptor pool.
- `descriptorPool` is the descriptor pool from which the descriptor sets were allocated.
- `descriptorSetCount` is the number of elements in the `pDescriptorSets` array.
- `pDescriptorSets` is a pointer to an array of handles to VkDescriptorSet objects.

After calling `vkFreeDescriptorSets`, all descriptor sets in `pDescriptorSets` are invalid.

## Valid Usage

- VUID-vkFreeDescriptorSets-pDescriptorSets-00309
  All submitted commands that refer to any element of `pDescriptorSets` **must** have completed execution
- VUID-vkFreeDescriptorSets-pDescriptorSets-00310
  `pDescriptorSets` **must** be a valid pointer to an array of `descriptorSetCount` `VkDescriptorSet` handles, each element of which **must** either be a valid handle or VK_NULL_HANDLE
- VUID-vkFreeDescriptorSets-descriptorPool-00312
  `descriptorPool` **must** have been created with the `VK_DESCRIPTOR_POOL_CREATE_FREE_DESCRIPTOR_SET_BIT` flag

## Valid Usage (Implicit)

- VUID-vkFreeDescriptorSets-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkFreeDescriptorSets-descriptorPool-parameter
  `descriptorPool` **must** be a valid VkDescriptorPool handle
- VUID-vkFreeDescriptorSets-descriptorSetCount-arraylength
  `descriptorSetCount` **must** be greater than `0`
- VUID-vkFreeDescriptorSets-descriptorPool-parent
  `descriptorPool` **must** have been created, allocated, or retrieved from `device`
- VUID-vkFreeDescriptorSets-pDescriptorSets-parent
  Each element of `pDescriptorSets` that is a valid handle **must** have been created, allocated, or retrieved from `descriptorPool`

## Host Synchronization

- Host access to `descriptorPool` **must** be externally synchronized
- Host access to each member of `pDescriptorSets` **must** be externally synchronized

## Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

None

To return all descriptor sets allocated from a given pool to the pool, rather than freeing individual descriptor sets, call:

```
// Provided by VK_VERSION_1_0
VkResult vkResetDescriptorPool(
    VkDevice                                    device,
    VkDescriptorPool                            descriptorPool,
    VkDescriptorPoolResetFlags                  flags);
```

- `device` is the logical device that owns the descriptor pool.
- `descriptorPool` is the descriptor pool to be reset.
- `flags` is reserved for future use.

Resetting a descriptor pool recycles all of the resources from all of the descriptor sets allocated from the descriptor pool back to the descriptor pool, and the descriptor sets are implicitly freed.

## Valid Usage

- VUID-vkResetDescriptorPool-descriptorPool-00313
  All uses of `descriptorPool` (via any allocated descriptor sets) **must** have completed execution

## Valid Usage (Implicit)

- VUID-vkResetDescriptorPool-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkResetDescriptorPool-descriptorPool-parameter
  `descriptorPool` **must** be a valid VkDescriptorPool handle
- VUID-vkResetDescriptorPool-flags-zerobitmask
  `flags` **must** be `0`
- VUID-vkResetDescriptorPool-descriptorPool-parent
  `descriptorPool` **must** have been created, allocated, or retrieved from `device`

### Host Synchronization

- Host access to `descriptorPool` **must** be externally synchronized
- Host access to any `VkDescriptorSet` objects allocated from `descriptorPool` **must** be externally synchronized

### Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

None

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkDescriptorPoolResetFlags;
```

`VkDescriptorPoolResetFlags` is a bitmask type for setting a mask, but is currently reserved for future use.

## 14.2.4. Descriptor Set Updates

Once allocated, descriptor sets **can** be updated with a combination of write and copy operations. To update descriptor sets, call:

```
// Provided by VK_VERSION_1_0
void vkUpdateDescriptorSets(
    VkDevice                                    device,
    uint32_t                                    descriptorWriteCount,
    const VkWriteDescriptorSet*                 pDescriptorWrites,
    uint32_t                                    descriptorCopyCount,
    const VkCopyDescriptorSet*                  pDescriptorCopies);
```

- `device` is the logical device that updates the descriptor sets.
- `descriptorWriteCount` is the number of elements in the `pDescriptorWrites` array.
- `pDescriptorWrites` is a pointer to an array of VkWriteDescriptorSet structures describing the descriptor sets to write to.
- `descriptorCopyCount` is the number of elements in the `pDescriptorCopies` array.
- `pDescriptorCopies` is a pointer to an array of VkCopyDescriptorSet structures describing the descriptor sets to copy between.

The operations described by `pDescriptorWrites` are performed first, followed by the operations described by `pDescriptorCopies`. Within each array, the operations are performed in the order they

appear in the array.

Each element in the pDescriptorWrites array describes an operation updating the descriptor set using descriptors for resources specified in the structure.

Each element in the pDescriptorCopies array is a VkCopyDescriptorSet structure describing an operation copying descriptors between sets.

If the dstSet member of any element of pDescriptorWrites or pDescriptorCopies is bound, accessed, or modified by any command that was recorded to a command buffer which is currently in the recording or executable state, and any of the descriptor bindings that are updated were not created with the VK_DESCRIPTOR_BINDING_UPDATE_AFTER_BIND_BIT or VK_DESCRIPTOR_BINDING_UPDATE_UNUSED_WHILE_PENDING_BIT bits set, that command buffer becomes invalid.

## Valid Usage

- VUID-vkUpdateDescriptorSets-pDescriptorWrites-06236
  For each element i where pDescriptorWrites[i].descriptorType is VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER or VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER, elements of the pTexelBufferView member of pDescriptorWrites[i] **must** have been created on device
- VUID-vkUpdateDescriptorSets-pDescriptorWrites-06237
  For each element i where pDescriptorWrites[i].descriptorType is VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER, VK_DESCRIPTOR_TYPE_STORAGE_BUFFER, VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC, or VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC, the buffer member of any element of the pBufferInfo member of pDescriptorWrites[i] **must** have been created on device
- VUID-vkUpdateDescriptorSets-pDescriptorWrites-06238
  For each element i where pDescriptorWrites[i].descriptorType is VK_DESCRIPTOR_TYPE_SAMPLER or VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER, and dstSet was not allocated with a layout that included immutable samplers for dstBinding with descriptorType, the sampler member of any element of the pImageInfo member of pDescriptorWrites[i] **must** have been created on device
- VUID-vkUpdateDescriptorSets-pDescriptorWrites-06239
  For each element i where pDescriptorWrites[i].descriptorType is VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE, VK_DESCRIPTOR_TYPE_STORAGE_IMAGE, VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT, or VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER the imageView member of any element of pDescriptorWrites[i] **must** have been created on device
- VUID-vkUpdateDescriptorSets-pDescriptorWrites-06240
  For each element i where pDescriptorWrites[i].descriptorType is VK_DESCRIPTOR_TYPE_ACCELERATION_STRUCTURE_KHR, elements of the pAccelerationStructures member of a VkWriteDescriptorSetAccelerationStructureKHR structure in the pNext chain of pDescriptorWrites[i] **must** have been created on device
- VUID-vkUpdateDescriptorSets-pDescriptorWrites-06241

For each element i where pDescriptorWrites[i].descriptorType is VK_DESCRIPTOR_TYPE_ACCELERATION_STRUCTURE_NV, elements of the pAccelerationStructures member of a VkWriteDescriptorSetAccelerationStructureNV structure in the pNext chain of pDescriptorWrites[i] **must** have been created on device

- VUID-vkUpdateDescriptorSets-pDescriptorWrites-06940
  For each element i where pDescriptorWrites[i].descriptorType is VK_DESCRIPTOR_TYPE_SAMPLE_WEIGHT_IMAGE_QCOM or VK_DESCRIPTOR_TYPE_BLOCK_MATCH_IMAGE_QCOM, the imageView member of any element of pDescriptorWrites[i] **must** have been created on device

- VUID-vkUpdateDescriptorSets-pDescriptorWrites-06493
  For each element i where pDescriptorWrites[i].descriptorType is VK_DESCRIPTOR_TYPE_SAMPLER, VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER, VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE, VK_DESCRIPTOR_TYPE_STORAGE_IMAGE, or VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT, pDescriptorWrites[i].pImageInfo **must** be a valid pointer to an array of pDescriptorWrites[i].descriptorCount valid VkDescriptorImageInfo structures

- VUID-vkUpdateDescriptorSets-pDescriptorWrites-06941
  For each element i where pDescriptorWrites[i].descriptorType is VK_DESCRIPTOR_TYPE_SAMPLE_WEIGHT_IMAGE_QCOM or VK_DESCRIPTOR_TYPE_BLOCK_MATCH_IMAGE_QCOM, pDescriptorWrites[i].pImageInfo **must** be a valid pointer to an array of pDescriptorWrites[i].descriptorCount valid VkDescriptorImageInfo structures

- VUID-vkUpdateDescriptorSets-None-03047
  The dstSet member of each element of pDescriptorWrites or pDescriptorCopies for bindings which were created without the VK_DESCRIPTOR_BINDING_UPDATE_AFTER_BIND_BIT or VK_DESCRIPTOR_BINDING_UPDATE_UNUSED_WHILE_PENDING_BIT bits set **must** not be used by any command that was recorded to a command buffer which is in the pending state

- VUID-vkUpdateDescriptorSets-pDescriptorWrites-06993
  Host access to pDescriptorWrites[i].dstSet and pDescriptorCopies[i].dstSet **must** be externally synchronized unless explicitly denoted otherwise for specific flags

## Valid Usage (Implicit)

- VUID-vkUpdateDescriptorSets-device-parameter
  device **must** be a valid VkDevice handle

- VUID-vkUpdateDescriptorSets-pDescriptorWrites-parameter
  If descriptorWriteCount is not 0, pDescriptorWrites **must** be a valid pointer to an array of descriptorWriteCount valid VkWriteDescriptorSet structures

- VUID-vkUpdateDescriptorSets-pDescriptorCopies-parameter
  If descriptorCopyCount is not 0, pDescriptorCopies **must** be a valid pointer to an array of descriptorCopyCount valid VkCopyDescriptorSet structures

The VkWriteDescriptorSet structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkWriteDescriptorSet {
    VkStructureType                  sType;
    const void*                      pNext;
    VkDescriptorSet                  dstSet;
    uint32_t                         dstBinding;
    uint32_t                         dstArrayElement;
    uint32_t                         descriptorCount;
    VkDescriptorType                 descriptorType;
    const VkDescriptorImageInfo*     pImageInfo;
    const VkDescriptorBufferInfo*    pBufferInfo;
    const VkBufferView*              pTexelBufferView;
} VkWriteDescriptorSet;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `dstSet` is the destination descriptor set to update.
- `dstBinding` is the descriptor binding within that set.
- `dstArrayElement` is the starting element in that array. If the descriptor binding identified by `dstSet` and `dstBinding` has a descriptor type of `VK_DESCRIPTOR_TYPE_INLINE_UNIFORM_BLOCK` then `dstArrayElement` specifies the starting byte offset within the binding.
- `descriptorCount` is the number of descriptors to update. If the descriptor binding identified by `dstSet` and `dstBinding` has a descriptor type of `VK_DESCRIPTOR_TYPE_INLINE_UNIFORM_BLOCK`, then `descriptorCount` specifies the number of bytes to update. Otherwise, `descriptorCount` is one of
  - the number of elements in `pImageInfo`
  - the number of elements in `pBufferInfo`
  - the number of elements in `pTexelBufferView`
  - a value matching the `dataSize` member of a VkWriteDescriptorSetInlineUniformBlock structure in the `pNext` chain
  - a value matching the `accelerationStructureCount` of a VkWriteDescriptorSetAccelerationStructureKHR structure in the `pNext` chain
- `descriptorType` is a VkDescriptorType specifying the type of each descriptor in `pImageInfo`, `pBufferInfo`, or `pTexelBufferView`, as described below. If `VkDescriptorSetLayoutBinding` for `dstSet` at `dstBinding` is not equal to `VK_DESCRIPTOR_TYPE_MUTABLE_EXT`, `descriptorType` **must** be the same type as the `descriptorType` specified in `VkDescriptorSetLayoutBinding` for `dstSet` at `dstBinding`. The type of the descriptor also controls which array the descriptors are taken from.
- `pImageInfo` is a pointer to an array of VkDescriptorImageInfo structures or is ignored, as described below.
- `pBufferInfo` is a pointer to an array of VkDescriptorBufferInfo structures or is ignored, as described below.
- `pTexelBufferView` is a pointer to an array of VkBufferView handles as described in the Buffer Views section or is ignored, as described below.

Only one of pImageInfo, pBufferInfo, or pTexelBufferView members is used according to the descriptor type specified in the descriptorType member of the containing VkWriteDescriptorSet structure, or none of them in case descriptorType is VK_DESCRIPTOR_TYPE_INLINE_UNIFORM_BLOCK, in which case the source data for the descriptor writes is taken from the VkWriteDescriptorSetInlineUniformBlock structure included in the pNext chain of VkWriteDescriptorSet, or if descriptorType is VK_DESCRIPTOR_TYPE_ACCELERATION_STRUCTURE_KHR, in which case the source data for the descriptor writes is taken from the VkWriteDescriptorSetAccelerationStructureKHR structure in the pNext chain of VkWriteDescriptorSet, or if descriptorType is VK_DESCRIPTOR_TYPE_ACCELERATION_STRUCTURE_NV, in which case the source data for the descriptor writes is taken from the VkWriteDescriptorSetAccelerationStructureNV structure in the pNext chain of VkWriteDescriptorSet, as specified below.

If the nullDescriptor feature is enabled, the buffer, acceleration structure, imageView, or bufferView **can** be VK_NULL_HANDLE. Loads from a null descriptor return zero values and stores and atomics to a null descriptor are discarded. A null acceleration structure descriptor results in the miss shader being invoked.

If the destination descriptor is a mutable descriptor, the active descriptor type for the destination descriptor becomes descriptorType.

If the dstBinding has fewer than descriptorCount array elements remaining starting from dstArrayElement, then the remainder will be used to update the subsequent binding - dstBinding+1 starting at array element zero. If a binding has a descriptorCount of zero, it is skipped. This behavior applies recursively, with the update affecting consecutive bindings as needed to update all descriptorCount descriptors. Consecutive bindings **must** have identical VkDescriptorType, VkShaderStageFlags, VkDescriptorBindingFlagBits, and immutable samplers references. In addition, if the VkDescriptorType is VK_DESCRIPTOR_TYPE_MUTABLE_EXT, the supported descriptor types in VkMutableDescriptorTypeCreateInfoEXT **must** be equally defined.

**NOTE** The same behavior applies to bindings with a descriptor type of VK_DESCRIPTOR_TYPE_INLINE_UNIFORM_BLOCK where descriptorCount specifies the number of bytes to update while dstArrayElement specifies the starting byte offset, thus in this case if the dstBinding has a smaller byte size than the sum of dstArrayElement and descriptorCount, then the remainder will be used to update the subsequent binding - dstBinding+1 starting at offset zero. This falls out as a special case of the above rule.

### Valid Usage

- VUID-VkWriteDescriptorSet-dstBinding-00315
  dstBinding **must** be less than or equal to the maximum value of binding of all VkDescriptorSetLayoutBinding structures specified when dstSet's descriptor set layout was created
- VUID-VkWriteDescriptorSet-dstBinding-00316
  dstBinding **must** be a binding with a non-zero descriptorCount
- VUID-VkWriteDescriptorSet-dstBinding-10009

dstBinding **must** be a binding with a non-zero VkDescriptorSetLayoutCreateInfo::bindingCount

- VUID-VkWriteDescriptorSet-descriptorCount-00317
  All consecutive bindings updated via a single VkWriteDescriptorSet structure, except those with a descriptorCount of zero, **must** have identical descriptorType and stageFlags
- VUID-VkWriteDescriptorSet-descriptorCount-00318
  All consecutive bindings updated via a single VkWriteDescriptorSet structure, except those with a descriptorCount of zero, **must** all either use immutable samplers or **must** all not use immutable samplers
- VUID-VkWriteDescriptorSet-descriptorType-00319
  descriptorType **must** match the type of dstBinding within dstSet
- VUID-VkWriteDescriptorSet-dstSet-00320
  dstSet **must** be a valid VkDescriptorSet handle
- VUID-VkWriteDescriptorSet-dstArrayElement-00321
  The sum of dstArrayElement and descriptorCount **must** be less than or equal to the number of array elements in the descriptor set binding specified by dstBinding, and all applicable consecutive bindings, as described by consecutive binding updates
- VUID-VkWriteDescriptorSet-descriptorType-02219
  If descriptorType is VK_DESCRIPTOR_TYPE_INLINE_UNIFORM_BLOCK, dstArrayElement **must** be an integer multiple of 4
- VUID-VkWriteDescriptorSet-descriptorType-02220
  If descriptorType is VK_DESCRIPTOR_TYPE_INLINE_UNIFORM_BLOCK, descriptorCount **must** be an integer multiple of 4
- VUID-VkWriteDescriptorSet-descriptorType-02994
  If descriptorType is VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER or VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER, each element of pTexelBufferView **must** be either a valid VkBufferView handle or VK_NULL_HANDLE
- VUID-VkWriteDescriptorSet-descriptorType-02995
  If descriptorType is VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER or VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER and the nullDescriptor feature is not enabled, each element of pTexelBufferView **must** not be VK_NULL_HANDLE
- VUID-VkWriteDescriptorSet-descriptorType-00324
  If descriptorType is VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER, VK_DESCRIPTOR_TYPE_STORAGE_BUFFER, VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC, or VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC, pBufferInfo **must** be a valid pointer to an array of descriptorCount valid VkDescriptorBufferInfo structures
- VUID-VkWriteDescriptorSet-descriptorType-00325
  If descriptorType is VK_DESCRIPTOR_TYPE_SAMPLER or VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER, and dstSet was not allocated with a layout that included immutable samplers for dstBinding with descriptorType, the sampler member of each element of pImageInfo **must** be a valid VkSampler object
- VUID-VkWriteDescriptorSet-descriptorType-02996
  If descriptorType is VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER,

VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE, or VK_DESCRIPTOR_TYPE_STORAGE_IMAGE, the imageView member of each element of pImageInfo **must** be either a valid VkImageView handle or VK_NULL_HANDLE

- VUID-VkWriteDescriptorSet-descriptorType-02997
  If descriptorType is VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER, VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE, or VK_DESCRIPTOR_TYPE_STORAGE_IMAGE, and the nullDescriptor feature is not enabled, the imageView member of each element of pImageInfo **must** not be VK_NULL_HANDLE
- VUID-VkWriteDescriptorSet-descriptorType-07683
  If descriptorType is VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT, the imageView member of each element of pImageInfo **must** not be VK_NULL_HANDLE
- VUID-VkWriteDescriptorSet-descriptorType-02221
  If descriptorType is VK_DESCRIPTOR_TYPE_INLINE_UNIFORM_BLOCK, the pNext chain **must** include a VkWriteDescriptorSetInlineUniformBlock structure whose dataSize member equals descriptorCount
- VUID-VkWriteDescriptorSet-descriptorType-02382
  If descriptorType is VK_DESCRIPTOR_TYPE_ACCELERATION_STRUCTURE_KHR, the pNext chain **must** include a VkWriteDescriptorSetAccelerationStructureKHR structure whose accelerationStructureCount member equals descriptorCount
- VUID-VkWriteDescriptorSet-descriptorType-03817
  If descriptorType is VK_DESCRIPTOR_TYPE_ACCELERATION_STRUCTURE_NV, the pNext chain **must** include a VkWriteDescriptorSetAccelerationStructureNV structure whose accelerationStructureCount member equals descriptorCount
- VUID-VkWriteDescriptorSet-descriptorType-01946
  If descriptorType is VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE, then the imageView member of each pImageInfo element **must** have been created without a VkSamplerYcbcrConversionInfo structure in its pNext chain
- VUID-VkWriteDescriptorSet-descriptorType-02738
  If descriptorType is VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER, and if any element of pImageInfo has an imageView member that was created with a VkSamplerYcbcrConversionInfo structure in its pNext chain, then dstSet **must** have been allocated with a layout that included immutable samplers for dstBinding, and the corresponding immutable sampler **must** have been created with an *identically defined* VkSamplerYcbcrConversionInfo object
- VUID-VkWriteDescriptorSet-descriptorType-01948
  If descriptorType is VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER, and dstSet was allocated with a layout that included immutable samplers for dstBinding, then the imageView member of each element of pImageInfo which corresponds to an immutable sampler that enables sampler $Y'C_BC_R$ conversion **must** have been created with a VkSamplerYcbcrConversionInfo structure in its pNext chain with an *identically defined* VkSamplerYcbcrConversionInfo to the corresponding immutable sampler
- VUID-VkWriteDescriptorSet-descriptorType-09506
  If descriptorType is VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER, dstSet was allocated with a layout that included immutable samplers for dstBinding, and those samplers enable

sampler $Y'C_BC_R$ conversion, then imageView **must** not be VK_NULL_HANDLE

- VUID-VkWriteDescriptorSet-descriptorType-00327
  If descriptorType is VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER or VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC, the offset member of each element of pBufferInfo **must** be a multiple of VkPhysicalDeviceLimits::minUniformBufferOffsetAlignment
- VUID-VkWriteDescriptorSet-descriptorType-00328
  If descriptorType is VK_DESCRIPTOR_TYPE_STORAGE_BUFFER or VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC, the offset member of each element of pBufferInfo **must** be a multiple of VkPhysicalDeviceLimits::minStorageBufferOffsetAlignment
- VUID-VkWriteDescriptorSet-descriptorType-00329
  If descriptorType is VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER, VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC, VK_DESCRIPTOR_TYPE_STORAGE_BUFFER, or VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC, and the buffer member of any element of pBufferInfo is the handle of a non-sparse buffer, then that buffer **must** be bound completely and contiguously to a single VkDeviceMemory object
- VUID-VkWriteDescriptorSet-descriptorType-00330
  If descriptorType is VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER or VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC, the buffer member of each element of pBufferInfo **must** have been created with VK_BUFFER_USAGE_UNIFORM_BUFFER_BIT set
- VUID-VkWriteDescriptorSet-descriptorType-00331
  If descriptorType is VK_DESCRIPTOR_TYPE_STORAGE_BUFFER or VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC, the buffer member of each element of pBufferInfo **must** have been created with VK_BUFFER_USAGE_STORAGE_BUFFER_BIT set
- VUID-VkWriteDescriptorSet-descriptorType-00332
  If descriptorType is VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER or VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC, the range member of each element of pBufferInfo, or the effective range if range is VK_WHOLE_SIZE, **must** be less than or equal to VkPhysicalDeviceLimits::maxUniformBufferRange
- VUID-VkWriteDescriptorSet-descriptorType-00333
  If descriptorType is VK_DESCRIPTOR_TYPE_STORAGE_BUFFER or VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC, the range member of each element of pBufferInfo, or the effective range if range is VK_WHOLE_SIZE, **must** be less than or equal to VkPhysicalDeviceLimits::maxStorageBufferRange
- VUID-VkWriteDescriptorSet-descriptorType-08765
  If descriptorType is VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER, the pTexelBufferView buffer view usage **must** include VK_BUFFER_USAGE_UNIFORM_TEXEL_BUFFER_BIT
- VUID-VkWriteDescriptorSet-descriptorType-08766
  If descriptorType is VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER, the pTexelBufferView buffer view usage **must** include VK_BUFFER_USAGE_STORAGE_TEXEL_BUFFER_BIT
- VUID-VkWriteDescriptorSet-descriptorType-00336
  If descriptorType is VK_DESCRIPTOR_TYPE_STORAGE_IMAGE or VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT, the imageView member of each element of pImageInfo

**must** have been created with the identity swizzle

- VUID-VkWriteDescriptorSet-descriptorType-00337
  If descriptorType is VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE or VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER, the imageView member of each element of pImageInfo **must** have been created with VK_IMAGE_USAGE_SAMPLED_BIT set
- VUID-VkWriteDescriptorSet-descriptorType-04149
  If descriptorType is VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE the imageLayout member of each element of pImageInfo **must** be a member of the list given in Sampled Image
- VUID-VkWriteDescriptorSet-descriptorType-04150
  If descriptorType is VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER the imageLayout member of each element of pImageInfo **must** be a member of the list given in Combined Image Sampler
- VUID-VkWriteDescriptorSet-descriptorType-04151
  If descriptorType is VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT the imageLayout member of each element of pImageInfo **must** be a member of the list given in Input Attachment
- VUID-VkWriteDescriptorSet-descriptorType-04152
  If descriptorType is VK_DESCRIPTOR_TYPE_STORAGE_IMAGE the imageLayout member of each element of pImageInfo **must** be a member of the list given in Storage Image
- VUID-VkWriteDescriptorSet-descriptorType-00338
  If descriptorType is VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT, the imageView member of each element of pImageInfo **must** have been created with VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT set
- VUID-VkWriteDescriptorSet-descriptorType-00339
  If descriptorType is VK_DESCRIPTOR_TYPE_STORAGE_IMAGE, the imageView member of each element of pImageInfo **must** have been created with VK_IMAGE_USAGE_STORAGE_BIT set
- VUID-VkWriteDescriptorSet-descriptorType-02752
  If descriptorType is VK_DESCRIPTOR_TYPE_SAMPLER, then dstSet **must** not have been allocated with a layout that included immutable samplers for dstBinding
- VUID-VkWriteDescriptorSet-dstSet-04611
  If the VkDescriptorSetLayoutBinding for dstSet at dstBinding is VK_DESCRIPTOR_TYPE_MUTABLE_EXT, the new active descriptor type descriptorType **must** exist in the corresponding pMutableDescriptorTypeLists list for dstBinding
- VUID-VkWriteDescriptorSet-descriptorType-06450
  If descriptorType is VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT, the imageView member of each element of pImageInfo **must** have either been created without a VkImageViewMinLodCreateInfoEXT included in the pNext chain or with a VkImageViewMinLodCreateInfoEXT::minLod of 0.0
- VUID-VkWriteDescriptorSet-descriptorType-06942
  If descriptorType is VK_DESCRIPTOR_TYPE_SAMPLE_WEIGHT_IMAGE_QCOM, the imageView member of each element of pImageInfo **must** have been created with a view created with an image created with VK_IMAGE_USAGE_SAMPLE_WEIGHT_BIT_QCOM
- VUID-VkWriteDescriptorSet-descriptorType-06943
  If descriptorType is VK_DESCRIPTOR_TYPE_BLOCK_MATCH_IMAGE_QCOM, the imageView member of

each element of `pImageInfo` **must** have been created with a view created with an `image` created with `VK_IMAGE_USAGE_SAMPLE_BLOCK_MATCH_BIT_QCOM`

### Valid Usage (Implicit)

- VUID-VkWriteDescriptorSet-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_WRITE_DESCRIPTOR_SET`
- VUID-VkWriteDescriptorSet-pNext-pNext
  Each `pNext` member of any structure (including this one) in the `pNext` chain **must** be either `NULL` or a pointer to a valid instance of VkWriteDescriptorSetAccelerationStructureKHR, VkWriteDescriptorSetAccelerationStructureNV, or VkWriteDescriptorSetInlineUniformBlock
- VUID-VkWriteDescriptorSet-sType-unique
  The `sType` value of each struct in the `pNext` chain **must** be unique
- VUID-VkWriteDescriptorSet-descriptorType-parameter
  `descriptorType` **must** be a valid VkDescriptorType value
- VUID-VkWriteDescriptorSet-descriptorCount-arraylength
  `descriptorCount` **must** be greater than `0`
- VUID-VkWriteDescriptorSet-commonparent
  Both of `dstSet`, and the elements of `pTexelBufferView` that are valid handles of non-ignored parameters **must** have been created, allocated, or retrieved from the same VkDevice

The type of descriptors in a descriptor set is specified by VkWriteDescriptorSet::`descriptorType`, which **must** be one of the values:

```
// Provided by VK_VERSION_1_0
typedef enum VkDescriptorType {
    VK_DESCRIPTOR_TYPE_SAMPLER = 0,
    VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER = 1,
    VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE = 2,
    VK_DESCRIPTOR_TYPE_STORAGE_IMAGE = 3,
    VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER = 4,
    VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER = 5,
    VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER = 6,
    VK_DESCRIPTOR_TYPE_STORAGE_BUFFER = 7,
    VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC = 8,
    VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC = 9,
    VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT = 10,
    VK_DESCRIPTOR_TYPE_INLINE_UNIFORM_BLOCK = 1000138000,
  // Provided by VK_KHR_acceleration_structure
    VK_DESCRIPTOR_TYPE_ACCELERATION_STRUCTURE_KHR = 1000150000,
  // Provided by VK_NV_ray_tracing
    VK_DESCRIPTOR_TYPE_ACCELERATION_STRUCTURE_NV = 1000165000,
  // Provided by VK_QCOM_image_processing
    VK_DESCRIPTOR_TYPE_SAMPLE_WEIGHT_IMAGE_QCOM = 1000440000,
  // Provided by VK_QCOM_image_processing
    VK_DESCRIPTOR_TYPE_BLOCK_MATCH_IMAGE_QCOM = 1000440001,
  // Provided by VK_EXT_mutable_descriptor_type
    VK_DESCRIPTOR_TYPE_MUTABLE_EXT = 1000351000,
  // Provided by VK_EXT_inline_uniform_block
    VK_DESCRIPTOR_TYPE_INLINE_UNIFORM_BLOCK_EXT =
VK_DESCRIPTOR_TYPE_INLINE_UNIFORM_BLOCK,
  // Provided by VK_VALVE_mutable_descriptor_type
    VK_DESCRIPTOR_TYPE_MUTABLE_VALVE = VK_DESCRIPTOR_TYPE_MUTABLE_EXT,
} VkDescriptorType;
```

- `VK_DESCRIPTOR_TYPE_SAMPLER` specifies a sampler descriptor.
- `VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER` specifies a combined image sampler descriptor.
- `VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE` specifies a sampled image descriptor.
- `VK_DESCRIPTOR_TYPE_STORAGE_IMAGE` specifies a storage image descriptor.
- `VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER` specifies a uniform texel buffer descriptor.
- `VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER` specifies a storage texel buffer descriptor.
- `VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER` specifies a uniform buffer descriptor.
- `VK_DESCRIPTOR_TYPE_STORAGE_BUFFER` specifies a storage buffer descriptor.
- `VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC` specifies a dynamic uniform buffer descriptor.
- `VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC` specifies a dynamic storage buffer descriptor.
- `VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT` specifies an input attachment descriptor.
- `VK_DESCRIPTOR_TYPE_INLINE_UNIFORM_BLOCK` specifies an inline uniform block.
- `VK_DESCRIPTOR_TYPE_MUTABLE_EXT` specifies a descriptor of mutable type.

- VK_DESCRIPTOR_TYPE_SAMPLE_WEIGHT_IMAGE_QCOM specifies a sampled weight image descriptor.
- VK_DESCRIPTOR_TYPE_BLOCK_MATCH_IMAGE_QCOM specifies a block matching image descriptor.

When a descriptor set is updated via elements of VkWriteDescriptorSet, members of pImageInfo, pBufferInfo and pTexelBufferView are only accessed by the implementation when they correspond to descriptor type being defined - otherwise they are ignored. The members accessed are as follows for each descriptor type:

- For VK_DESCRIPTOR_TYPE_SAMPLER, only the sampler member of each element of VkWriteDescriptorSet::pImageInfo is accessed.
- For VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE, VK_DESCRIPTOR_TYPE_STORAGE_IMAGE, or VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT, only the imageView and imageLayout members of each element of VkWriteDescriptorSet::pImageInfo are accessed.
- For VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER, all members of each element of VkWriteDescriptorSet::pImageInfo are accessed.
- For VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER, VK_DESCRIPTOR_TYPE_STORAGE_BUFFER, VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC, or VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC, all members of each element of VkWriteDescriptorSet::pBufferInfo are accessed.
- For VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER or VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER, each element of VkWriteDescriptorSet::pTexelBufferView is accessed.

When updating descriptors with a descriptorType of VK_DESCRIPTOR_TYPE_INLINE_UNIFORM_BLOCK, none of the pImageInfo, pBufferInfo, or pTexelBufferView members are accessed, instead the source data of the descriptor update operation is taken from the VkWriteDescriptorSetInlineUniformBlock structure in the pNext chain of VkWriteDescriptorSet. When updating descriptors with a descriptorType of VK_DESCRIPTOR_TYPE_ACCELERATION_STRUCTURE_KHR, none of the pImageInfo, pBufferInfo, or pTexelBufferView members are accessed, instead the source data of the descriptor update operation is taken from the VkWriteDescriptorSetAccelerationStructureKHR structure in the pNext chain of VkWriteDescriptorSet. When updating descriptors with a descriptorType of VK_DESCRIPTOR_TYPE_ACCELERATION_STRUCTURE_NV, none of the pImageInfo, pBufferInfo, or pTexelBufferView members are accessed, instead the source data of the descriptor update operation is taken from the VkWriteDescriptorSetAccelerationStructureNV structure in the pNext chain of VkWriteDescriptorSet.

The VkDescriptorBufferInfo structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkDescriptorBufferInfo {
    VkBuffer        buffer;
    VkDeviceSize    offset;
    VkDeviceSize    range;
} VkDescriptorBufferInfo;
```

- buffer is VK_NULL_HANDLE or the buffer resource.
- offset is the offset in bytes from the start of buffer. Access to buffer memory via this descriptor uses addressing that is relative to this starting offset.

- `range` is the size in bytes that is used for this descriptor update, or `VK_WHOLE_SIZE` to use the range from `offset` to the end of the buffer.

| | |
|---|---|
| **NOTE** | When setting `range` to `VK_WHOLE_SIZE`, the effective range **must** not be larger than the maximum range for the descriptor type (`maxUniformBufferRange` or `maxStorageBufferRange`). This means that `VK_WHOLE_SIZE` is not typically useful in the common case where uniform buffer descriptors are suballocated from a buffer that is much larger than `maxUniformBufferRange`. |

For `VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC` and `VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC` descriptor types, `offset` is the base offset from which the dynamic offset is applied and `range` is the static size used for all dynamic offsets.

When `range` is `VK_WHOLE_SIZE` the effective range is calculated at vkUpdateDescriptorSets is by taking the size of `buffer` minus the `offset`.

### Valid Usage

- VUID-VkDescriptorBufferInfo-offset-00340
  `offset` **must** be less than the size of `buffer`
- VUID-VkDescriptorBufferInfo-range-00341
  If `range` is not equal to `VK_WHOLE_SIZE`, `range` **must** be greater than `0`
- VUID-VkDescriptorBufferInfo-range-00342
  If `range` is not equal to `VK_WHOLE_SIZE`, `range` **must** be less than or equal to the size of `buffer` minus `offset`
- VUID-VkDescriptorBufferInfo-buffer-02998
  If the `nullDescriptor` feature is not enabled, `buffer` **must** not be VK_NULL_HANDLE
- VUID-VkDescriptorBufferInfo-buffer-02999
  If `buffer` is VK_NULL_HANDLE, `offset` **must** be zero and `range` **must** be `VK_WHOLE_SIZE`

### Valid Usage (Implicit)

- VUID-VkDescriptorBufferInfo-buffer-parameter
  If `buffer` is not VK_NULL_HANDLE, `buffer` **must** be a valid VkBuffer handle

The `VkDescriptorImageInfo` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkDescriptorImageInfo {
    VkSampler        sampler;
    VkImageView      imageView;
    VkImageLayout    imageLayout;
} VkDescriptorImageInfo;
```

- sampler is a sampler handle, and is used in descriptor updates for types VK_DESCRIPTOR_TYPE_SAMPLER and VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER if the binding being updated does not use immutable samplers.
- imageView is VK_NULL_HANDLE or an image view handle, and is used in descriptor updates for types VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE, VK_DESCRIPTOR_TYPE_STORAGE_IMAGE, VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER, and VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT.
- imageLayout is the layout that the image subresources accessible from imageView will be in at the time this descriptor is accessed. imageLayout is used in descriptor updates for types VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE, VK_DESCRIPTOR_TYPE_STORAGE_IMAGE, VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER, and VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT.

Members of VkDescriptorImageInfo that are not used in an update (as described above) are ignored.

### Valid Usage

- VUID-VkDescriptorImageInfo-imageView-06712
  imageView **must** not be a 2D array image view created from a 3D image
- VUID-VkDescriptorImageInfo-imageView-07795
  If imageView is a 2D view created from a 3D image, then descriptorType **must** be VK_DESCRIPTOR_TYPE_STORAGE_IMAGE, VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE, or VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER
- VUID-VkDescriptorImageInfo-imageView-07796
  If imageView is a 2D view created from a 3D image, then the image **must** have been created with VK_IMAGE_CREATE_2D_VIEW_COMPATIBLE_BIT_EXT set
- VUID-VkDescriptorImageInfo-descriptorType-06713
  If the image2DViewOf3D feature is not enabled or descriptorType is not VK_DESCRIPTOR_TYPE_STORAGE_IMAGE then imageView **must** not be a 2D view created from a 3D image
- VUID-VkDescriptorImageInfo-descriptorType-06714
  If the sampler2DViewOf3D feature is not enabled or descriptorType is not VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE or VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER then imageView **must** not be a 2D view created from a 3D image
- VUID-VkDescriptorImageInfo-imageView-01976
  If imageView is created from a depth/stencil image, the aspectMask used to create the imageView **must** include either VK_IMAGE_ASPECT_DEPTH_BIT or VK_IMAGE_ASPECT_STENCIL_BIT but not both
- VUID-VkDescriptorImageInfo-imageLayout-09425
  If imageLayout is VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL, then the aspectMask used to create imageView **must** not include either VK_IMAGE_ASPECT_DEPTH_BIT or VK_IMAGE_ASPECT_STENCIL_BIT
- VUID-VkDescriptorImageInfo-imageLayout-09426
  If imageLayout is VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_STENCIL_ATTACHMENT_OPTIMAL, VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_STENCIL_READ_ONLY_OPTIMAL, VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL, VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL,

VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL, VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL, VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL or VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL, then the aspectMask used to create imageView **must** not include VK_IMAGE_ASPECT_COLOR_BIT

- VUID-VkDescriptorImageInfo-imageLayout-00344
  imageLayout **must** match the actual VkImageLayout of each subresource accessible from imageView at the time this descriptor is accessed as defined by the image layout matching rules
- VUID-VkDescriptorImageInfo-sampler-01564
  If sampler is used and the VkFormat of the image is a multi-planar format, the image **must** have been created with VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT, and the aspectMask of the imageView **must** be a valid multi-planar aspect mask bit
- VUID-VkDescriptorImageInfo-mutableComparisonSamplers-04450
  If the VK_KHR_portability_subset extension is enabled, and VkPhysicalDevicePortabilitySubsetFeaturesKHR::mutableComparisonSamplers is VK_FALSE, then sampler **must** have been created with VkSamplerCreateInfo::compareEnable set to VK_FALSE

### Valid Usage (Implicit)

- VUID-VkDescriptorImageInfo-commonparent
  Both of imageView, and sampler that are valid handles of non-ignored parameters **must** have been created, allocated, or retrieved from the same VkDevice

If the descriptorType member of VkWriteDescriptorSet is VK_DESCRIPTOR_TYPE_INLINE_UNIFORM_BLOCK then the data to write to the descriptor set is specified through a VkWriteDescriptorSetInlineUniformBlock structure included in the pNext chain of VkWriteDescriptorSet.

The VkWriteDescriptorSetInlineUniformBlock structure is defined as:

```
typedef struct VkWriteDescriptorSetInlineUniformBlock {
    VkStructureType    sType;
    const void*        pNext;
    uint32_t           dataSize;
    const void*        pData;
} VkWriteDescriptorSetInlineUniformBlock;
```

or the equivalent

```
// Provided by VK_EXT_inline_uniform_block
typedef VkWriteDescriptorSetInlineUniformBlock
VkWriteDescriptorSetInlineUniformBlockEXT;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- dataSize is the number of bytes of inline uniform block data pointed to by pData.
- pData is a pointer to dataSize number of bytes of data to write to the inline uniform block.

### Valid Usage

- VUID-VkWriteDescriptorSetInlineUniformBlock-dataSize-02222
  dataSize **must** be an integer multiple of 4

### Valid Usage (Implicit)

- VUID-VkWriteDescriptorSetInlineUniformBlock-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_WRITE_DESCRIPTOR_SET_INLINE_UNIFORM_BLOCK
- VUID-VkWriteDescriptorSetInlineUniformBlock-pData-parameter
  pData **must** be a valid pointer to an array of dataSize bytes
- VUID-VkWriteDescriptorSetInlineUniformBlock-dataSize-arraylength
  dataSize **must** be greater than 0

The VkWriteDescriptorSetAccelerationStructureKHR structure is defined as:

```
// Provided by VK_KHR_acceleration_structure
typedef struct VkWriteDescriptorSetAccelerationStructureKHR {
    VkStructureType                     sType;
    const void*                         pNext;
    uint32_t                            accelerationStructureCount;
    const VkAccelerationStructureKHR*   pAccelerationStructures;
} VkWriteDescriptorSetAccelerationStructureKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- accelerationStructureCount is the number of elements in pAccelerationStructures.
- pAccelerationStructures is a pointer to an array of VkAccelerationStructureKHR structures specifying the acceleration structures to update.

### Valid Usage

- VUID-VkWriteDescriptorSetAccelerationStructureKHR-accelerationStructureCount-02236
  accelerationStructureCount **must** be equal to descriptorCount in the extended structure
- VUID-VkWriteDescriptorSetAccelerationStructureKHR-pAccelerationStructures-03579
  Each acceleration structure in pAccelerationStructures **must** have been created with a

type of VK_ACCELERATION_STRUCTURE_TYPE_TOP_LEVEL_KHR or VK_ACCELERATION_STRUCTURE_TYPE_GENERIC_KHR

- VUID-VkWriteDescriptorSetAccelerationStructureKHR-pAccelerationStructures-03580
  If the nullDescriptor feature is not enabled, each element of pAccelerationStructures **must** not be VK_NULL_HANDLE

### Valid Usage (Implicit)

- VUID-VkWriteDescriptorSetAccelerationStructureKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_WRITE_DESCRIPTOR_SET_ACCELERATION_STRUCTURE_KHR
- VUID-VkWriteDescriptorSetAccelerationStructureKHR-pAccelerationStructures-parameter
  pAccelerationStructures **must** be a valid pointer to an array of accelerationStructureCount valid or VK_NULL_HANDLE VkAccelerationStructureKHR handles
- VUID-VkWriteDescriptorSetAccelerationStructureKHR-accelerationStructureCount-arraylength
  accelerationStructureCount **must** be greater than 0

The VkWriteDescriptorSetAccelerationStructureNV structure is defined as:

```
// Provided by VK_NV_ray_tracing
typedef struct VkWriteDescriptorSetAccelerationStructureNV {
    VkStructureType                    sType;
    const void*                        pNext;
    uint32_t                           accelerationStructureCount;
    const VkAccelerationStructureNV*   pAccelerationStructures;
} VkWriteDescriptorSetAccelerationStructureNV;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- accelerationStructureCount is the number of elements in pAccelerationStructures.
- pAccelerationStructures is a pointer to an array of VkAccelerationStructureNV structures specifying the acceleration structures to update.

### Valid Usage

- VUID-VkWriteDescriptorSetAccelerationStructureNV-accelerationStructureCount-03747
  accelerationStructureCount **must** be equal to descriptorCount in the extended structure
- VUID-VkWriteDescriptorSetAccelerationStructureNV-pAccelerationStructures-03748
  Each acceleration structure in pAccelerationStructures **must** have been created with VK_ACCELERATION_STRUCTURE_TYPE_TOP_LEVEL_KHR
- VUID-VkWriteDescriptorSetAccelerationStructureNV-pAccelerationStructures-03749

If the nullDescriptor feature is not enabled, each member of pAccelerationStructures **must** not be VK_NULL_HANDLE

### Valid Usage (Implicit)

- VUID-VkWriteDescriptorSetAccelerationStructureNV-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_WRITE_DESCRIPTOR_SET_ACCELERATION_STRUCTURE_NV
- VUID-VkWriteDescriptorSetAccelerationStructureNV-pAccelerationStructures-parameter
  pAccelerationStructures **must** be a valid pointer to an array of accelerationStructureCount valid or VK_NULL_HANDLE VkAccelerationStructureNV handles
- VUID-VkWriteDescriptorSetAccelerationStructureNV-accelerationStructureCount-arraylength
  accelerationStructureCount **must** be greater than 0

The VkCopyDescriptorSet structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkCopyDescriptorSet {
    VkStructureType    sType;
    const void*        pNext;
    VkDescriptorSet    srcSet;
    uint32_t           srcBinding;
    uint32_t           srcArrayElement;
    VkDescriptorSet    dstSet;
    uint32_t           dstBinding;
    uint32_t           dstArrayElement;
    uint32_t           descriptorCount;
} VkCopyDescriptorSet;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- srcSet, srcBinding, and srcArrayElement are the source set, binding, and array element, respectively. If the descriptor binding identified by srcSet and srcBinding has a descriptor type of VK_DESCRIPTOR_TYPE_INLINE_UNIFORM_BLOCK then srcArrayElement specifies the starting byte offset within the binding to copy from.
- dstSet, dstBinding, and dstArrayElement are the destination set, binding, and array element, respectively. If the descriptor binding identified by dstSet and dstBinding has a descriptor type of VK_DESCRIPTOR_TYPE_INLINE_UNIFORM_BLOCK then dstArrayElement specifies the starting byte offset within the binding to copy to.
- descriptorCount is the number of descriptors to copy from the source to destination. If descriptorCount is greater than the number of remaining array elements in the source or destination binding, those affect consecutive bindings in a manner similar to VkWriteDescriptorSet above. If the descriptor binding identified by srcSet and srcBinding has a descriptor type of VK_DESCRIPTOR_TYPE_INLINE_UNIFORM_BLOCK then descriptorCount specifies the

number of bytes to copy and the remaining array elements in the source or destination binding refer to the remaining number of bytes in those.

If the `VkDescriptorSetLayoutBinding` for `dstBinding` is `VK_DESCRIPTOR_TYPE_MUTABLE_EXT` and `srcBinding` is not `VK_DESCRIPTOR_TYPE_MUTABLE_EXT`, the new active descriptor type becomes the descriptor type of `srcBinding`. If both `VkDescriptorSetLayoutBinding` for `srcBinding` and `dstBinding` are `VK_DESCRIPTOR_TYPE_MUTABLE_EXT`, the active descriptor type in each source descriptor is copied into the corresponding destination descriptor. The active descriptor type **can** be different for each source descriptor.

**NOTE** The intention is that copies to and from mutable descriptors is a simple memcpy. Copies between non-mutable and mutable descriptors are expected to require one memcpy per descriptor to handle the difference in size, but this use case with more than one `descriptorCount` is considered rare.

## Valid Usage

- VUID-VkCopyDescriptorSet-srcBinding-00345
  `srcBinding` **must** be a valid binding within `srcSet`
- VUID-VkCopyDescriptorSet-srcArrayElement-00346
  The sum of `srcArrayElement` and `descriptorCount` **must** be less than or equal to the number of array elements in the descriptor set binding specified by `srcBinding`, and all applicable consecutive bindings, as described by consecutive binding updates
- VUID-VkCopyDescriptorSet-dstBinding-00347
  `dstBinding` **must** be a valid binding within `dstSet`
- VUID-VkCopyDescriptorSet-dstArrayElement-00348
  The sum of `dstArrayElement` and `descriptorCount` **must** be less than or equal to the number of array elements in the descriptor set binding specified by `dstBinding`, and all applicable consecutive bindings, as described by consecutive binding updates
- VUID-VkCopyDescriptorSet-dstBinding-02632
  The type of `dstBinding` within `dstSet` **must** be equal to the type of `srcBinding` within `srcSet`
- VUID-VkCopyDescriptorSet-srcSet-00349
  If `srcSet` is equal to `dstSet`, then the source and destination ranges of descriptors **must** not overlap, where the ranges **may** include array elements from consecutive bindings as described by consecutive binding updates
- VUID-VkCopyDescriptorSet-srcBinding-02223
  If the descriptor type of the descriptor set binding specified by `srcBinding` is `VK_DESCRIPTOR_TYPE_INLINE_UNIFORM_BLOCK`, `srcArrayElement` **must** be an integer multiple of `4`
- VUID-VkCopyDescriptorSet-dstBinding-02224
  If the descriptor type of the descriptor set binding specified by `dstBinding` is `VK_DESCRIPTOR_TYPE_INLINE_UNIFORM_BLOCK`, `dstArrayElement` **must** be an integer multiple of `4`
- VUID-VkCopyDescriptorSet-srcBinding-02225

If the descriptor type of the descriptor set binding specified by either srcBinding or dstBinding is VK_DESCRIPTOR_TYPE_INLINE_UNIFORM_BLOCK, descriptorCount **must** be an integer multiple of 4

- VUID-VkCopyDescriptorSet-srcSet-01918
  If srcSet's layout was created with the VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT flag set, then dstSet's layout **must** also have been created with the VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT flag set
- VUID-VkCopyDescriptorSet-srcSet-04885
  If srcSet's layout was created without either the VK_DESCRIPTOR_SET_LAYOUT_CREATE_HOST_ONLY_POOL_BIT_EXT flag or the VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT flag set, then dstSet's layout **must** have been created without the VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT flag set
- VUID-VkCopyDescriptorSet-srcSet-01920
  If the descriptor pool from which srcSet was allocated was created with the VK_DESCRIPTOR_POOL_CREATE_UPDATE_AFTER_BIND_BIT flag set, then the descriptor pool from which dstSet was allocated **must** also have been created with the VK_DESCRIPTOR_POOL_CREATE_UPDATE_AFTER_BIND_BIT flag set
- VUID-VkCopyDescriptorSet-srcSet-04887
  If the descriptor pool from which srcSet was allocated was created without either the VK_DESCRIPTOR_POOL_CREATE_HOST_ONLY_BIT_EXT flag or the VK_DESCRIPTOR_POOL_CREATE_UPDATE_AFTER_BIND_BIT flag set, then the descriptor pool from which dstSet was allocated **must** have been created without the VK_DESCRIPTOR_POOL_CREATE_UPDATE_AFTER_BIND_BIT flag set
- VUID-VkCopyDescriptorSet-dstBinding-02753
  If the descriptor type of the descriptor set binding specified by dstBinding is VK_DESCRIPTOR_TYPE_SAMPLER, then dstSet **must** not have been allocated with a layout that included immutable samplers for dstBinding
- VUID-VkCopyDescriptorSet-dstSet-04612
  If VkDescriptorSetLayoutBinding for dstSet at dstBinding is VK_DESCRIPTOR_TYPE_MUTABLE_EXT, the new active descriptor type **must** exist in the corresponding pMutableDescriptorTypeLists list for dstBinding if the new active descriptor type is not VK_DESCRIPTOR_TYPE_MUTABLE_EXT
- VUID-VkCopyDescriptorSet-srcSet-04613
  If VkDescriptorSetLayoutBinding for srcSet at srcBinding is VK_DESCRIPTOR_TYPE_MUTABLE_EXT and the VkDescriptorSetLayoutBinding for dstSet at dstBinding is not VK_DESCRIPTOR_TYPE_MUTABLE_EXT, the active descriptor type for the source descriptor **must** match the descriptor type of dstBinding
- VUID-VkCopyDescriptorSet-dstSet-04614
  If VkDescriptorSetLayoutBinding for dstSet at dstBinding is VK_DESCRIPTOR_TYPE_MUTABLE_EXT, and the new active descriptor type is VK_DESCRIPTOR_TYPE_MUTABLE_EXT, the pMutableDescriptorTypeLists for srcBinding and dstBinding **must** match exactly

### Valid Usage (Implicit)

- VUID-VkCopyDescriptorSet-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_COPY_DESCRIPTOR_SET`
- VUID-VkCopyDescriptorSet-pNext-pNext
  `pNext` **must** be `NULL`
- VUID-VkCopyDescriptorSet-srcSet-parameter
  `srcSet` **must** be a valid VkDescriptorSet handle
- VUID-VkCopyDescriptorSet-dstSet-parameter
  `dstSet` **must** be a valid VkDescriptorSet handle
- VUID-VkCopyDescriptorSet-commonparent
  Both of `dstSet`, and `srcSet` **must** have been created, allocated, or retrieved from the same VkDevice

## 14.2.5. Descriptor Update Templates

A descriptor update template specifies a mapping from descriptor update information in host memory to descriptors in a descriptor set. It is designed to avoid passing redundant information to the driver when frequently updating the same set of descriptors in descriptor sets.

Descriptor update template objects are represented by `VkDescriptorUpdateTemplate` handles:

```
// Provided by VK_VERSION_1_1
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkDescriptorUpdateTemplate)
```

or the equivalent

```
// Provided by VK_KHR_descriptor_update_template
typedef VkDescriptorUpdateTemplate VkDescriptorUpdateTemplateKHR;
```

## 14.2.6. Descriptor Set Updates With Templates

Updating a large `VkDescriptorSet` array **can** be an expensive operation since an application **must** specify one VkWriteDescriptorSet structure for each descriptor or descriptor array to update, each of which re-specifies the same state when updating the same descriptor in multiple descriptor sets. For cases when an application wishes to update the same set of descriptors in multiple descriptor sets allocated using the same `VkDescriptorSetLayout`, vkUpdateDescriptorSetWithTemplate **can** be used as a replacement for vkUpdateDescriptorSets.

`VkDescriptorUpdateTemplate` allows implementations to convert a set of descriptor update operations on a single descriptor set to an internal format that, in conjunction with vkUpdateDescriptorSetWithTemplate or vkCmdPushDescriptorSetWithTemplateKHR , **can** be more efficient compared to calling vkUpdateDescriptorSets or vkCmdPushDescriptorSetKHR . The descriptors themselves are not specified in the `VkDescriptorUpdateTemplate`, rather, offsets into an

application provided pointer to host memory are specified, which are combined with a pointer passed to vkUpdateDescriptorSetWithTemplate or vkCmdPushDescriptorSetWithTemplateKHR . This allows large batches of updates to be executed without having to convert application data structures into a strictly-defined Vulkan data structure.

To create a descriptor update template, call:

```
// Provided by VK_VERSION_1_1
VkResult vkCreateDescriptorUpdateTemplate(
    VkDevice                                    device,
    const VkDescriptorUpdateTemplateCreateInfo* pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkDescriptorUpdateTemplate*                 pDescriptorUpdateTemplate);
```

or the equivalent command

```
// Provided by VK_KHR_descriptor_update_template
VkResult vkCreateDescriptorUpdateTemplateKHR(
    VkDevice                                    device,
    const VkDescriptorUpdateTemplateCreateInfo* pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkDescriptorUpdateTemplate*                 pDescriptorUpdateTemplate);
```

- `device` is the logical device that creates the descriptor update template.
- `pCreateInfo` is a pointer to a VkDescriptorUpdateTemplateCreateInfo structure specifying the set of descriptors to update with a single call to vkCmdPushDescriptorSetWithTemplateKHR or vkUpdateDescriptorSetWithTemplate.
- `pAllocator` controls host memory allocation as described in the Memory Allocation chapter.
- `pDescriptorUpdateTemplate` is a pointer to a `VkDescriptorUpdateTemplate` handle in which the resulting descriptor update template object is returned.

### Valid Usage (Implicit)

- VUID-vkCreateDescriptorUpdateTemplate-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkCreateDescriptorUpdateTemplate-pCreateInfo-parameter
  `pCreateInfo` **must** be a valid pointer to a valid VkDescriptorUpdateTemplateCreateInfo structure
- VUID-vkCreateDescriptorUpdateTemplate-pAllocator-parameter
  If `pAllocator` is not `NULL`, `pAllocator` **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkCreateDescriptorUpdateTemplate-pDescriptorUpdateTemplate-parameter
  `pDescriptorUpdateTemplate` **must** be a valid pointer to a VkDescriptorUpdateTemplate handle

### Return Codes

**Success**

- VK_SUCCESS

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY

The VkDescriptorUpdateTemplateCreateInfo structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkDescriptorUpdateTemplateCreateInfo {
    VkStructureType                             sType;
    const void*                                 pNext;
    VkDescriptorUpdateTemplateCreateFlags       flags;
    uint32_t                                    descriptorUpdateEntryCount;
    const VkDescriptorUpdateTemplateEntry*      pDescriptorUpdateEntries;
    VkDescriptorUpdateTemplateType              templateType;
    VkDescriptorSetLayout                       descriptorSetLayout;
    VkPipelineBindPoint                         pipelineBindPoint;
    VkPipelineLayout                            pipelineLayout;
    uint32_t                                    set;
} VkDescriptorUpdateTemplateCreateInfo;
```

or the equivalent

```
// Provided by VK_KHR_descriptor_update_template
typedef VkDescriptorUpdateTemplateCreateInfo VkDescriptorUpdateTemplateCreateInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- flags is reserved for future use.
- descriptorUpdateEntryCount is the number of elements in the pDescriptorUpdateEntries array.
- pDescriptorUpdateEntries is a pointer to an array of VkDescriptorUpdateTemplateEntry structures describing the descriptors to be updated by the descriptor update template.
- templateType Specifies the type of the descriptor update template. If set to VK_DESCRIPTOR_UPDATE_TEMPLATE_TYPE_DESCRIPTOR_SET it **can** only be used to update descriptor sets with a fixed descriptorSetLayout. If set to VK_DESCRIPTOR_UPDATE_TEMPLATE_TYPE_PUSH_DESCRIPTORS_KHR it **can** only be used to push descriptor sets using the provided pipelineBindPoint, pipelineLayout, and set number.
- descriptorSetLayout is the descriptor set layout used to build the descriptor update template. All descriptor sets which are going to be updated through the newly created descriptor update

template **must** be created with a layout that matches (is the same as, or defined identically to) this layout. This parameter is ignored if `templateType` is not `VK_DESCRIPTOR_UPDATE_TEMPLATE_TYPE_DESCRIPTOR_SET`.

- `pipelineBindPoint` is a VkPipelineBindPoint indicating the type of the pipeline that will use the descriptors. This parameter is ignored if `templateType` is not `VK_DESCRIPTOR_UPDATE_TEMPLATE_TYPE_PUSH_DESCRIPTORS_KHR`
- `pipelineLayout` is a VkPipelineLayout object used to program the bindings. This parameter is ignored if `templateType` is not `VK_DESCRIPTOR_UPDATE_TEMPLATE_TYPE_PUSH_DESCRIPTORS_KHR`
- `set` is the set number of the descriptor set in the pipeline layout that will be updated. This parameter is ignored if `templateType` is not `VK_DESCRIPTOR_UPDATE_TEMPLATE_TYPE_PUSH_DESCRIPTORS_KHR`

## Valid Usage

- VUID-VkDescriptorUpdateTemplateCreateInfo-templateType-00350
  If `templateType` is `VK_DESCRIPTOR_UPDATE_TEMPLATE_TYPE_DESCRIPTOR_SET`, `descriptorSetLayout` **must** be a valid `VkDescriptorSetLayout` handle
- VUID-VkDescriptorUpdateTemplateCreateInfo-templateType-00351
  If `templateType` is `VK_DESCRIPTOR_UPDATE_TEMPLATE_TYPE_PUSH_DESCRIPTORS_KHR`, `pipelineBindPoint` **must** be a valid VkPipelineBindPoint value
- VUID-VkDescriptorUpdateTemplateCreateInfo-templateType-00352
  If `templateType` is `VK_DESCRIPTOR_UPDATE_TEMPLATE_TYPE_PUSH_DESCRIPTORS_KHR`, `pipelineLayout` **must** be a valid `VkPipelineLayout` handle
- VUID-VkDescriptorUpdateTemplateCreateInfo-templateType-00353
  If `templateType` is `VK_DESCRIPTOR_UPDATE_TEMPLATE_TYPE_PUSH_DESCRIPTORS_KHR`, `set` **must** be the unique set number in the pipeline layout that uses a descriptor set layout that was created with `VK_DESCRIPTOR_SET_LAYOUT_CREATE_PUSH_DESCRIPTOR_BIT_KHR`
- VUID-VkDescriptorUpdateTemplateCreateInfo-templateType-04615
  If `templateType` is `VK_DESCRIPTOR_UPDATE_TEMPLATE_TYPE_DESCRIPTOR_SET`, `descriptorSetLayout` **must** not contain a binding with type `VK_DESCRIPTOR_TYPE_MUTABLE_EXT`

## Valid Usage (Implicit)

- VUID-VkDescriptorUpdateTemplateCreateInfo-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_DESCRIPTOR_UPDATE_TEMPLATE_CREATE_INFO`
- VUID-VkDescriptorUpdateTemplateCreateInfo-pNext-pNext
  `pNext` **must** be `NULL`
- VUID-VkDescriptorUpdateTemplateCreateInfo-flags-zerobitmask
  `flags` **must** be `0`
- VUID-VkDescriptorUpdateTemplateCreateInfo-pDescriptorUpdateEntries-parameter
  `pDescriptorUpdateEntries` **must** be a valid pointer to an array of `descriptorUpdateEntryCount` valid VkDescriptorUpdateTemplateEntry structures

- VUID-VkDescriptorUpdateTemplateCreateInfo-templateType-parameter
  `templateType` **must** be a valid VkDescriptorUpdateTemplateType value
- VUID-VkDescriptorUpdateTemplateCreateInfo-descriptorUpdateEntryCount-arraylength
  `descriptorUpdateEntryCount` **must** be greater than `0`
- VUID-VkDescriptorUpdateTemplateCreateInfo-commonparent
  Both of `descriptorSetLayout`, and `pipelineLayout` that are valid handles of non-ignored parameters **must** have been created, allocated, or retrieved from the same VkDevice

```
// Provided by VK_VERSION_1_1
typedef VkFlags VkDescriptorUpdateTemplateCreateFlags;
```

or the equivalent

```
// Provided by VK_KHR_descriptor_update_template
typedef VkDescriptorUpdateTemplateCreateFlags
VkDescriptorUpdateTemplateCreateFlagsKHR;
```

`VkDescriptorUpdateTemplateCreateFlags` is a bitmask type for setting a mask, but is currently reserved for future use.

The descriptor update template type is determined by the VkDescriptorUpdateTemplateCreateInfo ::`templateType` property, which takes the following values:

```
// Provided by VK_VERSION_1_1
typedef enum VkDescriptorUpdateTemplateType {
    VK_DESCRIPTOR_UPDATE_TEMPLATE_TYPE_DESCRIPTOR_SET = 0,
  // Provided by VK_KHR_descriptor_update_template with VK_KHR_push_descriptor,
VK_KHR_push_descriptor with VK_VERSION_1_1 or VK_KHR_descriptor_update_template
    VK_DESCRIPTOR_UPDATE_TEMPLATE_TYPE_PUSH_DESCRIPTORS_KHR = 1,
  // Provided by VK_KHR_descriptor_update_template
    VK_DESCRIPTOR_UPDATE_TEMPLATE_TYPE_DESCRIPTOR_SET_KHR =
VK_DESCRIPTOR_UPDATE_TEMPLATE_TYPE_DESCRIPTOR_SET,
} VkDescriptorUpdateTemplateType;
```

or the equivalent

```
// Provided by VK_KHR_descriptor_update_template
typedef VkDescriptorUpdateTemplateType VkDescriptorUpdateTemplateTypeKHR;
```

- `VK_DESCRIPTOR_UPDATE_TEMPLATE_TYPE_DESCRIPTOR_SET` specifies that the descriptor update template will be used for descriptor set updates only.
- `VK_DESCRIPTOR_UPDATE_TEMPLATE_TYPE_PUSH_DESCRIPTORS_KHR` specifies that the descriptor update template will be used for push descriptor updates only.

The VkDescriptorUpdateTemplateEntry structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkDescriptorUpdateTemplateEntry {
    uint32_t            dstBinding;
    uint32_t            dstArrayElement;
    uint32_t            descriptorCount;
    VkDescriptorType    descriptorType;
    size_t              offset;
    size_t              stride;
} VkDescriptorUpdateTemplateEntry;
```

or the equivalent

```
// Provided by VK_KHR_descriptor_update_template
typedef VkDescriptorUpdateTemplateEntry VkDescriptorUpdateTemplateEntryKHR;
```

- dstBinding is the descriptor binding to update when using this descriptor update template.
- dstArrayElement is the starting element in the array belonging to dstBinding. If the descriptor binding identified by dstBinding has a descriptor type of VK_DESCRIPTOR_TYPE_INLINE_UNIFORM_BLOCK then dstArrayElement specifies the starting byte offset to update.
- descriptorCount is the number of descriptors to update. If descriptorCount is greater than the number of remaining array elements in the destination binding, those affect consecutive bindings in a manner similar to VkWriteDescriptorSet above. If the descriptor binding identified by dstBinding has a descriptor type of VK_DESCRIPTOR_TYPE_INLINE_UNIFORM_BLOCK then descriptorCount specifies the number of bytes to update and the remaining array elements in the destination binding refer to the remaining number of bytes in it.
- descriptorType is a VkDescriptorType specifying the type of the descriptor.
- offset is the offset in bytes of the first binding in the raw data structure.
- stride is the stride in bytes between two consecutive array elements of the descriptor update information in the raw data structure. The actual pointer ptr for each array element j of update entry i is computed using the following formula:

  ```
  const char *ptr = (const char *)pData + pDescriptorUpdateEntries[i].offset + j
  * pDescriptorUpdateEntries[i].stride
  ```

  The stride is useful in case the bindings are stored in structs along with other data. If descriptorType is VK_DESCRIPTOR_TYPE_INLINE_UNIFORM_BLOCK then the value of stride is ignored and the stride is assumed to be 1, i.e. the descriptor update information for them is always specified as a contiguous range.

### Valid Usage

- VUID-VkDescriptorUpdateTemplateEntry-dstBinding-00354
  `dstBinding` **must** be a valid binding in the descriptor set layout implicitly specified when using a descriptor update template to update descriptors
- VUID-VkDescriptorUpdateTemplateEntry-dstArrayElement-00355
  `dstArrayElement` and `descriptorCount` **must** be less than or equal to the number of array elements in the descriptor set binding implicitly specified when using a descriptor update template to update descriptors, and all applicable consecutive bindings, as described by consecutive binding updates
- VUID-VkDescriptorUpdateTemplateEntry-descriptor-02226
  If `descriptor` type is `VK_DESCRIPTOR_TYPE_INLINE_UNIFORM_BLOCK`, `dstArrayElement` **must** be an integer multiple of `4`
- VUID-VkDescriptorUpdateTemplateEntry-descriptor-02227
  If `descriptor` type is `VK_DESCRIPTOR_TYPE_INLINE_UNIFORM_BLOCK`, `descriptorCount` **must** be an integer multiple of `4`

### Valid Usage (Implicit)

- VUID-VkDescriptorUpdateTemplateEntry-descriptorType-parameter
  `descriptorType` **must** be a valid VkDescriptorType value

To destroy a descriptor update template, call:

```
// Provided by VK_VERSION_1_1
void vkDestroyDescriptorUpdateTemplate(
    VkDevice                                    device,
    VkDescriptorUpdateTemplate                  descriptorUpdateTemplate,
    const VkAllocationCallbacks*                pAllocator);
```

or the equivalent command

```
// Provided by VK_KHR_descriptor_update_template
void vkDestroyDescriptorUpdateTemplateKHR(
    VkDevice                                    device,
    VkDescriptorUpdateTemplate                  descriptorUpdateTemplate,
    const VkAllocationCallbacks*                pAllocator);
```

- `device` is the logical device that has been used to create the descriptor update template
- `descriptorUpdateTemplate` is the descriptor update template to destroy.
- `pAllocator` controls host memory allocation as described in the Memory Allocation chapter.

### Valid Usage

- VUID-vkDestroyDescriptorUpdateTemplate-descriptorSetLayout-00356
  If `VkAllocationCallbacks` were provided when `descriptorUpdateTemplate` was created, a compatible set of callbacks **must** be provided here
- VUID-vkDestroyDescriptorUpdateTemplate-descriptorSetLayout-00357
  If no `VkAllocationCallbacks` were provided when `descriptorUpdateTemplate` was created, `pAllocator` **must** be `NULL`

### Valid Usage (Implicit)

- VUID-vkDestroyDescriptorUpdateTemplate-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkDestroyDescriptorUpdateTemplate-descriptorUpdateTemplate-parameter
  If `descriptorUpdateTemplate` is not VK_NULL_HANDLE, `descriptorUpdateTemplate` **must** be a valid VkDescriptorUpdateTemplate handle
- VUID-vkDestroyDescriptorUpdateTemplate-pAllocator-parameter
  If `pAllocator` is not `NULL`, `pAllocator` **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkDestroyDescriptorUpdateTemplate-descriptorUpdateTemplate-parent
  If `descriptorUpdateTemplate` is a valid handle, it **must** have been created, allocated, or retrieved from `device`

### Host Synchronization

- Host access to `descriptorUpdateTemplate` **must** be externally synchronized

Once a `VkDescriptorUpdateTemplate` has been created, descriptor sets **can** be updated by calling:

```
// Provided by VK_VERSION_1_1
void vkUpdateDescriptorSetWithTemplate(
    VkDevice                                    device,
    VkDescriptorSet                             descriptorSet,
    VkDescriptorUpdateTemplate                  descriptorUpdateTemplate,
    const void*                                 pData);
```

or the equivalent command

```
// Provided by VK_KHR_descriptor_update_template
void vkUpdateDescriptorSetWithTemplateKHR(
    VkDevice                                    device,
    VkDescriptorSet                             descriptorSet,
    VkDescriptorUpdateTemplate                  descriptorUpdateTemplate,
    const void*                                 pData);
```

- `device` is the logical device that updates the descriptor set.
- `descriptorSet` is the descriptor set to update
- `descriptorUpdateTemplate` is a VkDescriptorUpdateTemplate object specifying the update mapping between `pData` and the descriptor set to update.
- `pData` is a pointer to memory containing one or more VkDescriptorImageInfo, VkDescriptorBufferInfo, or VkBufferView structures or VkAccelerationStructureKHR or VkAccelerationStructureNV handles used to write the descriptors.

## Valid Usage

- VUID-vkUpdateDescriptorSetWithTemplate-pData-01685
  `pData` **must** be a valid pointer to a memory containing one or more valid instances of VkDescriptorImageInfo, VkDescriptorBufferInfo, or VkBufferView in a layout defined by `descriptorUpdateTemplate` when it was created with vkCreateDescriptorUpdateTemplate
- VUID-vkUpdateDescriptorSetWithTemplate-descriptorSet-06995
  Host access to `descriptorSet` **must** be externally synchronized unless explicitly denoted otherwise for specific flags

## Valid Usage (Implicit)

- VUID-vkUpdateDescriptorSetWithTemplate-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkUpdateDescriptorSetWithTemplate-descriptorSet-parameter
  `descriptorSet` **must** be a valid VkDescriptorSet handle
- VUID-vkUpdateDescriptorSetWithTemplate-descriptorUpdateTemplate-parameter
  `descriptorUpdateTemplate` **must** be a valid VkDescriptorUpdateTemplate handle
- VUID-vkUpdateDescriptorSetWithTemplate-descriptorSet-parent
  `descriptorSet` **must** have been created, allocated, or retrieved from `device`
- VUID-vkUpdateDescriptorSetWithTemplate-descriptorUpdateTemplate-parent
  `descriptorUpdateTemplate` **must** have been created, allocated, or retrieved from `device`

*API example*

```
struct AppBufferView {
    VkBufferView bufferView;
```

```
    uint32_t     applicationRelatedInformation;
};

struct AppDataStructure
{
    VkDescriptorImageInfo  imageInfo;          // a single image info
    VkDescriptorBufferInfo bufferInfoArray[3]; // 3 buffer infos in an array
    AppBufferView          bufferView[2];      // An application-defined structure
containing a bufferView
    // ... some more application-related data
};

const VkDescriptorUpdateTemplateEntry descriptorUpdateTemplateEntries[] =
{
    // binding to a single image descriptor
    {
        .binding = 0,
        .dstArrayElement = 0,
        .descriptorCount = 1,
        .descriptorType = VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER,
        .offset = offsetof(AppDataStructure, imageInfo),
        .stride = 0         // stride not required if descriptorCount is 1
    },

    // binding to an array of buffer descriptors
    {
        .binding = 1,
        .dstArrayElement = 0,
        .descriptorCount = 3,
        .descriptorType = VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER,
        .offset = offsetof(AppDataStructure, bufferInfoArray),
        .stride = sizeof(VkDescriptorBufferInfo)    // descriptor buffer infos are
compact
    },

    // binding to an array of buffer views
    {
        .binding = 2,
        .dstArrayElement = 0,
        .descriptorCount = 2,
        .descriptorType = VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER,
        .offset = offsetof(AppDataStructure, bufferView) +
                  offsetof(AppBufferView, bufferView),
        .stride = sizeof(AppBufferView)             // bufferViews do not have to be
compact
    },
};

// create a descriptor update template for descriptor set updates
const VkDescriptorUpdateTemplateCreateInfo createInfo =
{
```

```
    .sType = VK_STRUCTURE_TYPE_DESCRIPTOR_UPDATE_TEMPLATE_CREATE_INFO,
    .pNext = NULL,
    .flags = 0,
    .descriptorUpdateEntryCount = 3,
    .pDescriptorUpdateEntries = descriptorUpdateTemplateEntries,
    .templateType = VK_DESCRIPTOR_UPDATE_TEMPLATE_TYPE_DESCRIPTOR_SET,
    .descriptorSetLayout = myLayout,
    .pipelineBindPoint = 0,     // ignored by given templateType
    .pipelineLayout = 0,        // ignored by given templateType
    .set = 0,                   // ignored by given templateType
};

VkDescriptorUpdateTemplate myDescriptorUpdateTemplate;
myResult = vkCreateDescriptorUpdateTemplate(
    myDevice,
    &createInfo,
    NULL,
    &myDescriptorUpdateTemplate);

AppDataStructure appData;

// fill appData here or cache it in your engine
vkUpdateDescriptorSetWithTemplate(myDevice, myDescriptorSet,
myDescriptorUpdateTemplate, &appData);
```

### 14.2.7. Descriptor Set Binding

To bind one or more descriptor sets to a command buffer, call:

```
// Provided by VK_VERSION_1_0
void vkCmdBindDescriptorSets(
    VkCommandBuffer                             commandBuffer,
    VkPipelineBindPoint                         pipelineBindPoint,
    VkPipelineLayout                            layout,
    uint32_t                                    firstSet,
    uint32_t                                    descriptorSetCount,
    const VkDescriptorSet*                      pDescriptorSets,
    uint32_t                                    dynamicOffsetCount,
    const uint32_t*                             pDynamicOffsets);
```

- `commandBuffer` is the command buffer that the descriptor sets will be bound to.
- `pipelineBindPoint` is a VkPipelineBindPoint indicating the type of the pipeline that will use the descriptors. There is a separate set of bind points for each pipeline type, so binding one does not disturb the others.
- `layout` is a VkPipelineLayout object used to program the bindings.
- `firstSet` is the set number of the first descriptor set to be bound.
- `descriptorSetCount` is the number of elements in the `pDescriptorSets` array.

- pDescriptorSets is a pointer to an array of handles to VkDescriptorSet objects describing the descriptor sets to bind to.
- dynamicOffsetCount is the number of dynamic offsets in the pDynamicOffsets array.
- pDynamicOffsets is a pointer to an array of uint32_t values specifying dynamic offsets.

vkCmdBindDescriptorSets binds descriptor sets pDescriptorSets[0..descriptorSetCount-1] to set numbers [firstSet..firstSet+descriptorSetCount-1] for subsequent bound pipeline commands set by pipelineBindPoint. Any bindings that were previously applied via these sets , or calls to vkCmdSetDescriptorBufferOffsetsEXT or vkCmdBindDescriptorBufferEmbeddedSamplersEXT, are no longer valid.

Once bound, a descriptor set affects rendering of subsequent commands that interact with the given pipeline type in the command buffer until either a different set is bound to the same set number, or the set is disturbed as described in Pipeline Layout Compatibility.

A compatible descriptor set **must** be bound for all set numbers that any shaders in a pipeline access, at the time that a drawing or dispatching command is recorded to execute using that pipeline. However, if none of the shaders in a pipeline statically use any bindings with a particular set number, then no descriptor set need be bound for that set number, even if the pipeline layout includes a non-trivial descriptor set layout for that set number.

When consuming a descriptor, a descriptor is considered valid if the descriptor is not undefined as described by descriptor set allocation. If the nullDescriptor feature is enabled, a null descriptor is also considered valid. A descriptor that was disturbed by Pipeline Layout Compatibility, or was never bound by vkCmdBindDescriptorSets is not considered valid. If a pipeline accesses a descriptor either statically or dynamically depending on the VkDescriptorBindingFlagBits, the consuming descriptor type in the pipeline **must** match the VkDescriptorType in VkDescriptorSetLayoutCreateInfo for the descriptor to be considered valid. If a descriptor is a mutable descriptor, the consuming descriptor type in the pipeline **must** match the active descriptor type for the descriptor to be considered valid.

| | |
|---|---|
| **NOTE** | Further validation may be carried out beyond validation for descriptor types, e.g. Texel Input Validation. |

If any of the sets being bound include dynamic uniform or storage buffers, then pDynamicOffsets includes one element for each array element in each dynamic descriptor type binding in each set. Values are taken from pDynamicOffsets in an order such that all entries for set N come before set N+1; within a set, entries are ordered by the binding numbers in the descriptor set layouts; and within a binding array, elements are in order. dynamicOffsetCount **must** equal the total number of dynamic descriptors in the sets being bound.

The effective offset used for dynamic uniform and storage buffer bindings is the sum of the relative offset taken from pDynamicOffsets, and the base address of the buffer plus base offset in the descriptor set. The range of the dynamic uniform and storage buffer bindings is the buffer range as specified in the descriptor set.

Each of the pDescriptorSets **must** be compatible with the pipeline layout specified by layout. The layout used to program the bindings **must** also be compatible with the pipeline used in subsequent

bound pipeline commands with that pipeline type, as defined in the Pipeline Layout Compatibility section.

The descriptor set contents bound by a call to vkCmdBindDescriptorSets **may** be consumed at the following times:

- For descriptor bindings created with the VK_DESCRIPTOR_BINDING_UPDATE_AFTER_BIND_BIT bit set, the contents **may** be consumed when the command buffer is submitted to a queue, or during shader execution of the resulting draws and dispatches, or any time in between. Otherwise,
- during host execution of the command, or during shader execution of the resulting draws and dispatches, or any time in between.

Thus, the contents of a descriptor set binding **must** not be altered (overwritten by an update command, or freed) between the first point in time that it **may** be consumed, and when the command completes executing on the queue.

The contents of pDynamicOffsets are consumed immediately during execution of vkCmdBindDescriptorSets. Once all pending uses have completed, it is legal to update and reuse a descriptor set.

### Valid Usage

- VUID-vkCmdBindDescriptorSets-pDescriptorSets-00358
  Each element of pDescriptorSets **must** have been allocated with a VkDescriptorSetLayout that matches (is the same as, or identically defined as) the VkDescriptorSetLayout at set *n* in layout, where *n* is the sum of firstSet and the index into pDescriptorSets
- VUID-vkCmdBindDescriptorSets-dynamicOffsetCount-00359
  dynamicOffsetCount **must** be equal to the total number of dynamic descriptors in pDescriptorSets
- VUID-vkCmdBindDescriptorSets-firstSet-00360
  The sum of firstSet and descriptorSetCount **must** be less than or equal to VkPipelineLayoutCreateInfo::setLayoutCount provided when layout was created
- VUID-vkCmdBindDescriptorSets-pDynamicOffsets-01971
  Each element of pDynamicOffsets which corresponds to a descriptor binding with type VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC **must** be a multiple of VkPhysicalDeviceLimits::minUniformBufferOffsetAlignment
- VUID-vkCmdBindDescriptorSets-pDynamicOffsets-01972
  Each element of pDynamicOffsets which corresponds to a descriptor binding with type VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC **must** be a multiple of VkPhysicalDeviceLimits::minStorageBufferOffsetAlignment
- VUID-vkCmdBindDescriptorSets-pDescriptorSets-01979
  For each dynamic uniform or storage buffer binding in pDescriptorSets, the sum of the effective offset and the range of the binding **must** be less than or equal to the size of the buffer
- VUID-vkCmdBindDescriptorSets-pDescriptorSets-06715
  For each dynamic uniform or storage buffer binding in pDescriptorSets, if the range was

set with VK_WHOLE_SIZE then pDynamicOffsets which corresponds to the descriptor binding **must** be 0

- VUID-vkCmdBindDescriptorSets-pDescriptorSets-04616
  Each element of pDescriptorSets **must** not have been allocated from a VkDescriptorPool with the VK_DESCRIPTOR_POOL_CREATE_HOST_ONLY_BIT_EXT flag set
- VUID-vkCmdBindDescriptorSets-pDescriptorSets-06563
  If graphicsPipelineLibrary is not enabled, each element of pDescriptorSets **must** be a valid VkDescriptorSet
- VUID-vkCmdBindDescriptorSets-pDescriptorSets-08010
  Each element of pDescriptorSets **must** have been allocated with a VkDescriptorSetLayout which was not created with VK_DESCRIPTOR_SET_LAYOUT_CREATE_DESCRIPTOR_BUFFER_BIT_EXT
- VUID-vkCmdBindDescriptorSets-pipelineBindPoint-00361
  pipelineBindPoint **must** be supported by the commandBuffer's parent VkCommandPool's queue family

## Valid Usage (Implicit)

- VUID-vkCmdBindDescriptorSets-commandBuffer-parameter
  commandBuffer **must** be a valid VkCommandBuffer handle
- VUID-vkCmdBindDescriptorSets-pipelineBindPoint-parameter
  pipelineBindPoint **must** be a valid VkPipelineBindPoint value
- VUID-vkCmdBindDescriptorSets-layout-parameter
  layout **must** be a valid VkPipelineLayout handle
- VUID-vkCmdBindDescriptorSets-pDescriptorSets-parameter
  pDescriptorSets **must** be a valid pointer to an array of descriptorSetCount valid or VK_NULL_HANDLE VkDescriptorSet handles
- VUID-vkCmdBindDescriptorSets-pDynamicOffsets-parameter
  If dynamicOffsetCount is not 0, pDynamicOffsets **must** be a valid pointer to an array of dynamicOffsetCount uint32_t values
- VUID-vkCmdBindDescriptorSets-commandBuffer-recording
  commandBuffer **must** be in the recording state
- VUID-vkCmdBindDescriptorSets-commandBuffer-cmdpool
  The VkCommandPool that commandBuffer was allocated from **must** support graphics, or compute operations
- VUID-vkCmdBindDescriptorSets-videocoding
  This command **must** only be called outside of a video coding scope
- VUID-vkCmdBindDescriptorSets-descriptorSetCount-arraylength
  descriptorSetCount **must** be greater than 0
- VUID-vkCmdBindDescriptorSets-commonparent
  Each of commandBuffer, layout, and the elements of pDescriptorSets that are valid handles of non-ignored parameters **must** have been created, allocated, or retrieved from the same VkDevice

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics<br>Compute | State |

Alternatively, to bind one or more descriptor sets to a command buffer, call:

```
// Provided by VK_KHR_maintenance6
void vkCmdBindDescriptorSets2KHR(
    VkCommandBuffer                             commandBuffer,
    const VkBindDescriptorSetsInfoKHR*          pBindDescriptorSetsInfo);
```

- `commandBuffer` is the command buffer that the descriptor sets will be bound to.
- `pBindDescriptorSetsInfo` is a pointer to a `VkBindDescriptorSetsInfoKHR` structure.

## Valid Usage

- VUID-vkCmdBindDescriptorSets2KHR-pBindDescriptorSetsInfo-09467
  Each bit in `pBindDescriptorSetsInfo->stageFlags` **must** be a stage supported by the `commandBuffer`'s parent `VkCommandPool`'s queue family

## Valid Usage (Implicit)

- VUID-vkCmdBindDescriptorSets2KHR-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdBindDescriptorSets2KHR-pBindDescriptorSetsInfo-parameter
  `pBindDescriptorSetsInfo` **must** be a valid pointer to a valid VkBindDescriptorSetsInfoKHR structure
- VUID-vkCmdBindDescriptorSets2KHR-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdBindDescriptorSets2KHR-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics, or compute operations

- VUID-vkCmdBindDescriptorSets2KHR-videocoding
  This command **must** only be called outside of a video coding scope

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics<br>Compute | State |

The `VkBindDescriptorSetsInfoKHR` structure is defined as:

```
// Provided by VK_KHR_maintenance6
typedef struct VkBindDescriptorSetsInfoKHR {
    VkStructureType           sType;
    const void*               pNext;
    VkShaderStageFlags        stageFlags;
    VkPipelineLayout          layout;
    uint32_t                  firstSet;
    uint32_t                  descriptorSetCount;
    const VkDescriptorSet*    pDescriptorSets;
    uint32_t                  dynamicOffsetCount;
    const uint32_t*           pDynamicOffsets;
} VkBindDescriptorSetsInfoKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `stageFlags` is a bitmask of VkShaderStageFlagBits specifying the shader stages the descriptor sets will be bound to.
- `layout` is a VkPipelineLayout object used to program the bindings. If the `dynamicPipelineLayout` feature is enabled, `layout` **can** be VK_NULL_HANDLE and the layout **must** be specified by chaining the VkPipelineLayoutCreateInfo structure off the `pNext`
- `firstSet` is the set number of the first descriptor set to be bound.
- `descriptorSetCount` is the number of elements in the `pDescriptorSets` array.
- `pDescriptorSets` is a pointer to an array of handles to VkDescriptorSet objects describing the

descriptor sets to bind to.

- dynamicOffsetCount is the number of dynamic offsets in the pDynamicOffsets array.
- pDynamicOffsets is a pointer to an array of uint32_t values specifying dynamic offsets.

If stageFlags specifies a subset of all stages corresponding to one or more pipeline bind points, the binding operation still affects all stages corresponding to the given pipeline bind point(s) as if the equivalent original version of this command had been called with the same parameters. For example, specifying a stageFlags value of VK_SHADER_STAGE_VERTEX_BIT | VK_SHADER_STAGE_FRAGMENT_BIT | VK_SHADER_STAGE_COMPUTE_BIT is equivalent to calling the original version of this command once with VK_PIPELINE_BIND_POINT_GRAPHICS and once with VK_PIPELINE_BIND_POINT_COMPUTE.

## Valid Usage

- VUID-VkBindDescriptorSetsInfoKHR-pDescriptorSets-00358
  Each element of pDescriptorSets **must** have been allocated with a VkDescriptorSetLayout that matches (is the same as, or identically defined as) the VkDescriptorSetLayout at set *n* in layout, where *n* is the sum of firstSet and the index into pDescriptorSets
- VUID-VkBindDescriptorSetsInfoKHR-dynamicOffsetCount-00359
  dynamicOffsetCount **must** be equal to the total number of dynamic descriptors in pDescriptorSets
- VUID-VkBindDescriptorSetsInfoKHR-firstSet-00360
  The sum of firstSet and descriptorSetCount **must** be less than or equal to VkPipelineLayoutCreateInfo::setLayoutCount provided when layout was created
- VUID-VkBindDescriptorSetsInfoKHR-pDynamicOffsets-01971
  Each element of pDynamicOffsets which corresponds to a descriptor binding with type VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC **must** be a multiple of VkPhysicalDeviceLimits::minUniformBufferOffsetAlignment
- VUID-VkBindDescriptorSetsInfoKHR-pDynamicOffsets-01972
  Each element of pDynamicOffsets which corresponds to a descriptor binding with type VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC **must** be a multiple of VkPhysicalDeviceLimits::minStorageBufferOffsetAlignment
- VUID-VkBindDescriptorSetsInfoKHR-pDescriptorSets-01979
  For each dynamic uniform or storage buffer binding in pDescriptorSets, the sum of the effective offset and the range of the binding **must** be less than or equal to the size of the buffer
- VUID-VkBindDescriptorSetsInfoKHR-pDescriptorSets-06715
  For each dynamic uniform or storage buffer binding in pDescriptorSets, if the range was set with VK_WHOLE_SIZE then pDynamicOffsets which corresponds to the descriptor binding **must** be 0
- VUID-VkBindDescriptorSetsInfoKHR-pDescriptorSets-04616
  Each element of pDescriptorSets **must** not have been allocated from a VkDescriptorPool with the VK_DESCRIPTOR_POOL_CREATE_HOST_ONLY_BIT_EXT flag set
- VUID-VkBindDescriptorSetsInfoKHR-pDescriptorSets-06563

If graphicsPipelineLibrary is not enabled, each element of pDescriptorSets **must** be a valid VkDescriptorSet

- VUID-VkBindDescriptorSetsInfoKHR-pDescriptorSets-08010
  Each element of pDescriptorSets **must** have been allocated with a VkDescriptorSetLayout which was not created with VK_DESCRIPTOR_SET_LAYOUT_CREATE_DESCRIPTOR_BUFFER_BIT_EXT

- VUID-VkBindDescriptorSetsInfoKHR-None-09495
  If the dynamicPipelineLayout feature is not enabled, layout **must** be a valid VkPipelineLayout handle

- VUID-VkBindDescriptorSetsInfoKHR-layout-09496
  If layout is VK_NULL_HANDLE, the pNext chain **must** include a valid VkPipelineLayoutCreateInfo structure

### Valid Usage (Implicit)

- VUID-VkBindDescriptorSetsInfoKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_BIND_DESCRIPTOR_SETS_INFO_KHR

- VUID-VkBindDescriptorSetsInfoKHR-pNext-pNext
  pNext **must** be NULL or a pointer to a valid instance of VkPipelineLayoutCreateInfo

- VUID-VkBindDescriptorSetsInfoKHR-sType-unique
  The sType value of each struct in the pNext chain **must** be unique

- VUID-VkBindDescriptorSetsInfoKHR-stageFlags-parameter
  stageFlags **must** be a valid combination of VkShaderStageFlagBits values

- VUID-VkBindDescriptorSetsInfoKHR-stageFlags-requiredbitmask
  stageFlags **must** not be 0

- VUID-VkBindDescriptorSetsInfoKHR-layout-parameter
  If layout is not VK_NULL_HANDLE, layout **must** be a valid VkPipelineLayout handle

- VUID-VkBindDescriptorSetsInfoKHR-pDescriptorSets-parameter
  pDescriptorSets **must** be a valid pointer to an array of descriptorSetCount valid VkDescriptorSet handles

- VUID-VkBindDescriptorSetsInfoKHR-pDynamicOffsets-parameter
  If dynamicOffsetCount is not 0, and pDynamicOffsets is not NULL, pDynamicOffsets **must** be a valid pointer to an array of dynamicOffsetCount or VK_NULL_HANDLE uint32_t values

- VUID-VkBindDescriptorSetsInfoKHR-descriptorSetCount-arraylength
  descriptorSetCount **must** be greater than 0

- VUID-VkBindDescriptorSetsInfoKHR-commonparent
  Both of layout, and the elements of pDescriptorSets that are valid handles of non-ignored parameters **must** have been created, allocated, or retrieved from the same VkDevice

## 14.2.8. Push Descriptor Updates

In addition to allocating descriptor sets and binding them to a command buffer, an application **can**

record descriptor updates into the command buffer.

To push descriptor updates into a command buffer, call:

```
// Provided by VK_KHR_push_descriptor
void vkCmdPushDescriptorSetKHR(
    VkCommandBuffer                             commandBuffer,
    VkPipelineBindPoint                         pipelineBindPoint,
    VkPipelineLayout                            layout,
    uint32_t                                    set,
    uint32_t                                    descriptorWriteCount,
    const VkWriteDescriptorSet*                 pDescriptorWrites);
```

- `commandBuffer` is the command buffer that the descriptors will be recorded in.
- `pipelineBindPoint` is a VkPipelineBindPoint indicating the type of the pipeline that will use the descriptors. There is a separate set of push descriptor bindings for each pipeline type, so binding one does not disturb the others.
- `layout` is a VkPipelineLayout object used to program the bindings.
- `set` is the set number of the descriptor set in the pipeline layout that will be updated.
- `descriptorWriteCount` is the number of elements in the `pDescriptorWrites` array.
- `pDescriptorWrites` is a pointer to an array of VkWriteDescriptorSet structures describing the descriptors to be updated.

*Push descriptors* are a small bank of descriptors whose storage is internally managed by the command buffer rather than being written into a descriptor set and later bound to a command buffer. Push descriptors allow for incremental updates of descriptors without managing the lifetime of descriptor sets.

When a command buffer begins recording, all push descriptors are undefined. Push descriptors **can** be updated incrementally and cause shaders to use the updated descriptors for subsequent bound pipeline commands with the pipeline type set by `pipelineBindPoint` until the descriptor is overwritten, or else until the set is disturbed as described in Pipeline Layout Compatibility. When the set is disturbed or push descriptors with a different descriptor set layout are set, all push descriptors are undefined.

Push descriptors that are statically used by a pipeline **must** not be undefined at the time that a drawing or dispatching command is recorded to execute using that pipeline. This includes immutable sampler descriptors, which **must** be pushed before they are accessed by a pipeline (the immutable samplers are pushed, rather than the samplers in `pDescriptorWrites`). Push descriptors that are not statically used **can** remain undefined.

Push descriptors do not use dynamic offsets. Instead, the corresponding non-dynamic descriptor types **can** be used and the `offset` member of VkDescriptorBufferInfo **can** be changed each time the descriptor is written.

Each element of `pDescriptorWrites` is interpreted as in VkWriteDescriptorSet, except the `dstSet` member is ignored.

To push an immutable sampler, use a VkWriteDescriptorSet with dstBinding and dstArrayElement selecting the immutable sampler's binding. If the descriptor type is VK_DESCRIPTOR_TYPE_SAMPLER, the pImageInfo parameter is ignored and the immutable sampler is taken from the push descriptor set layout in the pipeline layout. If the descriptor type is VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER, the sampler member of the pImageInfo parameter is ignored and the immutable sampler is taken from the push descriptor set layout in the pipeline layout.

## Valid Usage

- VUID-vkCmdPushDescriptorSetKHR-set-00364
  set **must** be less than VkPipelineLayoutCreateInfo::setLayoutCount provided when layout was created
- VUID-vkCmdPushDescriptorSetKHR-set-00365
  set **must** be the unique set number in the pipeline layout that uses a descriptor set layout that was created with VK_DESCRIPTOR_SET_LAYOUT_CREATE_PUSH_DESCRIPTOR_BIT_KHR
- VUID-vkCmdPushDescriptorSetKHR-pDescriptorWrites-06494
  For each element i where pDescriptorWrites[i].descriptorType is VK_DESCRIPTOR_TYPE_SAMPLER, VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER, VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE, VK_DESCRIPTOR_TYPE_STORAGE_IMAGE, or VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT, pDescriptorWrites[i].pImageInfo **must** be a valid pointer to an array of pDescriptorWrites[i].descriptorCount valid VkDescriptorImageInfo structures
- VUID-vkCmdPushDescriptorSetKHR-pipelineBindPoint-00363
  pipelineBindPoint **must** be supported by the commandBuffer's parent VkCommandPool's queue family

## Valid Usage (Implicit)

- VUID-vkCmdPushDescriptorSetKHR-commandBuffer-parameter
  commandBuffer **must** be a valid VkCommandBuffer handle
- VUID-vkCmdPushDescriptorSetKHR-pipelineBindPoint-parameter
  pipelineBindPoint **must** be a valid VkPipelineBindPoint value
- VUID-vkCmdPushDescriptorSetKHR-layout-parameter
  layout **must** be a valid VkPipelineLayout handle
- VUID-vkCmdPushDescriptorSetKHR-pDescriptorWrites-parameter
  pDescriptorWrites **must** be a valid pointer to an array of descriptorWriteCount valid VkWriteDescriptorSet structures
- VUID-vkCmdPushDescriptorSetKHR-commandBuffer-recording
  commandBuffer **must** be in the recording state
- VUID-vkCmdPushDescriptorSetKHR-commandBuffer-cmdpool
  The VkCommandPool that commandBuffer was allocated from **must** support graphics, or compute operations
- VUID-vkCmdPushDescriptorSetKHR-videocoding

This command **must** only be called outside of a video coding scope

- VUID-vkCmdPushDescriptorSetKHR-descriptorWriteCount-arraylength
  `descriptorWriteCount` **must** be greater than `0`
- VUID-vkCmdPushDescriptorSetKHR-commonparent
  Both of `commandBuffer`, and `layout` **must** have been created, allocated, or retrieved from the same VkDevice

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics<br>Compute | State |

Alternatively, to push descriptor updates into a command buffer, call:

```
// Provided by VK_KHR_maintenance6 with VK_KHR_push_descriptor
void vkCmdPushDescriptorSet2KHR(
    VkCommandBuffer                             commandBuffer,
    const VkPushDescriptorSetInfoKHR*           pPushDescriptorSetInfo);
```

- `commandBuffer` is the command buffer that the descriptors will be recorded in.
- `pPushDescriptorSetInfo` is a pointer to a `VkPushDescriptorSetInfoKHR` structure.

## Valid Usage

- VUID-vkCmdPushDescriptorSet2KHR-pPushDescriptorSetInfo-09468
  Each bit in `pPushDescriptorSetInfo->stageFlags` **must** be a stage supported by the `commandBuffer`'s parent `VkCommandPool`'s queue family

## Valid Usage (Implicit)

- VUID-vkCmdPushDescriptorSet2KHR-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle

- VUID-vkCmdPushDescriptorSet2KHR-pPushDescriptorSetInfo-parameter
  `pPushDescriptorSetInfo` **must** be a valid pointer to a valid VkPushDescriptorSetInfoKHR structure
- VUID-vkCmdPushDescriptorSet2KHR-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdPushDescriptorSet2KHR-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics, or compute operations
- VUID-vkCmdPushDescriptorSet2KHR-videocoding
  This command **must** only be called outside of a video coding scope

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics<br>Compute | State |

The `VkPushDescriptorSetInfoKHR` structure is defined as:

```
// Provided by VK_KHR_maintenance6 with VK_KHR_push_descriptor
typedef struct VkPushDescriptorSetInfoKHR {
    VkStructureType                 sType;
    const void*                     pNext;
    VkShaderStageFlags              stageFlags;
    VkPipelineLayout                layout;
    uint32_t                        set;
    uint32_t                        descriptorWriteCount;
    const VkWriteDescriptorSet*     pDescriptorWrites;
} VkPushDescriptorSetInfoKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `stageFlags` is a bitmask of VkShaderStageFlagBits specifying the shader stages that will use the descriptors.

- layout is a VkPipelineLayout object used to program the bindings. If the dynamicPipelineLayout feature is enabled, layout **can** be VK_NULL_HANDLE and the layout **must** be specified by chaining VkPipelineLayoutCreateInfo structure off the pNext
- set is the set number of the descriptor set in the pipeline layout that will be updated.
- descriptorWriteCount is the number of elements in the pDescriptorWrites array.
- pDescriptorWrites is a pointer to an array of VkWriteDescriptorSet structures describing the descriptors to be updated.

If stageFlags specifies a subset of all stages corresponding to one or more pipeline bind points, the binding operation still affects all stages corresponding to the given pipeline bind point(s) as if the equivalent original version of this command had been called with the same parameters. For example, specifying a stageFlags value of VK_SHADER_STAGE_VERTEX_BIT | VK_SHADER_STAGE_FRAGMENT_BIT | VK_SHADER_STAGE_COMPUTE_BIT is equivalent to calling the original version of this command once with VK_PIPELINE_BIND_POINT_GRAPHICS and once with VK_PIPELINE_BIND_POINT_COMPUTE.

## Valid Usage

- VUID-VkPushDescriptorSetInfoKHR-set-00364
  set **must** be less than VkPipelineLayoutCreateInfo::setLayoutCount provided when layout was created
- VUID-VkPushDescriptorSetInfoKHR-set-00365
  set **must** be the unique set number in the pipeline layout that uses a descriptor set layout that was created with VK_DESCRIPTOR_SET_LAYOUT_CREATE_PUSH_DESCRIPTOR_BIT_KHR
- VUID-VkPushDescriptorSetInfoKHR-pDescriptorWrites-06494
  For each element i where pDescriptorWrites[i].descriptorType is VK_DESCRIPTOR_TYPE_SAMPLER, VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER, VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE, VK_DESCRIPTOR_TYPE_STORAGE_IMAGE, or VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT, pDescriptorWrites[i].pImageInfo **must** be a valid pointer to an array of pDescriptorWrites[i].descriptorCount valid VkDescriptorImageInfo structures
- VUID-VkPushDescriptorSetInfoKHR-None-09495
  If the dynamicPipelineLayout feature is not enabled, layout **must** be a valid VkPipelineLayout handle
- VUID-VkPushDescriptorSetInfoKHR-layout-09496
  If layout is VK_NULL_HANDLE, the pNext chain **must** include a valid VkPipelineLayoutCreateInfo structure

## Valid Usage (Implicit)

- VUID-VkPushDescriptorSetInfoKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PUSH_DESCRIPTOR_SET_INFO_KHR
- VUID-VkPushDescriptorSetInfoKHR-pNext-pNext

pNext **must** be NULL or a pointer to a valid instance of VkPipelineLayoutCreateInfo

- VUID-VkPushDescriptorSetInfoKHR-sType-unique
  The sType value of each struct in the pNext chain **must** be unique
- VUID-VkPushDescriptorSetInfoKHR-stageFlags-parameter
  stageFlags **must** be a valid combination of VkShaderStageFlagBits values
- VUID-VkPushDescriptorSetInfoKHR-stageFlags-requiredbitmask
  stageFlags **must** not be 0
- VUID-VkPushDescriptorSetInfoKHR-layout-parameter
  If layout is not VK_NULL_HANDLE, layout **must** be a valid VkPipelineLayout handle
- VUID-VkPushDescriptorSetInfoKHR-pDescriptorWrites-parameter
  pDescriptorWrites **must** be a valid pointer to an array of descriptorWriteCount valid VkWriteDescriptorSet structures
- VUID-VkPushDescriptorSetInfoKHR-descriptorWriteCount-arraylength
  descriptorWriteCount **must** be greater than 0

### 14.2.9. Push Descriptor Updates With Descriptor Update Templates

It is also possible to use a descriptor update template to specify the push descriptors to update. To do so, call:

```
// Provided by VK_KHR_descriptor_update_template with VK_KHR_push_descriptor,
VK_KHR_push_descriptor with VK_VERSION_1_1 or VK_KHR_descriptor_update_template
void vkCmdPushDescriptorSetWithTemplateKHR(
    VkCommandBuffer                             commandBuffer,
    VkDescriptorUpdateTemplate                  descriptorUpdateTemplate,
    VkPipelineLayout                            layout,
    uint32_t                                    set,
    const void*                                 pData);
```

- commandBuffer is the command buffer that the descriptors will be recorded in.
- descriptorUpdateTemplate is a descriptor update template defining how to interpret the descriptor information in pData.
- layout is a VkPipelineLayout object used to program the bindings. It **must** be compatible with the layout used to create the descriptorUpdateTemplate handle.
- set is the set number of the descriptor set in the pipeline layout that will be updated. This **must** be the same number used to create the descriptorUpdateTemplate handle.
- pData is a pointer to memory containing descriptors for the templated update.

#### Valid Usage

- VUID-vkCmdPushDescriptorSetWithTemplateKHR-commandBuffer-00366
  The pipelineBindPoint specified during the creation of the descriptor update template

**must** be supported by the `commandBuffer`'s parent `VkCommandPool`'s queue family

- VUID-vkCmdPushDescriptorSetWithTemplateKHR-pData-01686
  `pData` **must** be a valid pointer to a memory containing one or more valid instances of VkDescriptorImageInfo, VkDescriptorBufferInfo, or VkBufferView in a layout defined by `descriptorUpdateTemplate` when it was created with vkCreateDescriptorUpdateTemplate
- VUID-vkCmdPushDescriptorSetWithTemplateKHR-layout-07993
  `layout` **must** be compatible with the layout used to create `descriptorUpdateTemplate`
- VUID-vkCmdPushDescriptorSetWithTemplateKHR-descriptorUpdateTemplate-07994
  `descriptorUpdateTemplate` **must** have been created with a `templateType` of `VK_DESCRIPTOR_UPDATE_TEMPLATE_TYPE_PUSH_DESCRIPTORS_KHR`
- VUID-vkCmdPushDescriptorSetWithTemplateKHR-set-07995
  `set` **must** be the same value used to create `descriptorUpdateTemplate`
- VUID-vkCmdPushDescriptorSetWithTemplateKHR-set-07304
  `set` **must** be less than VkPipelineLayoutCreateInfo::`setLayoutCount` provided when `layout` was created
- VUID-vkCmdPushDescriptorSetWithTemplateKHR-set-07305
  `set` **must** be the unique set number in the pipeline layout that uses a descriptor set layout that was created with `VK_DESCRIPTOR_SET_LAYOUT_CREATE_PUSH_DESCRIPTOR_BIT_KHR`

## Valid Usage (Implicit)

- VUID-vkCmdPushDescriptorSetWithTemplateKHR-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdPushDescriptorSetWithTemplateKHR-descriptorUpdateTemplate-parameter
  `descriptorUpdateTemplate` **must** be a valid VkDescriptorUpdateTemplate handle
- VUID-vkCmdPushDescriptorSetWithTemplateKHR-layout-parameter
  `layout` **must** be a valid VkPipelineLayout handle
- VUID-vkCmdPushDescriptorSetWithTemplateKHR-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdPushDescriptorSetWithTemplateKHR-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics, or compute operations
- VUID-vkCmdPushDescriptorSetWithTemplateKHR-videocoding
  This command **must** only be called outside of a video coding scope
- VUID-vkCmdPushDescriptorSetWithTemplateKHR-commonparent
  Each of `commandBuffer`, `descriptorUpdateTemplate`, and `layout` **must** have been created, allocated, or retrieved from the same VkDevice

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized

- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics<br>Compute | State |

*API example*

```
struct AppDataStructure
{
    VkDescriptorImageInfo  imageInfo;          // a single image info
    // ... some more application-related data
};

const VkDescriptorUpdateTemplateEntry descriptorUpdateTemplateEntries[] =
{
    // binding to a single image descriptor
    {
        .binding = 0,
        .dstArrayElement = 0,
        .descriptorCount = 1,
        .descriptorType = VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER,
        .offset = offsetof(AppDataStructure, imageInfo),
        .stride = 0     // not required if descriptorCount is 1
    }
};

// create a descriptor update template for push descriptor set updates
const VkDescriptorUpdateTemplateCreateInfo createInfo =
{
    .sType = VK_STRUCTURE_TYPE_DESCRIPTOR_UPDATE_TEMPLATE_CREATE_INFO,
    .pNext = NULL,
    .flags = 0,
    .descriptorUpdateEntryCount = 1,
    .pDescriptorUpdateEntries = descriptorUpdateTemplateEntries,
    .templateType = VK_DESCRIPTOR_UPDATE_TEMPLATE_TYPE_PUSH_DESCRIPTORS_KHR,
    .descriptorSetLayout = 0,   // ignored by given templateType
    .pipelineBindPoint = VK_PIPELINE_BIND_POINT_GRAPHICS,
    .pipelineLayout = myPipelineLayout,
    .set = 0,
};

VkDescriptorUpdateTemplate myDescriptorUpdateTemplate;
myResult = vkCreateDescriptorUpdateTemplate(
```

```
    myDevice,
    &createInfo,
    NULL,
    &myDescriptorUpdateTemplate);

AppDataStructure appData;
// fill appData here or cache it in your engine
vkCmdPushDescriptorSetWithTemplateKHR(myCmdBuffer, myDescriptorUpdateTemplate,
myPipelineLayout, 0,&appData);
```

Alternatively, to use a descriptor update template to specify the push descriptors to update, call:

```
// Provided by VK_KHR_maintenance6 with VK_KHR_push_descriptor
void vkCmdPushDescriptorSetWithTemplate2KHR(
    VkCommandBuffer                             commandBuffer,
    const VkPushDescriptorSetWithTemplateInfoKHR* pPushDescriptorSetWithTemplateInfo);
```

- `commandBuffer` is the command buffer that the descriptors will be recorded in.
- `pPushDescriptorSetWithTemplateInfo` is a pointer to a `VkPushDescriptorSetWithTemplateInfoKHR` structure.

## Valid Usage (Implicit)

- VUID-vkCmdPushDescriptorSetWithTemplate2KHR-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdPushDescriptorSetWithTemplate2KHR-pPushDescriptorSetWithTemplateInfo-parameter
  `pPushDescriptorSetWithTemplateInfo` **must** be a valid pointer to a valid VkPushDescriptorSetWithTemplateInfoKHR structure
- VUID-vkCmdPushDescriptorSetWithTemplate2KHR-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdPushDescriptorSetWithTemplate2KHR-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics, or compute operations
- VUID-vkCmdPushDescriptorSetWithTemplate2KHR-videocoding
  This command **must** only be called outside of a video coding scope

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics<br>Compute | State |

The VkPushDescriptorSetWithTemplateInfoKHR structure is defined as:

```
// Provided by VK_KHR_maintenance6 with VK_KHR_push_descriptor
typedef struct VkPushDescriptorSetWithTemplateInfoKHR {
    VkStructureType                 sType;
    const void*                     pNext;
    VkDescriptorUpdateTemplate      descriptorUpdateTemplate;
    VkPipelineLayout                layout;
    uint32_t                        set;
    const void*                     pData;
} VkPushDescriptorSetWithTemplateInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- descriptorUpdateTemplate is a descriptor update template defining how to interpret the descriptor information in pData.
- layout is a VkPipelineLayout object used to program the bindings. It **must** be compatible with the layout used to create the descriptorUpdateTemplate handle. If the dynamicPipelineLayout feature is enabled, layout **can** be VK_NULL_HANDLE and the layout **must** be specified by chaining VkPipelineLayoutCreateInfo structure off the pNext
- set is the set number of the descriptor set in the pipeline layout that will be updated. This **must** be the same number used to create the descriptorUpdateTemplate handle.
- pData is a pointer to memory containing descriptors for the templated update.

## Valid Usage

- VUID-VkPushDescriptorSetWithTemplateInfoKHR-commandBuffer-00366
  The pipelineBindPoint specified during the creation of the descriptor update template **must** be supported by the commandBuffer's parent VkCommandPool's queue family
- VUID-VkPushDescriptorSetWithTemplateInfoKHR-pData-01686
  pData **must** be a valid pointer to a memory containing one or more valid instances of VkDescriptorImageInfo, VkDescriptorBufferInfo, or VkBufferView in a layout defined by descriptorUpdateTemplate when it was created with vkCreateDescriptorUpdateTemplate
- VUID-VkPushDescriptorSetWithTemplateInfoKHR-layout-07993
  layout **must** be compatible with the layout used to create descriptorUpdateTemplate

- VUID-VkPushDescriptorSetWithTemplateInfoKHR-descriptorUpdateTemplate-07994
  `descriptorUpdateTemplate` **must** have been created with a `templateType` of `VK_DESCRIPTOR_UPDATE_TEMPLATE_TYPE_PUSH_DESCRIPTORS_KHR`
- VUID-VkPushDescriptorSetWithTemplateInfoKHR-set-07995
  `set` **must** be the same value used to create `descriptorUpdateTemplate`
- VUID-VkPushDescriptorSetWithTemplateInfoKHR-set-07304
  `set` **must** be less than VkPipelineLayoutCreateInfo::`setLayoutCount` provided when `layout` was created
- VUID-VkPushDescriptorSetWithTemplateInfoKHR-set-07305
  `set` **must** be the unique set number in the pipeline layout that uses a descriptor set layout that was created with `VK_DESCRIPTOR_SET_LAYOUT_CREATE_PUSH_DESCRIPTOR_BIT_KHR`
- VUID-VkPushDescriptorSetWithTemplateInfoKHR-None-09495
  If the `dynamicPipelineLayout` feature is not enabled, `layout` **must** be a valid VkPipelineLayout handle
- VUID-VkPushDescriptorSetWithTemplateInfoKHR-layout-09496
  If `layout` is VK_NULL_HANDLE, the `pNext` chain **must** include a valid VkPipelineLayoutCreateInfo structure

### Valid Usage (Implicit)

- VUID-VkPushDescriptorSetWithTemplateInfoKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PUSH_DESCRIPTOR_SET_WITH_TEMPLATE_INFO_KHR`
- VUID-VkPushDescriptorSetWithTemplateInfoKHR-pNext-pNext
  `pNext` **must** be `NULL` or a pointer to a valid instance of VkPipelineLayoutCreateInfo
- VUID-VkPushDescriptorSetWithTemplateInfoKHR-sType-unique
  The `sType` value of each struct in the `pNext` chain **must** be unique
- VUID-VkPushDescriptorSetWithTemplateInfoKHR-descriptorUpdateTemplate-parameter
  `descriptorUpdateTemplate` **must** be a valid VkDescriptorUpdateTemplate handle
- VUID-VkPushDescriptorSetWithTemplateInfoKHR-layout-parameter
  If `layout` is not VK_NULL_HANDLE, `layout` **must** be a valid VkPipelineLayout handle
- VUID-VkPushDescriptorSetWithTemplateInfoKHR-pData-parameter
  `pData` **must** be a pointer value
- VUID-VkPushDescriptorSetWithTemplateInfoKHR-commonparent
  Both of `descriptorUpdateTemplate`, and `layout` that are valid handles of non-ignored parameters **must** have been created, allocated, or retrieved from the same VkDevice

## 14.2.10. Push Constant Updates

As described above in section Pipeline Layouts, the pipeline layout defines shader push constants which are updated via Vulkan commands rather than via writes to memory or copy commands.

**NOTE** Push constants represent a high speed path to modify constant data in pipelines that is expected to outperform memory-backed resource updates.

To update push constants, call:

```
// Provided by VK_VERSION_1_0
void vkCmdPushConstants(
    VkCommandBuffer                             commandBuffer,
    VkPipelineLayout                            layout,
    VkShaderStageFlags                          stageFlags,
    uint32_t                                    offset,
    uint32_t                                    size,
    const void*                                 pValues);
```

- `commandBuffer` is the command buffer in which the push constant update will be recorded.
- `layout` is the pipeline layout used to program the push constant updates.
- `stageFlags` is a bitmask of VkShaderStageFlagBits specifying the shader stages that will use the push constants in the updated range.
- `offset` is the start offset of the push constant range to update, in units of bytes.
- `size` is the size of the push constant range to update, in units of bytes.
- `pValues` is a pointer to an array of `size` bytes containing the new push constant values.

When a command buffer begins recording, all push constant values are undefined. Reads of undefined push constant values by the executing shader return undefined values.

Push constant values **can** be updated incrementally, causing shader stages in `stageFlags` to read the new data from `pValues` for push constants modified by this command, while still reading the previous data for push constants not modified by this command. When a bound pipeline command is issued, the bound pipeline's layout **must** be compatible with the layouts used to set the values of all push constants in the pipeline layout's push constant ranges, as described in Pipeline Layout Compatibility. Binding a pipeline with a layout that is not compatible with the push constant layout does not disturb the push constant values.

**NOTE** As `stageFlags` needs to include all flags the relevant push constant ranges were created with, any flags that are not supported by the queue family that the VkCommandPool used to allocate `commandBuffer` was created on are ignored.

### Valid Usage

- VUID-vkCmdPushConstants-offset-01795
  For each byte in the range specified by `offset` and `size` and for each shader stage in `stageFlags`, there **must** be a push constant range in `layout` that includes that byte and that stage
- VUID-vkCmdPushConstants-offset-01796
  For each byte in the range specified by `offset` and `size` and for each push constant range

that overlaps that byte, `stageFlags` **must** include all stages in that push constant range's VkPushConstantRange::`stageFlags`

- VUID-vkCmdPushConstants-offset-00368
  `offset` **must** be a multiple of `4`
- VUID-vkCmdPushConstants-size-00369
  `size` **must** be a multiple of `4`
- VUID-vkCmdPushConstants-offset-00370
  `offset` **must** be less than `VkPhysicalDeviceLimits`::`maxPushConstantsSize`
- VUID-vkCmdPushConstants-size-00371
  `size` **must** be less than or equal to `VkPhysicalDeviceLimits`::`maxPushConstantsSize` minus `offset`

## Valid Usage (Implicit)

- VUID-vkCmdPushConstants-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdPushConstants-layout-parameter
  `layout` **must** be a valid VkPipelineLayout handle
- VUID-vkCmdPushConstants-stageFlags-parameter
  `stageFlags` **must** be a valid combination of VkShaderStageFlagBits values
- VUID-vkCmdPushConstants-stageFlags-requiredbitmask
  `stageFlags` **must** not be `0`
- VUID-vkCmdPushConstants-pValues-parameter
  `pValues` **must** be a valid pointer to an array of `size` bytes
- VUID-vkCmdPushConstants-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdPushConstants-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics, or compute operations
- VUID-vkCmdPushConstants-videocoding
  This command **must** only be called outside of a video coding scope
- VUID-vkCmdPushConstants-size-arraylength
  `size` **must** be greater than `0`
- VUID-vkCmdPushConstants-commonparent
  Both of `commandBuffer`, and `layout` **must** have been created, allocated, or retrieved from the same VkDevice

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized

- Host access to the VkCommandPool that commandBuffer was allocated from **must** be externally synchronized

### Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics<br>Compute | State |

Alternatively, to update push constants, call:

```
// Provided by VK_KHR_maintenance6
void vkCmdPushConstants2KHR(
    VkCommandBuffer                             commandBuffer,
    const VkPushConstantsInfoKHR*               pPushConstantsInfo);
```

- commandBuffer is the command buffer in which the push constant update will be recorded.
- pPushConstantsInfo is a pointer to a VkPushConstantsInfoKHR structure.

### Valid Usage (Implicit)

- VUID-vkCmdPushConstants2KHR-commandBuffer-parameter
  commandBuffer **must** be a valid VkCommandBuffer handle
- VUID-vkCmdPushConstants2KHR-pPushConstantsInfo-parameter
  pPushConstantsInfo **must** be a valid pointer to a valid VkPushConstantsInfoKHR structure
- VUID-vkCmdPushConstants2KHR-commandBuffer-recording
  commandBuffer **must** be in the recording state
- VUID-vkCmdPushConstants2KHR-commandBuffer-cmdpool
  The VkCommandPool that commandBuffer was allocated from **must** support graphics, or compute operations
- VUID-vkCmdPushConstants2KHR-videocoding
  This command **must** only be called outside of a video coding scope

### Host Synchronization

- Host access to commandBuffer **must** be externally synchronized
- Host access to the VkCommandPool that commandBuffer was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics<br>Compute | State |

The VkPushConstantsInfoKHR structure is defined as:

```
// Provided by VK_KHR_maintenance6
typedef struct VkPushConstantsInfoKHR {
    VkStructureType       sType;
    const void*           pNext;
    VkPipelineLayout      layout;
    VkShaderStageFlags    stageFlags;
    uint32_t              offset;
    uint32_t              size;
    const void*           pValues;
} VkPushConstantsInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- layout is the pipeline layout used to program the push constant updates. If the dynamicPipelineLayout feature is enabled, layout **can** be VK_NULL_HANDLE and the layout **must** be specified by chaining VkPipelineLayoutCreateInfo structure off the pNext
- stageFlags is a bitmask of VkShaderStageFlagBits specifying the shader stages that will use the push constants in the updated range.
- offset is the start offset of the push constant range to update, in units of bytes.
- size is the size of the push constant range to update, in units of bytes.
- pValues is a pointer to an array of size bytes containing the new push constant values.

## Valid Usage

- VUID-VkPushConstantsInfoKHR-offset-01795
  For each byte in the range specified by offset and size and for each shader stage in stageFlags, there **must** be a push constant range in layout that includes that byte and that stage
- VUID-VkPushConstantsInfoKHR-offset-01796
  For each byte in the range specified by offset and size and for each push constant range that overlaps that byte, stageFlags **must** include all stages in that push constant range's VkPushConstantRange::stageFlags
- VUID-VkPushConstantsInfoKHR-offset-00368

offset **must** be a multiple of 4

- VUID-VkPushConstantsInfoKHR-size-00369
  size **must** be a multiple of 4
- VUID-VkPushConstantsInfoKHR-offset-00370
  offset **must** be less than VkPhysicalDeviceLimits::maxPushConstantsSize
- VUID-VkPushConstantsInfoKHR-size-00371
  size **must** be less than or equal to VkPhysicalDeviceLimits::maxPushConstantsSize minus offset
- VUID-VkPushConstantsInfoKHR-None-09495
  If the dynamicPipelineLayout feature is not enabled, layout **must** be a valid VkPipelineLayout handle
- VUID-VkPushConstantsInfoKHR-layout-09496
  If layout is VK_NULL_HANDLE, the pNext chain **must** include a valid VkPipelineLayoutCreateInfo structure

### Valid Usage (Implicit)

- VUID-VkPushConstantsInfoKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PUSH_CONSTANTS_INFO_KHR
- VUID-VkPushConstantsInfoKHR-pNext-pNext
  pNext **must** be NULL or a pointer to a valid instance of VkPipelineLayoutCreateInfo
- VUID-VkPushConstantsInfoKHR-sType-unique
  The sType value of each struct in the pNext chain **must** be unique
- VUID-VkPushConstantsInfoKHR-layout-parameter
  If layout is not VK_NULL_HANDLE, layout **must** be a valid VkPipelineLayout handle
- VUID-VkPushConstantsInfoKHR-stageFlags-parameter
  stageFlags **must** be a valid combination of VkShaderStageFlagBits values
- VUID-VkPushConstantsInfoKHR-stageFlags-requiredbitmask
  stageFlags **must** not be 0
- VUID-VkPushConstantsInfoKHR-pValues-parameter
  pValues **must** be a valid pointer to an array of size bytes
- VUID-VkPushConstantsInfoKHR-size-arraylength
  size **must** be greater than 0

## 14.3. Physical Storage Buffer Access

To query a 64-bit buffer device address value through which buffer memory **can** be accessed in a shader, call:

```
// Provided by VK_KHR_buffer_device_address
VkDeviceAddress vkGetBufferDeviceAddressKHR(
    VkDevice                                    device,
    const VkBufferDeviceAddressInfo*            pInfo);
```

or the equivalent command

```
// Provided by VK_EXT_buffer_device_address
VkDeviceAddress vkGetBufferDeviceAddressEXT(
    VkDevice                                    device,
    const VkBufferDeviceAddressInfo*            pInfo);
```

- `device` is the logical device that the buffer was created on.
- `pInfo` is a pointer to a VkBufferDeviceAddressInfo structure specifying the buffer to retrieve an address for.

The 64-bit return value is an address of the start of `pInfo->buffer`. The address range starting at this value and whose size is the size of the buffer **can** be used in a shader to access the memory bound to that buffer, using the `SPV_KHR_physical_storage_buffer` extension or the equivalent `SPV_EXT_physical_storage_buffer` extension and the `PhysicalStorageBuffer` storage class. For example, this value **can** be stored in a uniform buffer, and the shader **can** read the value from the uniform buffer and use it to do a dependent read/write to this buffer. A value of zero is reserved as a "null" pointer and **must** not be returned as a valid buffer device address. All loads, stores, and atomics in a shader through `PhysicalStorageBuffer` pointers **must** access addresses in the address range of some buffer.

If the buffer was created with a non-zero value of VkBufferOpaqueCaptureAddressCreateInfo::`opaqueCaptureAddress` or VkBufferDeviceAddressCreateInfoEXT::`deviceAddress`, the return value will be the same address that was returned at capture time.

The returned address **must** satisfy the alignment requirement specified by VkMemoryRequirements::`alignment` for the buffer in VkBufferDeviceAddressInfo::`buffer`.

If multiple VkBuffer objects are bound to overlapping ranges of VkDeviceMemory, implementations **may** return address ranges which overlap. In this case, it is ambiguous which VkBuffer is associated with any given device address. For purposes of valid usage, if multiple VkBuffer objects **can** be attributed to a device address, a VkBuffer is selected such that valid usage passes, if it exists.

### Valid Usage

- VUID-vkGetBufferDeviceAddress-bufferDeviceAddress-03324
  The `bufferDeviceAddress` or `VkPhysicalDeviceBufferDeviceAddressFeaturesEXT::bufferDeviceAddress` feature **must** be enabled
- VUID-vkGetBufferDeviceAddress-device-03325
  If `device` was created with multiple physical devices, then the

bufferDeviceAddressMultiDevice or VkPhysicalDeviceBufferDeviceAddressFeaturesEXT::bufferDeviceAddressMultiDevice feature **must** be enabled

### Valid Usage (Implicit)

- VUID-vkGetBufferDeviceAddress-device-parameter
  device **must** be a valid VkDevice handle
- VUID-vkGetBufferDeviceAddress-pInfo-parameter
  pInfo **must** be a valid pointer to a valid VkBufferDeviceAddressInfo structure

The VkBufferDeviceAddressInfo structure is defined as:

```
typedef struct VkBufferDeviceAddressInfo {
    VkStructureType    sType;
    const void*        pNext;
    VkBuffer           buffer;
} VkBufferDeviceAddressInfo;
```

or the equivalent

```
// Provided by VK_KHR_buffer_device_address
typedef VkBufferDeviceAddressInfo VkBufferDeviceAddressInfoKHR;
```

or the equivalent

```
// Provided by VK_EXT_buffer_device_address
typedef VkBufferDeviceAddressInfo VkBufferDeviceAddressInfoEXT;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- buffer specifies the buffer whose address is being queried.

### Valid Usage

- VUID-VkBufferDeviceAddressInfo-buffer-02600
  If buffer is non-sparse and was not created with the VK_BUFFER_CREATE_DEVICE_ADDRESS_CAPTURE_REPLAY_BIT flag, then it **must** be bound completely and contiguously to a single VkDeviceMemory object
- VUID-VkBufferDeviceAddressInfo-buffer-02601
  buffer **must** have been created with VK_BUFFER_USAGE_SHADER_DEVICE_ADDRESS_BIT

### Valid Usage (Implicit)

- VUID-VkBufferDeviceAddressInfo-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_BUFFER_DEVICE_ADDRESS_INFO`
- VUID-VkBufferDeviceAddressInfo-pNext-pNext
  `pNext` **must** be `NULL`
- VUID-VkBufferDeviceAddressInfo-buffer-parameter
  `buffer` **must** be a valid VkBuffer handle

To query a 64-bit buffer opaque capture address, call:

```
// Provided by VK_KHR_buffer_device_address
uint64_t vkGetBufferOpaqueCaptureAddressKHR(
    VkDevice                                    device,
    const VkBufferDeviceAddressInfo*            pInfo);
```

- `device` is the logical device that the buffer was created on.
- `pInfo` is a pointer to a VkBufferDeviceAddressInfo structure specifying the buffer to retrieve an address for.

The 64-bit return value is an opaque capture address of the start of `pInfo->buffer`.

If the buffer was created with a non-zero value of VkBufferOpaqueCaptureAddressCreateInfo ::`opaqueCaptureAddress` the return value **must** be the same address.

### Valid Usage

- VUID-vkGetBufferOpaqueCaptureAddress-None-03326
  The `bufferDeviceAddress` feature **must** be enabled
- VUID-vkGetBufferOpaqueCaptureAddress-device-03327
  If `device` was created with multiple physical devices, then the `bufferDeviceAddressMultiDevice` feature **must** be enabled

### Valid Usage (Implicit)

- VUID-vkGetBufferOpaqueCaptureAddress-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkGetBufferOpaqueCaptureAddress-pInfo-parameter
  `pInfo` **must** be a valid pointer to a valid VkBufferDeviceAddressInfo structure

The `VkStridedDeviceAddressRegionKHR` structure is defined as:

```
// Provided by VK_KHR_ray_tracing_pipeline
typedef struct VkStridedDeviceAddressRegionKHR {
    VkDeviceAddress    deviceAddress;
    VkDeviceSize       stride;
    VkDeviceSize       size;
} VkStridedDeviceAddressRegionKHR;
```

- `deviceAddress` is the device address (as returned by the vkGetBufferDeviceAddressKHR command) at which the region starts, or zero if the region is unused.
- `stride` is the byte stride between consecutive elements.
- `size` is the size in bytes of the region starting at `deviceAddress`.

**Valid Usage**

- VUID-VkStridedDeviceAddressRegionKHR-size-04631
  If `size` is not zero, all addresses between `deviceAddress` and `deviceAddress` + `size` - 1 **must** be in the buffer device address range of the same buffer
- VUID-VkStridedDeviceAddressRegionKHR-size-04632
  If `size` is not zero, `stride` **must** be less than or equal to the size of the buffer from which `deviceAddress` was queried

# 14.4. Descriptor Buffers

If the `descriptorBuffer` feature is enabled, an alternative way to specify descriptor sets is via buffers, rather than descriptor set objects.

## 14.4.1. Putting Descriptors in Memory

Commands are provided to retrieve descriptor data, and also to locate where in memory that data **must** be written to match the given descriptor set layout.

To determine the amount of memory needed to store all descriptors with a given layout, call:

```
// Provided by VK_EXT_descriptor_buffer
void vkGetDescriptorSetLayoutSizeEXT(
    VkDevice                                    device,
    VkDescriptorSetLayout                       layout,
    VkDeviceSize*                               pLayoutSizeInBytes);
```

- `device` is the logical device that gets the size.
- `layout` is the descriptor set layout being queried.
- `pLayoutSizeInBytes` is a pointer to VkDeviceSize where the size in bytes will be written.

The size of a descriptor set layout will be at least as large as the sum total of the size of all

descriptors in the layout, and **may** be larger. This size represents the amount of memory that will be required to store all of the descriptors for this layout in memory, when placed according to the layout's offsets as obtained by vkGetDescriptorSetLayoutBindingOffsetEXT.

If any `binding` in `layout` is `VK_DESCRIPTOR_BINDING_VARIABLE_DESCRIPTOR_COUNT_BIT`, the returned size includes space for the maximum `descriptorCount` descriptors as declared for that `binding`. To compute the required size of a descriptor set with a `VK_DESCRIPTOR_BINDING_VARIABLE_DESCRIPTOR_COUNT_BIT`:

size = offset + descriptorSize × variableDescriptorCount

where offset is obtained by vkGetDescriptorSetLayoutBindingOffsetEXT and descriptorSize is the size of the relevant descriptor as obtained from VkPhysicalDeviceDescriptorBufferPropertiesEXT, and variableDescriptorCount is the equivalent of VkDescriptorSetVariableDescriptorCountAllocateInfo::`pDescriptorCounts`. For `VK_DESCRIPTOR_TYPE_INLINE_UNIFORM_BLOCK`, variableDescriptorCount is the size in bytes for the inline uniform block, and descriptorSize is 1.

If VkPhysicalDeviceDescriptorBufferPropertiesEXT::`combinedImageSamplerDescriptorSingleArray` is `VK_FALSE` and the variable descriptor type is `VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER`, variableDescriptorCount is always considered to be the upper bound.

### Valid Usage

- VUID-vkGetDescriptorSetLayoutSizeEXT-None-08011
  The `descriptorBuffer` feature **must** be enabled
- VUID-vkGetDescriptorSetLayoutSizeEXT-layout-08012
  `layout` **must** have been created with the `VK_DESCRIPTOR_SET_LAYOUT_CREATE_DESCRIPTOR_BUFFER_BIT_EXT` flag set

### Valid Usage (Implicit)

- VUID-vkGetDescriptorSetLayoutSizeEXT-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkGetDescriptorSetLayoutSizeEXT-layout-parameter
  `layout` **must** be a valid VkDescriptorSetLayout handle
- VUID-vkGetDescriptorSetLayoutSizeEXT-pLayoutSizeInBytes-parameter
  `pLayoutSizeInBytes` **must** be a valid pointer to a VkDeviceSize value
- VUID-vkGetDescriptorSetLayoutSizeEXT-layout-parent
  `layout` **must** have been created, allocated, or retrieved from `device`

To get the offset of a binding within a descriptor set layout in memory, call:

```
// Provided by VK_EXT_descriptor_buffer
void vkGetDescriptorSetLayoutBindingOffsetEXT(
    VkDevice                                    device,
    VkDescriptorSetLayout                       layout,
    uint32_t                                    binding,
    VkDeviceSize*                               pOffset);
```

- `device` is the logical device that gets the offset.
- `layout` is the descriptor set layout being queried.
- `binding` is the binding number being queried.
- `pOffset` is a pointer to VkDeviceSize where the byte offset of the binding will be written.

Each binding in a descriptor set layout is assigned an offset in memory by the implementation. When a shader accesses a resource with that binding, it will access the bound descriptor buffer from that offset to look for its descriptor. This command provides an application with that offset, so that descriptors can be placed in the correct locations. The precise location accessed by a shader for a given descriptor is as follows:

$$\text{location} = \text{bufferAddress} + \text{setOffset} + \text{descriptorOffset} + (\text{arrayElement} \times \text{descriptorSize})$$

where bufferAddress and setOffset are the base address and offset for the identified descriptor set as specified by vkCmdBindDescriptorBuffersEXT and vkCmdSetDescriptorBufferOffsetsEXT, descriptorOffset is the offset for the binding returned by this command, arrayElement is the index into the array specified in the shader, and descriptorSize is the size of the relevant descriptor as obtained from VkPhysicalDeviceDescriptorBufferPropertiesEXT. Applications are responsible for placing valid descriptors at the expected location in order for a shader to access it. The overall offset added to bufferAddress to calculate location **must** be less than VkPhysicalDeviceDescriptorBufferPropertiesEXT::`maxSamplerDescriptorBufferRange` for samplers and VkPhysicalDeviceDescriptorBufferPropertiesEXT::`maxResourceDescriptorBufferRange` for resources.

If any `binding` in `layout` is `VK_DESCRIPTOR_BINDING_VARIABLE_DESCRIPTOR_COUNT_BIT`, that `binding` **must** have the largest offset of any `binding`.

A descriptor `binding` with type `VK_DESCRIPTOR_TYPE_MUTABLE_VALVE` **can** be used. Any potential types in VkMutableDescriptorTypeCreateInfoVALVE::`pDescriptorTypes` for `binding` share the same offset. If the size of the mutable descriptor is larger than the size of a concrete descriptor type being accessed, the padding area is ignored by the implementation.

### Valid Usage

- VUID-vkGetDescriptorSetLayoutBindingOffsetEXT-None-08013
  The `descriptorBuffer` feature **must** be enabled
- VUID-vkGetDescriptorSetLayoutBindingOffsetEXT-layout-08014
  `layout` **must** have been created with the

VK_DESCRIPTOR_SET_LAYOUT_CREATE_DESCRIPTOR_BUFFER_BIT_EXT flag set

### Valid Usage (Implicit)

- VUID-vkGetDescriptorSetLayoutBindingOffsetEXT-device-parameter
  device **must** be a valid VkDevice handle
- VUID-vkGetDescriptorSetLayoutBindingOffsetEXT-layout-parameter
  layout **must** be a valid VkDescriptorSetLayout handle
- VUID-vkGetDescriptorSetLayoutBindingOffsetEXT-pOffset-parameter
  pOffset **must** be a valid pointer to a VkDeviceSize value
- VUID-vkGetDescriptorSetLayoutBindingOffsetEXT-layout-parent
  layout **must** have been created, allocated, or retrieved from device

To get descriptor data to place in a buffer, call:

```
// Provided by VK_EXT_descriptor_buffer
void vkGetDescriptorEXT(
    VkDevice                                    device,
    const VkDescriptorGetInfoEXT*               pDescriptorInfo,
    size_t                                      dataSize,
    void*                                       pDescriptor);
```

- device is the logical device that gets the descriptor.
- pDescriptorInfo is a pointer to a VkDescriptorGetInfoEXT structure specifying the parameters of the descriptor to get.
- dataSize is the amount of the descriptor data to get in bytes.
- pDescriptor is a pointer to an application-allocated buffer where the descriptor will be written.

The size of the data for each descriptor type is determined by the value in VkPhysicalDeviceDescriptorBufferPropertiesEXT. This value also defines the stride in bytes for arrays of that descriptor type.

If the VkPhysicalDeviceDescriptorBufferPropertiesEXT::combinedImageSamplerDescriptorSingleArray property is VK_FALSE the implementation requires an array of VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER descriptors to be written into a descriptor buffer as an array of image descriptors, immediately followed by an array of sampler descriptors. Applications **must** write the first VkPhysicalDeviceDescriptorBufferPropertiesEXT::sampledImageDescriptorSize bytes of the data returned through pDescriptor to the first array, and the remaining VkPhysicalDeviceDescriptorBufferPropertiesEXT::samplerDescriptorSize bytes of the data to the second array. For variable-sized descriptor bindings of VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER descriptors, the two arrays each have a size equal to the upper bound descriptorCount of that binding.

A descriptor obtained by this command references the underlying VkImageView or VkSampler, and

these objects **must** not be destroyed before the last time a descriptor is dynamically accessed. For descriptor types which consume an address instead of an object, the underlying VkBuffer is referenced instead.

### Valid Usage

- VUID-vkGetDescriptorEXT-None-08015
  The `descriptorBuffer` feature **must** be enabled
- VUID-vkGetDescriptorEXT-dataSize-08125
  If `pDescriptorInfo->type` is not `VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER` or `pDescriptorInfo->data.pCombinedImageSampler` has an `imageView` member that was not created with a `VkSamplerYcbcrConversionInfo` structure in its `pNext` chain, `dataSize` **must** equal the size of a descriptor of type VkDescriptorGetInfoEXT::`type` determined by the value in VkPhysicalDeviceDescriptorBufferPropertiesEXT , or determined by VkPhysicalDeviceDescriptorBufferDensityMapPropertiesEXT::`combinedImageSamplerDensityMapDescriptorSize` if `pDescriptorInfo` specifies a `VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER` whose VkSampler was created with `VK_SAMPLER_CREATE_SUBSAMPLED_BIT_EXT` set
- VUID-vkGetDescriptorEXT-descriptorType-09469
  If `pDescriptorInfo->type` is `VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER` and `pDescriptorInfo->data.pCombinedImageSampler` has an `imageView` member that was created with a `VkSamplerYcbcrConversionInfo` structure in its `pNext` chain, `dataSize` **must** equal the size of VkPhysicalDeviceDescriptorBufferPropertiesEXT::`combinedImageSamplerDescriptorSize` times VkSamplerYcbcrConversionImageFormatProperties::`combinedImageSamplerDescriptorCount`
- VUID-vkGetDescriptorEXT-pDescriptorInfo-09507
  If `pDescriptorInfo->type` is `VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER` and it has a `imageView` that is VK_NULL_HANDLE then `dataSize` **must** be equal to the size of VkPhysicalDeviceDescriptorBufferPropertiesEXT::`combinedImageSamplerDescriptorSize`

### Valid Usage (Implicit)

- VUID-vkGetDescriptorEXT-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkGetDescriptorEXT-pDescriptorInfo-parameter
  `pDescriptorInfo` **must** be a valid pointer to a valid VkDescriptorGetInfoEXT structure
- VUID-vkGetDescriptorEXT-pDescriptor-parameter
  `pDescriptor` **must** be a valid pointer to an array of `dataSize` bytes
- VUID-vkGetDescriptorEXT-dataSize-arraylength
  `dataSize` **must** be greater than `0`

Information about the descriptor to get is passed in a `VkDescriptorGetInfoEXT` structure:

```
// Provided by VK_EXT_descriptor_buffer
typedef struct VkDescriptorGetInfoEXT {
    VkStructureType        sType;
    const void*            pNext;
    VkDescriptorType       type;
    VkDescriptorDataEXT    data;
} VkDescriptorGetInfoEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `type` is the type of descriptor to get.
- `data` is a structure containing the information needed to get the descriptor.

## Valid Usage

- VUID-VkDescriptorGetInfoEXT-type-08018
  `type` **must** not be `VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC`, `VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC` or `VK_DESCRIPTOR_TYPE_INLINE_UNIFORM_BLOCK`
- VUID-VkDescriptorGetInfoEXT-type-08019
  If `type` is `VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER`, the `pCombinedImageSampler->sampler` member of `data` **must** be a VkSampler created on `device`
- VUID-VkDescriptorGetInfoEXT-type-08020
  If `type` is `VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER`, the `pCombinedImageSampler->imageView` member of `data` **must** be a VkImageView created on `device`, or VK_NULL_HANDLE
- VUID-VkDescriptorGetInfoEXT-type-08021
  If `type` is `VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT`, the `pInputAttachmentImage->imageView` member of `data` **must** be a VkImageView created on `device`
- VUID-VkDescriptorGetInfoEXT-type-08022
  If `type` is `VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE`, and if `pSampledImage` is not `NULL`, the `pSampledImage->imageView` member of `data` **must** be a VkImageView created on `device`, or VK_NULL_HANDLE
- VUID-VkDescriptorGetInfoEXT-type-08023
  If `type` is `VK_DESCRIPTOR_TYPE_STORAGE_IMAGE`, and if `pStorageImage` is not `NULL`, the `pStorageImage->imageView` member of `data` **must** be a VkImageView created on `device`, or VK_NULL_HANDLE
- VUID-VkDescriptorGetInfoEXT-type-08024
  If `type` is `VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER`, `pUniformTexelBuffer` is not `NULL` and `pUniformTexelBuffer->address` is not zero, `pUniformTexelBuffer->address` **must** be an address within a VkBuffer created on `device`
- VUID-VkDescriptorGetInfoEXT-type-08025
  If `type` is `VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER`, `pStorageTexelBuffer` is not `NULL` and `pStorageTexelBuffer->address` is not zero, `pStorageTexelBuffer->address` **must** be an

address within a VkBuffer created on device

- VUID-VkDescriptorGetInfoEXT-type-08026
  If type is VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER, pUniformBuffer is not NULL and pUniformBuffer->address is not zero, pUniformBuffer->address **must** be an address within a VkBuffer created on device
- VUID-VkDescriptorGetInfoEXT-type-08027
  If type is VK_DESCRIPTOR_TYPE_STORAGE_BUFFER, pStorageBuffer is not NULL and pStorageBuffer->address is not zero, pStorageBuffer->address **must** be an address within a VkBuffer created on device
- VUID-VkDescriptorGetInfoEXT-type-09427
  If type is VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER, pUniformBuffer is not NULL , the number of texel buffer elements given by (⌊pUniformBuffer->range / (texel block size)⌋ × (texels per block)) where texel block size and texels per block are as defined in the Compatible Formats table for pUniformBuffer->format, **must** be less than or equal to VkPhysicalDeviceLimits::maxTexelBufferElements
- VUID-VkDescriptorGetInfoEXT-type-09428
  If type is VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER, pStorageBuffer is not NULL , the number of texel buffer elements given by (⌊pStorageBuffer->range / (texel block size)⌋ × (texels per block)) where texel block size and texels per block are as defined in the Compatible Formats table for pStorageBuffer->format, **must** be less than or equal to VkPhysicalDeviceLimits::maxTexelBufferElements
- VUID-VkDescriptorGetInfoEXT-type-08028
  If type is VK_DESCRIPTOR_TYPE_ACCELERATION_STRUCTURE_KHR and accelerationStructure is not 0, accelerationStructure **must** contain the address of a VkAccelerationStructureKHR created on device
- VUID-VkDescriptorGetInfoEXT-type-08029
  If type is VK_DESCRIPTOR_TYPE_ACCELERATION_STRUCTURE_NV and accelerationStructure is not 0, accelerationStructure **must** contain the handle of a VkAccelerationStructureNV created on device, returned by vkGetAccelerationStructureHandleNV

## Valid Usage (Implicit)

- VUID-VkDescriptorGetInfoEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_DESCRIPTOR_GET_INFO_EXT
- VUID-VkDescriptorGetInfoEXT-pNext-pNext
  pNext **must** be NULL
- VUID-VkDescriptorGetInfoEXT-type-parameter
  type **must** be a valid VkDescriptorType value
- VUID-VkDescriptorGetInfoEXT-pSampler-parameter
  If type is VK_DESCRIPTOR_TYPE_SAMPLER, the pSampler member of data **must** be a valid pointer to a valid VkSampler handle
- VUID-VkDescriptorGetInfoEXT-pCombinedImageSampler-parameter
  If type is VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER, the pCombinedImageSampler member

of `data` **must** be a valid pointer to a valid VkDescriptorImageInfo structure

- VUID-VkDescriptorGetInfoEXT-pInputAttachmentImage-parameter
  If `type` is `VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT`, the `pInputAttachmentImage` member of `data` **must** be a valid pointer to a valid VkDescriptorImageInfo structure
- VUID-VkDescriptorGetInfoEXT-pSampledImage-parameter
  If `type` is `VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE`, and if `pSampledImage` is not `NULL`, the `pSampledImage` member of `data` **must** be a valid pointer to a valid VkDescriptorImageInfo structure
- VUID-VkDescriptorGetInfoEXT-pStorageImage-parameter
  If `type` is `VK_DESCRIPTOR_TYPE_STORAGE_IMAGE`, and if `pStorageImage` is not `NULL`, the `pStorageImage` member of `data` **must** be a valid pointer to a valid VkDescriptorImageInfo structure
- VUID-VkDescriptorGetInfoEXT-pUniformTexelBuffer-parameter
  If `type` is `VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER`, and if `pUniformTexelBuffer` is not `NULL`, the `pUniformTexelBuffer` member of `data` **must** be a valid pointer to a valid VkDescriptorAddressInfoEXT structure
- VUID-VkDescriptorGetInfoEXT-pStorageTexelBuffer-parameter
  If `type` is `VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER`, and if `pStorageTexelBuffer` is not `NULL`, the `pStorageTexelBuffer` member of `data` **must** be a valid pointer to a valid VkDescriptorAddressInfoEXT structure
- VUID-VkDescriptorGetInfoEXT-pUniformBuffer-parameter
  If `type` is `VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER`, and if `pUniformBuffer` is not `NULL`, the `pUniformBuffer` member of `data` **must** be a valid pointer to a valid VkDescriptorAddressInfoEXT structure
- VUID-VkDescriptorGetInfoEXT-pStorageBuffer-parameter
  If `type` is `VK_DESCRIPTOR_TYPE_STORAGE_BUFFER`, and if `pStorageBuffer` is not `NULL`, the `pStorageBuffer` member of `data` **must** be a valid pointer to a valid VkDescriptorAddressInfoEXT structure

Data describing the descriptor is passed in a `VkDescriptorDataEXT` structure:

```
// Provided by VK_EXT_descriptor_buffer
typedef union VkDescriptorDataEXT {
    const VkSampler*                    pSampler;
    const VkDescriptorImageInfo*        pCombinedImageSampler;
    const VkDescriptorImageInfo*        pInputAttachmentImage;
    const VkDescriptorImageInfo*        pSampledImage;
    const VkDescriptorImageInfo*        pStorageImage;
    const VkDescriptorAddressInfoEXT*   pUniformTexelBuffer;
    const VkDescriptorAddressInfoEXT*   pStorageTexelBuffer;
    const VkDescriptorAddressInfoEXT*   pUniformBuffer;
    const VkDescriptorAddressInfoEXT*   pStorageBuffer;
    VkDeviceAddress                     accelerationStructure;
} VkDescriptorDataEXT;
```

- pSampler is a pointer to a VkSampler handle specifying the parameters of a VK_DESCRIPTOR_TYPE_SAMPLER descriptor.
- pCombinedImageSampler is a pointer to a VkDescriptorImageInfo structure specifying the parameters of a VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER descriptor.
- pInputAttachmentImage is a pointer to a VkDescriptorImageInfo structure specifying the parameters of a VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT descriptor.
- pSampledImage is a pointer to a VkDescriptorImageInfo structure specifying the parameters of a VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE descriptor.
- pStorageImage is a pointer to a VkDescriptorImageInfo structure specifying the parameters of a VK_DESCRIPTOR_TYPE_STORAGE_IMAGE descriptor.
- pUniformTexelBuffer is a pointer to a VkDescriptorAddressInfoEXT structure specifying the parameters of a VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER descriptor.
- pStorageTexelBuffer is a pointer to a VkDescriptorAddressInfoEXT structure specifying the parameters of a VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER descriptor.
- pUniformBuffer is a pointer to a VkDescriptorAddressInfoEXT structure specifying the parameters of a VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER descriptor.
- pStorageBuffer is a pointer to a VkDescriptorAddressInfoEXT structure specifying the parameters of a VK_DESCRIPTOR_TYPE_STORAGE_BUFFER descriptor.
- accelerationStructure is the address of a VkAccelerationStructureKHR specifying the parameters of a VK_DESCRIPTOR_TYPE_ACCELERATION_STRUCTURE_KHR descriptor , or a VkAccelerationStructureNV handle specifying the parameters of a VK_DESCRIPTOR_TYPE_ACCELERATION_STRUCTURE_NV descriptor.

If the nullDescriptor feature is enabled, pSampledImage, pStorageImage, pUniformTexelBuffer, pStorageTexelBuffer, pUniformBuffer, and pStorageBuffer **can** each be NULL. Loads from a null descriptor return zero values and stores and atomics to a null descriptor are discarded.

If the nullDescriptor feature is enabled, accelerationStructure **can** be 0. A null acceleration structure descriptor results in the miss shader being invoked.

### Valid Usage

- VUID-VkDescriptorDataEXT-type-08030
  If VkDescriptorGetInfoEXT:type is VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER, and pUniformBuffer->address is the address of a non-sparse buffer, then that buffer **must** be bound completely and contiguously to a single VkDeviceMemory object
- VUID-VkDescriptorDataEXT-type-08031
  If VkDescriptorGetInfoEXT:type is VK_DESCRIPTOR_TYPE_STORAGE_BUFFER, and pStorageBuffer->address is the address of a non-sparse buffer, then that buffer **must** be bound completely and contiguously to a single VkDeviceMemory object
- VUID-VkDescriptorDataEXT-type-08032
  If VkDescriptorGetInfoEXT:type is VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER, and pUniformTexelBuffer->address is the address of a non-sparse buffer, then that buffer **must** be bound completely and contiguously to a single VkDeviceMemory object

- VUID-VkDescriptorDataEXT-type-08033
  If VkDescriptorGetInfoEXT:type is VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER, and pStorageTexelBuffer->address is the address of a non-sparse buffer, then that buffer **must** be bound completely and contiguously to a single VkDeviceMemory object
- VUID-VkDescriptorDataEXT-type-08034
  If VkDescriptorGetInfoEXT:type is VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER, and the nullDescriptor feature is not enabled, pCombinedImageSampler->imageView **must** not be VK_NULL_HANDLE
- VUID-VkDescriptorDataEXT-type-08035
  If VkDescriptorGetInfoEXT:type is VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE, and the nullDescriptor feature is not enabled, pSampledImage **must** not be NULL and pSampledImage->imageView **must** not be VK_NULL_HANDLE
- VUID-VkDescriptorDataEXT-type-08036
  If VkDescriptorGetInfoEXT:type is VK_DESCRIPTOR_TYPE_STORAGE_IMAGE, and the nullDescriptor feature is not enabled, pStorageImage **must** not be NULL and pStorageImage->imageView **must** not be VK_NULL_HANDLE
- VUID-VkDescriptorDataEXT-type-08037
  If VkDescriptorGetInfoEXT:type is VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER, and the nullDescriptor feature is not enabled, pUniformTexelBuffer **must** not be NULL
- VUID-VkDescriptorDataEXT-type-08038
  If VkDescriptorGetInfoEXT:type is VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER, and the nullDescriptor feature is not enabled, pStorageTexelBuffer **must** not be NULL
- VUID-VkDescriptorDataEXT-type-08039
  If VkDescriptorGetInfoEXT:type is VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER, and the nullDescriptor feature is not enabled, pUniformBuffer **must** not be NULL
- VUID-VkDescriptorDataEXT-type-08040
  If VkDescriptorGetInfoEXT:type is VK_DESCRIPTOR_TYPE_STORAGE_BUFFER, and the nullDescriptor feature is not enabled, pStorageBuffer **must** not be NULL
- VUID-VkDescriptorDataEXT-type-08041
  If VkDescriptorGetInfoEXT:type is VK_DESCRIPTOR_TYPE_ACCELERATION_STRUCTURE_KHR, and the nullDescriptor feature is not enabled, accelerationStructure **must** not be 0
- VUID-VkDescriptorDataEXT-type-08042
  If VkDescriptorGetInfoEXT:type is VK_DESCRIPTOR_TYPE_ACCELERATION_STRUCTURE_NV, and the nullDescriptor feature is not enabled, accelerationStructure **must** not be 0

Data describing a VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER, VK_DESCRIPTOR_TYPE_STORAGE_BUFFER, VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER, or VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER descriptor is passed in a VkDescriptorAddressInfoEXT structure:

```
// Provided by VK_EXT_descriptor_buffer
typedef struct VkDescriptorAddressInfoEXT {
    VkStructureType    sType;
    void*              pNext;
    VkDeviceAddress    address;
    VkDeviceSize       range;
    VkFormat           format;
} VkDescriptorAddressInfoEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `address` is either `0` or a device address at an offset in a buffer, where the base address can be queried from vkGetBufferDeviceAddressKHR.
- `range` is the size in bytes of the buffer or buffer view used by the descriptor.
- `format` is the format of the data elements in the buffer view and is ignored for buffers.

## Valid Usage

- VUID-VkDescriptorAddressInfoEXT-None-09508
  If `address` is not zero, and the descriptor is of type `VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER` or `VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER`, then `format` **must** not be `VK_FORMAT_UNDEFINED`
- VUID-VkDescriptorAddressInfoEXT-address-08043
  If the `nullDescriptor` feature is not enabled, `address` **must** not be zero
- VUID-VkDescriptorAddressInfoEXT-nullDescriptor-08938
  If `address` is zero, `range` **must** be `VK_WHOLE_SIZE`
- VUID-VkDescriptorAddressInfoEXT-nullDescriptor-08939
  If `address` is not zero, `range` **must** not be `VK_WHOLE_SIZE`
- VUID-VkDescriptorAddressInfoEXT-None-08044
  If `address` is not zero, `address` **must** be a valid device address at an offset within a VkBuffer
- VUID-VkDescriptorAddressInfoEXT-range-08045
  `range` **must** be less than or equal to the size of the buffer containing `address` minus the offset of `address` from the base address of the buffer
- VUID-VkDescriptorAddressInfoEXT-range-08940
  `range` **must** not be zero

## Valid Usage (Implicit)

- VUID-VkDescriptorAddressInfoEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_DESCRIPTOR_ADDRESS_INFO_EXT`

- VUID-VkDescriptorAddressInfoEXT-pNext-pNext
  `pNext` **must** be `NULL`
- VUID-VkDescriptorAddressInfoEXT-format-parameter
  `format` **must** be a valid VkFormat value

If the `nullDescriptor` feature is enabled, `address` **can** be zero. Loads from a null descriptor return zero values and stores and atomics to a null descriptor are discarded.

Immutable samplers specified in a descriptor set layout through `pImmutableSamplers` **must** be provided by applications when obtaining descriptor data. Immutable samplers written in a descriptor buffer **must** have identical parameters to the immutable samplers in the descriptor set layout that consumes the sampler.

| | |
|---|---|
| **NOTE** | If the descriptor set layout was created with `VK_DESCRIPTOR_SET_LAYOUT_CREATE_EMBEDDED_IMMUTABLE_SAMPLERS_BIT_EXT`, there is no buffer backing for the immutable sampler, so this requirement does not exist. The implementation handles allocation of these descriptors internally. |
| **NOTE** | As descriptors are now in regular memory, drivers cannot hide copies of immutable samplers that end up in descriptor sets from the application. As such, applications are required to provide these samplers as if they were not provided immutably. |

### 14.4.2. Binding Descriptor Buffers

Descriptor buffers have their own separate binding point on the command buffer, with buffers bound using vkCmdBindDescriptorBuffersEXT. vkCmdSetDescriptorBufferOffsetsEXT assigns pairs of buffer binding indices and buffer offsets to the same binding point on the command buffer as vkCmdBindDescriptorSets, allowing subsequent bound pipeline commands to use the specified descriptor buffers. Bindings applied via vkCmdBindDescriptorSets **cannot** exist simultaneously with those applied via calls to vkCmdSetDescriptorBufferOffsetsEXT or vkCmdBindDescriptorBufferEmbeddedSamplersEXT, as calls to vkCmdSetDescriptorBufferOffsetsEXT or vkCmdBindDescriptorBufferEmbeddedSamplersEXT invalidate any bindings by previous calls to vkCmdBindDescriptorSets and vice-versa.

To bind descriptor buffers to a command buffer, call:

```
// Provided by VK_EXT_descriptor_buffer
void vkCmdBindDescriptorBuffersEXT(
    VkCommandBuffer                             commandBuffer,
    uint32_t                                    bufferCount,
    const VkDescriptorBufferBindingInfoEXT*     pBindingInfos);
```

- `commandBuffer` is the command buffer that the descriptor buffers will be bound to.
- `bufferCount` is the number of elements in the `pBindingInfos` array.
- `pBindingInfos` is a pointer to an array of VkDescriptorBufferBindingInfoEXT structures.

vkCmdBindDescriptorBuffersEXT causes any offsets previously set by vkCmdSetDescriptorBufferOffsetsEXT that use the bindings numbered [0.. bufferCount-1] to be no longer valid for subsequent bound pipeline commands. Any previously bound buffers at binding points greater than or equal to bufferCount are unbound.

## Valid Usage

- VUID-vkCmdBindDescriptorBuffersEXT-None-08047
  The descriptorBuffer feature **must** be enabled
- VUID-vkCmdBindDescriptorBuffersEXT-maxSamplerDescriptorBufferBindings-08048
  There **must** be no more than VkPhysicalDeviceDescriptorBufferPropertiesEXT::maxSamplerDescriptorBufferBindings descriptor buffers containing sampler descriptor data bound
- VUID-vkCmdBindDescriptorBuffersEXT-maxResourceDescriptorBufferBindings-08049
  There **must** be no more than VkPhysicalDeviceDescriptorBufferPropertiesEXT::maxResourceDescriptorBufferBindings descriptor buffers containing resource descriptor data bound
- VUID-vkCmdBindDescriptorBuffersEXT-None-08050
  There **must** be no more than 1 descriptor buffer bound that was created with the VK_BUFFER_USAGE_PUSH_DESCRIPTORS_DESCRIPTOR_BUFFER_BIT_EXT bit set
- VUID-vkCmdBindDescriptorBuffersEXT-bufferCount-08051
  bufferCount **must** be less than or equal to VkPhysicalDeviceDescriptorBufferPropertiesEXT::maxDescriptorBufferBindings
- VUID-vkCmdBindDescriptorBuffersEXT-pBindingInfos-08052
  For any element of pBindingInfos, if the buffer from which address was queried is non-sparse then it **must** be bound completely and contiguously to a single VkDeviceMemory object
- VUID-vkCmdBindDescriptorBuffersEXT-pBindingInfos-08053
  For any element of pBindingInfos, the buffer from which address was queried **must** have been created with the VK_BUFFER_USAGE_SAMPLER_DESCRIPTOR_BUFFER_BIT_EXT bit set if it contains sampler descriptor data
- VUID-vkCmdBindDescriptorBuffersEXT-pBindingInfos-08054
  For any element of pBindingInfos, the buffer from which address was queried **must** have been created with the VK_BUFFER_USAGE_RESOURCE_DESCRIPTOR_BUFFER_BIT_EXT bit set if it contains resource descriptor data
- VUID-vkCmdBindDescriptorBuffersEXT-pBindingInfos-08055
  For any element of pBindingInfos, usage **must** match the buffer from which address was queried

## Valid Usage (Implicit)

- VUID-vkCmdBindDescriptorBuffersEXT-commandBuffer-parameter
  commandBuffer **must** be a valid VkCommandBuffer handle

- VUID-vkCmdBindDescriptorBuffersEXT-pBindingInfos-parameter
  `pBindingInfos` **must** be a valid pointer to an array of `bufferCount` valid VkDescriptorBufferBindingInfoEXT structures
- VUID-vkCmdBindDescriptorBuffersEXT-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdBindDescriptorBuffersEXT-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics, or compute operations
- VUID-vkCmdBindDescriptorBuffersEXT-videocoding
  This command **must** only be called outside of a video coding scope
- VUID-vkCmdBindDescriptorBuffersEXT-bufferCount-arraylength
  `bufferCount` **must** be greater than `0`

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics<br>Compute | State |

Data describing a descriptor buffer binding is passed in a `VkDescriptorBufferBindingInfoEXT` structure:

```
// Provided by VK_EXT_descriptor_buffer
typedef struct VkDescriptorBufferBindingInfoEXT {
    VkStructureType       sType;
    const void*           pNext;
    VkDeviceAddress       address;
    VkBufferUsageFlags    usage;
} VkDescriptorBufferBindingInfoEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `address` is a VkDeviceAddress specifying the device address defining the descriptor buffer to be bound.

- usage is a bitmask of VkBufferUsageFlagBits specifying the VkBufferCreateInfo::usage for the buffer from which address was queried.

If the pNext chain includes a VkBufferUsageFlags2CreateInfoKHR structure, VkBufferUsageFlags2CreateInfoKHR::usage from that structure is used instead of usage from this structure.

## Valid Usage

- VUID-VkDescriptorBufferBindingInfoEXT-None-09499
  If the pNext chain does not include a VkBufferUsageFlags2CreateInfoKHR structure, usage **must** be a valid combination of VkBufferUsageFlagBits values
- VUID-VkDescriptorBufferBindingInfoEXT-None-09500
  If the pNext chain does not include a VkBufferUsageFlags2CreateInfoKHR structure, usage **must** not be 0
- VUID-VkDescriptorBufferBindingInfoEXT-bufferlessPushDescriptors-08056
  If VkPhysicalDeviceDescriptorBufferPropertiesEXT::bufferlessPushDescriptors is VK_FALSE, and usage contains VK_BUFFER_USAGE_PUSH_DESCRIPTORS_DESCRIPTOR_BUFFER_BIT_EXT, then the pNext chain **must** include a VkDescriptorBufferBindingPushDescriptorBufferHandleEXT structure
- VUID-VkDescriptorBufferBindingInfoEXT-address-08057
  address **must** be aligned to VkPhysicalDeviceDescriptorBufferPropertiesEXT::descriptorBufferOffsetAlignment
- VUID-VkDescriptorBufferBindingInfoEXT-usage-08122
  If usage includes VK_BUFFER_USAGE_SAMPLER_DESCRIPTOR_BUFFER_BIT_EXT, address **must** be an address within a valid buffer that was created with VK_BUFFER_USAGE_SAMPLER_DESCRIPTOR_BUFFER_BIT_EXT
- VUID-VkDescriptorBufferBindingInfoEXT-usage-08123
  If usage includes VK_BUFFER_USAGE_RESOURCE_DESCRIPTOR_BUFFER_BIT_EXT, address **must** be an address within a valid buffer that was created with VK_BUFFER_USAGE_RESOURCE_DESCRIPTOR_BUFFER_BIT_EXT
- VUID-VkDescriptorBufferBindingInfoEXT-usage-08124
  If usage includes VK_BUFFER_USAGE_PUSH_DESCRIPTORS_DESCRIPTOR_BUFFER_BIT_EXT, address **must** be an address within a valid buffer that was created with VK_BUFFER_USAGE_PUSH_DESCRIPTORS_DESCRIPTOR_BUFFER_BIT_EXT

## Valid Usage (Implicit)

- VUID-VkDescriptorBufferBindingInfoEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_DESCRIPTOR_BUFFER_BINDING_INFO_EXT
- VUID-VkDescriptorBufferBindingInfoEXT-pNext-pNext
  Each pNext member of any structure (including this one) in the pNext chain **must** be either NULL or a pointer to a valid instance of VkBufferUsageFlags2CreateInfoKHR or VkDescriptorBufferBindingPushDescriptorBufferHandleEXT

- VUID-VkDescriptorBufferBindingInfoEXT-sType-unique
  The `sType` value of each struct in the `pNext` chain **must** be unique

When the `VkPhysicalDeviceDescriptorBufferPropertiesEXT::bufferlessPushDescriptors` property is `VK_FALSE`, the `VkBuffer` handle of the buffer for push descriptors is passed in a `VkDescriptorBufferBindingPushDescriptorBufferHandleEXT` structure:

```
// Provided by VK_EXT_descriptor_buffer
typedef struct VkDescriptorBufferBindingPushDescriptorBufferHandleEXT {
    VkStructureType    sType;
    const void*        pNext;
    VkBuffer           buffer;
} VkDescriptorBufferBindingPushDescriptorBufferHandleEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `buffer` is the `VkBuffer` handle of the buffer for push descriptors.

### Valid Usage

- VUID-VkDescriptorBufferBindingPushDescriptorBufferHandleEXT-bufferlessPushDescriptors-08059
  `VkPhysicalDeviceDescriptorBufferPropertiesEXT::bufferlessPushDescriptors` **must** be `VK_FALSE`

### Valid Usage (Implicit)

- VUID-VkDescriptorBufferBindingPushDescriptorBufferHandleEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_DESCRIPTOR_BUFFER_BINDING_PUSH_DESCRIPTOR_BUFFER_HANDLE_EXT`
- VUID-VkDescriptorBufferBindingPushDescriptorBufferHandleEXT-buffer-parameter
  `buffer` **must** be a valid VkBuffer handle

To set descriptor buffer offsets in a command buffer, call:

```
// Provided by VK_EXT_descriptor_buffer
void vkCmdSetDescriptorBufferOffsetsEXT(
    VkCommandBuffer                             commandBuffer,
    VkPipelineBindPoint                         pipelineBindPoint,
    VkPipelineLayout                            layout,
    uint32_t                                    firstSet,
    uint32_t                                    setCount,
    const uint32_t*                             pBufferIndices,
    const VkDeviceSize*                         pOffsets);
```

- `commandBuffer` is the command buffer in which the descriptor buffer offsets will be set.
- `pipelineBindPoint` is a VkPipelineBindPoint indicating the type of the pipeline that will use the descriptors.
- `layout` is a VkPipelineLayout object used to program the bindings.
- `firstSet` is the number of the first set to be bound.
- `setCount` is the number of elements in the `pBufferIndices` and `pOffsets` arrays.
- `pBufferIndices` is a pointer to an array of indices into the descriptor buffer binding points set by vkCmdBindDescriptorBuffersEXT.
- `pOffsets` is a pointer to an array of VkDeviceSize offsets to apply to the bound descriptor buffers.

`vkCmdSetDescriptorBufferOffsetsEXT` binds `setCount` pairs of descriptor buffers, specified by indices into the binding points bound using vkCmdBindDescriptorBuffersEXT, and buffer offsets to set numbers [`firstSet`..`firstSet`+`descriptorSetCount`-1] for subsequent bound pipeline commands set by `pipelineBindPoint`. Set [`firstSet` + i] is bound to the descriptor buffer at binding `pBufferIndices`[i] at an offset of `pOffsets`[i]. Any bindings that were previously applied via these sets, or calls to vkCmdBindDescriptorSets, are no longer valid. Other sets will also be invalidated upon calling this command if `layout` differs from the pipeline layout used to bind those other sets, as described in Pipeline Layout Compatibility.

After binding descriptors, applications **can** modify descriptor memory either by performing writes on the host or with device commands. When descriptor memory is updated with device commands, visibility for the shader stage accessing a descriptor is ensured with the `VK_ACCESS_2_DESCRIPTOR_BUFFER_READ_BIT_EXT` access flag. Implementations **must** not access resources referenced by these descriptors unless they are dynamically accessed by shaders. Descriptors bound with this call **can** be undefined if they are not dynamically accessed by shaders.

Implementations **may** read descriptor data for any statically accessed descriptor if the `binding` in `layout` is not declared with the `VK_DESCRIPTOR_BINDING_VARIABLE_DESCRIPTOR_COUNT_BIT` flag. If the `binding` in `layout` is declared with `VK_DESCRIPTOR_BINDING_VARIABLE_DESCRIPTOR_COUNT_BIT`, implementations **must** not read descriptor data that is not dynamically accessed.

Applications **must** ensure that any descriptor which the implementation **may** read **must** be in-bounds of the underlying descriptor buffer binding.

**NOTE** Applications can freely decide how large a variable descriptor buffer binding is, so

it may not be safe to read such descriptor payloads statically. The intention of these rules is to allow implementations to speculatively prefetch descriptor payloads where feasible.

Dynamically accessing a resource through descriptor data from an unbound region of a sparse partially-resident buffer will result in invalid descriptor data being read, and therefore undefined behavior.

**NOTE** For descriptors written by the host, visibility is implied through the automatic visibility operation on queue submit, and there is no need to consider `VK_ACCESS_2_DESCRIPTOR_BUFFER_READ_BIT`. Explicit synchronization for descriptors is only required when descriptors are updated on the device.

**NOTE** The requirements above imply that all descriptor bindings have been defined with the equivalent of `VK_DESCRIPTOR_BINDING_UPDATE_AFTER_BIND_BIT`, `VK_DESCRIPTOR_BINDING_UPDATE_UNUSED_WHILE_PENDING_BIT` and `VK_DESCRIPTOR_BINDING_PARTIALLY_BOUND_BIT`, but enabling those features is not required to get this behavior.

## Valid Usage

- VUID-vkCmdSetDescriptorBufferOffsetsEXT-pOffsets-08061
  The offsets in `pOffsets` **must** be aligned to VkPhysicalDeviceDescriptorBufferPropertiesEXT::`descriptorBufferOffsetAlignment`
- VUID-vkCmdSetDescriptorBufferOffsetsEXT-pOffsets-08063
  The offsets in `pOffsets` **must** be small enough such that any descriptor binding referenced by `layout` without the `VK_DESCRIPTOR_BINDING_VARIABLE_DESCRIPTOR_COUNT_BIT` flag computes a valid address inside the underlying VkBuffer
- VUID-vkCmdSetDescriptorBufferOffsetsEXT-pOffsets-08126
  The offsets in `pOffsets` **must** be small enough such that any location accessed by a shader as a sampler descriptor **must** be within VkPhysicalDeviceDescriptorBufferPropertiesEXT::`maxSamplerDescriptorBufferRange` of the sampler descriptor buffer binding
- VUID-vkCmdSetDescriptorBufferOffsetsEXT-pOffsets-08127
  The offsets in `pOffsets` **must** be small enough such that any location accessed by a shader as a resource descriptor **must** be within VkPhysicalDeviceDescriptorBufferPropertiesEXT::`maxResourceDescriptorBufferRange` of the resource descriptor buffer binding
- VUID-vkCmdSetDescriptorBufferOffsetsEXT-pBufferIndices-08064
  Each element of `pBufferIndices` **must** be less than VkPhysicalDeviceDescriptorBufferPropertiesEXT::`maxDescriptorBufferBindings`
- VUID-vkCmdSetDescriptorBufferOffsetsEXT-pBufferIndices-08065
  Each element of `pBufferIndices` **must** reference a valid descriptor buffer binding set by a previous call to vkCmdBindDescriptorBuffersEXT in `commandBuffer`
- VUID-vkCmdSetDescriptorBufferOffsetsEXT-firstSet-08066

The sum of `firstSet` and `setCount` **must** be less than or equal to VkPipelineLayoutCreateInfo::`setLayoutCount` provided when `layout` was created

- VUID-vkCmdSetDescriptorBufferOffsetsEXT-firstSet-09006
  The VkDescriptorSetLayout for each set from `firstSet` to `firstSet` + `setCount` when `layout` was created **must** have been created with the `VK_DESCRIPTOR_SET_LAYOUT_CREATE_DESCRIPTOR_BUFFER_BIT_EXT` bit set
- VUID-vkCmdSetDescriptorBufferOffsetsEXT-None-08060
  The `descriptorBuffer` feature **must** be enabled
- VUID-vkCmdSetDescriptorBufferOffsetsEXT-pipelineBindPoint-08067
  `pipelineBindPoint` **must** be supported by the `commandBuffer`'s parent `VkCommandPool`'s queue family

## Valid Usage (Implicit)

- VUID-vkCmdSetDescriptorBufferOffsetsEXT-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdSetDescriptorBufferOffsetsEXT-pipelineBindPoint-parameter
  `pipelineBindPoint` **must** be a valid VkPipelineBindPoint value
- VUID-vkCmdSetDescriptorBufferOffsetsEXT-layout-parameter
  `layout` **must** be a valid VkPipelineLayout handle
- VUID-vkCmdSetDescriptorBufferOffsetsEXT-pBufferIndices-parameter
  `pBufferIndices` **must** be a valid pointer to an array of `setCount` `uint32_t` values
- VUID-vkCmdSetDescriptorBufferOffsetsEXT-pOffsets-parameter
  `pOffsets` **must** be a valid pointer to an array of `setCount` VkDeviceSize values
- VUID-vkCmdSetDescriptorBufferOffsetsEXT-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdSetDescriptorBufferOffsetsEXT-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics, or compute operations
- VUID-vkCmdSetDescriptorBufferOffsetsEXT-videocoding
  This command **must** only be called outside of a video coding scope
- VUID-vkCmdSetDescriptorBufferOffsetsEXT-setCount-arraylength
  `setCount` **must** be greater than `0`
- VUID-vkCmdSetDescriptorBufferOffsetsEXT-commonparent
  Both of `commandBuffer`, and `layout` **must** have been created, allocated, or retrieved from the same VkDevice

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized

- Host access to the VkCommandPool that commandBuffer was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics<br>Compute | State |

Alternatively, to set descriptor buffer offsets in a command buffer, call:

```
// Provided by VK_KHR_maintenance6 with VK_EXT_descriptor_buffer
void vkCmdSetDescriptorBufferOffsets2EXT(
    VkCommandBuffer                             commandBuffer,
    const VkSetDescriptorBufferOffsetsInfoEXT*  pSetDescriptorBufferOffsetsInfo);
```

- commandBuffer is the command buffer in which the descriptor buffer offsets will be set.
- pSetDescriptorBufferOffsetsInfo is a pointer to a VkSetDescriptorBufferOffsetsInfoEXT structure.

## Valid Usage

- VUID-vkCmdSetDescriptorBufferOffsets2EXT-descriptorBuffer-09470
  The descriptorBuffer feature **must** be enabled
- VUID-vkCmdSetDescriptorBufferOffsets2EXT-pSetDescriptorBufferOffsetsInfo-09471
  Each bit in pSetDescriptorBufferOffsetsInfo->stageFlags **must** be a stage supported by the commandBuffer's parent VkCommandPool's queue family

## Valid Usage (Implicit)

- VUID-vkCmdSetDescriptorBufferOffsets2EXT-commandBuffer-parameter
  commandBuffer **must** be a valid VkCommandBuffer handle
- VUID-vkCmdSetDescriptorBufferOffsets2EXT-pSetDescriptorBufferOffsetsInfo-parameter
  pSetDescriptorBufferOffsetsInfo **must** be a valid pointer to a valid VkSetDescriptorBufferOffsetsInfoEXT structure
- VUID-vkCmdSetDescriptorBufferOffsets2EXT-commandBuffer-recording
  commandBuffer **must** be in the recording state
- VUID-vkCmdSetDescriptorBufferOffsets2EXT-commandBuffer-cmdpool
  The VkCommandPool that commandBuffer was allocated from **must** support graphics, or compute operations
- VUID-vkCmdSetDescriptorBufferOffsets2EXT-videocoding

This command **must** only be called outside of a video coding scope

### Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

### Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics<br>Compute | State |

The `VkSetDescriptorBufferOffsetsInfoEXT` structure is defined as:

```
// Provided by VK_KHR_maintenance6 with VK_EXT_descriptor_buffer
typedef struct VkSetDescriptorBufferOffsetsInfoEXT {
    VkStructureType        sType;
    const void*            pNext;
    VkShaderStageFlags     stageFlags;
    VkPipelineLayout       layout;
    uint32_t               firstSet;
    uint32_t               setCount;
    const uint32_t*        pBufferIndices;
    const VkDeviceSize*    pOffsets;
} VkSetDescriptorBufferOffsetsInfoEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `stageFlags` is a bitmask of VkShaderStageFlagBits specifying the shader stages the descriptor sets will be bound to
- `layout` is a VkPipelineLayout object used to program the bindings. If the `dynamicPipelineLayout` feature is enabled, `layout` **can** be VK_NULL_HANDLE and the layout **must** be specified by chaining VkPipelineLayoutCreateInfo structure off the `pNext`
- `firstSet` is the number of the first set to be bound.
- `setCount` is the number of elements in the `pBufferIndices` and `pOffsets` arrays.
- `pBufferIndices` is a pointer to an array of indices into the descriptor buffer binding points set by vkCmdBindDescriptorBuffersEXT.
- `pOffsets` is a pointer to an array of VkDeviceSize offsets to apply to the bound descriptor

buffers.

If stageFlags specifies a subset of all stages corresponding to one or more pipeline bind points, the binding operation still affects all stages corresponding to the given pipeline bind point(s) as if the equivalent original version of this command had been called with the same parameters. For example, specifying a stageFlags value of VK_SHADER_STAGE_VERTEX_BIT | VK_SHADER_STAGE_FRAGMENT_BIT | VK_SHADER_STAGE_COMPUTE_BIT is equivalent to calling the original version of this command once with VK_PIPELINE_BIND_POINT_GRAPHICS and once with VK_PIPELINE_BIND_POINT_COMPUTE.

## Valid Usage

- VUID-VkSetDescriptorBufferOffsetsInfoEXT-pOffsets-08061
  The offsets in pOffsets **must** be aligned to VkPhysicalDeviceDescriptorBufferPropertiesEXT::descriptorBufferOffsetAlignment
- VUID-VkSetDescriptorBufferOffsetsInfoEXT-pOffsets-08063
  The offsets in pOffsets **must** be small enough such that any descriptor binding referenced by layout without the VK_DESCRIPTOR_BINDING_VARIABLE_DESCRIPTOR_COUNT_BIT flag computes a valid address inside the underlying VkBuffer
- VUID-VkSetDescriptorBufferOffsetsInfoEXT-pOffsets-08126
  The offsets in pOffsets **must** be small enough such that any location accessed by a shader as a sampler descriptor **must** be within VkPhysicalDeviceDescriptorBufferPropertiesEXT::maxSamplerDescriptorBufferRange of the sampler descriptor buffer binding
- VUID-VkSetDescriptorBufferOffsetsInfoEXT-pOffsets-08127
  The offsets in pOffsets **must** be small enough such that any location accessed by a shader as a resource descriptor **must** be within VkPhysicalDeviceDescriptorBufferPropertiesEXT::maxResourceDescriptorBufferRange of the resource descriptor buffer binding
- VUID-VkSetDescriptorBufferOffsetsInfoEXT-pBufferIndices-08064
  Each element of pBufferIndices **must** be less than VkPhysicalDeviceDescriptorBufferPropertiesEXT::maxDescriptorBufferBindings
- VUID-VkSetDescriptorBufferOffsetsInfoEXT-pBufferIndices-08065
  Each element of pBufferIndices **must** reference a valid descriptor buffer binding set by a previous call to vkCmdBindDescriptorBuffersEXT in commandBuffer
- VUID-VkSetDescriptorBufferOffsetsInfoEXT-firstSet-08066
  The sum of firstSet and setCount **must** be less than or equal to VkPipelineLayoutCreateInfo::setLayoutCount provided when layout was created
- VUID-VkSetDescriptorBufferOffsetsInfoEXT-firstSet-09006
  The VkDescriptorSetLayout for each set from firstSet to firstSet + setCount when layout was created **must** have been created with the VK_DESCRIPTOR_SET_LAYOUT_CREATE_DESCRIPTOR_BUFFER_BIT_EXT bit set
- VUID-VkSetDescriptorBufferOffsetsInfoEXT-None-09495
  If the dynamicPipelineLayout feature is not enabled, layout **must** be a valid VkPipelineLayout handle

- VUID-VkSetDescriptorBufferOffsetsInfoEXT-layout-09496
  If `layout` is VK_NULL_HANDLE, the `pNext` chain **must** include a valid VkPipelineLayoutCreateInfo structure

### Valid Usage (Implicit)

- VUID-VkSetDescriptorBufferOffsetsInfoEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_SET_DESCRIPTOR_BUFFER_OFFSETS_INFO_EXT`
- VUID-VkSetDescriptorBufferOffsetsInfoEXT-pNext-pNext
  `pNext` **must** be `NULL` or a pointer to a valid instance of VkPipelineLayoutCreateInfo
- VUID-VkSetDescriptorBufferOffsetsInfoEXT-sType-unique
  The `sType` value of each struct in the `pNext` chain **must** be unique
- VUID-VkSetDescriptorBufferOffsetsInfoEXT-stageFlags-parameter
  `stageFlags` **must** be a valid combination of VkShaderStageFlagBits values
- VUID-VkSetDescriptorBufferOffsetsInfoEXT-stageFlags-requiredbitmask
  `stageFlags` **must** not be `0`
- VUID-VkSetDescriptorBufferOffsetsInfoEXT-layout-parameter
  If `layout` is not VK_NULL_HANDLE, `layout` **must** be a valid VkPipelineLayout handle
- VUID-VkSetDescriptorBufferOffsetsInfoEXT-pBufferIndices-parameter
  `pBufferIndices` **must** be a valid pointer to an array of `setCount` `uint32_t` values
- VUID-VkSetDescriptorBufferOffsetsInfoEXT-pOffsets-parameter
  `pOffsets` **must** be a valid pointer to an array of `setCount` VkDeviceSize values
- VUID-VkSetDescriptorBufferOffsetsInfoEXT-setCount-arraylength
  `setCount` **must** be greater than `0`

To bind an embedded immutable sampler set to a command buffer, call:

```
// Provided by VK_EXT_descriptor_buffer
void vkCmdBindDescriptorBufferEmbeddedSamplersEXT(
    VkCommandBuffer                             commandBuffer,
    VkPipelineBindPoint                         pipelineBindPoint,
    VkPipelineLayout                            layout,
    uint32_t                                    set);
```

- `commandBuffer` is the command buffer that the embedded immutable samplers will be bound to.
- `pipelineBindPoint` is a VkPipelineBindPoint indicating the type of the pipeline that will use the embedded immutable samplers.
- `layout` is a VkPipelineLayout object used to program the bindings.
- `set` is the number of the set to be bound.

`vkCmdBindDescriptorBufferEmbeddedSamplersEXT` binds the embedded immutable samplers in `set` of

layout to set for the command buffer for subsequent bound pipeline commands set by pipelineBindPoint. Any previous binding to this set by vkCmdSetDescriptorBufferOffsetsEXT or this command is overwritten. Any sets that were last bound by a call to vkCmdBindDescriptorSets are invalidated upon calling this command. Other sets will also be invalidated upon calling this command if layout differs from the pipeline layout used to bind those other sets, as described in Pipeline Layout Compatibility.

### Valid Usage

- VUID-vkCmdBindDescriptorBufferEmbeddedSamplersEXT-set-08070
  The VkDescriptorSetLayout at index set when layout was created **must** have been created with the VK_DESCRIPTOR_SET_LAYOUT_CREATE_EMBEDDED_IMMUTABLE_SAMPLERS_BIT_EXT bit set
- VUID-vkCmdBindDescriptorBufferEmbeddedSamplersEXT-set-08071
  set **must** be less than or equal to VkPipelineLayoutCreateInfo::setLayoutCount provided when layout was created
- VUID-vkCmdBindDescriptorBufferEmbeddedSamplersEXT-None-08068
  The descriptorBuffer feature **must** be enabled
- VUID-vkCmdBindDescriptorBufferEmbeddedSamplersEXT-pipelineBindPoint-08069
  pipelineBindPoint **must** be supported by the commandBuffer's parent VkCommandPool's queue family

### Valid Usage (Implicit)

- VUID-vkCmdBindDescriptorBufferEmbeddedSamplersEXT-commandBuffer-parameter
  commandBuffer **must** be a valid VkCommandBuffer handle
- VUID-vkCmdBindDescriptorBufferEmbeddedSamplersEXT-pipelineBindPoint-parameter
  pipelineBindPoint **must** be a valid VkPipelineBindPoint value
- VUID-vkCmdBindDescriptorBufferEmbeddedSamplersEXT-layout-parameter
  layout **must** be a valid VkPipelineLayout handle
- VUID-vkCmdBindDescriptorBufferEmbeddedSamplersEXT-commandBuffer-recording
  commandBuffer **must** be in the recording state
- VUID-vkCmdBindDescriptorBufferEmbeddedSamplersEXT-commandBuffer-cmdpool
  The VkCommandPool that commandBuffer was allocated from **must** support graphics, or compute operations
- VUID-vkCmdBindDescriptorBufferEmbeddedSamplersEXT-videocoding
  This command **must** only be called outside of a video coding scope
- VUID-vkCmdBindDescriptorBufferEmbeddedSamplersEXT-commonparent
  Both of commandBuffer, and layout **must** have been created, allocated, or retrieved from the same VkDevice

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics<br>Compute | State |

Alternatively, to bind an embedded immutable sampler set to a command buffer, call:

```
// Provided by VK_KHR_maintenance6 with VK_EXT_descriptor_buffer
void vkCmdBindDescriptorBufferEmbeddedSamplers2EXT(
    VkCommandBuffer                                 commandBuffer,
    const VkBindDescriptorBufferEmbeddedSamplersInfoEXT*
pBindDescriptorBufferEmbeddedSamplersInfo);
```

- `commandBuffer` is the command buffer that the embedded immutable samplers will be bound to.
- `pBindDescriptorBufferEmbeddedSamplersInfo` is a pointer to a `VkBindDescriptorBufferEmbeddedSamplersInfoEXT` structure.

## Valid Usage

- VUID-vkCmdBindDescriptorBufferEmbeddedSamplers2EXT-descriptorBuffer-09472
  The `descriptorBuffer` feature **must** be enabled
- VUID-vkCmdBindDescriptorBufferEmbeddedSamplers2EXT-pBindDescriptorBufferEmbeddedSamplersInfo-09473
  Each bit in `pBindDescriptorBufferEmbeddedSamplersInfo->stageFlags` **must** be a stage supported by the `commandBuffer`'s parent `VkCommandPool`'s queue family

## Valid Usage (Implicit)

- VUID-vkCmdBindDescriptorBufferEmbeddedSamplers2EXT-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdBindDescriptorBufferEmbeddedSamplers2EXT-pBindDescriptorBufferEmbeddedSamplersInfo-parameter
  `pBindDescriptorBufferEmbeddedSamplersInfo` **must** be a valid pointer to a valid

VkBindDescriptorBufferEmbeddedSamplersInfoEXT structure

- VUID-vkCmdBindDescriptorBufferEmbeddedSamplers2EXT-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdBindDescriptorBufferEmbeddedSamplers2EXT-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics, or compute operations
- VUID-vkCmdBindDescriptorBufferEmbeddedSamplers2EXT-videocoding
  This command **must** only be called outside of a video coding scope

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics<br>Compute | State |

The `VkBindDescriptorBufferEmbeddedSamplersInfoEXT` structure is defined as:

```
// Provided by VK_KHR_maintenance6 with VK_EXT_descriptor_buffer
typedef struct VkBindDescriptorBufferEmbeddedSamplersInfoEXT {
    VkStructureType       sType;
    const void*           pNext;
    VkShaderStageFlags    stageFlags;
    VkPipelineLayout      layout;
    uint32_t              set;
} VkBindDescriptorBufferEmbeddedSamplersInfoEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `stageFlags` is a bitmask of VkShaderStageFlagBits specifying the shader stages that will use the embedded immutable samplers.
- `layout` is a VkPipelineLayout object used to program the bindings. If the `dynamicPipelineLayout` feature is enabled, `layout` **can** be VK_NULL_HANDLE and the layout **must** be specified by chaining VkPipelineLayoutCreateInfo structure off the `pNext`
- `set` is the number of the set to be bound.

If stageFlags specifies a subset of all stages corresponding to one or more pipeline bind points, the binding operation still affects all stages corresponding to the given pipeline bind point(s) as if the equivalent original version of this command had been called with the same parameters. For example, specifying a stageFlags value of VK_SHADER_STAGE_VERTEX_BIT | VK_SHADER_STAGE_FRAGMENT_BIT | VK_SHADER_STAGE_COMPUTE_BIT is equivalent to calling the original version of this command once with VK_PIPELINE_BIND_POINT_GRAPHICS and once with VK_PIPELINE_BIND_POINT_COMPUTE.

### Valid Usage

- VUID-VkBindDescriptorBufferEmbeddedSamplersInfoEXT-set-08070
  The VkDescriptorSetLayout at index set when layout was created **must** have been created with the VK_DESCRIPTOR_SET_LAYOUT_CREATE_EMBEDDED_IMMUTABLE_SAMPLERS_BIT_EXT bit set
- VUID-VkBindDescriptorBufferEmbeddedSamplersInfoEXT-set-08071
  set **must** be less than or equal to VkPipelineLayoutCreateInfo::setLayoutCount provided when layout was created
- VUID-VkBindDescriptorBufferEmbeddedSamplersInfoEXT-None-09495
  If the dynamicPipelineLayout feature is not enabled, layout **must** be a valid VkPipelineLayout handle
- VUID-VkBindDescriptorBufferEmbeddedSamplersInfoEXT-layout-09496
  If layout is VK_NULL_HANDLE, the pNext chain **must** include a valid VkPipelineLayoutCreateInfo structure

### Valid Usage (Implicit)

- VUID-VkBindDescriptorBufferEmbeddedSamplersInfoEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_BIND_DESCRIPTOR_BUFFER_EMBEDDED_SAMPLERS_INFO_EXT
- VUID-VkBindDescriptorBufferEmbeddedSamplersInfoEXT-pNext-pNext
  pNext **must** be NULL or a pointer to a valid instance of VkPipelineLayoutCreateInfo
- VUID-VkBindDescriptorBufferEmbeddedSamplersInfoEXT-sType-unique
  The sType value of each struct in the pNext chain **must** be unique
- VUID-VkBindDescriptorBufferEmbeddedSamplersInfoEXT-stageFlags-parameter
  stageFlags **must** be a valid combination of VkShaderStageFlagBits values
- VUID-VkBindDescriptorBufferEmbeddedSamplersInfoEXT-stageFlags-requiredbitmask
  stageFlags **must** not be 0
- VUID-VkBindDescriptorBufferEmbeddedSamplersInfoEXT-layout-parameter
  If layout is not VK_NULL_HANDLE, layout **must** be a valid VkPipelineLayout handle

## 14.4.3. Updating Descriptor Buffers

Updates to descriptor data in buffers **can** be performed by any operation on either the host or device that **can** access memory.

Descriptor buffer reads **can** be synchronized using VK_ACCESS_2_DESCRIPTOR_BUFFER_READ_BIT_EXT in the relevant shader stage.

### 14.4.4. Push Descriptors With Descriptor Buffers

If the descriptorBufferPushDescriptors feature is enabled, push descriptors **can** be used with descriptor buffers in the same way as with descriptor sets.

The VkPhysicalDeviceDescriptorBufferPropertiesEXT::bufferlessPushDescriptors property indicates whether the implementation requires a buffer to back push descriptors. If the property is VK_FALSE then before recording any push descriptors the application **must** bind exactly 1 descriptor buffer that was created with the VK_BUFFER_USAGE_PUSH_DESCRIPTORS_DESCRIPTOR_BUFFER_BIT_EXT bit set. When this buffer is bound any previously recorded push descriptors that are required for a subsequent command **must** be recorded again.

### 14.4.5. Capture and Replay

In a similar way to bufferDeviceAddressCaptureReplay, the descriptorBufferCaptureReplay feature allows the creation of opaque handles for objects at capture time that **can** be passed into object creation calls in a future replay, causing descriptors to be created with the same data. The opaque memory address for any memory used by these resources **must** have been captured using vkGetDeviceMemoryOpaqueCaptureAddressKHR and be replayed using VkMemoryOpaqueCaptureAddressAllocateInfo.

To get the opaque descriptor data for a buffer, call:

```
// Provided by VK_EXT_descriptor_buffer
VkResult vkGetBufferOpaqueCaptureDescriptorDataEXT(
    VkDevice                                    device,
    const VkBufferCaptureDescriptorDataInfoEXT* pInfo,
    void*                                       pData);
```

- device is the logical device that gets the data.
- pInfo is a pointer to a VkBufferCaptureDescriptorDataInfoEXT structure specifying the buffer.
- pData is a pointer to an application-allocated buffer where the data will be written.

#### Valid Usage

- VUID-vkGetBufferOpaqueCaptureDescriptorDataEXT-None-08072
  The descriptorBufferCaptureReplay feature **must** be enabled
- VUID-vkGetBufferOpaqueCaptureDescriptorDataEXT-pData-08073
  pData **must** point to a buffer that is at least VkPhysicalDeviceDescriptorBufferPropertiesEXT::bufferCaptureReplayDescriptorDataSize bytes in size
- VUID-vkGetBufferOpaqueCaptureDescriptorDataEXT-device-08074
  If device was created with multiple physical devices, then the

bufferDeviceAddressMultiDevice feature **must** be enabled

## Valid Usage (Implicit)

- VUID-vkGetBufferOpaqueCaptureDescriptorDataEXT-device-parameter
  device **must** be a valid VkDevice handle
- VUID-vkGetBufferOpaqueCaptureDescriptorDataEXT-pInfo-parameter
  pInfo **must** be a valid pointer to a valid VkBufferCaptureDescriptorDataInfoEXT structure
- VUID-vkGetBufferOpaqueCaptureDescriptorDataEXT-pData-parameter
  pData **must** be a pointer value

## Return Codes

**Success**

- VK_SUCCESS

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY

Information about the buffer to get descriptor buffer capture data for is passed in a VkBufferCaptureDescriptorDataInfoEXT structure:

```
// Provided by VK_EXT_descriptor_buffer
typedef struct VkBufferCaptureDescriptorDataInfoEXT {
    VkStructureType    sType;
    const void*        pNext;
    VkBuffer           buffer;
} VkBufferCaptureDescriptorDataInfoEXT;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- buffer is the VkBuffer handle of the buffer to get opaque capture data for.

## Valid Usage

- VUID-VkBufferCaptureDescriptorDataInfoEXT-buffer-08075
  buffer **must** have been created with VK_BUFFER_CREATE_DESCRIPTOR_BUFFER_CAPTURE_REPLAY_BIT_EXT set in VkBufferCreateInfo::flags

### Valid Usage (Implicit)

- VUID-VkBufferCaptureDescriptorDataInfoEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_BUFFER_CAPTURE_DESCRIPTOR_DATA_INFO_EXT`
- VUID-VkBufferCaptureDescriptorDataInfoEXT-pNext-pNext
  `pNext` **must** be `NULL`
- VUID-VkBufferCaptureDescriptorDataInfoEXT-buffer-parameter
  `buffer` **must** be a valid VkBuffer handle

To get the opaque capture descriptor data for an image, call:

```
// Provided by VK_EXT_descriptor_buffer
VkResult vkGetImageOpaqueCaptureDescriptorDataEXT(
    VkDevice                                    device,
    const VkImageCaptureDescriptorDataInfoEXT*  pInfo,
    void*                                       pData);
```

- `device` is the logical device that gets the data.
- `pInfo` is a pointer to a VkImageCaptureDescriptorDataInfoEXT structure specifying the image.
- `pData` is a pointer to an application-allocated buffer where the data will be written.

### Valid Usage

- VUID-vkGetImageOpaqueCaptureDescriptorDataEXT-None-08076
  The `descriptorBufferCaptureReplay` feature **must** be enabled
- VUID-vkGetImageOpaqueCaptureDescriptorDataEXT-pData-08077
  `pData` **must** point to a buffer that is at least VkPhysicalDeviceDescriptorBufferPropertiesEXT::`imageCaptureReplayDescriptorDataSize` bytes in size
- VUID-vkGetImageOpaqueCaptureDescriptorDataEXT-device-08078
  If `device` was created with multiple physical devices, then the `bufferDeviceAddressMultiDevice` feature **must** be enabled

### Valid Usage (Implicit)

- VUID-vkGetImageOpaqueCaptureDescriptorDataEXT-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkGetImageOpaqueCaptureDescriptorDataEXT-pInfo-parameter
  `pInfo` **must** be a valid pointer to a valid VkImageCaptureDescriptorDataInfoEXT structure
- VUID-vkGetImageOpaqueCaptureDescriptorDataEXT-pData-parameter
  `pData` **must** be a pointer value

### Return Codes

**Success**

- VK_SUCCESS

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY

Information about the image to get descriptor buffer capture data for is passed in a VkImageCaptureDescriptorDataInfoEXT structure:

```
// Provided by VK_EXT_descriptor_buffer
typedef struct VkImageCaptureDescriptorDataInfoEXT {
    VkStructureType    sType;
    const void*        pNext;
    VkImage            image;
} VkImageCaptureDescriptorDataInfoEXT;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- image is the VkImage handle of the image to get opaque capture data for.

### Valid Usage

- VUID-VkImageCaptureDescriptorDataInfoEXT-image-08079
  image **must** have been created with VK_IMAGE_CREATE_DESCRIPTOR_BUFFER_CAPTURE_REPLAY_BIT_EXT set in VkImageCreateInfo::flags

### Valid Usage (Implicit)

- VUID-VkImageCaptureDescriptorDataInfoEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_IMAGE_CAPTURE_DESCRIPTOR_DATA_INFO_EXT
- VUID-VkImageCaptureDescriptorDataInfoEXT-pNext-pNext
  pNext **must** be NULL
- VUID-VkImageCaptureDescriptorDataInfoEXT-image-parameter
  image **must** be a valid VkImage handle

To get the opaque capture descriptor data for an image view, call:

```
// Provided by VK_EXT_descriptor_buffer
VkResult vkGetImageViewOpaqueCaptureDescriptorDataEXT(
    VkDevice                                    device,
    const VkImageViewCaptureDescriptorDataInfoEXT* pInfo,
    void*                                       pData);
```

- `device` is the logical device that gets the data.
- `pInfo` is a pointer to a VkImageViewCaptureDescriptorDataInfoEXT structure specifying the image view.
- `pData` is a pointer to an application-allocated buffer where the data will be written.

## Valid Usage

- VUID-vkGetImageViewOpaqueCaptureDescriptorDataEXT-None-08080
  The `descriptorBufferCaptureReplay` feature **must** be enabled
- VUID-vkGetImageViewOpaqueCaptureDescriptorDataEXT-pData-08081
  `pData` **must** point to a buffer that is at least VkPhysicalDeviceDescriptorBufferPropertiesEXT::`imageViewCaptureReplayDescriptorDataSize` bytes in size
- VUID-vkGetImageViewOpaqueCaptureDescriptorDataEXT-device-08082
  If `device` was created with multiple physical devices, then the `bufferDeviceAddressMultiDevice` feature **must** be enabled

## Valid Usage (Implicit)

- VUID-vkGetImageViewOpaqueCaptureDescriptorDataEXT-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkGetImageViewOpaqueCaptureDescriptorDataEXT-pInfo-parameter
  `pInfo` **must** be a valid pointer to a valid VkImageViewCaptureDescriptorDataInfoEXT structure
- VUID-vkGetImageViewOpaqueCaptureDescriptorDataEXT-pData-parameter
  `pData` **must** be a pointer value

## Return Codes

### Success

- `VK_SUCCESS`

### Failure

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`

Information about the image view to get descriptor buffer capture data for is passed in a VkImageViewCaptureDescriptorDataInfoEXT structure:

```
// Provided by VK_EXT_descriptor_buffer
typedef struct VkImageViewCaptureDescriptorDataInfoEXT {
    VkStructureType    sType;
    const void*        pNext;
    VkImageView        imageView;
} VkImageViewCaptureDescriptorDataInfoEXT;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- imageView is the VkImageView handle of the image view to get opaque capture data for.

### Valid Usage

- VUID-VkImageViewCaptureDescriptorDataInfoEXT-imageView-08083
  imageView **must** have been created with VK_IMAGE_VIEW_CREATE_DESCRIPTOR_BUFFER_CAPTURE_REPLAY_BIT_EXT set in VkImageViewCreateInfo::flags

### Valid Usage (Implicit)

- VUID-VkImageViewCaptureDescriptorDataInfoEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_IMAGE_VIEW_CAPTURE_DESCRIPTOR_DATA_INFO_EXT
- VUID-VkImageViewCaptureDescriptorDataInfoEXT-pNext-pNext
  pNext **must** be NULL
- VUID-VkImageViewCaptureDescriptorDataInfoEXT-imageView-parameter
  imageView **must** be a valid VkImageView handle

To get the opaque capture descriptor data for a sampler, call:

```
// Provided by VK_EXT_descriptor_buffer
VkResult vkGetSamplerOpaqueCaptureDescriptorDataEXT(
    VkDevice                                    device,
    const VkSamplerCaptureDescriptorDataInfoEXT* pInfo,
    void*                                       pData);
```

- device is the logical device that gets the data.
- pInfo is a pointer to a VkSamplerCaptureDescriptorDataInfoEXT structure specifying the sampler.
- pData is a pointer to an application-allocated buffer where the data will be written.

## Valid Usage

- VUID-vkGetSamplerOpaqueCaptureDescriptorDataEXT-None-08084
  The `descriptorBufferCaptureReplay` feature **must** be enabled
- VUID-vkGetSamplerOpaqueCaptureDescriptorDataEXT-pData-08085
  `pData` **must** point to a buffer that is at least VkPhysicalDeviceDescriptorBufferPropertiesEXT::`samplerCaptureReplayDescriptorDataSize` bytes in size
- VUID-vkGetSamplerOpaqueCaptureDescriptorDataEXT-device-08086
  If `device` was created with multiple physical devices, then the `bufferDeviceAddressMultiDevice` feature **must** be enabled

## Valid Usage (Implicit)

- VUID-vkGetSamplerOpaqueCaptureDescriptorDataEXT-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkGetSamplerOpaqueCaptureDescriptorDataEXT-pInfo-parameter
  `pInfo` **must** be a valid pointer to a valid VkSamplerCaptureDescriptorDataInfoEXT structure
- VUID-vkGetSamplerOpaqueCaptureDescriptorDataEXT-pData-parameter
  `pData` **must** be a pointer value

## Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`

Information about the sampler to get descriptor buffer capture data for is passed in a `VkSamplerCaptureDescriptorDataInfoEXT` structure:

```
// Provided by VK_EXT_descriptor_buffer
typedef struct VkSamplerCaptureDescriptorDataInfoEXT {
    VkStructureType    sType;
    const void*        pNext;
    VkSampler          sampler;
} VkSamplerCaptureDescriptorDataInfoEXT;
```

- `sType` is a VkStructureType value identifying this structure.

- pNext is NULL or a pointer to a structure extending this structure.
- sampler is the VkSampler handle of the sampler to get opaque capture data for.

### Valid Usage

- VUID-VkSamplerCaptureDescriptorDataInfoEXT-sampler-08087
  sampler **must** have been created with VK_SAMPLER_CREATE_DESCRIPTOR_BUFFER_CAPTURE_REPLAY_BIT_EXT set in VkSamplerCreateInfo::flags

### Valid Usage (Implicit)

- VUID-VkSamplerCaptureDescriptorDataInfoEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_SAMPLER_CAPTURE_DESCRIPTOR_DATA_INFO_EXT
- VUID-VkSamplerCaptureDescriptorDataInfoEXT-pNext-pNext
  pNext **must** be NULL
- VUID-VkSamplerCaptureDescriptorDataInfoEXT-sampler-parameter
  sampler **must** be a valid VkSampler handle

To get the opaque capture descriptor data for an acceleration structure, call:

```
// Provided by VK_EXT_descriptor_buffer with VK_KHR_acceleration_structure or
VK_NV_ray_tracing
VkResult vkGetAccelerationStructureOpaqueCaptureDescriptorDataEXT(
    VkDevice                                    device,
    const VkAccelerationStructureCaptureDescriptorDataInfoEXT* pInfo,
    void*                                       pData);
```

- device is the logical device that gets the data.
- pInfo is a pointer to a VkAccelerationStructureCaptureDescriptorDataInfoEXT structure specifying the acceleration structure.
- pData is a pointer to an application-allocated buffer where the data will be written.

### Valid Usage

- VUID-vkGetAccelerationStructureOpaqueCaptureDescriptorDataEXT-None-08088
  The descriptorBufferCaptureReplay feature **must** be enabled
- VUID-vkGetAccelerationStructureOpaqueCaptureDescriptorDataEXT-pData-08089
  pData **must** point to a buffer that is at least VkPhysicalDeviceDescriptorBufferPropertiesEXT::accelerationStructureCaptureReplayDescriptorDataSize bytes in size
- VUID-vkGetAccelerationStructureOpaqueCaptureDescriptorDataEXT-device-08090

If device was created with multiple physical devices, then the bufferDeviceAddressMultiDevice feature **must** be enabled

### Valid Usage (Implicit)

- VUID-vkGetAccelerationStructureOpaqueCaptureDescriptorDataEXT-device-parameter device **must** be a valid VkDevice handle
- VUID-vkGetAccelerationStructureOpaqueCaptureDescriptorDataEXT-pInfo-parameter pInfo **must** be a valid pointer to a valid VkAccelerationStructureCaptureDescriptorDataInfoEXT structure
- VUID-vkGetAccelerationStructureOpaqueCaptureDescriptorDataEXT-pData-parameter pData **must** be a pointer value

### Return Codes

**Success**

- VK_SUCCESS

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY

Information about the acceleration structure to get descriptor buffer capture data for is passed in a VkAccelerationStructureCaptureDescriptorDataInfoEXT structure:

```
// Provided by VK_EXT_descriptor_buffer with VK_KHR_acceleration_structure or
VK_NV_ray_tracing
typedef struct VkAccelerationStructureCaptureDescriptorDataInfoEXT {
    VkStructureType                 sType;
    const void*                     pNext;
    VkAccelerationStructureKHR      accelerationStructure;
    VkAccelerationStructureNV       accelerationStructureNV;
} VkAccelerationStructureCaptureDescriptorDataInfoEXT;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- accelerationStructure is the VkAccelerationStructureKHR handle of the acceleration structure to get opaque capture data for.
- accelerationStructureNV is the VkAccelerationStructureNV handle of the acceleration structure to get opaque capture data for.

## Valid Usage

- VUID-VkAccelerationStructureCaptureDescriptorDataInfoEXT-accelerationStructure-08091
  If `accelerationStructure` is not VK_NULL_HANDLE then `accelerationStructure` **must** have been created with `VK_ACCELERATION_STRUCTURE_CREATE_DESCRIPTOR_BUFFER_CAPTURE_REPLAY_BIT_EXT` set in VkAccelerationStructureCreateInfoKHR::`createFlags`
- VUID-VkAccelerationStructureCaptureDescriptorDataInfoEXT-accelerationStructureNV-08092
  If `accelerationStructureNV` is not VK_NULL_HANDLE then `accelerationStructureNV` **must** have been created with `VK_ACCELERATION_STRUCTURE_CREATE_DESCRIPTOR_BUFFER_CAPTURE_REPLAY_BIT_EXT` set in VkAccelerationStructureCreateInfoNV::`info.flags`
- VUID-VkAccelerationStructureCaptureDescriptorDataInfoEXT-accelerationStructure-08093
  If `accelerationStructure` is not VK_NULL_HANDLE then `accelerationStructureNV` **must** be VK_NULL_HANDLE
- VUID-VkAccelerationStructureCaptureDescriptorDataInfoEXT-accelerationStructureNV-08094
  If `accelerationStructureNV` is not VK_NULL_HANDLE then `accelerationStructure` **must** be VK_NULL_HANDLE

## Valid Usage (Implicit)

- VUID-VkAccelerationStructureCaptureDescriptorDataInfoEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_ACCELERATION_STRUCTURE_CAPTURE_DESCRIPTOR_DATA_INFO_EXT`
- VUID-VkAccelerationStructureCaptureDescriptorDataInfoEXT-pNext-pNext
  `pNext` **must** be `NULL`
- VUID-VkAccelerationStructureCaptureDescriptorDataInfoEXT-accelerationStructure-parameter
  If `accelerationStructure` is not VK_NULL_HANDLE, `accelerationStructure` **must** be a valid VkAccelerationStructureKHR handle
- VUID-VkAccelerationStructureCaptureDescriptorDataInfoEXT-accelerationStructureNV-parameter
  If `accelerationStructureNV` is not VK_NULL_HANDLE, `accelerationStructureNV` **must** be a valid VkAccelerationStructureNV handle
- VUID-VkAccelerationStructureCaptureDescriptorDataInfoEXT-commonparent
  Both of `accelerationStructure`, and `accelerationStructureNV` that are valid handles of non-ignored parameters **must** have been created, allocated, or retrieved from the same VkDevice

The VkOpaqueCaptureDescriptorDataCreateInfoEXT structure is defined as:

```
// Provided by VK_EXT_descriptor_buffer
typedef struct VkOpaqueCaptureDescriptorDataCreateInfoEXT {
    VkStructureType    sType;
    const void*        pNext;
    const void*        opaqueCaptureDescriptorData;
} VkOpaqueCaptureDescriptorDataCreateInfoEXT;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- opaqueCaptureDescriptorData is a pointer to an application-allocated buffer containing opaque capture data retrieved using vkGetBufferOpaqueCaptureDescriptorDataEXT, vkGetImageOpaqueCaptureDescriptorDataEXT, vkGetImageViewOpaqueCaptureDescriptorDataEXT, vkGetSamplerOpaqueCaptureDescriptorDataEXT, or vkGetAccelerationStructureOpaqueCaptureDescriptorDataEXT.

During replay, opaque descriptor capture data **can** be specified by adding a VkOpaqueCaptureDescriptorDataCreateInfoEXT structure to the relevant pNext chain of a VkBufferCreateInfo, VkImageCreateInfo, VkImageViewCreateInfo, VkSamplerCreateInfo, VkAccelerationStructureCreateInfoNV or VkAccelerationStructureCreateInfoKHR structure.

### Valid Usage (Implicit)

- VUID-VkOpaqueCaptureDescriptorDataCreateInfoEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_OPAQUE_CAPTURE_DESCRIPTOR_DATA_CREATE_INFO_EXT
- VUID-VkOpaqueCaptureDescriptorDataCreateInfoEXT-opaqueCaptureDescriptorData-parameter
  opaqueCaptureDescriptorData **must** be a pointer value

# Chapter 15. Shader Interfaces

When a pipeline is created, the set of shaders specified in the corresponding `VkPipelineCreateInfo` structure are implicitly linked at a number of different interfaces.

- Shader Input and Output Interface
- Vertex Input Interface
- Fragment Output Interface
- Fragment Tile Image Interface
- Fragment Input Attachment Interface
- Ray Tracing Pipeline Interface
- Shader Resource Interface
- Geometry Shader Passthrough

This chapter describes valid uses for a set of SPIR-V decorations. Any other use of one of these decorations is invalid, with the exception that, when using SPIR-V versions 1.4 and earlier: `Block`, `BufferBlock`, `Offset`, `ArrayStride`, and `MatrixStride` can also decorate types and type members used by variables in the `Private` and `Function` storage classes.

| | |
|---|---|
| **NOTE** | In this chapter, there are references to SPIR-V terms such as the `MeshNV` execution model. These terms will appear even in a build of the specification which does not support any extensions. This is as intended, since these terms appear in the unified SPIR-V specification without such qualifiers. |

## 15.1. Shader Input and Output Interfaces

When multiple stages are present in a pipeline, the outputs of one stage form an interface with the inputs of the next stage. When such an interface involves a shader, shader outputs are matched against the inputs of the next stage, and shader inputs are matched against the outputs of the previous stage.

All the variables forming the shader input and output *interfaces* are listed as operands to the `OpEntryPoint` instruction and are declared with the `Input` or `Output` storage classes, respectively, in the SPIR-V module. These generally form the interfaces between consecutive shader stages, regardless of any non-shader stages between the consecutive shader stages.

There are two classes of variables that **can** be matched between shader stages, built-in variables and user-defined variables. Each class has a different set of matching criteria.

`Output` variables of a shader stage have undefined values until the shader writes to them or uses the `Initializer` operand when declaring the variable.

### 15.1.1. Built-in Interface Block

Shader built-in variables meeting the following requirements define the *built-in interface block*.

They **must**

- be explicitly declared (there are no implicit built-ins),
- be identified with a `BuiltIn` decoration,
- form object types as described in the Built-in Variables section, and
- be declared in a block whose top-level members are the built-ins.

There **must** be no more than one built-in interface block per shader per interface , except for the mesh output interface where there **must** be at most one built-in interface block decorated with the `PerPrimitiveEXT` decoration and at most one built-in interface block without this decoration .

Built-ins **must** not have any `Location` or `Component` decorations.

### 15.1.2. User-defined Variable Interface

The non-built-in variables listed by `OpEntryPoint` with the `Input` or `Output` storage class form the *user-defined variable interface*. These **must** have numeric type or, recursively, composite types of such types. If an implementation supports `storageInputOutput16`, components **can** have a width of 16 bits. These variables **must** be identified with a `Location` decoration and **can** also be identified with a `Component` decoration.

### 15.1.3. Interface Matching

An output variable, block, or structure member in a given shader stage has an interface match with an input variable, block, or structure member in a subsequent shader stage if they both adhere to the following conditions:

- They have equivalent decorations, other than:
  - `XfbBuffer`, `XfbStride`, `Offset`, and `Stream`
  - one is not decorated with `Component` and the other is declared with a `Component` of `0`
  - Interpolation decorations
  - `RelaxedPrecision` if one is an input variable and the other an output variable
- Their types match as follows:
  - if the input is declared in a tessellation control or geometry shader as an `OpTypeArray` with an `Element Type` equivalent to the `OpType*` declaration of the output, and neither is a structure member; or
  - if the `maintenance4` feature is enabled, they are declared as `OpTypeVector` variables, and the output has a `Component Count` value higher than that of the input but the same `Component Type`; or
  - if the output is declared in a mesh shader as an `OpTypeArray` with an `Element Type` equivalent to the `OpType*` declaration of the input, and neither is a structure member; or
  - if the input is decorated with `PerVertexKHR`, and is declared in a fragment shader as an `OpTypeArray` with an `Element Type` equivalent to the `OpType*` declaration of the output, and neither the input nor the output is a structure member; or

    - if in any other case they are declared with an equivalent `OpType*` declaration.
- If both are structures and every member has an interface match.

| | |
|---|---|
| **NOTE** | The word "structure" above refers to both variables that have an `OpTypeStruct` type and interface blocks (which are also declared as `OpTypeStruct`). |

If the pipeline is compiled as separate graphics pipeline libraries and the `graphicsPipelineLibraryIndependentInterpolationDecoration` limit is not supported, matches are not found if the interpolation decorations differ between the last pre-rasterization shader stage and the fragment shader stage.

All input variables and blocks **must** have an interface match in the preceding shader stage, except for built-in variables in fragment shaders. Shaders **can** declare and write to output variables that are not declared or read by the subsequent stage.

Matching rules for *passthrough geometry shaders* are slightly different and are described in the Passthrough Interface Matching section.

The value of an input variable is undefined if the preceding stage does not write to a matching output variable, as described above.

### 15.1.4. Location Assignment

This section describes `Location` assignments for user-defined variables and how many `Location` slots are consumed by a given user-variable type. As mentioned above, some inputs and outputs have an additional level of arrayness relative to other shader inputs and outputs. This outer array level is removed from the type before considering how many `Location` slots the type consumes.

The `Location` value specifies an interface slot comprised of a 32-bit four-component vector conveyed between stages. The `Component` specifies word components within these vector `Location` slots. Only types with widths of 16, 32 or 64 are supported in shader interfaces.

Inputs and outputs of the following types consume a single interface `Location`:

- 16-bit scalar and vector types, and
- 32-bit scalar and vector types, and
- 64-bit scalar and 2-component vector types.

64-bit three- and four-component vectors consume two consecutive `Location` slots.

If a declared input or output is an array of size $n$ and each element takes $m$ `Location` slots, it will be assigned $m \times n$ consecutive `Location` slots starting with the specified `Location`.

If the declared input or output is an $n \times m$ 16-, 32- or 64-bit matrix, it will be assigned multiple `Location` slots starting with the specified `Location`. The number of `Location` slots assigned for each matrix will be the same as for an $n$-element array of $m$-component vectors.

An `OpVariable` with a structure type that is not a block **must** be decorated with a `Location`.

When an `OpVariable` with a structure type (either block or non-block) is decorated with a `Location`, the members in the structure type **must** not be decorated with a `Location`. The `OpVariable`'s members are assigned consecutive `Location` slots in declaration order, starting from the first member, which is assigned the `Location` decoration from the `OpVariable`.

When a block-type `OpVariable` is declared without a `Location` decoration, each member in its structure type **must** be decorated with a `Location`. Types nested deeper than the top-level members **must** not have `Location` decorations.

The `Location` slots consumed by block and structure members are determined by applying the rules above in a depth-first traversal of the instantiated members as though the structure or block member were declared as an input or output variable of the same type.

Any two inputs listed as operands on the same `OpEntryPoint` **must** not be assigned the same `Location` slot and `Component` word, either explicitly or implicitly. Any two outputs listed as operands on the same `OpEntryPoint` **must** not be assigned the same `Location` slot and `Component` word, either explicitly or implicitly.

The number of input and output `Location` slots available for a shader input or output interface is limited, and dependent on the shader stage as described in Shader Input and Output Locations. All variables in both the built-in interface block and the user-defined variable interface count against these limits. Each effective `Location` **must** have a value less than the number of `Location` slots available for the given interface, as specified in the "Locations Available" column in Shader Input and Output Locations.

*Table 19. Shader Input and Output Locations*

| Shader Interface | Locations Available |
|---|---|
| vertex input | `maxVertexInputAttributes` |
| vertex output | `maxVertexOutputComponents` / 4 |
| tessellation control input | `maxTessellationControlPerVertexInputComponents` / 4 |
| tessellation control output | `maxTessellationControlPerVertexOutputComponents` / 4 |
| tessellation evaluation input | `maxTessellationEvaluationInputComponents` / 4 |
| tessellation evaluation output | `maxTessellationEvaluationOutputComponents` / 4 |
| geometry input | `maxGeometryInputComponents` / 4 |
| geometry output | `maxGeometryOutputComponents` / 4 |
| fragment input | `maxFragmentInputComponents` / 4 |
| fragment output | `maxFragmentOutputAttachments` |
| mesh output | `maxMeshOutputComponents` / 4 |
| cluster culling output | `maxOutputClusterCount` |

### 15.1.5. Component Assignment

The `Component` decoration allows the `Location` to be more finely specified for scalars and vectors, down to the individual `Component` word within a `Location` slot that are consumed. The `Component` word within a `Location` are 0, 1, 2, and 3. A variable or block member starting at `Component` N will consume `Component` words N, N+1, N+2, ... up through its size. For 16-, and 32-bit types, it is invalid if this sequence of `Component` words gets larger than 3. A scalar 64-bit type will consume two of these `Component` words in sequence, and a two-component 64-bit vector type will consume all four `Component` words available within a `Location`. A three- or four-component 64-bit vector type **must** not specify a non-zero `Component` decoration. A three-component 64-bit vector type will consume all four `Component` words of the first `Location` and `Component` 0 and 1 of the second `Location`. This leaves `Component` 2 and 3 available for other component-qualified declarations.

A scalar or two-component 64-bit data type **must** not specify a `Component` decoration of 1 or 3. A `Component` decoration **must** not be specified for any type that is not a scalar or vector.

A four-component 64-bit data type will consume all four `Component` words of the first `Location` and all four `Component` words of the second `Location`.

## 15.2. Vertex Input Interface

When the vertex stage is present in a pipeline, the vertex shader input variables form an interface with the vertex input attributes. The vertex shader input variables are matched by the `Location` and `Component` decorations to the vertex input attributes specified in the `pVertexInputState` member of the VkGraphicsPipelineCreateInfo structure.

The vertex shader input variables listed by `OpEntryPoint` with the `Input` storage class form the *vertex input interface*. These variables **must** be identified with a `Location` decoration and **can** also be identified with a `Component` decoration.

For the purposes of interface matching: variables declared without a `Component` decoration are considered to have a `Component` decoration of zero. The number of available vertex input `Location` slots is given by the `maxVertexInputAttributes` member of the `VkPhysicalDeviceLimits` structure.

See Attribute Location and Component Assignment for details.

All vertex shader inputs declared as above **must** have a corresponding attribute and binding in the pipeline.

## 15.3. Fragment Output Interface

When the fragment stage is present in a pipeline, the fragment shader outputs form an interface with the output attachments defined by a render pass instance. The fragment shader output variables are matched by the `Location` and `Component` decorations to specified color attachments.

The fragment shader output variables listed by `OpEntryPoint` with the `Output` storage class form the *fragment output interface*. These variables **must** be identified with a `Location` decoration. They **can** also be identified with a `Component` decoration and/or an `Index` decoration. For the purposes of interface matching: variables declared without a `Component` decoration are considered to have a

Component decoration of zero, and variables declared without an Index decoration are considered to have an Index decoration of zero.

A fragment shader output variable identified with a Location decoration of *i* is associated with the color attachment indicated by VkRenderingInfo::pColorAttachments[*i*]. When using render pass objects, it is associated with the color attachment indicated by VkSubpassDescription::pColorAttachments[*i*]. Values are written to those attachments after passing through the blending unit as described in Blending, if enabled. Locations are consumed as described in Location Assignment. The number of available fragment output Location slots is given by the maxFragmentOutputAttachments member of the VkPhysicalDeviceLimits structure.

If the dynamicRenderingLocalRead feature is supported, fragment output locations **can** be remapped when using dynamic rendering.

To set the fragment output location mappings during rendering, call:

```
// Provided by VK_KHR_dynamic_rendering_local_read
void vkCmdSetRenderingAttachmentLocationsKHR(
    VkCommandBuffer                             commandBuffer,
    const VkRenderingAttachmentLocationInfoKHR* pLocationInfo);
```

- commandBuffer is the command buffer into which the command will be recorded.
- pLocationInfo is a VkRenderingAttachmentLocationInfoKHR structure indicating the new mappings.

This command sets the attachment location mappings for subsequent drawing commands, and **must** match the mappings provided to the currently bound pipeline, if one is bound, which **can** be set by chaining VkRenderingAttachmentLocationInfoKHR to VkGraphicsPipelineCreateInfo.

Until this command is called, mappings in the command buffer state are treated as each color attachment specified in vkCmdBeginRenderingKHR having a location equal to its index in VkRenderingInfo::pColorAttachments. This state is reset whenever vkCmdBeginRenderingKHR is called.

### Valid Usage

- VUID-vkCmdSetRenderingAttachmentLocationsKHR-dynamicRenderingLocalRead-09509
  dynamicRenderingLocalRead **must** be enabled
- VUID-vkCmdSetRenderingAttachmentLocationsKHR-pLocationInfo-09510
  pLocationInfo->colorAttachmentCount **must** be equal to the value of VkRenderingInfo::colorAttachmentCount used to begin the current render pass instance
- VUID-vkCmdSetRenderingAttachmentLocationsKHR-commandBuffer-09511
  The current render pass instance **must** have been started or resumed by vkCmdBeginRenderingKHR in this commandBuffer

## Valid Usage (Implicit)

- VUID-vkCmdSetRenderingAttachmentLocationsKHR-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdSetRenderingAttachmentLocationsKHR-pLocationInfo-parameter
  `pLocationInfo` **must** be a valid pointer to a valid VkRenderingAttachmentLocationInfoKHR structure
- VUID-vkCmdSetRenderingAttachmentLocationsKHR-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdSetRenderingAttachmentLocationsKHR-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- VUID-vkCmdSetRenderingAttachmentLocationsKHR-renderpass
  This command **must** only be called inside of a render pass instance
- VUID-vkCmdSetRenderingAttachmentLocationsKHR-videocoding
  This command **must** only be called outside of a video coding scope

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Inside | Outside | Graphics | State |

The `VkRenderingAttachmentLocationInfoKHR` structure is defined as:

```
// Provided by VK_KHR_dynamic_rendering_local_read
typedef struct VkRenderingAttachmentLocationInfoKHR {
    VkStructureType    sType;
    const void*        pNext;
    uint32_t           colorAttachmentCount;
    const uint32_t*    pColorAttachmentLocations;
} VkRenderingAttachmentLocationInfoKHR;
```

- `sType` is a VkStructureType value identifying this structure.

- pNext is NULL or a pointer to a structure extending this structure.
- colorAttachmentCount is the number of elements in pColorAttachmentLocations.
- pColorAttachmentLocations is a pointer to an array of colorAttachmentCount uint32_t values defining remapped locations for color attachments.

This structure allows applications to remap the locations of color attachments to different fragment shader output locations.

Each element of pColorAttachmentLocations set to VK_ATTACHMENT_UNUSED will be inaccessible to this pipeline as a color attachment; no location will map to it. Each element of pColorAttachmentLocations set to any other value will map the specified location value to the color attachment specified in the render pass at the corresponding index in the pColorAttachmentLocations array. Any writes to a fragment output location that is not mapped to an attachment **must** be discarded.

If pColorAttachmentLocations is NULL, it is equivalent to setting each element to its index within the array.

This structure **can** be included in the pNext chain of a VkGraphicsPipelineCreateInfo structure to set this state for a pipeline. If this structure is not included in the pNext chain of VkGraphicsPipelineCreateInfo, it is equivalent to specifying this structure with the following properties:

- colorAttachmentCount set to VkPipelineRenderingCreateInfo::colorAttachmentCount.
- pColorAttachmentLocations set to NULL.

This structure **can** be included in the pNext chain of a VkCommandBufferInheritanceInfo structure to specify inherited state from the primary command buffer. If VkCommandBufferInheritanceInfo::renderPass is not VK_NULL_HANDLE, or VK_COMMAND_BUFFER_USAGE_RENDER_PASS_CONTINUE_BIT is not specified in VkCommandBufferBeginInfo::flags, members of this structure are ignored. If this structure is not included in the pNext chain of VkCommandBufferInheritanceInfo, it is equivalent to specifying this structure with the following properties:

- colorAttachmentCount set to VkCommandBufferInheritanceRenderingInfo::colorAttachmentCount.
- pColorAttachmentLocations set to NULL.

## Valid Usage

- VUID-VkRenderingAttachmentLocationInfoKHR-dynamicRenderingLocalRead-09512
  If the dynamicRenderingLocalRead feature is not enabled, and pColorAttachmentLocations is not NULL, each element **must** be set to the value of its index within the array
- VUID-VkRenderingAttachmentLocationInfoKHR-pColorAttachmentLocations-09513
  Elements of pColorAttachmentLocations that are not VK_ATTACHMENT_UNUSED **must** each be unique
- VUID-VkRenderingAttachmentLocationInfoKHR-colorAttachmentCount-09514
  colorAttachmentCount **must** be less than or equal to maxColorAttachments

- VUID-VkRenderingAttachmentLocationInfoKHR-pColorAttachmentLocations-09515
  Each element of `pColorAttachmentLocations` **must** be less than `maxColorAttachments`

### Valid Usage (Implicit)

- VUID-VkRenderingAttachmentLocationInfoKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_RENDERING_ATTACHMENT_LOCATION_INFO_KHR`

When an active fragment shader invocation finishes, the values of all fragment shader outputs are copied out and used as blend inputs or color attachments writes. If the invocation does not set a value for them, the input values to those blending or color attachment writes are undefined.

Components of the output variables are assigned as described in Component Assignment. Output `Component` words identified as 0, 1, 2, and 3 will be directed to the R, G, B, and A inputs to the blending unit, respectively, or to the output attachment if blending is disabled. If two variables are placed within the same `Location`, they **must** have the same underlying type (floating-point or integer). `Component` words which do not correspond to any fragment shader output will also result in undefined values for blending or color attachment writes.

Fragment outputs identified with an `Index` of zero are directed to the first input of the blending unit associated with the corresponding `Location`. Outputs identified with an `Index` of one are directed to the second input of the corresponding blending unit.

There **must** be no output variable which has the same `Location`, `Component`, and `Index` as any other, either explicitly declared or implied.

Output values written by a fragment shader **must** be declared with either `OpTypeFloat` or `OpTypeInt`, and a `Width` of 32. If `storageInputOutput16` is supported, output values written by a fragment shader **can** be also declared with either `OpTypeFloat` or `OpTypeInt` and a `Width` of 16. Composites of these types are also permitted. If the color attachment has a signed or unsigned normalized fixed-point format, color values are assumed to be floating-point and are converted to fixed-point as described in Conversion From Floating-Point to Normalized Fixed-Point; If the color attachment has an integer format, color values are assumed to be integers and converted to the bit-depth of the target. Any value that cannot be represented in the attachment's format is undefined. For any other attachment format no conversion is performed. If the type of the values written by the fragment shader do not match the format of the corresponding color attachment, the resulting values are undefined for those components.

## 15.4. Legacy Dithering

The application **can** enable dithering to be applied to the color output of a subpass, by using the `VK_SUBPASS_DESCRIPTION_ENABLE_LEGACY_DITHERING_BIT_EXT` flag. For use in a dynamic render pass, the `VK_RENDERING_ENABLE_LEGACY_DITHERING_BIT_EXT` flag **must** be used. In that case, the pipelines used **must** have been created with `VK_PIPELINE_CREATE_2_ENABLE_LEGACY_DITHERING_BIT_EXT`.

When dithering is enabled, the implementation **may** modify the output color value c by one ULP. This modification **must** only depend on the framebuffer coordinates $(x_f,y_f)$ of the sample, as well as

on the value of c.

The exact details of the dithering algorithm are unspecified, including the algorithm itself, the formats dithering is applied to, and the stage in which it is applied.

| | |
|---|---|
| **NOTE** | This extension is intended only for use by OpenGL emulation layers, and as such the dithering algorithm applied to the subpass **should** be equivalent to the vendor's OpenGL implementation, if any. |

## 15.5. Fragment Tile Image Interface

When a fragment stage is present in a pipeline, the fragment shader tile image variables decorated with `Location` form an interface with the color attachments defined by the render pass instance. The fragment shader tile image variables are matched by `Location` decorations to the color attachments specified in the `pColorAttachments` array of the VkRenderingInfoKHR structure describing the render pass instance the fragment shader is executed in.

The fragment shader variables listed by `OpEntryPoint` with the `TileImageEXT` storage class and a decoration of `Location` form the *fragment tile image interface*. These variables **must** be declared with a type of `OpTypeImage`, and a `Dim` operand of `TileImageDataEXT`. The `Component` decoration is not supported for these variables.

Reading from a tile image variable with a `Location` decoration of *i* reads from the color attachment identified by the element of VkRenderingInfoKHR::`pColorAttachments` with a `location` equal to *i*. If the tile image variable is declared as an array of size N, it consumes N consecutive tile image locations, starting with the index specified. There **must** not be more than one tile image variable with the same `Location` whether explicitly declared or implied by an array declaration. The number of available tile image locations is the same as the number of available fragment output locations as given by the `maxFragmentOutputAttachments` member of the `VkPhysicalDeviceLimits` structure.

The basic data type (floating-point, integer, unsigned integer) of the tile image variable **must** match the basic format of the corresponding color attachment, or the values read from the tile image variables are undefined.

## 15.6. Fragment Input Attachment Interface

When a fragment stage is present in a pipeline, the fragment shader subpass inputs form an interface with the input attachments of the current subpass. The fragment shader subpass input variables are matched by `InputAttachmentIndex` decorations to the input attachments specified in the `pInputAttachments` array of the VkSubpassDescription structure describing the subpass that the fragment shader is executed in.

The fragment shader subpass input variables with the `UniformConstant` storage class and a decoration of `InputAttachmentIndex` that are statically used by `OpEntryPoint` form the *fragment input attachment interface*. These variables **must** be declared with a type of `OpTypeImage`, a `Dim` operand of `SubpassData`, an `Arrayed` operand of 0, and a `Sampled` operand of 2. The `MS` operand of the `OpTypeImage` **must** be 0 if the `samples` field of the corresponding VkAttachmentDescription is `VK_SAMPLE_COUNT_1_BIT` and multisampled-render-to-single-sampled is not enabled, and 1 otherwise.

A subpass input variable identified with an InputAttachmentIndex decoration of *i* reads from the input attachment indicated by pInputAttachments[*i*] member of VkSubpassDescription. If the subpass input variable is declared as an array of size N, it consumes N consecutive input attachments, starting with the index specified. There **must** not be more than one input variable with the same InputAttachmentIndex whether explicitly declared or implied by an array declaration per image aspect. A multi-aspect image (e.g. a depth/stencil format) **can** use the same input variable. The number of available input attachment indices is given by the maxPerStageDescriptorInputAttachments member of the VkPhysicalDeviceLimits structure.

When using dynamic rendering with the dynamicRenderingLocalRead feature enabled, a subpass input variable with a InputAttachmentIndex decoration of *i* can be mapped to a color, depth, or stencil attachment.

To set the input attachment index mappings during dynamic rendering, call:

```
// Provided by VK_KHR_dynamic_rendering_local_read
void vkCmdSetRenderingInputAttachmentIndicesKHR(
    VkCommandBuffer                             commandBuffer,
    const VkRenderingInputAttachmentIndexInfoKHR* pInputAttachmentIndexInfo);
```

- commandBuffer is the command buffer into which the command will be recorded.
- pInputAttachmentIndexInfo is a VkRenderingInputAttachmentIndexInfoKHR structure indicating the new mappings.

This command sets the input attachment index mappings for subsequent drawing commands, and **must** match the mappings provided to the currently bound pipeline, if one is bound, which **can** be set by chaining VkRenderingInputAttachmentIndexInfoKHR to VkGraphicsPipelineCreateInfo.

Until this command is called, mappings in the command buffer state are treated as each color attachment specified in vkCmdBeginRenderingKHR mapping to subpass inputs with a InputAttachmentIndex equal to its index in VkRenderingInfo::pColorAttachments, and depth/stencil attachments mapping to input attachments without these decorations. This state is reset whenever vkCmdBeginRenderingKHR is called.

### Valid Usage

- VUID-vkCmdSetRenderingInputAttachmentIndicesKHR-dynamicRenderingLocalRead-09516
  dynamicRenderingLocalRead **must** be enabled
- VUID-vkCmdSetRenderingInputAttachmentIndicesKHR-pInputAttachmentIndexInfo-09517
  pInputAttachmentIndexInfo->colorAttachmentCount **must** be equal to the value of VkRenderingInfo::colorAttachmentCount used to begin the current render pass instance
- VUID-vkCmdSetRenderingInputAttachmentIndicesKHR-commandBuffer-09518
  The current render pass instance **must** have been started or resumed by vkCmdBeginRenderingKHR in this commandBuffer

## Valid Usage (Implicit)

- VUID-vkCmdSetRenderingInputAttachmentIndicesKHR-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdSetRenderingInputAttachmentIndicesKHR-pInputAttachmentIndexInfo-parameter
  `pInputAttachmentIndexInfo` **must** be a valid pointer to a valid VkRenderingInputAttachmentIndexInfoKHR structure
- VUID-vkCmdSetRenderingInputAttachmentIndicesKHR-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdSetRenderingInputAttachmentIndicesKHR-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- VUID-vkCmdSetRenderingInputAttachmentIndicesKHR-renderpass
  This command **must** only be called inside of a render pass instance
- VUID-vkCmdSetRenderingInputAttachmentIndicesKHR-videocoding
  This command **must** only be called outside of a video coding scope

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Inside | Outside | Graphics | State |

The `VkRenderingInputAttachmentIndexInfoKHR` structure is defined as:

```
// Provided by VK_KHR_dynamic_rendering_local_read
typedef struct VkRenderingInputAttachmentIndexInfoKHR {
    VkStructureType    sType;
    const void*        pNext;
    uint32_t           colorAttachmentCount;
    const uint32_t*    pColorAttachmentInputIndices;
    const uint32_t*    pDepthInputAttachmentIndex;
    const uint32_t*    pStencilInputAttachmentIndex;
} VkRenderingInputAttachmentIndexInfoKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `colorAttachmentCount` is the number of elements in `pColorAttachmentInputIndices`.
- `pColorAttachmentInputIndices` is a pointer to an array of `colorAttachmentCount` `uint32_t` values defining indices for color attachments to be used as input attachments.
- `pDepthInputAttachmentIndex` is either `NULL`, or a pointer to a `uint32_t` value defining the index for the depth attachment to be used as an input attachment.
- `pStencilInputAttachmentIndex` is either `NULL`, or a pointer to a `uint32_t` value defining the index for the stencil attachment to be used as an input attachment.

This structure allows applications to remap attachments to different input attachment indices.

Each element of `pColorAttachmentInputIndices` set to a value of `VK_ATTACHMENT_UNUSED` indicates that the corresponding attachment will not be used as an input attachment in this pipeline. Any other value in each of those elements will map the corresponding attachment to a `InputAttachmentIndex` value defined in shader code.

If `pColorAttachmentInputIndices` is `NULL`, it is equivalent to setting each element to its index within the array.

If `pDepthInputAttachmentIndex` or `pStencilInputAttachmentIndex` are set to `NULL`, they map to input attachments without a `InputAttachmentIndex` decoration. If they point to a value of `VK_ATTACHMENT_UNUSED`, it indicates that the corresponding attachment will not be used as an input attachment in this pipeline. If they point to any other value it maps the corresponding attachment to a `InputAttachmentIndex` value defined in shader code.

This structure **can** be included in the `pNext` chain of a VkGraphicsPipelineCreateInfo structure to set this state for a pipeline. If this structure is not included in the `pNext` chain of VkGraphicsPipelineCreateInfo, it is equivalent to specifying this structure with the following properties:

- `colorAttachmentCount` set to VkPipelineRenderingCreateInfo::`colorAttachmentCount`.
- `pColorAttachmentInputIndices` set to `NULL`.
- `pDepthInputAttachmentIndex` set to `NULL`.
- `pStencilInputAttachmentIndex` set to `NULL`.

This structure **can** be included in the pNext chain of a VkCommandBufferInheritanceInfo structure to specify inherited state from the primary command buffer. If this structure is not included in the pNext chain of VkCommandBufferInheritanceInfo, it is equivalent to specifying this structure with the following properties:

- colorAttachmentCount set to VkCommandBufferInheritanceRenderingInfo::colorAttachmentCount.
- pColorAttachmentInputIndices set to NULL.
- pDepthInputAttachmentIndex set to NULL.
- pStencilInputAttachmentIndex set to NULL.

## Valid Usage

- VUID-VkRenderingInputAttachmentIndexInfoKHR-dynamicRenderingLocalRead-09519
  If the dynamicRenderingLocalRead feature is not enabled, and pColorAttachmentInputIndices is not NULL, each element **must** be set to VK_ATTACHMENT_UNUSED
- VUID-VkRenderingInputAttachmentIndexInfoKHR-dynamicRenderingLocalRead-09520
  If the dynamicRenderingLocalRead feature is not enabled, pDepthInputAttachmentIndex **must** be a valid pointer to a value of VK_ATTACHMENT_UNUSED
- VUID-VkRenderingInputAttachmentIndexInfoKHR-dynamicRenderingLocalRead-09521
  If the dynamicRenderingLocalRead feature is not enabled, pStencilInputAttachmentIndex **must** be a valid pointer to a value of VK_ATTACHMENT_UNUSED
- VUID-VkRenderingInputAttachmentIndexInfoKHR-pColorAttachmentInputIndices-09522
  Elements of pColorAttachmentInputIndices that are not VK_ATTACHMENT_UNUSED **must** each be unique
- VUID-VkRenderingInputAttachmentIndexInfoKHR-pColorAttachmentInputIndices-09523
  Elements of pColorAttachmentInputIndices that are not VK_ATTACHMENT_UNUSED **must** not take the same value as the content of pDepthInputAttachmentIndex
- VUID-VkRenderingInputAttachmentIndexInfoKHR-pColorAttachmentInputIndices-09524
  Elements of pColorAttachmentInputIndices that are not VK_ATTACHMENT_UNUSED **must** not take the same value as the content of pStencilInputAttachmentIndex
- VUID-VkRenderingInputAttachmentIndexInfoKHR-colorAttachmentCount-09525
  colorAttachmentCount **must** be less than or equal to maxColorAttachments

## Valid Usage (Implicit)

- VUID-VkRenderingInputAttachmentIndexInfoKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_RENDERING_INPUT_ATTACHMENT_INDEX_INFO_KHR
- VUID-VkRenderingInputAttachmentIndexInfoKHR-pColorAttachmentInputIndices-parameter
  If colorAttachmentCount is not 0, and pColorAttachmentInputIndices is not NULL, pColorAttachmentInputIndices **must** be a valid pointer to an array of colorAttachmentCount uint32_t values

- VUID-VkRenderingInputAttachmentIndexInfoKHR-pDepthInputAttachmentIndex-parameter
  If `pDepthInputAttachmentIndex` is not `NULL`, `pDepthInputAttachmentIndex` **must** be a valid pointer to a valid `uint32_t` value
- VUID-VkRenderingInputAttachmentIndexInfoKHR-pStencilInputAttachmentIndex-parameter
  If `pStencilInputAttachmentIndex` is not `NULL`, `pStencilInputAttachmentIndex` **must** be a valid pointer to a valid `uint32_t` value

Variables identified with the `InputAttachmentIndex` **must** only be used by a fragment stage. The numeric format of the subpass input **must** match the format of the corresponding input attachment, or the values of subpass loads from these variables are undefined. If the framebuffer attachment contains both depth and stencil aspects, the numeric format of the subpass input determines if depth or stencil aspect is accessed by the shader.

See Input Attachment for more details.

### 15.6.1. Fragment Input Attachment Compatibility

An input attachment that is statically accessed by a fragment shader **must** be backed by a descriptor that is equivalent to the VkImageView in the VkFramebuffer, except for `subresourceRange.aspectMask`. The `aspectMask` **must** be equal to the aspect accessed by the shader.

## 15.7. Ray Tracing Pipeline Interface

Ray tracing pipelines **may** have more stages than other pipelines with multiple instances of each stage and more dynamic interactions between the stages, but still have interface structures that obey the same general rules as interfaces between shader stages in other pipelines. The three types of inter-stage interface variables for ray tracing pipelines are:

- Ray payloads containing data tracked for the entire lifetime of the ray.
- Hit attributes containing data about a specific hit for the duration of its processing.
- Callable data for passing data into and out of a callable shader.

Ray payloads and callable data are used in explicit shader call instructions, so they have an incoming variant to distinguish the parameter passed to the invocation from any other payloads or data being used by subsequent shader call instructions.

An interface structure used between stages **must** match between the stages using it. Specifically:

- The hit attribute structure read in an any-hit or closest hit shader **must** be the same structure as the hit attribute structure written in the corresponding intersection shader in the same hit group.
- The incoming callable data for a callable shader **must** be the same structure as the callable data referenced by the execute callable instruction in the calling shader.
- The ray payload for a shader invoked by a ray tracing command **must** be the same structure for

all shader stages using the payload for that ray.

Any shader with an incoming ray payload, incoming callable data, or hit attribute **must** only declare one variable of that type.

*Table 20. Ray Pipeline Shader Interface*

| Shader Stage | Ray Payload | Incoming Ray Payload | Hit Attribute | Callable Data | Incoming Callable Data |
|---|---|---|---|---|---|
| Ray Generation | r/w | | | r/w | |
| Intersection | | | r/w | | |
| Any-Hit | | r/w | r | | |
| Closest Hit | r/w | r/w | r | r/w | |
| Miss | r/w | r/w | | r/w | |
| Callable | | | | r/w | r/w |

# 15.8. Shader Resource Interface

When a shader stage accesses buffer or image resources, as described in the Resource Descriptors section, the shader resource variables **must** be matched with the pipeline layout that is provided at pipeline creation time.

The set of shader variables that form the *shader resource interface* for a stage are the variables statically used by that stage's `OpEntryPoint` with a storage class of `Uniform`, `UniformConstant`, `StorageBuffer`, or `PushConstant`. For the fragment shader, this includes the fragment input attachment interface.

The shader resource interface consists of two sub-interfaces: the push constant interface and the descriptor set interface.

## 15.8.1. Push Constant Interface

The shader variables defined with a storage class of `PushConstant` that are statically used by the shader entry points for the pipeline define the *push constant interface*. They **must** be:

- typed as `OpTypeStruct`,
- identified with a `Block` decoration, and
- laid out explicitly using the `Offset`, `ArrayStride`, and `MatrixStride` decorations as specified in Offset and Stride Assignment.

There **must** be no more than one push constant block statically used per shader entry point.

Each statically used member of a push constant block **must** be placed at an `Offset` such that the entire member is entirely contained within the VkPushConstantRange for each `OpEntryPoint` that

uses it, and the stageFlags for that range **must** specify the appropriate VkShaderStageFlagBits for that stage. The Offset decoration for any member of a push constant block **must** not cause the space required for that member to extend outside the range [0, maxPushConstantsSize).

Any member of a push constant block that is declared as an array **must** only be accessed with *dynamically uniform* indices.

### 15.8.2. Descriptor Set Interface

The *descriptor set interface* is comprised of the shader variables with the storage class of StorageBuffer, Uniform or UniformConstant (including the variables in the fragment input attachment interface) that are statically used by the shader entry points for the pipeline.

These variables **must** have DescriptorSet and Binding decorations specified, which are assigned and matched with the VkDescriptorSetLayout objects in the pipeline layout as described in DescriptorSet and Binding Assignment.

The Image Format of an OpTypeImage declaration **must** not be **Unknown**, for variables which are used for OpImageRead, OpImageSparseRead, or OpImageWrite operations, except under the following conditions:

- For OpImageWrite, if the image format is listed in the storage without format list and if the shaderStorageImageWriteWithoutFormat feature is enabled and the shader module declares the StorageImageWriteWithoutFormat capability.
- For OpImageWrite, if the image format supports VK_FORMAT_FEATURE_2_STORAGE_WRITE_WITHOUT_FORMAT_BIT and the shader module declares the StorageImageWriteWithoutFormat capability.
- For OpImageRead or OpImageSparseRead, if the image format is listed in the storage without format list and if the shaderStorageImageReadWithoutFormat feature is enabled and the shader module declares the StorageImageReadWithoutFormat capability.
- For OpImageRead or OpImageSparseRead, if the image format supports VK_FORMAT_FEATURE_2_STORAGE_READ_WITHOUT_FORMAT_BIT and the shader module declares the StorageImageReadWithoutFormat capability.
- For OpImageRead, if Dim is SubpassData (indicating a read from an input attachment).

The Image Format of an OpTypeImage declaration **must** not be **Unknown**, for variables which are used for OpAtomic* operations.

Variables identified with the Uniform storage class are used to access transparent buffer backed resources. Such variables **must** be:

- typed as OpTypeStruct, or an array of this type,
- identified with a Block or BufferBlock decoration, and
- laid out explicitly using the Offset, ArrayStride, and MatrixStride decorations as specified in Offset and Stride Assignment.

Variables identified with the StorageBuffer storage class are used to access transparent buffer

backed resources. Such variables **must** be:

- typed as `OpTypeStruct`, or an array of this type,
- identified with a `Block` decoration, and
- laid out explicitly using the `Offset`, `ArrayStride`, and `MatrixStride` decorations as specified in Offset and Stride Assignment.

The `Offset` decoration for any member of a `Block`-decorated variable in the `Uniform` storage class **must** not cause the space required for that variable to extend outside the range [0, `maxUniformBufferRange`). The `Offset` decoration for any member of a `Block`-decorated variable in the `StorageBuffer` storage class **must** not cause the space required for that variable to extend outside the range [0, `maxStorageBufferRange`).

Variables identified with the `Uniform` storage class **can** also be used to access transparent descriptor set backed resources when the variable is assigned to a descriptor set layout binding with a `descriptorType` of `VK_DESCRIPTOR_TYPE_INLINE_UNIFORM_BLOCK`. In this case the variable **must** be typed as `OpTypeStruct` and **cannot** be aggregated into arrays of that type. Further, the `Offset` decoration for any member of such a variable **must** not cause the space required for that variable to extend outside the range [0,`maxInlineUniformBlockSize`).

Variables identified with a storage class of `UniformConstant` and a decoration of `InputAttachmentIndex` **must** be declared as described in Fragment Input Attachment Interface.

SPIR-V variables decorated with a descriptor set and binding that identify a combined image sampler descriptor **can** have a type of `OpTypeImage`, `OpTypeSampler` (`Sampled`=1), or `OpTypeSampledImage`.

When accessing a resource through such a variable, the resource **must** be selected via compile time constant expressions unless features are enabled to allow dynamically uniform or non-uniform expressions, as described below:

- Storage images (except storage texel buffers and input attachments):
  - Dynamically uniform: `shaderStorageImageArrayDynamicIndexing` and `StorageImageArrayDynamicIndexing`
  - Non-uniform: `shaderStorageImageArrayNonUniformIndexing` and `StorageImageArrayNonUniformIndexing`
- Storage texel buffers:
  - Dynamically uniform: `shaderStorageTexelBufferArrayDynamicIndexing` and `StorageTexelBufferArrayDynamicIndexing`
  - Non-uniform: `shaderStorageTexelBufferArrayNonUniformIndexing` and `StorageTexelBufferArrayNonUniformIndexing`
- Input attachments:
  - Dynamically uniform: `shaderInputAttachmentArrayDynamicIndexing` and `InputAttachmentArrayDynamicIndexing`
  - Non-uniform: `shaderInputAttachmentArrayNonUniformIndexing` and `InputAttachmentArrayNonUniformIndexing`

- Sampled images (except uniform texel buffers), samplers and combined image samplers:
  - Dynamically uniform: `shaderSampledImageArrayDynamicIndexing` and `SampledImageArrayDynamicIndexing`
  - Non-uniform: `shaderSampledImageArrayNonUniformIndexing` and `SampledImageArrayNonUniformIndexing`
- Uniform texel buffers:
  - Dynamically uniform: `shaderUniformTexelBufferArrayDynamicIndexing` and `UniformTexelBufferArrayDynamicIndexing`
  - Non-uniform: `shaderUniformTexelBufferArrayNonUniformIndexing` and `UniformTexelBufferArrayNonUniformIndexing`
- Uniform buffers:
  - Dynamically uniform: `shaderUniformBufferArrayDynamicIndexing` and `UniformBufferArrayDynamicIndexing`
  - Non-uniform: `shaderUniformBufferArrayNonUniformIndexing` and `UniformBufferArrayNonUniformIndexing`
- Storage buffers:
  - Dynamically uniform: `shaderStorageBufferArrayDynamicIndexing` and `StorageBufferArrayDynamicIndexing`
  - Non-uniform: `shaderStorageBufferArrayNonUniformIndexing` and `StorageBufferArrayNonUniformIndexing`
- Acceleration structures:
  - Dynamically uniform: Always supported.
  - Non-uniform: Always supported.
- weight image:
  - Dynamically uniform: Always supported.
  - Non-uniform: Never supported.
- Block matching image:
  - Dynamically uniform: Always supported.
  - Non-uniform: Never supported.

*Example 1. Note*

> Implementations must take care when implementing this if subgroups are not necessarily a subset of the invocation group for their hardware (e.g. if multiple draw calls are packed together). If such an implementation needs uniformity across the subgroup for any resource access, the implementation is responsible for ensuring that indices that are only dynamically uniform across the invocation group still function as expected if they advertise the respective feature.

A combined image sampler in an array that enables sampler $Y'C_BC_R$ conversion or samples a

subsampled image **must** only be indexed by constant integral expressions.

*Table 21. Shader Resource and Descriptor Type Correspondence*

| Resource type | Descriptor Type |
|---|---|
| sampler | VK_DESCRIPTOR_TYPE_SAMPLER or VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER |
| sampled image | VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE or VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER |
| storage image | VK_DESCRIPTOR_TYPE_STORAGE_IMAGE |
| combined image sampler | VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER |
| uniform texel buffer | VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER |
| storage texel buffer | VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER |
| uniform buffer | VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER or VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC |
| storage buffer | VK_DESCRIPTOR_TYPE_STORAGE_BUFFER or VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC |
| input attachment | VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT |
| inline uniform block | VK_DESCRIPTOR_TYPE_INLINE_UNIFORM_BLOCK |
| acceleration structure | VK_DESCRIPTOR_TYPE_ACCELERATION_STRUCTURE_KHR or VK_DESCRIPTOR_TYPE_ACCELERATION_STRUCTURE_NV |
| weight image | VK_DESCRIPTOR_TYPE_SAMPLE_WEIGHT_IMAGE_QCOM |
| block matching image | VK_DESCRIPTOR_TYPE_BLOCK_MATCH_IMAGE_QCOM |

*Table 22. Shader Resource and Storage Class Correspondence*

| Resource type | Storage Class | Type[1] | Decoration(s)[2] |
|---|---|---|---|
| sampler | UniformConstant | OpTypeSampler | |
| sampled image | UniformConstant | OpTypeImage (Sampled=1) | |
| storage image | UniformConstant | OpTypeImage (Sampled=2) | |
| combined image sampler | UniformConstant | OpTypeSampledImage<br>OpTypeImage (Sampled=1)<br>OpTypeSampler | |
| uniform texel buffer | UniformConstant | OpTypeImage (Dim=Buffer, Sampled=1) | |
| storage texel buffer | UniformConstant | OpTypeImage (Dim=Buffer, Sampled=2) | |
| uniform buffer | Uniform | OpTypeStruct | Block, Offset, (ArrayStride), (MatrixStride) |

| Resource type | Storage Class | Type[1] | Decoration(s)[2] |
|---|---|---|---|
| storage buffer | Uniform | OpTypeStruct | BufferBlock, Offset, (ArrayStride), (MatrixStride) |
| | StorageBuffer | | Block, Offset, (ArrayStride), (MatrixStride) |
| input attachment | UniformConstant | OpTypeImage (Dim =SubpassData, Sampled=2) | InputAttachmentIndex |
| inline uniform block | Uniform | OpTypeStruct | Block, Offset, (ArrayStride), (MatrixStride) |
| acceleration structure | UniformConstant | OpTypeAccelerationStructureKHR | |
| sample weight image | UniformConstant | OpTypeImage (Depth=0, Dim =2D, Arrayed=1, MS=0, Sampled=1) | WeightTextureQCOM |
| block matching image | UniformConstant | OpTypeImage (Depth=0, Dim =2D, Arrayed=0, MS=0, Sampled=1) | BlockMatchTextureQCOM |

**1**

Where OpTypeImage is referenced, the Dim values Buffer and Subpassdata are only accepted where they are specifically referenced. They do not correspond to resource types where a generic OpTypeImage is specified.

**2**

In addition to DescriptorSet and Binding.

### 15.8.3. DescriptorSet and Binding Assignment

A variable decorated with a DescriptorSet decoration of s and a Binding decoration of b indicates that this variable is associated with the VkDescriptorSetLayoutBinding that has a binding equal to b in pSetLayouts[*s*] that was specified in VkPipelineLayoutCreateInfo.

DescriptorSet decoration values **must** be between zero and maxBoundDescriptorSets minus one, inclusive. Binding decoration values **can** be any 32-bit unsigned integer value, as described in Descriptor Set Layout. Each descriptor set has its own binding name space.

If the Binding decoration is used with an array, the entire array is assigned that binding value. The array **must** be a single-dimensional array and size of the array **must** be no larger than the number of descriptors in the binding. If the array is runtime-sized, then array elements greater than or equal to the size of that binding in the bound descriptor set **must** not be used. If the array is runtime-sized, the runtimeDescriptorArray feature **must** be enabled and the RuntimeDescriptorArray capability **must** be declared. The index of each element of the array is referred to as the *arrayElement*. For the purposes of interface matching and descriptor set operations, if a resource

variable is not an array, it is treated as if it has an arrayElement of zero.

There is a limit on the number of resources of each type that **can** be accessed by a pipeline stage as shown in Shader Resource Limits. The “Resources Per Stage” column gives the limit on the number each type of resource that **can** be statically used for an entry point in any given stage in a pipeline. The “Resource Types” column lists which resource types are counted against the limit. Some resource types count against multiple limits. The `VK_DESCRIPTOR_TYPE_MUTABLE_EXT` descriptor type counts as one individual resource and one for every unique resource limit per descriptor set type that is present in the associated binding’s VkMutableDescriptorTypeListEXT. If multiple descriptor types in VkMutableDescriptorTypeListEXT map to the same resource limit, only one descriptor is consumed for purposes of computing resource limits.

The pipeline layout **may** include descriptor sets and bindings which are not referenced by any variables statically used by the entry points for the shader stages in the binding’s `stageFlags`.

However, if a variable assigned to a given `DescriptorSet` and `Binding` is statically used by the entry point for a shader stage, the pipeline layout **must** contain a descriptor set layout binding in that descriptor set layout and for that binding number, and that binding’s `stageFlags` **must** include the appropriate VkShaderStageFlagBits for that stage. The variable **must** be of a valid resource type determined by its SPIR-V type and storage class, as defined in Shader Resource and Storage Class Correspondence. The descriptor set layout binding **must** be of a corresponding descriptor type, as defined in Shader Resource and Descriptor Type Correspondence.

**NOTE**

There are no limits on the number of shader variables that can have overlapping set and binding values in a shader; but which resources are statically used has an impact. If any shader variable identifying a resource is statically used in a shader, then the underlying descriptor bound at the declared set and binding **must** support the declared type in the shader when the shader executes.

If multiple shader variables are declared with the same set and binding values, and with the same underlying descriptor type, they can all be statically used within the same shader. However, accesses are not automatically synchronized, and `Aliased` decorations should be used to avoid data hazards (see section 2.18.2 Aliasing in the SPIR-V specification).

If multiple shader variables with the same set and binding values are declared in a single shader, but with different declared types, where any of those are not supported by the relevant bound descriptor, that shader can only be executed if the variables with the unsupported type are not statically used.

A noteworthy example of using multiple statically-used shader variables sharing the same descriptor set and binding values is a descriptor of type `VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER` that has multiple corresponding shader variables in the `UniformConstant` storage class, where some could be `OpTypeImage` (`Sampled`=1), some could be `OpTypeSampler`, and some could be `OpTypeSampledImage`.

*Table 23. Shader Resource Limits*

| Resources per Stage | Resource Types |
|---|---|
| maxPerStageDescriptorSamplers or maxPerStageDescriptorUpdateAfterBindSamplers | sampler |
| | combined image sampler |
| maxPerStageDescriptorSampledImages or maxPerStageDescriptorUpdateAfterBindSampledImages | sampled image |
| | combined image sampler |
| | uniform texel buffer |
| | sample weight image |
| | block matching image |
| maxPerStageDescriptorStorageImages or maxPerStageDescriptorUpdateAfterBindStorageImages | storage image |
| | storage texel buffer |
| maxPerStageDescriptorUniformBuffers or maxPerStageDescriptorUpdateAfterBindUniformBuffers | uniform buffer |
| | uniform buffer dynamic |
| maxPerStageDescriptorStorageBuffers or maxPerStageDescriptorUpdateAfterBindStorageBuffers | storage buffer |
| | storage buffer dynamic |
| maxPerStageDescriptorInputAttachments or maxPerStageDescriptorUpdateAfterBindInputAttachments | input attachment[1] |
| maxPerStageDescriptorInlineUniformBlocks or maxPerStageDescriptorUpdateAfterBindInlineUniformBlocks | inline uniform block |
| VkPhysicalDeviceRayTracingPropertiesNV::maxDescriptorSetAccelerationStructures or maxPerStageDescriptorAccelerationStructures or maxPerStageDescriptorUpdateAfterBindAccelerationStructures | acceleration structure |

**1**

Input attachments **can** only be used in the fragment shader stage

## 15.8.4. Offset and Stride Assignment

Certain objects **must** be explicitly laid out using the Offset, ArrayStride, and MatrixStride, as described in SPIR-V explicit layout validation rules. All such layouts also **must** conform to the following requirements.

| | |
|---|---|
| **NOTE** | The numeric order of Offset decorations does not need to follow member declaration order. |

**Alignment Requirements**

There are different alignment requirements depending on the specific resources and on the

features enabled on the device.

Matrix types are defined in terms of arrays as follows:

- A column-major matrix with C columns and R rows is equivalent to a C element array of vectors with R components.
- A row-major matrix with C columns and R rows is equivalent to an R element array of vectors with C components.

The *scalar alignment* of the type of an `OpTypeStruct` member is defined recursively as follows:

- A scalar of size N has a scalar alignment of N.
- A vector type has a scalar alignment equal to that of its component type.
- An array type has a scalar alignment equal to that of its element type.
- A structure has a scalar alignment equal to the largest scalar alignment of any of its members.
- A matrix type inherits *scalar alignment* from the equivalent array declaration.

The *base alignment* of the type of an `OpTypeStruct` member is defined recursively as follows:

- A scalar has a base alignment equal to its scalar alignment.
- A two-component vector has a base alignment equal to twice its scalar alignment.
- A three- or four-component vector has a base alignment equal to four times its scalar alignment.
- An array has a base alignment equal to the base alignment of its element type.
- A structure has a base alignment equal to the largest base alignment of any of its members. An empty structure has a base alignment equal to the size of the smallest scalar type permitted by the capabilities declared in the SPIR-V module. (e.g., for a 1 byte aligned empty struct in the `StorageBuffer` storage class, `StorageBuffer8BitAccess` or `UniformAndStorageBuffer8BitAccess` **must** be declared in the SPIR-V module.)
- A matrix type inherits *base alignment* from the equivalent array declaration.

The *extended alignment* of the type of an `OpTypeStruct` member is similarly defined as follows:

- A scalar or vector type has an extended alignment equal to its base alignment.
- An array or structure type has an extended alignment equal to the largest extended alignment of any of its members, rounded up to a multiple of 16.
- A matrix type inherits extended alignment from the equivalent array declaration.

A member is defined to *improperly straddle* if either of the following are true:

- It is a vector with total size less than or equal to 16 bytes, and has `Offset` decorations placing its first byte at F and its last byte at L, where floor(F / 16) != floor(L / 16).
- It is a vector with total size greater than 16 bytes and has its `Offset` decorations placing its first byte at a non-integer multiple of 16.

**Standard Buffer Layout**

Every member of an `OpTypeStruct` that is required to be explicitly laid out **must** be aligned according to the first matching rule as follows. If the struct is contained in pointer types of multiple storage classes, it **must** satisfy the requirements for every storage class used to reference it.

1. If the `scalarBlockLayout` feature is enabled on the device and the storage class is `Uniform`, `StorageBuffer`, `PhysicalStorageBuffer`, `ShaderRecordBufferKHR`, or `PushConstant` then every member **must** be aligned according to its scalar alignment.
2. If the `workgroupMemoryExplicitLayoutScalarBlockLayout` feature is enabled on the device and the storage class is `Workgroup` then every member **must** be aligned according to its scalar alignment.
3. All vectors **must** be aligned according to their scalar alignment.
4. If the `uniformBufferStandardLayout` feature is not enabled on the device, then any member of an `OpTypeStruct` with a storage class of `Uniform` and a decoration of `Block` **must** be aligned according to its extended alignment.
5. Every other member **must** be aligned according to its base alignment.

| | |
|---|---|
| **NOTE** | Even if scalar alignment is supported, it is generally more performant to use the *base alignment*. |

The memory layout **must** obey the following rules:

- The `Offset` decoration of any member **must** be a multiple of its alignment.
- Any `ArrayStride` or `MatrixStride` decoration **must** be a multiple of the alignment of the array or matrix as defined above.

If one of the conditions below applies

- The storage class is `Uniform`, `StorageBuffer`, `PhysicalStorageBuffer`, `ShaderRecordBufferKHR`, or `PushConstant`, and the `scalarBlockLayout` feature is not enabled on the device.
- The storage class is `Workgroup`, and either the struct member is not part of a `Block` or the `workgroupMemoryExplicitLayoutScalarBlockLayout` feature is not enabled on the device.
- The storage class is any other storage class.

the memory layout **must** also obey the following rules:

- Vectors **must** not improperly straddle, as defined above.
- The `Offset` decoration of a member **must** not place it between the end of a structure, an array or a matrix and the next multiple of the alignment of that structure, array or matrix.

| | |
|---|---|
| **NOTE** | The **std430 layout** in GLSL satisfies these rules for types using the base alignment. The **std140 layout** satisfies the rules for types using the extended alignment. |

## 15.9. Built-In Variables

Built-in variables are accessed in shaders by declaring a variable decorated with a `BuiltIn` SPIR-V decoration. The meaning of each `BuiltIn` decoration is as follows. In the remainder of this section,

the name of a built-in is used interchangeably with a term equivalent to a variable decorated with that particular built-in. Built-ins that represent integer values **can** be declared as either signed or unsigned 32-bit integers.

As mentioned above, some inputs and outputs have an additional level of arrayness relative to other shader inputs and outputs. This level of arrayness is not included in the type descriptions below, but **must** be included when declaring the built-in.

Any two `Input` storage class `OpVariable` declarations listed as operands on the same `OpEntryPoint` **must** not have the same `BuiltIn` decoration. Any two `Output` storage class `OpVariable` declarations listed as operands on the same `OpEntryPoint` **must** not have the same `BuiltIn` decoration.

#### `BaryCoordKHR`

The `BaryCoordKHR` decoration **can** be used to decorate a fragment shader input variable. This variable will contain a three-component floating-point vector with barycentric weights that indicate the location of the fragment relative to the screen-space locations of vertices of its primitive, obtained using perspective interpolation.

### Valid Usage

- VUID-BaryCoordKHR-BaryCoordKHR-04154
  The `BaryCoordKHR` decoration **must** be used only within the `Fragment Execution Model`
- VUID-BaryCoordKHR-BaryCoordKHR-04155
  The variable decorated with `BaryCoordKHR` **must** be declared using the `Input Storage Class`
- VUID-BaryCoordKHR-BaryCoordKHR-04156
  The variable decorated with `BaryCoordKHR` **must** be declared as a three-component vector of 32-bit floating-point values

#### `BaryCoordNoPerspAMD`

The `BaryCoordNoPerspAMD` decoration **can** be used to decorate a fragment shader input variable. This variable will contain the (I,J) pair of the barycentric coordinates corresponding to the fragment evaluated using linear interpolation at the fragment's center. The K coordinate of the barycentric coordinates **can** be derived given the identity I + J + K = 1.0.

### Valid Usage

- VUID-BaryCoordNoPerspAMD-BaryCoordNoPerspAMD-04157
  The `BaryCoordNoPerspAMD` decoration **must** be used only within the `Fragment Execution Model`
- VUID-BaryCoordNoPerspAMD-BaryCoordNoPerspAMD-04158
  The variable decorated with `BaryCoordNoPerspAMD` **must** be declared using the `Input Storage Class`
- VUID-BaryCoordNoPerspAMD-BaryCoordNoPerspAMD-04159
  The variable decorated with `BaryCoordNoPerspAMD` **must** be declared as a two-component vector of 32-bit floating-point values

`BaryCoordNoPerspKHR`

The `BaryCoordNoPerspKHR` decoration **can** be used to decorate a fragment shader input variable. This variable will contain a three-component floating-point vector with barycentric weights that indicate the location of the fragment relative to the screen-space locations of vertices of its primitive, obtained using linear interpolation.

### Valid Usage

- VUID-BaryCoordNoPerspKHR-BaryCoordNoPerspKHR-04160
  The `BaryCoordNoPerspKHR` decoration **must** be used only within the `Fragment Execution Model`
- VUID-BaryCoordNoPerspKHR-BaryCoordNoPerspKHR-04161
  The variable decorated with `BaryCoordNoPerspKHR` **must** be declared using the `Input Storage Class`
- VUID-BaryCoordNoPerspKHR-BaryCoordNoPerspKHR-04162
  The variable decorated with `BaryCoordNoPerspKHR` **must** be declared as a three-component vector of 32-bit floating-point values

`BaryCoordNoPerspCentroidAMD`

The `BaryCoordNoPerspCentroidAMD` decoration **can** be used to decorate a fragment shader input variable. This variable will contain the (I,J) pair of the barycentric coordinates corresponding to the fragment evaluated using linear interpolation at the centroid. The K coordinate of the barycentric coordinates **can** be derived given the identity I + J + K = 1.0.

### Valid Usage

- VUID-BaryCoordNoPerspCentroidAMD-BaryCoordNoPerspCentroidAMD-04163
  The `BaryCoordNoPerspCentroidAMD` decoration **must** be used only within the `Fragment Execution Model`
- VUID-BaryCoordNoPerspCentroidAMD-BaryCoordNoPerspCentroidAMD-04164
  The variable decorated with `BaryCoordNoPerspCentroidAMD` **must** be declared using the `Input Storage Class`
- VUID-BaryCoordNoPerspCentroidAMD-BaryCoordNoPerspCentroidAMD-04165
  The variable decorated with `BaryCoordNoPerspCentroidAMD` **must** be declared as a three-component vector of 32-bit floating-point values

`BaryCoordNoPerspSampleAMD`

The `BaryCoordNoPerspSampleAMD` decoration **can** be used to decorate a fragment shader input variable. This variable will contain the (I,J) pair of the barycentric coordinates corresponding to the fragment evaluated using linear interpolation at each covered sample. The K coordinate of the barycentric coordinates **can** be derived given the identity I + J + K = 1.0.

### Valid Usage

- VUID-BaryCoordNoPerspSampleAMD-BaryCoordNoPerspSampleAMD-04166
  The `BaryCoordNoPerspSampleAMD` decoration **must** be used only within the `Fragment Execution Model`
- VUID-BaryCoordNoPerspSampleAMD-BaryCoordNoPerspSampleAMD-04167
  The variable decorated with `BaryCoordNoPerspSampleAMD` **must** be declared using the `Input Storage Class`
- VUID-BaryCoordNoPerspSampleAMD-BaryCoordNoPerspSampleAMD-04168
  The variable decorated with `BaryCoordNoPerspSampleAMD` **must** be declared as a two-component vector of 32-bit floating-point values

### `BaryCoordPullModelAMD`

The `BaryCoordPullModelAMD` decoration **can** be used to decorate a fragment shader input variable. This variable will contain (1/W, 1/I, 1/J) evaluated at the fragment center and **can** be used to calculate gradients and then interpolate I, J, and W at any desired sample location.

#### Valid Usage

- VUID-BaryCoordPullModelAMD-BaryCoordPullModelAMD-04169
  The `BaryCoordPullModelAMD` decoration **must** be used only within the `Fragment Execution Model`
- VUID-BaryCoordPullModelAMD-BaryCoordPullModelAMD-04170
  The variable decorated with `BaryCoordPullModelAMD` **must** be declared using the `Input Storage Class`
- VUID-BaryCoordPullModelAMD-BaryCoordPullModelAMD-04171
  The variable decorated with `BaryCoordPullModelAMD` **must** be declared as a three-component vector of 32-bit floating-point values

### `BaryCoordSmoothAMD`

The `BaryCoordSmoothAMD` decoration **can** be used to decorate a fragment shader input variable. This variable will contain the (I,J) pair of the barycentric coordinates corresponding to the fragment evaluated using perspective interpolation at the fragment's center. The K coordinate of the barycentric coordinates **can** be derived given the identity I + J + K = 1.0.

#### Valid Usage

- VUID-BaryCoordSmoothAMD-BaryCoordSmoothAMD-04172
  The `BaryCoordSmoothAMD` decoration **must** be used only within the `Fragment Execution Model`
- VUID-BaryCoordSmoothAMD-BaryCoordSmoothAMD-04173
  The variable decorated with `BaryCoordSmoothAMD` **must** be declared using the `Input Storage Class`
- VUID-BaryCoordSmoothAMD-BaryCoordSmoothAMD-04174
  The variable decorated with `BaryCoordSmoothAMD` **must** be declared as a two-component vector of 32-bit floating-point values

`BaryCoordSmoothCentroidAMD`

The `BaryCoordSmoothCentroidAMD` decoration **can** be used to decorate a fragment shader input variable. This variable will contain the (I,J) pair of the barycentric coordinates corresponding to the fragment evaluated using perspective interpolation at the centroid. The K coordinate of the barycentric coordinates **can** be derived given the identity I + J + K = 1.0.

### Valid Usage

- VUID-BaryCoordSmoothCentroidAMD-BaryCoordSmoothCentroidAMD-04175
  The `BaryCoordSmoothCentroidAMD` decoration **must** be used only within the `Fragment Execution Model`
- VUID-BaryCoordSmoothCentroidAMD-BaryCoordSmoothCentroidAMD-04176
  The variable decorated with `BaryCoordSmoothCentroidAMD` **must** be declared using the `Input Storage Class`
- VUID-BaryCoordSmoothCentroidAMD-BaryCoordSmoothCentroidAMD-04177
  The variable decorated with `BaryCoordSmoothCentroidAMD` **must** be declared as a two-component vector of 32-bit floating-point values

`BaryCoordSmoothSampleAMD`

The `BaryCoordSmoothSampleAMD` decoration **can** be used to decorate a fragment shader input variable. This variable will contain the (I,J) pair of the barycentric coordinates corresponding to the fragment evaluated using perspective interpolation at each covered sample. The K coordinate of the barycentric coordinates **can** be derived given the identity I + J + K = 1.0.

### Valid Usage

- VUID-BaryCoordSmoothSampleAMD-BaryCoordSmoothSampleAMD-04178
  The `BaryCoordSmoothSampleAMD` decoration **must** be used only within the `Fragment Execution Model`
- VUID-BaryCoordSmoothSampleAMD-BaryCoordSmoothSampleAMD-04179
  The variable decorated with `BaryCoordSmoothSampleAMD` **must** be declared using the `Input Storage Class`
- VUID-BaryCoordSmoothSampleAMD-BaryCoordSmoothSampleAMD-04180
  The variable decorated with `BaryCoordSmoothSampleAMD` **must** be declared as a two-component vector of 32-bit floating-point values

`BaseInstance`

Decorating a variable with the `BaseInstance` built-in will make that variable contain the integer value corresponding to the first instance that was passed to the command that invoked the current vertex shader invocation. `BaseInstance` is the `firstInstance` parameter to a *direct drawing command* or the `firstInstance` member of a structure consumed by an *indirect drawing command*.

### Valid Usage

- VUID-BaseInstance-BaseInstance-04181
  The `BaseInstance` decoration **must** be used only within the `Vertex Execution Model`
- VUID-BaseInstance-BaseInstance-04182
  The variable decorated with `BaseInstance` **must** be declared using the `Input Storage Class`
- VUID-BaseInstance-BaseInstance-04183
  The variable decorated with `BaseInstance` **must** be declared as a scalar 32-bit integer value

`BaseVertex`

Decorating a variable with the `BaseVertex` built-in will make that variable contain the integer value corresponding to the first vertex or vertex offset that was passed to the command that invoked the current vertex shader invocation. For *non-indexed drawing commands*, this variable is the `firstVertex` parameter to a *direct drawing command* or the `firstVertex` member of the structure consumed by an *indirect drawing command*. For *indexed drawing commands*, this variable is the `vertexOffset` parameter to a *direct drawing command* or the `vertexOffset` member of the structure consumed by an *indirect drawing command*.

### Valid Usage

- VUID-BaseVertex-BaseVertex-04184
  The `BaseVertex` decoration **must** be used only within the `Vertex Execution Model`
- VUID-BaseVertex-BaseVertex-04185
  The variable decorated with `BaseVertex` **must** be declared using the `Input Storage Class`
- VUID-BaseVertex-BaseVertex-04186
  The variable decorated with `BaseVertex` **must** be declared as a scalar 32-bit integer value

`ClipDistance`

Decorating a variable with the `ClipDistance` built-in decoration will make that variable contain the mechanism for controlling user clipping. `ClipDistance` is an array such that the $i^{th}$ element of the array specifies the clip distance for plane i. A clip distance of 0 means the vertex is on the plane, a positive distance means the vertex is inside the clip half-space, and a negative distance means the vertex is outside the clip half-space.

| | |
|---|---|
| **NOTE** | The array variable decorated with `ClipDistance` is explicitly sized by the shader. |
| **NOTE** | In the last pre-rasterization shader stage, these values will be linearly interpolated across the primitive and the portion of the primitive with interpolated distances less than 0 will be considered outside the clip volume. If `ClipDistance` is then used by a fragment shader, `ClipDistance` contains these linearly interpolated values. |

### Valid Usage

- VUID-ClipDistance-ClipDistance-04187
  The `ClipDistance` decoration **must** be used only within the `MeshEXT`, `MeshNV`, `Vertex`, `Fragment`, `TessellationControl`, `TessellationEvaluation`, or `Geometry` `Execution Model`
- VUID-ClipDistance-ClipDistance-04188
  The variable decorated with `ClipDistance` within the `MeshEXT`, `MeshNV`, or `Vertex` `Execution Model` **must** be declared using the `Output` `Storage Class`
- VUID-ClipDistance-ClipDistance-04189
  The variable decorated with `ClipDistance` within the `Fragment` `Execution Model` **must** be declared using the `Input` `Storage Class`
- VUID-ClipDistance-ClipDistance-04190
  The variable decorated with `ClipDistance` within the `TessellationControl`, `TessellationEvaluation`, or `Geometry` `Execution Model` **must** not be declared in a `Storage Class` other than `Input` or `Output`
- VUID-ClipDistance-ClipDistance-04191
  The variable decorated with `ClipDistance` **must** be declared as an array of 32-bit floating-point values

#### `ClipDistancePerViewNV`

Decorating a variable with the `ClipDistancePerViewNV` built-in decoration will make that variable contain the per-view clip distances. The per-view clip distances have the same semantics as `ClipDistance`.

### Valid Usage

- VUID-ClipDistancePerViewNV-ClipDistancePerViewNV-04192
  The `ClipDistancePerViewNV` decoration **must** be used only within the `MeshNV` `Execution Model`
- VUID-ClipDistancePerViewNV-ClipDistancePerViewNV-04193
  The variable decorated with `ClipDistancePerViewNV` **must** be declared using the `Output` `Storage Class`
- VUID-ClipDistancePerViewNV-ClipDistancePerViewNV-04194
  The variable decorated with `ClipDistancePerViewNV` **must** also be decorated with the `PerViewNV` decoration
- VUID-ClipDistancePerViewNV-ClipDistancePerViewNV-04195
  The variable decorated with `ClipDistancePerViewNV` **must** be declared as a two-dimensional array of 32-bit floating-point values

#### `ClusterIDHUAWEI`

The `ClusterIDHUAWEI` decoration can be used to decorate a cluster culling shader output variable,this variable will contain an integer value that specifies the id of cluster being rendered by this drawing command. When Cluster Culling Shader enable, `ClusterIDHUAWEI` will replace gl_DrawID pass to vertex shader for cluster-related information fetching.

### Valid Usage

- VUID-ClusterIDHUAWEI-ClusterIDHUAWEI-07797
  The `ClusterIDHUAWEI` decoration **must** be used only within the `ClusterCullingHUAWEI` `Execution Model`
- VUID-ClusterIDHUAWEI-ClusterIDHUAWEI-07798
  The variable decorated with `ClusterIDHUAWEI` **must** be declared as a scalar 32-bit integer value

`ClusterShadingRateHUAWEI`

The `ClusterShadingRateHUAWEI` decoration can be used to decorate a cluster culling shader output variable. This variable will contain an integer value specifying the shading rate of a rendering cluster.

### Valid Usage

- VUID-ClusterShadingRateHUAWEI-ClusterShadingRateHUAWEI-09448
  The `ClusterShadingRateHUAWEI` decoration **must** be used only within the `ClusterCullingHUAWEI` `Execution Model`
- VUID-ClusterShadingRateHUAWEI-ClusterShadingRateHUAWEI-09449
  The variable decorated with `ClusterShadingRateHUAWEI` **must** be declared as a scalar 32-bit integer value

`CullDistance`

Decorating a variable with the `CullDistance` built-in decoration will make that variable contain the mechanism for controlling user culling. If any member of this array is assigned a negative value for all vertices belonging to a primitive, then the primitive is discarded before rasterization.

| | |
|---|---|
| **NOTE** | In fragment shaders, the values of the `CullDistance` array are linearly interpolated across each primitive. |
| **NOTE** | If `CullDistance` decorates an input variable, that variable will contain the corresponding value from the `CullDistance` decorated output variable from the previous shader stage. |

### Valid Usage

- VUID-CullDistance-CullDistance-04196
  The `CullDistance` decoration **must** be used only within the `MeshEXT`, `MeshNV`, `Vertex`, `Fragment`, `TessellationControl`, `TessellationEvaluation`, or `Geometry` `Execution Model`
- VUID-CullDistance-CullDistance-04197
  The variable decorated with `CullDistance` within the `MeshEXT`, `MeshNV` or `Vertex` `Execution`

Model **must** be declared using the Output Storage Class

- VUID-CullDistance-CullDistance-04198
  The variable decorated with CullDistance within the Fragment Execution Model **must** be declared using the Input Storage Class
- VUID-CullDistance-CullDistance-04199
  The variable decorated with CullDistance within the TessellationControl, TessellationEvaluation, or Geometry Execution Model **must** not be declared using a Storage Class other than Input or Output
- VUID-CullDistance-CullDistance-04200
  The variable decorated with CullDistance **must** be declared as an array of 32-bit floating-point values

### CullDistancePerViewNV

Decorating a variable with the CullDistancePerViewNV built-in decoration will make that variable contain the per-view cull distances. The per-view cull distances have the same semantics as CullDistance.

#### Valid Usage

- VUID-CullDistancePerViewNV-CullDistancePerViewNV-04201
  The CullDistancePerViewNV decoration **must** be used only within the MeshNV Execution Model
- VUID-CullDistancePerViewNV-CullDistancePerViewNV-04202
  The variable decorated with CullDistancePerViewNV **must** be declared using the Output Storage Class
- VUID-CullDistancePerViewNV-CullDistancePerViewNV-04203
  The variable decorated with CullDistancePerViewNV **must** also be decorated with the PerViewNV decoration
- VUID-CullDistancePerViewNV-CullDistancePerViewNV-04204
  The variable decorated with CullDistancePerViewNV **must** be declared as a two-dimensional array of 32-bit floating-point values

### CullPrimitiveEXT

Decorating a variable with the CullPrimitiveEXT built-in decoration will make that variable contain the culling state of output primitives. If the per-primitive boolean value is true, the primitive will be culled, if it is false it will not be culled.

#### Valid Usage

- VUID-CullPrimitiveEXT-CullPrimitiveEXT-07034
  The CullPrimitiveEXT decoration **must** be used only within the MeshEXT Execution Model
- VUID-CullPrimitiveEXT-CullPrimitiveEXT-07035
  The variable decorated with CullPrimitiveEXT **must** be declared using the Output Storage Class

- VUID-CullPrimitiveEXT-CullPrimitiveEXT-07036
  The variable decorated with `CullPrimitiveEXT` **must** be declared as an array of boolean values
- VUID-CullPrimitiveEXT-CullPrimitiveEXT-07037
  The size of the array decorated with `CullPrimitiveEXT` **must** match the value specified by `OutputPrimitivesEXT`
- VUID-CullPrimitiveEXT-CullPrimitiveEXT-07038
  The variable decorated with `CullPrimitiveEXT` within the `MeshEXT Execution Model` **must** also be decorated with the `PerPrimitiveEXT` decoration

`CullMaskKHR`

A variable decorated with the `CullMaskKHR` decoration will specify the cull mask of the ray being processed. The value is given by the `Cull Mask` parameter passed into one of the `OpTrace*` instructions.

### Valid Usage

- VUID-CullMaskKHR-CullMaskKHR-06735
  The `CullMaskKHR` decoration **must** be used only within the `IntersectionKHR`, `AnyHitKHR`, `ClosestHitKHR`, or `MissKHR Execution Model`
- VUID-CullMaskKHR-CullMaskKHR-06736
  The variable decorated with `CullMaskKHR` **must** be declared using the `Input Storage Class`
- VUID-CullMaskKHR-CullMaskKHR-06737
  The variable decorated with `CullMaskKHR` **must** be declared as a scalar 32-bit integer value

`CurrentRayTimeNV`

A variable decorated with the `CurrentRayTimeNV` decoration contains the time value passed in to `OpTraceRayMotionNV` which called this shader.

### Valid Usage

- VUID-CurrentRayTimeNV-CurrentRayTimeNV-04942
  The `CurrentRayTimeNV` decoration **must** be used only within the `IntersectionKHR`, `AnyHitKHR`, `ClosestHitKHR`, or `MissKHR Execution Model`
- VUID-CurrentRayTimeNV-CurrentRayTimeNV-04943
  The variable decorated with `CurrentRayTimeNV` **must** be declared using the `Input Storage Class`
- VUID-CurrentRayTimeNV-CurrentRayTimeNV-04944
  The variable decorated with `CurrentRayTimeNV` **must** be declared as a scalar 32-bit floating-point value

`DeviceIndex`

The `DeviceIndex` decoration **can** be applied to a shader input which will be filled with the device index of the physical device that is executing the current shader invocation. This value will be in the range $[0, max(1, physicalDeviceCount))$, where physicalDeviceCount is the `physicalDeviceCount` member of VkDeviceGroupDeviceCreateInfo.

### Valid Usage

- VUID-DeviceIndex-DeviceIndex-04205
  The variable decorated with `DeviceIndex` **must** be declared using the `Input Storage Class`
- VUID-DeviceIndex-DeviceIndex-04206
  The variable decorated with `DeviceIndex` **must** be declared as a scalar 32-bit integer value

`DrawIndex`

Decorating a variable with the `DrawIndex` built-in will make that variable contain the integer value corresponding to the zero-based index of the draw that invoked the current task, mesh, or vertex shader invocation. For *indirect drawing commands*, `DrawIndex` begins at zero and increments by one for each draw executed. The number of draws is given by the `drawCount` parameter. For *direct drawing commands*, if vkCmdDrawMultiEXT or vkCmdDrawMultiIndexedEXT is used, this variable contains the integer value corresponding to the zero-based index of the draw. Otherwise `DrawIndex` is always zero. `DrawIndex` is dynamically uniform.

When task or mesh shaders are used, only the first active stage will have proper access to the variable. The value read by other stages is undefined.

### Valid Usage

- VUID-DrawIndex-DrawIndex-04207
  The `DrawIndex` decoration **must** be used only within the `Vertex`, `MeshEXT`, `TaskEXT`, `MeshNV`, or `TaskNV Execution Model`
- VUID-DrawIndex-DrawIndex-04208
  The variable decorated with `DrawIndex` **must** be declared using the `Input Storage Class`
- VUID-DrawIndex-DrawIndex-04209
  The variable decorated with `DrawIndex` **must** be declared as a scalar 32-bit integer value

`FirstIndexHUAWEI`

The `FirstIndexHUAWEI` decoration can be used to decorate a cluster culling shader output variable,this indexed mode specific variable will contain an integer value that specifies the base index within the index buffer corresponding to a cluster.

### Valid Usage

- VUID-FirstIndexHUAWEI-FirstIndexHUAWEI-07799

The `FirstIndexHUAWEI` decoration **must** be used only within the `ClusterCullingHUAWEI` `Execution Model`

- VUID-FirstIndexHUAWEI-FirstIndexHUAWEI-07800
  The variable decorated with `FirstIndexHUAWEI` **must** be declared as a scalar 32-bit integer value

### `FragCoord`

Decorating a variable with the `FragCoord` built-in decoration will make that variable contain the framebuffer coordinate $(x, y, z, \frac{1}{w})$ of the fragment being processed. The (x,y) coordinate (0,0) is the upper left corner of the upper left pixel in the framebuffer.

When Sample Shading is enabled, the x and y components of `FragCoord` reflect the location of one of the samples corresponding to the shader invocation.

Otherwise, the x and y components of `FragCoord` reflect the location of the center of the fragment.

The z component of `FragCoord` is the interpolated depth value of the primitive.

The w component is the interpolated $\frac{1}{w}$.

The `Centroid` interpolation decoration is ignored, but allowed, on `FragCoord`.

#### Valid Usage

- VUID-FragCoord-FragCoord-04210
  The `FragCoord` decoration **must** be used only within the `Fragment Execution Model`
- VUID-FragCoord-FragCoord-04211
  The variable decorated with `FragCoord` **must** be declared using the `Input Storage Class`
- VUID-FragCoord-FragCoord-04212
  The variable decorated with `FragCoord` **must** be declared as a four-component vector of 32-bit floating-point values

### `FragDepth`

To have a shader supply a fragment-depth value, the shader **must** declare the `DepthReplacing` execution mode. Such a shader's fragment-depth value will come from the variable decorated with the `FragDepth` built-in decoration.

This value will be used for any subsequent depth testing performed by the implementation or writes to the depth attachment. See fragment shader depth replacement for details.

#### Valid Usage

- VUID-FragDepth-FragDepth-04213
  The `FragDepth` decoration **must** be used only within the `Fragment Execution Model`

- VUID-FragDepth-FragDepth-04214
  The variable decorated with `FragDepth` **must** be declared using the `Output Storage Class`
- VUID-FragDepth-FragDepth-04215
  The variable decorated with `FragDepth` **must** be declared as a scalar 32-bit floating-point value
- VUID-FragDepth-FragDepth-04216
  If the shader dynamically writes to the variable decorated with `FragDepth`, the `DepthReplacing Execution Mode` **must** be declared

`FirstInstanceHUAWEI`

The `FirstInstanceHUAWEI` decoration can be used to decorate a cluster culling shader output variable,this variable will contain an integer value that specifies the instance ID of the first instance to draw.

### Valid Usage

- VUID-FirstInstanceHUAWEI-FirstInstanceHUAWEI-07801
  The `FirstInstanceHUAWEI` decoration **must** be used only within the `ClusterCullingHUAWEI Execution Model`
- VUID-FirstInstanceHUAWEI-FirstInstanceHUAWEI-07802
  The variable decorated with `FirstInstanceHUAWEI` **must** be declared as a scalar 32-bit integer value

`FirstVertexHUAWEI`

The `FirstVertexHUAWEI` decoration can be used to decorate a cluster culling shader output variable,this non-indexed mode specific variable will contain an integer value that specifies the index of the first vertex in a cluster to draw.

### Valid Usage

- VUID-FirstVertexHUAWEI-FirstVertexHUAWEI-07803
  The `FirstVertexHUAWEI` decoration **must** be used only within the `FirstVertexHUAWEI Execution Model`
- VUID-FirstVertexHUAWEI-FirstVertexHUAWEI-07804
  The variable decorated with `FirstVertexHUAWEI` **must** be declared as a scalar 32-bit integer value

`FragInvocationCountEXT`

Decorating a variable with the `FragInvocationCountEXT` built-in decoration will make that variable contain the maximum number of fragment shader invocations for the fragment, as determined by `minSampleShading`.

If Sample Shading is not enabled, `FragInvocationCountEXT` will be filled with a value of 1.

### Valid Usage

- VUID-FragInvocationCountEXT-FragInvocationCountEXT-04217
  The `FragInvocationCountEXT` decoration **must** be used only within the `Fragment Execution Model`
- VUID-FragInvocationCountEXT-FragInvocationCountEXT-04218
  The variable decorated with `FragInvocationCountEXT` **must** be declared using the `Input Storage Class`
- VUID-FragInvocationCountEXT-FragInvocationCountEXT-04219
  The variable decorated with `FragInvocationCountEXT` **must** be declared as a scalar 32-bit integer value

`FragSizeEXT`

Decorating a variable with the `FragSizeEXT` built-in decoration will make that variable contain the dimensions in pixels of the area that the fragment covers for that invocation.

If fragment density map is not enabled, `FragSizeEXT` will be filled with a value of (1,1).

### Valid Usage

- VUID-FragSizeEXT-FragSizeEXT-04220
  The `FragSizeEXT` decoration **must** be used only within the `Fragment Execution Model`
- VUID-FragSizeEXT-FragSizeEXT-04221
  The variable decorated with `FragSizeEXT` **must** be declared using the `Input Storage Class`
- VUID-FragSizeEXT-FragSizeEXT-04222
  The variable decorated with `FragSizeEXT` **must** be declared as a two-component vector of 32-bit integer values

`FragStencilRefEXT`

Decorating a variable with the `FragStencilRefEXT` built-in decoration will make that variable contain the new stencil reference value for all samples covered by the fragment. This value will be used as the stencil reference value used in stencil testing.

To write to `FragStencilRefEXT`, a shader **must** declare the `StencilRefReplacingEXT` execution mode. If a shader declares the `StencilRefReplacingEXT` execution mode and there is an execution path through the shader that does not set `FragStencilRefEXT`, then the fragment's stencil reference value is undefined for executions of the shader that take that path.

Only the least significant **s** bits of the integer value of the variable decorated with `FragStencilRefEXT` are considered for stencil testing, where **s** is the number of bits in the stencil framebuffer attachment, and higher order bits are discarded.

See fragment shader stencil reference replacement for more details.

### Valid Usage

- VUID-FragStencilRefEXT-FragStencilRefEXT-04223
  The `FragStencilRefEXT` decoration **must** be used only within the `Fragment Execution Model`
- VUID-FragStencilRefEXT-FragStencilRefEXT-04224
  The variable decorated with `FragStencilRefEXT` **must** be declared using the `Output Storage Class`
- VUID-FragStencilRefEXT-FragStencilRefEXT-04225
  The variable decorated with `FragStencilRefEXT` **must** be declared as a scalar integer value

`FragmentSizeNV`

Decorating a variable with the `FragmentSizeNV` built-in decoration will make that variable contain the width and height of the fragment.

### Valid Usage

- VUID-FragmentSizeNV-FragmentSizeNV-04226
  The `FragmentSizeNV` decoration **must** be used only within the `Fragment Execution Model`
- VUID-FragmentSizeNV-FragmentSizeNV-04227
  The variable decorated with `FragmentSizeNV` **must** be declared using the `Input Storage Class`
- VUID-FragmentSizeNV-FragmentSizeNV-04228
  The variable decorated with `FragmentSizeNV` **must** be declared as a two-component vector of 32-bit integer values

`FrontFacing`

Decorating a variable with the `FrontFacing` built-in decoration will make that variable contain whether the fragment is front or back facing. This variable is non-zero if the current fragment is considered to be part of a front-facing polygon primitive or of a non-polygon primitive and is zero if the fragment is considered to be part of a back-facing polygon primitive.

### Valid Usage

- VUID-FrontFacing-FrontFacing-04229
  The `FrontFacing` decoration **must** be used only within the `Fragment Execution Model`
- VUID-FrontFacing-FrontFacing-04230
  The variable decorated with `FrontFacing` **must** be declared using the `Input Storage Class`
- VUID-FrontFacing-FrontFacing-04231
  The variable decorated with `FrontFacing` **must** be declared as a boolean value

`FullyCoveredEXT`

Decorating a variable with the `FullyCoveredEXT` built-in decoration will make that variable

indicate whether the fragment area is fully covered by the generating primitive. This variable is non-zero if conservative rasterization is enabled and the current fragment area is fully covered by the generating primitive, and is zero if the fragment is not covered or partially covered, or conservative rasterization is disabled.

### Valid Usage

- VUID-FullyCoveredEXT-FullyCoveredEXT-04232
  The `FullyCoveredEXT` decoration **must** be used only within the `Fragment Execution Model`
- VUID-FullyCoveredEXT-FullyCoveredEXT-04233
  The variable decorated with `FullyCoveredEXT` **must** be declared using the `Input Storage Class`
- VUID-FullyCoveredEXT-FullyCoveredEXT-04234
  The variable decorated with `FullyCoveredEXT` **must** be declared as a boolean value
- VUID-FullyCoveredEXT-conservativeRasterizationPostDepthCoverage-04235
  If `VkPhysicalDeviceConservativeRasterizationPropertiesEXT::conservativeRasterizationPostDepthCoverage` is not supported the `PostDepthCoverage Execution Mode` **must** not be declared, when a variable with the `FullyCoveredEXT` decoration is declared

#### `GlobalInvocationId`

Decorating a variable with the `GlobalInvocationId` built-in decoration will make that variable contain the location of the current invocation within the global workgroup. Each component is equal to the index of the local workgroup multiplied by the size of the local workgroup plus `LocalInvocationId`.

### Valid Usage

- VUID-GlobalInvocationId-GlobalInvocationId-04236
  The `GlobalInvocationId` decoration **must** be used only within the `GLCompute`, `MeshEXT`, `TaskEXT`, `MeshNV`, or `TaskNV Execution Model`
- VUID-GlobalInvocationId-GlobalInvocationId-04237
  The variable decorated with `GlobalInvocationId` **must** be declared using the `Input Storage Class`
- VUID-GlobalInvocationId-GlobalInvocationId-04238
  The variable decorated with `GlobalInvocationId` **must** be declared as a three-component vector of 32-bit integer values

#### `HelperInvocation`

Decorating a variable with the `HelperInvocation` built-in decoration will make that variable contain whether the current invocation is a helper invocation. This variable is non-zero if the current fragment being shaded is a helper invocation and zero otherwise. A helper invocation is an invocation of the shader that is produced to satisfy internal requirements such as the generation of derivatives.

| | |
|---|---|
| **NOTE** | It is very likely that a helper invocation will have a value of `SampleMask` fragment shader input value that is zero. |

### Valid Usage

- VUID-HelperInvocation-HelperInvocation-04239
  The `HelperInvocation` decoration **must** be used only within the `Fragment Execution Model`
- VUID-HelperInvocation-HelperInvocation-04240
  The variable decorated with `HelperInvocation` **must** be declared using the `Input Storage Class`
- VUID-HelperInvocation-HelperInvocation-04241
  The variable decorated with `HelperInvocation` **must** be declared as a boolean value

#### `HitKindKHR`

A variable decorated with the `HitKindKHR` decoration will describe the intersection that triggered the execution of the current shader. The values are determined by the intersection shader. For user-defined intersection shaders this is the value that was passed to the "Hit Kind" operand of `OpReportIntersectionKHR`. For triangle intersection candidates, this will be one of `HitKindFrontFacingTriangleKHR` or `HitKindBackFacingTriangleKHR`.

### Valid Usage

- VUID-HitKindKHR-HitKindKHR-04242
  The `HitKindKHR` decoration **must** be used only within the `AnyHitKHR` or `ClosestHitKHR` `Execution Model`
- VUID-HitKindKHR-HitKindKHR-04243
  The variable decorated with `HitKindKHR` **must** be declared using the `Input Storage Class`
- VUID-HitKindKHR-HitKindKHR-04244
  The variable decorated with `HitKindKHR` **must** be declared as a scalar 32-bit integer value

#### `HitTNV`

A variable decorated with the `HitTNV` decoration is equivalent to a variable decorated with the `RayTmaxKHR` decoration.

### Valid Usage

- VUID-HitTNV-HitTNV-04245
  The `HitTNV` decoration **must** be used only within the `AnyHitNV` or `ClosestHitNV` `Execution Model`
- VUID-HitTNV-HitTNV-04246
  The variable decorated with `HitTNV` **must** be declared using the `Input Storage Class`
- VUID-HitTNV-HitTNV-04247

The variable decorated with `HitTNV` **must** be declared as a scalar 32-bit floating-point value

`HitTriangleVertexPositionsKHR`

A variable decorated with the `HitTriangleVertexPositionsKHR` decoration will specify the object space vertices of the triangle at the current intersection in application-provided order. The positions returned are transformed by the geometry transform, which is performed at standard floating-point precision, but without a specifically defined order of floating-point operations to perform the matrix multiplication.

### Valid Usage

- VUID-HitTriangleVertexPositionsKHR-HitTriangleVertexPositionsKHR-08747
  The `HitTriangleVertexPositionsKHR` decoration **must** be used only within the `AnyHitKHR` or `ClosestHitKHR` `Execution Model`
- VUID-HitTriangleVertexPositionsKHR-HitTriangleVertexPositionsKHR-08748
  The variable decorated with `HitTriangleVertexPositionsKHR` **must** be declared using the `Input` `Storage Class`
- VUID-HitTriangleVertexPositionsKHR-HitTriangleVertexPositionsKHR-08749
  The variable decorated with `HitTriangleVertexPositionsKHR` **must** be declared as an array of three vectors of three 32-bit float values
- VUID-HitTriangleVertexPositionsKHR-HitTriangleVertexPositionsKHR-08750
  The variable decorated with `HitTriangleVertexPositionsKHR` **must** be used only if the value of `HitKindKHR` is `HitKindFrontFacingTriangleKHR` or `HitKindBackFacingTriangleKHR`
- VUID-HitTriangleVertexPositionsKHR-None-08751
  The acceleration structure corresponding to the current intersection **must** have been built with `VK_BUILD_ACCELERATION_STRUCTURE_ALLOW_DATA_ACCESS_KHR`

`IncomingRayFlagsKHR`

A variable with the `IncomingRayFlagsKHR` decoration will contain the ray flags passed in to the trace call that invoked this particular shader. Setting pipeline flags on the ray tracing pipeline **must** not cause any corresponding flags to be set in variables with this decoration.

### Valid Usage

- VUID-IncomingRayFlagsKHR-IncomingRayFlagsKHR-04248
  The `IncomingRayFlagsKHR` decoration **must** be used only within the `IntersectionKHR`, `AnyHitKHR`, `ClosestHitKHR`, or `MissKHR` `Execution Model`
- VUID-IncomingRayFlagsKHR-IncomingRayFlagsKHR-04249
  The variable decorated with `IncomingRayFlagsKHR` **must** be declared using the `Input` `Storage Class`
- VUID-IncomingRayFlagsKHR-IncomingRayFlagsKHR-04250
  The variable decorated with `IncomingRayFlagsKHR` **must** be declared as a scalar 32-bit

integer value

`IndexCountHUAWEI`

The `IndexCountHUAWEI` decoration can be used to decorate a cluster culling shader output variable,this indexed mode specific variable will contain an integer value that specifies the number of indexed vertices in a cluster to draw.

### Valid Usage

- VUID-IndexCountHUAWEI-IndexCountHUAWEI-07805
  The `IndexCountHUAWEI` decoration **must** be used only within the `ClusterCullingHUAWEI Execution Model`
- VUID-IndexCountHUAWEI-IndexCountHUAWEI-07806
  The variable decorated with `IndexCountHUAWEI` **must** be declared as a scalar 32-bit integer value

`InstanceCountHUAWEI`

The `InstanceCountHUAWEI` decoration can be used to decorate a cluster culling shader output variable,this variable will contain an integer value that specifies the number of instance to draw in a cluster.

### Valid Usage

- VUID-InstanceCountHUAWEI-InstanceCountHUAWEI-07807
  The `InstanceCountHUAWEI` decoration **must** be used only within the `ClusterCullingHUAWEI Execution Model`
- VUID-InstanceCountHUAWEI-InstanceCountHUAWEI-07808
  The variable decorated with `InstanceCountHUAWEI` **must** be declared as a scalar 32-bit integer value

`InstanceCustomIndexKHR`

A variable decorated with the `InstanceCustomIndexKHR` decoration will contain the application-defined value of the instance that intersects the current ray. This variable contains the value that was specified in VkAccelerationStructureInstanceKHR::`instanceCustomIndex` for the current acceleration structure instance in the lower 24 bits and the upper 8 bits will be zero.

### Valid Usage

- VUID-InstanceCustomIndexKHR-InstanceCustomIndexKHR-04251
  The `InstanceCustomIndexKHR` decoration **must** be used only within the `IntersectionKHR`, `AnyHitKHR`, or `ClosestHitKHR Execution Model`
- VUID-InstanceCustomIndexKHR-InstanceCustomIndexKHR-04252
  The variable decorated with `InstanceCustomIndexKHR` **must** be declared using the `Input`

Storage Class

- VUID-InstanceCustomIndexKHR-InstanceCustomIndexKHR-04253
  The variable decorated with InstanceCustomIndexKHR **must** be declared as a scalar 32-bit integer value

InstanceId

Decorating a variable in an intersection, any-hit, or closest hit shader with the InstanceId decoration will make that variable contain the index of the instance that intersects the current ray.

### Valid Usage

- VUID-InstanceId-InstanceId-04254
  The InstanceId decoration **must** be used only within the IntersectionKHR, AnyHitKHR, or ClosestHitKHR Execution Model
- VUID-InstanceId-InstanceId-04255
  The variable decorated with InstanceId **must** be declared using the Input Storage Class
- VUID-InstanceId-InstanceId-04256
  The variable decorated with InstanceId **must** be declared as a scalar 32-bit integer value

InvocationId

Decorating a variable with the InvocationId built-in decoration will make that variable contain the index of the current shader invocation in a geometry shader, or the index of the output patch vertex in a tessellation control shader.

In a geometry shader, the index of the current shader invocation ranges from zero to the number of instances declared in the shader minus one. If the instance count of the geometry shader is one or is not specified, then InvocationId will be zero.

### Valid Usage

- VUID-InvocationId-InvocationId-04257
  The InvocationId decoration **must** be used only within the TessellationControl or Geometry Execution Model
- VUID-InvocationId-InvocationId-04258
  The variable decorated with InvocationId **must** be declared using the Input Storage Class
- VUID-InvocationId-InvocationId-04259
  The variable decorated with InvocationId **must** be declared as a scalar 32-bit integer value

InvocationsPerPixelNV

Decorating a variable with the InvocationsPerPixelNV built-in decoration will make that variable contain the maximum number of fragment shader invocations per pixel, as derived from the

effective shading rate for the fragment. If a primitive does not fully cover a pixel, the number of fragment shader invocations for that pixel **may** be less than the value of `InvocationsPerPixelNV`. If the shading rate indicates a fragment covering multiple pixels, then `InvocationsPerPixelNV` will be one.

### Valid Usage

- VUID-InvocationsPerPixelNV-InvocationsPerPixelNV-04260
  The `InvocationsPerPixelNV` decoration **must** be used only within the `Fragment Execution Model`
- VUID-InvocationsPerPixelNV-InvocationsPerPixelNV-04261
  The variable decorated with `InvocationsPerPixelNV` **must** be declared using the `Input Storage Class`
- VUID-InvocationsPerPixelNV-InvocationsPerPixelNV-04262
  The variable decorated with `InvocationsPerPixelNV` **must** be declared as a scalar 32-bit integer value

#### `InstanceIndex`

Decorating a variable in a vertex shader with the `InstanceIndex` built-in decoration will make that variable contain the index of the instance that is being processed by the current vertex shader invocation. `InstanceIndex` begins at the `firstInstance` parameter to vkCmdDraw or vkCmdDrawIndexed or at the `firstInstance` member of a structure consumed by vkCmdDrawIndirect or vkCmdDrawIndexedIndirect.

### Valid Usage

- VUID-InstanceIndex-InstanceIndex-04263
  The `InstanceIndex` decoration **must** be used only within the `Vertex Execution Model`
- VUID-InstanceIndex-InstanceIndex-04264
  The variable decorated with `InstanceIndex` **must** be declared using the `Input Storage Class`
- VUID-InstanceIndex-InstanceIndex-04265
  The variable decorated with `InstanceIndex` **must** be declared as a scalar 32-bit integer value

#### `LaunchIdKHR`

A variable decorated with the `LaunchIdKHR` decoration will specify the index of the work item being processed. One work item is generated for each of the `width` × `height` × `depth` items dispatched by a vkCmdTraceRaysKHR command. All shader invocations inherit the same value for variables decorated with `LaunchIdKHR`.

### Valid Usage

- VUID-LaunchIdKHR-LaunchIdKHR-04266
  The `LaunchIdKHR` decoration **must** be used only within the `RayGenerationKHR`,

IntersectionKHR, AnyHitKHR, ClosestHitKHR, MissKHR, or CallableKHR Execution Model

- VUID-LaunchIdKHR-LaunchIdKHR-04267
  The variable decorated with LaunchIdKHR **must** be declared using the Input Storage Class
- VUID-LaunchIdKHR-LaunchIdKHR-04268
  The variable decorated with LaunchIdKHR **must** be declared as a three-component vector of 32-bit integer values

#### LaunchSizeKHR

A variable decorated with the LaunchSizeKHR decoration will contain the width, height, and depth dimensions passed to the vkCmdTraceRaysKHR command that initiated this shader execution. The width is in the first component, the height is in the second component, and the depth is in the third component.

**Valid Usage**

- VUID-LaunchSizeKHR-LaunchSizeKHR-04269
  The LaunchSizeKHR decoration **must** be used only within the RayGenerationKHR, IntersectionKHR, AnyHitKHR, ClosestHitKHR, MissKHR, or CallableKHR Execution Model
- VUID-LaunchSizeKHR-LaunchSizeKHR-04270
  The variable decorated with LaunchSizeKHR **must** be declared using the Input Storage Class
- VUID-LaunchSizeKHR-LaunchSizeKHR-04271
  The variable decorated with LaunchSizeKHR **must** be declared as a three-component vector of 32-bit integer values

#### Layer

Decorating a variable with the Layer built-in decoration will make that variable contain the select layer of a multi-layer framebuffer attachment.

In a mesh, vertex, tessellation evaluation, or geometry shader, any variable decorated with Layer can be written with the framebuffer layer index to which the primitive produced by that shader will be directed.

The last active pre-rasterization shader stage (in pipeline order) controls the Layer that is used. Outputs in previous shader stages are not used, even if the last stage fails to write the Layer.

If the last active pre-rasterization shader stage shader entry point's interface does not include a variable decorated with Layer, then the first layer is used. If a pre-rasterization shader stage shader entry point's interface includes a variable decorated with Layer, it **must** write the same value to Layer for all output vertices of a given primitive. If the Layer value is less than 0 or greater than or equal to the number of layers in the framebuffer, then primitives **may** still be rasterized, fragment shaders **may** be executed, and the framebuffer values for all layers are undefined. In a mesh shader this also applies when the Layer value is greater than or equal to the maxMeshOutputLayers limit.

If a variable with the Layer decoration is also decorated with ViewportRelativeNV, then the

ViewportIndex is added to the layer that is used for rendering and that is made available in the fragment shader.

If the shader writes to a variable decorated ViewportMaskNV, then the layer selected has a different value for each viewport a primitive is rendered to.

In a fragment shader, a variable decorated with Layer contains the layer index of the primitive that the fragment invocation belongs to.

### Valid Usage

- VUID-Layer-Layer-04272
  The Layer decoration **must** be used only within the MeshEXT, MeshNV, Vertex, TessellationEvaluation, Geometry, or Fragment Execution Model
- VUID-Layer-Layer-04274
  The variable decorated with Layer within the MeshEXT, MeshNV, Vertex, TessellationEvaluation, or Geometry Execution Model **must** be declared using the Output Storage Class
- VUID-Layer-Layer-04275
  The variable decorated with Layer within the Fragment Execution Model **must** be declared using the Input Storage Class
- VUID-Layer-Layer-04276
  The variable decorated with Layer **must** be declared as a scalar 32-bit integer value
- VUID-Layer-Layer-07039
  The variable decorated with Layer within the MeshEXT Execution Model **must** also be decorated with the PerPrimitiveEXT decoration

#### LayerPerViewNV

Decorating a variable with the LayerPerViewNV built-in decoration will make that variable contain the per-view layer information. The per-view layer has the same semantics as Layer, for each view.

### Valid Usage

- VUID-LayerPerViewNV-LayerPerViewNV-04277
  The LayerPerViewNV decoration **must** be used only within the MeshNV Execution Model
- VUID-LayerPerViewNV-LayerPerViewNV-04278
  The variable decorated with LayerPerViewNV **must** be declared using the Output Storage Class
- VUID-LayerPerViewNV-LayerPerViewNV-04279
  The variable decorated with LayerPerViewNV **must** also be decorated with the PerViewNV decoration
- VUID-LayerPerViewNV-LayerPerViewNV-04280
  The variable decorated with LayerPerViewNV **must** be declared as an array of scalar 32-bit

integer values

### LocalInvocationId

Decorating a variable with the LocalInvocationId built-in decoration will make that variable contain the location of the current cluster culling, task, mesh, or compute shader invocation within the local workgroup. Each component ranges from zero through to the size of the workgroup in that dimension minus one.

**NOTE** If the size of the workgroup in a particular dimension is one, then the LocalInvocationId in that dimension will be zero. If the workgroup is effectively two-dimensional, then LocalInvocationId.z will be zero. If the workgroup is effectively one-dimensional, then both LocalInvocationId.y and LocalInvocationId.z will be zero.

#### Valid Usage

- VUID-LocalInvocationId-LocalInvocationId-04281
  The LocalInvocationId decoration **must** be used only within the GLCompute, MeshEXT, TaskEXT, MeshNV, or TaskNV Execution Model
- VUID-LocalInvocationId-LocalInvocationId-04282
  The variable decorated with LocalInvocationId **must** be declared using the Input Storage Class
- VUID-LocalInvocationId-LocalInvocationId-04283
  The variable decorated with LocalInvocationId **must** be declared as a three-component vector of 32-bit integer values

### LocalInvocationIndex

Decorating a variable with the LocalInvocationIndex built-in decoration will make that variable contain a one-dimensional representation of LocalInvocationId. This is computed as:

```
LocalInvocationIndex =
    LocalInvocationId.z * WorkgroupSize.x * WorkgroupSize.y +
    LocalInvocationId.y * WorkgroupSize.x +
    LocalInvocationId.x;
```

#### Valid Usage

- VUID-LocalInvocationIndex-LocalInvocationIndex-04284
  The LocalInvocationIndex decoration **must** be used only within the GLCompute, MeshEXT, TaskEXT, MeshNV, or TaskNV Execution Model
- VUID-LocalInvocationIndex-LocalInvocationIndex-04285
  The variable decorated with LocalInvocationIndex **must** be declared using the Input Storage Class

- VUID-LocalInvocationIndex-LocalInvocationIndex-04286
  The variable decorated with `LocalInvocationIndex` **must** be declared as a scalar 32-bit integer value

`MeshViewCountNV`

Decorating a variable with the `MeshViewCountNV` built-in decoration will make that variable contain the number of views processed by the current mesh or task shader invocations.

### Valid Usage

- VUID-MeshViewCountNV-MeshViewCountNV-04287
  The `MeshViewCountNV` decoration **must** be used only within the `MeshNV` or `TaskNV Execution Model`
- VUID-MeshViewCountNV-MeshViewCountNV-04288
  The variable decorated with `MeshViewCountNV` **must** be declared using the `Input Storage Class`
- VUID-MeshViewCountNV-MeshViewCountNV-04289
  The variable decorated with `MeshViewCountNV` **must** be declared as a scalar 32-bit integer value

`MeshViewIndicesNV`

Decorating a variable with the `MeshViewIndicesNV` built-in decoration will make that variable contain the mesh view indices. The mesh view indices is an array of values where each element holds the view number of one of the views being processed by the current mesh or task shader invocations. The values of array elements with indices greater than or equal to `MeshViewCountNV` are undefined. If the value of `MeshViewIndicesNV`[i] is j, then any outputs decorated with `PerViewNV` will take on the value of array element i when processing primitives for view index j.

### Valid Usage

- VUID-MeshViewIndicesNV-MeshViewIndicesNV-04290
  The `MeshViewIndicesNV` decoration **must** be used only within the `MeshNV` or `TaskNV Execution Model`
- VUID-MeshViewIndicesNV-MeshViewIndicesNV-04291
  The variable decorated with `MeshViewIndicesNV` **must** be declared using the `Input Storage Class`
- VUID-MeshViewIndicesNV-MeshViewIndicesNV-04292
  The variable decorated with `MeshViewIndicesNV` **must** be declared as an array of scalar 32-bit integer values

`NumSubgroups`

Decorating a variable with the `NumSubgroups` built-in decoration will make that variable contain the number of subgroups in the local workgroup.

### Valid Usage

- VUID-NumSubgroups-NumSubgroups-04293
  The `NumSubgroups` decoration **must** be used only within the `GLCompute`, `MeshEXT`, `TaskEXT`, `MeshNV`, or `TaskNV` `Execution Model`
- VUID-NumSubgroups-NumSubgroups-04294
  The variable decorated with `NumSubgroups` **must** be declared using the `Input` `Storage Class`
- VUID-NumSubgroups-NumSubgroups-04295
  The variable decorated with `NumSubgroups` **must** be declared as a scalar 32-bit integer value

`NumWorkgroups`

Decorating a variable with the `NumWorkgroups` built-in decoration will make that variable contain the number of local workgroups that are part of the dispatch that the invocation belongs to. Each component is equal to the values of the workgroup count parameters passed into the dispatching commands.

### Valid Usage

- VUID-NumWorkgroups-NumWorkgroups-04296
  The `NumWorkgroups` decoration **must** be used only within the `GLCompute`, `MeshEXT`, or `TaskEXT` `Execution Model`
- VUID-NumWorkgroups-NumWorkgroups-04297
  The variable decorated with `NumWorkgroups` **must** be declared using the `Input` `Storage Class`
- VUID-NumWorkgroups-NumWorkgroups-04298
  The variable decorated with `NumWorkgroups` **must** be declared as a three-component vector of 32-bit integer values

`ObjectRayDirectionKHR`

A variable decorated with the `ObjectRayDirectionKHR` decoration will specify the direction of the ray being processed, in object space.

### Valid Usage

- VUID-ObjectRayDirectionKHR-ObjectRayDirectionKHR-04299
  The `ObjectRayDirectionKHR` decoration **must** be used only within the `IntersectionKHR`, `AnyHitKHR`, or `ClosestHitKHR` `Execution Model`
- VUID-ObjectRayDirectionKHR-ObjectRayDirectionKHR-04300
  The variable decorated with `ObjectRayDirectionKHR` **must** be declared using the `Input` `Storage Class`
- VUID-ObjectRayDirectionKHR-ObjectRayDirectionKHR-04301
  The variable decorated with `ObjectRayDirectionKHR` **must** be declared as a three-component vector of 32-bit floating-point values

`ObjectRayOriginKHR`

A variable decorated with the `ObjectRayOriginKHR` decoration will specify the origin of the ray being processed, in object space.

### Valid Usage

- VUID-ObjectRayOriginKHR-ObjectRayOriginKHR-04302
  The `ObjectRayOriginKHR` decoration **must** be used only within the `IntersectionKHR`, `AnyHitKHR`, or `ClosestHitKHR Execution Model`
- VUID-ObjectRayOriginKHR-ObjectRayOriginKHR-04303
  The variable decorated with `ObjectRayOriginKHR` **must** be declared using the `Input Storage Class`
- VUID-ObjectRayOriginKHR-ObjectRayOriginKHR-04304
  The variable decorated with `ObjectRayOriginKHR` **must** be declared as a three-component vector of 32-bit floating-point values

`ObjectToWorldKHR`

A variable decorated with the `ObjectToWorldKHR` decoration will contain the current object-to-world transformation matrix, which is determined by the instance of the current intersection.

### Valid Usage

- VUID-ObjectToWorldKHR-ObjectToWorldKHR-04305
  The `ObjectToWorldKHR` decoration **must** be used only within the `IntersectionKHR`, `AnyHitKHR`, or `ClosestHitKHR Execution Model`
- VUID-ObjectToWorldKHR-ObjectToWorldKHR-04306
  The variable decorated with `ObjectToWorldKHR` **must** be declared using the `Input Storage Class`
- VUID-ObjectToWorldKHR-ObjectToWorldKHR-04307
  The variable decorated with `ObjectToWorldKHR` **must** be declared as a matrix with four columns of three-component vectors of 32-bit floating-point values

`PatchVertices`

Decorating a variable with the `PatchVertices` built-in decoration will make that variable contain the number of vertices in the input patch being processed by the shader. In a Tessellation Control Shader, this is the same as the name:patchControlPoints member of VkPipelineTessellationStateCreateInfo. In a Tessellation Evaluation Shader, `PatchVertices` is equal to the tessellation control output patch size. When the same shader is used in different pipelines where the patch sizes are configured differently, the value of the `PatchVertices` variable will also differ.

### Valid Usage

- VUID-PatchVertices-PatchVertices-04308
  The `PatchVertices` decoration **must** be used only within the `TessellationControl` or `TessellationEvaluation` `Execution Model`
- VUID-PatchVertices-PatchVertices-04309
  The variable decorated with `PatchVertices` **must** be declared using the `Input` `Storage Class`
- VUID-PatchVertices-PatchVertices-04310
  The variable decorated with `PatchVertices` **must** be declared as a scalar 32-bit integer value

### `PointCoord`

Decorating a variable with the `PointCoord` built-in decoration will make that variable contain the coordinate of the current fragment within the point being rasterized, normalized to the size of the point with origin in the upper left corner of the point, as described in Basic Point Rasterization. If the primitive the fragment shader invocation belongs to is not a point, then the variable decorated with `PointCoord` contains an undefined value.

**NOTE** Depending on how the point is rasterized, `PointCoord` **may** never reach (0,0) or (1,1).

#### Valid Usage

- VUID-PointCoord-PointCoord-04311
  The `PointCoord` decoration **must** be used only within the `Fragment` `Execution Model`
- VUID-PointCoord-PointCoord-04312
  The variable decorated with `PointCoord` **must** be declared using the `Input` `Storage Class`
- VUID-PointCoord-PointCoord-04313
  The variable decorated with `PointCoord` **must** be declared as a two-component vector of 32-bit floating-point values

### `PointSize`

Decorating a variable with the `PointSize` built-in decoration will make that variable contain the size of point primitives or the final rasterization of polygons if polygon mode is `VK_POLYGON_MODE_POINT` when `VkPhysicalDeviceMaintenance5PropertiesKHR`::`polygonModePointSize` is set to `VK_TRUE` . The value written to the variable decorated with `PointSize` by the last pre-rasterization shader stage in the pipeline is used as the framebuffer-space size of points produced by rasterization. If `maintenance5` is enabled and a value is not written to a variable decorated with `PointSize`, a value of 1.0 is used as the size of points.

**NOTE** When `PointSize` decorates a variable in the `Input` `Storage Class`, it contains the data written to the output variable decorated with `PointSize` from the previous shader stage.

### Valid Usage

- VUID-PointSize-PointSize-04314
  The `PointSize` decoration **must** be used only within the `MeshEXT`, `MeshNV`, `Vertex`, `TessellationControl`, `TessellationEvaluation`, or `Geometry Execution Model`
- VUID-PointSize-PointSize-04315
  The variable decorated with `PointSize` within the `MeshEXT`, `MeshNV`, or `Vertex Execution Model` **must** be declared using the `Output Storage Class`
- VUID-PointSize-PointSize-04316
  The variable decorated with `PointSize` within the `TessellationControl`, `TessellationEvaluation`, or `Geometry Execution Model` **must** not be declared using a `Storage Class` other than `Input` or `Output`
- VUID-PointSize-PointSize-04317
  The variable decorated with `PointSize` **must** be declared as a scalar 32-bit floating-point value

`Position`

Decorating a variable with the `Position` built-in decoration will make that variable contain the position of the current vertex. In the last pre-rasterization shader stage, the value of the variable decorated with `Position` is used in subsequent primitive assembly, clipping, and rasterization operations.

| | |
|---|---|
| **NOTE** | When `Position` decorates a variable in the `Input Storage Class`, it contains the data written to the output variable decorated with `Position` from the previous shader stage. |

### Valid Usage

- VUID-Position-Position-04318
  The `Position` decoration **must** be used only within the `MeshEXT`, `MeshNV`, `Vertex`, `TessellationControl`, `TessellationEvaluation`, or `Geometry Execution Model`
- VUID-Position-Position-04319
  The variable decorated with `Position` within the `MeshEXT`, `MeshNV`, or `Vertex Execution Model` **must** be declared using the `Output Storage Class`
- VUID-Position-Position-04320
  The variable decorated with `Position` within the `TessellationControl`, `TessellationEvaluation`, or `Geometry Execution Model` **must** not be declared using a `Storage Class` other than `Input` or `Output`
- VUID-Position-Position-04321
  The variable decorated with `Position` **must** be declared as a four-component vector of 32-bit floating-point values

`PositionPerViewNV`

Decorating a variable with the `PositionPerViewNV` built-in decoration will make that variable contain the position of the current vertex, for each view.

Elements of the array correspond to views in a multiview subpass, and those elements corresponding to views in the view mask of the subpass the shader is compiled against will be used as the position value for those views. For the final pre-rasterization shader stage in the pipeline, values written to an output variable decorated with `PositionPerViewNV` are used in subsequent primitive assembly, clipping, and rasterization operations, as with `Position`. `PositionPerViewNV` output in an earlier pre-rasterization shader stage is available as an input in the subsequent pre-rasterization shader stage.

If a shader is compiled against a subpass that has the `VK_SUBPASS_DESCRIPTION_PER_VIEW_POSITION_X_ONLY_BIT_NVX` bit set, then the position values for each view **must** not differ in any component other than the X component. If the values do differ, one will be chosen in an implementation-dependent manner.

### Valid Usage

- VUID-PositionPerViewNV-PositionPerViewNV-04322
  The `PositionPerViewNV` decoration **must** be used only within the `MeshNV`, `Vertex`, `TessellationControl`, `TessellationEvaluation`, or `Geometry Execution Model`
- VUID-PositionPerViewNV-PositionPerViewNV-04323
  The variable decorated with `PositionPerViewNV` within the `Vertex`, or `MeshNV Execution Model` **must** be declared using the `Output Storage Class`
- VUID-PositionPerViewNV-PositionPerViewNV-04324
  The variable decorated with `PositionPerViewNV` within the `TessellationControl`, `TessellationEvaluation`, or `Geometry Execution Model` **must** not be declared using a `Storage Class` other than `Input` or `Output`
- VUID-PositionPerViewNV-PositionPerViewNV-04325
  The variable decorated with `PositionPerViewNV` **must** be declared as an array of four-component vector of 32-bit floating-point values with at least as many elements as the maximum view in the subpass's view mask plus one
- VUID-PositionPerViewNV-PositionPerViewNV-04326
  The array variable decorated with `PositionPerViewNV` **must** only be indexed by a constant or specialization constant

`PrimitiveCountNV`

Decorating a variable with the `PrimitiveCountNV` decoration will make that variable contain the primitive count. The primitive count specifies the number of primitives in the output mesh produced by the mesh shader that will be processed by subsequent pipeline stages.

### Valid Usage

- VUID-PrimitiveCountNV-PrimitiveCountNV-04327

The `PrimitiveCountNV` decoration **must** be used only within the `MeshNV Execution Model`

- VUID-PrimitiveCountNV-PrimitiveCountNV-04328
  The variable decorated with `PrimitiveCountNV` **must** be declared using the `Output Storage Class`
- VUID-PrimitiveCountNV-PrimitiveCountNV-04329
  The variable decorated with `PrimitiveCountNV` **must** be declared as a scalar 32-bit integer value

`PrimitiveId`

Decorating a variable with the `PrimitiveId` built-in decoration will make that variable contain the index of the current primitive.

The index of the first primitive generated by a drawing command is zero, and the index is incremented after every individual point, line, or triangle primitive is processed.

For triangles drawn as points or line segments (see Polygon Mode), the primitive index is incremented only once, even if multiple points or lines are eventually drawn.

Variables decorated with `PrimitiveId` are reset to zero between each instance drawn.

Restarting a primitive topology using primitive restart has no effect on the value of variables decorated with `PrimitiveId`.

In tessellation control and tessellation evaluation shaders, it will contain the index of the patch within the current set of rendering primitives that corresponds to the shader invocation.

In a geometry shader, it will contain the number of primitives presented as input to the shader since the current set of rendering primitives was started.

In a fragment shader, it will contain the primitive index written by the mesh shader if a mesh shader is present, or the primitive index written by the geometry shader if a geometry shader is present, or with the value that would have been presented as input to the geometry shader had it been present.

In an intersection, any-hit, or closest hit shader, it will contain the index within the geometry of the triangle or bounding box being processed.

**NOTE**

When the `PrimitiveId` decoration is applied to an output variable in the mesh shader or geometry shader, the resulting value is seen through the `PrimitiveId` decorated input variable in the fragment shader.

The fragment shader using `PrimitiveId` will need to declare either the `MeshShadingNV`, `MeshShadingEXT`, `Geometry` or `Tessellation` capability to satisfy the requirement SPIR-V has to use `PrimitiveId`.

### Valid Usage

- VUID-PrimitiveId-PrimitiveId-04330

The PrimitiveId decoration **must** be used only within the MeshEXT, MeshNV, IntersectionKHR, AnyHitKHR, ClosestHitKHR, TessellationControl, TessellationEvaluation, Geometry, or Fragment Execution Model

- VUID-PrimitiveId-Fragment-04331
  If pipeline contains both the Fragment and Geometry Execution Model and a variable decorated with PrimitiveId is read from Fragment shader, then the Geometry shader **must** write to the output variables decorated with PrimitiveId in all execution paths
- VUID-PrimitiveId-Fragment-04332
  If pipeline contains both the Fragment and MeshEXT or MeshNV Execution Model and a variable decorated with PrimitiveId is read from Fragment shader, then the MeshEXT or MeshNV shader **must** write to the output variables decorated with PrimitiveId in all execution paths
- VUID-PrimitiveId-Fragment-04333
  If Fragment Execution Model contains a variable decorated with PrimitiveId, then either the MeshShadingEXT, MeshShadingNV, Geometry or Tessellation capability **must** also be declared
- VUID-PrimitiveId-PrimitiveId-04334
  The variable decorated with PrimitiveId within the TessellationControl, TessellationEvaluation, Fragment, IntersectionKHR, AnyHitKHR, or ClosestHitKHR Execution Model **must** be declared using the Input Storage Class
- VUID-PrimitiveId-PrimitiveId-04335
  The variable decorated with PrimitiveId within the Geometry Execution Model **must** be declared using the Input or Output Storage Class
- VUID-PrimitiveId-PrimitiveId-04336
  The variable decorated with PrimitiveId within the MeshEXT or MeshNV Execution Model **must** be declared using the Output Storage Class
- VUID-PrimitiveId-PrimitiveId-04337
  The variable decorated with PrimitiveId **must** be declared as a scalar 32-bit integer value
- VUID-PrimitiveId-PrimitiveId-07040
  The variable decorated with PrimitiveId within the MeshEXT Execution Model **must** also be decorated with the PerPrimitiveEXT decoration

### PrimitiveIndicesNV

Decorating a variable with the PrimitiveIndicesNV decoration will make that variable contain the output array of vertex index values. Depending on the output primitive type declared using the execution mode, the indices are split into groups of one (OutputPoints), two (OutputLinesNV), or three (OutputTrianglesNV) indices and each group generates a primitive.

#### Valid Usage

- VUID-PrimitiveIndicesNV-PrimitiveIndicesNV-04338
  The PrimitiveIndicesNV decoration **must** be used only within the MeshNV Execution Model
- VUID-PrimitiveIndicesNV-PrimitiveIndicesNV-04339
  The variable decorated with PrimitiveIndicesNV **must** be declared using the Output Storage

Class

- VUID-PrimitiveIndicesNV-PrimitiveIndicesNV-04340
  The variable decorated with PrimitiveIndicesNV **must** be declared as an array of scalar 32-bit integer values
- VUID-PrimitiveIndicesNV-PrimitiveIndicesNV-04341
  All index values of the array decorated with PrimitiveIndicesNV **must** be in the range [0, N-1], where N is the value specified by the OutputVertices Execution Mode
- VUID-PrimitiveIndicesNV-OutputPoints-04342
  If the Execution Mode is OutputPoints, then the array decorated with PrimitiveIndicesNV **must** be the size of the value specified by OutputPrimitivesNV
- VUID-PrimitiveIndicesNV-OutputLinesNV-04343
  If the Execution Mode is OutputLinesNV, then the array decorated with PrimitiveIndicesNV **must** be the size of two times the value specified by OutputPrimitivesNV
- VUID-PrimitiveIndicesNV-OutputTrianglesNV-04344
  If the Execution Mode is OutputTrianglesNV, then the array decorated with PrimitiveIndicesNV **must** be the size of three times the value specified by OutputPrimitivesNV

PrimitivePointIndicesEXT

Decorating a variable with the PrimitivePointIndicesEXT decoration will make that variable contain the output array of vertex index values for point primitives.

### Valid Usage

- VUID-PrimitivePointIndicesEXT-PrimitivePointIndicesEXT-07041
  The PrimitivePointIndicesEXT decoration **must** be used only within the MeshEXT Execution Model
- VUID-PrimitivePointIndicesEXT-PrimitivePointIndicesEXT-07042
  The PrimitivePointIndicesEXT decoration **must** be used with the OutputPoints Execution Mode
- VUID-PrimitivePointIndicesEXT-PrimitivePointIndicesEXT-07043
  The variable decorated with PrimitivePointIndicesEXT **must** be declared using the Output Storage Class
- VUID-PrimitivePointIndicesEXT-PrimitivePointIndicesEXT-07044
  The variable decorated with PrimitivePointIndicesEXT **must** be declared as an array of scalar 32-bit integer values
- VUID-PrimitivePointIndicesEXT-PrimitivePointIndicesEXT-07045
  All index values of the array decorated with PrimitivePointIndicesEXT **must** be in the range [0, N-1], where N is the value specified by the OutputVertices Execution Mode
- VUID-PrimitivePointIndicesEXT-PrimitivePointIndicesEXT-07046
  The size of the array decorated with PrimitivePointIndicesEXT **must** match the value specified by OutputPrimitivesEXT

`PrimitiveLineIndicesEXT`

Decorating a variable with the `PrimitiveLineIndicesEXT` decoration will make that variable contain the output array of vertex index values for line primitives.

### Valid Usage

- VUID-PrimitiveLineIndicesEXT-PrimitiveLineIndicesEXT-07047
  The `PrimitiveLineIndicesEXT` decoration **must** be used only within the `MeshEXT Execution Model`
- VUID-PrimitiveLineIndicesEXT-PrimitiveLineIndicesEXT-07048
  The `PrimitiveLineIndicesEXT` decoration **must** be used with the `OutputLinesEXT Execution Mode`
- VUID-PrimitiveLineIndicesEXT-PrimitiveLineIndicesEXT-07049
  The variable decorated with `PrimitiveLineIndicesEXT` **must** be declared using the `Output Storage Class`
- VUID-PrimitiveLineIndicesEXT-PrimitiveLineIndicesEXT-07050
  The variable decorated with `PrimitiveLineIndicesEXT` **must** be declared as an array of two component vector 32-bit integer values
- VUID-PrimitiveLineIndicesEXT-PrimitiveLineIndicesEXT-07051
  All index values of the array decorated with `PrimitiveLineIndicesEXT` **must** be in the range [0, N-1], where N is the value specified by the `OutputVertices Execution Mode`
- VUID-PrimitiveLineIndicesEXT-PrimitiveLineIndicesEXT-07052
  The size of the array decorated with `PrimitiveLineIndicesEXT` **must** match the value specified by `OutputPrimitivesEXT`

`PrimitiveTriangleIndicesEXT`

Decorating a variable with the `PrimitiveTriangleIndicesEXT` decoration will make that variable contain the output array of vertex index values for triangle primitives.

### Valid Usage

- VUID-PrimitiveTriangleIndicesEXT-PrimitiveTriangleIndicesEXT-07053
  The `PrimitiveTriangleIndicesEXT` decoration **must** be used only within the `MeshEXT Execution Model`
- VUID-PrimitiveTriangleIndicesEXT-PrimitiveTriangleIndicesEXT-07054
  The `PrimitiveTriangleIndicesEXT` decoration **must** be used with the `OutputTrianglesEXT Execution Mode`
- VUID-PrimitiveTriangleIndicesEXT-PrimitiveTriangleIndicesEXT-07055
  The variable decorated with `PrimitiveTriangleIndicesEXT` **must** be declared using the `Output Storage Class`
- VUID-PrimitiveTriangleIndicesEXT-PrimitiveTriangleIndicesEXT-07056
  The variable decorated with `PrimitiveTriangleIndicesEXT` **must** be declared as an array of three component vector 32-bit integer values

- VUID-PrimitiveTriangleIndicesEXT-PrimitiveTriangleIndicesEXT-07057
  All index values of the array decorated with `PrimitiveTriangleIndicesEXT` **must** be in the range [0, N-1], where N is the value specified by the `OutputVertices Execution Mode`
- VUID-PrimitiveTriangleIndicesEXT-PrimitiveTriangleIndicesEXT-07058
  The size of the array decorated with `PrimitiveTriangleIndicesEXT` **must** match the value specified by `OutputPrimitivesEXT`

`PrimitiveShadingRateKHR`

Decorating a variable with the `PrimitiveShadingRateKHR` built-in decoration will make that variable contain the primitive fragment shading rate.

The value written to the variable decorated with `PrimitiveShadingRateKHR` by the last pre-rasterization shader stage in the pipeline is used as the primitive fragment shading rate. Outputs in previous shader stages are ignored.

If the last active pre-rasterization shader stage shader entry point's interface does not include a variable decorated with `PrimitiveShadingRateKHR`, then it is as if the shader specified a fragment shading rate value of 0, indicating a horizontal and vertical rate of 1 pixel.

If a shader has `PrimitiveShadingRateKHR` in the output interface and there is an execution path through the shader that does not write to it, its value is undefined for executions of the shader that take that path.

### Valid Usage

- VUID-PrimitiveShadingRateKHR-PrimitiveShadingRateKHR-04484
  The `PrimitiveShadingRateKHR` decoration **must** be used only within the `MeshEXT`, `MeshNV`, `Vertex`, or `Geometry Execution Model`
- VUID-PrimitiveShadingRateKHR-PrimitiveShadingRateKHR-04485
  The variable decorated with `PrimitiveShadingRateKHR` **must** be declared using the `Output Storage Class`
- VUID-PrimitiveShadingRateKHR-PrimitiveShadingRateKHR-04486
  The variable decorated with `PrimitiveShadingRateKHR` **must** be declared as a scalar 32-bit integer value
- VUID-PrimitiveShadingRateKHR-PrimitiveShadingRateKHR-04487
  The value written to `PrimitiveShadingRateKHR` **must** include no more than one of `Vertical2Pixels` and `Vertical4Pixels`
- VUID-PrimitiveShadingRateKHR-PrimitiveShadingRateKHR-04488
  The value written to `PrimitiveShadingRateKHR` **must** include no more than one of `Horizontal2Pixels` and `Horizontal4Pixels`
- VUID-PrimitiveShadingRateKHR-PrimitiveShadingRateKHR-04489
  The value written to `PrimitiveShadingRateKHR` **must** not have any bits set other than those defined by **Fragment Shading Rate Flags** enumerants in the SPIR-V specification
- VUID-PrimitiveShadingRateKHR-PrimitiveShadingRateKHR-07059
  The variable decorated with `PrimitiveShadingRateKHR` within the `MeshEXT Execution Model`

**must** also be decorated with the `PerPrimitiveEXT` decoration

### `RayGeometryIndexKHR`

A variable decorated with the `RayGeometryIndexKHR` decoration will contain the geometry index for the acceleration structure geometry currently being shaded.

#### Valid Usage

- VUID-RayGeometryIndexKHR-RayGeometryIndexKHR-04345
  The `RayGeometryIndexKHR` decoration **must** be used only within the `IntersectionKHR`, `AnyHitKHR`, or `ClosestHitKHR` `Execution Model`
- VUID-RayGeometryIndexKHR-RayGeometryIndexKHR-04346
  The variable decorated with `RayGeometryIndexKHR` **must** be declared using the `Input` `Storage Class`
- VUID-RayGeometryIndexKHR-RayGeometryIndexKHR-04347
  The variable decorated with `RayGeometryIndexKHR` **must** be declared as a scalar 32-bit integer value

### `RayTmaxKHR`

A variable decorated with the `RayTmaxKHR` decoration will contain the parametric $t_{max}$ value of the ray being processed. The value is independent of the space in which the ray origin and direction exist. The value is initialized to the parameter passed into the pipeline trace ray instruction.

The $t_{max}$ value changes throughout the lifetime of the ray that produced the intersection. In the closest hit shader, the value reflects the closest distance to the intersected primitive. In the any-hit shader, it reflects the distance to the primitive currently being intersected. In the intersection shader, it reflects the distance to the closest primitive intersected so far or the initial value. The value can change in the intersection shader after calling `OpReportIntersectionKHR` if the corresponding any-hit shader does not ignore the intersection. In a miss shader, the value is identical to the parameter passed into the pipeline trace ray instruction.

#### Valid Usage

- VUID-RayTmaxKHR-RayTmaxKHR-04348
  The `RayTmaxKHR` decoration **must** be used only within the `IntersectionKHR`, `AnyHitKHR`, `ClosestHitKHR`, or `MissKHR` `Execution Model`
- VUID-RayTmaxKHR-RayTmaxKHR-04349
  The variable decorated with `RayTmaxKHR` **must** be declared using the `Input` `Storage Class`
- VUID-RayTmaxKHR-RayTmaxKHR-04350
  The variable decorated with `RayTmaxKHR` **must** be declared as a scalar 32-bit floating-point value

### RayTminKHR

A variable decorated with the RayTminKHR decoration will contain the parametric $t_{min}$ value of the ray being processed. The value is independent of the space in which the ray origin and direction exist. The value is the parameter passed into the pipeline trace ray instruction.

The $t_{min}$ value remains constant for the duration of the ray query.

#### Valid Usage

- VUID-RayTminKHR-RayTminKHR-04351
  The RayTminKHR decoration **must** be used only within the IntersectionKHR, AnyHitKHR, ClosestHitKHR, or MissKHR Execution Model
- VUID-RayTminKHR-RayTminKHR-04352
  The variable decorated with RayTminKHR **must** be declared using the Input Storage Class
- VUID-RayTminKHR-RayTminKHR-04353
  The variable decorated with RayTminKHR **must** be declared as a scalar 32-bit floating-point value

### SampleId

Decorating a variable with the SampleId built-in decoration will make that variable contain the coverage index for the current fragment shader invocation. SampleId ranges from zero to the number of samples in the framebuffer minus one. If a fragment shader entry point's interface includes an input variable decorated with SampleId, Sample Shading is considered enabled with a minSampleShading value of 1.0.

#### Valid Usage

- VUID-SampleId-SampleId-04354
  The SampleId decoration **must** be used only within the Fragment Execution Model
- VUID-SampleId-SampleId-04355
  The variable decorated with SampleId **must** be declared using the Input Storage Class
- VUID-SampleId-SampleId-04356
  The variable decorated with SampleId **must** be declared as a scalar 32-bit integer value

### SampleMask

Decorating a variable with the SampleMask built-in decoration will make any variable contain the sample mask for the current fragment shader invocation.

A variable in the Input storage class decorated with SampleMask will contain a bitmask of the set of samples covered by the primitive generating the fragment during rasterization. It has a sample bit set if and only if the sample is considered covered for this fragment shader invocation. SampleMask[] is an array of integers. Bits are mapped to samples in a manner where bit B of mask M (SampleMask[M]) corresponds to sample $32 \times M + B$.

A variable in the `Output` storage class decorated with `SampleMask` is an array of integers forming a bit array in a manner similar to an input variable decorated with `SampleMask`, but where each bit represents coverage as computed by the shader. This computed `SampleMask` is combined with the generated coverage mask in the multisample coverage operation.

Variables decorated with `SampleMask` **must** be either an unsized array, or explicitly sized to be no larger than the implementation-dependent maximum sample-mask (as an array of 32-bit elements), determined by the maximum number of samples.

If a fragment shader entry point's interface includes an output variable decorated with `SampleMask`, the sample mask will be undefined for any array elements of any fragment shader invocations that fail to assign a value. If a fragment shader entry point's interface does not include an output variable decorated with `SampleMask`, the sample mask has no effect on the processing of a fragment.

### Valid Usage

- VUID-SampleMask-SampleMask-04357
  The `SampleMask` decoration **must** be used only within the `Fragment Execution Model`
- VUID-SampleMask-SampleMask-04358
  The variable decorated with `SampleMask` **must** be declared using the `Input` or `Output Storage Class`
- VUID-SampleMask-SampleMask-04359
  The variable decorated with `SampleMask` **must** be declared as an array of 32-bit integer values

#### `SamplePosition`

Decorating a variable with the `SamplePosition` built-in decoration will make that variable contain the sub-pixel position of the sample being shaded. The top left of the pixel is considered to be at coordinate (0,0) and the bottom right of the pixel is considered to be at coordinate (1,1).

If the render pass has a fragment density map attachment, the variable will instead contain the sub-fragment position of the sample being shaded. The top left of the fragment is considered to be at coordinate (0,0) and the bottom right of the fragment is considered to be at coordinate (1,1) for any fragment area.

If a fragment shader entry point's interface includes an input variable decorated with `SamplePosition`, Sample Shading is considered enabled with a `minSampleShading` value of 1.0.

If the current pipeline uses custom sample locations the value of any variable decorated with the `SamplePosition` built-in decoration is undefined.

### Valid Usage

- VUID-SamplePosition-SamplePosition-04360
  The `SamplePosition` decoration **must** be used only within the `Fragment Execution Model`

- VUID-SamplePosition-SamplePosition-04361
  The variable decorated with `SamplePosition` **must** be declared using the `Input Storage Class`
- VUID-SamplePosition-SamplePosition-04362
  The variable decorated with `SamplePosition` **must** be declared as a two-component vector of 32-bit floating-point values

`ShadingRateKHR`

Decorating a variable with the `ShadingRateKHR` built-in decoration will make that variable contain the fragment shading rate for the current fragment invocation.

### Valid Usage

- VUID-ShadingRateKHR-ShadingRateKHR-04490
  The `ShadingRateKHR` decoration **must** be used only within the `Fragment Execution Model`
- VUID-ShadingRateKHR-ShadingRateKHR-04491
  The variable decorated with `ShadingRateKHR` **must** be declared using the `Input Storage Class`
- VUID-ShadingRateKHR-ShadingRateKHR-04492
  The variable decorated with `ShadingRateKHR` **must** be declared as a scalar 32-bit integer value

`SMCountNV`

Decorating a variable with the `SMCountNV` built-in decoration will make that variable contain the number of SMs on the device.

### Valid Usage

- VUID-SMCountNV-SMCountNV-04363
  The variable decorated with `SMCountNV` **must** be declared using the `Input Storage Class`
- VUID-SMCountNV-SMCountNV-04364
  The variable decorated with `SMCountNV` **must** be declared as a scalar 32-bit integer value

`SMIDNV`

Decorating a variable with the `SMIDNV` built-in decoration will make that variable contain the ID of the SM on which the current shader invocation is running. This variable is in the range [0, `SMCountNV`-1].

### Valid Usage

- VUID-SMIDNV-SMIDNV-04365
  The variable decorated with `SMIDNV` **must** be declared using the `Input Storage Class`
- VUID-SMIDNV-SMIDNV-04366

The variable decorated with SMIDNV **must** be declared as a scalar 32-bit integer value

#### SubgroupId

Decorating a variable with the SubgroupId built-in decoration will make that variable contain the index of the subgroup within the local workgroup. This variable is in range [0, NumSubgroups-1].

### Valid Usage

- VUID-SubgroupId-SubgroupId-04367
  The SubgroupId decoration **must** be used only within the GLCompute, MeshEXT, TaskEXT, MeshNV, or TaskNV Execution Model
- VUID-SubgroupId-SubgroupId-04368
  The variable decorated with SubgroupId **must** be declared using the Input Storage Class
- VUID-SubgroupId-SubgroupId-04369
  The variable decorated with SubgroupId **must** be declared as a scalar 32-bit integer value

#### SubgroupEqMask

Decorating a variable with the SubgroupEqMask builtin decoration will make that variable contain the *subgroup mask* of the current subgroup invocation. The bit corresponding to the SubgroupLocalInvocationId is set in the variable decorated with SubgroupEqMask. All other bits are set to zero.

SubgroupEqMaskKHR is an alias of SubgroupEqMask.

### Valid Usage

- VUID-SubgroupEqMask-SubgroupEqMask-04370
  The variable decorated with SubgroupEqMask **must** be declared using the Input Storage Class
- VUID-SubgroupEqMask-SubgroupEqMask-04371
  The variable decorated with SubgroupEqMask **must** be declared as a four-component vector of 32-bit integer values

#### SubgroupGeMask

Decorating a variable with the SubgroupGeMask builtin decoration will make that variable contain the *subgroup mask* of the current subgroup invocation. The bits corresponding to the invocations greater than or equal to SubgroupLocalInvocationId through SubgroupSize-1 are set in the variable decorated with SubgroupGeMask. All other bits are set to zero.

SubgroupGeMaskKHR is an alias of SubgroupGeMask.

### Valid Usage

- VUID-SubgroupGeMask-SubgroupGeMask-04372

The variable decorated with `SubgroupGeMask` **must** be declared using the `Input Storage Class`

- VUID-SubgroupGeMask-SubgroupGeMask-04373
  The variable decorated with `SubgroupGeMask` **must** be declared as a four-component vector of 32-bit integer values

`SubgroupGtMask`

Decorating a variable with the `SubgroupGtMask` builtin decoration will make that variable contain the *subgroup mask* of the current subgroup invocation. The bits corresponding to the invocations greater than `SubgroupLocalInvocationId` through `SubgroupSize`-1 are set in the variable decorated with `SubgroupGtMask`. All other bits are set to zero.

`SubgroupGtMaskKHR` is an alias of `SubgroupGtMask`.

### Valid Usage

- VUID-SubgroupGtMask-SubgroupGtMask-04374
  The variable decorated with `SubgroupGtMask` **must** be declared using the `Input Storage Class`
- VUID-SubgroupGtMask-SubgroupGtMask-04375
  The variable decorated with `SubgroupGtMask` **must** be declared as a four-component vector of 32-bit integer values

`SubgroupLeMask`

Decorating a variable with the `SubgroupLeMask` builtin decoration will make that variable contain the *subgroup mask* of the current subgroup invocation. The bits corresponding to the invocations less than or equal to `SubgroupLocalInvocationId` are set in the variable decorated with `SubgroupLeMask`. All other bits are set to zero.

`SubgroupLeMaskKHR` is an alias of `SubgroupLeMask`.

### Valid Usage

- VUID-SubgroupLeMask-SubgroupLeMask-04376
  The variable decorated with `SubgroupLeMask` **must** be declared using the `Input Storage Class`
- VUID-SubgroupLeMask-SubgroupLeMask-04377
  The variable decorated with `SubgroupLeMask` **must** be declared as a four-component vector of 32-bit integer values

`SubgroupLtMask`

Decorating a variable with the `SubgroupLtMask` builtin decoration will make that variable contain the *subgroup mask* of the current subgroup invocation. The bits corresponding to the invocations less than `SubgroupLocalInvocationId` are set in the variable decorated with

SubgroupLtMask. All other bits are set to zero.

SubgroupLtMaskKHR is an alias of SubgroupLtMask.

### Valid Usage

- VUID-SubgroupLtMask-SubgroupLtMask-04378
  The variable decorated with SubgroupLtMask **must** be declared using the Input Storage Class
- VUID-SubgroupLtMask-SubgroupLtMask-04379
  The variable decorated with SubgroupLtMask **must** be declared as a four-component vector of 32-bit integer values

#### SubgroupLocalInvocationId

Decorating a variable with the SubgroupLocalInvocationId builtin decoration will make that variable contain the index of the invocation within the subgroup. This variable is in range [0,SubgroupSize-1].

If VK_PIPELINE_SHADER_STAGE_CREATE_REQUIRE_FULL_SUBGROUPS_BIT is specified, full subgroups are enabled for that pipeline stage. When full subgroups are enabled, subgroups **must** be launched with all invocations active, i.e., there is an active invocation with SubgroupLocalInvocationId for each value in range [0,SubgroupSize-1].

**NOTE**

There is no direct relationship between SubgroupLocalInvocationId and LocalInvocationId or LocalInvocationIndex. If the pipeline or shader object was created with full subgroups applications can compute their own local invocation index to serve the same purpose:

index = SubgroupLocalInvocationId + SubgroupId × SubgroupSize

If full subgroups are not enabled, some subgroups may be dispatched with inactive invocations that do not correspond to a local workgroup invocation, making the value of index unreliable.

### Valid Usage

- VUID-SubgroupLocalInvocationId-SubgroupLocalInvocationId-04380
  The variable decorated with SubgroupLocalInvocationId **must** be declared using the Input Storage Class
- VUID-SubgroupLocalInvocationId-SubgroupLocalInvocationId-04381
  The variable decorated with SubgroupLocalInvocationId **must** be declared as a scalar 32-bit integer value

#### SubgroupSize

Decorating a variable with the SubgroupSize builtin decoration will make that variable contain

the implementation-dependent number of invocations in a subgroup. This value **must** be a power-of-two integer.

If the pipeline was created with the VK_PIPELINE_SHADER_STAGE_CREATE_ALLOW_VARYING_SUBGROUP_SIZE_BIT flag set, or the shader object was created with the VK_SHADER_CREATE_ALLOW_VARYING_SUBGROUP_SIZE_BIT_EXT flag set, the SubgroupSize decorated variable will contain the subgroup size for each subgroup that gets dispatched. This value **must** be between minSubgroupSize and maxSubgroupSize and **must** be uniform with subgroup scope. The value **may** vary across a single draw call, and for fragment shaders **may** vary across a single primitive. In compute dispatches, SubgroupSize **must** be uniform with command scope.

If the pipeline was created with a chained VkPipelineShaderStageRequiredSubgroupSizeCreateInfo structure, or the shader object was created with a chained VkShaderRequiredSubgroupSizeCreateInfoEXT structure, the SubgroupSize decorated variable will match requiredSubgroupSize.

If the pipeline was not created with the VK_PIPELINE_SHADER_STAGE_CREATE_ALLOW_VARYING_SUBGROUP_SIZE_BIT flag set and no VkPipelineShaderStageRequiredSubgroupSizeCreateInfo structure was chained, and the shader was not created with the VK_SHADER_CREATE_ALLOW_VARYING_SUBGROUP_SIZE_BIT_EXT flag set and no VkShaderRequiredSubgroupSizeCreateInfoEXT structure was chained, the variable decorated with SubgroupSize will match subgroupSize.

The maximum number of invocations that an implementation can support per subgroup is 128.

### Valid Usage

- VUID-SubgroupSize-SubgroupSize-04382
  The variable decorated with SubgroupSize **must** be declared using the Input Storage Class
- VUID-SubgroupSize-SubgroupSize-04383
  The variable decorated with SubgroupSize **must** be declared as a scalar 32-bit integer value

#### TaskCountNV

Decorating a variable with the TaskCountNV decoration will make that variable contain the task count. The task count specifies the number of subsequent mesh shader workgroups that get generated upon completion of the task shader.

### Valid Usage

- VUID-TaskCountNV-TaskCountNV-04384
  The TaskCountNV decoration **must** be used only within the TaskNV Execution Model
- VUID-TaskCountNV-TaskCountNV-04385
  The variable decorated with TaskCountNV **must** be declared using the Output Storage Class
- VUID-TaskCountNV-TaskCountNV-04386

The variable decorated with `TaskCountNV` **must** be declared as a scalar 32-bit integer value

### `TessCoord`

Decorating a variable with the `TessCoord` built-in decoration will make that variable contain the three-dimensional (u,v,w) barycentric coordinate of the tessellated vertex within the patch. u, v, and w are in the range [0,1] and vary linearly across the primitive being subdivided. For the tessellation modes of `Quads` or `IsoLines`, the third component is always zero.

#### Valid Usage

- VUID-TessCoord-TessCoord-04387
  The `TessCoord` decoration **must** be used only within the `TessellationEvaluation Execution Model`
- VUID-TessCoord-TessCoord-04388
  The variable decorated with `TessCoord` **must** be declared using the `Input Storage Class`
- VUID-TessCoord-TessCoord-04389
  The variable decorated with `TessCoord` **must** be declared as a three-component vector of 32-bit floating-point values

### `TessLevelOuter`

Decorating a variable with the `TessLevelOuter` built-in decoration will make that variable contain the outer tessellation levels for the current patch.

In tessellation control shaders, the variable decorated with `TessLevelOuter` **can** be written to, controlling the tessellation factors for the resulting patch. These values are used by the tessellator to control primitive tessellation and **can** be read by tessellation evaluation shaders.

In tessellation evaluation shaders, the variable decorated with `TessLevelOuter` **can** read the values written by the tessellation control shader.

#### Valid Usage

- VUID-TessLevelOuter-TessLevelOuter-04390
  The `TessLevelOuter` decoration **must** be used only within the `TessellationControl` or `TessellationEvaluation Execution Model`
- VUID-TessLevelOuter-TessLevelOuter-04391
  The variable decorated with `TessLevelOuter` within the `TessellationControl Execution Model` **must** be declared using the `Output Storage Class`
- VUID-TessLevelOuter-TessLevelOuter-04392
  The variable decorated with `TessLevelOuter` within the `TessellationEvaluation Execution Model` **must** be declared using the `Input Storage Class`
- VUID-TessLevelOuter-TessLevelOuter-04393
  The variable decorated with `TessLevelOuter` **must** be declared as an array of size four, containing 32-bit floating-point values

`TessLevelInner`

Decorating a variable with the `TessLevelInner` built-in decoration will make that variable contain the inner tessellation levels for the current patch.

In tessellation control shaders, the variable decorated with `TessLevelInner` **can** be written to, controlling the tessellation factors for the resulting patch. These values are used by the tessellator to control primitive tessellation and **can** be read by tessellation evaluation shaders.

In tessellation evaluation shaders, the variable decorated with `TessLevelInner` **can** read the values written by the tessellation control shader.

### Valid Usage

- VUID-TessLevelInner-TessLevelInner-04394
  The `TessLevelInner` decoration **must** be used only within the `TessellationControl` or `TessellationEvaluation Execution Model`
- VUID-TessLevelInner-TessLevelInner-04395
  The variable decorated with `TessLevelInner` within the `TessellationControl Execution Model` **must** be declared using the `Output Storage Class`
- VUID-TessLevelInner-TessLevelInner-04396
  The variable decorated with `TessLevelInner` within the `TessellationEvaluation Execution Model` **must** be declared using the `Input Storage Class`
- VUID-TessLevelInner-TessLevelInner-04397
  The variable decorated with `TessLevelInner` **must** be declared as an array of size two, containing 32-bit floating-point values

`VertexCountHUAWEI`

The `VertexCountHUAWEI` decoration can be used to decorate a cluster culling shader output variable,this non-indexed mode specific variable will contain an integer value that specifies the number of vertices in a cluster to draw.

### Valid Usage

- VUID-VertexCountHUAWEI-VertexCountHUAWEI-07809
  The `VertexCountHUAWEI` decoration **must** be used only within the `ClusterCullingHUAWEI Execution Model`
- VUID-VertexCountHUAWEI-VertexCountHUAWEI-07810
  The variable decorated with `VertexCountHUAWEI` **must** be declared as a scalar 32-bit integer value

`VertexIndex`

Decorating a variable with the `VertexIndex` built-in decoration will make that variable contain the index of the vertex that is being processed by the current vertex shader invocation. For non-indexed draws, this variable begins at the `firstVertex` parameter to vkCmdDraw or the

firstVertex member of a structure consumed by vkCmdDrawIndirect and increments by one for each vertex in the draw. For indexed draws, its value is the content of the index buffer for the vertex plus the vertexOffset parameter to vkCmdDrawIndexed or the vertexOffset member of the structure consumed by vkCmdDrawIndexedIndirect.

**NOTE** VertexIndex starts at the same starting value for each instance.

### Valid Usage

- VUID-VertexIndex-VertexIndex-04398
  The VertexIndex decoration **must** be used only within the Vertex Execution Model
- VUID-VertexIndex-VertexIndex-04399
  The variable decorated with VertexIndex **must** be declared using the Input Storage Class
- VUID-VertexIndex-VertexIndex-04400
  The variable decorated with VertexIndex **must** be declared as a scalar 32-bit integer value

#### VertexOffsetHUAWEI

The VertexOffsetHUAWEI decoration can be used to decorate a cluster culling shader output variable,this indexed mode specific variable will contain an integer value that specifies an offset value added to the vertex index of a cluster before indexing into the vertex buffer.

### Valid Usage

- VUID-VertexOffsetHUAWEI-VertexOffsetHUAWEI-07811
  The VertexOffsetHUAWEI decoration **must** be used only within the ClusterCullingHUAWEI Execution Model
- VUID-VertexOffsetHUAWEI-VertexOffsetHUAWEI-07812
  The variable decorated with VertexOffsetHUAWEI **must** be declared as a scalar 32-bit integer value

#### ViewIndex

The ViewIndex decoration **can** be applied to a shader input which will be filled with the index of the view that is being processed by the current shader invocation.

If multiview is enabled in the render pass, this value will be one of the bits set in the view mask of the subpass the pipeline is compiled against. If multiview is not enabled in the render pass, this value will be zero.

### Valid Usage

- VUID-ViewIndex-ViewIndex-04401
  The ViewIndex decoration **must** be used only within the MeshEXT, Vertex, Geometry, TessellationControl, TessellationEvaluation or Fragment Execution Model
- VUID-ViewIndex-ViewIndex-04402

The variable decorated with `ViewIndex` **must** be declared using the `Input Storage Class`

- VUID-ViewIndex-ViewIndex-04403
  The variable decorated with `ViewIndex` **must** be declared as a scalar 32-bit integer value

`ViewportIndex`

Decorating a variable with the `ViewportIndex` built-in decoration will make that variable contain the index of the viewport.

In a mesh, vertex, tessellation evaluation, or geometry shader, the variable decorated with `ViewportIndex` can be written to with the viewport index to which the primitive produced by that shader will be directed.

The selected viewport index is used to select the viewport transform, scissor rectangle, and exclusive scissor rectangle.

The last active *pre-rasterization shader stage* (in pipeline order) controls the `ViewportIndex` that is used. Outputs in previous shader stages are not used, even if the last stage fails to write the `ViewportIndex`.

If the last active pre-rasterization shader stage shader entry point's interface does not include a variable decorated with `ViewportIndex` , and if multiviewPerViewViewports is not enabled, then the first viewport is used. If a pre-rasterization shader stage shader entry point's interface includes a variable decorated with `ViewportIndex`, it **must** write the same value to `ViewportIndex` for all output vertices of a given primitive.

In a fragment shader, the variable decorated with `ViewportIndex` contains the viewport index of the primitive that the fragment invocation belongs to.

If `multiviewPerViewViewports` is enabled, and if the last active pre-rasterization shader stage shader entry point's interface does not include a variable decorated with `ViewportIndex`, then the value of `ViewIndex` is used as an index to select the viewport transform and scissor rectangle, and the value of `ViewportIndex` in the fragment shader is undefined:.

### Valid Usage

- VUID-ViewportIndex-ViewportIndex-04404
  The `ViewportIndex` decoration **must** be used only within the `MeshEXT`, `MeshNV`, `Vertex`, `TessellationEvaluation`, `Geometry`, or `Fragment Execution Model`
- VUID-ViewportIndex-ViewportIndex-04406
  The variable decorated with `ViewportIndex` within the `MeshEXT`, `MeshNV`, `Vertex`, `TessellationEvaluation`, or `Geometry Execution Model` **must** be declared using the `Output Storage Class`
- VUID-ViewportIndex-ViewportIndex-04407
  The variable decorated with `ViewportIndex` within the `Fragment Execution Model` **must** be declared using the `Input Storage Class`
- VUID-ViewportIndex-ViewportIndex-04408

The variable decorated with `ViewportIndex` **must** be declared as a scalar 32-bit integer value

- VUID-ViewportIndex-ViewportIndex-07060
  The variable decorated with `ViewportIndex` within the `MeshEXT Execution Model` **must** also be decorated with the `PerPrimitiveEXT` decoration

`ViewportMaskNV`

Decorating a variable with the `ViewportMaskNV` built-in decoration will make that variable contain the viewport mask.

In a mesh, vertex, tessellation evaluation, or geometry shader, the variable decorated with `ViewportMaskNV` can be written to with the mask of which viewports the primitive produced by that shader will directed.

The `ViewportMaskNV` variable **must** be an array that has ⌈(`VkPhysicalDeviceLimits`::`maxViewports` / 32)⌉ elements. When a shader writes to this variable, bit B of element M controls whether a primitive is emitted to viewport 32 × M + B. The viewports indicated by the mask are used to select the viewport transform, scissor rectangle, and exclusive scissor rectangle that a primitive will be transformed by.

The last active *pre-rasterization shader stage* (in pipeline order) controls the `ViewportMaskNV` that is used. Outputs in previous shader stages are not used, even if the last stage fails to write the `ViewportMaskNV`. When `ViewportMaskNV` is written by the final pre-rasterization shader stage, any variable decorated with `ViewportIndex` in the fragment shader will have the index of the viewport that was used in generating that fragment.

If a pre-rasterization shader stage shader entry point's interface includes a variable decorated with `ViewportMaskNV`, it **must** write the same value to `ViewportMaskNV` for all output vertices of a given primitive.

### Valid Usage

- VUID-ViewportMaskNV-ViewportMaskNV-04409
  The `ViewportMaskNV` decoration **must** be used only within the `Vertex`, `MeshNV`, `TessellationEvaluation`, or `Geometry Execution Model`
- VUID-ViewportMaskNV-ViewportMaskNV-04410
  The variable decorated with `ViewportMaskNV` **must** be declared using the `Output Storage Class`
- VUID-ViewportMaskNV-ViewportMaskNV-04411
  The variable decorated with `ViewportMaskNV` **must** be declared as an array of 32-bit integer values

`ViewportMaskPerViewNV`

Decorating a variable with the `ViewportMaskPerViewNV` built-in decoration will make that variable contain the mask of viewports primitives are broadcast to, for each view.

The value written to an element of `ViewportMaskPerViewNV` in the last pre-rasterization shader stage is a bitmask indicating which viewports the primitive will be directed to. The primitive will be broadcast to the viewport corresponding to each non-zero bit of the bitmask, and that viewport index is used to select the viewport transform, scissor rectangle, and exclusive scissor rectangle, for each view. The same values **must** be written to all vertices in a given primitive, or else the set of viewports used for that primitive is undefined.

Elements of the array correspond to views in a multiview subpass, and those elements corresponding to views in the view mask of the subpass the shader is compiled against will be used as the viewport mask value for those views. `ViewportMaskPerViewNV` output in an earlier pre-rasterization shader stage is not available as an input in the subsequent pre-rasterization shader stage.

Although `ViewportMaskNV` is an array, `ViewportMaskPerViewNV` is not a two-dimensional array. Instead, `ViewportMaskPerViewNV` is limited to 32 viewports.

### Valid Usage

- VUID-ViewportMaskPerViewNV-ViewportMaskPerViewNV-04412
  The `ViewportMaskPerViewNV` decoration **must** be used only within the `Vertex`, `MeshNV`, `TessellationControl`, `TessellationEvaluation`, or `Geometry Execution Model`
- VUID-ViewportMaskPerViewNV-ViewportMaskPerViewNV-04413
  The variable decorated with `ViewportMaskPerViewNV` **must** be declared using the `Output Storage Class`
- VUID-ViewportMaskPerViewNV-ViewportMaskPerViewNV-04414
  The variable decorated with `ViewportMaskPerViewNV` **must** be declared as an array of 32-bit integer values
- VUID-ViewportMaskPerViewNV-ViewportMaskPerViewNV-04415
  The array decorated with `ViewportMaskPerViewNV` **must** be a size less than or equal to 32
- VUID-ViewportMaskPerViewNV-ViewportMaskPerViewNV-04416
  The array decorated with `ViewportMaskPerViewNV` **must** be a size greater than the maximum view in the subpass's view mask
- VUID-ViewportMaskPerViewNV-ViewportMaskPerViewNV-04417
  The array variable decorated with `ViewportMaskPerViewNV` **must** only be indexed by a constant or specialization constant

#### `WarpsPerSMNV`

Decorating a variable with the `WarpsPerSMNV` built-in decoration will make that variable contain the maximum number of warps executing on a SM.

### Valid Usage

- VUID-WarpsPerSMNV-WarpsPerSMNV-04418
  The variable decorated with `WarpsPerSMNV` **must** be declared using the `Input Storage Class`

- VUID-WarpsPerSMNV-WarpsPerSMNV-04419
  The variable decorated with `WarpsPerSMNV` **must** be declared as a scalar 32-bit integer value

`WarpIDNV`

Decorating a variable with the `WarpIDNV` built-in decoration will make that variable contain the ID of the warp on a SM on which the current shader invocation is running. This variable is in the range [0, `WarpsPerSMNV`-1].

### Valid Usage

- VUID-WarpIDNV-WarpIDNV-04420
  The variable decorated with `WarpIDNV` **must** be declared using the `Input Storage Class`
- VUID-WarpIDNV-WarpIDNV-04421
  The variable decorated with `WarpIDNV` **must** be declared as a scalar 32-bit integer value

`WorkgroupId`

Decorating a variable with the `WorkgroupId` built-in decoration will make that variable contain the global workgroup that the current invocation is a member of. Each component ranges from a base value to a base + count value, based on the parameters passed into the dispatching commands.

### Valid Usage

- VUID-WorkgroupId-WorkgroupId-04422
  The `WorkgroupId` decoration **must** be used only within the `GLCompute`, `MeshEXT`, `TaskEXT`, `MeshNV`, or `TaskNV Execution Model`
- VUID-WorkgroupId-WorkgroupId-04423
  The variable decorated with `WorkgroupId` **must** be declared using the `Input Storage Class`
- VUID-WorkgroupId-WorkgroupId-04424
  The variable decorated with `WorkgroupId` **must** be declared as a three-component vector of 32-bit integer values

`WorkgroupSize`

Decorating an object with the `WorkgroupSize` built-in decoration will make that object contain the dimensions of a local workgroup. If an object is decorated with the `WorkgroupSize` decoration, this takes precedence over any `LocalSize` or `LocalSizeId` execution mode.

### Valid Usage

- VUID-WorkgroupSize-WorkgroupSize-04425
  The `WorkgroupSize` decoration **must** be used only within the `GLCompute`, `MeshEXT`, `TaskEXT`, `MeshNV`, or `TaskNV Execution Model`

- VUID-WorkgroupSize-WorkgroupSize-04426
  The variable decorated with `WorkgroupSize` **must** be a specialization constant or a constant
- VUID-WorkgroupSize-WorkgroupSize-04427
  The variable decorated with `WorkgroupSize` **must** be declared as a three-component vector of 32-bit integer values

`WorldRayDirectionKHR`

A variable decorated with the `WorldRayDirectionKHR` decoration will specify the direction of the ray being processed, in world space. The value is the parameter passed into the pipeline trace ray instruction.

### Valid Usage

- VUID-WorldRayDirectionKHR-WorldRayDirectionKHR-04428
  The `WorldRayDirectionKHR` decoration **must** be used only within the `IntersectionKHR`, `AnyHitKHR`, `ClosestHitKHR`, or `MissKHR` `Execution Model`
- VUID-WorldRayDirectionKHR-WorldRayDirectionKHR-04429
  The variable decorated with `WorldRayDirectionKHR` **must** be declared using the `Input` `Storage Class`
- VUID-WorldRayDirectionKHR-WorldRayDirectionKHR-04430
  The variable decorated with `WorldRayDirectionKHR` **must** be declared as a three-component vector of 32-bit floating-point values

`WorldRayOriginKHR`

A variable decorated with the `WorldRayOriginKHR` decoration will specify the origin of the ray being processed, in world space. The value is the parameter passed into the pipeline trace ray instruction.

### Valid Usage

- VUID-WorldRayOriginKHR-WorldRayOriginKHR-04431
  The `WorldRayOriginKHR` decoration **must** be used only within the `IntersectionKHR`, `AnyHitKHR`, `ClosestHitKHR`, or `MissKHR` `Execution Model`
- VUID-WorldRayOriginKHR-WorldRayOriginKHR-04432
  The variable decorated with `WorldRayOriginKHR` **must** be declared using the `Input` `Storage Class`
- VUID-WorldRayOriginKHR-WorldRayOriginKHR-04433
  The variable decorated with `WorldRayOriginKHR` **must** be declared as a three-component vector of 32-bit floating-point values

`WorldToObjectKHR`

A variable decorated with the `WorldToObjectKHR` decoration will contain the current world-to-object transformation matrix, which is determined by the instance of the current intersection.

### Valid Usage

- VUID-WorldToObjectKHR-WorldToObjectKHR-04434
  The `WorldToObjectKHR` decoration **must** be used only within the `IntersectionKHR`, `AnyHitKHR`, or `ClosestHitKHR` `Execution Model`
- VUID-WorldToObjectKHR-WorldToObjectKHR-04435
  The variable decorated with `WorldToObjectKHR` **must** be declared using the `Input` `Storage Class`
- VUID-WorldToObjectKHR-WorldToObjectKHR-04436
  The variable decorated with `WorldToObjectKHR` **must** be declared as a matrix with four columns of three-component vectors of 32-bit floating-point values

`CoreCountARM`

Decorating a variable with the `CoreCountARM` built-in decoration will make that variable contain the number of cores on the device.

### Valid Usage

- VUID-CoreCountARM-CoreCountARM-07595
  The variable decorated with `CoreCountARM` **must** be declared using the `Input` `Storage Class`
- VUID-CoreCountARM-CoreCountARM-07596
  The variable decorated with `CoreCountARM` **must** be declared as a scalar 32-bit integer value

`CoreMaxIDARM`

Decorating a variable with the `CoreMaxIDARM` built-in decoration will make that variable contain the max ID of any shader core on the device on which the current shader invocation is running.

### Valid Usage

- VUID-CoreMaxIDARM-CoreMaxIDARM-07597
  The variable decorated with `CoreMaxIDARM` **must** be declared using the `Input` `Storage Class`
- VUID-CoreMaxIDARM-CoreMaxIDARM-07598
  The variable decorated with `CoreMaxIDARM` **must** be declared as a scalar 32-bit integer value

`CoreIDARM`

Decorating a variable with the `CoreIDARM` built-in decoration will make that variable contain the ID of the core on which the current shader invocation is running. This variable is in the range [0, `CoreMaxIDARM`].

### Valid Usage

- VUID-CoreIDARM-CoreIDARM-07599
  The variable decorated with `CoreIDARM` **must** be declared using the `Input Storage Class`
- VUID-CoreIDARM-CoreIDARM-07600
  The variable decorated with `CoreIDARM` **must** be declared as a scalar 32-bit integer value

`WarpMaxIDARM`

Decorating a variable with the `WarpMaxIDARM` built-in decoration will make that variable contain the maximum warp ID for the core on which the current invocation is running.

### Valid Usage

- VUID-WarpMaxIDARM-WarpMaxIDARM-07601
  The variable decorated with `WarpMaxIDARM` **must** be declared using the `Input Storage Class`
- VUID-WarpMaxIDARM-WarpMaxIDARM-07602
  The variable decorated with `WarpMaxIDARM` **must** be declared as a scalar 32-bit integer value

`WarpIDARM`

Decorating a variable with the `WarpIDARM` built-in decoration will make that variable contain the ID of the warp on a core on which the current shader invocation is running. This variable is in the range [0, `WarpMaxIDARM`].

### Valid Usage

- VUID-WarpIDARM-WarpIDARM-07603
  The variable decorated with `WarpIDARM` **must** be declared using the `Input Storage Class`
- VUID-WarpIDARM-WarpIDARM-07604
  The variable decorated with `WarpIDARM` **must** be declared as a scalar 32-bit integer value

`CoalescedInputCountAMDX`

Decorating a variable with the `CoalescedInputCountAMDX` built-in decoration will make that variable contain the number of node dispatches that the implementation coalesced into the input for the current shader. This variable will take a value in the range [1, arraySize), where arraySize is the maximum size of the input payload array for the shader.

### Valid Usage

- VUID-CoalescedInputCountAMDX-CoalescedInputCountAMDX-09172
  The variable decorated with `CoalescedInputCountAMDX` **must** be declared using the `Input Storage Class`

- VUID-CoalescedInputCountAMDX-CoalescedInputCountAMDX-09173
  If a variable is decorated with `CoalescedInputCountAMDX`, the `CoalescingAMDX` execution mode **must** be declared
- VUID-CoalescedInputCountAMDX-CoalescedInputCountAMDX-09174
  The variable decorated with `CoalescedInputCountAMDX` **must** be declared as a scalar 32-bit integer value

`ShaderIndexAMDX`

Decorating a variable with the `ShaderIndexAMDX` built-in decoration will make that variable contain the index of the shader specified when it was compiled, either via VkPipelineShaderStageNodeCreateInfoAMDX::`index` or by the `ShaderIndexAMDX` execution mode.

### Valid Usage

- VUID-ShaderIndexAMDX-ShaderIndexAMDX-09175
  The variable decorated with `ShaderIndexAMDX` **must** be declared using the `Input Storage Class`
- VUID-ShaderIndexAMDX-ShaderIndexAMDX-09176
  The variable decorated with `ShaderIndexAMDX` **must** be declared as a scalar 32-bit integer value

# Chapter 16. Image Operations

## 16.1. Image Operations Overview

Vulkan Image Operations are operations performed by those SPIR-V Image Instructions which take an `OpTypeImage` (representing a `VkImageView`) or `OpTypeSampledImage` (representing a (`VkImageView`, `VkSampler`) pair). Read, write, and atomic operations also take texel coordinates as operands, and return a value based on a neighborhood of texture elements (*texels*) within the image. Query operations return properties of the bound image or of the lookup itself. The "Depth" operand of `OpTypeImage` is ignored.

| | |
|---|---|
| **NOTE** | Texel is a term which is a combination of the words texture and element. Early interactive computer graphics supported texture operations on textures, a small subset of the image operations on images described here. The discrete samples remain essentially equivalent, however, so we retain the historical term texel to refer to them. |

Image Operations include the functionality of the following SPIR-V Image Instructions:

- `OpImageSample*` and `OpImageSparseSample*` read one or more neighboring texels of the image, and filter the texel values based on the state of the sampler.
  - Instructions with `ImplicitLod` in the name determine the LOD used in the sampling operation based on the coordinates used in neighboring fragments.
  - Instructions with `ExplicitLod` in the name determine the LOD used in the sampling operation based on additional coordinates.
  - Instructions with `Proj` in the name apply homogeneous projection to the coordinates.
- `OpImageFetch` and `OpImageSparseFetch` return a single texel of the image. No sampler is used.
- `OpImage*Gather` and `OpImageSparse*Gather` read neighboring texels and return a single component of each.
- `OpImageRead` (and `OpImageSparseRead`) and `OpImageWrite` read and write, respectively, a texel in the image. No sampler is used.
- `OpImageSampleFootprintNV` identifies and returns information about the set of texels in the image that would be accessed by an equivalent `OpImageSample*` instruction.
- `OpImage*Dref*` instructions apply depth comparison on the texel values.
- `OpImageSparse*` instructions additionally return a sparse residency code.
- `OpImageQuerySize`, `OpImageQuerySizeLod`, `OpImageQueryLevels`, and `OpImageQuerySamples` return properties of the image descriptor that would be accessed. The image itself is not accessed.
- `OpImageQueryLod` returns the LOD parameters that would be used in a sample operation. The actual operation is not performed.
- `OpImageWeightedSampleQCOM` reads a 2D neighborhood of texels and computes a weighted average using weight values from a separate weight texture.
- `opImageBlockMatchSADQCOM` and `opTextureBlockMatchSSD` compare 2D neighborhoods of texels from

two textures.

- `OpImageBoxFilterQCOM` reads a 2D neighborhood of texels and computes a weighted average of the texels.
- `opImageBlockMatchWindowSADQCOM` and `opImageBlockMatchWindowSSDQCOM` compare 2D neighborhoods of texels from two textures with the comparison repeated across a window region in the target texture.
- `opImageBlockMatchGatherSADQCOM` and `opImageBlockMatchWindowSSDQCOM` compares four 2D neighborhoods of texels from a target texture with a single 2D neighborhood in the reference texture. The R component of each comparison is gathered and returned in the output.

### 16.1.1. Texel Coordinate Systems

Images are addressed by *texel coordinates*. There are three *texel coordinate systems*:

- normalized texel coordinates [0.0, 1.0]
- unnormalized texel coordinates [0.0, width / height / depth)
- integer texel coordinates [0, width / height / depth)

SPIR-V `OpImageFetch`, `OpImageSparseFetch`, `OpImageRead`, `OpImageSparseRead`, `opImageBlockMatchSADQCOM`, `opImageBlockMatchSSDQCOM`, `opImageBlockMatchWindowSADQCOM`, `opImageBlockMatchWindowSSDQCOM`, and `OpImageWrite` instructions use integer texel coordinates.

Other image instructions **can** use either normalized or unnormalized texel coordinates (selected by the `unnormalizedCoordinates` state of the sampler used in the instruction), but there are limitations on what operations, image state, and sampler state is supported. Normalized coordinates are logically converted to unnormalized as part of image operations, and certain steps are only performed on normalized coordinates. The array layer coordinate is always treated as unnormalized even when other coordinates are normalized.

Normalized texel coordinates are referred to as (s,t,r,q,a), with the coordinates having the following meanings:

- s: Coordinate in the first dimension of an image.
- t: Coordinate in the second dimension of an image.
- r: Coordinate in the third dimension of an image.
  - (s,t,r) are interpreted as a direction vector for Cube images.
- q: Fourth coordinate, for homogeneous (projective) coordinates.
- a: Coordinate for array layer.

The coordinates are extracted from the SPIR-V operand based on the dimensionality of the image variable and type of instruction. For `Proj` instructions, the components are in order (s, [t,] [r,] q), with t and r being conditionally present based on the `Dim` of the image. For non-`Proj` instructions, the coordinates are (s [,t] [,r] [,a]), with t and r being conditionally present based on the `Dim` of the image and a being conditionally present based on the `Arrayed` property of the image. Projective image instructions are not supported on `Arrayed` images.

Unnormalized texel coordinates are referred to as (u,v,w,a), with the coordinates having the following meanings:

- u: Coordinate in the first dimension of an image.
- v: Coordinate in the second dimension of an image.
- w: Coordinate in the third dimension of an image.
- a: Coordinate for array layer.

Only the u and v coordinates are directly extracted from the SPIR-V operand, because only 1D and 2D (non-`Arrayed`) dimensionalities support unnormalized coordinates. The components are in order (u [,v]), with v being conditionally present when the dimensionality is 2D. When normalized coordinates are converted to unnormalized coordinates, all four coordinates are used.

Integer texel coordinates are referred to as (i,j,k,l,n), with the coordinates having the following meanings:

- i: Coordinate in the first dimension of an image.
- j: Coordinate in the second dimension of an image.
- k: Coordinate in the third dimension of an image.
- l: Coordinate for array layer.
- n: Index of the sample within the texel.

They are extracted from the SPIR-V operand in order (i [,j] [,k] [,l] [,n]), with j and k conditionally present based on the `Dim` of the image, and l conditionally present based on the `Arrayed` property of the image. n is conditionally present and is taken from the `Sample` image operand.

If an accessed image was created from a view using VkImageViewSlicedCreateInfoEXT and accessed through a `VK_DESCRIPTOR_TYPE_STORAGE_IMAGE` descriptor, then the value of k is incremented by VkImageViewSlicedCreateInfoEXT::`sliceOffset`, giving k ← sliceOffset + k. The image's accessible range in the third dimension is k < sliceOffset + sliceCount. If VkImageViewSlicedCreateInfoEXT::`sliceCount` is `VK_REMAINING_3D_SLICES_EXT`, the range is inherited from the image's depth extent as specified by Image Mip Level Sizing.

For all coordinate types, unused coordinates are assigned a value of zero.

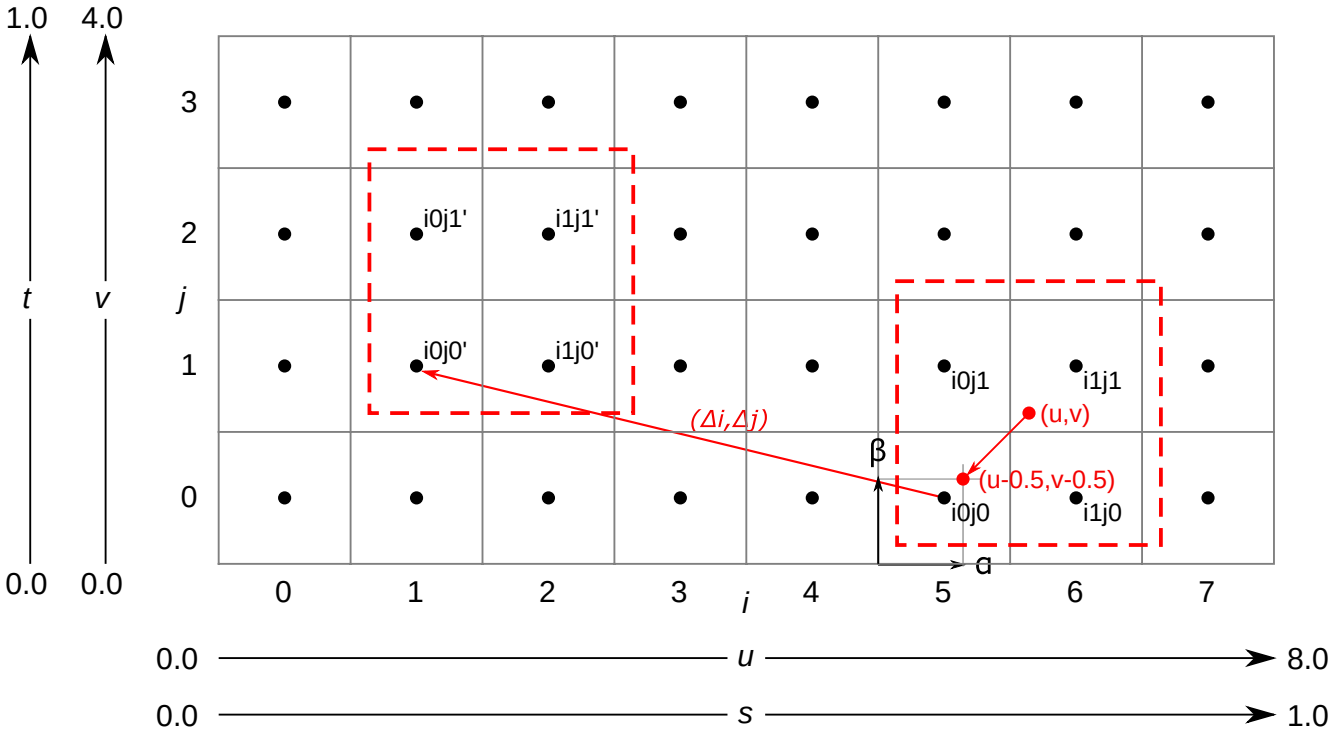



*Figure 3. Texel Coordinate Systems, Linear Filtering*

The Texel Coordinate Systems - For the example shown of an 8×4 texel two dimensional image.

- Normalized texel coordinates:
  - The s coordinate goes from 0.0 to 1.0.
  - The t coordinate goes from 0.0 to 1.0.
- Unnormalized texel coordinates:
  - The u coordinate within the range 0.0 to 8.0 is within the image, otherwise it is outside the image.
  - The v coordinate within the range 0.0 to 4.0 is within the image, otherwise it is outside the image.
- Integer texel coordinates:
  - The i coordinate within the range 0 to 7 addresses texels within the image, otherwise it is outside the image.
  - The j coordinate within the range 0 to 3 addresses texels within the image, otherwise it is outside the image.
- Also shown for linear filtering:
  - Given the unnormalized coordinates (u,v), the four texels selected are $i_0j_0$, $i_1j_0$, $i_0j_1$, and $i_1j_1$.
  - The fractions α and β.
  - Given the offset $\Delta_i$ and $\Delta_j$, the four texels selected by the offset are $i_0j'_0$, $i_1j'_0$, $i_0j'_1$, and $i_1j'_1$.

| | |
|---|---|
| **NOTE** | For formats with reduced-resolution components, $\Delta_i$ and $\Delta_j$ are relative to the resolution of the highest-resolution component, and therefore may be divided by two relative to the unnormalized coordinate space of the lower-resolution components. |

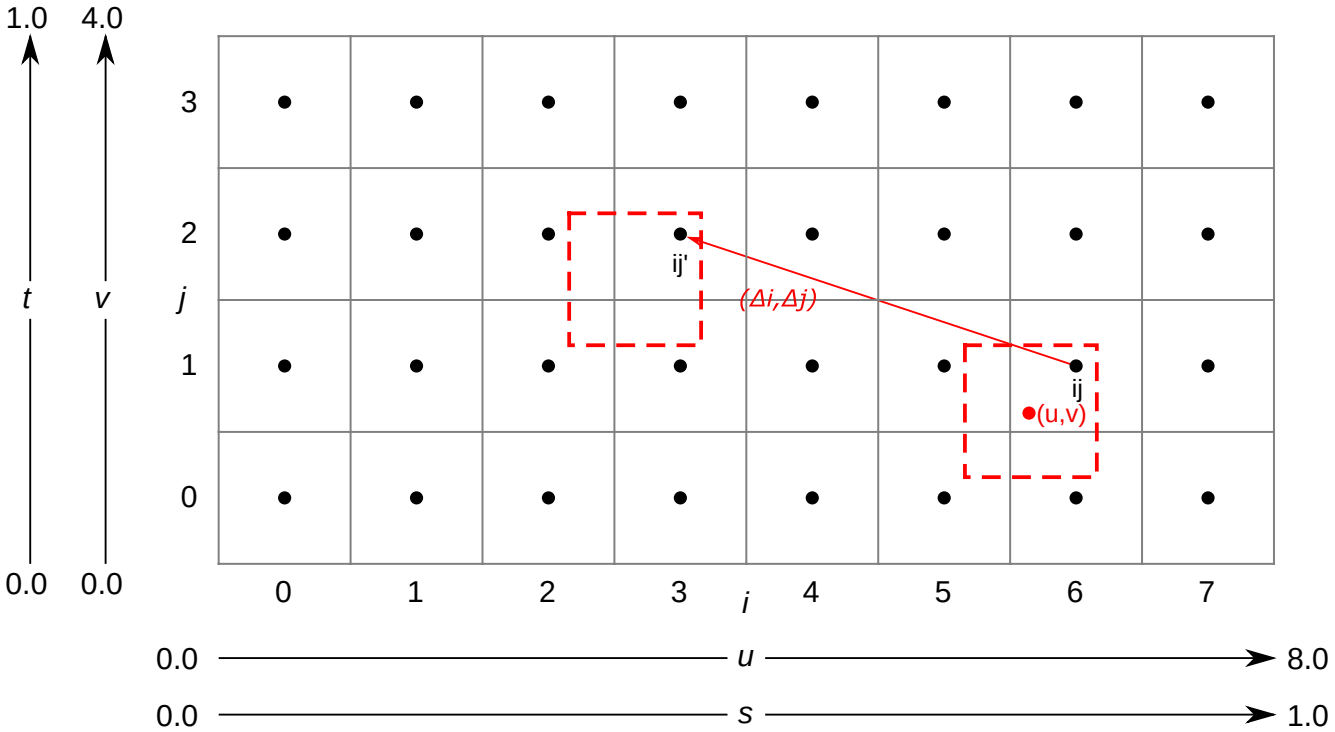



*Figure 4. Texel Coordinate Systems, Nearest Filtering*

The Texel Coordinate Systems - For the example shown of an 8×4 texel two dimensional image.

- Texel coordinates as above. Also shown for nearest filtering:
  - Given the unnormalized coordinates (u,v), the texel selected is ij.
  - Given the offset $\Delta_i$ and $\Delta_j$, the texel selected by the offset is ij'.

For corner-sampled images, the texel samples are located at the grid intersections instead of the texel centers.

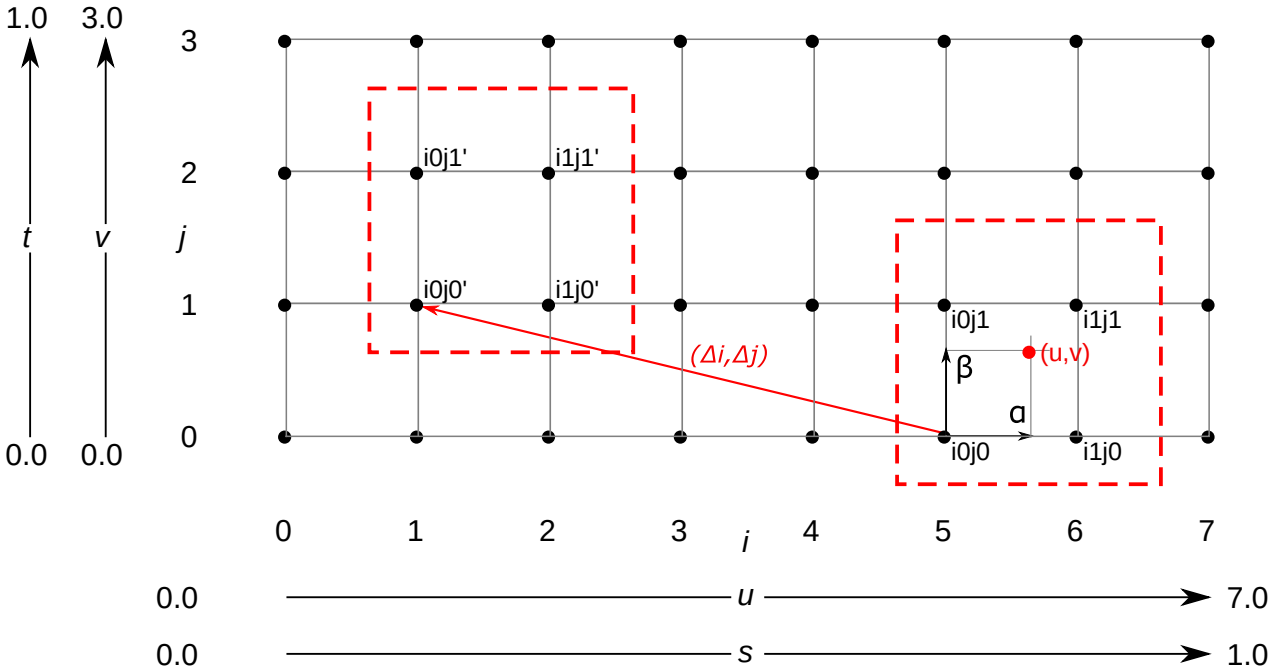



*Figure 5. Texel Coordinate Systems, Corner Sampling*

# 16.2. Conversion Formulas

## 16.2.1. RGB to Shared Exponent Conversion

An RGB color (red, green, blue) is transformed to a shared exponent color ($red_{shared}$, $green_{shared}$, $blue_{shared}$, $exp_{shared}$) as follows:

First, the components (red, green, blue) are clamped to ($red_{clamped}$, $green_{clamped}$, $blue_{clamped}$) as:

$$red_{clamped} = \max(0, \min(sharedexp_{max}, red))$$

$$green_{clamped} = \max(0, \min(sharedexp_{max}, green))$$

$$blue_{clamped} = \max(0, \min(sharedexp_{max}, blue))$$

where:

$$\begin{aligned} N &= 9 && \text{number of mantissa bits per component} \\ B &= 15 && \text{exponent bias} \\ E_{max} &= 31 && \text{maximum possible biased exponent value} \\ sharedexp_{max} &= \frac{(2^N - 1)}{2^N} \times 2^{(E_{max} - B)} \end{aligned}$$

**NOTE** NaN, if supported, is handled as in IEEE 754-2008 `minNum()` and `maxNum()`. This results in any NaN being mapped to zero.

The largest clamped component, $max_{clamped}$ is determined:

$$max_{clamped} = \max(red_{clamped}, green_{clamped}, blue_{clamped})$$

A preliminary shared exponent exp' is computed:

$$exp' = \begin{cases} \lfloor \log_2(max_{clamped}) \rfloor + (B+1) & \text{for } max_{clamped} > 2^{-(B+1)} \\ 0 & \text{for } max_{clamped} \le 2^{-(B+1)} \end{cases}$$

The shared exponent $exp_{shared}$ is computed:

$$max_{shared} = \lfloor \frac{max_{clamped}}{2^{(exp' - B - N)}} + \frac{1}{2} \rfloor$$

$$exp_{shared} = \begin{cases} exp' & \text{for } 0 \le max_{shared} < 2^N \\ exp' + 1 & \text{for } max_{shared} = 2^N \end{cases}$$

Finally, three integer values in the range 0 to $2^N$ are computed:

$$red_{shared} = \lfloor \frac{red_{clamped}}{2^{(exp_{shared} - B - N)}} + \frac{1}{2} \rfloor$$
$$green_{shared} = \lfloor \frac{green_{clamped}}{2^{(exp_{shared} - B - N)}} + \frac{1}{2} \rfloor$$
$$blue_{shared} = \lfloor \frac{blue_{clamped}}{2^{(exp_{shared} - B - N)}} + \frac{1}{2} \rfloor$$

### 16.2.2. Shared Exponent to RGB

A shared exponent color ($red_{shared}$, $green_{shared}$, $blue_{shared}$, $exp_{shared}$) is transformed to an RGB color (red, green, blue) as follows:

$$red = red_{shared} \times 2^{(exp_{shared} - B - N)}$$

$$green = green_{shared} \times 2^{(exp_{shared} - B - N)}$$

$$blue = blue_{shared} \times 2^{(exp_{shared} - B - N)}$$

where:

N = 9 (number of mantissa bits per component)

B = 15 (exponent bias)

## 16.3. Texel Input Operations

*Texel input instructions* are SPIR-V image instructions that read from an image. *Texel input operations* are a set of steps that are performed on state, coordinates, and texel values while processing a texel input instruction, and which are common to some or all texel input instructions. They include the following steps, which are performed in the listed order:

- Validation operations
  - Instruction/Sampler/Image validation
  - Coordinate validation
  - Sparse validation
  - Layout validation
- Format conversion
- Texel replacement

- Depth comparison
- Conversion to RGBA
- Component swizzle
- Chroma reconstruction
- $Y'C_BC_R$ conversion

For texel input instructions involving multiple texels (for sampling or gathering), these steps are applied for each texel that is used in the instruction. Depending on the type of image instruction, other steps are conditionally performed between these steps or involving multiple coordinate or texel values.

If Chroma Reconstruction is implicit, Texel Filtering instead takes place during chroma reconstruction, before sampler $Y'C_BC_R$ conversion occurs.

The operations described in block matching and weight image sampling are performed before Conversion to RGBA and Component swizzle.

## 16.3.1. Texel Input Validation Operations

*Texel input validation operations* inspect instruction/image/sampler state or coordinates, and in certain circumstances cause the texel value to be replaced or become undefined. There are a series of validations that the texel undergoes.

#### Instruction/Sampler/Image View Validation

There are a number of cases where a SPIR-V instruction **can** mismatch with the sampler, the image view, or both, and a number of further cases where the sampler **can** mismatch with the image view. In such cases the value of the texel returned is undefined.

These cases include:

- The sampler `borderColor` is an integer type and the image view `format` is not one of the VkFormat integer types or a stencil component of a depth/stencil format.
- The sampler `borderColor` is a float type and the image view `format` is not one of the VkFormat float types or a depth component of a depth/stencil format.
- The sampler `borderColor` is one of the opaque black colors (`VK_BORDER_COLOR_FLOAT_OPAQUE_BLACK` or `VK_BORDER_COLOR_INT_OPAQUE_BLACK`) and the image view VkComponentSwizzle for any of the VkComponentMapping components is not the identity swizzle, and VkPhysicalDeviceBorderColorSwizzleFeaturesEXT::`borderColorSwizzleFromImage` feature is not enabled, and VkSamplerBorderColorComponentMappingCreateInfoEXT is not specified.
- VkSamplerBorderColorComponentMappingCreateInfoEXT::`components`, if specified, has a component swizzle that does not match the component swizzle of the image view, and either component swizzle is not a form of identity swizzle.
- VkSamplerBorderColorComponentMappingCreateInfoEXT::`srgb`, if specified, does not match the sRGB encoding of the image view.
- The sampler `borderColor` is a custom color (`VK_BORDER_COLOR_FLOAT_CUSTOM_EXT` or

VK_BORDER_COLOR_INT_CUSTOM_EXT) and the supplied VkSamplerCustomBorderColorCreateInfoEXT::customBorderColor is outside the bounds of the values representable in the image view's format.

- The sampler borderColor is a custom color (VK_BORDER_COLOR_FLOAT_CUSTOM_EXT or VK_BORDER_COLOR_INT_CUSTOM_EXT) and the image view VkComponentSwizzle for any of the VkComponentMapping components is not the identity swizzle, and VkPhysicalDeviceBorderColorSwizzleFeaturesEXT::borderColorSwizzleFromImage feature is not enabled, and VkSamplerBorderColorComponentMappingCreateInfoEXT is not specified.
- The VkImageLayout of any subresource in the image view does not match the VkDescriptorImageInfo::imageLayout used to write the image descriptor.
- The SPIR-V Image Format is not compatible with the image view's format.
- The sampler unnormalizedCoordinates is VK_TRUE and any of the limitations of unnormalized coordinates are violated.
- The sampler was created with flags containing VK_SAMPLER_CREATE_SUBSAMPLED_BIT_EXT and the image was not created with flags containing VK_IMAGE_CREATE_SUBSAMPLED_BIT_EXT.
- The sampler was not created with flags containing VK_SAMPLER_CREATE_SUBSAMPLED_BIT_EXT and the image was created with flags containing VK_IMAGE_CREATE_SUBSAMPLED_BIT_EXT.
- The sampler was created with flags containing VK_SAMPLER_CREATE_SUBSAMPLED_BIT_EXT and is used with a function that is not OpImageSampleImplicitLod or OpImageSampleExplicitLod, or is used with operands Offset or ConstOffsets.
- The SPIR-V instruction is one of the OpImage*Dref* instructions and the sampler compareEnable is VK_FALSE
- The SPIR-V instruction is not one of the OpImage*Dref* instructions and the sampler compareEnable is VK_TRUE
- The SPIR-V instruction is one of the OpImage*Dref* instructions, the image view format is one of the depth/stencil formats, and the image view aspect is not VK_IMAGE_ASPECT_DEPTH_BIT.
- The SPIR-V instruction's image variable's properties are not compatible with the image view:
  - If the image view's viewType is one of VK_IMAGE_VIEW_TYPE_1D_ARRAY, VK_IMAGE_VIEW_TYPE_2D_ARRAY, or VK_IMAGE_VIEW_TYPE_CUBE_ARRAY then the instruction **must** have Arrayed = 1. Otherwise the instruction **must** have Arrayed = 0.
  - If the image was created with VkImageCreateInfo::samples equal to VK_SAMPLE_COUNT_1_BIT, the instruction **must** have MS = 0.
  - If the image was created with VkImageCreateInfo::samples not equal to VK_SAMPLE_COUNT_1_BIT, the instruction **must** have MS = 1.
  - If the Sampled Type of the OpTypeImage does not match the SPIR-V Type.
  - If the signedness of any read or sample operation does not match the signedness of the image's format.
- If the image was created with VkImageCreateInfo::flags containing VK_IMAGE_CREATE_CORNER_SAMPLED_BIT_NV, the sampler addressing modes **must** only use a VkSamplerAddressMode of VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_EDGE.
- The SPIR-V instruction is OpImageSampleFootprintNV with Dim = 2D and addressModeU or

addressModeV in the sampler is not VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_EDGE.

- The SPIR-V instruction is OpImageSampleFootprintNV with Dim = 3D and addressModeU, addressModeV, or addressModeW in the sampler is not VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_EDGE.
- The sampler was created with a specified VkSamplerCustomBorderColorCreateInfoEXT::format which does not match the VkFormat of the image view(s) it is sampling.
- The sampler is sampling an image view of VK_FORMAT_B4G4R4A4_UNORM_PACK16, VK_FORMAT_B5G6R5_UNORM_PACK16, or VK_FORMAT_B5G5R5A1_UNORM_PACK16 format without a specified VkSamplerCustomBorderColorCreateInfoEXT::format.

Only OpImageSample* and OpImageSparseSample* **can** be used with a sampler or image view that enables sampler $Y'C_BC_R$ conversion.

OpImageFetch, OpImageSparseFetch, OpImage*Gather, and OpImageSparse*Gather **must** not be used with a sampler or image view that enables sampler $Y'C_BC_R$ conversion.

The ConstOffset and Offset operands **must** not be used with a sampler or image view that enables sampler $Y'C_BC_R$ conversion.

If the underlying VkImage format has an X component in its format description, undefined values are read from those bits.

**NOTE** If the VkImage format and VkImageView format are the same, these bits will be unused by format conversion and this will have no effect. However, if the VkImageView format is different, then some bits of the result may be undefined. For example, when a VK_FORMAT_R10X6_UNORM_PACK16 VkImage is sampled via a VK_FORMAT_R16_UNORM VkImageView, the low 6 bits of the value before format conversion are undefined and format conversion may return a range of different values.

**NOTE** Some implementations will return undefined values in the case where a sampler uses a VkSamplerAddressMode of VK_SAMPLER_ADDRESS_MODE_MIRRORED_REPEAT, the sampler is used with operands Offset, ConstOffset, or ConstOffsets, and the value of the offset is larger than or equal to the corresponding width, height, or depth of any accessed image level.

This behavior was not tested prior to Vulkan conformance test suite version 1.3.8.0. Affected implementations will have a conformance test waiver for this issue.

#### Integer Texel Coordinate Validation

Integer texel coordinates are validated against the size of the image level, and the number of layers and number of samples in the image. For SPIR-V instructions that use integer texel coordinates, this is performed directly on the integer coordinates. For instructions that use normalized or unnormalized texel coordinates, this is performed on the coordinates that result after conversion to integer texel coordinates.

If the integer texel coordinates do not satisfy all of the conditions

$0 \leq i < w_s$

$0 \leq j < h_s$

$0 \leq k < d_s$

$0 \leq l < layers$

$0 \leq n < samples$

where:

$w_s$ = width of the image level

$h_s$ = height of the image level

$d_s$ = depth of the image level

layers = number of layers in the image

samples = number of samples per texel in the image

then the texel fails integer texel coordinate validation.

There are four cases to consider:

1. Valid Texel Coordinates
   - If the texel coordinates pass validation (that is, the coordinates lie within the image),

     then the texel value comes from the value in image memory.
2. Border Texel
   - If the texel coordinates fail validation, and
   - If the read is the result of an image sample instruction or image gather instruction, and
   - If the image is not a cube image, or if a sampler created with `VK_SAMPLER_CREATE_NON_SEAMLESS_CUBE_MAP_BIT_EXT` is used,

   then the texel is a border texel and texel replacement is performed.

3. Invalid Texel
   - If the texel coordinates fail validation, and
   - If the read is the result of an image fetch instruction, image read instruction, or atomic instruction,

     then the texel is an invalid texel and texel replacement is performed.
4. Cube Map Edge or Corner

   Otherwise the texel coordinates lie beyond the edges or corners of the selected cube map face, and Cube map edge handling is performed.

#### Cube Map Edge Handling

If the texel coordinates lie beyond the edges or corners of the selected cube map face (as described in the prior section), the following steps are performed. Note that this does not occur when using `VK_FILTER_NEAREST` filtering within a mip level, since `VK_FILTER_NEAREST` is treated as using `VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_EDGE`.

- Cube Map Edge Texel
  - If the texel lies beyond the selected cube map face in either only i or only j, then the coordinates (i,j) and the array layer l are transformed to select the adjacent texel from the appropriate neighboring face.
- Cube Map Corner Texel
  - If the texel lies beyond the selected cube map face in both i and j, then there is no unique neighboring face from which to read that texel. The texel **should** be replaced by the average of the three values of the adjacent texels in each incident face. However, implementations **may** replace the cube map corner texel by other methods. The methods are subject to the constraint that for linear filtering if the three available texels have the same value, the resulting filtered texel **must** have that value, and for cubic filtering if the twelve available samples have the same value, the resulting filtered texel **must** have that value.

#### Sparse Validation

If the texel reads from an unbound region of a sparse image, the texel is a *sparse unbound texel*, and processing continues with texel replacement.

#### Layout Validation

If all planes of a *disjoint multi-planar* image are not in the same image layout, the image **must** not be sampled with sampler $Y'C_BC_R$ conversion enabled.

### 16.3.2. Format Conversion

Texels undergo a format conversion from the VkFormat of the image view to a vector of either floating-point or signed or unsigned integer components, with the number of components based on the number of components present in the format.

- Color formats have one, two, three, or four components, according to the format.
- Depth/stencil formats are one component. The depth or stencil component is selected by the `aspectMask` of the image view.

Each component is converted based on its type and size (as defined in the Format Definition section for each VkFormat), using the appropriate equations in 16-Bit Floating-Point Numbers, Unsigned 11-Bit Floating-Point Numbers, Unsigned 10-Bit Floating-Point Numbers, Fixed-Point Data Conversion, and Shared Exponent to RGB. Signed integer components smaller than 32 bits are sign-extended.

If the image view format is sRGB, the color components are first converted as if they are UNORM, and then sRGB to linear conversion is applied to the R, G, and B components as described in the "sRGB EOTF" section of the Khronos Data Format Specification. The A component, if present, is unchanged.

If VkSamplerYcbcrConversionYcbcrDegammaCreateInfoQCOM::`enableYDegamma` is equal to `VK_TRUE`, then sRGB to linear conversion is applied to the G component as described in the "sRGB EOTF" section of the Khronos Data Format Specification. If VkSamplerYcbcrConversionYcbcrDegammaCreateInfoQCOM::`enableCbCrDegamma` is equal to `VK_TRUE`, then sRGB to linear conversion is applied to the R and B components as described in the "sRGB EOTF" section of the Khronos Data Format Specification. The A component, if present, is unchanged.

If the image view format is block-compressed, then the texel value is first decoded, then converted based on the type and number of components defined by the compressed format.

### 16.3.3. Texel Replacement

A texel is replaced if it is one (and only one) of:

- a border texel,
- an invalid texel, or
- a sparse unbound texel.

Border texels are replaced with a value based on the image format and the `borderColor` of the sampler. The border color is:

*Table 24. Border Color B, Custom Border Color VkSamplerCustomBorderColorCreateInfoEXT ::`customBorderColor` U*

| **Sampler** `borderColor` | **Corresponding Border Color** |
|---|---|
| `VK_BORDER_COLOR_FLOAT_TRANSPARENT_BLACK` | $[B_r, B_g, B_b, B_a] = [0.0, 0.0, 0.0, 0.0]$ |
| `VK_BORDER_COLOR_FLOAT_OPAQUE_BLACK` | $[B_r, B_g, B_b, B_a] = [0.0, 0.0, 0.0, 1.0]$ |
| `VK_BORDER_COLOR_FLOAT_OPAQUE_WHITE` | $[B_r, B_g, B_b, B_a] = [1.0, 1.0, 1.0, 1.0]$ |
| `VK_BORDER_COLOR_INT_TRANSPARENT_BLACK` | $[B_r, B_g, B_b, B_a] = [0, 0, 0, 0]$ |
| `VK_BORDER_COLOR_INT_OPAQUE_BLACK` | $[B_r, B_g, B_b, B_a] = [0, 0, 0, 1]$ |

| Sampler `borderColor` | Corresponding Border Color |
|---|---|
| `VK_BORDER_COLOR_INT_OPAQUE_WHITE` | $[B_r, B_g, B_b, B_a] = [1, 1, 1, 1]$ |
| `VK_BORDER_COLOR_FLOAT_CUSTOM_EXT` | $[B_r, B_g, B_b, B_a] = [U_r, U_g, U_b, U_a]$ |
| `VK_BORDER_COLOR_INT_CUSTOM_EXT` | $[B_r, B_g, B_b, B_a] = [U_r, U_g, U_b, U_a]$ |

The custom border color (U) **may** be rounded by implementations prior to texel replacement, but the error introduced by such a rounding **must** not exceed one ULP of the image's `format`.

**NOTE** The names `VK_BORDER_COLOR_*_TRANSPARENT_BLACK`, `VK_BORDER_COLOR_*_OPAQUE_BLACK`, and `VK_BORDER_COLOR_*_OPAQUE_WHITE` are meant to describe which components are zeros and ones in the vocabulary of compositing, and are not meant to imply that the numerical value of `VK_BORDER_COLOR_INT_OPAQUE_WHITE` is a saturating value for integers.

This is substituted for the texel value by replacing the number of components in the image format

*Table 25. Border Texel Components After Replacement*

| Texel Aspect or Format | Component Assignment |
|---|---|
| Depth aspect | $D = B_r$ |
| Stencil aspect | $S = B_r$† |
| One component color format | $Color_r = B_r$ |
| Two component color format | $[Color_r,Color_g] = [B_r,B_g]$ |
| Three component color format | $[Color_r,Color_g,Color_b] = [B_r,B_g,B_b]$ |
| Four component color format | $[Color_r,Color_g,Color_b,Color_a] = [B_r,B_g,B_b,B_a]$ |
| Single component alpha format | $[Color_r,Color_g,Color_b, Color_a] = [0,0,0,B_a]$ |

† $S = B_g$ **may** be substituted as the replacement method by the implementation when VkSamplerCreateInfo::`borderColor` is `VK_BORDER_COLOR_INT_CUSTOM_EXT` and VkSamplerCustomBorderColorCreateInfoEXT::`format` is `VK_FORMAT_UNDEFINED`. Implementations **should** use $S = B_r$ as the replacement method.

The value returned by a read of an invalid texel is undefined, unless that read operation is from a buffer resource and the `robustBufferAccess` feature is enabled. In that case, an invalid texel is replaced as described by the `robustBufferAccess` feature. If the access is to an image resource and the x, y, z, or layer coordinate validation fails and the `robustImageAccess` feature is enabled, then zero **must** be returned for the R, G, and B components, if present. Either zero or one **must** be returned for the A component, if present. If If the `robustImageAccess2` feature is enabled, zero values **must** be returned. If only the sample index was invalid, the values returned are undefined.

Additionally, if the `robustImageAccess` feature is enabled, but the `robustImageAccess2` feature is not, any invalid texels **may** be expanded to four components prior to texel replacement. This means that components not present in the image format may be replaced with 0 or may undergo conversion to RGBA as normal.

Loads from a null descriptor return a four component color value of all zeros. However, for storage images and storage texel buffers using an explicit SPIR-V Image Format, loads from a null descriptor **may** return an alpha value of 1 (float or integer, depending on format) if the format does not include alpha.

If the VkPhysicalDeviceSparseProperties::`residencyNonResidentStrict` property is `VK_TRUE`, a sparse unbound texel is replaced with 0 or 0.0 values for integer and floating-point components of the image format, respectively.

If `residencyNonResidentStrict` is `VK_FALSE`, the value of the sparse unbound texel is undefined.

### 16.3.4. Depth Compare Operation

If the image view has a depth/stencil format, the depth component is selected by the `aspectMask`, and the operation is an `OpImage*Dref*` instruction, a depth comparison is performed. The result is 1.0 if the comparison evaluates to true, and 0.0 otherwise. This value replaces the depth component D.

The compare operation is selected by the VkCompareOp value set by VkSamplerCreateInfo::`compareOp`. The reference value from the SPIR-V operand $D_{ref}$ and the texel depth value $D_{tex}$ are used as the *reference* and *test* values, respectively, in that operation.

If the image being sampled has an unsigned normalized fixed-point format, then $D_{ref}$ is clamped to [0,1] before the compare operation.

### 16.3.5. Conversion to RGBA

The texel is expanded from one, two, or three components to four components based on the image base color:

*Table 26. Texel Color After Conversion To RGBA*

| Texel Aspect or Format | RGBA Color |
|---|---|
| Depth aspect | $[Color_r,Color_g,Color_b, Color_a] = [D,0,0,one]$ |
| Stencil aspect | $[Color_r,Color_g,Color_b, Color_a] = [S,0,0,one]$ |
| One component color format | $[Color_r,Color_g,Color_b, Color_a] = [Color_r,0,0,one]$ |
| Two component color format | $[Color_r,Color_g,Color_b, Color_a] = [Color_r,Color_g,0,one]$ |
| Three component color format | $[Color_r,Color_g,Color_b, Color_a] = [Color_r,Color_g,Color_b,one]$ |
| Four component color format | $[Color_r,Color_g,Color_b, Color_a] = [Color_r,Color_g,Color_b,Color_a]$ |
| One alpha component color format | $[Color_r,Color_g,Color_b, Color_a] = [0,0,0,Color_a]$ |

where one = 1.0f for floating-point formats and depth aspects, and one = 1 for integer formats and stencil aspects.

### 16.3.6. Component Swizzle

All texel input instructions apply a *swizzle* based on:

- the VkComponentSwizzle enums in the `components` member of the VkImageViewCreateInfo structure for the image being read if sampler $Y'C_BC_R$ conversion is not enabled, and
- the VkComponentSwizzle enums in the `components` member of the VkSamplerYcbcrConversionCreateInfo structure for the sampler $Y'C_BC_R$ conversion if sampler $Y'C_BC_R$ conversion is enabled.

The swizzle **can** rearrange the components of the texel, or substitute zero or one for any components. It is defined as follows for each color component:

$$Color'_{component} = \begin{cases} Color_r & \text{for RED swizzle} \\ Color_g & \text{for GREEN swizzle} \\ Color_b & \text{for BLUE swizzle} \\ Color_a & \text{for ALPHA swizzle} \\ 0 & \text{for ZERO swizzle} \\ one & \text{for ONE swizzle} \\ identity & \text{for IDENTITY swizzle} \end{cases}$$

where:

$$one = \begin{cases} 1.0\text{f} & \text{for floating-point components} \\ 1 & \text{for integer components} \end{cases}$$

$$identity = \begin{cases} Color_r & \text{for}\, component = r \\ Color_g & \text{for}\, component = g \\ Color_b & \text{for}\, component = b \\ Color_a & \text{for}\, component = a \end{cases}$$

If the border color is one of the `VK_BORDER_COLOR_*_OPAQUE_BLACK` enums and the VkComponentSwizzle is not the identity swizzle for all components, the value of the texel after swizzle is undefined.

If the image view has a depth/stencil format and the VkComponentSwizzle is `VK_COMPONENT_SWIZZLE_ONE`, and `VkPhysicalDeviceMaintenance5PropertiesKHR::depthStencilSwizzleOneSupport` is not set to `VK_TRUE`, the value of the texel after swizzle is undefined.

### 16.3.7. Sparse Residency

`OpImageSparse*` instructions return a structure which includes a *residency code* indicating whether any texels accessed by the instruction are sparse unbound texels. This code **can** be interpreted by the `OpImageSparseTexelsResident` instruction which converts the residency code to a boolean value.

### 16.3.8. Chroma Reconstruction

In some color models, the color representation is defined in terms of monochromatic light intensity (often called “luma”) and color differences relative to this intensity, often called “chroma”. It is common for color models other than RGB to represent the chroma components at lower spatial

resolution than the luma component. This approach is used to take advantage of the eye's lower spatial sensitivity to color compared with its sensitivity to brightness. Less commonly, the same approach is used with additive color, since the green component dominates the eye's sensitivity to light intensity and the spatial sensitivity to color introduced by red and blue is lower.

Lower-resolution components are "downsampled" by resizing them to a lower spatial resolution than the component representing luminance. This process is also commonly known as "chroma subsampling". There is one luminance sample in each texture texel, but each chrominance sample may be shared among several texels in one or both texture dimensions.

- "`_444`" formats do not spatially downsample chroma values compared with luma: there are unique chroma samples for each texel.
- "`_422`" formats have downsampling in the x dimension (corresponding to $u$ or $s$ coordinates): they are sampled at half the resolution of luma in that dimension.
- "`_420`" formats have downsampling in the x dimension (corresponding to $u$ or $s$ coordinates) and the y dimension (corresponding to $v$ or $t$ coordinates): they are sampled at half the resolution of luma in both dimensions.

The process of reconstructing a full color value for texture access involves accessing both chroma and luma values at the same location. To generate the color accurately, the values of the lower-resolution components at the location of the luma samples are reconstructed from the lower-resolution sample locations, an operation known here as "chroma reconstruction" irrespective of the actual color model.

The location of the chroma samples relative to the luma coordinates is determined by the `xChromaOffset` and `yChromaOffset` members of the VkSamplerYcbcrConversionCreateInfo structure used to create the sampler $Y'C_BC_R$ conversion.

The following diagrams show the relationship between unnormalized ($u$,$v$) coordinates and ($i$,$j$) integer texel positions in the luma component (shown in black, with circles showing integer sample positions) and the texel coordinates of reduced-resolution chroma components, shown as crosses in red.

**NOTE** If the chroma values are reconstructed at the locations of the luma samples by means of interpolation, chroma samples from outside the image bounds are needed; these are determined according to Wrapping Operation. These diagrams represent this by showing the bounds of the "chroma texel" extending beyond the image bounds, and including additional chroma sample positions where required for interpolation. The limits of a sample for `NEAREST` sampling is shown as a grid.

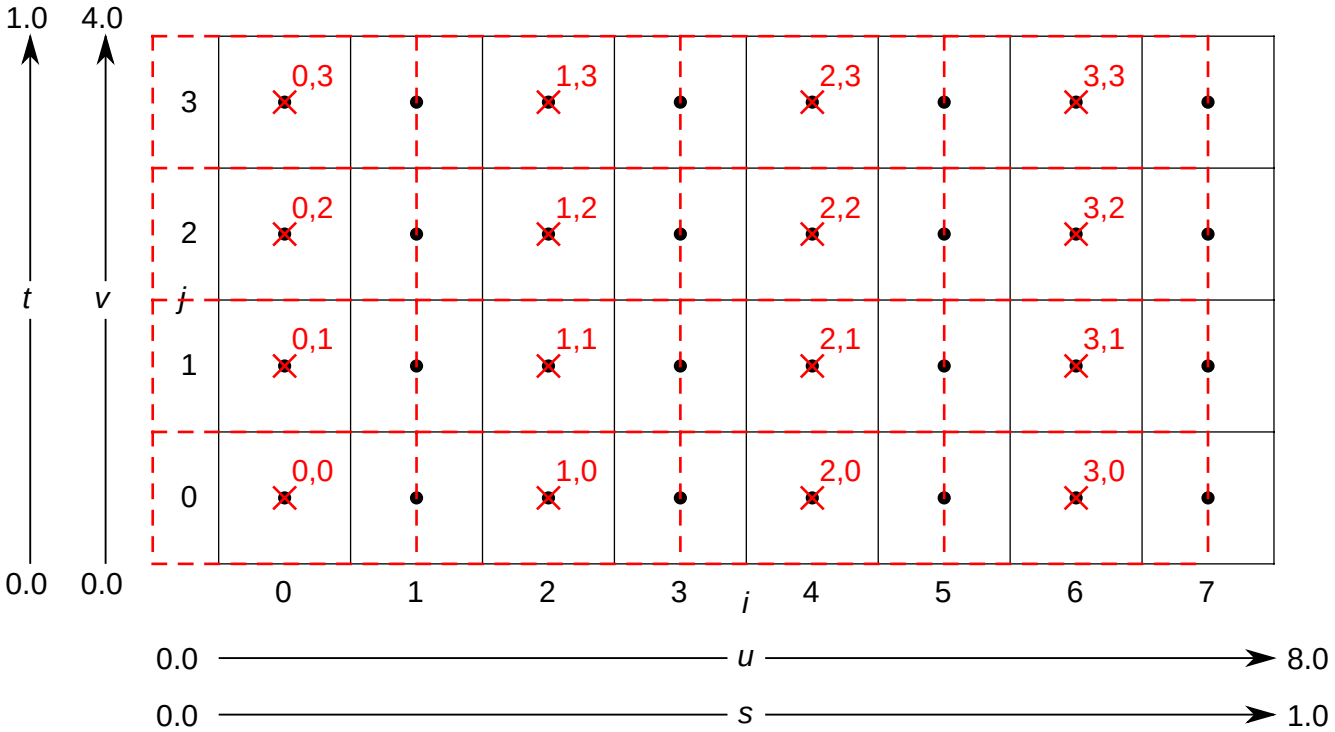



*Figure 6. 422 downsampling, xChromaOffset=COSITED_EVEN*

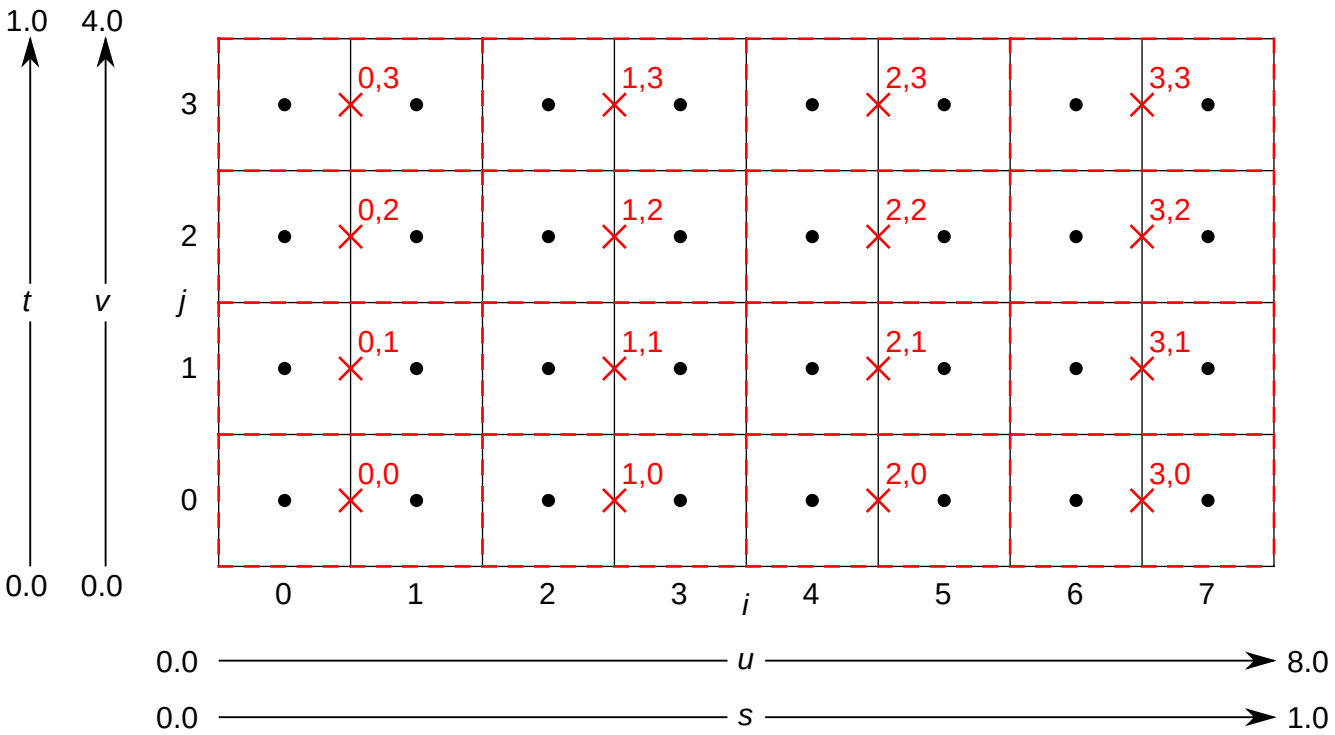



*Figure 7. 422 downsampling, xChromaOffset=MIDPOINT*

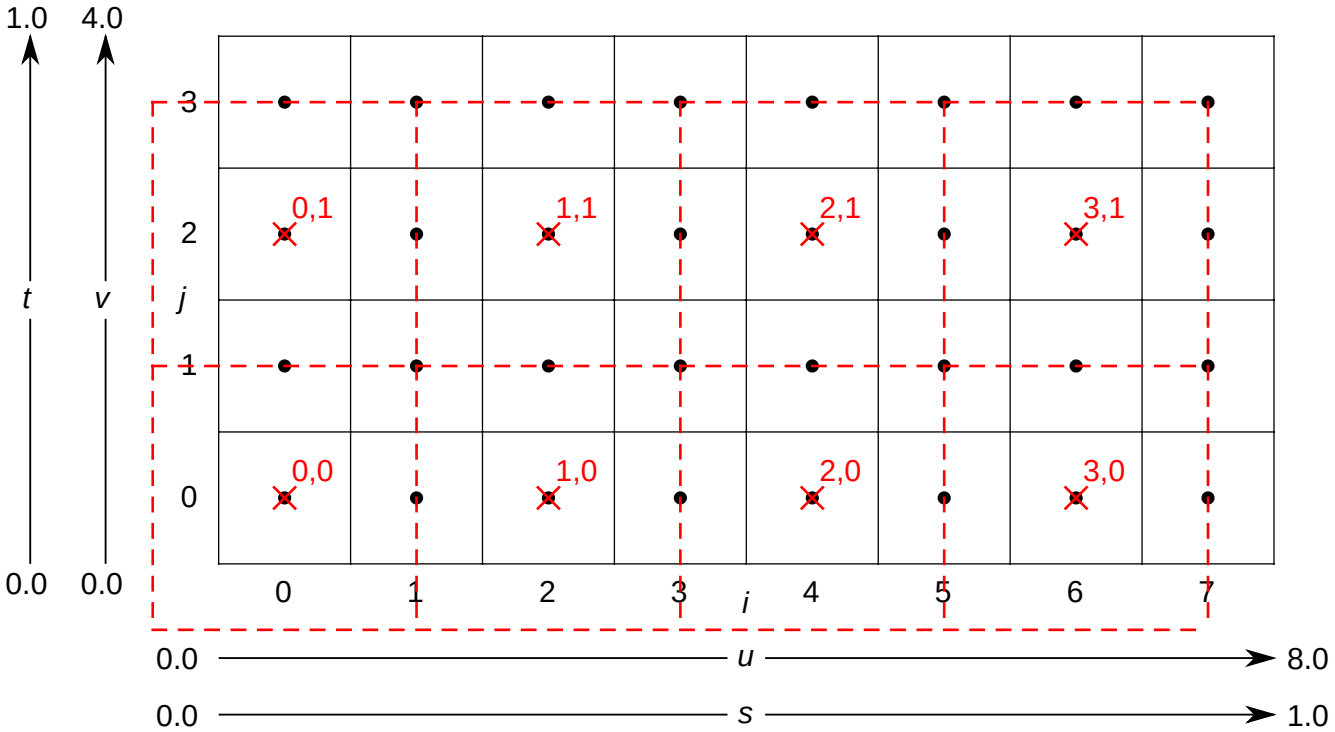



*Figure 8. 420 downsampling, xChromaOffset=COSITED_EVEN, yChromaOffset=COSITED_EVEN*

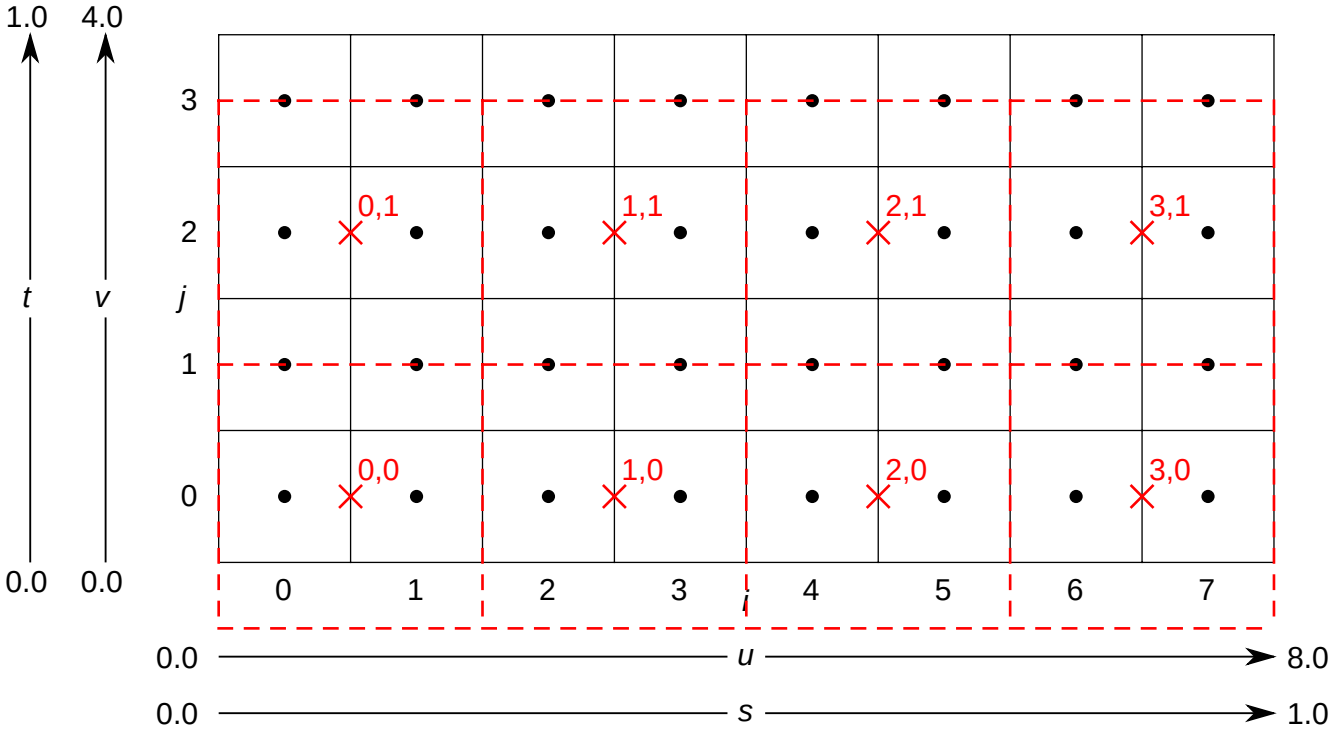



*Figure 9. 420 downsampling, xChromaOffset=MIDPOINT, yChromaOffset=COSITED_EVEN*

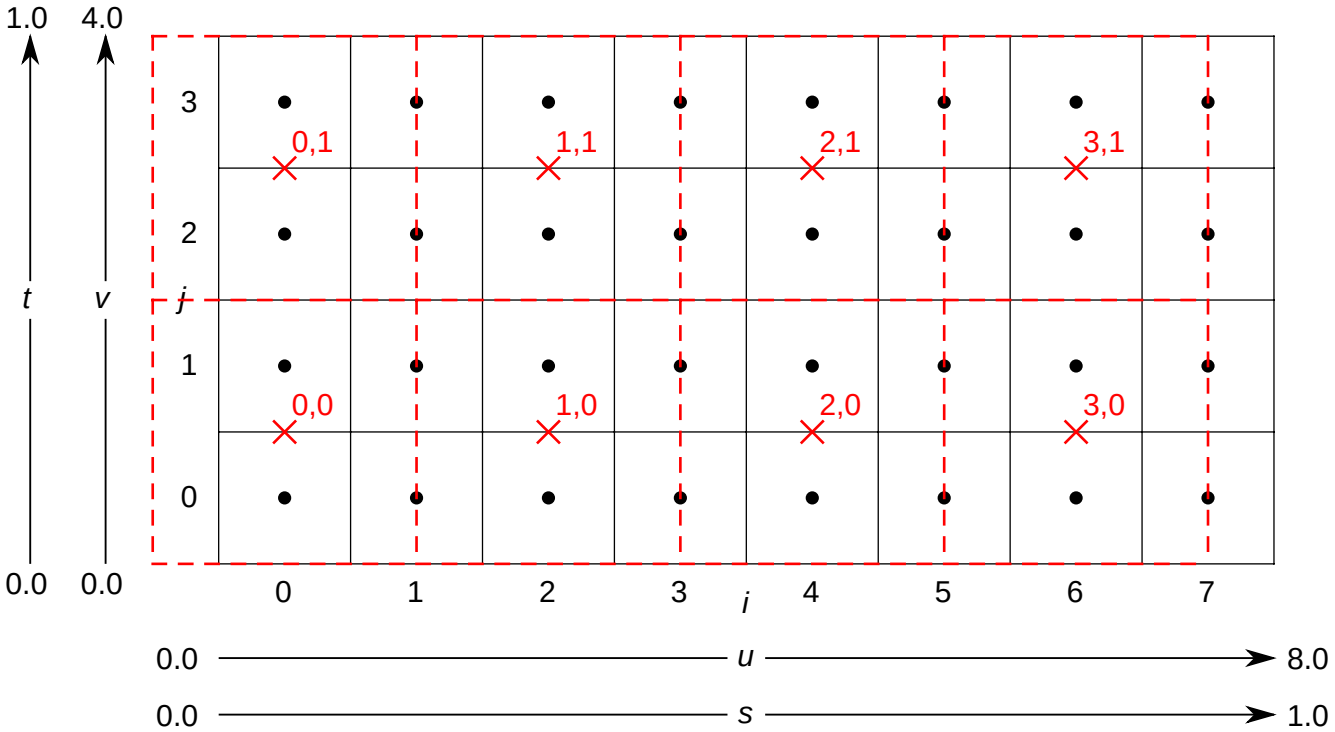



*Figure 10. 420 downsampling, xChromaOffset=COSITED_EVEN, yChromaOffset=MIDPOINT*

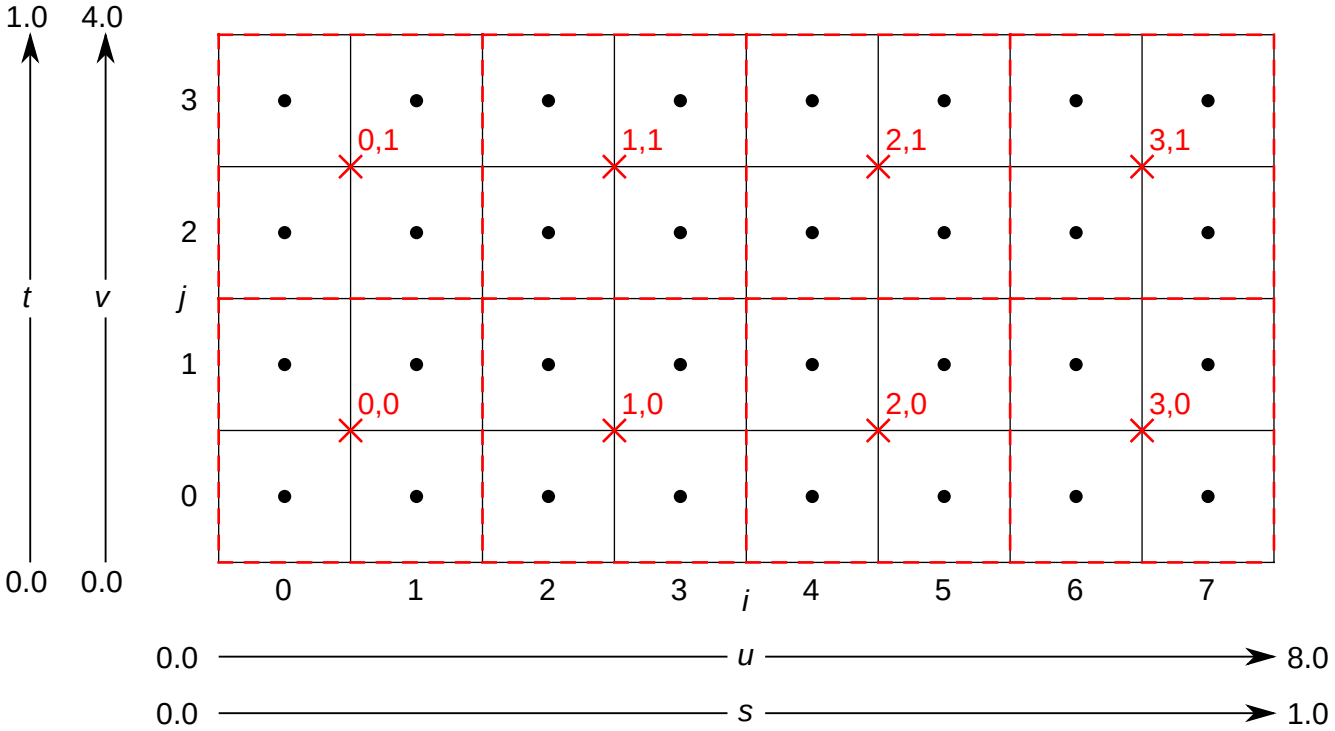



*Figure 11. 420 downsampling, xChromaOffset=MIDPOINT, yChromaOffset=MIDPOINT*

Reconstruction is implemented in one of two ways:

If the format of the image that is to be sampled sets `VK_FORMAT_FEATURE_SAMPLED_IMAGE_YCBCR_CONVERSION_CHROMA_RECONSTRUCTION_EXPLICIT_BIT`, or the VkSamplerYcbcrConversionCreateInfo's `forceExplicitReconstruction` is set to `VK_TRUE`, reconstruction is performed as an explicit step independent of filtering, described in the Explicit Reconstruction section.

If the format of the image that is to be sampled does not set `VK_FORMAT_FEATURE_SAMPLED_IMAGE_YCBCR_CONVERSION_CHROMA_RECONSTRUCTION_EXPLICIT_BIT` and if the

VkSamplerYcbcrConversionCreateInfo's forceExplicitReconstruction is set to VK_FALSE, reconstruction is performed as an implicit part of filtering prior to color model conversion, with no separate post-conversion texel filtering step, as described in the Implicit Reconstruction section.

**Explicit Reconstruction**

- If the chromaFilter member of the VkSamplerYcbcrConversionCreateInfo structure is VK_FILTER_NEAREST:
  - If the format's R and B components are reduced in resolution in just width by a factor of two relative to the G component (i.e. this is a "_422" format), the $\tau_{ijk}[level]$ values accessed by texel filtering are reconstructed as follows:

$$\tau_R{}'(i,\ j) = \tau_R(\lfloor i \times 0.5 \rfloor,\ j)[level]$$
$$\tau_B{}'(i,\ j) = \tau_B(\lfloor i \times 0.5 \rfloor,\ j)[level]$$

  - If the format's R and B components are reduced in resolution in width and height by a factor of two relative to the G component (i.e. this is a "_420" format), the $\tau_{ijk}[level]$ values accessed by texel filtering are reconstructed as follows:

$$\tau_R{}'(i,\ j) = \tau_R(\lfloor i \times 0.5 \rfloor,\ \lfloor j \times 0.5 \rfloor)[level]$$
$$\tau_B{}'(i,\ j) = \tau_B(\lfloor i \times 0.5 \rfloor,\ \lfloor j \times 0.5 \rfloor)[level]$$

**NOTE** xChromaOffset and yChromaOffset have no effect if chromaFilter is VK_FILTER_NEAREST for explicit reconstruction.

- If the chromaFilter member of the VkSamplerYcbcrConversionCreateInfo structure is VK_FILTER_LINEAR:
  - If the format's R and B components are reduced in resolution in just width by a factor of two relative to the G component (i.e. this is a "_422" format):
    - If xChromaOffset is VK_CHROMA_LOCATION_COSITED_EVEN:

$$\tau_{RB}{}'(i,\ j) = \begin{cases} \tau_{RB}(\lfloor i \times 0.5 \rfloor,\ j)[level], & 0.5 \times i = \lfloor 0.5 \times i \rfloor \\ 0.5 \times \tau_{RB}(\lfloor i \times 0.5 \rfloor,\ j)[level] + & \\ 0.5 \times \tau_{RB}(\lfloor i \times 0.5 \rfloor + 1,\ j)[level], & 0.5 \times i \neq \lfloor 0.5 \times i \rfloor \end{cases}$$

    - If xChromaOffset is VK_CHROMA_LOCATION_MIDPOINT:

$$\tau_{RB}{}'(i,\ j) = \begin{cases} 0.25 \times \tau_{RB}(\lfloor i \times 0.5 \rfloor - 1,\ j)[level] + & \\ 0.75 \times \tau_{RB}(\lfloor i \times 0.5 \rfloor,\ j)[level], & 0.5 \times i = \lfloor 0.5 \times i \rfloor \\ 0.75 \times \tau_{RB}(\lfloor i \times 0.5 \rfloor,\ j)[level] + & \\ 0.25 \times \tau_{RB}(\lfloor i \times 0.5 \rfloor + 1,\ j)[level], & 0.5 \times i \neq \lfloor 0.5 \times i \rfloor \end{cases}$$

  - If the format's R and B components are reduced in resolution in width and height by a factor of two relative to the G component (i.e. this is a "_420" format), a similar relationship applies. Due to the number of options, these formulae are expressed more concisely as follows:

$$i_{RB} = \begin{cases} 0.5 \times (i) & \text{xChromaOffset=COSITED_EVEN} \\ 0.5 \times (i - 0.5) & \text{xChromaOffset=MIDPOINT} \end{cases}$$

$$j_{RB} = \begin{cases} 0.5 \times (j) & \text{yChromaOffset=COSITED_EVEN} \\ 0.5 \times (j - 0.5) & \text{yChromaOffset=MIDPOINT} \end{cases}$$

$$i_{floor} = \lfloor i_{RB} \rfloor$$
$$j_{floor} = \lfloor j_{RB} \rfloor$$
$$i_{frac} = i_{RB} - i_{floor}$$
$$j_{frac} = j_{RB} - j_{floor}$$

$$\begin{aligned} \tau_{RB}{}'(i, j) = \; & \tau_{RB}(i_{floor}, j_{floor})[level] && \times (1 - i_{frac}) && \times && (1 - j_{frac}) + \\ & \tau_{RB}(1 + i_{floor}, j_{floor})[level] && \times (i_{frac}) && \times && (1 - j_{frac}) + \\ & \tau_{RB}(i_{floor}, 1 + j_{floor})[level] && \times (1 - i_{frac}) && \times && (j_{frac}) + \\ & \tau_{RB}(1 + i_{floor}, 1 + j_{floor})[level] && \times (i_{frac}) && \times && (j_{frac}) \end{aligned}$$

**NOTE** In the case where the texture itself is bilinearly interpolated as described in Texel Filtering, thus requiring four full-color samples for the filtering operation, and where the reconstruction of these samples uses bilinear interpolation in the chroma components due to `chromaFilter=VK_FILTER_LINEAR`, up to nine chroma samples may be required, depending on the sample location.

**Implicit Reconstruction**

Implicit reconstruction takes place by the samples being interpolated, as required by the filter settings of the sampler, except that `chromaFilter` takes precedence for the chroma samples.

If `chromaFilter` is `VK_FILTER_NEAREST`, an implementation **may** behave as if `xChromaOffset` and `yChromaOffset` were both `VK_CHROMA_LOCATION_MIDPOINT`, irrespective of the values set.

**NOTE** This will not have any visible effect if the locations of the luma samples coincide with the location of the samples used for rasterization.

The sample coordinates are adjusted by the downsample factor of the component (such that, for example, the sample coordinates are divided by two if the component has a downsample factor of two relative to the luma component):

$$u_{RB}{}'(422/420) = \begin{cases} 0.5 \times (u + 0.5), & \text{xChromaOffset=COSITED_EVEN} \\ 0.5 \times u, & \text{xChromaOffset=MIDPOINT} \end{cases}$$

$$v_{RB}{}'(420) = \begin{cases} 0.5 \times (v + 0.5), & \text{yChromaOffset=COSITED_EVEN} \\ 0.5 \times v, & \text{yChromaOffset=MIDPOINT} \end{cases}$$

### 16.3.9. Sampler $Y'C_BC_R$ Conversion

Sampler $Y'C_BC_R$ conversion performs the following operations, which an implementation **may** combine into a single mathematical operation:

- Sampler $Y'C_BC_R$ Range Expansion
- Sampler $Y'C_BC_R$ Model Conversion

### Sampler $Y'C_BC_R$ Range Expansion

Sampler $Y'C_BC_R$ range expansion is applied to color component values after all texel input operations which are not specific to sampler $Y'C_BC_R$ conversion. For example, the input values to this stage have been converted using the normal format conversion rules.

The input values to this stage may have been converted using sRGB to linear conversion if `ycbcrDegamma` is enabled.

Sampler $Y'C_BC_R$ range expansion is not applied if `ycbcrModel` is `VK_SAMPLER_YCBCR_MODEL_CONVERSION_RGB_IDENTITY`. That is, the shader receives the vector $C'_{rgba}$ as output by the Component Swizzle stage without further modification.

For other values of `ycbcrModel`, range expansion is applied to the texel component values output by the Component Swizzle defined by the `components` member of VkSamplerYcbcrConversionCreateInfo. Range expansion applies independently to each component of the image. For the purposes of range expansion and $Y'C_BC_R$ model conversion, the R and B components contain color difference (chroma) values and the G component contains luma. The A component is not modified by sampler $Y'C_BC_R$ range expansion.

The range expansion to be applied is defined by the `ycbcrRange` member of the VkSamplerYcbcrConversionCreateInfo structure:

- If `ycbcrRange` is `VK_SAMPLER_YCBCR_RANGE_ITU_FULL`, the following transformations are applied:

$$
\begin{aligned}
Y' &= C'_{rgba}[G] \\
C_B &= C'_{rgba}[B] - \frac{2^{(n-1)}}{(2^n)-1} \\
C_R &= C'_{rgba}[R] - \frac{2^{(n-1)}}{(2^n)-1}
\end{aligned}
$$

> **NOTE**
>
> These formulae correspond to the “full range” encoding in the “Quantization schemes” chapter of the Khronos Data Format Specification.
>
> Should any future amendments be made to the ITU specifications from which these equations are derived, the formulae used by Vulkan **may** also be updated to maintain parity.

- If `ycbcrRange` is `VK_SAMPLER_YCBCR_RANGE_ITU_NARROW`, the following transformations are applied:

$$Y' = \frac{C'_{rgba}[G] \times (2^n - 1) - 16 \times 2^{n-8}}{219 \times 2^{n-8}}$$
$$C_B = \frac{C'_{rgba}[B] \times (2^n - 1) - 128 \times 2^{n-8}}{224 \times 2^{n-8}}$$
$$C_R = \frac{C'_{rgba}[R] \times (2^n - 1) - 128 \times 2^{n-8}}{224 \times 2^{n-8}}$$

**NOTE** These formulae correspond to the "narrow range" encoding in the "Quantization schemes" chapter of the Khronos Data Format Specification.

- $n$ is the bit-depth of the components in the format.

The precision of the operations performed during range expansion **must** be at least that of the source format.

An implementation **may** clamp the results of these range expansion operations such that Y' falls in the range [0,1], and/or such that $C_B$ and $C_R$ fall in the range [-0.5,0.5].

#### Sampler $Y'C_BC_R$ Model Conversion

The range-expanded values are converted between color models, according to the color model conversion specified in the `ycbcrModel` member:

`VK_SAMPLER_YCBCR_MODEL_CONVERSION_RGB_IDENTITY`

The color components are not modified by the color model conversion since they are assumed already to represent the desired color model in which the shader is operating; $Y'C_BC_R$ range expansion is also ignored.

`VK_SAMPLER_YCBCR_MODEL_CONVERSION_YCBCR_IDENTITY`

The color components are not modified by the color model conversion and are assumed to be treated as though in $Y'C_BC_R$ form both in memory and in the shader; $Y'C_BC_R$ range expansion is applied to the components as for other $Y'C_BC_R$ models, with the vector ($C_R$,Y',$C_B$,A) provided to the shader.

`VK_SAMPLER_YCBCR_MODEL_CONVERSION_YCBCR_709`

The color components are transformed from a $Y'C_BC_R$ representation to an R'G'B' representation as described in the "BT.709 $Y'C_BC_R$ conversion" section of the Khronos Data Format Specification.

`VK_SAMPLER_YCBCR_MODEL_CONVERSION_YCBCR_601`

The color components are transformed from a $Y'C_BC_R$ representation to an R'G'B' representation as described in the "BT.601 $Y'C_BC_R$ conversion" section of the Khronos Data Format Specification.

`VK_SAMPLER_YCBCR_MODEL_CONVERSION_YCBCR_2020`

The color components are transformed from a $Y'C_BC_R$ representation to an R'G'B' representation as described in the "BT.2020 $Y'C_BC_R$ conversion" section of the Khronos Data Format Specification.

In this operation, each output component is dependent on each input component.

An implementation **may** clamp the R'G'B' results of these conversions to the range [0,1].

The precision of the operations performed during model conversion **must** be at least that of the source format.

The alpha component is not modified by these model conversions.

**NOTE**

Sampling operations in a non-linear color space can introduce color and intensity shifts at sharp transition boundaries. To avoid this issue, the technically precise color correction sequence described in the “Introduction to Color Conversions” chapter of the Khronos Data Format Specification may be performed as follows:

- Calculate the unnormalized texel coordinates corresponding to the desired sample position.
- For a `minFilter` or `magFilter` of `VK_FILTER_NEAREST`:
  1. Calculate (*i*,*j*) for the sample location as described under the “nearest filtering” formulae in (u,v,w,a) to (i,j,k,l,n) Transformation and Array Layer Selection
  2. Calculate the normalized texel coordinates corresponding to these integer coordinates.
  3. Sample using sampler $Y'C_BC_R$ conversion at this location.
- For a `minFilter` or `magFilter` of `VK_FILTER_LINEAR`:
  1. Calculate ($i_{[0,1]}$,$j_{[0,1]}$) for the sample location as described under the “linear filtering” formulae in (u,v,w,a) to (i,j,k,l,n) Transformation and Array Layer Selection
  2. Calculate the normalized texel coordinates corresponding to these integer coordinates.
  3. Sample using sampler $Y'C_BC_R$ conversion at each of these locations.
  4. Convert the non-linear A'R'G'B' outputs of the $Y'C_BC_R$ conversions to linear ARGB values as described in the “Transfer Functions” chapter of the Khronos Data Format Specification.
  5. Interpolate the linear ARGB values using the α and β values described in the “linear filtering” section of (u,v,w,a) to (i,j,k,l,n) Transformation and Array Layer Selection and the equations in Texel Filtering.

The additional calculations and, especially, additional number of sampling operations in the `VK_FILTER_LINEAR` case can be expected to have a performance impact compared with using the outputs directly. Since the variations from “correct” results are subtle for most content, the application author should determine whether a more costly implementation is strictly necessary.

If `chromaFilter`, and `minFilter` or `magFilter` are both `VK_FILTER_NEAREST`, these operations are redundant and sampling using sampler $Y'C_BC_R$ conversion at the desired sample coordinates will produce the “correct” results without further processing.

# 16.4. Texel Output Operations

*Texel output instructions* are SPIR-V image instructions that write to an image. *Texel output operations* are a set of steps that are performed on state, coordinates, and texel values while processing a texel output instruction, and which are common to some or all texel output instructions. They include the following steps, which are performed in the listed order:

- Validation operations
  - Format validation
  - Type validation
  - Coordinate validation
  - Sparse validation
- Texel output format conversion

## 16.4.1. Texel Output Validation Operations

*Texel output validation operations* inspect instruction/image state or coordinates, and in certain circumstances cause the write to have no effect. There are a series of validations that the texel undergoes.

#### Texel Format Validation

If the image format of the `OpTypeImage` is not compatible with the `VkImageView`'s `format`, the write causes the contents of the image's memory to become undefined.

#### Texel Type Validation

If the `Sampled Type` of the `OpTypeImage` does not match the SPIR-V Type, the write causes the value of the texel to become undefined. For integer types, if the signedness of the access does not match the signedness of the accessed resource, the write causes the value of the texel to become undefined.

## 16.4.2. Integer Texel Coordinate Validation

The integer texel coordinates are validated according to the same rules as for texel input coordinate validation.

If the texel fails integer texel coordinate validation, then the write has no effect.

## 16.4.3. Sparse Texel Operation

If the texel attempts to write to an unbound region of a sparse image, the texel is a sparse unbound texel. In such a case, if the VkPhysicalDeviceSparseProperties::`residencyNonResidentStrict` property is `VK_TRUE`, the sparse unbound texel write has no effect. If `residencyNonResidentStrict` is `VK_FALSE`, the write **may** have a side effect that becomes visible to other accesses to unbound texels in any resource, but will not be visible to any device memory allocated by the application.

### 16.4.4. Texel Output Format Conversion

If the image format is sRGB, a linear to sRGB conversion is applied to the R, G, and B components as described in the “sRGB EOTF” section of the Khronos Data Format Specification. The A component, if present, is unchanged.

Texels then undergo a format conversion from the floating-point, signed, or unsigned integer type of the texel data to the VkFormat of the image view. If the number of components in the texel data is larger than the number of components in the format, additional components are discarded.

Each component is converted based on its type and size (as defined in the Format Definition section for each VkFormat). Floating-point outputs are converted as described in Floating-Point Format Conversions and Fixed-Point Data Conversion. Integer outputs are converted such that their value is preserved. The converted value of any integer that cannot be represented in the target format is undefined.

If the `VkImageView` format has an X component in its format description, undefined values are written to those bits.

If the underlying `VkImage` format has an X component in its format description, undefined values are also written to those bits, even if result format conversion produces a valid value for those bits because the `VkImageView` format is different.

## 16.5. Normalized Texel Coordinate Operations

If the image sampler instruction provides normalized texel coordinates, some of the following operations are performed.

### 16.5.1. Projection Operation

For `Proj` image operations, the normalized texel coordinates (s,t,r,q,a) and (if present) the $D_{ref}$ coordinate are transformed as follows:

$$
\begin{aligned}
s &= \frac{s}{q}, && \text{for 1D, 2D, or 3D image} \\
t &= \frac{t}{q}, && \text{for 2D or 3D image} \\
r &= \frac{r}{q}, && \text{for 3D image} \\
D_{ref} &= \frac{D_{ref}}{q}, && \text{if provided}
\end{aligned}
$$

### 16.5.2. Derivative Image Operations

Derivatives are used for LOD selection. These derivatives are either implicit (in an `ImplicitLod` image instruction in a mesh, task, compute, or fragment shader) or explicit (provided explicitly by shader to the image instruction in any shader).

For implicit derivatives image instructions, the derivatives of texel coordinates are calculated in the same manner as derivative operations. That is:

$$\begin{aligned} \partial s / \partial x &= dPdx(s), & \partial s / \partial y &= dPdy(s), & &\text{for 1D, 2D, Cube, or 3D image} \\ \partial t / \partial x &= dPdx(t), & \partial t / \partial y &= dPdy(t), & &\text{for 2D, Cube, or 3D image} \\ \partial r / \partial x &= dPdx(r), & \partial r / \partial y &= dPdy(r), & &\text{for Cube or 3D image} \end{aligned}$$

Partial derivatives not defined above for certain image dimensionalities are set to zero.

For explicit LOD image instructions, if the **optional** SPIR-V operand `Grad` is provided, then the operand values are used for the derivatives. The number of components present in each derivative for a given image dimensionality matches the number of partial derivatives computed above.

If the **optional** SPIR-V operand `Lod` is provided, then derivatives are set to zero, the cube map derivative transformation is skipped, and the scale factor operation is skipped. Instead, the floating-point scalar coordinate is directly assigned to $\lambda_{base}$ as described in LOD Operation.

If the image or sampler object used by an implicit derivative image instruction is not uniform across the quad and `quadDivergentImplicitLod` is not supported, then the derivative and LOD values are undefined. Implicit derivatives are well-defined when the image and sampler and control flow are uniform across the quad, even if they diverge between different quads.

If `quadDivergentImplicitLod` is supported, then derivatives and implicit LOD values are well-defined even if the image or sampler object are not uniform within a quad. The derivatives are computed as specified above, and the implicit LOD calculation proceeds for each shader invocation using its respective image and sampler object.

### 16.5.3. Cube Map Face Selection and Transformations

For cube map image instructions, the (s,t,r) coordinates are treated as a direction vector $(r_x,r_y,r_z)$. The direction vector is used to select a cube map face. The direction vector is transformed to a per-face texel coordinate system $(s_{face},t_{face})$, The direction vector is also used to transform the derivatives to per-face derivatives.

### 16.5.4. Cube Map Face Selection

The direction vector selects one of the cube map's faces based on the largest magnitude coordinate direction (the major axis direction). Since two or more coordinates **can** have identical magnitude, the implementation **must** have rules to disambiguate this situation.

The rules **should** have as the first rule that $r_z$ wins over $r_y$ and $r_x$, and the second rule that $r_y$ wins over $r_x$. An implementation **may** choose other rules, but the rules **must** be deterministic and depend only on $(r_x,r_y,r_z)$.

The layer number (corresponding to a cube map face), the coordinate selections for $s_c$, $t_c$, $r_c$, and the selection of derivatives, are determined by the major axis direction as specified in the following two tables.

*Table 27. Cube map face and coordinate selection*

| Major Axis Direction | Layer Number | Cube Map Face | $s_c$ | $t_c$ | $r_c$ |
|---|---|---|---|---|---|
| $+r_x$ | 0 | Positive X | $-r_z$ | $-r_y$ | $r_x$ |
| $-r_x$ | 1 | Negative X | $+r_z$ | $-r_y$ | $r_x$ |
| $+r_y$ | 2 | Positive Y | $+r_x$ | $+r_z$ | $r_y$ |
| $-r_y$ | 3 | Negative Y | $+r_x$ | $-r_z$ | $r_y$ |
| $+r_z$ | 4 | Positive Z | $+r_x$ | $-r_y$ | $r_z$ |
| $-r_z$ | 5 | Negative Z | $-r_x$ | $-r_y$ | $r_z$ |

*Table 28. Cube map derivative selection*

| Major Axis Direction | $\partial s_c / \partial x$ | $\partial s_c / \partial y$ | $\partial t_c / \partial x$ | $\partial t_c / \partial y$ | $\partial r_c / \partial x$ | $\partial r_c / \partial y$ |
|---|---|---|---|---|---|---|
| $+r_x$ | $-\partial r_z / \partial x$ | $-\partial r_z / \partial y$ | $-\partial r_y / \partial x$ | $-\partial r_y / \partial y$ | $+\partial r_x / \partial x$ | $+\partial r_x / \partial y$ |
| $-r_x$ | $+\partial r_z / \partial x$ | $+\partial r_z / \partial y$ | $-\partial r_y / \partial x$ | $-\partial r_y / \partial y$ | $-\partial r_x / \partial x$ | $-\partial r_x / \partial y$ |
| $+r_y$ | $+\partial r_x / \partial x$ | $+\partial r_x / \partial y$ | $+\partial r_z / \partial x$ | $+\partial r_z / \partial y$ | $+\partial r_y / \partial x$ | $+\partial r_y / \partial y$ |
| $-r_y$ | $+\partial r_x / \partial x$ | $+\partial r_x / \partial y$ | $-\partial r_z / \partial x$ | $-\partial r_z / \partial y$ | $-\partial r_y / \partial x$ | $-\partial r_y / \partial y$ |
| $+r_z$ | $+\partial r_x / \partial x$ | $+\partial r_x / \partial y$ | $-\partial r_y / \partial x$ | $-\partial r_y / \partial y$ | $+\partial r_z / \partial x$ | $+\partial r_z / \partial y$ |
| $-r_z$ | $-\partial r_x / \partial x$ | $-\partial r_x / \partial y$ | $-\partial r_y / \partial x$ | $-\partial r_y / \partial y$ | $-\partial r_z / \partial x$ | $-\partial r_z / \partial y$ |

### 16.5.5. Cube Map Coordinate Transformation

$$s_{face} = \frac{1}{2} \times \frac{s_c}{|r_c|} + \frac{1}{2}$$
$$t_{face} = \frac{1}{2} \times \frac{t_c}{|r_c|} + \frac{1}{2}$$

### 16.5.6. Cube Map Derivative Transformation

$$\frac{\partial s_{face}}{\partial x} = \frac{\partial}{\partial x}\left(\frac{1}{2} \times \frac{s_c}{|r_c|} + \frac{1}{2}\right)$$
$$\frac{\partial s_{face}}{\partial x} = \frac{1}{2} \times \frac{\partial}{\partial x}\left(\frac{s_c}{|r_c|}\right)$$
$$\frac{\partial s_{face}}{\partial x} = \frac{1}{2} \times \left(\frac{|r_c| \times \partial s_c / \partial x - s_c \times \partial r_c / \partial x}{(r_c)^2}\right)$$

$$\frac{\partial s_{face}}{\partial y} = \frac{1}{2} \times \left( \frac{|r_c| \times \partial s_c / \partial y - s_c \times \partial r_c / \partial y}{(r_c)^2} \right)$$
$$\frac{\partial t_{face}}{\partial x} = \frac{1}{2} \times \left( \frac{|r_c| \times \partial t_c / \partial x - t_c \times \partial r_c / \partial x}{(r_c)^2} \right)$$
$$\frac{\partial t_{face}}{\partial y} = \frac{1}{2} \times \left( \frac{|r_c| \times \partial t_c / \partial y - t_c \times \partial r_c / \partial y}{(r_c)^2} \right)$$

### 16.5.7. Scale Factor Operation, LOD Operation and Image Level(s) Selection

LOD selection **can** be either explicit (provided explicitly by the image instruction) or implicit (determined from a scale factor calculated from the derivatives). The LOD **must** be computed with `mipmapPrecisionBits` of accuracy.

#### Scale Factor Operation

The magnitude of the derivatives are calculated by:

$m_{ux} = |\partial s/\partial x| \times w_{base}$

$m_{vx} = |\partial t/\partial x| \times h_{base}$

$m_{wx} = |\partial r/\partial x| \times d_{base}$

$m_{uy} = |\partial s/\partial y| \times w_{base}$

$m_{vy} = |\partial t/\partial y| \times h_{base}$

$m_{wy} = |\partial r/\partial y| \times d_{base}$

where:

$\partial t/\partial x = \partial t/\partial y = 0$ (for 1D images)

$\partial r/\partial x = \partial r/\partial y = 0$ (for 1D, 2D or Cube images)

and:

$w_{base}$ = image.w

$h_{base}$ = image.h

$d_{base}$ = image.d

(for the `baseMipLevel`, from the image descriptor).

For corner-sampled images, the $w_{base}$, $h_{base}$, and $d_{base}$ are instead:

$w_{base}$ = image.w - 1

$h_{base}$ = image.h - 1

$d_{base}$ = image.d - 1

A point sampled in screen space has an elliptical footprint in texture space. The minimum and maximum scale factors ($\rho_{min}$, $\rho_{max}$) **should** be the minor and major axes of this ellipse.

The *scale factors* $\rho_x$ and $\rho_y$, calculated from the magnitude of the derivatives in x and y, are used to compute the minimum and maximum scale factors.

$\rho_x$ and $\rho_y$ **may** be approximated with functions $f_x$ and $f_y$, subject to the following constraints:

$$f_x \text{is continuous and monotonically increasing in each of} m_{ux},\ m_{vx},\ \text{and} m_{wx}$$
$$f_y \text{is continuous and monotonically increasing in each of} m_{uy},\ m_{vy},\ \text{and} m_{wy}$$

$$\max(|m_{ux}|,\ |m_{vx}|,\ |m_{wx}|) \le f_x \le \sqrt{2}(|m_{ux}| + |m_{vx}| + |m_{wx}|)$$
$$\max(|m_{uy}|,\ |m_{vy}|,\ |m_{wy}|) \le f_y \le \sqrt{2}(|m_{uy}| + |m_{vy}| + |m_{wy}|)$$

The minimum and maximum scale factors ($\rho_{min}$,$\rho_{max}$) are determined by:

$\rho_{max} = \max(\rho_x, \rho_y)$

$\rho_{min} = \min(\rho_x, \rho_y)$

The ratio of anisotropy is determined by:

$\eta = \min(\rho_{max}/\rho_{min}, \max_{Aniso})$

where:

$$sampler.max_{Aniso} = \texttt{maxAnisotropy} \text{ (from sampler descriptor)}$$

$$limits.max_{Aniso} = \texttt{maxSamplerAnisotropy} \text{ (from physical device limits)}$$

$$max_{Aniso} = \min(sampler.max_{Aniso}, limits.max_{Aniso})$$

If $\rho_{max} = \rho_{min} = 0$, then all the partial derivatives are zero, the fragment's footprint in texel space is a point, and $\eta$ **should** be treated as 1. If $\rho_{max} \neq 0$ and $\rho_{min} = 0$ then all partial derivatives along one axis are zero, the fragment's footprint in texel space is a line segment, and $\eta$ **should** be treated as $max_{Aniso}$. However, anytime the footprint is small in texel space the implementation **may** use a smaller value of $\eta$, even when $\rho_{min}$ is zero or close to zero. If either VkPhysicalDeviceFeatures::`samplerAnisotropy` or VkSamplerCreateInfo::`anisotropyEnable` are `VK_FALSE`, $max_{Aniso}$ is set to 1.

If $\eta = 1$, sampling is isotropic. If $\eta > 1$, sampling is anisotropic.

The sampling rate (N) is derived as:

$$N = \lceil \eta \rceil$$

An implementation **may** round N up to the nearest supported sampling rate. An implementation **may** use the value of N as an approximation of $\eta$.

### LOD Operation

The LOD parameter $\lambda$ is computed as follows:

$$\lambda_{base}(x, y) = \begin{cases} shaderOp.Lod & \text{(from optional SPIR-V operand)} \\ \log_2\left(\frac{\rho_{max}}{\eta}\right) & \text{otherwise} \end{cases}$$

$$\lambda'(x, y) = \lambda_{base} + \text{clamp}(sampler.bias + shaderOp.bias, -maxSamplerLodBias, maxSamplerLodBias)$$

$$\lambda = \begin{cases} lod_{max}, & \lambda' > lod_{max} \\ \lambda', & lod_{min} \leq \lambda' \leq lod_{max} \\ lod_{min}, & \lambda' < lod_{min} \\ undefined, & lod_{min} > lod_{max} \end{cases}$$

where:

$$sampler.bias = mipLodBias \qquad \text{(from sampler descriptor)}$$

$$shaderOp.bias = \begin{cases} Bias & \text{(from optional SPIR-V operand)} \\ 0 & \text{otherwise} \end{cases}$$

$$sampler.lod_{min} = minLod \qquad \text{(from sampler descriptor)}$$

$$shaderOp.lod_{min} = \begin{cases} MinLod & \text{(from optional SPIR-V operand)} \\ 0 & \text{otherwise} \end{cases}$$

$$lod_{min} = \max(sampler.lod_{min}, shaderOp.lod_{min})$$

$$lod_{max} = maxLod \qquad \text{(from sampler descriptor)}$$

and maxSamplerLodBias is the value of the VkPhysicalDeviceLimits feature `maxSamplerLodBias`.

**Image Level(s) Selection**

The image level(s) d, $d_{hi}$, and $d_{lo}$ which texels are read from are determined by an image-level parameter $d_l$, which is computed based on the LOD parameter, as follows:

$$d_l = \begin{cases} nearest(d'), & \text{mipmapMode is VK_SAMPLER_MIPMAP_MODE_NEAREST} \\ d', & \text{otherwise} \end{cases}$$

where:

$$d' = max(level_{base} + \text{clamp}(\lambda, 0, q), minLod_{imageView})$$

$$nearest(d') = \begin{cases} \lceil d' + 0.5 \rceil - 1, & \text{preferred} \\ \lfloor d' + 0.5 \rfloor, & \text{alternative} \end{cases}$$

and:

$$minLod_{imageView} = \begin{cases} minLodFloat_{imageView}, & \text{preferred} \\ minLodInteger_{imageView}, & \text{alternative} \end{cases}$$
$$level_{base} = baseMipLevel$$
$$q = levelCount - 1$$

`baseMipLevel` and `levelCount` are taken from the `subresourceRange` of the image view.

$minLod_{imageView}$ **must** be less or equal to $level_{base}$ + q.

If the sampler's `mipmapMode` is `VK_SAMPLER_MIPMAP_MODE_NEAREST`, then the level selected is d = $d_l$.

If the sampler's `mipmapMode` is `VK_SAMPLER_MIPMAP_MODE_LINEAR`, two neighboring levels are selected:

$$d_{hi} = \lfloor d_l \rfloor$$
$$d_{lo} = min(d_{hi} + 1, level_{base} + q)$$
$$\delta = d_l - d_{hi}$$

δ is the fractional value, quantized to the number of mipmap precision bits, used for linear filtering between levels.

## 16.5.8. (s,t,r,q,a) to (u,v,w,a) Transformation

The normalized texel coordinates are scaled by the image level dimensions and the array layer is selected.

This transformation is performed once for each level used in filtering (either d, or $d_{hi}$ and $d_{lo}$).

$$u(x,\ y) = s(x,\ y) \times width_{scale} + \Delta_i$$

$$v(x,\ y) = \begin{cases} 0 & \text{for 1D images} \\ t(x,\ y) \times height_{scale} + \Delta_j & \text{otherwise} \end{cases}$$

$$w(x,\ y) = \begin{cases} 0 & \text{for 2D or Cube images} \\ r(x,\ y) \times depth_{scale} + \Delta_k & \text{otherwise} \end{cases}$$

$$a(x,\ y) = \begin{cases} a(x,\ y) & \text{for array images} \\ 0 & \text{otherwise} \end{cases}$$

where:

$width_{scale} = width_{level}$

$height_{scale} = height_{level}$

$depth_{scale} = depth_{level}$

for conventional images, and:

$width_{scale} = width_{level} - 1$

$height_{scale} = height_{level} - 1$

$depth_{scale} = depth_{level} - 1$

for corner-sampled images.

and where ($\Delta_i$, $\Delta_j$, $\Delta_k$) are taken from the image instruction if it includes a `ConstOffset` or `Offset` operand, otherwise they are taken to be zero.

Operations then proceed to Unnormalized Texel Coordinate Operations.

# 16.6. Unnormalized Texel Coordinate Operations

## 16.6.1. (u,v,w,a) to (i,j,k,l,n) Transformation and Array Layer Selection

The unnormalized texel coordinates are transformed to integer texel coordinates relative to the selected mipmap level.

The layer index l is computed as:

l = clamp(RNE(a), 0, `layerCount` - 1) + `baseArrayLayer`

where `layerCount` is the number of layers in the image subresource range of the image view, `baseArrayLayer` is the first layer from the subresource range, and where:

$$\mathrm{RNE}(a) = \begin{cases} \mathrm{roundTiesToEven}(a) & \text{preferred, from IEEE Std 754-2008 Floating-Point Arithmetic} \\ \lfloor a + 0.5 \rfloor & \text{alternative} \end{cases}$$

The sample index n is assigned the value 0.

Nearest filtering (`VK_FILTER_NEAREST`) computes the integer texel coordinates that the unnormalized coordinates lie within:

$$\begin{aligned} i &= \lfloor u + shift \rfloor \\ j &= \lfloor v + shift \rfloor \\ k &= \lfloor w + shift \rfloor \end{aligned}$$

where:

shift = 0.0

for conventional images, and:

shift = 0.5

for corner-sampled images.

Linear filtering (`VK_FILTER_LINEAR`) computes a set of neighboring coordinates which bound the unnormalized coordinates. The integer texel coordinates are combinations of $i_0$ or $i_1$, $j_0$ or $j_1$, $k_0$ or $k_1$, as well as weights α, β, and γ.

$$\begin{aligned} i_0 &= \lfloor u - shift \rfloor \\ i_1 &= i_0 + 1 \\ j_0 &= \lfloor v - shift \rfloor \\ j_1 &= j_0 + 1 \\ k_0 &= \lfloor w - shift \rfloor \\ k_1 &= k_0 + 1 \end{aligned}$$

$$\alpha = \mathrm{frac}(u - shift)$$

$$\beta = \mathrm{frac}(v - shift)$$

$$\gamma = \mathrm{frac}(w - shift)$$

where:

shift = 0.5

for conventional images, and:

shift = 0.0

for corner-sampled images, and where:

$$\mathrm{frac}\,(x) = x - \lfloor x \rfloor$$

where the number of fraction bits retained is specified by `VkPhysicalDeviceLimits::subTexelPrecisionBits`.

Cubic filtering (`VK_FILTER_CUBIC_EXT`) computes a set of neighboring coordinates which bound the unnormalized coordinates. The integer texel coordinates are combinations of $i_0$, $i_1$, $i_2$ or $i_3$, $j_0$, $j_1$, $j_2$ or $j_3$, $k_0$, $k_1$, $k_2$ or $k_3$, as well as weights α, β, and γ.

$$i_0 = \lfloor u - \frac{3}{2} \rfloor \qquad i_1 = i_0 + 1 \qquad i_2 = i_1 + 1 \qquad i_3 = i_2 + 1$$

$$j_0 = \lfloor v - \frac{3}{2} \rfloor \qquad j_1 = j_0 + 1 \qquad j_2 = j_1 + 1 \qquad j_3 = j_2 + 1$$

$$k_0 = \lfloor w - \frac{3}{2} \rfloor \qquad k_1 = k_0 + 1 \qquad k_2 = k_1 + 1 \qquad k_3 = k_2 + 1$$

$$\alpha = \mathrm{frac}\left(u - \frac{1}{2}\right)$$

$$\beta = \mathrm{frac}\left(v - \frac{1}{2}\right)$$

$$\gamma = \mathrm{frac}\left(w - \frac{1}{2}\right)$$

where:

$$\mathrm{frac}\,(x) = x - \lfloor x \rfloor$$

where the number of fraction bits retained is specified by `VkPhysicalDeviceLimits::subTexelPrecisionBits`.

## 16.7. Integer Texel Coordinate Operations

Integer texel coordinate operations **may** supply a LOD which texels are to be read from or written to using the optional SPIR-V operand `Lod`. If the `Lod` is provided then it **must** be an integer.

The image level selected is:

$$d = level_{base} + \begin{cases} Lod & \text{(from optional SPIR-V operand)} \\ 0 & \text{otherwise} \end{cases}$$

If d does not lie in the range [`baseMipLevel`, `baseMipLevel` + `levelCount`) or d is less than $\text{minLodInteger}_{\text{imageView}}$, then any values fetched are zero if the `robustImageAccess2` feature is enabled, otherwise are undefined, and any writes (if supported) are discarded.

# 16.8. Image Sample Operations

## 16.8.1. Wrapping Operation

If the used sampler was created without `VK_SAMPLER_CREATE_NON_SEAMLESS_CUBE_MAP_BIT_EXT`, `Cube` images ignore the wrap modes specified in the sampler. Instead, if `VK_FILTER_NEAREST` is used within a mip level then `VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_EDGE` is used, and if `VK_FILTER_LINEAR` is used within a mip level then sampling at the edges is performed as described earlier in the Cube map edge handling section.

The first integer texel coordinate i is transformed based on the `addressModeU` parameter of the sampler.

$$i = \begin{cases} i \bmod size & \text{for repeat} \\ (size - 1) - \text{mirror}\left((i \bmod (2 \times size)) - size\right) & \text{for mirrored repeat} \\ \text{clamp}\left(i, 0, size - 1\right) & \text{for clamp to edge} \\ \text{clamp}\left(i, -1, size\right) & \text{for clamp to border} \\ \text{clamp}\left(\text{mirror}\left(i\right), 0, size - 1\right) & \text{for mirror clamp to edge} \end{cases}$$

where:

$$\text{mirror}\left(n\right) = \begin{cases} n & \text{for } n \geq 0 \\ -(1 + n) & \text{otherwise} \end{cases}$$

j (for 2D and Cube image) and k (for 3D image) are similarly transformed based on the `addressModeV` and `addressModeW` parameters of the sampler, respectively.

## 16.8.2. Texel Gathering

SPIR-V instructions with `Gather` in the name return a vector derived from 4 texels in the base level of the image view. The rules for the `VK_FILTER_LINEAR` minification filter are applied to identify the four selected texels. Each texel is then converted to an RGBA value according to conversion to RGBA and then swizzled. A four-component vector is then assembled by taking the component indicated by the `Component` value in the instruction from the swizzled color value of the four texels. If the operation does not use the `ConstOffsets` image operand then the four texels form the 2 × 2 rectangle used for texture filtering:

$$\tau[R] = \tau_{i0j1}[level_{base}][comp]$$
$$\tau[G] = \tau_{i1j1}[level_{base}][comp]$$
$$\tau[B] = \tau_{i1j0}[level_{base}][comp]$$
$$\tau[A] = \tau_{i0j0}[level_{base}][comp]$$

If the operation does use the `ConstOffsets` image operand then the offsets allow a custom filter to be defined:

$$\tau[R] = \tau_{i0j0+\Delta_0}[level_{base}][comp]$$
$$\tau[G] = \tau_{i0j0+\Delta_1}[level_{base}][comp]$$
$$\tau[B] = \tau_{i0j0+\Delta_2}[level_{base}][comp]$$
$$\tau[A] = \tau_{i0j0+\Delta_3}[level_{base}][comp]$$

where:

$$\tau[level_{base}][comp] = \begin{cases} \tau[level_{base}][R], & \text{for } comp = 0 \\ \tau[level_{base}][G], & \text{for } comp = 1 \\ \tau[level_{base}][B], & \text{for } comp = 2 \\ \tau[level_{base}][A], & \text{for } comp = 3 \end{cases}$$
$$comp \text{ from SPIR-V operand Component}$$

`OpImage*Gather` **must** not be used on a sampled image with sampler $Y'C_BC_R$ conversion enabled.

If $level_{base} < minLodInteger_{imageView}$, then any values fetched are zero if `robustImageAccess2` is enabled. Otherwise values are undefined.

### 16.8.3. Texel Filtering

Texel filtering is first performed for each level (either d or $d_{hi}$ and $d_{lo}$).

If λ is less than or equal to zero, the texture is said to be *magnified*, and the filter mode within a mip level is selected by the `magFilter` in the sampler. If λ is greater than zero, the texture is said to be *minified*, and the filter mode within a mip level is selected by the `minFilter` in the sampler.

#### Texel Nearest Filtering

Within a mip level, `VK_FILTER_NEAREST` filtering selects a single value using the (i, j, k) texel coordinates, with all texels taken from layer l.

$$\tau[level] = \begin{cases} \tau_{ijk}[level], & \text{for 3D image} \\ \tau_{ij}[level], & \text{for 2D or Cube image} \\ \tau_{i}[level], & \text{for 1D image} \end{cases}$$

#### Texel Linear Filtering

Within a mip level, `VK_FILTER_LINEAR` filtering combines 8 (for 3D), 4 (for 2D or Cube), or 2 (for 1D)

texel values, together with their linear weights. The linear weights are derived from the fractions computed earlier:

$$
\begin{aligned}
w_{i_0} &= (1-\alpha) \\
w_{i_1} &= (\alpha) \\
w_{j_0} &= (1-\beta) \\
w_{j_1} &= (\beta) \\
w_{k_0} &= (1-\gamma) \\
w_{k_1} &= (\gamma)
\end{aligned}
$$

The values of multiple texels, together with their weights, are combined to produce a filtered value.

The VkSamplerReductionModeCreateInfo::`reductionMode` **can** control the process by which multiple texels, together with their weights, are combined to produce a filtered texture value.

When the `reductionMode` is set (explicitly or implicitly) to `VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE`, a weighted average is computed:

$$
\begin{aligned}
\tau_{3D} &= \sum_{k=k_0}^{k_1} \sum_{j=j_0}^{j_1} \sum_{i=i_0}^{i_1} (w_i)(w_j)(w_k)\tau_{ijk} \\
\tau_{2D} &= \sum_{j=j_0}^{j_1} \sum_{i=i_0}^{i_1} (w_i)(w_j)\tau_{ij} \\
\tau_{1D} &= \sum_{i=i_0}^{i_1} (w_i)\tau_i
\end{aligned}
$$

However, if the reduction mode is `VK_SAMPLER_REDUCTION_MODE_MIN` or `VK_SAMPLER_REDUCTION_MODE_MAX`, the process operates on the above set of multiple texels, together with their weights, computing a component-wise minimum or maximum, respectively, of the components of the set of texels with non-zero weights.

#### Texel Cubic Filtering

Within a mip level, `VK_FILTER_CUBIC_EXT`, filtering computes a weighted average of 64 (for 3D), 16 (for 2D), or 4 (for 1D) texel values, together with their Catmull-Rom, Zero Tangent Cardinal, B-Spline, or Mitchell-Netravali weights as specified by VkSamplerCubicWeightsCreateInfoQCOM.

Catmull-Rom weights specified by `VK_CUBIC_FILTER_WEIGHTS_CATMULL_ROM_QCOM` are derived from the fractions computed earlier.

$$\begin{bmatrix} w_{i_0} & w_{i_1} & w_{i_2} & w_{i_3} \end{bmatrix} = \frac{1}{2}\begin{bmatrix} 1 & \alpha & \alpha^2 & \alpha^3 \end{bmatrix}\begin{bmatrix} 0 & 2 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 2 & -5 & 4 & -1 \\ -1 & 3 & -3 & 1 \end{bmatrix}$$

$$\begin{bmatrix} w_{j_0} & w_{j_1} & w_{j_2} & w_{j_3} \end{bmatrix} = \frac{1}{2}\begin{bmatrix} 1 & \beta & \beta^2 & \beta^3 \end{bmatrix}\begin{bmatrix} 0 & 2 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 2 & -5 & 4 & -1 \\ -1 & 3 & -3 & 1 \end{bmatrix}$$

$$\begin{bmatrix} w_{k_0} & w_{k_1} & w_{k_2} & w_{k_3} \end{bmatrix} = \frac{1}{2}\begin{bmatrix} 1 & \gamma & \gamma^2 & \gamma^3 \end{bmatrix}\begin{bmatrix} 0 & 2 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 2 & -5 & 4 & -1 \\ -1 & 3 & -3 & 1 \end{bmatrix}$$

Zero Tangent Cardinal weights specified by `VK_CUBIC_FILTER_WEIGHTS_ZERO_TANGENT_CARDINAL_QCOM` are derived from the fractions computed earlier.

$$\begin{bmatrix} w_{i_0} & w_{i_1} & w_{i_2} & w_{i_3} \end{bmatrix} = \frac{1}{2}\begin{bmatrix} 1 & \alpha & \alpha^2 & \alpha^3 \end{bmatrix}\begin{bmatrix} 0 & 2 & 0 & 0 \\ -2 & 0 & 2 & 0 \\ 4 & -4 & 2 & -2 \\ -2 & 2 & -2 & 1 \end{bmatrix}$$

$$\begin{bmatrix} w_{j_0} & w_{j_1} & w_{j_2} & w_{j_3} \end{bmatrix} = \frac{1}{2}\begin{bmatrix} 1 & \beta & \beta^2 & \beta^3 \end{bmatrix}\begin{bmatrix} 0 & 2 & 0 & 0 \\ -2 & 0 & 2 & 0 \\ 4 & -4 & 2 & -2 \\ -2 & 2 & -2 & 1 \end{bmatrix}$$

$$\begin{bmatrix} w_{k_0} & w_{k_1} & w_{k_2} & w_{k_3} \end{bmatrix} = \frac{1}{2}\begin{bmatrix} 1 & \gamma & \gamma^2 & \gamma^3 \end{bmatrix}\begin{bmatrix} 0 & 2 & 0 & 0 \\ -2 & 0 & 2 & 0 \\ 4 & -4 & 2 & -2 \\ -2 & 2 & -2 & 1 \end{bmatrix}$$

B-Spline weights specified by `VK_CUBIC_FILTER_WEIGHTS_B_SPLINE_QCOM` are derived from the fractions computed earlier.

$$\begin{bmatrix} w_{i_0} & w_{i_1} & w_{i_2} & w_{i_3} \end{bmatrix} = \frac{1}{6}\begin{bmatrix} 1 & \alpha & \alpha^2 & \alpha^3 \end{bmatrix}\begin{bmatrix} 1 & 4 & 1 & 0 \\ -3 & 0 & 3 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{bmatrix}$$

$$\begin{bmatrix} w_{j_0} & w_{j_1} & w_{j_2} & w_{j_3} \end{bmatrix} = \frac{1}{6}\begin{bmatrix} 1 & \beta & \beta^2 & \beta^3 \end{bmatrix}\begin{bmatrix} 1 & 4 & 1 & 0 \\ -3 & 0 & 3 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{bmatrix}$$

$$\begin{bmatrix} w_{k_0} & w_{k_1} & w_{k_2} & w_{k_3} \end{bmatrix} = \frac{1}{6}\begin{bmatrix} 1 & \gamma & \gamma^2 & \gamma^3 \end{bmatrix}\begin{bmatrix} 1 & 4 & 1 & 0 \\ -3 & 0 & 3 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{bmatrix}$$

Mitchell-Netravali weights specified by `VK_CUBIC_FILTER_WEIGHTS_MITCHELL_NETRAVALI_QCOM` are derived from the fractions computed earlier.

$$\begin{bmatrix} w_{i_0} & w_{i_1} & w_{i_2} & w_{i_3} \end{bmatrix} = \frac{1}{18}\begin{bmatrix} 1 & \alpha & \alpha^2 & \alpha^3 \end{bmatrix}\begin{bmatrix} 1 & 16 & 1 & 0 \\ -9 & 0 & 9 & 0 \\ 15 & -36 & 27 & -6 \\ -7 & 21 & -21 & 7 \end{bmatrix}$$

$$\begin{bmatrix} w_{j_0} & w_{j_1} & w_{j_2} & w_{j_3} \end{bmatrix} = \frac{1}{18}\begin{bmatrix} 1 & \beta & \beta^2 & \beta^3 \end{bmatrix}\begin{bmatrix} 1 & 16 & 1 & 0 \\ -9 & 0 & 9 & 0 \\ 15 & -36 & 27 & -6 \\ -7 & 21 & -21 & 7 \end{bmatrix}$$

$$\begin{bmatrix} w_{k_0} & w_{k_1} & w_{k_2} & w_{k_3} \end{bmatrix} = \frac{1}{18}\begin{bmatrix} 1 & \gamma & \gamma^2 & \gamma^3 \end{bmatrix}\begin{bmatrix} 1 & 16 & 1 & 0 \\ -9 & 0 & 9 & 0 \\ 15 & -36 & 27 & -6 \\ -7 & 21 & -21 & 7 \end{bmatrix}$$

The values of multiple texels, together with their weights, are combined to produce a filtered value.

The VkSamplerReductionModeCreateInfo::`reductionMode` **can** control the process by which multiple texels, together with their weights, are combined to produce a filtered texture value.

When the `reductionMode` is set (explicitly or implicitly) to `VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE` or `VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE_RANGECLAMP_QCOM`, a weighted average is computed:

$$\tau_{3D} = \sum_{k=j_0}^{k_3} \sum_{j=j_0}^{j_3} \sum_{i=i_0}^{i_3} (w_i)(w_j)(w_k)\tau_{ijk}$$

$$\tau_{2D} = \sum_{j=j_0}^{j_3} \sum_{i=i_0}^{i_3} (w_i)(w_j)\tau_{ij}$$

$$\tau_{1D} = \sum_{i=i_0}^{i_3} (w_i)\tau_i$$

However, if the reduction mode is `VK_SAMPLER_REDUCTION_MODE_MIN` or `VK_SAMPLER_REDUCTION_MODE_MAX`, the process operates on the above set of multiple texels, together with their weights, computing a component-wise minimum or maximum, respectively, of the components of the set of texels with non-zero weights.

#### Texel Range Clamp

When the `reductionMode` is set to `VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE_RANGECLAMP_QCOM`, the weighted average is clamped to be within the component-wise minimum and maximum of the set of texels with non-zero weights.

#### Texel Mipmap Filtering

`VK_SAMPLER_MIPMAP_MODE_NEAREST` filtering returns the value of a single mipmap level,

$\tau = \tau[d]$.

`VK_SAMPLER_MIPMAP_MODE_LINEAR` filtering combines the values of multiple mipmap levels ($\tau[hi]$ and

$\tau$[lo]), together with their linear weights.

The linear weights are derived from the fraction computed earlier:

$$w_{hi} = (1 - \delta)$$
$$w_{lo} = (\delta)$$

The values of multiple mipmap levels, together with their weights, are combined to produce a final filtered value.

The VkSamplerReductionModeCreateInfo::`reductionMode` **can** control the process by which multiple texels, together with their weights, are combined to produce a filtered texture value.

When the `reductionMode` is set (explicitly or implicitly) to `VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE`, a weighted average is computed:

$$\tau = (w_{hi})\tau[hi] + (w_{lo})\tau[lo]$$

However, if the reduction mode is `VK_SAMPLER_REDUCTION_MODE_MIN` or `VK_SAMPLER_REDUCTION_MODE_MAX`, the process operates on the above values, together with their weights, computing a component-wise minimum or maximum, respectively, of the components of the values with non-zero weights.

#### Texel Anisotropic Filtering

Anisotropic filtering is enabled by the `anisotropyEnable` in the sampler. When enabled, the image filtering scheme accounts for a degree of anisotropy.

The particular scheme for anisotropic texture filtering is implementation-dependent. Implementations **should** consider the `magFilter`, `minFilter` and `mipmapMode` of the sampler to control the specifics of the anisotropic filtering scheme used. In addition, implementations **should** consider `minLod` and `maxLod` of the sampler.

**NOTE**

For historical reasons, vendor implementations of anisotropic filtering interpret these sampler parameters in different ways, particularly in corner cases such as `magFilter`, `minFilter` of `NEAREST` or `maxAnisotropy` equal to 1.0. Applications should not expect consistent behavior in such cases, and should use anisotropic filtering only with parameters which are expected to give a quality improvement relative to `LINEAR` filtering.

The following describes one particular approach to implementing anisotropic filtering for the 2D Image case; implementations **may** choose other methods:

Given a `magFilter`, `minFilter` of `VK_FILTER_LINEAR` and a `mipmapMode` of `VK_SAMPLER_MIPMAP_MODE_NEAREST`:

Instead of a single isotropic sample, N isotropic samples are sampled within the image footprint of the image level d to approximate an anisotropic filter. The sum $\tau_{2Daniso}$ is defined using the single isotropic $\tau_{2D}$(u,v) at level d.

$$\tau_{2Daniso} = \frac{1}{N}\sum_{i=1}^{N} \tau_{2D}\left(u\left(x - \frac{1}{2} + \frac{i}{N+1}, y\right), v\left(x - \frac{1}{2} + \frac{i}{N+1}, y\right)\right), \qquad \text{when } \rho_x > \rho_y$$

$$\tau_{2Daniso} = \frac{1}{N}\sum_{i=1}^{N} \tau_{2D}\left(u\left(x, y - \frac{1}{2} + \frac{i}{N+1}\right), v\left(x, y - \frac{1}{2} + \frac{i}{N+1}\right)\right), \qquad \text{when } \rho_y \geq \rho_x$$

When VkSamplerReductionModeCreateInfo::`reductionMode` is set to `VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE`, the above summation is used. However, if the reduction mode is `VK_SAMPLER_REDUCTION_MODE_MIN` or `VK_SAMPLER_REDUCTION_MODE_MAX`, the process operates on the above values, together with their weights, computing a component-wise minimum or maximum, respectively, of the components of the values with non-zero weights.

## 16.9. Texel Footprint Evaluation

The SPIR-V instruction `OpImageSampleFootprintNV` evaluates the set of texels from a single mip level that would be accessed during a texel filtering operation. In addition to the inputs that would be accepted by an equivalent `OpImageSample*` instruction, `OpImageSampleFootprintNV` accepts two additional inputs. The `Granularity` input is an integer identifying the size of texel groups used to evaluate the footprint. Each bit in the returned footprint mask corresponds to an aligned block of texels whose size is given by the following table:

*Table 29. Texel footprint granularity values*

| `Granularity` | `Dim` = **2D** | `Dim` = **3D** |
|---|---|---|
| 0 | unsupported | unsupported |
| 1 | 2x2 | 2x2x2 |
| 2 | 4x2 | unsupported |
| 3 | 4x4 | 4x4x2 |
| 4 | 8x4 | unsupported |
| 5 | 8x8 | unsupported |
| 6 | 16x8 | unsupported |
| 7 | 16x16 | unsupported |
| 8 | unsupported | unsupported |
| 9 | unsupported | unsupported |
| 10 | unsupported | 16x16x16 |
| 11 | 64x64 | 32x16x16 |
| 12 | 128x64 | 32x32x16 |
| 13 | 128x128 | 32x32x32 |
| 14 | 256x128 | 64x32x32 |
| 15 | 256x256 | unsupported |

The `Coarse` input is used to select between the two mip levels that **may** be accessed during texel

filtering when using a `mipmapMode` of `VK_SAMPLER_MIPMAP_MODE_LINEAR`. When filtering between two mip levels, a `Coarse` value of `true` requests the footprint in the lower-resolution mip level (higher level number), while `false` requests the footprint in the higher-resolution mip level. If texel filtering would access only a single mip level, the footprint in that level would be returned when `Coarse` is set to `false`; an empty footprint would be returned when `Coarse` is set to `true`.

The footprint for `OpImageSampleFootprintNV` is returned in a structure with six members:

- The first member is a boolean value that is true if the texel filtering operation would access only a single mip level.
- The second member is a two- or three-component integer vector holding the footprint anchor location. For two-dimensional images, the returned components are in units of eight texel groups. For three-dimensional images, the returned components are in units of four texel groups.
- The third member is a two- or three-component integer vector holding a footprint offset relative to the anchor. All returned components are in units of texel groups.
- The fourth member is a two-component integer vector mask, which holds a bitfield identifying the set of texel groups in an 8x8 or 4x4x4 neighborhood relative to the anchor and offset.
- The fifth member is an integer identifying the mip level containing the footprint identified by the anchor, offset, and mask.
- The sixth member is an integer identifying the granularity of the returned footprint.

For footprints in two-dimensional images (`Dim2D`), the mask returned by `OpImageSampleFootprintNV` indicates whether each texel group in a 8x8 local neighborhood of texel groups would have one or more texels accessed during texel filtering. In the mask, the texel group with local group coordinates $(lgx, lgy)$ is considered covered if and only if

$$0 \neq ((mask.x + (mask.y << 32)) \& (1 << (lgy \times 8 + lgx)))$$

where:

- $0 \leq lgx < 8$ and $0 \leq lgy < 8$; and
- $mask$ is the returned two-component mask.

The local group with coordinates $(lgx, lgy)$ in the mask is considered covered if and only if the texel filtering operation would access one or more texels $\tau_{ij}$ in the returned mip level where:

$$\begin{aligned}
i0 &= \begin{cases} gran.x \times (8 \times anchor.x + lgx), & \text{if} lgx + offset.x < 8 \\ gran.x \times (8 \times (anchor.x - 1) + lgx), & \text{otherwise} \end{cases} \\
i1 &= i0 + gran.x - 1 \\
j0 &= \begin{cases} gran.y \times (8 \times anchor.y + lgy), & \text{if} lgy + offset.y < 8 \\ gran.y \times (8 \times (anchor.y - 1) + lgy), & otherwise \end{cases} \\
j1 &= j0 + gran.y - 1
\end{aligned}$$

and

- $i0 \le i \le i1$ and $j0 \le j \le j1$;
- $gran$ is a two-component vector holding the width and height of the texel group identified by the granularity;
- $anchor$ is the returned two-component anchor vector; and
- $offset$ is the returned two-component offset vector.

For footprints in three-dimensional images (`Dim3D`), the mask returned by `OpImageSampleFootprintNV` indicates whether each texel group in a 4x4x4 local neighborhood of texel groups would have one or more texels accessed during texel filtering. In the mask, the texel group with local group coordinates $(lgx, lgy, lgz)$, is considered covered if and only if:

$$0 \neq ((mask.x + (mask.y << 32)) \& (1 << (lgz \times 16 + lgy \times 4 + lgx)))$$

where:

- $0 \le lgx < 4$, $0 \le lgy < 4$, and $0 \le lgz < 4$; and
- $mask$ is the returned two-component mask.

The local group with coordinates $(lgx, lgy, lgz)$ in the mask is considered covered if and only if the texel filtering operation would access one or more texels $\tau_{ijk}$ in the returned mip level where:

$$\begin{aligned}
i0 &= \begin{cases} gran.x \times (4 \times anchor.x + lgx), & \text{if} lgx + offset.x < 4 \\ gran.x \times (4 \times (anchor.x - 1) + lgx), & \text{otherwise} \end{cases} \\
i1 &= i0 + gran.x - 1 \\
j0 &= \begin{cases} gran.y \times (4 \times anchor.y + lgy), & \text{if} lgy + offset.y < 4 \\ gran.y \times (4 \times (anchor.y - 1) + lgy), & otherwise \end{cases} \\
j1 &= j0 + gran.y - 1 \\
k0 &= \begin{cases} gran.z \times (4 \times anchor.z + lgz), & \text{if} lgz + offset.z < 4 \\ gran.z \times (4 \times (anchor.z - 1) + lgz), & otherwise \end{cases} \\
k1 &= k0 + gran.z - 1
\end{aligned}$$

and

- $i0 \le i \le i1$, $j0 \le j \le j1$, $k0 \le k \le k1$;
- $gran$ is a three-component vector holding the width, height, and depth of the texel group identified by the granularity;
- $anchor$ is the returned three-component anchor vector; and
- $offset$ is the returned three-component offset vector.

If the sampler used by `OpImageSampleFootprintNV` enables anisotropic texel filtering via `anisotropyEnable`, it is possible that the set of texel groups accessed in a mip level may be too large to be expressed using an 8x8 or 4x4x4 mask using the granularity requested in the instruction. In this case, the implementation uses a texel group larger than the requested granularity. When a larger texel group size is used, `OpImageSampleFootprintNV` returns an integer granularity value that **can** be interpreted in the same manner as the granularity value provided to the instruction to

determine the texel group size used. If anisotropic texel filtering is disabled in the sampler, or if an anisotropic footprint can be represented as an 8x8 or 4x4x4 mask with the requested granularity, OpImageSampleFootprintNV will use the requested granularity as-is and return a granularity value of zero.

OpImageSampleFootprintNV supports only two- and three-dimensional image accesses (Dim2D and Dim3D), and the footprint returned is undefined if a sampler uses an addressing mode other than VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_EDGE.

# 16.10. Weight Image Sampling

The SPIR-V instruction OpImageWeightedSampleQCOM specifies a texture sampling operation involving two images: the *sampled image* and the *weight image*. It is similar to bilinear filtering except more than 2x2 texels may participate in the filter and the filter weights are application-specified rather than computed by fixed-function hardware. The weight image view defines a 2D kernel weights used during sampling.

The OpImageWeightedSampleQCOM support normalized or unnormalized texel coordinates. In addition to the inputs that would be accepted by an equivalent OpImageSample* instruction, OpImageWeightedSampleQCOM accepts a weight input that specifies the view of a sample weight image

The input weight **must** be a view of a 2D or 1D image with miplevels equal to 1, samples equal to VK_SAMPLE_COUNT_1_BIT, created with an identity swizzle, and created with usage that includes VK_IMAGE_USAGE_SAMPLE_WEIGHT_BIT_QCOM. The VkImageViewSampleWeightCreateInfoQCOM specifies additional parameters of the view: filterCenter, filterSize, and numPhases. described in more detail below.

The weight input **must** be bound using a sample weight image descriptor type. The weight view defines a filtering kernel that is a region of view's subresource range. The kernel spans a region from integer texel coordinate (0,0) to (filterSize.x-1, filterSize.y-1). It is valid for the view's subresource to have dimensions larger than the kernel but the texels with integer coordinates greater than (filterSize.width-1, filterSize.height-1) are ignored by weight sampling. The value returned by queries OpImageQuerySize, OpImageQuerySizeLod, OpImageQueryLevels, and OpImageQuerySamples return for a weight image is undefined.

filterCenter designates an integer texel coordinate within the filter kernel as being the 'center' of the kernel. The center **must** be in the range (0,0) to (filterSize.x-1, filterSize.y-1). numPhases describes the number of filter phases used to provide sub-pixel filtering. Both are described in more detail below.

## 16.10.1. Weight Image Layout

The weight image specifies filtering kernel weight values. A 2D image view can be used to specify a 2D matrix of filter weights. For separable filers, a 1D image view can be used to specity the horizontal and vertical weights.

### 2D Non-Separable Weight Filters

A 2D image view defined with VkImageViewSampleWeightCreateInfoQCOM describes a 2D matrix

(`filterSize.width` × `filterSize.height`) of weight elements with filter's center point at `filterCenter`. Note that `filterSize` can be smaller than the view's subresource, but the filter will always be located starting at integer texel coordinate (0,0).

The following figure illustrates a 2D convolution filter having `filterSize` of (4,3) and `filterCenter` at (1, 1).

| | | | |
|---|---|---|---|
| *(0,0)* | *(1,0)* | *(2,0)* | *(3,0)* |
| *(0,1)* | *origin* | *(2,1)* | *(3,1)* |
| *(0,2)* | *(1,2)* | *(2,2)* | *(3,2)* |

*2D Convolution Filter*

*Figure 12. 2D Convolution Filter*

For a 2D weight filter, the phases are stored as layers of a 2D array image. The width and height of the view's subresource range **must** be less than or equal to VkPhysicalDeviceImageProcessingPropertiesQCOM::`maxWeightFilterDimension`. The layers are stored in horizontal phase major order. Expressed as a formula, the layer index for each filter phase is computed as:

```
layerIndex(horizPhase,vertPhase,horizPhaseCount) = (vertPhase * horizPhaseCount) +
horizPhase
```

### 1D Separable Weight Filters

A separable weight filter is a 2D filter that can be specified by two 1D filters in the x and y directions such that their product yields the 2D filter. The following example shows a 2D filter and its associated separable 1D horizontal and vertical filters.

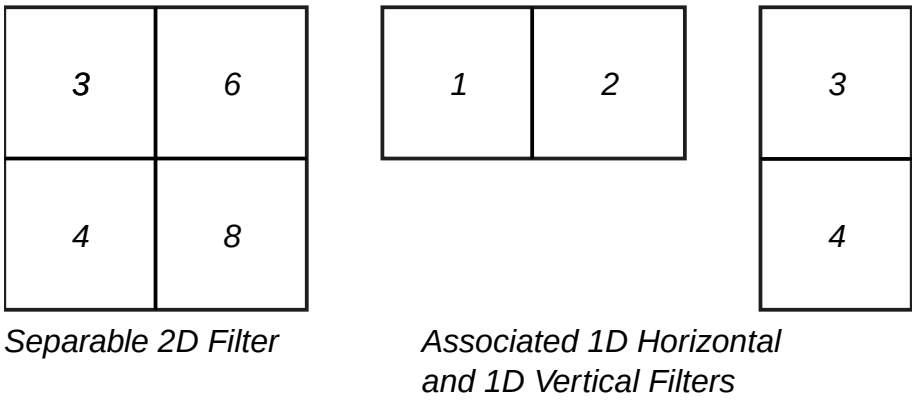



*Figure 13. Separable 2D Convolution Filter*

A 1D array image view defined with VkImageViewSampleWeightCreateInfoQCOM and with `layerCount` equal to '2' describes a separable weight filter. The horizontal weights are specified in slice '0' and the vertical weights in slice '1'. The `filterSize` and `filterCenter` specify the size and origin of the of the horizontal and vertical filters. For many use cases, 1D separable filters can offer a performance advantage over 2D filters.

For a 1D separable weight filter, the phases are arranged into a 1D array image with two layers. The horizontal weights are stored in layer 0 and the vertical weights in layer 1. Within each layer of the 1D array image, the weights are arranged into groups of 4, and then arranged by phase. Expressed as a formula, the 1D texel offset for each weight within each layer is computed as:

```
// Let horizontal weights have a weightIndex of [0, filterSize.width - 1]
// Let vertical weights have a weightIndex of [0, filterSize.height - 1]
// Let phaseCount be the number of phases in either the vertical or horizontal
direction.

texelOffset(phaseIndex,weightIndex,phaseCount) = (phaseCount * 4 * (weightIndex / 4))
+ (phaseIndex * 4) + (weightIndex % 4)
```

### 16.10.2. Weight Sampling Phases

When using weight image sampling, the texture coordinates may not align with a texel center in the sampled image. In this case, the filter weights can be adjusted based on the subpixel location. This is termed “subpixel filtering” to indicate that the origin of the filter lies at a subpixel location other than the texel center. Conceptually, this means that the weight filter is positioned such that filter taps do not align with sampled texels exactly. In such a case, modified filter weights may be needed to adjust for the off-center filter taps. Unlike bilinear filtering where the subpixel weights are computed by the implementation, subpixel weight image sampling requires that the per-phase filter weights are pre-computed by the application and stored in an array where each slice of the array is a “filter phase”. The array is indexed by the implementation based on subpixel positioning. Rather than a single 2D kernel of filter weights, the application provides an array of kernels, one set

of filter weights per phase.

The number of phases are restricted by following requirements, which apply to both separable and non-separable filters:

- The number of phases in the vertical direction, $\text{phaseCount}_{\text{vert}}$, **must** be a power of two (i.e., 1, 2, 4, etc.).
- The number of phases in the horizontal direction $\text{phaseCount}_{\text{horiz}}$, **must** equal $\text{phaseCount}_{\text{vert}}$.
- The total number of phases, $\text{phaseCount}_{\text{vert}} \times \text{phaseCount}_{\text{horiz}}$, **must** be less than or equal to VkPhysicalDeviceImageProcessingPropertiesQCOM::`maxWeightFilterPhases`.

### 16.10.3. Weight Sampler Parameters

Weight sampling requires `VkSamplerCreateInfo` `addressModeU` and `addressModeV` **must** be set to `VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_EDGE` or `VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_BORDER`. If `VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_BORDER` is used, then the border color **must** be set to transparent black.

### 16.10.4. Weight Sampling Operation

The 2D unnormalized texel coordinates $(u, v)$ are transformed by $filterCenter$ to specify coordinates $i_0$, $j_0$.

$$i_0 = \lfloor u - filterCenter_x \rfloor$$

$$j_0 = \lfloor v - filterCenter_y \rfloor$$

where $filterCenter$ is specified by VkImageViewSampleWeightCreateInfoQCOM::`filterCenter`.

Two sets of neighboring integer 2D texel coordinates are generated. The first set is used for selecting texels from the sampled image $\tau$ and the second set used for selecting texels from the weight image $w$. The first set of neighboring coordinates are combinations of $i_0$ to $i_{filterWidth-1}$ and $j_0$ to $j_{filterHeight-1}$. The second set of neighboring coordinates are combinations of $k_0$ to $k_{filterWidth-1}$ and $l_0$ to $l_{filterHeight-1}$. The first and second sets each contain $(filterWidth \times filterHeight)$ of pairs of $(i, j)$ and $(k, l)$ coordinates respectively.

$$\{i_q\}_{q=0}^{q=filterWidth-1} = i_0 + q$$

$$\{j_q\}_{q=0}^{q=filterHeight-1} = j_0 + q$$

$$\{k_q\}_{q=0}^{q=filterWidth-1} = q$$

$$\{l_q\}_{q=0}^{q=filterHeight-1} = q$$

where $filterWidth$ and $filterHeight$ are specified by VkImageViewSampleWeightCreateInfoQCOM::`filterSize`.

Each of the generated integer coordinates $(i_q, j_q)$ is transformed by texture wrapping operation,

followed by integer texel coordinate validation, If any coordinate fails coordinate validation, it is a Border Texel and texel replacement is performed.

The phase index $\psi$ is computed from the fraction bits of the unnormalized 2D texel coordinates:

$$
\begin{aligned}
phaseCount_h = phaseCount_v &= \sqrt{numPhases} \\
hPhase &= \lfloor \operatorname{frac}(u) \times phaseCount_h \rfloor \\
vPhase &= \lfloor \operatorname{frac}(v) \times phaseCount_v \rfloor \\
\psi &= (vPhase \times phaseCount_h) + hPhase
\end{aligned}
$$

where the number of fraction bits retained is $\log2(numPhases)$ specified by VkImageViewSampleWeightCreateInfoQCOM::`numPhases`

Each pair of texel coordinates $(i, j)$ in the first set selects a single texel value $\tau_{ij}$ from the sampled image. Each pair of texel coordinates $(k, l)$ in the second set, combined with phaseIndex $\psi$, selects a single weight from the weight image $w(k, l, \psi)$ .

$$
w(k, l, \psi) = \begin{cases} w_{kl}[\psi] \quad (\psi \text{as layer index}) & \text{for 2D array view (non-separable filter)} \\ weight_h \times weight_v & \text{for 1D array view (separable filter)} \end{cases}
$$

If $w$ is a 2D array view, then non-separable filtering is specified, and integer coordinates $(k, l)$ are used to select texels from layer $\psi$ of $(w)$. If $w$ is a 1D array view, then separable filtering is specified and integer coordinates $(k, l)$ are transformed to $(k_{packed}, l_{packed})$, and used to select horizontal weight $(weight_h)$ and vertical weight $(weight_v)$ texels from layer 0 and layer 1 of $(w)$ respectively.

$$
\begin{aligned}
k_{packed} &= (phaseCount_h \times 4 \times \lfloor k / 4 \rfloor) + (hPhase \times 4) + (k \% 4) \\
l_{packed} &= (phaseCount_v \times 4 \times \lfloor l / 4 \rfloor) + (vPhase \times 4) + (l \% 4) \\
weight_h &= w_{k_{packed}}[0] && \text{(horizontal weights packed in layer 0)} \\
weight_v &= w_{l_{packed}}[1] && \text{(vertical weights packed in layer 1)}
\end{aligned}
$$

Where $\%$ refers to the integer modulo operator.

The values of multiple texels, together with their weights, are combined to produce a filtered value.

$$
\tau_{weightSampling} = \sum_{\substack{j = j_0 \\ l = l_0}}^{\substack{j_{blockHeight} - 1 \\ l_{blockHeight} - 1}} \sum_{\substack{i = i_0 \\ k = k_0}}^{\substack{i_{blockWidth} - 1 \\ k_{blockWidth} - 1}} w(k, l, \psi)\tau_{ij}
$$

When VkSamplerReductionModeCreateInfo::`reductionMode` is set to `VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE`, the above summation is used. However, if the reduction mode is `VK_SAMPLER_REDUCTION_MODE_MIN` or `VK_SAMPLER_REDUCTION_MODE_MAX`, the process operates on the above values, computing a component-wise minimum or maximum of the texels with non-zero weights. If the reduction mode is `VK_SAMPLER_REDUCTION_MODE_MIN` or

VK_SAMPLER_REDUCTION_MODE_MAX, each $w(k, l, \psi)$ weight **must** be equal to 0.0 or 1.0, otherwise the undefined values are returned.

Finally, the operations described in Conversion to RGBA and Component swizzle are performed and the final result is returned to the shader.

# 16.11. Block Matching

The SPIR-V instruction opImageBlockMatchSAD and opImageBlockMatchSSD specify texture block matching operations where a block or region of texels within a *target image* is compared with a same-sized region a *reference image*. The instructions make use of two image views: the *target view* and the *reference view*. The target view and reference view can be the same view, allowing block matching of two blocks within a single image.

Similar to an equivalent OpImageFetch instruction, opImageBlockMatchSAD and opImageBlockMatchSAD specify an image and an integer texel coordinate which describes the bottom-left texel of the target block. There are three additional inputs. The reference and refCoodinate specifies bottom-left texel of the reference block. The blockSize specifies the integer width and height of the target and reference blocks to be compared, and **must** not be greater than VkPhysicalDeviceImageProcessingPropertiesQCOM.maxBlockMatchRegion.

opImageBlockMatchWindowSAD and opImageBlockMatchWindowSAD take the same input parameters as the corresponding non-window instructions. The block matching comparison is performed for all pixel values within a 2D window whose dimensions are specified in the sampler.

## 16.11.1. Block Matching Sampler Parameters

For opImageBlockMatchSAD and opImageBlockMatchSSD, the input sampler **must** be created with addressModeU and addressModeV, equal to VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_EDGE, or VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_BORDER with VK_BORDER_COLOR_FLOAT_TRANSPARENT_BLACK. The input sampler **must** be created with unnormalizedCoordinates equal to VK_TRUE. The input sampler **must** be created with components equal to VK_COMPONENT_SWIZZLE_IDENTITY.

For opImageBlockMatchWindowSAD and opImageBlockMatchWindowSSD instructions, the target sampler **must** have been created with VkSamplerBlockMatchWindowCreateInfoQCOM in the pNext chain.

For opImageBlockMatchWindowSAD, opImageBlockMatchWindowSSD, opImageBlockMatchGatherSAD, or opImageBlockMatchGatherSSDinstructions, the input sampler **must** be created with addressModeU and addressModeV, equal to VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_BORDER with VK_BORDER_COLOR_FLOAT_TRANSPARENT_BLACK.

Other sampler states are ignored.

## 16.11.2. Block Matching Operation

Block matching SPIR-V instructions opImageBlockMatchSAD and opImageBlockMatchSSD specify two sets of 2D integer texel coordinates: target coordinates $(u, v)$ and reference coordinates $(s, t)$.

The coordinates define the bottom-left texel of the target block $(i_0, j_0)$ and the reference block $(k_0, l_0)$.

$$i_0 = u$$

$$j_0 = v$$

$$k_0 = s$$

$$l_0 = t$$

For the target block, a set of neighboring integer texel coordinates are generated. The neighboring coordinates are combinations of $i_0$ to $i_{blockWidth-1}$ and $j_0$ to $j_{blockHeight-1}$. The set is of size $blockWidth \times blockHeight$.

$$\{i_q\}_{q=0}^{q=blockWidth-1} = i_0 + q$$

$$\{j_q\}_{q=0}^{q=blockHeight-1} = j_0 + q$$

where $blockWidth$ and $blockHeight$ is specified by the `blockSize` operand.

If any target integer texel coordinate $(i, j)$ in the set fails integer texel coordinate validation, then the texel is an invalid texel and texel replacement is performed.

Similarly for the reference block, a set of neighboring integer texel coordinates are generated.

$$\{k_q\}_{q=0}^{q=blockWidth-1} = k_0 + q$$

$$\{l_q\}_{q=0}^{q=blockHeight-1} = l_0 + q$$

Each reference texel coordinate $(k, l)$ in the set **must** not fail integer texel coordinate validation. To avoid undefined behavior, application shader should guarantee that the reference block is fully within the bounds of the reference image.

Each pair of texel coordinates $(i, j)$ in the set selects a single texel value from the target image $\tau_{ij}$. Each pair of texel coordinates $(k, l)$ in the set selects a single texel value from the reference image $\upsilon_{kl}$.

The difference between target and reference texel values is summed to compute a difference metric. The `opTextureBlockMatchSAD` computes the sum of absolute differences.

$$\tau_{SAD} = \sum_{\substack{j=j_0 \\ l=l_0}}^{\substack{j_{blockHeight}-1 \\ l_{blockHeight}-1}} \sum_{\substack{i=i_0 \\ k=k_0}}^{\substack{i_{blockWidth}-1 \\ k_{blockWidth}-1}} |\upsilon_{kl} - \tau_{ij}|$$

The `opImageBlockMatchSSD` computes the sum of the squared differences.

$$\tau_{SSD} = \sum_{\substack{j = j_0 \\ l = l_0}}^{\substack{j_{blockHeight} - 1 \\ l_{blockHeight} - 1}} \sum_{\substack{i = i_0 \\ k = k_0}}^{\substack{i_{blockWidth} - 1 \\ k_{blockWidth} - 1}} |\upsilon_{kl} - \tau_{ij}|^2$$

When VkSamplerReductionModeCreateInfo::`reductionMode` is set to `VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE`, the above summation is used. However, if the reduction mode is `VK_SAMPLER_REDUCTION_MODE_MIN` or `VK_SAMPLER_REDUCTION_MODE_MAX`, the process operates on the above values, computing a component-wise minimum or maximum of $|\upsilon_{kl} - \tau_{ij}|$, respectively. For $\tau_{SAD}$, the minimum or maximum difference is computed and for $\tau_{SSD}$, the square of the minimum or maximum is computed.

Finally, the operations described in Conversion to RGBA and Component swizzle are performed and the final result is returned to the shader. The component swizzle is specified by the *target image* descriptor; any swizzle specified by the *reference image* descriptor is ignored.

#### Block Matching Window Operation

Window block matching SPIR-V instructions `opImageBlockMatchWindowSAD` and `opImageBlockMatchWindowSSD` specify two sets of 2D integer texel coordinates: target coordinates $(u, v)$ and reference coordinates $(s, t)$. The block matching operation is performed repeatedly, for multiple sets of target integer coordinates within the specified window. These instructions effectively search a region or "window" within the target texture and identify the window coordinates where the minimum or maximum error metric is found. These instructions only support single component image formats.

The target coordinates are combinations of coordinates from $(u, v)$ to $(u + windowWidth - 1, v + windowHeight - 1)$ where $windowHeight$ and $windowWidth$ are specified by VkSamplerBlockMatchWindowCreateInfoQCOM::`windowExtent`. At each target coordinate, a block matching operation is performed, resulting in a difference metric. The reference coordinate $(s, t)$ is fixed. The block matching operation is repeated $windowWidth \times windowHeight$ times.

The resulting minimum or maximum error is returned in the R component of the output. The integer window coordinates $(x, y)$ are returned in the G and B components of the output. The A component is 0. The minimum or maximum behavior is selected by VkSamplerBlockMatchWindowCreateInfoQCOM::`windowCompareMode`.

The following pseudocode describes the operation `opImageBlockMatchWindowSAD`. The pseudocode for `opImageBlockMatchWindowSSD` follows an identical pattern.

```
vec4 opImageBlockMatchGatherSAD( sampler2D target,
                                 uvec2 targetCoord,
                                 samler2D reference,
                                 uvec2 refCoord,
                                 uvec2 blocksize) {
    // Two parameters are sourced from the VkSampler associated with
    // `target`:
    //    compareMode  (which can be either `MIN` or `MAX`)
```

```
    //    uvec2 window (which defines the search window)

    minSAD = INF;
    maxSAD = -INF;
    uvec2 minCoord;
    uvec2 maxCoord;

    for (uint x=0, x<window.width; x++) {
        for (uint y=0; y<window.height; y++) {
            float SAD = textureBlockMatchSAD(target,
                                             targetCoord + uvec2(x, y),
                                             reference,
                                             refCoord,
                                             blocksize).x;
            if (SAD < minSAD) {
                minSAD = SAD;
                minCoord = uvec2(x,y);
            }
            if (SAD > maxSAD) {
                maxSAD = SAD;
                maxCoord = uvec2(x,y);
            }
        }
    }
    if (compareMode==MIN) {
        return vec4(minSAD, minCoord.x, minCoord.y, 0.0);
    } else {
        return vec4(maxSAD, maxCoord.x, maxCoord.y, 0.0);
    }
}
```

### Block Matching Gather Operation

Block matching Gather SPIR-V instructions `opImageBlockMatchGatherSAD` and `opImageBlockMatchGatherSSD` specify two sets of 2D integer texel coordinates: target coordinates $(u, v)$ and reference coordinates $(s, t)$.

These instructions perform the block matching operation 4 times, using integer target coordinates $(u, v)$, $(u+1, v)$, $(u+2, v)$, and $(u+3, v)$. The R component from each of those 4 operations is gathered and returned in the R, G, B, and A components of the output respectively. For each block match operation, the reference coordinate is $(s, t)$. For each block match operation, only the R component of the target and reference images are compared. The following pseudocode describes the operation opImageBlockMatchGatherSAD. The pseudocode for opImageBlockMatchGatherSSD follows an identical pattern.

```
vec4 opImageBlockMatchGatherSAD(sampler2D target,
                                uvec2 targetCoord,
                                samler2D reference,
                                uvec2 refCoord,
```

```
                                uvec2 blocksize) {
    vec4 out;
    for (uint x=0, x<4; x++) {
            float SAD = textureBlockMatchSAD(target,
                                             targetCoord + uvec2(x, 0),
                                             reference,
                                             refCoord,
                                             blocksize).x;
            if (x == 0) {
                out.x = SAD;
            }
            if (x == 1) {
                out.y = SAD;
            }
            if (x == 2) {
                out.z = SAD;
            }
            if (x == 3) {
                out.w = SAD;
            }
    }
    return out;
}
```

## 16.12. Box Filter Sampling

The SPIR-V instruction `OpImageBoxFilterQCOM` specifies texture box filtering operation where a weighted average of a region of texels is computed, with the weights proportional to the coverage of each of the texels.

In addition to the inputs that would be accepted by an equivalent `OpImageSample*` instruction, `OpImageBoxFilterQCOM` accepts one additional input, `boxSize` which specifies the width and height in texels of the region to be averaged.

The figure below shows an example of using `OpImageBoxFilterQCOM` to sample from a 8 × 4 texel two-dimensional image, with unnormalized texture coordinates (4.125, 2.625) and `boxSize` of (2.75, 2.25). The filter will read 12 texel values and compute a weights based portion of each texel covered by the box.

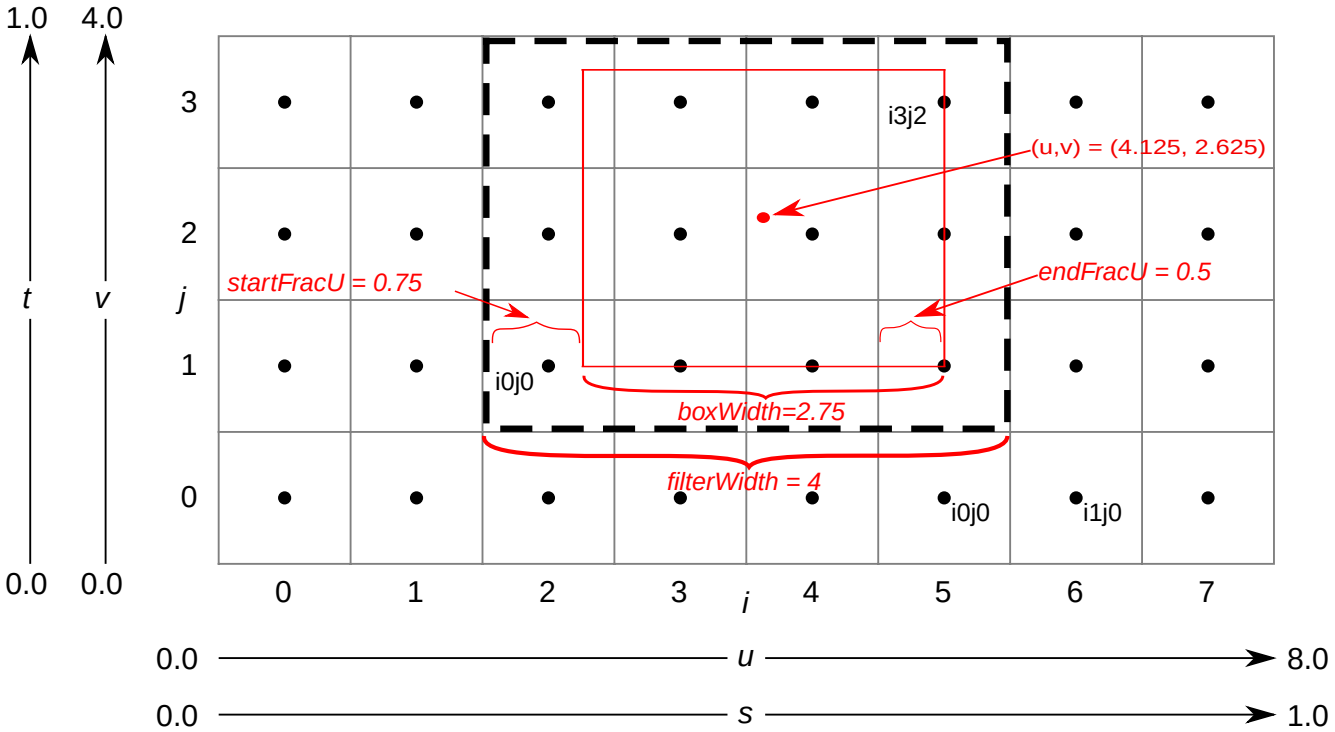



*Figure 14. Box Filter Sampling Example*

If `boxSize` has height and width both equal to 1.0, then this instruction will behave as traditional bilinear filtering. The `boxSize` parameter **must** be greater than or equal to 1.0 and **must** not be greater than VkPhysicalDeviceImageProcessingPropertiesQCOM.`maxBoxFilterBlockSize`.

### 16.12.1. Box Filter Sampler Parameters

The input `sampler` **must** be created with `addressModeU` and `addressModeV`, equal to `VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_EDGE`, or `VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_BORDER` with `VK_BORDER_COLOR_FLOAT_TRANSPARENT_BLACK`.

### 16.12.2. Box Filter Operation

The 2D unnormalized texel coordinates $(u, v)$ are transformed by $boxSize$ to specify integer texel coordinates $(i_0, j_0)$ of the bottom left texel for the filter.

$$i_0 = \lfloor u - \frac{boxWidth}{2} \rfloor$$

$$j_0 = \lfloor v - \frac{boxHeight}{2} \rfloor$$

where $boxWidth$ and $boxHeight$ are specified by the code:(x,y) components of the `boxSize` operand.

The filter dimensions $(filterWidth \times filterHeight)$ are computed from the fractional portion of the $(u, v)$ coordinates and the $boxSize$.

$$startFracU = \text{frac}\left(u - \frac{boxWidth}{2}\right)$$

$$startFracV = \text{frac}\left(v - \frac{boxHeight}{2}\right)$$

$$endFracU = \text{frac}\left(startFracU + boxWidth\right)$$

$$endFracV = \text{frac}\left(startFracV + boxHeight\right)$$

$$filterWidth = \lceil startFracU + boxWidth \rceil$$

$$filterHeight = \lceil startFracV + boxHeight \rceil$$

where the number of fraction bits retained by $frac()$ is specified by `VkPhysicalDeviceLimits::subTexelPrecisionBits`.

A set of neighboring integer texel coordinates are generated. The neighboring coordinates are combinations of $i_0$ to $i_{filterWidth-1}$ and $j_0$ to $j_{filterHeight-1}$, with $i_0$, $j_0$ being the top-left coordinate of this set. The set is of size $(filterWidth \times filterHeight)$.

$$\{i_q\}_{q=0}^{q=filterWidth-1} = i_0 + q$$

$$\{j_q\}_{q=0}^{q=filterHeight-1} = j_0 + q$$

Each of the generated integer coordinates $(i_q, j_q)$ is transformed by texture wrapping operation, followed by integer texel coordinate validation, If any coordinate fails coordinate validation, it is a Border Texel and texel replacement is performed.

Horizontal weights $horizWeight_0$ to $horizWeight_{boxWidth-1}$ and vertical weights $vertWeight_0$ to $vertWeight_{boxHeight-1}$ are computed. Texels that are fully covered by the box will have a horizontal and vertical weight of 1. Texels partially covered by the box will have will have a reduced weights proportional to the coverage.

$$horizWeight_i = \begin{cases} (1 - startFracU), & \text{for}(i == 0) \\ (endFracU), & \text{for}(i == filterWidth - 1)\text{and}(endFracU != 0) \\ (1), & \text{otherwise} \end{cases}$$

$$vertWeight_j = \begin{cases} (1 - startFracV), & \text{for}(j == 0) \\ (endFracV), & \text{for}(j == filterHeight - 1)\text{and}(endFracV != 0) \\ (1), & \text{otherwise} \end{cases}$$

The values of multiple texels, together with their horizontal and vertical weights, are combined to produce a box filtered value.

$$\tau_{boxFilter} = \frac{1}{boxHeight \times boxWidth} \sum_{j=j_0}^{j_{filterHeight-1}} \sum_{i=i_0}^{i_{filterWidth-1}} (horizWeight_i)(vertWeight_j)\tau_{ij}$$

When VkSamplerReductionModeCreateInfo::reductionMode is set to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE, the above summation is used. However, if the reduction mode is VK_SAMPLER_REDUCTION_MODE_MIN or VK_SAMPLER_REDUCTION_MODE_MAX, the process operates on the above values, computing a component-wise minimum or maximum of the texels.

# 16.13. Image Operation Steps

Each step described in this chapter is performed by a subset of the image instructions:

- Texel Input Validation Operations, Format Conversion, Texel Replacement, Conversion to RGBA, and Component Swizzle: Performed by all instructions except OpImageWrite.
- Depth Comparison: Performed by OpImage*Dref instructions.
- All Texel output operations: Performed by OpImageWrite.
- Projection: Performed by all OpImage*Proj instructions.
- Derivative Image Operations, Cube Map Operations, Scale Factor Operation, LOD Operation and Image Level(s) Selection, and Texel Anisotropic Filtering: Performed by all OpImageSample* and OpImageSparseSample* instructions.
- (s,t,r,q,a) to (u,v,w,a) Transformation, Wrapping, and (u,v,w,a) to (i,j,k,l,n) Transformation And Array Layer Selection: Performed by all OpImageSample, OpImageSparseSample, and OpImage*Gather instructions.
- Texel Gathering: Performed by OpImage*Gather instructions.
- Texel Footprint Evaluation: Performed by OpImageSampleFootprint instructions.
- Texel Filtering: Performed by all OpImageSample* and OpImageSparseSample* instructions.
- Sparse Residency: Performed by all OpImageSparse* instructions.
- (s,t,r,q,a) to (u,v,w,a) Transformation, Wrapping, and Weight Image Sampling: Performed by OpImageWeightedSample* instructions.
- (s,t,r,q,a) to (u,v,w,a) Transformation, Wrapping, and Block Matching: Performed by opImageBlockMatch* instructions.
- (s,t,r,q,a) to (u,v,w,a) Transformation, Wrapping, and Box Filter Sampling: Performed by OpImageBoxFilter* instructions.

# 16.14. Image Query Instructions

## 16.14.1. Image Property Queries

OpImageQuerySize, OpImageQuerySizeLod, OpImageQueryLevels, and OpImageQuerySamples query properties of the image descriptor that would be accessed by a shader image operation. They return 0 if the bound descriptor is a null descriptor.

OpImageQuerySizeLod returns the size of the image level identified by the Level of Detail operand. If that level does not exist in the image, and the descriptor is not null, then the value returned is undefined.

### 16.14.2. LOD Query

`OpImageQueryLod` returns the Lod parameters that would be used in an image operation with the given image and coordinates. If the descriptor that would be accessed is a null descriptor then (0,0) is returned. Otherwise, the steps described in this chapter are performed as if for `OpImageSampleImplicitLod`, up to Scale Factor Operation, LOD Operation and Image Level(s) Selection. The return value is the vector ($\lambda'$, $d_l$ - $level_{base}$). These values **may** be subject to implementation-specific maxima and minima for very large, out-of-range values.

# Chapter 17. Fragment Density Map Operations

## 17.1. Fragment Density Map Operations Overview

When a fragment is generated in a render pass that has a fragment density map attachment, its area is determined by the properties of the local framebuffer region that the fragment occupies. The framebuffer is divided into a uniform grid of these local regions, and their fragment area property is derived from the density map with the following operations:

- Fetch density value
  - Component swizzle
  - Component mapping
- Fragment area conversion
  - Fragment area filter
  - Fragment area clamp

## 17.2. Fetch Density Value

Each local framebuffer region at center coordinate (x,y) fetches a texel from the fragment density map.

First, the local framebuffer region center coordinate (x,y) is offset by the value specified in VkSubpassFragmentDensityMapOffsetEndInfoQCOM. If no offset is specified, then the default offset (0,0) is used. The offsetted coordinate (x',y') is computed as follows:

$$x' = \text{clamp}(x + pFragmentDensityOffsets[layer]_x, 0, framebuffer_{width} - 1)$$
$$y' = \text{clamp}(y + pFragmentDensityOffsets[layer]_y, 0, framebuffer_{height} - 1)$$

The offsetted coordinate (x',y') fetches a texel from the fragment density map at integer coordinates:

$$i = \lfloor \frac{x'}{fragmentDensityTexelSize_{width}} \rfloor$$

$$j = \lfloor \frac{y'}{fragmentDensityTexelSize_{height}} \rfloor$$

Where the size of each region in the framebuffer is:

$$fragmentDensityTexelSize'_{width} = 2^{\lceil \log_2 (\frac{framebuffer_{width}}{fragmentDensityMap_{width}}) \rceil}$$

$$fragmentDensityTexelSize'_{height} = 2^{\lceil \log_2 ( \frac{framebuffer_{height}}{fragmentDensityMap_{height}} ) \rceil}$$

This region is subject to the limits in `VkPhysicalDeviceFragmentDensityMapPropertiesEXT` and therefore the final region size is clamped:

$$fragmentDensityTexelSize_{width} = \text{clamp}(fragmentDensityTexelSize'_{width}, minFragmentDensityTexelSize_{width}, maxFragmentDensityTexelSize_{width})$$

$$fragmentDensityTexelSize_{height} = \text{clamp}(fragmentDensityTexelSize'_{height}, minFragmentDensityTexelSize_{height}, maxFragmentDensityTexelSize_{height})$$

When multiview is enabled for the render pass and the fragment density map attachment view was created with `layerCount` greater than `1`, the layer used for offsets and for fetching from the fragment density map is:

$$layer = baseArrayLayer + ViewIndex$$

Otherwise:

$$layer = baseArrayLayer$$

The texel fetched from the density map at (i,j,layer) is next converted to density with the following operations.

### 17.2.1. Component Swizzle

The `components` member of VkImageViewCreateInfo is applied to the fetched texel as defined in Image component swizzle.

### 17.2.2. Component Mapping

The swizzled texel's components are mapped to a density value:

$$densityValue_{xy} = (C'_r, C'_g)$$

## 17.3. Fragment Area Conversion

Fragment area for the framebuffer region is undefined if the density fetched is not a normalized floating-point value greater than `0.0`. Otherwise, the fetched fragment area for that region is derived as:

$$fragmentArea_{wh} = \frac{1.0}{densityValue_{xy}}$$

### 17.3.1. Fragment Area Filter

Optionally, the implementation **may** fetch additional density map texels in an implementation defined window around (i,j). The texels follow the standard conversion steps up to and including fragment area conversion.

A single fetched fragment area for the framebuffer region is chosen by the implementation and **must** have an area between the *min* and *max* areas of the fetched set.

### 17.3.2. Fragment Area Clamp

The implementation **may** clamp the fetched fragment area to one that it supports. The clamped fragment area **must** have a size less than or equal to the original fetched value. Implementations **may** vary the supported set of fragment areas per framebuffer region. Fragment area (1,1) **must** always be in the supported set.

| | |
|---|---|
| **NOTE** | For example, if the fetched fragment area is (1,4) but the implementation only supports areas of {(1,1),(2,2)}, it could choose to clamp the area to (2,2) since it has the same size as (1,4). While this would produce fragments that have lower quality strictly in the x-axis, the overall density is maintained. |

The clamped fragment area is assigned to the corresponding framebuffer region.

# Chapter 18. Queries

*Queries* provide a mechanism to return information about the processing of a sequence of Vulkan commands. Query operations are asynchronous, and as such, their results are not returned immediately. Instead, their results, and their availability status are stored in a Query Pool. The state of these queries **can** be read back on the host, or copied to a buffer object on the device.

The supported query types are Occlusion Queries, Pipeline Statistics Queries, Result Status Queries, Video Encode Feedback Queries and Timestamp Queries. Performance Queries are supported if the associated extension is available. Transform Feedback Queries are supported if the associated extension is available. Intel Performance Queries are supported if the associated extension is available. Mesh Shader Queries are supported if the associated extension is available.

Several additional queries with specific purposes associated with ray tracing are available if the corresponding extensions are supported, as described for VkQueryType.

## 18.1. Query Pools

Queries are managed using *query pool* objects. Each query pool is a collection of a specific number of queries of a particular type.

Query pools are represented by `VkQueryPool` handles:

```
// Provided by VK_VERSION_1_0
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkQueryPool)
```

To create a query pool, call:

```
// Provided by VK_VERSION_1_0
VkResult vkCreateQueryPool(
    VkDevice                                    device,
    const VkQueryPoolCreateInfo*                pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkQueryPool*                                pQueryPool);
```

- `device` is the logical device that creates the query pool.
- `pCreateInfo` is a pointer to a VkQueryPoolCreateInfo structure containing the number and type of queries to be managed by the pool.
- `pAllocator` controls host memory allocation as described in the Memory Allocation chapter.
- `pQueryPool` is a pointer to a VkQueryPool handle in which the resulting query pool object is returned.

### Valid Usage

- VUID-vkCreateQueryPool-device-09663

device **must** support at least one queue family with one of the VK_QUEUE_VIDEO_ENCODE_BIT_KHR, VK_QUEUE_VIDEO_DECODE_BIT_KHR, VK_QUEUE_COMPUTE_BIT, or VK_QUEUE_GRAPHICS_BIT capabilities

## Valid Usage (Implicit)

- VUID-vkCreateQueryPool-device-parameter
  device **must** be a valid VkDevice handle
- VUID-vkCreateQueryPool-pCreateInfo-parameter
  pCreateInfo **must** be a valid pointer to a valid VkQueryPoolCreateInfo structure
- VUID-vkCreateQueryPool-pAllocator-parameter
  If pAllocator is not NULL, pAllocator **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkCreateQueryPool-pQueryPool-parameter
  pQueryPool **must** be a valid pointer to a VkQueryPool handle

## Return Codes

**Success**

- VK_SUCCESS

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY

The VkQueryPoolCreateInfo structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkQueryPoolCreateInfo {
    VkStructureType                 sType;
    const void*                     pNext;
    VkQueryPoolCreateFlags          flags;
    VkQueryType                     queryType;
    uint32_t                        queryCount;
    VkQueryPipelineStatisticFlags   pipelineStatistics;
} VkQueryPoolCreateInfo;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- flags is reserved for future use.
- queryType is a VkQueryType value specifying the type of queries managed by the pool.
- queryCount is the number of queries managed by the pool.

- `pipelineStatistics` is a bitmask of VkQueryPipelineStatisticFlagBits specifying which counters will be returned in queries on the new pool, as described below in Pipeline Statistics Queries.

`pipelineStatistics` is ignored if `queryType` is not `VK_QUERY_TYPE_PIPELINE_STATISTICS`.

## Valid Usage

- VUID-VkQueryPoolCreateInfo-queryType-00791
  If the `pipelineStatisticsQuery` feature is not enabled, `queryType` **must** not be `VK_QUERY_TYPE_PIPELINE_STATISTICS`
- VUID-VkQueryPoolCreateInfo-meshShaderQueries-07068
  If the `meshShaderQueries` feature is not enabled, `queryType` **must** not be `VK_QUERY_TYPE_MESH_PRIMITIVES_GENERATED_EXT`
- VUID-VkQueryPoolCreateInfo-meshShaderQueries-07069
  If the `meshShaderQueries` feature is not enabled, and `queryType` is `VK_QUERY_TYPE_PIPELINE_STATISTICS`, `pipelineStatistics` **must** not contain `VK_QUERY_PIPELINE_STATISTIC_TASK_SHADER_INVOCATIONS_BIT_EXT` or `VK_QUERY_PIPELINE_STATISTIC_MESH_SHADER_INVOCATIONS_BIT_EXT`
- VUID-VkQueryPoolCreateInfo-queryType-00792
  If `queryType` is `VK_QUERY_TYPE_PIPELINE_STATISTICS`, `pipelineStatistics` **must** be a valid combination of VkQueryPipelineStatisticFlagBits values
- VUID-VkQueryPoolCreateInfo-queryType-09534
  If `queryType` is `VK_QUERY_TYPE_PIPELINE_STATISTICS`, `pipelineStatistics` **must** not be zero
- VUID-VkQueryPoolCreateInfo-queryType-03222
  If `queryType` is `VK_QUERY_TYPE_PERFORMANCE_QUERY_KHR`, the `pNext` chain **must** include a VkQueryPoolPerformanceCreateInfoKHR structure
- VUID-VkQueryPoolCreateInfo-queryCount-02763
  `queryCount` **must** be greater than 0
- VUID-VkQueryPoolCreateInfo-queryType-07133
  If `queryType` is `VK_QUERY_TYPE_VIDEO_ENCODE_FEEDBACK_KHR`, then the `pNext` chain **must** include a VkVideoProfileInfoKHR structure with `videoCodecOperation` specifying an encode operation
- VUID-VkQueryPoolCreateInfo-queryType-07906
  If `queryType` is `VK_QUERY_TYPE_VIDEO_ENCODE_FEEDBACK_KHR`, then the `pNext` chain **must** include a VkQueryPoolVideoEncodeFeedbackCreateInfoKHR structure
- VUID-VkQueryPoolCreateInfo-queryType-07907
  If `queryType` is `VK_QUERY_TYPE_VIDEO_ENCODE_FEEDBACK_KHR`, and the `pNext` chain includes a VkVideoProfileInfoKHR structure and a VkQueryPoolVideoEncodeFeedbackCreateInfoKHR structure, then VkQueryPoolVideoEncodeFeedbackCreateInfoKHR::`encodeFeedbackFlags` **must** not contain any bits that are not set in VkVideoEncodeCapabilitiesKHR::`supportedEncodeFeedbackFlags`, as returned by vkGetPhysicalDeviceVideoCapabilitiesKHR for the video profile described by VkVideoProfileInfoKHR and its `pNext` chain

### Valid Usage (Implicit)

- VUID-VkQueryPoolCreateInfo-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_QUERY_POOL_CREATE_INFO`
- VUID-VkQueryPoolCreateInfo-pNext-pNext
  Each `pNext` member of any structure (including this one) in the `pNext` chain **must** be either `NULL` or a pointer to a valid instance of VkQueryPoolPerformanceCreateInfoKHR, VkQueryPoolPerformanceQueryCreateInfoINTEL, VkQueryPoolVideoEncodeFeedbackCreateInfoKHR, VkVideoDecodeAV1ProfileInfoKHR, VkVideoDecodeH264ProfileInfoKHR, VkVideoDecodeH265ProfileInfoKHR, VkVideoDecodeUsageInfoKHR, VkVideoEncodeH264ProfileInfoKHR, VkVideoEncodeH265ProfileInfoKHR, VkVideoEncodeUsageInfoKHR, or VkVideoProfileInfoKHR
- VUID-VkQueryPoolCreateInfo-sType-unique
  The `sType` value of each struct in the `pNext` chain **must** be unique
- VUID-VkQueryPoolCreateInfo-flags-zerobitmask
  `flags` **must** be `0`
- VUID-VkQueryPoolCreateInfo-queryType-parameter
  `queryType` **must** be a valid VkQueryType value

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkQueryPoolCreateFlags;
```

`VkQueryPoolCreateFlags` is a bitmask type for setting a mask, but is currently reserved for future use.

The `VkQueryPoolPerformanceCreateInfoKHR` structure is defined as:

```
// Provided by VK_KHR_performance_query
typedef struct VkQueryPoolPerformanceCreateInfoKHR {
    VkStructureType    sType;
    const void*        pNext;
    uint32_t           queueFamilyIndex;
    uint32_t           counterIndexCount;
    const uint32_t*    pCounterIndices;
} VkQueryPoolPerformanceCreateInfoKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `queueFamilyIndex` is the queue family index to create this performance query pool for.
- `counterIndexCount` is the length of the `pCounterIndices` array.
- `pCounterIndices` is a pointer to an array of indices into the vkEnumeratePhysicalDeviceQueueFamilyPerformanceQueryCountersKHR::`pCounters` to enable

in this performance query pool.

### Valid Usage

- VUID-VkQueryPoolPerformanceCreateInfoKHR-queueFamilyIndex-03236
  queueFamilyIndex **must** be a valid queue family index of the device
- VUID-VkQueryPoolPerformanceCreateInfoKHR-performanceCounterQueryPools-03237
  The performanceCounterQueryPools feature **must** be enabled
- VUID-VkQueryPoolPerformanceCreateInfoKHR-pCounterIndices-03321
  Each element of pCounterIndices **must** be in the range of counters reported by vkEnumeratePhysicalDeviceQueueFamilyPerformanceQueryCountersKHR for the queue family specified in queueFamilyIndex

### Valid Usage (Implicit)

- VUID-VkQueryPoolPerformanceCreateInfoKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_QUERY_POOL_PERFORMANCE_CREATE_INFO_KHR
- VUID-VkQueryPoolPerformanceCreateInfoKHR-pCounterIndices-parameter
  pCounterIndices **must** be a valid pointer to an array of counterIndexCount uint32_t values
- VUID-VkQueryPoolPerformanceCreateInfoKHR-counterIndexCount-arraylength
  counterIndexCount **must** be greater than 0

To query the number of passes required to query a performance query pool on a physical device, call:

```
// Provided by VK_KHR_performance_query
void vkGetPhysicalDeviceQueueFamilyPerformanceQueryPassesKHR(
    VkPhysicalDevice                            physicalDevice,
    const VkQueryPoolPerformanceCreateInfoKHR*  pPerformanceQueryCreateInfo,
    uint32_t*                                   pNumPasses);
```

- physicalDevice is the handle to the physical device whose queue family performance query counter properties will be queried.
- pPerformanceQueryCreateInfo is a pointer to a VkQueryPoolPerformanceCreateInfoKHR of the performance query that is to be created.
- pNumPasses is a pointer to an integer related to the number of passes required to query the performance query pool, as described below.

The pPerformanceQueryCreateInfo member VkQueryPoolPerformanceCreateInfoKHR::queueFamilyIndex **must** be a queue family of physicalDevice. The number of passes required to capture the counters specified in the pPerformanceQueryCreateInfo member VkQueryPoolPerformanceCreateInfoKHR::pCounters is returned in pNumPasses.

### Valid Usage (Implicit)

- VUID-vkGetPhysicalDeviceQueueFamilyPerformanceQueryPassesKHR-physicalDevice-parameter
  `physicalDevice` **must** be a valid VkPhysicalDevice handle
- VUID-vkGetPhysicalDeviceQueueFamilyPerformanceQueryPassesKHR-pPerformanceQueryCreateInfo-parameter
  `pPerformanceQueryCreateInfo` **must** be a valid pointer to a valid VkQueryPoolPerformanceCreateInfoKHR structure
- VUID-vkGetPhysicalDeviceQueueFamilyPerformanceQueryPassesKHR-pNumPasses-parameter
  `pNumPasses` **must** be a valid pointer to a `uint32_t` value

To destroy a query pool, call:

```
// Provided by VK_VERSION_1_0
void vkDestroyQueryPool(
    VkDevice                                    device,
    VkQueryPool                                 queryPool,
    const VkAllocationCallbacks*                pAllocator);
```

- `device` is the logical device that destroys the query pool.
- `queryPool` is the query pool to destroy.
- `pAllocator` controls host memory allocation as described in the Memory Allocation chapter.

### Valid Usage

- VUID-vkDestroyQueryPool-queryPool-00793
  All submitted commands that refer to `queryPool` **must** have completed execution
- VUID-vkDestroyQueryPool-queryPool-00794
  If `VkAllocationCallbacks` were provided when `queryPool` was created, a compatible set of callbacks **must** be provided here
- VUID-vkDestroyQueryPool-queryPool-00795
  If no `VkAllocationCallbacks` were provided when `queryPool` was created, `pAllocator` **must** be `NULL`

**NOTE** Applications **can** verify that `queryPool` **can** be destroyed by checking that `vkGetQueryPoolResults`() without the `VK_QUERY_RESULT_PARTIAL_BIT` flag returns `VK_SUCCESS` for all queries that are used in command buffers submitted for execution.

## Valid Usage (Implicit)

- VUID-vkDestroyQueryPool-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkDestroyQueryPool-queryPool-parameter
  If `queryPool` is not VK_NULL_HANDLE, `queryPool` **must** be a valid VkQueryPool handle
- VUID-vkDestroyQueryPool-pAllocator-parameter
  If `pAllocator` is not `NULL`, `pAllocator` **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkDestroyQueryPool-queryPool-parent
  If `queryPool` is a valid handle, it **must** have been created, allocated, or retrieved from `device`

## Host Synchronization

- Host access to `queryPool` **must** be externally synchronized

Possible values of VkQueryPoolCreateInfo::`queryType`, specifying the type of queries managed by the pool, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkQueryType {
    VK_QUERY_TYPE_OCCLUSION = 0,
    VK_QUERY_TYPE_PIPELINE_STATISTICS = 1,
    VK_QUERY_TYPE_TIMESTAMP = 2,
  // Provided by VK_KHR_video_queue
    VK_QUERY_TYPE_RESULT_STATUS_ONLY_KHR = 1000023000,
  // Provided by VK_EXT_transform_feedback
    VK_QUERY_TYPE_TRANSFORM_FEEDBACK_STREAM_EXT = 1000028004,
  // Provided by VK_KHR_performance_query
    VK_QUERY_TYPE_PERFORMANCE_QUERY_KHR = 1000116000,
  // Provided by VK_KHR_acceleration_structure
    VK_QUERY_TYPE_ACCELERATION_STRUCTURE_COMPACTED_SIZE_KHR = 1000150000,
  // Provided by VK_KHR_acceleration_structure
    VK_QUERY_TYPE_ACCELERATION_STRUCTURE_SERIALIZATION_SIZE_KHR = 1000150001,
  // Provided by VK_NV_ray_tracing
    VK_QUERY_TYPE_ACCELERATION_STRUCTURE_COMPACTED_SIZE_NV = 1000165000,
  // Provided by VK_INTEL_performance_query
    VK_QUERY_TYPE_PERFORMANCE_QUERY_INTEL = 1000210000,
  // Provided by VK_KHR_video_encode_queue
    VK_QUERY_TYPE_VIDEO_ENCODE_FEEDBACK_KHR = 1000299000,
  // Provided by VK_EXT_mesh_shader
    VK_QUERY_TYPE_MESH_PRIMITIVES_GENERATED_EXT = 1000328000,
  // Provided by VK_EXT_primitives_generated_query
    VK_QUERY_TYPE_PRIMITIVES_GENERATED_EXT = 1000382000,
  // Provided by VK_KHR_ray_tracing_maintenance1
    VK_QUERY_TYPE_ACCELERATION_STRUCTURE_SERIALIZATION_BOTTOM_LEVEL_POINTERS_KHR =
1000386000,
  // Provided by VK_KHR_ray_tracing_maintenance1
    VK_QUERY_TYPE_ACCELERATION_STRUCTURE_SIZE_KHR = 1000386001,
  // Provided by VK_EXT_opacity_micromap
    VK_QUERY_TYPE_MICROMAP_SERIALIZATION_SIZE_EXT = 1000396000,
  // Provided by VK_EXT_opacity_micromap
    VK_QUERY_TYPE_MICROMAP_COMPACTED_SIZE_EXT = 1000396001,
} VkQueryType;
```

- VK_QUERY_TYPE_OCCLUSION specifies an occlusion query.
- VK_QUERY_TYPE_PIPELINE_STATISTICS specifies a pipeline statistics query.
- VK_QUERY_TYPE_TIMESTAMP specifies a timestamp query.
- VK_QUERY_TYPE_PERFORMANCE_QUERY_KHR specifies a performance query.
- VK_QUERY_TYPE_TRANSFORM_FEEDBACK_STREAM_EXT specifies a transform feedback query.
- VK_QUERY_TYPE_PRIMITIVES_GENERATED_EXT specifies a primitives generated query.
- VK_QUERY_TYPE_ACCELERATION_STRUCTURE_COMPACTED_SIZE_KHR specifies a acceleration structure size query for use with vkCmdWriteAccelerationStructuresPropertiesKHR or vkWriteAccelerationStructuresPropertiesKHR.
- VK_QUERY_TYPE_ACCELERATION_STRUCTURE_SERIALIZATION_SIZE_KHR specifies a serialization

acceleration structure size query.

- VK_QUERY_TYPE_ACCELERATION_STRUCTURE_SIZE_KHR specifies an acceleration structure size query for use with vkCmdWriteAccelerationStructuresPropertiesKHR or vkWriteAccelerationStructuresPropertiesKHR.
- VK_QUERY_TYPE_ACCELERATION_STRUCTURE_SERIALIZATION_BOTTOM_LEVEL_POINTERS_KHR specifies a serialization acceleration structure pointer count query.
- VK_QUERY_TYPE_ACCELERATION_STRUCTURE_COMPACTED_SIZE_NV specifies an acceleration structure size query for use with vkCmdWriteAccelerationStructuresPropertiesNV.
- VK_QUERY_TYPE_PERFORMANCE_QUERY_INTEL specifies a Intel performance query.
- VK_QUERY_TYPE_RESULT_STATUS_ONLY_KHR specifies a result status query.
- VK_QUERY_TYPE_VIDEO_ENCODE_FEEDBACK_KHR specifies a video encode feedback query.
- VK_QUERY_TYPE_MESH_PRIMITIVES_GENERATED_EXT specifies a generated mesh primitives query.

## 18.2. Query Operation

The operation of queries is controlled by the commands vkCmdBeginQuery, vkCmdEndQuery, vkCmdBeginQueryIndexedEXT, vkCmdEndQueryIndexedEXT, vkCmdResetQueryPool, vkCmdCopyQueryPoolResults, vkCmdWriteTimestamp2KHR, and vkCmdWriteTimestamp.

In order for a VkCommandBuffer to record query management commands, the queue family for which its VkCommandPool was created **must** support the appropriate type of operations (graphics, compute) suitable for the query type of a given query pool.

Each query in a query pool has a status that is either *unavailable* or *available*, and also has state to store the numerical results of a query operation of the type requested when the query pool was created. Resetting a query via vkCmdResetQueryPool or vkResetQueryPoolEXT sets the status to unavailable and makes the numerical results undefined. A query is made available by the operation of vkCmdEndQuery, vkCmdEndQueryIndexedEXT, vkCmdWriteTimestamp2KHR, or vkCmdWriteTimestamp. Both the availability status and numerical results **can** be retrieved by calling either vkGetQueryPoolResults or vkCmdCopyQueryPoolResults.

After query pool creation, each query is in an uninitialized state and **must** be reset before it is used. Queries **must** also be reset between uses.

If a logical device includes multiple physical devices, then each command that writes a query **must** execute on a single physical device, and any call to vkCmdBeginQuery **must** execute the corresponding vkCmdEndQuery command on the same physical device.

To reset a range of queries in a query pool on a queue, call:

```
// Provided by VK_VERSION_1_0
void vkCmdResetQueryPool(
    VkCommandBuffer                             commandBuffer,
    VkQueryPool                                 queryPool,
    uint32_t                                    firstQuery,
    uint32_t                                    queryCount);
```

- `commandBuffer` is the command buffer into which this command will be recorded.
- `queryPool` is the handle of the query pool managing the queries being reset.
- `firstQuery` is the initial query index to reset.
- `queryCount` is the number of queries to reset.

When executed on a queue, this command sets the status of query indices [`firstQuery`, `firstQuery` + `queryCount` - 1] to unavailable.

This command defines an execution dependency between other query commands that reference the same query.

The first synchronization scope includes all commands which reference the queries in `queryPool` indicated by `firstQuery` and `queryCount` that occur earlier in submission order.

The second synchronization scope includes all commands which reference the queries in `queryPool` indicated by `firstQuery` and `queryCount` that occur later in submission order.

The operation of this command happens after the first scope and happens before the second scope.

If the `queryType` used to create `queryPool` was `VK_QUERY_TYPE_PERFORMANCE_QUERY_KHR`, this command sets the status of query indices [`firstQuery`, `firstQuery` + `queryCount` - 1] to unavailable for each pass of `queryPool`, as indicated by a call to vkGetPhysicalDeviceQueueFamilyPerformanceQueryPassesKHR.

**NOTE** Because `vkCmdResetQueryPool` resets all the passes of the indicated queries, applications must not record a `vkCmdResetQueryPool` command for a `queryPool` created with `VK_QUERY_TYPE_PERFORMANCE_QUERY_KHR` in a command buffer that needs to be submitted multiple times as indicated by a call to vkGetPhysicalDeviceQueueFamilyPerformanceQueryPassesKHR. Otherwise applications will never be able to complete the recorded queries.

### Valid Usage

- VUID-vkCmdResetQueryPool-firstQuery-09436
  `firstQuery` **must** be less than the number of queries in `queryPool`
- VUID-vkCmdResetQueryPool-firstQuery-09437
  The sum of `firstQuery` and `queryCount` **must** be less than or equal to the number of queries in `queryPool`

- VUID-vkCmdResetQueryPool-None-02841
  All queries used by the command **must** not be active
- VUID-vkCmdResetQueryPool-firstQuery-02862
  If `queryPool` was created with `VK_QUERY_TYPE_PERFORMANCE_QUERY_KHR`, this command **must** not be recorded in a command buffer that, either directly or through secondary command buffers, also contains begin commands for a query from the set of queries [`firstQuery`, `firstQuery` + `queryCount` - 1]

## Valid Usage (Implicit)

- VUID-vkCmdResetQueryPool-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdResetQueryPool-queryPool-parameter
  `queryPool` **must** be a valid VkQueryPool handle
- VUID-vkCmdResetQueryPool-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdResetQueryPool-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics, compute, decode, encode, or optical flow operations
- VUID-vkCmdResetQueryPool-renderpass
  This command **must** only be called outside of a render pass instance
- VUID-vkCmdResetQueryPool-videocoding
  This command **must** only be called outside of a video coding scope
- VUID-vkCmdResetQueryPool-commonparent
  Both of `commandBuffer`, and `queryPool` **must** have been created, allocated, or retrieved from the same VkDevice

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

### Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Outside | Outside | Graphics<br>Compute<br>Decode<br>Encode<br>Opticalflow | Action |

To reset a range of queries in a query pool on the host, call:

```
// Provided by VK_EXT_host_query_reset
void vkResetQueryPoolEXT(
    VkDevice                                    device,
    VkQueryPool                                 queryPool,
    uint32_t                                    firstQuery,
    uint32_t                                    queryCount);
```

- `device` is the logical device that owns the query pool.
- `queryPool` is the handle of the query pool managing the queries being reset.
- `firstQuery` is the initial query index to reset.
- `queryCount` is the number of queries to reset.

This command sets the status of query indices [`firstQuery`, `firstQuery` + `queryCount` - 1] to unavailable.

If `queryPool` is `VK_QUERY_TYPE_PERFORMANCE_QUERY_KHR` this command sets the status of query indices [`firstQuery`, `firstQuery` + `queryCount` - 1] to unavailable for each pass.

### Valid Usage

- VUID-vkResetQueryPool-firstQuery-09436
  `firstQuery` **must** be less than the number of queries in `queryPool`
- VUID-vkResetQueryPool-firstQuery-09437
  The sum of `firstQuery` and `queryCount` **must** be less than or equal to the number of queries in `queryPool`
- VUID-vkResetQueryPool-None-02665
  The `hostQueryReset` feature **must** be enabled
- VUID-vkResetQueryPool-firstQuery-02741
  Submitted commands that refer to the range specified by `firstQuery` and `queryCount` in `queryPool` **must** have completed execution

- VUID-vkResetQueryPool-firstQuery-02742
  The range of queries specified by `firstQuery` and `queryCount` in `queryPool` **must** not be in use by calls to vkGetQueryPoolResults or `vkResetQueryPoolEXT` in other threads

### Valid Usage (Implicit)

- VUID-vkResetQueryPool-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkResetQueryPool-queryPool-parameter
  `queryPool` **must** be a valid VkQueryPool handle
- VUID-vkResetQueryPool-queryPool-parent
  `queryPool` **must** have been created, allocated, or retrieved from `device`

Once queries are reset and ready for use, query commands **can** be issued to a command buffer. Occlusion queries and pipeline statistics queries count events - drawn samples and pipeline stage invocations, respectively - resulting from commands that are recorded between a vkCmdBeginQuery command and a vkCmdEndQuery command within a specified command buffer, effectively scoping a set of drawing and/or dispatching commands. Timestamp queries write timestamps to a query pool. Performance queries record performance counters to a query pool.

A query **must** begin and end in the same command buffer, although if it is a primary command buffer, and the `inheritedQueries` feature is enabled, it **can** execute secondary command buffers during the query operation. For a secondary command buffer to be executed while a query is active, it **must** set the `occlusionQueryEnable`, `queryFlags`, and/or `pipelineStatistics` members of VkCommandBufferInheritanceInfo to conservative values, as described in the Command Buffer Recording section. A query **must** either begin and end inside the same subpass of a render pass instance, or **must** both begin and end outside of a render pass instance (i.e. contain entire render pass instances).

If queries are used while executing a render pass instance that has multiview enabled, the query uses N consecutive query indices in the query pool (starting at `query`) where N is the number of bits set in the view mask in the subpass the query is used in. How the numerical results of the query are distributed among the queries is implementation-dependent. For example, some implementations **may** write each view's results to a distinct query, while other implementations **may** write the total result to the first query and write zero to the other queries. However, the sum of the results in all the queries **must** accurately reflect the total result of the query summed over all views. Applications **can** sum the results from all the queries to compute the total result.

Queries used with multiview rendering **must** not span subpasses, i.e. they **must** begin and end in the same subpass.

A query **must** either begin and end inside the same video coding scope, or **must** both begin and end outside of a video coding scope and **must** not contain entire video coding scopes.

To begin a query, call:

```
// Provided by VK_VERSION_1_0
void vkCmdBeginQuery(
    VkCommandBuffer                             commandBuffer,
    VkQueryPool                                 queryPool,
    uint32_t                                    query,
    VkQueryControlFlags                         flags);
```

- `commandBuffer` is the command buffer into which this command will be recorded.
- `queryPool` is the query pool that will manage the results of the query.
- `query` is the query index within the query pool that will contain the results.
- `flags` is a bitmask of VkQueryControlFlagBits specifying constraints on the types of queries that **can** be performed.

If the `queryType` of the pool is `VK_QUERY_TYPE_OCCLUSION` and `flags` contains `VK_QUERY_CONTROL_PRECISE_BIT`, an implementation **must** return a result that matches the actual number of samples passed. This is described in more detail in Occlusion Queries.

Calling `vkCmdBeginQuery` is equivalent to calling vkCmdBeginQueryIndexedEXT with the `index` parameter set to zero.

After beginning a query, that query is considered *active* within the command buffer it was called in until that same query is ended. Queries active in a primary command buffer when secondary command buffers are executed are considered active for those secondary command buffers.

Furthermore, if the query is started within a video coding scope, the following command buffer states are initialized for the query type:

- The *active_query_index* is set to the value specified by `query`.
- The *last activatable query index* is also set to the value specified by `query`.

Each video coding operation stores a result to the query corresponding to the current active query index, followed by incrementing the active query index. If the active query index gets incremented past the last activatable query index, issuing any further video coding operations results in undefined behavior.

| | |
|---|---|
| **NOTE** | In practice, this means that currently no more than a single video coding operation **must** be issued between a begin and end query pair. |

This command defines an execution dependency between other query commands that reference the same query.

The first synchronization scope includes all commands which reference the queries in `queryPool` indicated by `query` that occur earlier in submission order.

The second synchronization scope includes all commands which reference the queries in `queryPool` indicated by `query` that occur later in submission order.

The operation of this command happens after the first scope and happens before the second scope.

## Valid Usage

- VUID-vkCmdBeginQuery-None-00807
  All queries used by the command **must** be *unavailable*
- VUID-vkCmdBeginQuery-queryType-02804
  The `queryType` used to create `queryPool` **must** not be `VK_QUERY_TYPE_TIMESTAMP`
- VUID-vkCmdBeginQuery-queryType-04728
  The `queryType` used to create `queryPool` **must** not be `VK_QUERY_TYPE_ACCELERATION_STRUCTURE_COMPACTED_SIZE_KHR` or `VK_QUERY_TYPE_ACCELERATION_STRUCTURE_SERIALIZATION_SIZE_KHR`
- VUID-vkCmdBeginQuery-queryType-06741
  The `queryType` used to create `queryPool` **must** not be `VK_QUERY_TYPE_ACCELERATION_STRUCTURE_SIZE_KHR` or `VK_QUERY_TYPE_ACCELERATION_STRUCTURE_SERIALIZATION_BOTTOM_LEVEL_POINTERS_KHR`
- VUID-vkCmdBeginQuery-queryType-04729
  The `queryType` used to create `queryPool` **must** not be `VK_QUERY_TYPE_ACCELERATION_STRUCTURE_COMPACTED_SIZE_NV`
- VUID-vkCmdBeginQuery-queryType-00800
  If the `occlusionQueryPrecise` feature is not enabled, or the `queryType` used to create `queryPool` was not `VK_QUERY_TYPE_OCCLUSION`, `flags` **must** not contain `VK_QUERY_CONTROL_PRECISE_BIT`
- VUID-vkCmdBeginQuery-query-00802
  `query` **must** be less than the number of queries in `queryPool`
- VUID-vkCmdBeginQuery-queryType-00803
  If the `queryType` used to create `queryPool` was `VK_QUERY_TYPE_OCCLUSION`, the `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- VUID-vkCmdBeginQuery-queryType-00804
  If the `queryType` used to create `queryPool` was `VK_QUERY_TYPE_PIPELINE_STATISTICS` and any of the `pipelineStatistics` indicate graphics operations, the `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- VUID-vkCmdBeginQuery-queryType-00805
  If the `queryType` used to create `queryPool` was `VK_QUERY_TYPE_PIPELINE_STATISTICS` and any of the `pipelineStatistics` indicate compute operations, the `VkCommandPool` that `commandBuffer` was allocated from **must** support compute operations
- VUID-vkCmdBeginQuery-commandBuffer-01885
  `commandBuffer` **must** not be a protected command buffer
- VUID-vkCmdBeginQuery-query-00808
  If called within a render pass instance, the sum of `query` and the number of bits set in the current subpass's view mask **must** be less than or equal to the number of queries in `queryPool`
- VUID-vkCmdBeginQuery-queryType-07126

If the queryType used to create queryPool was VK_QUERY_TYPE_RESULT_STATUS_ONLY_KHR, then the VkCommandPool that commandBuffer was allocated from **must** have been created with a queue family index that supports result status queries, as indicated by VkQueueFamilyQueryResultStatusPropertiesKHR::queryResultStatusSupport

- VUID-vkCmdBeginQuery-None-07127
  If there is a bound video session, then there **must** be no active queries
- VUID-vkCmdBeginQuery-None-08370
  If there is a bound video session, then it **must** not have been created with VK_VIDEO_SESSION_CREATE_INLINE_QUERIES_BIT_KHR
- VUID-vkCmdBeginQuery-queryType-07128
  If the queryType used to create queryPool was VK_QUERY_TYPE_RESULT_STATUS_ONLY_KHR and there is a bound video session, then queryPool **must** have been created with a VkVideoProfileInfoKHR structure included in the pNext chain of VkQueryPoolCreateInfo identical to the one specified in VkVideoSessionCreateInfoKHR::pVideoProfile the bound video session was created with
- VUID-vkCmdBeginQuery-queryType-07129
  If the queryType used to create queryPool was VK_QUERY_TYPE_VIDEO_ENCODE_FEEDBACK_KHR, then there **must** be a bound video session
- VUID-vkCmdBeginQuery-queryType-07130
  If the queryType used to create queryPool was VK_QUERY_TYPE_VIDEO_ENCODE_FEEDBACK_KHR and there is a bound video session, then queryPool **must** have been created with a VkVideoProfileInfoKHR structure included in the pNext chain of VkQueryPoolCreateInfo identical to the one specified in VkVideoSessionCreateInfoKHR::pVideoProfile the bound video session was created with
- VUID-vkCmdBeginQuery-queryType-07131
  If the queryType used to create queryPool was not VK_QUERY_TYPE_RESULT_STATUS_ONLY_KHR or VK_QUERY_TYPE_VIDEO_ENCODE_FEEDBACK_KHR, then there **must** be no bound video session
- VUID-vkCmdBeginQuery-queryPool-01922
  queryPool **must** have been created with a queryType that differs from that of any queries that are active within commandBuffer
- VUID-vkCmdBeginQuery-queryType-07070
  If the queryType used to create queryPool was VK_QUERY_TYPE_MESH_PRIMITIVES_GENERATED_EXT the VkCommandPool that commandBuffer was allocated from **must** support graphics operations
- VUID-vkCmdBeginQuery-queryType-02327
  If the queryType used to create queryPool was VK_QUERY_TYPE_TRANSFORM_FEEDBACK_STREAM_EXT the VkCommandPool that commandBuffer was allocated from **must** support graphics operations
- VUID-vkCmdBeginQuery-queryType-02328
  If the queryType used to create queryPool was VK_QUERY_TYPE_TRANSFORM_FEEDBACK_STREAM_EXT then VkPhysicalDeviceTransformFeedbackPropertiesEXT::transformFeedbackQueries **must** be supported
- VUID-vkCmdBeginQuery-queryType-06687

If the queryType used to create queryPool was VK_QUERY_TYPE_PRIMITIVES_GENERATED_EXT the VkCommandPool that commandBuffer was allocated from **must** support graphics operations

- VUID-vkCmdBeginQuery-queryType-06688
  If the queryType used to create queryPool was VK_QUERY_TYPE_PRIMITIVES_GENERATED_EXT then primitivesGeneratedQuery **must** be enabled

- VUID-vkCmdBeginQuery-queryPool-07289
  If queryPool was created with a queryType of VK_QUERY_TYPE_PERFORMANCE_QUERY_KHR, then the VkQueryPoolPerformanceCreateInfoKHR::queueFamilyIndex queryPool was created with **must** equal the queue family index of the VkCommandPool that commandBuffer was allocated from

- VUID-vkCmdBeginQuery-queryPool-03223
  If queryPool was created with a queryType of VK_QUERY_TYPE_PERFORMANCE_QUERY_KHR, the profiling lock **must** have been held before vkBeginCommandBuffer was called on commandBuffer

- VUID-vkCmdBeginQuery-queryPool-03224
  If queryPool was created with a queryType of VK_QUERY_TYPE_PERFORMANCE_QUERY_KHR and one of the counters used to create queryPool was VK_PERFORMANCE_COUNTER_SCOPE_COMMAND_BUFFER_KHR, the query begin **must** be the first recorded command in commandBuffer

- VUID-vkCmdBeginQuery-queryPool-03225
  If queryPool was created with a queryType of VK_QUERY_TYPE_PERFORMANCE_QUERY_KHR and one of the counters used to create queryPool was VK_PERFORMANCE_COUNTER_SCOPE_RENDER_PASS_KHR, the begin command **must** not be recorded within a render pass instance

- VUID-vkCmdBeginQuery-queryPool-03226
  If queryPool was created with a queryType of VK_QUERY_TYPE_PERFORMANCE_QUERY_KHR and another query pool with a queryType VK_QUERY_TYPE_PERFORMANCE_QUERY_KHR has been used within commandBuffer, its parent primary command buffer or secondary command buffer recorded within the same parent primary command buffer as commandBuffer, the performanceCounterMultipleQueryPools feature **must** be enabled

- VUID-vkCmdBeginQuery-None-02863
  If queryPool was created with a queryType of VK_QUERY_TYPE_PERFORMANCE_QUERY_KHR, this command **must** not be recorded in a command buffer that, either directly or through secondary command buffers, also contains a vkCmdResetQueryPool command affecting the same query

## Valid Usage (Implicit)

- VUID-vkCmdBeginQuery-commandBuffer-parameter
  commandBuffer **must** be a valid VkCommandBuffer handle

- VUID-vkCmdBeginQuery-queryPool-parameter
  queryPool **must** be a valid VkQueryPool handle

- VUID-vkCmdBeginQuery-flags-parameter

flags **must** be a valid combination of VkQueryControlFlagBits values

- VUID-vkCmdBeginQuery-commandBuffer-recording
  commandBuffer **must** be in the recording state
- VUID-vkCmdBeginQuery-commandBuffer-cmdpool
  The VkCommandPool that commandBuffer was allocated from **must** support graphics, compute, decode, or encode operations
- VUID-vkCmdBeginQuery-commonparent
  Both of commandBuffer, and queryPool **must** have been created, allocated, or retrieved from the same VkDevice

### Host Synchronization

- Host access to commandBuffer **must** be externally synchronized
- Host access to the VkCommandPool that commandBuffer was allocated from **must** be externally synchronized

### Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Both | Graphics<br>Compute<br>Decode<br>Encode | Action<br>State |

To begin an indexed query, call:

```
// Provided by VK_EXT_transform_feedback
void vkCmdBeginQueryIndexedEXT(
    VkCommandBuffer                             commandBuffer,
    VkQueryPool                                 queryPool,
    uint32_t                                    query,
    VkQueryControlFlags                         flags,
    uint32_t                                    index);
```

- commandBuffer is the command buffer into which this command will be recorded.
- queryPool is the query pool that will manage the results of the query.
- query is the query index within the query pool that will contain the results.
- flags is a bitmask of VkQueryControlFlagBits specifying constraints on the types of queries that **can** be performed.
- index is the query type specific index. When the query type is

VK_QUERY_TYPE_TRANSFORM_FEEDBACK_STREAM_EXT or VK_QUERY_TYPE_PRIMITIVES_GENERATED_EXT, the index represents the vertex stream.

The vkCmdBeginQueryIndexedEXT command operates the same as the vkCmdBeginQuery command, except that it also accepts a query type specific index parameter.

This command defines an execution dependency between other query commands that reference the same query index.

The first synchronization scope includes all commands which reference the queries in queryPool indicated by query and index that occur earlier in submission order.

The second synchronization scope includes all commands which reference the queries in queryPool indicated by query and index that occur later in submission order.

The operation of this command happens after the first scope and happens before the second scope.

## Valid Usage

- VUID-vkCmdBeginQueryIndexedEXT-None-00807
  All queries used by the command **must** be *unavailable*
- VUID-vkCmdBeginQueryIndexedEXT-queryType-02804
  The queryType used to create queryPool **must** not be VK_QUERY_TYPE_TIMESTAMP
- VUID-vkCmdBeginQueryIndexedEXT-queryType-04728
  The queryType used to create queryPool **must** not be VK_QUERY_TYPE_ACCELERATION_STRUCTURE_COMPACTED_SIZE_KHR or VK_QUERY_TYPE_ACCELERATION_STRUCTURE_SERIALIZATION_SIZE_KHR
- VUID-vkCmdBeginQueryIndexedEXT-queryType-06741
  The queryType used to create queryPool **must** not be VK_QUERY_TYPE_ACCELERATION_STRUCTURE_SIZE_KHR or VK_QUERY_TYPE_ACCELERATION_STRUCTURE_SERIALIZATION_BOTTOM_LEVEL_POINTERS_KHR
- VUID-vkCmdBeginQueryIndexedEXT-queryType-04729
  The queryType used to create queryPool **must** not be VK_QUERY_TYPE_ACCELERATION_STRUCTURE_COMPACTED_SIZE_NV
- VUID-vkCmdBeginQueryIndexedEXT-queryType-00800
  If the occlusionQueryPrecise feature is not enabled, or the queryType used to create queryPool was not VK_QUERY_TYPE_OCCLUSION, flags **must** not contain VK_QUERY_CONTROL_PRECISE_BIT
- VUID-vkCmdBeginQueryIndexedEXT-query-00802
  query **must** be less than the number of queries in queryPool
- VUID-vkCmdBeginQueryIndexedEXT-queryType-00803
  If the queryType used to create queryPool was VK_QUERY_TYPE_OCCLUSION, the VkCommandPool that commandBuffer was allocated from **must** support graphics operations
- VUID-vkCmdBeginQueryIndexedEXT-queryType-00804
  If the queryType used to create queryPool was VK_QUERY_TYPE_PIPELINE_STATISTICS and any

of the pipelineStatistics indicate graphics operations, the VkCommandPool that commandBuffer was allocated from **must** support graphics operations

- VUID-vkCmdBeginQueryIndexedEXT-queryType-00805
  If the queryType used to create queryPool was VK_QUERY_TYPE_PIPELINE_STATISTICS and any of the pipelineStatistics indicate compute operations, the VkCommandPool that commandBuffer was allocated from **must** support compute operations
- VUID-vkCmdBeginQueryIndexedEXT-commandBuffer-01885
  commandBuffer **must** not be a protected command buffer
- VUID-vkCmdBeginQueryIndexedEXT-query-00808
  If called within a render pass instance, the sum of query and the number of bits set in the current subpass's view mask **must** be less than or equal to the number of queries in queryPool
- VUID-vkCmdBeginQueryIndexedEXT-queryType-07126
  If the queryType used to create queryPool was VK_QUERY_TYPE_RESULT_STATUS_ONLY_KHR, then the VkCommandPool that commandBuffer was allocated from **must** have been created with a queue family index that supports result status queries, as indicated by VkQueueFamilyQueryResultStatusPropertiesKHR::queryResultStatusSupport
- VUID-vkCmdBeginQueryIndexedEXT-None-07127
  If there is a bound video session, then there **must** be no active queries
- VUID-vkCmdBeginQueryIndexedEXT-None-08370
  If there is a bound video session, then it **must** not have been created with VK_VIDEO_SESSION_CREATE_INLINE_QUERIES_BIT_KHR
- VUID-vkCmdBeginQueryIndexedEXT-queryType-07128
  If the queryType used to create queryPool was VK_QUERY_TYPE_RESULT_STATUS_ONLY_KHR and there is a bound video session, then queryPool **must** have been created with a VkVideoProfileInfoKHR structure included in the pNext chain of VkQueryPoolCreateInfo identical to the one specified in VkVideoSessionCreateInfoKHR::pVideoProfile the bound video session was created with
- VUID-vkCmdBeginQueryIndexedEXT-queryType-07129
  If the queryType used to create queryPool was VK_QUERY_TYPE_VIDEO_ENCODE_FEEDBACK_KHR, then there **must** be a bound video session
- VUID-vkCmdBeginQueryIndexedEXT-queryType-07130
  If the queryType used to create queryPool was VK_QUERY_TYPE_VIDEO_ENCODE_FEEDBACK_KHR and there is a bound video session, then queryPool **must** have been created with a VkVideoProfileInfoKHR structure included in the pNext chain of VkQueryPoolCreateInfo identical to the one specified in VkVideoSessionCreateInfoKHR::pVideoProfile the bound video session was created with
- VUID-vkCmdBeginQueryIndexedEXT-queryType-07131
  If the queryType used to create queryPool was not VK_QUERY_TYPE_RESULT_STATUS_ONLY_KHR or VK_QUERY_TYPE_VIDEO_ENCODE_FEEDBACK_KHR, then there **must** be no bound video session
- VUID-vkCmdBeginQueryIndexedEXT-queryPool-04753
  If the queryPool was created with the same queryType as that of another active query within commandBuffer, then index **must** not match the index used for the active query

- VUID-vkCmdBeginQueryIndexedEXT-queryType-02338
  If the queryType used to create queryPool was VK_QUERY_TYPE_TRANSFORM_FEEDBACK_STREAM_EXT the VkCommandPool that commandBuffer was allocated from **must** support graphics operations
- VUID-vkCmdBeginQueryIndexedEXT-queryType-02339
  If the queryType used to create queryPool was VK_QUERY_TYPE_TRANSFORM_FEEDBACK_STREAM_EXT the index parameter **must** be less than VkPhysicalDeviceTransformFeedbackPropertiesEXT::maxTransformFeedbackStreams
- VUID-vkCmdBeginQueryIndexedEXT-queryType-06692
  If the queryType used to create queryPool was not VK_QUERY_TYPE_TRANSFORM_FEEDBACK_STREAM_EXT and not VK_QUERY_TYPE_PRIMITIVES_GENERATED_EXT, the index **must** be zero
- VUID-vkCmdBeginQueryIndexedEXT-queryType-06689
  If the queryType used to create queryPool was VK_QUERY_TYPE_PRIMITIVES_GENERATED_EXT the VkCommandPool that commandBuffer was allocated from **must** support graphics operations
- VUID-vkCmdBeginQueryIndexedEXT-queryType-06690
  If the queryType used to create queryPool was VK_QUERY_TYPE_PRIMITIVES_GENERATED_EXT the index parameter **must** be less than VkPhysicalDeviceTransformFeedbackPropertiesEXT::maxTransformFeedbackStreams
- VUID-vkCmdBeginQueryIndexedEXT-queryType-06691
  If the queryType used to create queryPool was VK_QUERY_TYPE_PRIMITIVES_GENERATED_EXT and the primitivesGeneratedQueryWithNonZeroStreams feature is not enabled, the index parameter **must** be zero
- VUID-vkCmdBeginQueryIndexedEXT-queryType-06693
  If the queryType used to create queryPool was VK_QUERY_TYPE_PRIMITIVES_GENERATED_EXT then primitivesGeneratedQuery **must** be enabled
- VUID-vkCmdBeginQueryIndexedEXT-queryType-02341
  If the queryType used to create queryPool was VK_QUERY_TYPE_TRANSFORM_FEEDBACK_STREAM_EXT then VkPhysicalDeviceTransformFeedbackPropertiesEXT::transformFeedbackQueries **must** be supported
- VUID-vkCmdBeginQueryIndexedEXT-queryType-07071
  The queryType used to create queryPool **must** not be VK_QUERY_TYPE_MESH_PRIMITIVES_GENERATED_EXT
- VUID-vkCmdBeginQueryIndexedEXT-queryPool-07289
  If queryPool was created with a queryType of VK_QUERY_TYPE_PERFORMANCE_QUERY_KHR, then the VkQueryPoolPerformanceCreateInfoKHR::queueFamilyIndex queryPool was created with **must** equal the queue family index of the VkCommandPool that commandBuffer was allocated from
- VUID-vkCmdBeginQueryIndexedEXT-queryPool-03223
  If queryPool was created with a queryType of VK_QUERY_TYPE_PERFORMANCE_QUERY_KHR, the profiling lock **must** have been held before vkBeginCommandBuffer was called on commandBuffer
- VUID-vkCmdBeginQueryIndexedEXT-queryPool-03224

If queryPool was created with a queryType of VK_QUERY_TYPE_PERFORMANCE_QUERY_KHR and one of the counters used to create queryPool was VK_PERFORMANCE_COUNTER_SCOPE_COMMAND_BUFFER_KHR, the query begin **must** be the first recorded command in commandBuffer

- VUID-vkCmdBeginQueryIndexedEXT-queryPool-03225
  If queryPool was created with a queryType of VK_QUERY_TYPE_PERFORMANCE_QUERY_KHR and one of the counters used to create queryPool was VK_PERFORMANCE_COUNTER_SCOPE_RENDER_PASS_KHR, the begin command **must** not be recorded within a render pass instance
- VUID-vkCmdBeginQueryIndexedEXT-queryPool-03226
  If queryPool was created with a queryType of VK_QUERY_TYPE_PERFORMANCE_QUERY_KHR and another query pool with a queryType VK_QUERY_TYPE_PERFORMANCE_QUERY_KHR has been used within commandBuffer, its parent primary command buffer or secondary command buffer recorded within the same parent primary command buffer as commandBuffer, the performanceCounterMultipleQueryPools feature **must** be enabled
- VUID-vkCmdBeginQueryIndexedEXT-None-02863
  If queryPool was created with a queryType of VK_QUERY_TYPE_PERFORMANCE_QUERY_KHR, this command **must** not be recorded in a command buffer that, either directly or through secondary command buffers, also contains a vkCmdResetQueryPool command affecting the same query

## Valid Usage (Implicit)

- VUID-vkCmdBeginQueryIndexedEXT-commandBuffer-parameter
  commandBuffer **must** be a valid VkCommandBuffer handle
- VUID-vkCmdBeginQueryIndexedEXT-queryPool-parameter
  queryPool **must** be a valid VkQueryPool handle
- VUID-vkCmdBeginQueryIndexedEXT-flags-parameter
  flags **must** be a valid combination of VkQueryControlFlagBits values
- VUID-vkCmdBeginQueryIndexedEXT-commandBuffer-recording
  commandBuffer **must** be in the recording state
- VUID-vkCmdBeginQueryIndexedEXT-commandBuffer-cmdpool
  The VkCommandPool that commandBuffer was allocated from **must** support graphics, compute, decode, or encode operations
- VUID-vkCmdBeginQueryIndexedEXT-videocoding
  This command **must** only be called outside of a video coding scope
- VUID-vkCmdBeginQueryIndexedEXT-commonparent
  Both of commandBuffer, and queryPool **must** have been created, allocated, or retrieved from the same VkDevice

### Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

### Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics<br>Compute<br>Decode<br>Encode | Action<br>State |

Bits which **can** be set in vkCmdBeginQuery::`flags`, specifying constraints on the types of queries that **can** be performed, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkQueryControlFlagBits {
    VK_QUERY_CONTROL_PRECISE_BIT = 0x00000001,
} VkQueryControlFlagBits;
```

- `VK_QUERY_CONTROL_PRECISE_BIT` specifies the precision of occlusion queries.

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkQueryControlFlags;
```

`VkQueryControlFlags` is a bitmask type for setting a mask of zero or more VkQueryControlFlagBits.

To end a query after the set of desired drawing or dispatching commands is executed, call:

```
// Provided by VK_VERSION_1_0
void vkCmdEndQuery(
    VkCommandBuffer                             commandBuffer,
    VkQueryPool                                 queryPool,
    uint32_t                                    query);
```

- `commandBuffer` is the command buffer into which this command will be recorded.
- `queryPool` is the query pool that is managing the results of the query.
- `query` is the query index within the query pool where the result is stored.

The command completes the query in queryPool identified by query, and marks it as available.

This command defines an execution dependency between other query commands that reference the same query.

The first synchronization scope includes all commands which reference the queries in queryPool indicated by query that occur earlier in submission order.

The second synchronization scope includes only the operation of this command.

Calling vkCmdEndQuery is equivalent to calling vkCmdEndQueryIndexedEXT with the index parameter set to zero.

## Valid Usage

- VUID-vkCmdEndQuery-None-01923
  All queries used by the command **must** be active
- VUID-vkCmdEndQuery-query-00810
  query **must** be less than the number of queries in queryPool
- VUID-vkCmdEndQuery-commandBuffer-01886
  commandBuffer **must** not be a protected command buffer
- VUID-vkCmdEndQuery-query-00812
  If vkCmdEndQuery is called within a render pass instance, the sum of query and the number of bits set in the current subpass's view mask **must** be less than or equal to the number of queries in queryPool
- VUID-vkCmdEndQuery-queryPool-03227
  If queryPool was created with a queryType of VK_QUERY_TYPE_PERFORMANCE_QUERY_KHR and one or more of the counters used to create queryPool was VK_PERFORMANCE_COUNTER_SCOPE_COMMAND_BUFFER_KHR, the vkCmdEndQuery **must** be the last recorded command in commandBuffer
- VUID-vkCmdEndQuery-queryPool-03228
  If queryPool was created with a queryType of VK_QUERY_TYPE_PERFORMANCE_QUERY_KHR and one or more of the counters used to create queryPool was VK_PERFORMANCE_COUNTER_SCOPE_RENDER_PASS_KHR, the vkCmdEndQuery **must** not be recorded within a render pass instance
- VUID-vkCmdEndQuery-None-07007
  If called within a subpass of a render pass instance, the corresponding vkCmdBeginQuery* command **must** have been called previously within the same subpass

## Valid Usage (Implicit)

- VUID-vkCmdEndQuery-commandBuffer-parameter
  commandBuffer **must** be a valid VkCommandBuffer handle
- VUID-vkCmdEndQuery-queryPool-parameter

queryPool **must** be a valid VkQueryPool handle

- VUID-vkCmdEndQuery-commandBuffer-recording
  commandBuffer **must** be in the recording state
- VUID-vkCmdEndQuery-commandBuffer-cmdpool
  The VkCommandPool that commandBuffer was allocated from **must** support graphics, compute, decode, or encode operations
- VUID-vkCmdEndQuery-commonparent
  Both of commandBuffer, and queryPool **must** have been created, allocated, or retrieved from the same VkDevice

### Host Synchronization

- Host access to commandBuffer **must** be externally synchronized
- Host access to the VkCommandPool that commandBuffer was allocated from **must** be externally synchronized

### Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Both | Graphics<br>Compute<br>Decode<br>Encode | Action<br>State |

To end an indexed query after the set of desired drawing or dispatching commands is recorded, call:

```
// Provided by VK_EXT_transform_feedback
void vkCmdEndQueryIndexedEXT(
    VkCommandBuffer                                 commandBuffer,
    VkQueryPool                                     queryPool,
    uint32_t                                        query,
    uint32_t                                        index);
```

- commandBuffer is the command buffer into which this command will be recorded.
- queryPool is the query pool that is managing the results of the query.
- query is the query index within the query pool where the result is stored.
- index is the query type specific index.

The command completes the query in queryPool identified by query and index, and marks it as

available.

The vkCmdEndQueryIndexedEXT command operates the same as the vkCmdEndQuery command, except that it also accepts a query type specific index parameter.

This command defines an execution dependency between other query commands that reference the same query index.

The first synchronization scope includes all commands which reference the queries in queryPool indicated by query that occur earlier in submission order.

The second synchronization scope includes only the operation of this command.

## Valid Usage

- VUID-vkCmdEndQueryIndexedEXT-None-02342
  All queries used by the command **must** be active
- VUID-vkCmdEndQueryIndexedEXT-query-02343
  query **must** be less than the number of queries in queryPool
- VUID-vkCmdEndQueryIndexedEXT-commandBuffer-02344
  commandBuffer **must** not be a protected command buffer
- VUID-vkCmdEndQueryIndexedEXT-query-02345
  If vkCmdEndQueryIndexedEXT is called within a render pass instance, the sum of query and the number of bits set in the current subpass's view mask **must** be less than or equal to the number of queries in queryPool
- VUID-vkCmdEndQueryIndexedEXT-queryType-06694
  If the queryType used to create queryPool was VK_QUERY_TYPE_TRANSFORM_FEEDBACK_STREAM_EXT or VK_QUERY_TYPE_PRIMITIVES_GENERATED_EXT, the index parameter **must** be less than VkPhysicalDeviceTransformFeedbackPropertiesEXT::maxTransformFeedbackStreams
- VUID-vkCmdEndQueryIndexedEXT-queryType-06695
  If the queryType used to create queryPool was not VK_QUERY_TYPE_TRANSFORM_FEEDBACK_STREAM_EXT and not VK_QUERY_TYPE_PRIMITIVES_GENERATED_EXT, the index **must** be zero
- VUID-vkCmdEndQueryIndexedEXT-queryType-06696
  If the queryType used to create queryPool was VK_QUERY_TYPE_TRANSFORM_FEEDBACK_STREAM_EXT or VK_QUERY_TYPE_PRIMITIVES_GENERATED_EXT, index **must** equal the index used to begin the query
- VUID-vkCmdEndQueryIndexedEXT-None-07007
  If called within a subpass of a render pass instance, the corresponding vkCmdBeginQuery* command **must** have been called previously within the same subpass

## Valid Usage (Implicit)

- VUID-vkCmdEndQueryIndexedEXT-commandBuffer-parameter

commandBuffer **must** be a valid VkCommandBuffer handle

- VUID-vkCmdEndQueryIndexedEXT-queryPool-parameter
  queryPool **must** be a valid VkQueryPool handle
- VUID-vkCmdEndQueryIndexedEXT-commandBuffer-recording
  commandBuffer **must** be in the recording state
- VUID-vkCmdEndQueryIndexedEXT-commandBuffer-cmdpool
  The VkCommandPool that commandBuffer was allocated from **must** support graphics, compute, decode, or encode operations
- VUID-vkCmdEndQueryIndexedEXT-videocoding
  This command **must** only be called outside of a video coding scope
- VUID-vkCmdEndQueryIndexedEXT-commonparent
  Both of commandBuffer, and queryPool **must** have been created, allocated, or retrieved from the same VkDevice

## Host Synchronization

- Host access to commandBuffer **must** be externally synchronized
- Host access to the VkCommandPool that commandBuffer was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics<br>Compute<br>Decode<br>Encode | Action<br>State |

An application **can** retrieve results either by requesting they be written into application-provided memory, or by requesting they be copied into a VkBuffer. In either case, the layout in memory is defined as follows:

- The first query's result is written starting at the first byte requested by the command, and each subsequent query's result begins stride bytes later.
- Occlusion queries, pipeline statistics queries, transform feedback queries, primitives generated queries, mesh shader queries, video encode feedback queries, and timestamp queries store results in a tightly packed array of unsigned integers, either 32- or 64-bits as requested by the command, storing the numerical results and, if requested, the availability status.
- Performance queries store results in a tightly packed array whose type is determined by the unit member of the corresponding VkPerformanceCounterKHR.

- If VK_QUERY_RESULT_WITH_AVAILABILITY_BIT is used, the final element of each query's result is an integer indicating whether the query's result is available, with any non-zero value indicating that it is available.
- If VK_QUERY_RESULT_WITH_STATUS_BIT_KHR is used, the final element of each query's result is an integer value indicating that status of the query result. Positive values indicate success, negative values indicate failure, and 0 indicates that the result is not yet available. Specific error codes are encoded in the VkQueryResultStatusKHR enumeration.
- Occlusion queries write one integer value - the number of samples passed. Pipeline statistics queries write one integer value for each bit that is enabled in the pipelineStatistics when the pool is created, and the statistics values are written in bit order starting from the least significant bit. Timestamp queries write one integer value. Performance queries write one VkPerformanceCounterResultKHR value for each VkPerformanceCounterKHR in the query. Transform feedback queries write two integers; the first integer is the number of primitives successfully written to the corresponding transform feedback buffer and the second is the number of primitives output to the vertex stream, regardless of whether they were successfully captured or not. In other words, if the transform feedback buffer was sized too small for the number of primitives output by the vertex stream, the first integer represents the number of primitives actually written and the second is the number that would have been written if all the transform feedback buffers associated with that vertex stream were large enough. Primitives generated queries write the number of primitives output to the vertex stream, regardless of whether transform feedback is active or not, or whether they were successfully captured by transform feedback or not. This is identical to the second integer of the transform feedback queries if transform feedback is active. Mesh shader queries write a single integer. Video encode feedback queries write one or more integer values for each bit that is enabled in VkQueryPoolVideoEncodeFeedbackCreateInfoKHR::encodeFeedbackFlags when the pool is created, and the feedback values are written in bit order starting from the least significant bit, as described here.
- If more than one query is retrieved and stride is not at least as large as the size of the array of values corresponding to a single query, the values written to memory are undefined.

To retrieve status and results for a set of queries, call:

```
// Provided by VK_VERSION_1_0
VkResult vkGetQueryPoolResults(
    VkDevice                                    device,
    VkQueryPool                                 queryPool,
    uint32_t                                    firstQuery,
    uint32_t                                    queryCount,
    size_t                                      dataSize,
    void*                                       pData,
    VkDeviceSize                                stride,
    VkQueryResultFlags                          flags);
```

- device is the logical device that owns the query pool.
- queryPool is the query pool managing the queries containing the desired results.
- firstQuery is the initial query index.

- queryCount is the number of queries to read.
- dataSize is the size in bytes of the buffer pointed to by pData.
- pData is a pointer to an application-allocated buffer where the results will be written
- stride is the stride in bytes between results for individual queries within pData.
- flags is a bitmask of VkQueryResultFlagBits specifying how and when results are returned.

Any results written for a query are written according to a layout dependent on the query type.

If no bits are set in flags, and all requested queries are in the available state, results are written as an array of 32-bit unsigned integer values. Behavior when not all queries are available is described below.

If VK_QUERY_RESULT_WITH_AVAILABILITY_BIT is set, results for all queries in queryPool identified by firstQuery and queryCount are copied to pData, along with an extra availability or status value written directly after the results of each query and interpreted as an unsigned integer. A value of zero indicates that the results are not yet available, otherwise the query is complete and results are available. The size of the availability or status values is 64 bits if VK_QUERY_RESULT_64_BIT is set in flags. Otherwise, it is 32 bits.

If VK_QUERY_RESULT_WITH_STATUS_BIT_KHR is set, results for all queries in queryPool identified by firstQuery and queryCount are copied to pData, along with an extra status value written directly after the results of each query and interpreted as a signed integer. A value of zero indicates that the results are not yet available. Positive values indicate that the operations within the query completed successfully, and the query results are valid. Negative values indicate that the operations within the query completed unsuccessfully.

VkQueryResultStatusKHR defines specific meaning for values returned here, though implementations are free to return other values.

If the status value written is negative, indicating that the operations within the query completed unsuccessfully, then all other results written by this command are undefined unless otherwise specified for any of the results of the used query type.

| | |
|---|---|
| **NOTE** | If VK_QUERY_RESULT_WITH_AVAILABILITY_BIT or VK_QUERY_RESULT_WITH_STATUS_BIT_KHR is set, the layout of data in the buffer is a *(result,availability)* or *(result,status)* pair for each query returned, and stride is the stride between each pair. |

Results for any available query written by this command are final and represent the final result of the query. If VK_QUERY_RESULT_PARTIAL_BIT is set, then for any query that is unavailable, an intermediate result between zero and the final result value is written for that query. Otherwise, any result written by this command is undefined.

If VK_QUERY_RESULT_64_BIT is set, results and, if returned, availability or status values for all queries are written as an array of 64-bit values. If the queryPool was created with VK_QUERY_TYPE_PERFORMANCE_QUERY_KHR, results for each query are written as an array of the type indicated by VkPerformanceCounterKHR::storage for the counter being queried. Otherwise, results and availability or status values are written as an array of 32-bit values. If an unsigned integer query's value overflows the result type, the value **may** either wrap or saturate. If the maintenance7

feature is enabled, for an unsigned integer query, the 32-bit result value **must** be equal to the 32 least significant bits of the equivalent 64-bit result value. If a signed integer query's value overflows the result type, the value is undefined. If a floating-point query's value is not representable as the result type, the value is undefined.

If `VK_QUERY_RESULT_WAIT_BIT` is set, this command defines an execution dependency with any earlier commands that writes one of the identified queries. The first synchronization scope includes all instances of vkCmdEndQuery, vkCmdEndQueryIndexedEXT, vkCmdWriteTimestamp2KHR, and vkCmdWriteTimestamp that reference any query in `queryPool` indicated by `firstQuery` and `queryCount`. The second synchronization scope includes the host operations of this command.

If `VK_QUERY_RESULT_WAIT_BIT` is not set, `vkGetQueryPoolResults` **may** return `VK_NOT_READY` if there are queries in the unavailable state.

**NOTE**

Applications **must** take care to ensure that use of the `VK_QUERY_RESULT_WAIT_BIT` bit has the desired effect.

For example, if a query has been used previously and a command buffer records the commands `vkCmdResetQueryPool`, `vkCmdBeginQuery`, and `vkCmdEndQuery` for that query, then the query will remain in the available state until `vkResetQueryPoolEXT` is called or the `vkCmdResetQueryPool` command executes on a queue. Applications **can** use fences or events to ensure that a query has already been reset before checking for its results or availability status. Otherwise, a stale value could be returned from a previous use of the query.

The above also applies when `VK_QUERY_RESULT_WAIT_BIT` is used in combination with `VK_QUERY_RESULT_WITH_AVAILABILITY_BIT`. In this case, the returned availability status **may** reflect the result of a previous use of the query unless `vkResetQueryPoolEXT` is called or the `vkCmdResetQueryPool` command has been executed since the last use of the query.

A similar situation can arise with the `VK_QUERY_RESULT_WITH_STATUS_BIT_KHR` flag.

**NOTE**

Applications **can** double-buffer query pool usage, with a pool per frame, and reset queries at the end of the frame in which they are read.

## Valid Usage

- VUID-vkGetQueryPoolResults-firstQuery-09436
  `firstQuery` **must** be less than the number of queries in `queryPool`
- VUID-vkGetQueryPoolResults-firstQuery-09437
  The sum of `firstQuery` and `queryCount` **must** be less than or equal to the number of queries in `queryPool`
- VUID-vkGetQueryPoolResults-queryCount-09438
  If `queryCount` is greater than 1, `stride` **must** not be zero
- VUID-vkGetQueryPoolResults-queryType-09439

If the queryType used to create queryPool was VK_QUERY_TYPE_TIMESTAMP, flags **must** not contain VK_QUERY_RESULT_PARTIAL_BIT

- VUID-vkGetQueryPoolResults-queryType-09440
  If the queryType used to create queryPool was VK_QUERY_TYPE_PERFORMANCE_QUERY_KHR, flags **must** not contain VK_QUERY_RESULT_WITH_AVAILABILITY_BIT, VK_QUERY_RESULT_WITH_STATUS_BIT_KHR, VK_QUERY_RESULT_PARTIAL_BIT, or VK_QUERY_RESULT_64_BIT
- VUID-vkGetQueryPoolResults-queryType-09441
  If the queryType used to create queryPool was VK_QUERY_TYPE_PERFORMANCE_QUERY_KHR, the queryPool **must** have been recorded once for each pass as retrieved via a call to vkGetPhysicalDeviceQueueFamilyPerformanceQueryPassesKHR
- VUID-vkGetQueryPoolResults-queryType-09442
  If the queryType used to create queryPool was VK_QUERY_TYPE_RESULT_STATUS_ONLY_KHR, then flags **must** include VK_QUERY_RESULT_WITH_STATUS_BIT_KHR
- VUID-vkGetQueryPoolResults-flags-09443
  If flags includes VK_QUERY_RESULT_WITH_STATUS_BIT_KHR, then it **must** not include VK_QUERY_RESULT_WITH_AVAILABILITY_BIT
- VUID-vkGetQueryPoolResults-None-09401
  All queries used by the command **must** not be uninitialized
- VUID-vkGetQueryPoolResults-flags-02828
  If VK_QUERY_RESULT_64_BIT is not set in flags and the queryType used to create queryPool was not VK_QUERY_TYPE_PERFORMANCE_QUERY_KHR, then pData and stride **must** be multiples of 4
- VUID-vkGetQueryPoolResults-flags-00815
  If VK_QUERY_RESULT_64_BIT is set in flags then pData and stride **must** be multiples of 8
- VUID-vkGetQueryPoolResults-stride-08993
  If VK_QUERY_RESULT_WITH_AVAILABILITY_BIT is set, stride **must** be large enough to contain the unsigned integer representing availability or status in addition to the query result
- VUID-vkGetQueryPoolResults-queryType-03229
  If the queryType used to create queryPool was VK_QUERY_TYPE_PERFORMANCE_QUERY_KHR, then pData and stride **must** be multiples of the size of VkPerformanceCounterResultKHR
- VUID-vkGetQueryPoolResults-queryType-04519
  If the queryType used to create queryPool was VK_QUERY_TYPE_PERFORMANCE_QUERY_KHR, then stride **must** be large enough to contain the VkQueryPoolPerformanceCreateInfoKHR::counterIndexCount used to create queryPool times the size of VkPerformanceCounterResultKHR
- VUID-vkGetQueryPoolResults-dataSize-00817
  dataSize **must** be large enough to contain the result of each query, as described here

## Valid Usage (Implicit)

- VUID-vkGetQueryPoolResults-device-parameter
  device **must** be a valid VkDevice handle

- VUID-vkGetQueryPoolResults-queryPool-parameter
  queryPool **must** be a valid VkQueryPool handle
- VUID-vkGetQueryPoolResults-pData-parameter
  pData **must** be a valid pointer to an array of dataSize bytes
- VUID-vkGetQueryPoolResults-flags-parameter
  flags **must** be a valid combination of VkQueryResultFlagBits values
- VUID-vkGetQueryPoolResults-dataSize-arraylength
  dataSize **must** be greater than 0
- VUID-vkGetQueryPoolResults-queryPool-parent
  queryPool **must** have been created, allocated, or retrieved from device

## Return Codes

**Success**

- VK_SUCCESS
- VK_NOT_READY

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY
- VK_ERROR_DEVICE_LOST

Bits which **can** be set in vkGetQueryPoolResults::flags and vkCmdCopyQueryPoolResults::flags, specifying how and when results are returned, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkQueryResultFlagBits {
    VK_QUERY_RESULT_64_BIT = 0x00000001,
    VK_QUERY_RESULT_WAIT_BIT = 0x00000002,
    VK_QUERY_RESULT_WITH_AVAILABILITY_BIT = 0x00000004,
    VK_QUERY_RESULT_PARTIAL_BIT = 0x00000008,
  // Provided by VK_KHR_video_queue
    VK_QUERY_RESULT_WITH_STATUS_BIT_KHR = 0x00000010,
} VkQueryResultFlagBits;
```

- VK_QUERY_RESULT_64_BIT specifies the results will be written as an array of 64-bit unsigned integer values. If this bit is not set, the results will be written as an array of 32-bit unsigned integer values.
- VK_QUERY_RESULT_WAIT_BIT specifies that Vulkan will wait for each query's status to become available before retrieving its results.
- VK_QUERY_RESULT_WITH_AVAILABILITY_BIT specifies that the availability status accompanies the results.

- VK_QUERY_RESULT_PARTIAL_BIT specifies that returning partial results is acceptable.
- VK_QUERY_RESULT_WITH_STATUS_BIT_KHR specifies that the last value returned in the query is a VkQueryResultStatusKHR value. See result status query for information on how an application can determine whether the use of this flag bit is supported.

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkQueryResultFlags;
```

VkQueryResultFlags is a bitmask type for setting a mask of zero or more VkQueryResultFlagBits.

Specific status codes that **can** be returned from a query are:

```
// Provided by VK_KHR_video_queue
typedef enum VkQueryResultStatusKHR {
    VK_QUERY_RESULT_STATUS_ERROR_KHR = -1,
    VK_QUERY_RESULT_STATUS_NOT_READY_KHR = 0,
    VK_QUERY_RESULT_STATUS_COMPLETE_KHR = 1,
  // Provided by VK_KHR_video_encode_queue
    VK_QUERY_RESULT_STATUS_INSUFFICIENT_BITSTREAM_BUFFER_RANGE_KHR = -1000299000,
} VkQueryResultStatusKHR;
```

- VK_QUERY_RESULT_STATUS_NOT_READY_KHR indicates that the query result is not yet available.
- VK_QUERY_RESULT_STATUS_ERROR_KHR indicates that operations did not complete successfully.
- VK_QUERY_RESULT_STATUS_COMPLETE_KHR indicates that operations completed successfully and the query result is available.
- VK_QUERY_RESULT_STATUS_INSUFFICIENT_BITSTREAM_BUFFER_RANGE_KHR indicates that a video encode operation did not complete successfully due to the destination video bitstream buffer range not being sufficiently large to fit the encoded bitstream data.

To copy query statuses and numerical results directly to buffer memory, call:

```
// Provided by VK_VERSION_1_0
void vkCmdCopyQueryPoolResults(
    VkCommandBuffer                             commandBuffer,
    VkQueryPool                                 queryPool,
    uint32_t                                    firstQuery,
    uint32_t                                    queryCount,
    VkBuffer                                    dstBuffer,
    VkDeviceSize                                dstOffset,
    VkDeviceSize                                stride,
    VkQueryResultFlags                          flags);
```

- commandBuffer is the command buffer into which this command will be recorded.
- queryPool is the query pool managing the queries containing the desired results.

- firstQuery is the initial query index.
- queryCount is the number of queries. firstQuery and queryCount together define a range of queries.
- dstBuffer is a VkBuffer object that will receive the results of the copy command.
- dstOffset is an offset into dstBuffer.
- stride is the stride in bytes between results for individual queries within dstBuffer. The required size of the backing memory for dstBuffer is determined as described above for vkGetQueryPoolResults.
- flags is a bitmask of VkQueryResultFlagBits specifying how and when results are returned.

Any results written for a query are written according to a layout dependent on the query type.

Results for any query in queryPool identified by firstQuery and queryCount that is available are copied to dstBuffer.

If VK_QUERY_RESULT_WITH_AVAILABILITY_BIT is set, results for all queries in queryPool identified by firstQuery and queryCount are copied to dstBuffer, along with an extra availability value written directly after the results of each query and interpreted as an unsigned integer. A value of zero indicates that the results are not yet available, otherwise the query is complete and results are available.

If VK_QUERY_RESULT_WITH_STATUS_BIT_KHR is set, results for all queries in queryPool identified by firstQuery and queryCount are copied to dstBuffer, along with an extra status value written directly after the results of each query and interpreted as a signed integer. A value of zero indicates that the results are not yet available. Positive values indicate that the operations within the query completed successfully, and the query results are valid. Negative values indicate that the operations within the query completed unsuccessfully.

VkQueryResultStatusKHR defines specific meaning for values returned here, though implementations are free to return other values.

If the status value written is negative, indicating that the operations within the query completed unsuccessfully, then all other results written by this command are undefined unless otherwise specified for any of the results of the used query type.

Results for any available query written by this command are final and represent the final result of the query. If VK_QUERY_RESULT_PARTIAL_BIT is set, then for any query that is unavailable, an intermediate result between zero and the final result value is written for that query. Otherwise, any result written by this command is undefined.

If VK_QUERY_RESULT_64_BIT is set, results and availability or status values for all queries are written as an array of 64-bit values. If the queryPool was created with VK_QUERY_TYPE_PERFORMANCE_QUERY_KHR, results for each query are written as an array of the type indicated by VkPerformanceCounterKHR ::storage for the counter being queried. Otherwise, results and availability or status values are written as an array of 32-bit values. If an unsigned integer query's value overflows the result type, the value **may** either wrap or saturate. If the maintenance7 feature is enabled, for an unsigned integer query, the 32-bit result value **must** be equal to the 32 least significant bits of the equivalent 64-bit result value. If a signed integer query's value overflows the result type, the value is

undefined. If a floating-point query's value is not representable as the result type, the value is undefined.

This command defines an execution dependency between other query commands that reference the same query.

The first synchronization scope includes all commands which reference the queries in `queryPool` indicated by `query` that occur earlier in submission order. If `flags` does not include `VK_QUERY_RESULT_WAIT_BIT`, vkCmdEndQueryIndexedEXT, vkCmdWriteTimestamp2KHR, vkCmdEndQuery, and vkCmdWriteTimestamp are excluded from this scope.

The second synchronization scope includes all commands which reference the queries in `queryPool` indicated by `query` that occur later in submission order.

The operation of this command happens after the first scope and happens before the second scope.

`vkCmdCopyQueryPoolResults` is considered to be a transfer operation, and its writes to buffer memory **must** be synchronized using `VK_PIPELINE_STAGE_TRANSFER_BIT` and `VK_ACCESS_TRANSFER_WRITE_BIT` before using the results.

## Valid Usage

- VUID-vkCmdCopyQueryPoolResults-firstQuery-09436
  `firstQuery` **must** be less than the number of queries in `queryPool`
- VUID-vkCmdCopyQueryPoolResults-firstQuery-09437
  The sum of `firstQuery` and `queryCount` **must** be less than or equal to the number of queries in `queryPool`
- VUID-vkCmdCopyQueryPoolResults-queryCount-09438
  If `queryCount` is greater than 1, `stride` **must** not be zero
- VUID-vkCmdCopyQueryPoolResults-queryType-09439
  If the `queryType` used to create `queryPool` was `VK_QUERY_TYPE_TIMESTAMP`, `flags` **must** not contain `VK_QUERY_RESULT_PARTIAL_BIT`
- VUID-vkCmdCopyQueryPoolResults-queryType-09440
  If the `queryType` used to create `queryPool` was `VK_QUERY_TYPE_PERFORMANCE_QUERY_KHR`, `flags` **must** not contain `VK_QUERY_RESULT_WITH_AVAILABILITY_BIT`, `VK_QUERY_RESULT_WITH_STATUS_BIT_KHR`, `VK_QUERY_RESULT_PARTIAL_BIT`, or `VK_QUERY_RESULT_64_BIT`
- VUID-vkCmdCopyQueryPoolResults-queryType-09441
  If the `queryType` used to create `queryPool` was `VK_QUERY_TYPE_PERFORMANCE_QUERY_KHR`, the `queryPool` **must** have been recorded once for each pass as retrieved via a call to vkGetPhysicalDeviceQueueFamilyPerformanceQueryPassesKHR
- VUID-vkCmdCopyQueryPoolResults-queryType-09442
  If the `queryType` used to create `queryPool` was `VK_QUERY_TYPE_RESULT_STATUS_ONLY_KHR`, then `flags` **must** include `VK_QUERY_RESULT_WITH_STATUS_BIT_KHR`
- VUID-vkCmdCopyQueryPoolResults-flags-09443

If `flags` includes `VK_QUERY_RESULT_WITH_STATUS_BIT_KHR`, then it **must** not include `VK_QUERY_RESULT_WITH_AVAILABILITY_BIT`

- VUID-vkCmdCopyQueryPoolResults-None-09402
  All queries used by the command **must** not be uninitialized when the command is executed
- VUID-vkCmdCopyQueryPoolResults-dstOffset-00819
  `dstOffset` **must** be less than the size of `dstBuffer`
- VUID-vkCmdCopyQueryPoolResults-flags-00822
  If `VK_QUERY_RESULT_64_BIT` is not set in `flags` then `dstOffset` and `stride` **must** be multiples of `4`
- VUID-vkCmdCopyQueryPoolResults-flags-00823
  If `VK_QUERY_RESULT_64_BIT` is set in `flags` then `dstOffset` and `stride` **must** be multiples of `8`
- VUID-vkCmdCopyQueryPoolResults-dstBuffer-00824
  `dstBuffer` **must** have enough storage, from `dstOffset`, to contain the result of each query, as described here
- VUID-vkCmdCopyQueryPoolResults-dstBuffer-00825
  `dstBuffer` **must** have been created with `VK_BUFFER_USAGE_TRANSFER_DST_BIT` usage flag
- VUID-vkCmdCopyQueryPoolResults-dstBuffer-00826
  If `dstBuffer` is non-sparse then it **must** be bound completely and contiguously to a single `VkDeviceMemory` object
- VUID-vkCmdCopyQueryPoolResults-queryType-03232
  If the `queryType` used to create `queryPool` was `VK_QUERY_TYPE_PERFORMANCE_QUERY_KHR`, VkPhysicalDevicePerformanceQueryPropertiesKHR::`allowCommandBufferQueryCopies` **must** be `VK_TRUE`
- VUID-vkCmdCopyQueryPoolResults-queryType-02734
  vkCmdCopyQueryPoolResults **must** not be called if the `queryType` used to create `queryPool` was `VK_QUERY_TYPE_PERFORMANCE_QUERY_INTEL`
- VUID-vkCmdCopyQueryPoolResults-None-07429
  All queries used by the command **must** not be active
- VUID-vkCmdCopyQueryPoolResults-None-08752
  All queries used by the command **must** have been made *available* by prior executed commands

## Valid Usage (Implicit)

- VUID-vkCmdCopyQueryPoolResults-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdCopyQueryPoolResults-queryPool-parameter
  `queryPool` **must** be a valid VkQueryPool handle
- VUID-vkCmdCopyQueryPoolResults-dstBuffer-parameter
  `dstBuffer` **must** be a valid VkBuffer handle

- VUID-vkCmdCopyQueryPoolResults-flags-parameter
  `flags` **must** be a valid combination of VkQueryResultFlagBits values
- VUID-vkCmdCopyQueryPoolResults-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdCopyQueryPoolResults-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics, or compute operations
- VUID-vkCmdCopyQueryPoolResults-renderpass
  This command **must** only be called outside of a render pass instance
- VUID-vkCmdCopyQueryPoolResults-videocoding
  This command **must** only be called outside of a video coding scope
- VUID-vkCmdCopyQueryPoolResults-commonparent
  Each of `commandBuffer`, `dstBuffer`, and `queryPool` **must** have been created, allocated, or retrieved from the same VkDevice

### Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

### Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Outside | Outside | Graphics<br>Compute | Action |

Rendering operations such as clears, MSAA resolves, attachment load/store operations, and blits **may** count towards the results of queries. This behavior is implementation-dependent and **may** vary depending on the path used within an implementation. For example, some implementations have several types of clears, some of which **may** include vertices and some not.

## 18.3. Occlusion Queries

Occlusion queries track the number of samples that pass the per-fragment tests for a set of drawing commands. As such, occlusion queries are only available on queue families supporting graphics operations. The application **can** then use these results to inform future rendering decisions. An occlusion query is begun and ended by calling `vkCmdBeginQuery` and `vkCmdEndQuery`, respectively. When an occlusion query begins, the count of passing samples always starts at zero. For each drawing command, the count is incremented as described in Sample Counting. If `flags` does not contain `VK_QUERY_CONTROL_PRECISE_BIT` an implementation **may** generate any non-zero result value

for the query if the count of passing samples is non-zero.

**NOTE**

Not setting VK_QUERY_CONTROL_PRECISE_BIT mode **may** be more efficient on some implementations, and **should** be used where it is sufficient to know a boolean result on whether any samples passed the per-fragment tests. In this case, some implementations **may** only return zero or one, indifferent to the actual number of samples passing the per-fragment tests.

Setting VK_QUERY_CONTROL_PRECISE_BIT does not guarantee that different implementations return the same number of samples in an occlusion query. Some implementations may kill fragments in the pre-rasterization shader stage, and these killed fragments do not contribute to the final result of the query. It is possible that some implementations generate a zero result value for the query, while others generate a non-zero value.

When an occlusion query finishes, the result for that query is marked as available. The application **can** then either copy the result to a buffer (via vkCmdCopyQueryPoolResults) or request it be put into host memory (via vkGetQueryPoolResults).

**NOTE**

If occluding geometry is not drawn first, samples **can** pass the depth test, but still not be visible in a final image.

## 18.4. Pipeline Statistics Queries

Pipeline statistics queries allow the application to sample a specified set of VkPipeline counters. These counters are accumulated by Vulkan for a set of either drawing or dispatching commands while a pipeline statistics query is active. As such, pipeline statistics queries are available on queue families supporting either graphics or compute operations. The availability of pipeline statistics queries is indicated by the pipelineStatisticsQuery member of the VkPhysicalDeviceFeatures object (see vkGetPhysicalDeviceFeatures and vkCreateDevice for detecting and requesting this query type on a VkDevice).

A pipeline statistics query is begun and ended by calling vkCmdBeginQuery and vkCmdEndQuery, respectively. When a pipeline statistics query begins, all statistics counters are set to zero. While the query is active, the pipeline type determines which set of statistics are available, but these **must** be configured on the query pool when it is created. If a statistic counter is issued on a command buffer that does not support the corresponding operation, or the counter corresponds to a shading stage which is missing from any of the pipelines used while the query is active, the value of that counter is undefined after the query has been made available. At least one statistic counter relevant to the operations supported on the recording command buffer **must** be enabled.

Bits which **can** be set in VkQueryPoolCreateInfo::pipelineStatistics for query pools and in VkCommandBufferInheritanceInfo::pipelineStatistics for secondary command buffers, individually enabling pipeline statistics counters, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkQueryPipelineStatisticFlagBits {
    VK_QUERY_PIPELINE_STATISTIC_INPUT_ASSEMBLY_VERTICES_BIT = 0x00000001,
    VK_QUERY_PIPELINE_STATISTIC_INPUT_ASSEMBLY_PRIMITIVES_BIT = 0x00000002,
    VK_QUERY_PIPELINE_STATISTIC_VERTEX_SHADER_INVOCATIONS_BIT = 0x00000004,
    VK_QUERY_PIPELINE_STATISTIC_GEOMETRY_SHADER_INVOCATIONS_BIT = 0x00000008,
    VK_QUERY_PIPELINE_STATISTIC_GEOMETRY_SHADER_PRIMITIVES_BIT = 0x00000010,
    VK_QUERY_PIPELINE_STATISTIC_CLIPPING_INVOCATIONS_BIT = 0x00000020,
    VK_QUERY_PIPELINE_STATISTIC_CLIPPING_PRIMITIVES_BIT = 0x00000040,
    VK_QUERY_PIPELINE_STATISTIC_FRAGMENT_SHADER_INVOCATIONS_BIT = 0x00000080,
    VK_QUERY_PIPELINE_STATISTIC_TESSELLATION_CONTROL_SHADER_PATCHES_BIT = 0x00000100,
    VK_QUERY_PIPELINE_STATISTIC_TESSELLATION_EVALUATION_SHADER_INVOCATIONS_BIT =
0x00000200,
    VK_QUERY_PIPELINE_STATISTIC_COMPUTE_SHADER_INVOCATIONS_BIT = 0x00000400,
  // Provided by VK_EXT_mesh_shader
    VK_QUERY_PIPELINE_STATISTIC_TASK_SHADER_INVOCATIONS_BIT_EXT = 0x00000800,
  // Provided by VK_EXT_mesh_shader
    VK_QUERY_PIPELINE_STATISTIC_MESH_SHADER_INVOCATIONS_BIT_EXT = 0x00001000,
  // Provided by VK_HUAWEI_cluster_culling_shader
    VK_QUERY_PIPELINE_STATISTIC_CLUSTER_CULLING_SHADER_INVOCATIONS_BIT_HUAWEI =
0x00002000,
} VkQueryPipelineStatisticFlagBits;
```

- `VK_QUERY_PIPELINE_STATISTIC_INPUT_ASSEMBLY_VERTICES_BIT` specifies that queries managed by the pool will count the number of vertices processed by the input assembly stage. Vertices corresponding to incomplete primitives **may** contribute to the count.
- `VK_QUERY_PIPELINE_STATISTIC_INPUT_ASSEMBLY_PRIMITIVES_BIT` specifies that queries managed by the pool will count the number of primitives processed by the input assembly stage. If primitive restart is enabled, restarting the primitive topology has no effect on the count. Incomplete primitives **may** be counted.
- `VK_QUERY_PIPELINE_STATISTIC_VERTEX_SHADER_INVOCATIONS_BIT` specifies that queries managed by the pool will count the number of vertex shader invocations. This counter's value is incremented each time a vertex shader is invoked.
- `VK_QUERY_PIPELINE_STATISTIC_GEOMETRY_SHADER_INVOCATIONS_BIT` specifies that queries managed by the pool will count the number of geometry shader invocations. This counter's value is incremented each time a geometry shader is invoked. In the case of instanced geometry shaders, the geometry shader invocations count is incremented for each separate instanced invocation.
- `VK_QUERY_PIPELINE_STATISTIC_GEOMETRY_SHADER_PRIMITIVES_BIT` specifies that queries managed by the pool will count the number of primitives generated by geometry shader invocations. The counter's value is incremented each time the geometry shader emits a primitive. Restarting primitive topology using the SPIR-V instructions `OpEndPrimitive` or `OpEndStreamPrimitive` has no effect on the geometry shader output primitives count.
- `VK_QUERY_PIPELINE_STATISTIC_CLIPPING_INVOCATIONS_BIT` specifies that queries managed by the pool will count the number of primitives processed by the Primitive Clipping stage of the pipeline. The counter's value is incremented each time a primitive reaches the primitive

clipping stage.

- VK_QUERY_PIPELINE_STATISTIC_CLIPPING_PRIMITIVES_BIT specifies that queries managed by the pool will count the number of primitives output by the Primitive Clipping stage of the pipeline. The counter's value is incremented each time a primitive passes the primitive clipping stage. The actual number of primitives output by the primitive clipping stage for a particular input primitive is implementation-dependent but **must** satisfy the following conditions:
  - If at least one vertex of the input primitive lies inside the clipping volume, the counter is incremented by one or more.
  - Otherwise, the counter is incremented by zero or more.
- VK_QUERY_PIPELINE_STATISTIC_FRAGMENT_SHADER_INVOCATIONS_BIT specifies that queries managed by the pool will count the number of fragment shader invocations. The counter's value is incremented each time the fragment shader is invoked.
- VK_QUERY_PIPELINE_STATISTIC_TESSELLATION_CONTROL_SHADER_PATCHES_BIT specifies that queries managed by the pool will count the number of patches processed by the tessellation control shader. The counter's value is incremented once for each patch for which a tessellation control shader is invoked.
- VK_QUERY_PIPELINE_STATISTIC_TESSELLATION_EVALUATION_SHADER_INVOCATIONS_BIT specifies that queries managed by the pool will count the number of invocations of the tessellation evaluation shader. The counter's value is incremented each time the tessellation evaluation shader is invoked.
- VK_QUERY_PIPELINE_STATISTIC_COMPUTE_SHADER_INVOCATIONS_BIT specifies that queries managed by the pool will count the number of compute shader invocations. The counter's value is incremented every time the compute shader is invoked. Implementations **may** skip the execution of certain compute shader invocations or execute additional compute shader invocations for implementation-dependent reasons as long as the results of rendering otherwise remain unchanged.
- VK_QUERY_PIPELINE_STATISTIC_TASK_SHADER_INVOCATIONS_BIT_EXT specifies that queries managed by the pool will count the number of task shader invocations. The counter's value is incremented every time the task shader is invoked.
- VK_QUERY_PIPELINE_STATISTIC_MESH_SHADER_INVOCATIONS_BIT_EXT specifies that queries managed by the pool will count the number of mesh shader invocations. The counter's value is incremented every time the mesh shader is invoked.

These values are intended to measure relative statistics on one implementation. Various device architectures will count these values differently. Any or all counters **may** be affected by the issues described in Query Operation.

This counting difference is especially true if the pipeline contains mesh or task shaders, which may affect several of the counters in unexpected ways.

| | |
|---|---|
| **NOTE** | For example, tile-based rendering devices **may** need to replay the scene multiple times, affecting some of the counts. |

If a pipeline has rasterizerDiscardEnable enabled, implementations **may** discard primitives after the final pre-rasterization shader stage. As a result, if rasterizerDiscardEnable is enabled, the

clipping input and output primitives counters **may** not be incremented.

When a pipeline statistics query finishes, the result for that query is marked as available. The application **can** copy the result to a buffer (via vkCmdCopyQueryPoolResults), or request it be put into host memory (via vkGetQueryPoolResults).

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkQueryPipelineStatisticFlags;
```

VkQueryPipelineStatisticFlags is a bitmask type for setting a mask of zero or more VkQueryPipelineStatisticFlagBits.

## 18.5. Timestamp Queries

*Timestamps* provide applications with a mechanism for timing the execution of commands. A timestamp is an integer value generated by the VkPhysicalDevice. Unlike other queries, timestamps do not operate over a range, and so do not use vkCmdBeginQuery or vkCmdEndQuery. The mechanism is built around a set of commands that allow the application to tell the VkPhysicalDevice to write timestamp values to a query pool and then either read timestamp values on the host (using vkGetQueryPoolResults) or copy timestamp values to a VkBuffer (using vkCmdCopyQueryPoolResults). The application **can** then compute differences between timestamps to determine execution time.

The number of valid bits in a timestamp value is determined by the VkQueueFamilyProperties::timestampValidBits property of the queue on which the timestamp is written. Timestamps are supported on any queue which reports a non-zero value for timestampValidBits via vkGetPhysicalDeviceQueueFamilyProperties. If the timestampComputeAndGraphics limit is VK_TRUE, timestamps are supported by every queue family that supports either graphics or compute operations (see VkQueueFamilyProperties).

The number of nanoseconds it takes for a timestamp value to be incremented by 1 **can** be obtained from VkPhysicalDeviceLimits::timestampPeriod after a call to vkGetPhysicalDeviceProperties.

To request a timestamp and write the value to memory, call:

```
// Provided by VK_KHR_synchronization2
void vkCmdWriteTimestamp2KHR(
    VkCommandBuffer                             commandBuffer,
    VkPipelineStageFlags2                       stage,
    VkQueryPool                                 queryPool,
    uint32_t                                    query);
```

- commandBuffer is the command buffer into which the command will be recorded.
- stage specifies a stage of the pipeline.
- queryPool is the query pool that will manage the timestamp.
- query is the query within the query pool that will contain the timestamp.

When vkCmdWriteTimestamp2KHR is submitted to a queue, it defines an execution dependency on commands that were submitted before it, and writes a timestamp to a query pool.

The first synchronization scope includes all commands that occur earlier in submission order. The synchronization scope is limited to operations on the pipeline stage specified by stage.

The second synchronization scope includes only the timestamp write operation.

**NOTE** Implementations may write the timestamp at any stage that is logically later than stage.

Any timestamp write that happens-after another timestamp write in the same submission **must** not have a lower value unless its value overflows the maximum supported integer bit width of the query. If VK_KHR_calibrated_timestamps or VK_EXT_calibrated_timestamps is enabled, this extends to timestamp writes across all submissions on the same logical device: any timestamp write that happens-after another **must** not have a lower value unless its value overflows the maximum supported integer bit width of the query. Timestamps written by this command **must** be in the VK_TIME_DOMAIN_DEVICE_KHR time domain. If an overflow occurs, the timestamp value **must** wrap back to zero.

**NOTE** Comparisons between timestamps should be done between timestamps where they are guaranteed to not decrease. For example, subtracting an older timestamp from a newer one to determine the execution time of a sequence of commands is only a reliable measurement if the two timestamp writes were performed in the same submission, or if the writes were performed on the same logical device and VK_KHR_calibrated_timestamps or VK_EXT_calibrated_timestamps is enabled.

If vkCmdWriteTimestamp2KHR is called while executing a render pass instance that has multiview enabled, the timestamp uses N consecutive query indices in the query pool (starting at query) where N is the number of bits set in the view mask of the subpass the command is executed in. The resulting query values are determined by an implementation-dependent choice of one of the following behaviors:

- The first query is a timestamp value and (if more than one bit is set in the view mask) zero is written to the remaining queries. If two timestamps are written in the same subpass, the sum of the execution time of all views between those commands is the difference between the first query written by each command.
- All N queries are timestamp values. If two timestamps are written in the same subpass, the sum of the execution time of all views between those commands is the sum of the difference between corresponding queries written by each command. The difference between corresponding queries **may** be the execution time of a single view.

In either case, the application **can** sum the differences between all N queries to determine the total execution time.

### Valid Usage

- VUID-vkCmdWriteTimestamp2-stage-03929

If the geometryShader feature is not enabled, stage **must** not contain VK_PIPELINE_STAGE_2_GEOMETRY_SHADER_BIT

- VUID-vkCmdWriteTimestamp2-stage-03930
  If the tessellationShader feature is not enabled, stage **must** not contain VK_PIPELINE_STAGE_2_TESSELLATION_CONTROL_SHADER_BIT or VK_PIPELINE_STAGE_2_TESSELLATION_EVALUATION_SHADER_BIT
- VUID-vkCmdWriteTimestamp2-stage-03931
  If the conditionalRendering feature is not enabled, stage **must** not contain VK_PIPELINE_STAGE_2_CONDITIONAL_RENDERING_BIT_EXT
- VUID-vkCmdWriteTimestamp2-stage-03932
  If the fragmentDensityMap feature is not enabled, stage **must** not contain VK_PIPELINE_STAGE_2_FRAGMENT_DENSITY_PROCESS_BIT_EXT
- VUID-vkCmdWriteTimestamp2-stage-03933
  If the transformFeedback feature is not enabled, stage **must** not contain VK_PIPELINE_STAGE_2_TRANSFORM_FEEDBACK_BIT_EXT
- VUID-vkCmdWriteTimestamp2-stage-03934
  If the meshShader feature is not enabled, stage **must** not contain VK_PIPELINE_STAGE_2_MESH_SHADER_BIT_EXT
- VUID-vkCmdWriteTimestamp2-stage-03935
  If the taskShader feature is not enabled, stage **must** not contain VK_PIPELINE_STAGE_2_TASK_SHADER_BIT_EXT
- VUID-vkCmdWriteTimestamp2-stage-07316
  If neither the shadingRateImage or attachmentFragmentShadingRate are enabled, stage **must** not contain VK_PIPELINE_STAGE_2_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR
- VUID-vkCmdWriteTimestamp2-stage-04957
  If the subpassShading feature is not enabled, stage **must** not contain VK_PIPELINE_STAGE_2_SUBPASS_SHADER_BIT_HUAWEI
- VUID-vkCmdWriteTimestamp2-stage-04995
  If the invocationMask feature is not enabled, stage **must** not contain VK_PIPELINE_STAGE_2_INVOCATION_MASK_BIT_HUAWEI
- VUID-vkCmdWriteTimestamp2-stage-07946
  If neither the VK_NV_ray_tracing extension or rayTracingPipeline feature are enabled, stage **must** not contain VK_PIPELINE_STAGE_2_RAY_TRACING_SHADER_BIT_KHR
- VUID-vkCmdWriteTimestamp2-synchronization2-03858
  The synchronization2 feature **must** be enabled
- VUID-vkCmdWriteTimestamp2-stage-03859
  stage **must** only include a single pipeline stage
- VUID-vkCmdWriteTimestamp2-stage-03860
  stage **must** only include stages valid for the queue family that was used to create the command pool that commandBuffer was allocated from
- VUID-vkCmdWriteTimestamp2-queryPool-03861
  queryPool **must** have been created with a queryType of VK_QUERY_TYPE_TIMESTAMP

- VUID-vkCmdWriteTimestamp2-timestampValidBits-03863
  The command pool's queue family **must** support a non-zero `timestampValidBits`
- VUID-vkCmdWriteTimestamp2-query-04903
  `query` **must** be less than the number of queries in `queryPool`
- VUID-vkCmdWriteTimestamp2-None-03864
  All queries used by the command **must** be *unavailable*
- VUID-vkCmdWriteTimestamp2-query-03865
  If `vkCmdWriteTimestamp2KHR` is called within a render pass instance, the sum of `query` and the number of bits set in the current subpass's view mask **must** be less than or equal to the number of queries in `queryPool`

## Valid Usage (Implicit)

- VUID-vkCmdWriteTimestamp2-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdWriteTimestamp2-stage-parameter
  `stage` **must** be a valid combination of VkPipelineStageFlagBits2 values
- VUID-vkCmdWriteTimestamp2-queryPool-parameter
  `queryPool` **must** be a valid VkQueryPool handle
- VUID-vkCmdWriteTimestamp2-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdWriteTimestamp2-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support transfer, graphics, compute, decode, or encode operations
- VUID-vkCmdWriteTimestamp2-commonparent
  Both of `commandBuffer`, and `queryPool` **must** have been created, allocated, or retrieved from the same VkDevice

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

### Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Both | Transfer<br>Graphics<br>Compute<br>Decode<br>Encode | Action |

To request a timestamp and write the value to memory, call:

```
// Provided by VK_VERSION_1_0
void vkCmdWriteTimestamp(
    VkCommandBuffer                             commandBuffer,
    VkPipelineStageFlagBits                     pipelineStage,
    VkQueryPool                                 queryPool,
    uint32_t                                    query);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `pipelineStage` is a VkPipelineStageFlagBits value, specifying a stage of the pipeline.
- `queryPool` is the query pool that will manage the timestamp.
- `query` is the query within the query pool that will contain the timestamp.

When `vkCmdWriteTimestamp` is submitted to a queue, it defines an execution dependency on commands that were submitted before it, and writes a timestamp to a query pool.

The first synchronization scope includes all commands that occur earlier in submission order. The synchronization scope is limited to operations on the pipeline stage specified by `pipelineStage`.

The second synchronization scope includes only the timestamp write operation.

**NOTE** Implementations may write the timestamp at any stage that is logically later than `stage`.

Any timestamp write that happens-after another timestamp write in the same submission **must** not have a lower value unless its value overflows the maximum supported integer bit width of the query. If `VK_KHR_calibrated_timestamps` or `VK_EXT_calibrated_timestamps` is enabled, this extends to timestamp writes across all submissions on the same logical device: any timestamp write that happens-after another **must** not have a lower value unless its value overflows the maximum supported integer bit width of the query. Timestamps written by this command **must** be in the `VK_TIME_DOMAIN_DEVICE_KHR` time domain. If an overflow occurs, the timestamp value **must** wrap back to zero.

**NOTE** Comparisons between timestamps should be done between timestamps where they

> are guaranteed to not decrease. For example, subtracting an older timestamp from a newer one to determine the execution time of a sequence of commands is only a reliable measurement if the two timestamp writes were performed in the same submission, or if the writes were performed on the same logical device and VK_KHR_calibrated_timestamps or VK_EXT_calibrated_timestamps is enabled.

If vkCmdWriteTimestamp is called while executing a render pass instance that has multiview enabled, the timestamp uses N consecutive query indices in the query pool (starting at query) where N is the number of bits set in the view mask of the subpass the command is executed in. The resulting query values are determined by an implementation-dependent choice of one of the following behaviors:

- The first query is a timestamp value and (if more than one bit is set in the view mask) zero is written to the remaining queries. If two timestamps are written in the same subpass, the sum of the execution time of all views between those commands is the difference between the first query written by each command.
- All N queries are timestamp values. If two timestamps are written in the same subpass, the sum of the execution time of all views between those commands is the sum of the difference between corresponding queries written by each command. The difference between corresponding queries **may** be the execution time of a single view.

In either case, the application **can** sum the differences between all N queries to determine the total execution time.

### Valid Usage

- VUID-vkCmdWriteTimestamp-pipelineStage-04074
  pipelineStage **must** be a valid stage for the queue family that was used to create the command pool that commandBuffer was allocated from
- VUID-vkCmdWriteTimestamp-pipelineStage-04075
  If the geometryShader feature is not enabled, pipelineStage **must** not be VK_PIPELINE_STAGE_GEOMETRY_SHADER_BIT
- VUID-vkCmdWriteTimestamp-pipelineStage-04076
  If the tessellationShader feature is not enabled, pipelineStage **must** not be VK_PIPELINE_STAGE_TESSELLATION_CONTROL_SHADER_BIT or VK_PIPELINE_STAGE_TESSELLATION_EVALUATION_SHADER_BIT
- VUID-vkCmdWriteTimestamp-pipelineStage-04077
  If the conditionalRendering feature is not enabled, pipelineStage **must** not be VK_PIPELINE_STAGE_CONDITIONAL_RENDERING_BIT_EXT
- VUID-vkCmdWriteTimestamp-pipelineStage-04078
  If the fragmentDensityMap feature is not enabled, pipelineStage **must** not be VK_PIPELINE_STAGE_FRAGMENT_DENSITY_PROCESS_BIT_EXT
- VUID-vkCmdWriteTimestamp-pipelineStage-04079
  If the transformFeedback feature is not enabled, pipelineStage **must** not be VK_PIPELINE_STAGE_TRANSFORM_FEEDBACK_BIT_EXT
- VUID-vkCmdWriteTimestamp-pipelineStage-04080

If the meshShader feature is not enabled, pipelineStage **must** not be VK_PIPELINE_STAGE_MESH_SHADER_BIT_EXT

- VUID-vkCmdWriteTimestamp-pipelineStage-07077
  If the taskShader feature is not enabled, pipelineStage **must** not be VK_PIPELINE_STAGE_TASK_SHADER_BIT_EXT
- VUID-vkCmdWriteTimestamp-shadingRateImage-07314
  If neither the shadingRateImage or attachmentFragmentShadingRate are enabled, pipelineStage **must** not be VK_PIPELINE_STAGE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR
- VUID-vkCmdWriteTimestamp-synchronization2-06489
  If the synchronization2 feature is not enabled, pipelineStage **must** not be VK_PIPELINE_STAGE_NONE
- VUID-vkCmdWriteTimestamp-rayTracingPipeline-07943
  If neither the VK_NV_ray_tracing extension or rayTracingPipeline feature are enabled, pipelineStage **must** not be VK_PIPELINE_STAGE_RAY_TRACING_SHADER_BIT_KHR
- VUID-vkCmdWriteTimestamp-queryPool-01416
  queryPool **must** have been created with a queryType of VK_QUERY_TYPE_TIMESTAMP
- VUID-vkCmdWriteTimestamp-timestampValidBits-00829
  The command pool's queue family **must** support a non-zero timestampValidBits
- VUID-vkCmdWriteTimestamp-query-04904
  query **must** be less than the number of queries in queryPool
- VUID-vkCmdWriteTimestamp-None-00830
  All queries used by the command **must** be *unavailable*
- VUID-vkCmdWriteTimestamp-query-00831
  If vkCmdWriteTimestamp is called within a render pass instance, the sum of query and the number of bits set in the current subpass's view mask **must** be less than or equal to the number of queries in queryPool

## Valid Usage (Implicit)

- VUID-vkCmdWriteTimestamp-commandBuffer-parameter
  commandBuffer **must** be a valid VkCommandBuffer handle
- VUID-vkCmdWriteTimestamp-pipelineStage-parameter
  pipelineStage **must** be a valid VkPipelineStageFlagBits value
- VUID-vkCmdWriteTimestamp-queryPool-parameter
  queryPool **must** be a valid VkQueryPool handle
- VUID-vkCmdWriteTimestamp-commandBuffer-recording
  commandBuffer **must** be in the recording state
- VUID-vkCmdWriteTimestamp-commandBuffer-cmdpool
  The VkCommandPool that commandBuffer was allocated from **must** support transfer, graphics, compute, decode, encode, or optical flow operations
- VUID-vkCmdWriteTimestamp-commonparent

Both of `commandBuffer`, and `queryPool` **must** have been created, allocated, or retrieved from the same VkDevice

### Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

### Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Both | Transfer<br>Graphics<br>Compute<br>Decode<br>Encode<br>Opticalflow | Action |

## 18.6. Performance Queries

*Performance queries* provide applications with a mechanism for getting performance counter information about the execution of command buffers, render passes, and commands.

Each queue family advertises the performance counters that **can** be queried on a queue of that family via a call to vkEnumeratePhysicalDeviceQueueFamilyPerformanceQueryCountersKHR. Implementations **may** limit access to performance counters based on platform requirements or only to specialized drivers for development purposes.

**NOTE** This may include no performance counters being enumerated, or a reduced set. Please refer to platform-specific documentation for guidance on any such restrictions.

Performance queries use the existing vkCmdBeginQuery and vkCmdEndQuery to control what command buffers, render passes, or commands to get performance information for.

Implementations **may** require multiple passes where the command buffer, render passes, or commands being recorded are the same and are executed on the same queue to record performance counter data. This is achieved by submitting the same batch and providing a VkPerformanceQuerySubmitInfoKHR structure containing a counter pass index. The number of passes required for a given performance query pool **can** be queried via a call to vkGetPhysicalDeviceQueueFamilyPerformanceQueryPassesKHR.

| | |
|---|---|
| **NOTE** | Command buffers created with VK_COMMAND_BUFFER_USAGE_ONE_TIME_SUBMIT_BIT **must** not be re-submitted. Changing command buffer usage bits **may** affect performance. To avoid this, the application **should** re-record any command buffers with the VK_COMMAND_BUFFER_USAGE_ONE_TIME_SUBMIT_BIT when multiple counter passes are required. |

Performance counter results from a performance query pool **can** be obtained with the command vkGetQueryPoolResults.

The VkPerformanceCounterResultKHR union is defined as:

```
// Provided by VK_KHR_performance_query
typedef union VkPerformanceCounterResultKHR {
    int32_t     int32;
    int64_t     int64;
    uint32_t    uint32;
    uint64_t    uint64;
    float       float32;
    double      float64;
} VkPerformanceCounterResultKHR;
```

- int32 is a 32-bit signed integer value.
- int64 is a 64-bit signed integer value.
- uint32 is a 32-bit unsigned integer value.
- uint64 is a 64-bit unsigned integer value.
- float32 is a 32-bit floating-point value.
- float64 is a 64-bit floating-point value.

Performance query results are returned in an array of VkPerformanceCounterResultKHR unions containing the data associated with each counter in the query, stored in the same order as the counters supplied in pCounterIndices when creating the performance query. VkPerformanceCounterKHR::storage specifies how to parse the counter data.

### 18.6.1. Profiling Lock

To record and submit a command buffer containing a performance query pool the profiling lock **must** be held. The profiling lock **must** be acquired prior to any call to vkBeginCommandBuffer that will be using a performance query pool. The profiling lock **must** be held while any command buffer containing a performance query pool is in the *recording*, *executable*, or *pending state*. To acquire the profiling lock, call:

```
// Provided by VK_KHR_performance_query
VkResult vkAcquireProfilingLockKHR(
    VkDevice                                    device,
    const VkAcquireProfilingLockInfoKHR*        pInfo);
```

- device is the logical device to profile.
- pInfo is a pointer to a VkAcquireProfilingLockInfoKHR structure containing information about how the profiling is to be acquired.

Implementations **may** allow multiple actors to hold the profiling lock concurrently.

### Valid Usage (Implicit)

- VUID-vkAcquireProfilingLockKHR-device-parameter
  device **must** be a valid VkDevice handle
- VUID-vkAcquireProfilingLockKHR-pInfo-parameter
  pInfo **must** be a valid pointer to a valid VkAcquireProfilingLockInfoKHR structure

### Return Codes

**Success**

- VK_SUCCESS

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_TIMEOUT

The VkAcquireProfilingLockInfoKHR structure is defined as:

```
// Provided by VK_KHR_performance_query
typedef struct VkAcquireProfilingLockInfoKHR {
    VkStructureType                    sType;
    const void*                        pNext;
    VkAcquireProfilingLockFlagsKHR     flags;
    uint64_t                           timeout;
} VkAcquireProfilingLockInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- flags is reserved for future use.
- timeout indicates how long the function waits, in nanoseconds, if the profiling lock is not available.

### Valid Usage (Implicit)

- VUID-VkAcquireProfilingLockInfoKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_ACQUIRE_PROFILING_LOCK_INFO_KHR
- VUID-VkAcquireProfilingLockInfoKHR-pNext-pNext

pNext **must** be NULL

- VUID-VkAcquireProfilingLockInfoKHR-flags-zerobitmask
  flags **must** be 0

If timeout is 0, vkAcquireProfilingLockKHR will not block while attempting to acquire the profiling lock. If timeout is UINT64_MAX, the function will not return until the profiling lock was acquired.

```
// Provided by VK_KHR_performance_query
typedef enum VkAcquireProfilingLockFlagBitsKHR {
} VkAcquireProfilingLockFlagBitsKHR;
```

```
// Provided by VK_KHR_performance_query
typedef VkFlags VkAcquireProfilingLockFlagsKHR;
```

VkAcquireProfilingLockFlagsKHR is a bitmask type for setting a mask, but is currently reserved for future use.

To release the profiling lock, call:

```
// Provided by VK_KHR_performance_query
void vkReleaseProfilingLockKHR(
    VkDevice                                    device);
```

- device is the logical device to cease profiling on.

**Valid Usage**

- VUID-vkReleaseProfilingLockKHR-device-03235
  The profiling lock of device **must** have been held via a previous successful call to vkAcquireProfilingLockKHR

**Valid Usage (Implicit)**

- VUID-vkReleaseProfilingLockKHR-device-parameter
  device **must** be a valid VkDevice handle

## 18.7. Transform Feedback Queries

Transform feedback queries track the number of primitives attempted to be written and actually written, by the vertex stream being captured, to a transform feedback buffer. This query is updated during drawing commands while transform feedback is active. The number of primitives actually written will be less than the number attempted to be written if the bound transform feedback

buffer size was too small for the number of primitives actually drawn. Primitives are not written beyond the bound range of the transform feedback buffer. A transform feedback query is begun and ended by calling `vkCmdBeginQuery` and `vkCmdEndQuery`, respectively to query for vertex stream zero. `vkCmdBeginQueryIndexedEXT` and `vkCmdEndQueryIndexedEXT` **can** be used to begin and end transform feedback queries for any supported vertex stream. When a transform feedback query begins, the count of primitives written and primitives needed starts from zero. For each drawing command, the count is incremented as vertex attribute outputs are captured to the transform feedback buffers while transform feedback is active.

When a transform feedback query finishes, the result for that query is marked as available. The application **can** then either copy the result to a buffer (via `vkCmdCopyQueryPoolResults`) or request it be put into host memory (via `vkGetQueryPoolResults`).

## 18.8. Primitives Generated Queries

When a generated primitive query for a vertex stream is active, the primitives-generated count is incremented every time a primitive emitted to that stream reaches the transform feedback stage, whether or not transform feedback is active. A primitives generated query is begun and ended by calling `vkCmdBeginQuery` and `vkCmdEndQuery`, respectively to query for vertex stream zero. `vkCmdBeginQueryIndexedEXT` and `vkCmdEndQueryIndexedEXT` **can** be used to begin and end primitives generated queries for any supported vertex stream. When a primitives generated query begins, the count of primitives generated starts from zero.

When a primitives generated query finishes, the result for that query is marked as available. The application **can** then either copy the result to a buffer (via `vkCmdCopyQueryPoolResults`) or request it be put into host memory (via `vkGetQueryPoolResults`).

| | |
|---|---|
| **NOTE** | The result of this query is typically identical to `VK_QUERY_PIPELINE_STATISTIC_CLIPPING_INVOCATIONS_BIT`, but the primitives generated query is deterministic, i.e. it must be identical to the number of primitives processed. `VK_QUERY_PIPELINE_STATISTIC_CLIPPING_INVOCATIONS_BIT` may vary for implementation-dependent reasons, e.g. the same primitive may be processed multiple times for purposes of clipping. |

## 18.9. Mesh Shader Queries

When a generated mesh primitives query is active, the mesh-primitives-generated count is incremented every time a primitive emitted from the mesh shader stage reaches the fragment shader stage. When a generated mesh primitives query begins, the mesh-primitives-generated count starts from zero.

Mesh and task shader pipeline statistics queries function the same way that invocation queries work for other shader stages, counting the number of times the respective shader stage has been run. When the statistics query begins, the invocation counters start from zero.

# 18.10. Intel Performance Queries

Intel performance queries allow an application to capture performance data for a set of commands. Performance queries are used in a similar way than other types of queries. A main difference with existing queries is that the resulting data should be handed over to a library capable to produce human readable results rather than being read directly by an application.

Prior to creating a performance query pool, initialize the device for performance queries with the call:

```
// Provided by VK_INTEL_performance_query
VkResult vkInitializePerformanceApiINTEL(
    VkDevice                                    device,
    const VkInitializePerformanceApiInfoINTEL*  pInitializeInfo);
```

- `device` is the logical device used for the queries.
- `pInitializeInfo` is a pointer to a VkInitializePerformanceApiInfoINTEL structure specifying initialization parameters.

### Valid Usage (Implicit)

- VUID-vkInitializePerformanceApiINTEL-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkInitializePerformanceApiINTEL-pInitializeInfo-parameter
  `pInitializeInfo` **must** be a valid pointer to a valid VkInitializePerformanceApiInfoINTEL structure

### Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- `VK_ERROR_TOO_MANY_OBJECTS`
- `VK_ERROR_OUT_OF_HOST_MEMORY`

The `VkInitializePerformanceApiInfoINTEL` structure is defined as :

```
// Provided by VK_INTEL_performance_query
typedef struct VkInitializePerformanceApiInfoINTEL {
    VkStructureType    sType;
    const void*        pNext;
    void*              pUserData;
} VkInitializePerformanceApiInfoINTEL;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `pUserData` is a pointer for application data.

### Valid Usage (Implicit)

- VUID-VkInitializePerformanceApiInfoINTEL-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_INITIALIZE_PERFORMANCE_API_INFO_INTEL`
- VUID-VkInitializePerformanceApiInfoINTEL-pNext-pNext
  `pNext` **must** be `NULL`

Once performance query operations have completed, uninitialize the device for performance queries with the call:

```
// Provided by VK_INTEL_performance_query
void vkUninitializePerformanceApiINTEL(
    VkDevice                                    device);
```

- `device` is the logical device used for the queries.

### Valid Usage (Implicit)

- VUID-vkUninitializePerformanceApiINTEL-device-parameter
  `device` **must** be a valid VkDevice handle

Some performance query features of a device can be discovered with the call:

```
// Provided by VK_INTEL_performance_query
VkResult vkGetPerformanceParameterINTEL(
    VkDevice                                    device,
    VkPerformanceParameterTypeINTEL             parameter,
    VkPerformanceValueINTEL*                    pValue);
```

- `device` is the logical device to query.
- `parameter` is the parameter to query.

- pValue is a pointer to a VkPerformanceValueINTEL structure in which the type and value of the parameter are returned.

### Valid Usage (Implicit)

- VUID-vkGetPerformanceParameterINTEL-device-parameter
  device **must** be a valid VkDevice handle
- VUID-vkGetPerformanceParameterINTEL-parameter-parameter
  parameter **must** be a valid VkPerformanceParameterTypeINTEL value
- VUID-vkGetPerformanceParameterINTEL-pValue-parameter
  pValue **must** be a valid pointer to a VkPerformanceValueINTEL structure

### Return Codes

**Success**

- VK_SUCCESS

**Failure**

- VK_ERROR_TOO_MANY_OBJECTS
- VK_ERROR_OUT_OF_HOST_MEMORY

Possible values of vkGetPerformanceParameterINTEL::parameter, specifying a performance query feature, are:

```
// Provided by VK_INTEL_performance_query
typedef enum VkPerformanceParameterTypeINTEL {
    VK_PERFORMANCE_PARAMETER_TYPE_HW_COUNTERS_SUPPORTED_INTEL = 0,
    VK_PERFORMANCE_PARAMETER_TYPE_STREAM_MARKER_VALID_BITS_INTEL = 1,
} VkPerformanceParameterTypeINTEL;
```

- VK_PERFORMANCE_PARAMETER_TYPE_HW_COUNTERS_SUPPORTED_INTEL has a boolean result which tells whether hardware counters can be captured.
- VK_PERFORMANCE_PARAMETER_TYPE_STREAM_MARKER_VALID_BITS_INTEL has a 32 bits integer result which tells how many bits can be written into the VkPerformanceValueINTEL value.

The VkPerformanceValueINTEL structure is defined as:

```
// Provided by VK_INTEL_performance_query
typedef struct VkPerformanceValueINTEL {
    VkPerformanceValueTypeINTEL    type;
    VkPerformanceValueDataINTEL    data;
} VkPerformanceValueINTEL;
```

- type is a VkPerformanceValueTypeINTEL value specifying the type of the returned data.
- data is a VkPerformanceValueDataINTEL union specifying the value of the returned data.

Possible values of VkPerformanceValueINTEL::type, specifying the type of the data returned in VkPerformanceValueINTEL::data, are:

- VK_PERFORMANCE_VALUE_TYPE_UINT32_INTEL specifies that unsigned 32-bit integer data is returned in data.value32.
- VK_PERFORMANCE_VALUE_TYPE_UINT64_INTEL specifies that unsigned 64-bit integer data is returned in data.value64.
- VK_PERFORMANCE_VALUE_TYPE_FLOAT_INTEL specifies that floating-point data is returned in data.valueFloat.
- VK_PERFORMANCE_VALUE_TYPE_BOOL_INTEL specifies that VkBool32 data is returned in data.valueBool.
- VK_PERFORMANCE_VALUE_TYPE_STRING_INTEL specifies that a pointer to a null-terminated UTF-8 string is returned in data.valueString. The pointer is valid for the lifetime of the device parameter passed to vkGetPerformanceParameterINTEL.

```
// Provided by VK_INTEL_performance_query
typedef enum VkPerformanceValueTypeINTEL {
    VK_PERFORMANCE_VALUE_TYPE_UINT32_INTEL = 0,
    VK_PERFORMANCE_VALUE_TYPE_UINT64_INTEL = 1,
    VK_PERFORMANCE_VALUE_TYPE_FLOAT_INTEL = 2,
    VK_PERFORMANCE_VALUE_TYPE_BOOL_INTEL = 3,
    VK_PERFORMANCE_VALUE_TYPE_STRING_INTEL = 4,
} VkPerformanceValueTypeINTEL;
```

The VkPerformanceValueDataINTEL union is defined as:

```
// Provided by VK_INTEL_performance_query
typedef union VkPerformanceValueDataINTEL {
    uint32_t       value32;
    uint64_t       value64;
    float          valueFloat;
    VkBool32       valueBool;
    const char*    valueString;
} VkPerformanceValueDataINTEL;
```

- data.value32 represents 32-bit integer data.
- data.value64 represents 64-bit integer data.
- data.valueFloat represents floating-point data.
- data.valueBool represents VkBool32 data.
- data.valueString represents a pointer to a null-terminated UTF-8 string.

The correct member of the union is determined by the associated VkPerformanceValueTypeINTEL value.

The VkQueryPoolPerformanceQueryCreateInfoINTEL structure is defined as:

```
// Provided by VK_INTEL_performance_query
typedef struct VkQueryPoolPerformanceQueryCreateInfoINTEL {
    VkStructureType                 sType;
    const void*                     pNext;
    VkQueryPoolSamplingModeINTEL    performanceCountersSampling;
} VkQueryPoolPerformanceQueryCreateInfoINTEL;
```

```
// Provided by VK_INTEL_performance_query
typedef VkQueryPoolPerformanceQueryCreateInfoINTEL VkQueryPoolCreateInfoINTEL;
```

To create a pool for Intel performance queries, set VkQueryPoolCreateInfo::queryType to VK_QUERY_TYPE_PERFORMANCE_QUERY_INTEL and add a VkQueryPoolPerformanceQueryCreateInfoINTEL structure to the pNext chain of the VkQueryPoolCreateInfo structure.

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- performanceCountersSampling describe how performance queries should be captured.

### Valid Usage (Implicit)

- VUID-VkQueryPoolPerformanceQueryCreateInfoINTEL-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_QUERY_POOL_PERFORMANCE_QUERY_CREATE_INFO_INTEL
- VUID-VkQueryPoolPerformanceQueryCreateInfoINTEL-performanceCountersSampling-parameter
  performanceCountersSampling **must** be a valid VkQueryPoolSamplingModeINTEL value

Possible values of VkQueryPoolPerformanceQueryCreateInfoINTEL::performanceCountersSampling are:

```
// Provided by VK_INTEL_performance_query
typedef enum VkQueryPoolSamplingModeINTEL {
    VK_QUERY_POOL_SAMPLING_MODE_MANUAL_INTEL = 0,
} VkQueryPoolSamplingModeINTEL;
```

- VK_QUERY_POOL_SAMPLING_MODE_MANUAL_INTEL is the default mode in which the application calls vkCmdBeginQuery and vkCmdEndQuery to record performance data.

To help associate query results with a particular point at which an application emitted commands, markers can be set into the command buffers with the call:

```
// Provided by VK_INTEL_performance_query
VkResult vkCmdSetPerformanceMarkerINTEL(
    VkCommandBuffer                             commandBuffer,
    const VkPerformanceMarkerInfoINTEL*         pMarkerInfo);
```

The last marker set onto a command buffer before the end of a query will be part of the query result.

## Valid Usage (Implicit)

- VUID-vkCmdSetPerformanceMarkerINTEL-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdSetPerformanceMarkerINTEL-pMarkerInfo-parameter
  `pMarkerInfo` **must** be a valid pointer to a valid VkPerformanceMarkerInfoINTEL structure
- VUID-vkCmdSetPerformanceMarkerINTEL-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdSetPerformanceMarkerINTEL-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics, compute, or transfer operations
- VUID-vkCmdSetPerformanceMarkerINTEL-videocoding
  This command **must** only be called outside of a video coding scope

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics<br>Compute<br>Transfer | Action<br>State |

## Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- VK_ERROR_TOO_MANY_OBJECTS
- VK_ERROR_OUT_OF_HOST_MEMORY

The VkPerformanceMarkerInfoINTEL structure is defined as:

```
// Provided by VK_INTEL_performance_query
typedef struct VkPerformanceMarkerInfoINTEL {
    VkStructureType    sType;
    const void*        pNext;
    uint64_t           marker;
} VkPerformanceMarkerInfoINTEL;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- marker is the marker value that will be recorded into the opaque query results.

### Valid Usage (Implicit)

- VUID-VkPerformanceMarkerInfoINTEL-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PERFORMANCE_MARKER_INFO_INTEL
- VUID-VkPerformanceMarkerInfoINTEL-pNext-pNext
  pNext **must** be NULL

When monitoring the behavior of an application within the dataset generated by the entire set of applications running on the system, it is useful to identify draw calls within a potentially huge amount of performance data. To do so, application can generate stream markers that will be used to trace back a particular draw call with a particular performance data item.

```
// Provided by VK_INTEL_performance_query
VkResult vkCmdSetPerformanceStreamMarkerINTEL(
    VkCommandBuffer                             commandBuffer,
    const VkPerformanceStreamMarkerInfoINTEL*   pMarkerInfo);
```

- commandBuffer is a VkCommandBuffer into which a stream marker is added.
- pMarkerInfo is a pointer to a VkPerformanceStreamMarkerInfoINTEL structure describing the marker to insert.

### Valid Usage (Implicit)

- VUID-vkCmdSetPerformanceStreamMarkerINTEL-commandBuffer-parameter
  commandBuffer **must** be a valid VkCommandBuffer handle

- VUID-vkCmdSetPerformanceStreamMarkerINTEL-pMarkerInfo-parameter
  `pMarkerInfo` **must** be a valid pointer to a valid VkPerformanceStreamMarkerInfoINTEL structure
- VUID-vkCmdSetPerformanceStreamMarkerINTEL-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdSetPerformanceStreamMarkerINTEL-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics, compute, or transfer operations
- VUID-vkCmdSetPerformanceStreamMarkerINTEL-videocoding
  This command **must** only be called outside of a video coding scope

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics<br>Compute<br>Transfer | Action<br>State |

## Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- `VK_ERROR_TOO_MANY_OBJECTS`
- `VK_ERROR_OUT_OF_HOST_MEMORY`

The `VkPerformanceStreamMarkerInfoINTEL` structure is defined as:

```
// Provided by VK_INTEL_performance_query
typedef struct VkPerformanceStreamMarkerInfoINTEL {
    VkStructureType    sType;
    const void*        pNext;
    uint32_t           marker;
} VkPerformanceStreamMarkerInfoINTEL;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `marker` is the marker value that will be recorded into the reports consumed by an external application.

### Valid Usage

- VUID-VkPerformanceStreamMarkerInfoINTEL-marker-02735
  The value written by the application into `marker` **must** only used the valid bits as reported by vkGetPerformanceParameterINTEL with the `VK_PERFORMANCE_PARAMETER_TYPE_STREAM_MARKER_VALID_BITS_INTEL`

### Valid Usage (Implicit)

- VUID-VkPerformanceStreamMarkerInfoINTEL-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PERFORMANCE_STREAM_MARKER_INFO_INTEL`
- VUID-VkPerformanceStreamMarkerInfoINTEL-pNext-pNext
  `pNext` **must** be `NULL`

Some applications might want measure the effect of a set of commands with a different settings. It is possible to override a particular settings using :

```
// Provided by VK_INTEL_performance_query
VkResult vkCmdSetPerformanceOverrideINTEL(
    VkCommandBuffer                             commandBuffer,
    const VkPerformanceOverrideInfoINTEL*       pOverrideInfo);
```

- `commandBuffer` is the command buffer where the override takes place.
- `pOverrideInfo` is a pointer to a VkPerformanceOverrideInfoINTEL structure selecting the parameter to override.

### Valid Usage

- VUID-vkCmdSetPerformanceOverrideINTEL-pOverrideInfo-02736
  `pOverrideInfo` **must** not be used with a VkPerformanceOverrideTypeINTEL that is not

reported available by `vkGetPerformanceParameterINTEL`

## Valid Usage (Implicit)

- VUID-vkCmdSetPerformanceOverrideINTEL-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdSetPerformanceOverrideINTEL-pOverrideInfo-parameter
  `pOverrideInfo` **must** be a valid pointer to a valid VkPerformanceOverrideInfoINTEL structure
- VUID-vkCmdSetPerformanceOverrideINTEL-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdSetPerformanceOverrideINTEL-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics, compute, or transfer operations
- VUID-vkCmdSetPerformanceOverrideINTEL-videocoding
  This command **must** only be called outside of a video coding scope

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics<br>Compute<br>Transfer | State |

## Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- `VK_ERROR_TOO_MANY_OBJECTS`
- `VK_ERROR_OUT_OF_HOST_MEMORY`

The VkPerformanceOverrideInfoINTEL structure is defined as:

```
// Provided by VK_INTEL_performance_query
typedef struct VkPerformanceOverrideInfoINTEL {
    VkStructureType                    sType;
    const void*                        pNext;
    VkPerformanceOverrideTypeINTEL     type;
    VkBool32                           enable;
    uint64_t                           parameter;
} VkPerformanceOverrideInfoINTEL;
```

- type is the particular VkPerformanceOverrideTypeINTEL to set.
- enable defines whether the override is enabled.
- parameter is a potential required parameter for the override.

### Valid Usage (Implicit)

- VUID-VkPerformanceOverrideInfoINTEL-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PERFORMANCE_OVERRIDE_INFO_INTEL
- VUID-VkPerformanceOverrideInfoINTEL-pNext-pNext
  pNext **must** be NULL
- VUID-VkPerformanceOverrideInfoINTEL-type-parameter
  type **must** be a valid VkPerformanceOverrideTypeINTEL value

Possible values of VkPerformanceOverrideInfoINTEL::type, specifying performance override types, are:

```
// Provided by VK_INTEL_performance_query
typedef enum VkPerformanceOverrideTypeINTEL {
    VK_PERFORMANCE_OVERRIDE_TYPE_NULL_HARDWARE_INTEL = 0,
    VK_PERFORMANCE_OVERRIDE_TYPE_FLUSH_GPU_CACHES_INTEL = 1,
} VkPerformanceOverrideTypeINTEL;
```

- VK_PERFORMANCE_OVERRIDE_TYPE_NULL_HARDWARE_INTEL turns all rendering operations into noop.
- VK_PERFORMANCE_OVERRIDE_TYPE_FLUSH_GPU_CACHES_INTEL stalls the stream of commands until all previously emitted commands have completed and all caches been flushed and invalidated.

Before submitting command buffers containing performance queries commands to a device queue, the application **must** acquire and set a performance query configuration. The configuration can be released once all command buffers containing performance query commands are not in a pending state.

```
// Provided by VK_INTEL_performance_query
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkPerformanceConfigurationINTEL)
```

To acquire a device performance configuration, call:

```
// Provided by VK_INTEL_performance_query
VkResult vkAcquirePerformanceConfigurationINTEL(
    VkDevice                                    device,
    const VkPerformanceConfigurationAcquireInfoINTEL* pAcquireInfo,
    VkPerformanceConfigurationINTEL*            pConfiguration);
```

- `device` is the logical device that the performance query commands will be submitted to.
- `pAcquireInfo` is a pointer to a VkPerformanceConfigurationAcquireInfoINTEL structure, specifying the performance configuration to acquire.
- `pConfiguration` is a pointer to a `VkPerformanceConfigurationINTEL` handle in which the resulting configuration object is returned.

### Valid Usage (Implicit)

- VUID-vkAcquirePerformanceConfigurationINTEL-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkAcquirePerformanceConfigurationINTEL-pAcquireInfo-parameter
  `pAcquireInfo` **must** be a valid pointer to a valid VkPerformanceConfigurationAcquireInfoINTEL structure
- VUID-vkAcquirePerformanceConfigurationINTEL-pConfiguration-parameter
  `pConfiguration` **must** be a valid pointer to a VkPerformanceConfigurationINTEL handle

### Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- `VK_ERROR_TOO_MANY_OBJECTS`
- `VK_ERROR_OUT_OF_HOST_MEMORY`

The `VkPerformanceConfigurationAcquireInfoINTEL` structure is defined as:

```
// Provided by VK_INTEL_performance_query
typedef struct VkPerformanceConfigurationAcquireInfoINTEL {
    VkStructureType                         sType;
    const void*                             pNext;
    VkPerformanceConfigurationTypeINTEL     type;
} VkPerformanceConfigurationAcquireInfoINTEL;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `type` is one of the VkPerformanceConfigurationTypeINTEL type of performance configuration that will be acquired.

### Valid Usage (Implicit)

- VUID-VkPerformanceConfigurationAcquireInfoINTEL-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PERFORMANCE_CONFIGURATION_ACQUIRE_INFO_INTEL`
- VUID-VkPerformanceConfigurationAcquireInfoINTEL-pNext-pNext
  `pNext` **must** be `NULL`
- VUID-VkPerformanceConfigurationAcquireInfoINTEL-type-parameter
  `type` **must** be a valid VkPerformanceConfigurationTypeINTEL value

Possible values of VkPerformanceConfigurationAcquireInfoINTEL::`type`, specifying performance configuration types, are:

```
// Provided by VK_INTEL_performance_query
typedef enum VkPerformanceConfigurationTypeINTEL {
    VK_PERFORMANCE_CONFIGURATION_TYPE_COMMAND_QUEUE_METRICS_DISCOVERY_ACTIVATED_INTEL
= 0,
} VkPerformanceConfigurationTypeINTEL;
```

To set a performance configuration, call:

```
// Provided by VK_INTEL_performance_query
VkResult vkQueueSetPerformanceConfigurationINTEL(
    VkQueue                                     queue,
    VkPerformanceConfigurationINTEL             configuration);
```

- `queue` is the queue on which the configuration will be used.
- `configuration` is the configuration to use.

### Valid Usage (Implicit)

- VUID-vkQueueSetPerformanceConfigurationINTEL-queue-parameter
  `queue` **must** be a valid VkQueue handle
- VUID-vkQueueSetPerformanceConfigurationINTEL-configuration-parameter
  `configuration` **must** be a valid VkPerformanceConfigurationINTEL handle
- VUID-vkQueueSetPerformanceConfigurationINTEL-commonparent
  Both of `configuration`, and `queue` **must** have been created, allocated, or retrieved from the same VkDevice

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| - | - | - | Any | - |

## Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- `VK_ERROR_TOO_MANY_OBJECTS`
- `VK_ERROR_OUT_OF_HOST_MEMORY`

To release a device performance configuration, call:

```
// Provided by VK_INTEL_performance_query
VkResult vkReleasePerformanceConfigurationINTEL(
    VkDevice                                    device,
    VkPerformanceConfigurationINTEL             configuration);
```

- `device` is the device associated to the configuration object to release.
- `configuration` is the configuration object to release.

## Valid Usage

- VUID-vkReleasePerformanceConfigurationINTEL-configuration-02737
  `configuration` **must** not be released before all command buffers submitted while the configuration was set are in pending state

### Valid Usage (Implicit)

- VUID-vkReleasePerformanceConfigurationINTEL-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkReleasePerformanceConfigurationINTEL-configuration-parameter
  If `configuration` is not VK_NULL_HANDLE, `configuration` **must** be a valid VkPerformanceConfigurationINTEL handle
- VUID-vkReleasePerformanceConfigurationINTEL-configuration-parent
  If `configuration` is a valid handle, it **must** have been created, allocated, or retrieved from `device`

### Host Synchronization

- Host access to `configuration` **must** be externally synchronized

### Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- `VK_ERROR_TOO_MANY_OBJECTS`
- `VK_ERROR_OUT_OF_HOST_MEMORY`

## 18.11. Result Status Queries

Result status queries serve a single purpose: allowing the application to determine whether a set of operations have completed successfully or not, as indicated by the VkQueryResultStatusKHR value written when retrieving the result of a query using the `VK_QUERY_RESULT_WITH_STATUS_BIT_KHR` flag.

Unlike other query types, result status queries do not track or maintain any other data beyond the completion status, thus no other data is written when retrieving their results.

Support for result status queries is indicated by VkQueueFamilyQueryResultStatusPropertiesKHR::`queryResultStatusSupport` , as returned by vkGetPhysicalDeviceQueueFamilyProperties2 for the queue family in question.

## 18.12. Video Encode Feedback Queries

Video encode feedback queries allow the application to capture feedback values generated by video encode operations. As such, video encode feedback queries are available on queue families supporting video encode operations. The availability of individual video encode feedback values is indicated by the bits of VkVideoEncodeCapabilitiesKHR::`supportedEncodeFeedbackFlags`, as returned

by vkGetPhysicalDeviceVideoCapabilitiesKHR for the video profile the queries are intended to be used with.

The set of enabled video encode feedback values **must** be configured on the query pool when it is created using the encodeFeedbackFlags member of the VkQueryPoolVideoEncodeFeedbackCreateInfoKHR included in the pNext chain of VkQueryPoolCreateInfo.

The VkQueryPoolVideoEncodeFeedbackCreateInfoKHR structure is defined as:

```
// Provided by VK_KHR_video_encode_queue
typedef struct VkQueryPoolVideoEncodeFeedbackCreateInfoKHR {
    VkStructureType                 sType;
    const void*                     pNext;
    VkVideoEncodeFeedbackFlagsKHR   encodeFeedbackFlags;
} VkQueryPoolVideoEncodeFeedbackCreateInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- encodeFeedbackFlags is a bitmask of VkVideoEncodeFeedbackFlagBitsKHR values specifying the set of enabled video encode feedback values captured by queries of the new pool.

### Valid Usage (Implicit)

- VUID-VkQueryPoolVideoEncodeFeedbackCreateInfoKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_QUERY_POOL_VIDEO_ENCODE_FEEDBACK_CREATE_INFO_KHR
- VUID-VkQueryPoolVideoEncodeFeedbackCreateInfoKHR-encodeFeedbackFlags-parameter
  encodeFeedbackFlags **must** be a valid combination of VkVideoEncodeFeedbackFlagBitsKHR values
- VUID-VkQueryPoolVideoEncodeFeedbackCreateInfoKHR-encodeFeedbackFlags-requiredbitmask
  encodeFeedbackFlags **must** not be 0

Bits which **can** be set in VkQueryPoolVideoEncodeFeedbackCreateInfoKHR::encodeFeedbackFlags for video encode feedback query pools are:

```
// Provided by VK_KHR_video_encode_queue
typedef enum VkVideoEncodeFeedbackFlagBitsKHR {
    VK_VIDEO_ENCODE_FEEDBACK_BITSTREAM_BUFFER_OFFSET_BIT_KHR = 0x00000001,
    VK_VIDEO_ENCODE_FEEDBACK_BITSTREAM_BYTES_WRITTEN_BIT_KHR = 0x00000002,
    VK_VIDEO_ENCODE_FEEDBACK_BITSTREAM_HAS_OVERRIDES_BIT_KHR = 0x00000004,
} VkVideoEncodeFeedbackFlagBitsKHR;
```

- VK_VIDEO_ENCODE_FEEDBACK_BITSTREAM_BUFFER_OFFSET_BIT_KHR specifies that queries managed by the pool will capture the byte offset of the bitstream data written by the video encode operation

to the bitstream buffer specified in VkVideoEncodeInfoKHR::dstBuffer relative to the offset specified in VkVideoEncodeInfoKHR::dstBufferOffset. For the first video encode operation issued by any video encode command, this value will always be zero, meaning that bitstream data is always written to the buffer specified in VkVideoEncodeInfoKHR::dstBuffer starting from the offset specified in VkVideoEncodeInfoKHR::dstBufferOffset.

- VK_VIDEO_ENCODE_FEEDBACK_BITSTREAM_BYTES_WRITTEN_BIT_KHR specifies that queries managed by the pool will capture the number of bytes written by the video encode operation to the bitstream buffer specified in VkVideoEncodeInfoKHR::dstBuffer.
- VK_VIDEO_ENCODE_FEEDBACK_BITSTREAM_HAS_OVERRIDES_BIT_KHR specifies that queries managed by the pool will capture a boolean value indicating that the data written to the bitstream buffer specified in VkVideoEncodeInfoKHR::dstBuffer contains overridden parameters.

When retrieving the results of video encode feedback queries, the values corresponding to each enabled video encode feedback are written in the order of the bits defined above, followed by an optional value indicating availability or result status if VK_QUERY_RESULT_WITH_AVAILABILITY_BIT or VK_QUERY_RESULT_WITH_STATUS_BIT_KHR is specified, respectively.

If the result status of a video encode feedback query is negative, then the results of all enabled video encode feedback values will be undefined.

**NOTE** Thus it is recommended that applications always specify VK_QUERY_RESULT_WITH_STATUS_BIT_KHR when retrieving the results of video encode feedback queries and ignore such undefined video encode feedback values for any unsuccessfully completed video encode operations.

```
// Provided by VK_KHR_video_encode_queue
typedef VkFlags VkVideoEncodeFeedbackFlagsKHR;
```

VkVideoEncodeFeedbackFlagsKHR is a bitmask type for setting a mask of zero or more VkVideoEncodeFeedbackFlagBitsKHR.

# Chapter 19. Clear Commands

## 19.1. Clearing Images Outside a Render Pass Instance

Color and depth/stencil images **can** be cleared outside a render pass instance using vkCmdClearColorImage or vkCmdClearDepthStencilImage, respectively. These commands are only allowed outside of a render pass instance.

To clear one or more subranges of a color image, call:

```
// Provided by VK_VERSION_1_0
void vkCmdClearColorImage(
    VkCommandBuffer                             commandBuffer,
    VkImage                                     image,
    VkImageLayout                               imageLayout,
    const VkClearColorValue*                    pColor,
    uint32_t                                    rangeCount,
    const VkImageSubresourceRange*              pRanges);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `image` is the image to be cleared.
- `imageLayout` specifies the current layout of the image subresource ranges to be cleared, and **must** be `VK_IMAGE_LAYOUT_SHARED_PRESENT_KHR`, `VK_IMAGE_LAYOUT_GENERAL` or `VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL`.
- `pColor` is a pointer to a VkClearColorValue structure containing the values that the image subresource ranges will be cleared to (see Clear Values below).
- `rangeCount` is the number of image subresource range structures in `pRanges`.
- `pRanges` is a pointer to an array of VkImageSubresourceRange structures describing a range of mipmap levels, array layers, and aspects to be cleared, as described in Image Views.

Each specified range in `pRanges` is cleared to the value specified by `pColor`.

### Valid Usage

- VUID-vkCmdClearColorImage-image-01993
  The format features of `image` **must** contain `VK_FORMAT_FEATURE_TRANSFER_DST_BIT`
- VUID-vkCmdClearColorImage-image-00002
  `image` **must** have been created with `VK_IMAGE_USAGE_TRANSFER_DST_BIT` usage flag
- VUID-vkCmdClearColorImage-image-01545
  `image` **must** not use any of the formats that require a sampler $Y'C_BC_R$ conversion
- VUID-vkCmdClearColorImage-image-00003
  If `image` is non-sparse then it **must** be bound completely and contiguously to a single `VkDeviceMemory` object

- VUID-vkCmdClearColorImage-imageLayout-00004
  `imageLayout` **must** specify the layout of the image subresource ranges of `image` specified in `pRanges` at the time this command is executed on a `VkDevice`
- VUID-vkCmdClearColorImage-imageLayout-01394
  `imageLayout` **must** be `VK_IMAGE_LAYOUT_SHARED_PRESENT_KHR`, `VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL` or `VK_IMAGE_LAYOUT_GENERAL`
- VUID-vkCmdClearColorImage-aspectMask-02498
  The VkImageSubresourceRange::`aspectMask` members of the elements of the `pRanges` array **must** each only include `VK_IMAGE_ASPECT_COLOR_BIT`
- VUID-vkCmdClearColorImage-baseMipLevel-01470
  The VkImageSubresourceRange::`baseMipLevel` members of the elements of the `pRanges` array **must** each be less than the `mipLevels` specified in VkImageCreateInfo when `image` was created
- VUID-vkCmdClearColorImage-pRanges-01692
  For each VkImageSubresourceRange element of `pRanges`, if the `levelCount` member is not `VK_REMAINING_MIP_LEVELS`, then `baseMipLevel` + `levelCount` **must** be less than or equal to the `mipLevels` specified in VkImageCreateInfo when `image` was created
- VUID-vkCmdClearColorImage-baseArrayLayer-01472
  The VkImageSubresourceRange::`baseArrayLayer` members of the elements of the `pRanges` array **must** each be less than the `arrayLayers` specified in VkImageCreateInfo when `image` was created
- VUID-vkCmdClearColorImage-pRanges-01693
  For each VkImageSubresourceRange element of `pRanges`, if the `layerCount` member is not `VK_REMAINING_ARRAY_LAYERS`, then `baseArrayLayer` + `layerCount` **must** be less than or equal to the `arrayLayers` specified in VkImageCreateInfo when `image` was created
- VUID-vkCmdClearColorImage-image-00007
  `image` **must** not have a compressed or depth/stencil format
- VUID-vkCmdClearColorImage-pColor-04961
  `pColor` **must** be a valid pointer to a VkClearColorValue union
- VUID-vkCmdClearColorImage-commandBuffer-01805
  If `commandBuffer` is an unprotected command buffer and `protectedNoFault` is not supported, `image` **must** not be a protected image
- VUID-vkCmdClearColorImage-commandBuffer-01806
  If `commandBuffer` is a protected command buffer and `protectedNoFault` is not supported, **must** not be an unprotected image
- VUID-vkCmdClearColorImage-image-09678
  If `image`'s format has components other than R and G, it **must** not have a 64-bit component width

## Valid Usage (Implicit)

- VUID-vkCmdClearColorImage-commandBuffer-parameter

`commandBuffer` **must** be a valid VkCommandBuffer handle

- VUID-vkCmdClearColorImage-image-parameter
  `image` **must** be a valid VkImage handle
- VUID-vkCmdClearColorImage-imageLayout-parameter
  `imageLayout` **must** be a valid VkImageLayout value
- VUID-vkCmdClearColorImage-pRanges-parameter
  `pRanges` **must** be a valid pointer to an array of `rangeCount` valid VkImageSubresourceRange structures
- VUID-vkCmdClearColorImage-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdClearColorImage-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics, or compute operations
- VUID-vkCmdClearColorImage-renderpass
  This command **must** only be called outside of a render pass instance
- VUID-vkCmdClearColorImage-videocoding
  This command **must** only be called outside of a video coding scope
- VUID-vkCmdClearColorImage-rangeCount-arraylength
  `rangeCount` **must** be greater than `0`
- VUID-vkCmdClearColorImage-commonparent
  Both of `commandBuffer`, and `image` **must** have been created, allocated, or retrieved from the same VkDevice

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Outside | Outside | Graphics<br>Compute | Action |

To clear one or more subranges of a depth/stencil image, call:

```
// Provided by VK_VERSION_1_0
void vkCmdClearDepthStencilImage(
    VkCommandBuffer                             commandBuffer,
    VkImage                                     image,
    VkImageLayout                               imageLayout,
    const VkClearDepthStencilValue*             pDepthStencil,
    uint32_t                                    rangeCount,
    const VkImageSubresourceRange*              pRanges);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `image` is the image to be cleared.
- `imageLayout` specifies the current layout of the image subresource ranges to be cleared, and **must** be `VK_IMAGE_LAYOUT_GENERAL` or `VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL`.
- `pDepthStencil` is a pointer to a VkClearDepthStencilValue structure containing the values that the depth and stencil image subresource ranges will be cleared to (see Clear Values below).
- `rangeCount` is the number of image subresource range structures in `pRanges`.
- `pRanges` is a pointer to an array of VkImageSubresourceRange structures describing a range of mipmap levels, array layers, and aspects to be cleared, as described in Image Views.

## Valid Usage

- VUID-vkCmdClearDepthStencilImage-image-01994
  The format features of `image` **must** contain `VK_FORMAT_FEATURE_TRANSFER_DST_BIT`
- VUID-vkCmdClearDepthStencilImage-pRanges-02658
  If the `aspect` member of any element of `pRanges` includes `VK_IMAGE_ASPECT_STENCIL_BIT`, and `image` was created with separate stencil usage, `VK_IMAGE_USAGE_TRANSFER_DST_BIT` **must** have been included in the VkImageStencilUsageCreateInfo::`stencilUsage` used to create `image`
- VUID-vkCmdClearDepthStencilImage-pRanges-02659
  If the `aspect` member of any element of `pRanges` includes `VK_IMAGE_ASPECT_STENCIL_BIT`, and `image` was not created with separate stencil usage, `VK_IMAGE_USAGE_TRANSFER_DST_BIT` **must** have been included in the VkImageCreateInfo::`usage` used to create `image`
- VUID-vkCmdClearDepthStencilImage-pRanges-02660
  If the `aspect` member of any element of `pRanges` includes `VK_IMAGE_ASPECT_DEPTH_BIT`, `VK_IMAGE_USAGE_TRANSFER_DST_BIT` **must** have been included in the VkImageCreateInfo::`usage` used to create `image`
- VUID-vkCmdClearDepthStencilImage-image-00010
  If `image` is non-sparse then it **must** be bound completely and contiguously to a single `VkDeviceMemory` object
- VUID-vkCmdClearDepthStencilImage-imageLayout-00011
  `imageLayout` **must** specify the layout of the image subresource ranges of `image` specified in `pRanges` at the time this command is executed on a `VkDevice`
- VUID-vkCmdClearDepthStencilImage-imageLayout-00012

imageLayout **must** be either of VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL or VK_IMAGE_LAYOUT_GENERAL

- VUID-vkCmdClearDepthStencilImage-aspectMask-02824
  The VkImageSubresourceRange::aspectMask member of each element of the pRanges array **must** not include bits other than VK_IMAGE_ASPECT_DEPTH_BIT or VK_IMAGE_ASPECT_STENCIL_BIT
- VUID-vkCmdClearDepthStencilImage-image-02825
  If the image's format does not have a stencil component, then the VkImageSubresourceRange::aspectMask member of each element of the pRanges array **must** not include the VK_IMAGE_ASPECT_STENCIL_BIT bit
- VUID-vkCmdClearDepthStencilImage-image-02826
  If the image's format does not have a depth component, then the VkImageSubresourceRange::aspectMask member of each element of the pRanges array **must** not include the VK_IMAGE_ASPECT_DEPTH_BIT bit
- VUID-vkCmdClearDepthStencilImage-baseMipLevel-01474
  The VkImageSubresourceRange::baseMipLevel members of the elements of the pRanges array **must** each be less than the mipLevels specified in VkImageCreateInfo when image was created
- VUID-vkCmdClearDepthStencilImage-pRanges-01694
  For each VkImageSubresourceRange element of pRanges, if the levelCount member is not VK_REMAINING_MIP_LEVELS, then baseMipLevel + levelCount **must** be less than or equal to the mipLevels specified in VkImageCreateInfo when image was created
- VUID-vkCmdClearDepthStencilImage-baseArrayLayer-01476
  The VkImageSubresourceRange::baseArrayLayer members of the elements of the pRanges array **must** each be less than the arrayLayers specified in VkImageCreateInfo when image was created
- VUID-vkCmdClearDepthStencilImage-pRanges-01695
  For each VkImageSubresourceRange element of pRanges, if the layerCount member is not VK_REMAINING_ARRAY_LAYERS, then baseArrayLayer + layerCount **must** be less than or equal to the arrayLayers specified in VkImageCreateInfo when image was created
- VUID-vkCmdClearDepthStencilImage-image-00014
  image **must** have a depth/stencil format
- VUID-vkCmdClearDepthStencilImage-commandBuffer-01807
  If commandBuffer is an unprotected command buffer and protectedNoFault is not supported, image **must** not be a protected image
- VUID-vkCmdClearDepthStencilImage-commandBuffer-01808
  If commandBuffer is a protected command buffer and protectedNoFault is not supported, image **must** not be an unprotected image

## Valid Usage (Implicit)

- VUID-vkCmdClearDepthStencilImage-commandBuffer-parameter
  commandBuffer **must** be a valid VkCommandBuffer handle

- VUID-vkCmdClearDepthStencilImage-image-parameter
  `image` **must** be a valid VkImage handle
- VUID-vkCmdClearDepthStencilImage-imageLayout-parameter
  `imageLayout` **must** be a valid VkImageLayout value
- VUID-vkCmdClearDepthStencilImage-pDepthStencil-parameter
  `pDepthStencil` **must** be a valid pointer to a valid VkClearDepthStencilValue structure
- VUID-vkCmdClearDepthStencilImage-pRanges-parameter
  `pRanges` **must** be a valid pointer to an array of `rangeCount` valid VkImageSubresourceRange structures
- VUID-vkCmdClearDepthStencilImage-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdClearDepthStencilImage-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- VUID-vkCmdClearDepthStencilImage-renderpass
  This command **must** only be called outside of a render pass instance
- VUID-vkCmdClearDepthStencilImage-videocoding
  This command **must** only be called outside of a video coding scope
- VUID-vkCmdClearDepthStencilImage-rangeCount-arraylength
  `rangeCount` **must** be greater than `0`
- VUID-vkCmdClearDepthStencilImage-commonparent
  Both of `commandBuffer`, and `image` **must** have been created, allocated, or retrieved from the same VkDevice

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Outside | Outside | Graphics | Action |

Clears outside render pass instances are treated as transfer operations for the purposes of memory barriers.

## 19.2. Clearing Images Inside a Render Pass Instance

To clear one or more regions of color and depth/stencil attachments inside a render pass instance, call:

```
// Provided by VK_VERSION_1_0
void vkCmdClearAttachments(
    VkCommandBuffer                             commandBuffer,
    uint32_t                                    attachmentCount,
    const VkClearAttachment*                    pAttachments,
    uint32_t                                    rectCount,
    const VkClearRect*                          pRects);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `attachmentCount` is the number of entries in the `pAttachments` array.
- `pAttachments` is a pointer to an array of VkClearAttachment structures defining the attachments to clear and the clear values to use.
- `rectCount` is the number of entries in the `pRects` array.
- `pRects` is a pointer to an array of VkClearRect structures defining regions within each selected attachment to clear.

If the render pass has a fragment density map attachment, clears follow the operations of fragment density maps as if each clear region was a primitive which generates fragments. The clear color is applied to all pixels inside each fragment's area regardless if the pixels lie outside of the clear region. Clears **may** have a different set of supported fragment areas than draws.

Unlike other clear commands, vkCmdClearAttachments is not a transfer command. It performs its operations in rasterization order. For color attachments, the operations are executed as color attachment writes, by the `VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT` stage. For depth/stencil attachments, the operations are executed as depth writes and stencil writes by the `VK_PIPELINE_STAGE_EARLY_FRAGMENT_TESTS_BIT` and `VK_PIPELINE_STAGE_LATE_FRAGMENT_TESTS_BIT` stages.

`vkCmdClearAttachments` is not affected by the bound pipeline state.

| | |
|---|---|
| **NOTE** | It is generally preferable to clear attachments by using the `VK_ATTACHMENT_LOAD_OP_CLEAR` load operation at the start of rendering, as it is more efficient on some implementations. |

If any attachment's `aspectMask` to be cleared is not backed by an image view, the clear has no effect on that aspect.

If an attachment being cleared refers to an image view created with an `aspectMask` equal to one of `VK_IMAGE_ASPECT_PLANE_0_BIT`, `VK_IMAGE_ASPECT_PLANE_1_BIT` or `VK_IMAGE_ASPECT_PLANE_2_BIT`, it is considered to be `VK_IMAGE_ASPECT_COLOR_BIT` for purposes of this command, and **must** be cleared with the `VK_IMAGE_ASPECT_COLOR_BIT` aspect as specified by image view creation.

## Valid Usage

- VUID-vkCmdClearAttachments-aspectMask-07884
  If the current render pass instance does not use dynamic rendering, and the `aspectMask` member of any element of `pAttachments` contains `VK_IMAGE_ASPECT_DEPTH_BIT`, the current subpass instance's depth-stencil attachment **must** be either `VK_ATTACHMENT_UNUSED` or the attachment `format` **must** contain a depth component
- VUID-vkCmdClearAttachments-aspectMask-07885
  If the current render pass instance does not use dynamic rendering, and the `aspectMask` member of any element of `pAttachments` contains `VK_IMAGE_ASPECT_STENCIL_BIT`, the current subpass instance's depth-stencil attachment **must** be either `VK_ATTACHMENT_UNUSED` or the attachment `format` **must** contain a stencil component
- VUID-vkCmdClearAttachments-aspectMask-07271
  If the `aspectMask` member of any element of `pAttachments` contains `VK_IMAGE_ASPECT_COLOR_BIT`, the `colorAttachment` **must** be a valid color attachment index in the current render pass instance
- VUID-vkCmdClearAttachments-rect-02682
  The `rect` member of each element of `pRects` **must** have an `extent.width` greater than `0`
- VUID-vkCmdClearAttachments-rect-02683
  The `rect` member of each element of `pRects` **must** have an `extent.height` greater than `0`
- VUID-vkCmdClearAttachments-pRects-00016
  The rectangular region specified by each element of `pRects` **must** be contained within the render area of the current render pass instance
- VUID-vkCmdClearAttachments-pRects-06937
  The layers specified by each element of `pRects` **must** be contained within every attachment that `pAttachments` refers to, i.e. for each element of `pRects`, VkClearRect::`baseArrayLayer` + VkClearRect::`layerCount` **must** be less than or equal to the number of layers rendered to in the current render pass instance
- VUID-vkCmdClearAttachments-layerCount-01934
  The `layerCount` member of each element of `pRects` **must** not be `0`
- VUID-vkCmdClearAttachments-commandBuffer-02504
  If `commandBuffer` is an unprotected command buffer and `protectedNoFault` is not supported, each attachment to be cleared **must** not be a protected image
- VUID-vkCmdClearAttachments-commandBuffer-02505
  If `commandBuffer` is a protected command buffer and `protectedNoFault` is not supported, each attachment to be cleared **must** not be an unprotected image
- VUID-vkCmdClearAttachments-baseArrayLayer-00018
  If the render pass instance this is recorded in uses multiview, then `baseArrayLayer` **must** be zero and `layerCount` **must** be one
- VUID-vkCmdClearAttachments-colorAttachment-09503
  The `colorAttachment` member of each element of `pAttachments` **must** not identify a color attachment that is currently mapped to `VK_ATTACHMENT_UNUSED` in `commandBuffer` via VkRenderingAttachmentLocationInfoKHR

- VUID-vkCmdClearAttachments-aspectMask-09298
  If the subpass this is recorded in performs an external format resolve, the `aspectMask` member of any element of `pAttachments` **must** not include `VK_IMAGE_ASPECT_PLANE_i_BIT` for any index *i*
- VUID-vkCmdClearAttachments-None-09679
  If the attachment format has components other than R and G, it **must** not have a 64-bit component width

## Valid Usage (Implicit)

- VUID-vkCmdClearAttachments-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdClearAttachments-pAttachments-parameter
  `pAttachments` **must** be a valid pointer to an array of `attachmentCount` valid VkClearAttachment structures
- VUID-vkCmdClearAttachments-pRects-parameter
  `pRects` **must** be a valid pointer to an array of `rectCount` VkClearRect structures
- VUID-vkCmdClearAttachments-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdClearAttachments-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- VUID-vkCmdClearAttachments-renderpass
  This command **must** only be called inside of a render pass instance
- VUID-vkCmdClearAttachments-videocoding
  This command **must** only be called outside of a video coding scope
- VUID-vkCmdClearAttachments-attachmentCount-arraylength
  `attachmentCount` **must** be greater than `0`
- VUID-vkCmdClearAttachments-rectCount-arraylength
  `rectCount` **must** be greater than `0`

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

### Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Inside | Outside | Graphics | Action |

The `VkClearRect` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkClearRect {
    VkRect2D    rect;
    uint32_t    baseArrayLayer;
    uint32_t    layerCount;
} VkClearRect;
```

- `rect` is the two-dimensional region to be cleared.
- `baseArrayLayer` is the first layer to be cleared.
- `layerCount` is the number of layers to clear.

The layers [`baseArrayLayer`, `baseArrayLayer` + `layerCount`) counting from the base layer of the attachment image view are cleared.

The `VkClearAttachment` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkClearAttachment {
    VkImageAspectFlags    aspectMask;
    uint32_t              colorAttachment;
    VkClearValue          clearValue;
} VkClearAttachment;
```

- `aspectMask` is a mask selecting the color, depth and/or stencil aspects of the attachment to be cleared.
- `colorAttachment` is only meaningful if `VK_IMAGE_ASPECT_COLOR_BIT` is set in `aspectMask`, in which case it is an index into the currently bound color attachments.
- `clearValue` is the color or depth/stencil value to clear the attachment to, as described in Clear Values below.

### Valid Usage

- VUID-VkClearAttachment-aspectMask-00019
  If `aspectMask` includes `VK_IMAGE_ASPECT_COLOR_BIT`, it **must** not include

VK_IMAGE_ASPECT_DEPTH_BIT or VK_IMAGE_ASPECT_STENCIL_BIT

- VUID-VkClearAttachment-aspectMask-00020
  aspectMask **must** not include VK_IMAGE_ASPECT_METADATA_BIT
- VUID-VkClearAttachment-aspectMask-02246
  aspectMask **must** not include VK_IMAGE_ASPECT_MEMORY_PLANE_i_BIT_EXT for any index *i*

**Valid Usage (Implicit)**

- VUID-VkClearAttachment-aspectMask-parameter
  aspectMask **must** be a valid combination of VkImageAspectFlagBits values
- VUID-VkClearAttachment-aspectMask-requiredbitmask
  aspectMask **must** not be 0

## 19.3. Clear Values

The VkClearColorValue structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef union VkClearColorValue {
    float       float32[4];
    int32_t     int32[4];
    uint32_t    uint32[4];
} VkClearColorValue;
```

- float32 are the color clear values when the format of the image or attachment is one of the numeric formats with a numeric type that is floating-point. Floating-point values are automatically converted to the format of the image, with the clear value being treated as linear if the image is sRGB.
- int32 are the color clear values when the format of the image or attachment has a numeric type that is signed integer (SINT). Signed integer values are converted to the format of the image by casting to the smaller type (with negative 32-bit values mapping to negative values in the smaller type). If the integer clear value is not representable in the target type (e.g. would overflow in conversion to that type), the clear value is undefined.
- uint32 are the color clear values when the format of the image or attachment has a numeric type that is unsigned integer (UINT). Unsigned integer values are converted to the format of the image by casting to the integer type with fewer bits.

The four array elements of the clear color map to R, G, B, and A components of image formats, in order.

If the image has more than one sample, the same value is written to all samples for any pixels being cleared.

If the image or attachment format has a 64-bit component width, the first 2 array elements of each

of the arrays above are reinterpreted as a single 64-bit element for the R component. The next 2 array elements are used in the same way for the G component. In other words, the union behaves as if it had the following additional members:

```
double float64[2];
int64_t int64[2];
uint64_t uint64[2];
```

The VkClearDepthStencilValue structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkClearDepthStencilValue {
    float       depth;
    uint32_t    stencil;
} VkClearDepthStencilValue;
```

- depth is the clear value for the depth aspect of the depth/stencil attachment. It is a floating-point value which is automatically converted to the attachment's format.
- stencil is the clear value for the stencil aspect of the depth/stencil attachment. It is a 32-bit integer value which is converted to the attachment's format by taking the appropriate number of LSBs.

### Valid Usage

- VUID-VkClearDepthStencilValue-depth-00022
  Unless the VK_EXT_depth_range_unrestricted extension is enabled depth **must** be between 0.0 and 1.0, inclusive

The VkClearValue union is defined as:

```
// Provided by VK_VERSION_1_0
typedef union VkClearValue {
    VkClearColorValue           color;
    VkClearDepthStencilValue    depthStencil;
} VkClearValue;
```

- color specifies the color image clear values to use when clearing a color image or attachment.
- depthStencil specifies the depth and stencil clear values to use when clearing a depth/stencil image or attachment.

This union is used where part of the API requires either color or depth/stencil clear values, depending on the attachment, and defines the initial clear values in the VkRenderPassBeginInfo structure.

## 19.4. Filling Buffers

To clear buffer data, call:

```
// Provided by VK_VERSION_1_0
void vkCmdFillBuffer(
    VkCommandBuffer                             commandBuffer,
    VkBuffer                                    dstBuffer,
    VkDeviceSize                                dstOffset,
    VkDeviceSize                                size,
    uint32_t                                    data);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `dstBuffer` is the buffer to be filled.
- `dstOffset` is the byte offset into the buffer at which to start filling, and **must** be a multiple of 4.
- `size` is the number of bytes to fill, and **must** be either a multiple of 4, or `VK_WHOLE_SIZE` to fill the range from `offset` to the end of the buffer. If `VK_WHOLE_SIZE` is used and the remaining size of the buffer is not a multiple of 4, then the nearest smaller multiple is used.
- `data` is the 4-byte word written repeatedly to the buffer to fill `size` bytes of data. The data word is written to memory according to the host endianness.

`vkCmdFillBuffer` is treated as a "transfer" operation for the purposes of synchronization barriers. The `VK_BUFFER_USAGE_TRANSFER_DST_BIT` **must** be specified in `usage` of VkBufferCreateInfo in order for the buffer to be compatible with `vkCmdFillBuffer`.

### Valid Usage

- VUID-vkCmdFillBuffer-dstOffset-00024
  `dstOffset` **must** be less than the size of `dstBuffer`
- VUID-vkCmdFillBuffer-dstOffset-00025
  `dstOffset` **must** be a multiple of `4`
- VUID-vkCmdFillBuffer-size-00026
  If `size` is not equal to `VK_WHOLE_SIZE`, `size` **must** be greater than `0`
- VUID-vkCmdFillBuffer-size-00027
  If `size` is not equal to `VK_WHOLE_SIZE`, `size` **must** be less than or equal to the size of `dstBuffer` minus `dstOffset`
- VUID-vkCmdFillBuffer-size-00028
  If `size` is not equal to `VK_WHOLE_SIZE`, `size` **must** be a multiple of `4`
- VUID-vkCmdFillBuffer-dstBuffer-00029
  `dstBuffer` **must** have been created with `VK_BUFFER_USAGE_TRANSFER_DST_BIT` usage flag
- VUID-vkCmdFillBuffer-apiVersion-07894
  If the VK_KHR_maintenance1 extension is not enabled and VkPhysicalDeviceProperties::`apiVersion` is less than Vulkan 1.1, the VkCommandPool that `commandBuffer` was allocated

from **must** support graphics or compute operations

- VUID-vkCmdFillBuffer-dstBuffer-00031
  If `dstBuffer` is non-sparse then it **must** be bound completely and contiguously to a single VkDeviceMemory object
- VUID-vkCmdFillBuffer-commandBuffer-01811
  If `commandBuffer` is an unprotected command buffer and `protectedNoFault` is not supported, `dstBuffer` **must** not be a protected buffer
- VUID-vkCmdFillBuffer-commandBuffer-01812
  If `commandBuffer` is a protected command buffer and `protectedNoFault` is not supported, `dstBuffer` **must** not be an unprotected buffer

## Valid Usage (Implicit)

- VUID-vkCmdFillBuffer-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdFillBuffer-dstBuffer-parameter
  `dstBuffer` **must** be a valid VkBuffer handle
- VUID-vkCmdFillBuffer-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdFillBuffer-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support transfer, graphics or compute operations
- VUID-vkCmdFillBuffer-renderpass
  This command **must** only be called outside of a render pass instance
- VUID-vkCmdFillBuffer-videocoding
  This command **must** only be called outside of a video coding scope
- VUID-vkCmdFillBuffer-commonparent
  Both of `commandBuffer`, and `dstBuffer` **must** have been created, allocated, or retrieved from the same VkDevice

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

**Command Properties**

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Outside | Outside | Transfer<br>Graphics<br>Compute | Action |

## 19.5. Updating Buffers

To update buffer data inline in a command buffer, call:

```
// Provided by VK_VERSION_1_0
void vkCmdUpdateBuffer(
    VkCommandBuffer                             commandBuffer,
    VkBuffer                                    dstBuffer,
    VkDeviceSize                                dstOffset,
    VkDeviceSize                                dataSize,
    const void*                                 pData);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `dstBuffer` is a handle to the buffer to be updated.
- `dstOffset` is the byte offset into the buffer to start updating, and **must** be a multiple of 4.
- `dataSize` is the number of bytes to update, and **must** be a multiple of 4.
- `pData` is a pointer to the source data for the buffer update, and **must** be at least `dataSize` bytes in size.

`dataSize` **must** be less than or equal to 65536 bytes. For larger updates, applications **can** use buffer to buffer copies.

**NOTE**

Buffer updates performed with `vkCmdUpdateBuffer` first copy the data into command buffer memory when the command is recorded (which requires additional storage and may incur an additional allocation), and then copy the data from the command buffer into `dstBuffer` when the command is executed on a device.

The additional cost of this functionality compared to buffer to buffer copies means it is only recommended for very small amounts of data, and is why it is limited to only 65536 bytes.

Applications **can** work around this by issuing multiple `vkCmdUpdateBuffer` commands to different ranges of the same buffer, but it is strongly recommended that they **should** not.

The source data is copied from `pData` to the command buffer when the command is called.

vkCmdUpdateBuffer is only allowed outside of a render pass. This command is treated as a "transfer" operation for the purposes of synchronization barriers. The VK_BUFFER_USAGE_TRANSFER_DST_BIT **must** be specified in usage of VkBufferCreateInfo in order for the buffer to be compatible with vkCmdUpdateBuffer.

### Valid Usage

- VUID-vkCmdUpdateBuffer-dstOffset-00032
  dstOffset **must** be less than the size of dstBuffer
- VUID-vkCmdUpdateBuffer-dataSize-00033
  dataSize **must** be less than or equal to the size of dstBuffer minus dstOffset
- VUID-vkCmdUpdateBuffer-dstBuffer-00034
  dstBuffer **must** have been created with VK_BUFFER_USAGE_TRANSFER_DST_BIT usage flag
- VUID-vkCmdUpdateBuffer-dstBuffer-00035
  If dstBuffer is non-sparse then it **must** be bound completely and contiguously to a single VkDeviceMemory object
- VUID-vkCmdUpdateBuffer-dstOffset-00036
  dstOffset **must** be a multiple of 4
- VUID-vkCmdUpdateBuffer-dataSize-00037
  dataSize **must** be less than or equal to 65536
- VUID-vkCmdUpdateBuffer-dataSize-00038
  dataSize **must** be a multiple of 4
- VUID-vkCmdUpdateBuffer-commandBuffer-01813
  If commandBuffer is an unprotected command buffer and protectedNoFault is not supported, dstBuffer **must** not be a protected buffer
- VUID-vkCmdUpdateBuffer-commandBuffer-01814
  If commandBuffer is a protected command buffer and protectedNoFault is not supported, dstBuffer **must** not be an unprotected buffer

### Valid Usage (Implicit)

- VUID-vkCmdUpdateBuffer-commandBuffer-parameter
  commandBuffer **must** be a valid VkCommandBuffer handle
- VUID-vkCmdUpdateBuffer-dstBuffer-parameter
  dstBuffer **must** be a valid VkBuffer handle
- VUID-vkCmdUpdateBuffer-pData-parameter
  pData **must** be a valid pointer to an array of dataSize bytes
- VUID-vkCmdUpdateBuffer-commandBuffer-recording
  commandBuffer **must** be in the recording state
- VUID-vkCmdUpdateBuffer-commandBuffer-cmdpool
  The VkCommandPool that commandBuffer was allocated from **must** support transfer, graphics, or compute operations

- VUID-vkCmdUpdateBuffer-renderpass
  This command **must** only be called outside of a render pass instance
- VUID-vkCmdUpdateBuffer-videocoding
  This command **must** only be called outside of a video coding scope
- VUID-vkCmdUpdateBuffer-dataSize-arraylength
  `dataSize` **must** be greater than `0`
- VUID-vkCmdUpdateBuffer-commonparent
  Both of `commandBuffer`, and `dstBuffer` **must** have been created, allocated, or retrieved from the same VkDevice

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Outside | Outside | Transfer<br>Graphics<br>Compute | Action |

**NOTE** The `pData` parameter was of type `uint32_t*` instead of `void*` prior to version 1.0.19 of the Specification and VK_HEADER_VERSION 19 of the Vulkan Header Files. This was a historical anomaly, as the source data may be of other types.

# Chapter 20. Copy Commands

An application **can** copy buffer and image data using several methods described in this chapter, depending on the type of data transfer.

All copy commands are treated as “transfer” operations for the purposes of synchronization barriers.

All copy commands that have a source format with an X component in its format description read undefined values from those bits.

All copy commands that have a destination format with an X component in its format description write undefined values to those bits.

## 20.1. Copying Data Between Buffers

To copy data between buffer objects, call:

```
// Provided by VK_VERSION_1_0
void vkCmdCopyBuffer(
    VkCommandBuffer                             commandBuffer,
    VkBuffer                                    srcBuffer,
    VkBuffer                                    dstBuffer,
    uint32_t                                    regionCount,
    const VkBufferCopy*                         pRegions);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `srcBuffer` is the source buffer.
- `dstBuffer` is the destination buffer.
- `regionCount` is the number of regions to copy.
- `pRegions` is a pointer to an array of VkBufferCopy structures specifying the regions to copy.

Each source region specified by `pRegions` is copied from the source buffer to the destination region of the destination buffer. If any of the specified regions in `srcBuffer` overlaps in memory with any of the specified regions in `dstBuffer`, values read from those overlapping regions are undefined.

### Valid Usage

- VUID-vkCmdCopyBuffer-commandBuffer-01822
  If `commandBuffer` is an unprotected command buffer and `protectedNoFault` is not supported, `srcBuffer` **must** not be a protected buffer
- VUID-vkCmdCopyBuffer-commandBuffer-01823
  If `commandBuffer` is an unprotected command buffer and `protectedNoFault` is not supported, `dstBuffer` **must** not be a protected buffer
- VUID-vkCmdCopyBuffer-commandBuffer-01824

If `commandBuffer` is a protected command buffer and `protectedNoFault` is not supported, `dstBuffer` **must** not be an unprotected buffer

- VUID-vkCmdCopyBuffer-srcOffset-00113
  The `srcOffset` member of each element of `pRegions` **must** be less than the size of `srcBuffer`
- VUID-vkCmdCopyBuffer-dstOffset-00114
  The `dstOffset` member of each element of `pRegions` **must** be less than the size of `dstBuffer`
- VUID-vkCmdCopyBuffer-size-00115
  The `size` member of each element of `pRegions` **must** be less than or equal to the size of `srcBuffer` minus `srcOffset`
- VUID-vkCmdCopyBuffer-size-00116
  The `size` member of each element of `pRegions` **must** be less than or equal to the size of `dstBuffer` minus `dstOffset`
- VUID-vkCmdCopyBuffer-pRegions-00117
  The union of the source regions, and the union of the destination regions, specified by the elements of `pRegions`, **must** not overlap in memory
- VUID-vkCmdCopyBuffer-srcBuffer-00118
  `srcBuffer` **must** have been created with `VK_BUFFER_USAGE_TRANSFER_SRC_BIT` usage flag
- VUID-vkCmdCopyBuffer-srcBuffer-00119
  If `srcBuffer` is non-sparse then it **must** be bound completely and contiguously to a single `VkDeviceMemory` object
- VUID-vkCmdCopyBuffer-dstBuffer-00120
  `dstBuffer` **must** have been created with `VK_BUFFER_USAGE_TRANSFER_DST_BIT` usage flag
- VUID-vkCmdCopyBuffer-dstBuffer-00121
  If `dstBuffer` is non-sparse then it **must** be bound completely and contiguously to a single `VkDeviceMemory` object

## Valid Usage (Implicit)

- VUID-vkCmdCopyBuffer-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdCopyBuffer-srcBuffer-parameter
  `srcBuffer` **must** be a valid VkBuffer handle
- VUID-vkCmdCopyBuffer-dstBuffer-parameter
  `dstBuffer` **must** be a valid VkBuffer handle
- VUID-vkCmdCopyBuffer-pRegions-parameter
  `pRegions` **must** be a valid pointer to an array of `regionCount` valid VkBufferCopy structures
- VUID-vkCmdCopyBuffer-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdCopyBuffer-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support transfer, graphics, or compute operations

- VUID-vkCmdCopyBuffer-renderpass
  This command **must** only be called outside of a render pass instance
- VUID-vkCmdCopyBuffer-videocoding
  This command **must** only be called outside of a video coding scope
- VUID-vkCmdCopyBuffer-regionCount-arraylength
  `regionCount` **must** be greater than `0`
- VUID-vkCmdCopyBuffer-commonparent
  Each of `commandBuffer`, `dstBuffer`, and `srcBuffer` **must** have been created, allocated, or retrieved from the same VkDevice

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Outside | Outside | Transfer<br>Graphics<br>Compute | Action |

The `VkBufferCopy` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkBufferCopy {
    VkDeviceSize    srcOffset;
    VkDeviceSize    dstOffset;
    VkDeviceSize    size;
} VkBufferCopy;
```

- `srcOffset` is the starting offset in bytes from the start of `srcBuffer`.
- `dstOffset` is the starting offset in bytes from the start of `dstBuffer`.
- `size` is the number of bytes to copy.

## Valid Usage

- VUID-VkBufferCopy-size-01988
  The `size` **must** be greater than `0`

A more extensible version of the copy buffer command is defined below.

To copy data between buffer objects, call:

```
// Provided by VK_KHR_copy_commands2
void vkCmdCopyBuffer2KHR(
    VkCommandBuffer                             commandBuffer,
    const VkCopyBufferInfo2*                    pCopyBufferInfo);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `pCopyBufferInfo` is a pointer to a VkCopyBufferInfo2 structure describing the copy parameters.

Each source region specified by `pCopyBufferInfo->pRegions` is copied from the source buffer to the destination region of the destination buffer. If any of the specified regions in `pCopyBufferInfo->srcBuffer` overlaps in memory with any of the specified regions in `pCopyBufferInfo->dstBuffer`, values read from those overlapping regions are undefined.

### Valid Usage

- VUID-vkCmdCopyBuffer2-commandBuffer-01822
  If `commandBuffer` is an unprotected command buffer and `protectedNoFault` is not supported, `srcBuffer` **must** not be a protected buffer
- VUID-vkCmdCopyBuffer2-commandBuffer-01823
  If `commandBuffer` is an unprotected command buffer and `protectedNoFault` is not supported, `dstBuffer` **must** not be a protected buffer
- VUID-vkCmdCopyBuffer2-commandBuffer-01824
  If `commandBuffer` is a protected command buffer and `protectedNoFault` is not supported, `dstBuffer` **must** not be an unprotected buffer

### Valid Usage (Implicit)

- VUID-vkCmdCopyBuffer2-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdCopyBuffer2-pCopyBufferInfo-parameter
  `pCopyBufferInfo` **must** be a valid pointer to a valid VkCopyBufferInfo2 structure
- VUID-vkCmdCopyBuffer2-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdCopyBuffer2-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support transfer, graphics, or compute operations
- VUID-vkCmdCopyBuffer2-renderpass
  This command **must** only be called outside of a render pass instance
- VUID-vkCmdCopyBuffer2-videocoding

This command **must** only be called outside of a video coding scope

### Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

### Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Outside | Outside | Transfer<br>Graphics<br>Compute | Action |

The `VkCopyBufferInfo2` structure is defined as:

```
typedef struct VkCopyBufferInfo2 {
    VkStructureType         sType;
    const void*             pNext;
    VkBuffer                srcBuffer;
    VkBuffer                dstBuffer;
    uint32_t                regionCount;
    const VkBufferCopy2*    pRegions;
} VkCopyBufferInfo2;
```

or the equivalent

```
// Provided by VK_KHR_copy_commands2
typedef VkCopyBufferInfo2 VkCopyBufferInfo2KHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `srcBuffer` is the source buffer.
- `dstBuffer` is the destination buffer.
- `regionCount` is the number of regions to copy.
- `pRegions` is a pointer to an array of VkBufferCopy2 structures specifying the regions to copy.

## Valid Usage

- VUID-VkCopyBufferInfo2-srcOffset-00113
  The `srcOffset` member of each element of `pRegions` **must** be less than the size of `srcBuffer`
- VUID-VkCopyBufferInfo2-dstOffset-00114
  The `dstOffset` member of each element of `pRegions` **must** be less than the size of `dstBuffer`
- VUID-VkCopyBufferInfo2-size-00115
  The `size` member of each element of `pRegions` **must** be less than or equal to the size of `srcBuffer` minus `srcOffset`
- VUID-VkCopyBufferInfo2-size-00116
  The `size` member of each element of `pRegions` **must** be less than or equal to the size of `dstBuffer` minus `dstOffset`
- VUID-VkCopyBufferInfo2-pRegions-00117
  The union of the source regions, and the union of the destination regions, specified by the elements of `pRegions`, **must** not overlap in memory
- VUID-VkCopyBufferInfo2-srcBuffer-00118
  `srcBuffer` **must** have been created with `VK_BUFFER_USAGE_TRANSFER_SRC_BIT` usage flag
- VUID-VkCopyBufferInfo2-srcBuffer-00119
  If `srcBuffer` is non-sparse then it **must** be bound completely and contiguously to a single `VkDeviceMemory` object
- VUID-VkCopyBufferInfo2-dstBuffer-00120
  `dstBuffer` **must** have been created with `VK_BUFFER_USAGE_TRANSFER_DST_BIT` usage flag
- VUID-VkCopyBufferInfo2-dstBuffer-00121
  If `dstBuffer` is non-sparse then it **must** be bound completely and contiguously to a single `VkDeviceMemory` object

## Valid Usage (Implicit)

- VUID-VkCopyBufferInfo2-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_COPY_BUFFER_INFO_2`
- VUID-VkCopyBufferInfo2-pNext-pNext
  `pNext` **must** be `NULL`
- VUID-VkCopyBufferInfo2-srcBuffer-parameter
  `srcBuffer` **must** be a valid VkBuffer handle
- VUID-VkCopyBufferInfo2-dstBuffer-parameter
  `dstBuffer` **must** be a valid VkBuffer handle
- VUID-VkCopyBufferInfo2-pRegions-parameter
  `pRegions` **must** be a valid pointer to an array of `regionCount` valid VkBufferCopy2 structures
- VUID-VkCopyBufferInfo2-regionCount-arraylength
  `regionCount` **must** be greater than `0`

- VUID-VkCopyBufferInfo2-commonparent
  Both of `dstBuffer`, and `srcBuffer` **must** have been created, allocated, or retrieved from the same VkDevice

The `VkBufferCopy2` structure is defined as:

```
typedef struct VkBufferCopy2 {
    VkStructureType    sType;
    const void*        pNext;
    VkDeviceSize       srcOffset;
    VkDeviceSize       dstOffset;
    VkDeviceSize       size;
} VkBufferCopy2;
```

or the equivalent

```
// Provided by VK_KHR_copy_commands2
typedef VkBufferCopy2 VkBufferCopy2KHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `srcOffset` is the starting offset in bytes from the start of `srcBuffer`.
- `dstOffset` is the starting offset in bytes from the start of `dstBuffer`.
- `size` is the number of bytes to copy.

### Valid Usage

- VUID-VkBufferCopy2-size-01988
  The `size` **must** be greater than `0`

### Valid Usage (Implicit)

- VUID-VkBufferCopy2-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_BUFFER_COPY_2`
- VUID-VkBufferCopy2-pNext-pNext
  `pNext` **must** be `NULL`

## 20.2. Copying Data Between Images

To copy data between image objects, call:

```
// Provided by VK_VERSION_1_0
void vkCmdCopyImage(
    VkCommandBuffer                             commandBuffer,
    VkImage                                     srcImage,
    VkImageLayout                               srcImageLayout,
    VkImage                                     dstImage,
    VkImageLayout                               dstImageLayout,
    uint32_t                                    regionCount,
    const VkImageCopy*                          pRegions);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `srcImage` is the source image.
- `srcImageLayout` is the current layout of the source image subresource.
- `dstImage` is the destination image.
- `dstImageLayout` is the current layout of the destination image subresource.
- `regionCount` is the number of regions to copy.
- `pRegions` is a pointer to an array of VkImageCopy structures specifying the regions to copy.

Each source region specified by `pRegions` is copied from the source image to the destination region of the destination image. If any of the specified regions in `srcImage` overlaps in memory with any of the specified regions in `dstImage`, values read from those overlapping regions are undefined.

Multi-planar images **can** only be copied on a per-plane basis, and the subresources used in each region when copying to or from such images **must** specify only one plane, though different regions **can** specify different planes. When copying planes of multi-planar images, the format considered is the compatible format for that plane, rather than the format of the multi-planar image.

If the format of the destination image has a different block extent than the source image (e.g. one is a compressed format), the offset and extent for each of the regions specified is scaled according to the block extents of each format to match in size. Copy regions for each image **must** be aligned to a multiple of the texel block extent in each dimension, except at the edges of the image, where region extents **must** match the edge of the image.

Image data **can** be copied between images with different image types. If one image is `VK_IMAGE_TYPE_3D` and the other image is `VK_IMAGE_TYPE_2D` with multiple layers, then each slice is copied to or from a different layer; `depth` slices in the 3D image correspond to `layerCount` layers in the 2D image, with an effective `depth` of `1` used for the 2D image. If `maintenance5` is enabled, all other combinations are allowed and function as if 1D images are 2D images with a height of 1. Otherwise, other combinations of image types are disallowed.

### Valid Usage

- VUID-vkCmdCopyImage-commandBuffer-01825
  If `commandBuffer` is an unprotected command buffer and `protectedNoFault` is not supported, `srcImage` **must** not be a protected image

- VUID-vkCmdCopyImage-commandBuffer-01826
  If `commandBuffer` is an unprotected command buffer and `protectedNoFault` is not supported, `dstImage` **must** not be a protected image
- VUID-vkCmdCopyImage-commandBuffer-01827
  If `commandBuffer` is a protected command buffer and `protectedNoFault` is not supported, `dstImage` **must** not be an unprotected image
- VUID-vkCmdCopyImage-pRegions-00124
  The union of all source regions, and the union of all destination regions, specified by the elements of `pRegions`, **must** not overlap in memory
- VUID-vkCmdCopyImage-srcImage-01995
  The format features of `srcImage` **must** contain `VK_FORMAT_FEATURE_TRANSFER_SRC_BIT`
- VUID-vkCmdCopyImage-srcImageLayout-00128
  `srcImageLayout` **must** specify the layout of the image subresources of `srcImage` specified in `pRegions` at the time this command is executed on a `VkDevice`
- VUID-vkCmdCopyImage-srcImageLayout-01917
  `srcImageLayout` **must** be `VK_IMAGE_LAYOUT_SHARED_PRESENT_KHR`, `VK_IMAGE_LAYOUT_TRANSFER_SRC_OPTIMAL`, or `VK_IMAGE_LAYOUT_GENERAL`
- VUID-vkCmdCopyImage-srcImage-09460
  If `srcImage` and `dstImage` are the same, and any elements of `pRegions` contains the `srcSubresource` and `dstSubresource` with matching `mipLevel` and overlapping array layers, then the `srcImageLayout` and `dstImageLayout` **must** be `VK_IMAGE_LAYOUT_GENERAL` or `VK_IMAGE_LAYOUT_SHARED_PRESENT_KHR`
- VUID-vkCmdCopyImage-dstImage-01996
  The format features of `dstImage` **must** contain `VK_FORMAT_FEATURE_TRANSFER_DST_BIT`
- VUID-vkCmdCopyImage-dstImageLayout-00133
  `dstImageLayout` **must** specify the layout of the image subresources of `dstImage` specified in `pRegions` at the time this command is executed on a `VkDevice`
- VUID-vkCmdCopyImage-dstImageLayout-01395
  `dstImageLayout` **must** be `VK_IMAGE_LAYOUT_SHARED_PRESENT_KHR`, `VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL`, or `VK_IMAGE_LAYOUT_GENERAL`
- VUID-vkCmdCopyImage-srcImage-01548
  If the VkFormat of each of `srcImage` and `dstImage` is not a *multi-planar format*, the VkFormat of each of `srcImage` and `dstImage` **must** be size-compatible
- VUID-vkCmdCopyImage-None-01549
  In a copy to or from a plane of a multi-planar image, the VkFormat of the image and plane **must** be compatible according to the description of compatible planes for the plane being copied
- VUID-vkCmdCopyImage-srcImage-09247
  If the VkFormat of each of `srcImage` and `dstImage` is a compressed image format, the formats **must** have the same texel block extent
- VUID-vkCmdCopyImage-srcImage-00136
  The sample count of `srcImage` and `dstImage` **must** match

- VUID-vkCmdCopyImage-srcOffset-01783
  The `srcOffset` and `extent` members of each element of `pRegions` **must** respect the image transfer granularity requirements of `commandBuffer`'s command pool's queue family, as described in VkQueueFamilyProperties
- VUID-vkCmdCopyImage-dstOffset-01784
  The `dstOffset` and `extent` members of each element of `pRegions` **must** respect the image transfer granularity requirements of `commandBuffer`'s command pool's queue family, as described in VkQueueFamilyProperties
- VUID-vkCmdCopyImage-srcImage-01551
  If neither `srcImage` nor `dstImage` has a multi-planar image format then for each element of `pRegions`, `srcSubresource.aspectMask` and `dstSubresource.aspectMask` **must** match
- VUID-vkCmdCopyImage-srcImage-08713
  If `srcImage` has a multi-planar image format, then for each element of `pRegions`, `srcSubresource.aspectMask` **must** be a single valid multi-planar aspect mask bit
- VUID-vkCmdCopyImage-dstImage-08714
  If `dstImage` has a multi-planar image format, then for each element of `pRegions`, `dstSubresource.aspectMask` **must** be a single valid multi-planar aspect mask bit
- VUID-vkCmdCopyImage-srcImage-01556
  If `srcImage` has a multi-planar image format and the `dstImage` does not have a multi-planar image format, then for each element of `pRegions`, `dstSubresource.aspectMask` **must** be `VK_IMAGE_ASPECT_COLOR_BIT`
- VUID-vkCmdCopyImage-dstImage-01557
  If `dstImage` has a multi-planar image format and the `srcImage` does not have a multi-planar image format, then for each element of `pRegions`, `srcSubresource.aspectMask` **must** be `VK_IMAGE_ASPECT_COLOR_BIT`
- VUID-vkCmdCopyImage-apiVersion-07932
  If the VK_KHR_maintenance1 extension is not enabled, or VkPhysicalDeviceProperties::`apiVersion` is less than Vulkan 1.1, and either `srcImage` or `dstImage` is of type `VK_IMAGE_TYPE_3D`, then for each element of `pRegions`, `srcSubresource.baseArrayLayer` and `dstSubresource.baseArrayLayer` **must** both be `0`, and `srcSubresource.layerCount` and `dstSubresource.layerCount` **must** both be `1`
- VUID-vkCmdCopyImage-srcImage-04443
  If `srcImage` is of type `VK_IMAGE_TYPE_3D`, then for each element of `pRegions`, `srcSubresource.baseArrayLayer` **must** be `0` and `srcSubresource.layerCount` **must** be `1`
- VUID-vkCmdCopyImage-dstImage-04444
  If `dstImage` is of type `VK_IMAGE_TYPE_3D`, then for each element of `pRegions`, `dstSubresource.baseArrayLayer` **must** be `0` and `dstSubresource.layerCount` **must** be `1`
- VUID-vkCmdCopyImage-aspectMask-00142
  For each element of `pRegions`, `srcSubresource.aspectMask` **must** specify aspects present in `srcImage`
- VUID-vkCmdCopyImage-aspectMask-00143
  For each element of `pRegions`, `dstSubresource.aspectMask` **must** specify aspects present in `dstImage`

- VUID-vkCmdCopyImage-srcOffset-00144
  For each element of pRegions, srcOffset.x and (extent.width + srcOffset.x) **must** both be greater than or equal to 0 and less than or equal to the width of the specified srcSubresource of srcImage
- VUID-vkCmdCopyImage-srcOffset-00145
  For each element of pRegions, srcOffset.y and (extent.height + srcOffset.y) **must** both be greater than or equal to 0 and less than or equal to the height of the specified srcSubresource of srcImage
- VUID-vkCmdCopyImage-srcImage-00146
  If srcImage is of type VK_IMAGE_TYPE_1D, then for each element of pRegions, srcOffset.y **must** be 0 and extent.height **must** be 1
- VUID-vkCmdCopyImage-srcOffset-00147
  If srcImage is of type VK_IMAGE_TYPE_3D, then for each element of pRegions, srcOffset.z and (extent.depth + srcOffset.z) **must** both be greater than or equal to 0 and less than or equal to the depth of the specified srcSubresource of srcImage
- VUID-vkCmdCopyImage-srcImage-01785
  If srcImage is of type VK_IMAGE_TYPE_1D, then for each element of pRegions, srcOffset.z **must** be 0 and extent.depth **must** be 1
- VUID-vkCmdCopyImage-dstImage-01786
  If dstImage is of type VK_IMAGE_TYPE_1D, then for each element of pRegions, dstOffset.z **must** be 0 and extent.depth **must** be 1
- VUID-vkCmdCopyImage-srcImage-01787
  If srcImage is of type VK_IMAGE_TYPE_2D, then for each element of pRegions, srcOffset.z **must** be 0
- VUID-vkCmdCopyImage-dstImage-01788
  If dstImage is of type VK_IMAGE_TYPE_2D, then for each element of pRegions, dstOffset.z **must** be 0
- VUID-vkCmdCopyImage-apiVersion-07933
  If the VK_KHR_maintenance1 extension is not enabled, and VkPhysicalDeviceProperties::apiVersion is less than Vulkan 1.1, srcImage and dstImage **must** have the same VkImageType
- VUID-vkCmdCopyImage-apiVersion-08969
  If the VK_KHR_maintenance1 extension is not enabled, and VkPhysicalDeviceProperties::apiVersion is less than Vulkan 1.1, srcImage or dstImage is of type VK_IMAGE_TYPE_2D, then for each element of pRegions, extent.depth **must** be 1
- VUID-vkCmdCopyImage-srcImage-07743
  If srcImage and dstImage have a different VkImageType, and maintenance5 is not enabled, one **must** be VK_IMAGE_TYPE_3D and the other **must** be VK_IMAGE_TYPE_2D
- VUID-vkCmdCopyImage-srcImage-08793
  If srcImage and dstImage have the same VkImageType, for each element of pRegions, if neither of the layerCount members of srcSubresource or dstSubresource are VK_REMAINING_ARRAY_LAYERS, the layerCount members of srcSubresource or dstSubresource **must** match

- VUID-vkCmdCopyImage-srcImage-08794
  If srcImage and dstImage have the same VkImageType, and one of the layerCount members of srcSubresource or dstSubresource is VK_REMAINING_ARRAY_LAYERS, the other member **must** be either VK_REMAINING_ARRAY_LAYERS or equal to the arrayLayers member of the VkImageCreateInfo used to create the image minus baseArrayLayer
- VUID-vkCmdCopyImage-srcImage-01790
  If srcImage and dstImage are both of type VK_IMAGE_TYPE_2D, then for each element of pRegions, extent.depth **must** be 1
- VUID-vkCmdCopyImage-srcImage-01791
  If srcImage is of type VK_IMAGE_TYPE_2D, and dstImage is of type VK_IMAGE_TYPE_3D, then for each element of pRegions, extent.depth **must** equal srcSubresource.layerCount
- VUID-vkCmdCopyImage-dstImage-01792
  If dstImage is of type VK_IMAGE_TYPE_2D, and srcImage is of type VK_IMAGE_TYPE_3D, then for each element of pRegions, extent.depth **must** equal dstSubresource.layerCount
- VUID-vkCmdCopyImage-dstOffset-00150
  For each element of pRegions, dstOffset.x and (extent.width + dstOffset.x) **must** both be greater than or equal to 0 and less than or equal to the width of the specified dstSubresource of dstImage
- VUID-vkCmdCopyImage-dstOffset-00151
  For each element of pRegions, dstOffset.y and (extent.height + dstOffset.y) **must** both be greater than or equal to 0 and less than or equal to the height of the specified dstSubresource of dstImage
- VUID-vkCmdCopyImage-dstImage-00152
  If dstImage is of type VK_IMAGE_TYPE_1D, then for each element of pRegions, dstOffset.y **must** be 0 and extent.height **must** be 1
- VUID-vkCmdCopyImage-dstOffset-00153
  If dstImage is of type VK_IMAGE_TYPE_3D, then for each element of pRegions, dstOffset.z and (extent.depth + dstOffset.z) **must** both be greater than or equal to 0 and less than or equal to the depth of the specified dstSubresource of dstImage
- VUID-vkCmdCopyImage-pRegions-07278
  For each element of pRegions, srcOffset.x **must** be a multiple of the texel block extent width of the VkFormat of srcImage
- VUID-vkCmdCopyImage-pRegions-07279
  For each element of pRegions, srcOffset.y **must** be a multiple of the texel block extent height of the VkFormat of srcImage
- VUID-vkCmdCopyImage-pRegions-07280
  For each element of pRegions, srcOffset.z **must** be a multiple of the texel block extent depth of the VkFormat of srcImage
- VUID-vkCmdCopyImage-pRegions-07281
  For each element of pRegions, dstOffset.x **must** be a multiple of the texel block extent width of the VkFormat of dstImage
- VUID-vkCmdCopyImage-pRegions-07282
  For each element of pRegions, dstOffset.y **must** be a multiple of the texel block extent

height of the VkFormat of dstImage

- VUID-vkCmdCopyImage-pRegions-07283
  For each element of pRegions, dstOffset.z **must** be a multiple of the texel block extent depth of the VkFormat of dstImage
- VUID-vkCmdCopyImage-srcImage-01728
  For each element of pRegions, if the sum of srcOffset.x and extent.width does not equal the width of the subresource specified by srcSubresource, extent.width **must** be a multiple of the texel block extent width of the VkFormat of srcImage
- VUID-vkCmdCopyImage-srcImage-01729
  For each element of pRegions, if the sum of srcOffset.y and extent.height does not equal the height of the subresource specified by srcSubresource, extent.height **must** be a multiple of the texel block extent height of the VkFormat of srcImage
- VUID-vkCmdCopyImage-srcImage-01730
  For each element of pRegions, if the sum of srcOffset.z and extent.depth does not equal the depth of the subresource specified by srcSubresource, extent.depth **must** be a multiple of the texel block extent depth of the VkFormat of srcImage
- VUID-vkCmdCopyImage-dstImage-01732
  For each element of pRegions, if the sum of dstOffset.x and extent.width does not equal the width of the subresource specified by dstSubresource, extent.width **must** be a multiple of the texel block extent width of the VkFormat of dstImage
- VUID-vkCmdCopyImage-dstImage-01733
  For each element of pRegions, if the sum of dstOffset.y and extent.height does not equal the height of the subresource specified by dstSubresource, extent.height **must** be a multiple of the texel block extent height of the VkFormat of dstImage
- VUID-vkCmdCopyImage-dstImage-01734
  For each element of pRegions, if the sum of dstOffset.z and extent.depth does not equal the depth of the subresource specified by dstSubresource, extent.depth **must** be a multiple of the texel block extent depth of the VkFormat of dstImage
- VUID-vkCmdCopyImage-aspect-06662
  If the aspect member of any element of pRegions includes any flag other than VK_IMAGE_ASPECT_STENCIL_BIT or srcImage was not created with separate stencil usage, VK_IMAGE_USAGE_TRANSFER_SRC_BIT **must** have been included in the VkImageCreateInfo::usage used to create srcImage
- VUID-vkCmdCopyImage-aspect-06663
  If the aspect member of any element of pRegions includes any flag other than VK_IMAGE_ASPECT_STENCIL_BIT or dstImage was not created with separate stencil usage, VK_IMAGE_USAGE_TRANSFER_DST_BIT **must** have been included in the VkImageCreateInfo::usage used to create dstImage
- VUID-vkCmdCopyImage-aspect-06664
  If the aspect member of any element of pRegions includes VK_IMAGE_ASPECT_STENCIL_BIT, and srcImage was created with separate stencil usage, VK_IMAGE_USAGE_TRANSFER_SRC_BIT **must** have been included in the VkImageStencilUsageCreateInfo::stencilUsage used to create srcImage

- VUID-vkCmdCopyImage-aspect-06665
  If the `aspect` member of any element of `pRegions` includes `VK_IMAGE_ASPECT_STENCIL_BIT`, and `dstImage` was created with separate stencil usage, `VK_IMAGE_USAGE_TRANSFER_DST_BIT` **must** have been included in the VkImageStencilUsageCreateInfo::`stencilUsage` used to create `dstImage`

- VUID-vkCmdCopyImage-srcImage-07966
  If `srcImage` is non-sparse then the image or the specified *disjoint* plane **must** be bound completely and contiguously to a single `VkDeviceMemory` object
- VUID-vkCmdCopyImage-srcSubresource-07967
  The `srcSubresource.mipLevel` member of each element of `pRegions` **must** be less than the `mipLevels` specified in VkImageCreateInfo when `srcImage` was created
- VUID-vkCmdCopyImage-srcSubresource-07968
  If `srcSubresource.layerCount` is not `VK_REMAINING_ARRAY_LAYERS`, `srcSubresource.baseArrayLayer` + `srcSubresource.layerCount` of each element of `pRegions` **must** be less than or equal to the `arrayLayers` specified in VkImageCreateInfo when `srcImage` was created
- VUID-vkCmdCopyImage-srcImage-07969
  `srcImage` **must** not have been created with `flags` containing `VK_IMAGE_CREATE_SUBSAMPLED_BIT_EXT`

- VUID-vkCmdCopyImage-dstImage-07966
  If `dstImage` is non-sparse then the image or the specified *disjoint* plane **must** be bound completely and contiguously to a single `VkDeviceMemory` object
- VUID-vkCmdCopyImage-dstSubresource-07967
  The `dstSubresource.mipLevel` member of each element of `pRegions` **must** be less than the `mipLevels` specified in VkImageCreateInfo when `dstImage` was created
- VUID-vkCmdCopyImage-dstSubresource-07968
  If `dstSubresource.layerCount` is not `VK_REMAINING_ARRAY_LAYERS`, `dstSubresource.baseArrayLayer` + `dstSubresource.layerCount` of each element of `pRegions` **must** be less than or equal to the `arrayLayers` specified in VkImageCreateInfo when `dstImage` was created
- VUID-vkCmdCopyImage-dstImage-07969
  `dstImage` **must** not have been created with `flags` containing `VK_IMAGE_CREATE_SUBSAMPLED_BIT_EXT`

## Valid Usage (Implicit)

- VUID-vkCmdCopyImage-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdCopyImage-srcImage-parameter
  `srcImage` **must** be a valid VkImage handle
- VUID-vkCmdCopyImage-srcImageLayout-parameter
  `srcImageLayout` **must** be a valid VkImageLayout value

- VUID-vkCmdCopyImage-dstImage-parameter
  `dstImage` **must** be a valid VkImage handle
- VUID-vkCmdCopyImage-dstImageLayout-parameter
  `dstImageLayout` **must** be a valid VkImageLayout value
- VUID-vkCmdCopyImage-pRegions-parameter
  `pRegions` **must** be a valid pointer to an array of `regionCount` valid VkImageCopy structures
- VUID-vkCmdCopyImage-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdCopyImage-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support transfer, graphics, or compute operations
- VUID-vkCmdCopyImage-renderpass
  This command **must** only be called outside of a render pass instance
- VUID-vkCmdCopyImage-videocoding
  This command **must** only be called outside of a video coding scope
- VUID-vkCmdCopyImage-regionCount-arraylength
  `regionCount` **must** be greater than `0`
- VUID-vkCmdCopyImage-commonparent
  Each of `commandBuffer`, `dstImage`, and `srcImage` **must** have been created, allocated, or retrieved from the same VkDevice

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Outside | Outside | Transfer<br>Graphics<br>Compute | Action |

The `VkImageCopy` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkImageCopy {
    VkImageSubresourceLayers    srcSubresource;
    VkOffset3D                  srcOffset;
    VkImageSubresourceLayers    dstSubresource;
    VkOffset3D                  dstOffset;
    VkExtent3D                  extent;
} VkImageCopy;
```

- `srcSubresource` and `dstSubresource` are VkImageSubresourceLayers structures specifying the image subresources of the images used for the source and destination image data, respectively.
- `srcOffset` and `dstOffset` select the initial `x`, `y`, and `z` offsets in texels of the sub-regions of the source and destination image data.
- `extent` is the size in texels of the image to copy in `width`, `height` and `depth`.

## Valid Usage

- VUID-VkImageCopy-apiVersion-07940
  If the VK_KHR_sampler_ycbcr_conversion extension is not enabled, and VkPhysicalDeviceProperties::`apiVersion` is less than Vulkan 1.1, the `aspectMask` member of `srcSubresource` and `dstSubresource` **must** match
- VUID-VkImageCopy-apiVersion-07941
  If the VK_KHR_maintenance1 extension is not enabled, and VkPhysicalDeviceProperties::`apiVersion` is less than Vulkan 1.1, the `layerCount` member of `srcSubresource` and `dstSubresource` **must** match
- VUID-VkImageCopy-extent-06668
  `extent.width` **must** not be 0
- VUID-VkImageCopy-extent-06669
  `extent.height` **must** not be 0
- VUID-VkImageCopy-extent-06670
  `extent.depth` **must** not be 0

## Valid Usage (Implicit)

- VUID-VkImageCopy-srcSubresource-parameter
  `srcSubresource` **must** be a valid VkImageSubresourceLayers structure
- VUID-VkImageCopy-dstSubresource-parameter
  `dstSubresource` **must** be a valid VkImageSubresourceLayers structure

The `VkImageSubresourceLayers` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkImageSubresourceLayers {
    VkImageAspectFlags    aspectMask;
    uint32_t              mipLevel;
    uint32_t              baseArrayLayer;
    uint32_t              layerCount;
} VkImageSubresourceLayers;
```

- `aspectMask` is a combination of VkImageAspectFlagBits, selecting the color, depth and/or stencil aspects to be copied.
- `mipLevel` is the mipmap level to copy
- `baseArrayLayer` and `layerCount` are the starting layer and number of layers to copy.

### Valid Usage

- VUID-VkImageSubresourceLayers-aspectMask-00167
  If `aspectMask` contains `VK_IMAGE_ASPECT_COLOR_BIT`, it **must** not contain either of `VK_IMAGE_ASPECT_DEPTH_BIT` or `VK_IMAGE_ASPECT_STENCIL_BIT`
- VUID-VkImageSubresourceLayers-aspectMask-00168
  `aspectMask` **must** not contain `VK_IMAGE_ASPECT_METADATA_BIT`
- VUID-VkImageSubresourceLayers-aspectMask-02247
  `aspectMask` **must** not include `VK_IMAGE_ASPECT_MEMORY_PLANE_i_BIT_EXT` for any index *i*
- VUID-VkImageSubresourceLayers-layerCount-09243
  If the `maintenance5` feature is not enabled, `layerCount` **must** not be `VK_REMAINING_ARRAY_LAYERS`
- VUID-VkImageSubresourceLayers-layerCount-01700
  If `layerCount` is not `VK_REMAINING_ARRAY_LAYERS`, it **must** be greater than 0

### Valid Usage (Implicit)

- VUID-VkImageSubresourceLayers-aspectMask-parameter
  `aspectMask` **must** be a valid combination of VkImageAspectFlagBits values
- VUID-VkImageSubresourceLayers-aspectMask-requiredbitmask
  `aspectMask` **must** not be `0`

A more extensible version of the copy image command is defined below.

To copy data between image objects, call:

```
// Provided by VK_KHR_copy_commands2
void vkCmdCopyImage2KHR(
    VkCommandBuffer                             commandBuffer,
    const VkCopyImageInfo2*                     pCopyImageInfo);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `pCopyImageInfo` is a pointer to a VkCopyImageInfo2 structure describing the copy parameters.

This command is functionally identical to vkCmdCopyImage, but includes extensible sub-structures that include `sType` and `pNext` parameters, allowing them to be more easily extended.

## Valid Usage

- VUID-vkCmdCopyImage2-commandBuffer-01825
  If `commandBuffer` is an unprotected command buffer and `protectedNoFault` is not supported, `srcImage` **must** not be a protected image
- VUID-vkCmdCopyImage2-commandBuffer-01826
  If `commandBuffer` is an unprotected command buffer and `protectedNoFault` is not supported, `dstImage` **must** not be a protected image
- VUID-vkCmdCopyImage2-commandBuffer-01827
  If `commandBuffer` is a protected command buffer and `protectedNoFault` is not supported, `dstImage` **must** not be an unprotected image

## Valid Usage (Implicit)

- VUID-vkCmdCopyImage2-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdCopyImage2-pCopyImageInfo-parameter
  `pCopyImageInfo` **must** be a valid pointer to a valid VkCopyImageInfo2 structure
- VUID-vkCmdCopyImage2-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdCopyImage2-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support transfer, graphics, or compute operations
- VUID-vkCmdCopyImage2-renderpass
  This command **must** only be called outside of a render pass instance
- VUID-vkCmdCopyImage2-videocoding
  This command **must** only be called outside of a video coding scope

### Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

### Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Outside | Outside | Transfer<br>Graphics<br>Compute | Action |

The `VkCopyImageInfo2` structure is defined as:

```
typedef struct VkCopyImageInfo2 {
    VkStructureType        sType;
    const void*            pNext;
    VkImage                srcImage;
    VkImageLayout          srcImageLayout;
    VkImage                dstImage;
    VkImageLayout          dstImageLayout;
    uint32_t               regionCount;
    const VkImageCopy2*    pRegions;
} VkCopyImageInfo2;
```

or the equivalent

```
// Provided by VK_KHR_copy_commands2
typedef VkCopyImageInfo2 VkCopyImageInfo2KHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `srcImage` is the source image.
- `srcImageLayout` is the current layout of the source image subresource.
- `dstImage` is the destination image.
- `dstImageLayout` is the current layout of the destination image subresource.
- `regionCount` is the number of regions to copy.
- `pRegions` is a pointer to an array of VkImageCopy2 structures specifying the regions to copy.

## Valid Usage

- VUID-VkCopyImageInfo2-pRegions-00124
  The union of all source regions, and the union of all destination regions, specified by the elements of `pRegions`, **must** not overlap in memory
- VUID-VkCopyImageInfo2-srcImage-01995
  The format features of `srcImage` **must** contain `VK_FORMAT_FEATURE_TRANSFER_SRC_BIT`
- VUID-VkCopyImageInfo2-srcImageLayout-00128
  `srcImageLayout` **must** specify the layout of the image subresources of `srcImage` specified in `pRegions` at the time this command is executed on a `VkDevice`
- VUID-VkCopyImageInfo2-srcImageLayout-01917
  `srcImageLayout` **must** be `VK_IMAGE_LAYOUT_SHARED_PRESENT_KHR`, `VK_IMAGE_LAYOUT_TRANSFER_SRC_OPTIMAL`, or `VK_IMAGE_LAYOUT_GENERAL`
- VUID-VkCopyImageInfo2-srcImage-09460
  If `srcImage` and `dstImage` are the same, and any elements of `pRegions` contains the `srcSubresource` and `dstSubresource` with matching `mipLevel` and overlapping array layers, then the `srcImageLayout` and `dstImageLayout` **must** be `VK_IMAGE_LAYOUT_GENERAL` or `VK_IMAGE_LAYOUT_SHARED_PRESENT_KHR`
- VUID-VkCopyImageInfo2-dstImage-01996
  The format features of `dstImage` **must** contain `VK_FORMAT_FEATURE_TRANSFER_DST_BIT`
- VUID-VkCopyImageInfo2-dstImageLayout-00133
  `dstImageLayout` **must** specify the layout of the image subresources of `dstImage` specified in `pRegions` at the time this command is executed on a `VkDevice`
- VUID-VkCopyImageInfo2-dstImageLayout-01395
  `dstImageLayout` **must** be `VK_IMAGE_LAYOUT_SHARED_PRESENT_KHR`, `VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL`, or `VK_IMAGE_LAYOUT_GENERAL`
- VUID-VkCopyImageInfo2-srcImage-01548
  If the VkFormat of each of `srcImage` and `dstImage` is not a *multi-planar format*, the VkFormat of each of `srcImage` and `dstImage` **must** be size-compatible
- VUID-VkCopyImageInfo2-None-01549
  In a copy to or from a plane of a multi-planar image, the VkFormat of the image and plane **must** be compatible according to the description of compatible planes for the plane being copied
- VUID-VkCopyImageInfo2-srcImage-09247
  If the VkFormat of each of `srcImage` and `dstImage` is a compressed image format, the formats **must** have the same texel block extent
- VUID-VkCopyImageInfo2-srcImage-00136
  The sample count of `srcImage` and `dstImage` **must** match
- VUID-VkCopyImageInfo2-srcOffset-01783
  The `srcOffset` and `extent` members of each element of `pRegions` **must** respect the image transfer granularity requirements of `commandBuffer`'s command pool's queue family, as described in VkQueueFamilyProperties

- VUID-VkCopyImageInfo2-dstOffset-01784
  The `dstOffset` and `extent` members of each element of `pRegions` **must** respect the image transfer granularity requirements of `commandBuffer`'s command pool's queue family, as described in VkQueueFamilyProperties
- VUID-VkCopyImageInfo2-srcImage-01551
  If neither `srcImage` nor `dstImage` has a multi-planar image format then for each element of `pRegions`, `srcSubresource.aspectMask` and `dstSubresource.aspectMask` **must** match
- VUID-VkCopyImageInfo2-srcImage-08713
  If `srcImage` has a multi-planar image format, then for each element of `pRegions`, `srcSubresource.aspectMask` **must** be a single valid multi-planar aspect mask bit
- VUID-VkCopyImageInfo2-dstImage-08714
  If `dstImage` has a multi-planar image format, then for each element of `pRegions`, `dstSubresource.aspectMask` **must** be a single valid multi-planar aspect mask bit
- VUID-VkCopyImageInfo2-srcImage-01556
  If `srcImage` has a multi-planar image format and the `dstImage` does not have a multi-planar image format, then for each element of `pRegions`, `dstSubresource.aspectMask` **must** be `VK_IMAGE_ASPECT_COLOR_BIT`
- VUID-VkCopyImageInfo2-dstImage-01557
  If `dstImage` has a multi-planar image format and the `srcImage` does not have a multi-planar image format, then for each element of `pRegions`, `srcSubresource.aspectMask` **must** be `VK_IMAGE_ASPECT_COLOR_BIT`
- VUID-VkCopyImageInfo2-apiVersion-07932
  If the VK_KHR_maintenance1 extension is not enabled, or VkPhysicalDeviceProperties ::`apiVersion` is less than Vulkan 1.1, and either `srcImage` or `dstImage` is of type `VK_IMAGE_TYPE_3D`, then for each element of `pRegions`, `srcSubresource.baseArrayLayer` and `dstSubresource.baseArrayLayer` **must** both be `0`, and `srcSubresource.layerCount` and `dstSubresource.layerCount` **must** both be `1`
- VUID-VkCopyImageInfo2-srcImage-04443
  If `srcImage` is of type `VK_IMAGE_TYPE_3D`, then for each element of `pRegions`, `srcSubresource.baseArrayLayer` **must** be `0` and `srcSubresource.layerCount` **must** be `1`
- VUID-VkCopyImageInfo2-dstImage-04444
  If `dstImage` is of type `VK_IMAGE_TYPE_3D`, then for each element of `pRegions`, `dstSubresource.baseArrayLayer` **must** be `0` and `dstSubresource.layerCount` **must** be `1`
- VUID-VkCopyImageInfo2-aspectMask-00142
  For each element of `pRegions`, `srcSubresource.aspectMask` **must** specify aspects present in `srcImage`
- VUID-VkCopyImageInfo2-aspectMask-00143
  For each element of `pRegions`, `dstSubresource.aspectMask` **must** specify aspects present in `dstImage`
- VUID-VkCopyImageInfo2-srcOffset-00144
  For each element of `pRegions`, `srcOffset.x` and (`extent.width` + `srcOffset.x`) **must** both be greater than or equal to `0` and less than or equal to the width of the specified `srcSubresource` of `srcImage`

- VUID-VkCopyImageInfo2-srcOffset-00145
  For each element of pRegions, srcOffset.y and (extent.height + srcOffset.y) **must** both be greater than or equal to 0 and less than or equal to the height of the specified srcSubresource of srcImage
- VUID-VkCopyImageInfo2-srcImage-00146
  If srcImage is of type VK_IMAGE_TYPE_1D, then for each element of pRegions, srcOffset.y **must** be 0 and extent.height **must** be 1
- VUID-VkCopyImageInfo2-srcOffset-00147
  If srcImage is of type VK_IMAGE_TYPE_3D, then for each element of pRegions, srcOffset.z and (extent.depth + srcOffset.z) **must** both be greater than or equal to 0 and less than or equal to the depth of the specified srcSubresource of srcImage
- VUID-VkCopyImageInfo2-srcImage-01785
  If srcImage is of type VK_IMAGE_TYPE_1D, then for each element of pRegions, srcOffset.z **must** be 0 and extent.depth **must** be 1
- VUID-VkCopyImageInfo2-dstImage-01786
  If dstImage is of type VK_IMAGE_TYPE_1D, then for each element of pRegions, dstOffset.z **must** be 0 and extent.depth **must** be 1
- VUID-VkCopyImageInfo2-srcImage-01787
  If srcImage is of type VK_IMAGE_TYPE_2D, then for each element of pRegions, srcOffset.z **must** be 0
- VUID-VkCopyImageInfo2-dstImage-01788
  If dstImage is of type VK_IMAGE_TYPE_2D, then for each element of pRegions, dstOffset.z **must** be 0
- VUID-VkCopyImageInfo2-apiVersion-07933
  If the VK_KHR_maintenance1 extension is not enabled, and VkPhysicalDeviceProperties::apiVersion is less than Vulkan 1.1, srcImage and dstImage **must** have the same VkImageType
- VUID-VkCopyImageInfo2-apiVersion-08969
  If the VK_KHR_maintenance1 extension is not enabled, and VkPhysicalDeviceProperties::apiVersion is less than Vulkan 1.1, srcImage or dstImage is of type VK_IMAGE_TYPE_2D, then for each element of pRegions, extent.depth **must** be 1
- VUID-VkCopyImageInfo2-srcImage-07743
  If srcImage and dstImage have a different VkImageType, and maintenance5 is not enabled, one **must** be VK_IMAGE_TYPE_3D and the other **must** be VK_IMAGE_TYPE_2D
- VUID-VkCopyImageInfo2-srcImage-08793
  If srcImage and dstImage have the same VkImageType, for each element of pRegions, if neither of the layerCount members of srcSubresource or dstSubresource are VK_REMAINING_ARRAY_LAYERS, the layerCount members of srcSubresource or dstSubresource **must** match
- VUID-VkCopyImageInfo2-srcImage-08794
  If srcImage and dstImage have the same VkImageType, and one of the layerCount members of srcSubresource or dstSubresource is VK_REMAINING_ARRAY_LAYERS, the other member **must** be either VK_REMAINING_ARRAY_LAYERS or equal to the arrayLayers member of the

VkImageCreateInfo used to create the image minus baseArrayLayer

- VUID-VkCopyImageInfo2-srcImage-01790
  If srcImage and dstImage are both of type VK_IMAGE_TYPE_2D, then for each element of pRegions, extent.depth **must** be 1
- VUID-VkCopyImageInfo2-srcImage-01791
  If srcImage is of type VK_IMAGE_TYPE_2D, and dstImage is of type VK_IMAGE_TYPE_3D, then for each element of pRegions, extent.depth **must** equal srcSubresource.layerCount
- VUID-VkCopyImageInfo2-dstImage-01792
  If dstImage is of type VK_IMAGE_TYPE_2D, and srcImage is of type VK_IMAGE_TYPE_3D, then for each element of pRegions, extent.depth **must** equal dstSubresource.layerCount
- VUID-VkCopyImageInfo2-dstOffset-00150
  For each element of pRegions, dstOffset.x and (extent.width + dstOffset.x) **must** both be greater than or equal to 0 and less than or equal to the width of the specified dstSubresource of dstImage
- VUID-VkCopyImageInfo2-dstOffset-00151
  For each element of pRegions, dstOffset.y and (extent.height + dstOffset.y) **must** both be greater than or equal to 0 and less than or equal to the height of the specified dstSubresource of dstImage
- VUID-VkCopyImageInfo2-dstImage-00152
  If dstImage is of type VK_IMAGE_TYPE_1D, then for each element of pRegions, dstOffset.y **must** be 0 and extent.height **must** be 1
- VUID-VkCopyImageInfo2-dstOffset-00153
  If dstImage is of type VK_IMAGE_TYPE_3D, then for each element of pRegions, dstOffset.z and (extent.depth + dstOffset.z) **must** both be greater than or equal to 0 and less than or equal to the depth of the specified dstSubresource of dstImage
- VUID-VkCopyImageInfo2-pRegions-07278
  For each element of pRegions, srcOffset.x **must** be a multiple of the texel block extent width of the VkFormat of srcImage
- VUID-VkCopyImageInfo2-pRegions-07279
  For each element of pRegions, srcOffset.y **must** be a multiple of the texel block extent height of the VkFormat of srcImage
- VUID-VkCopyImageInfo2-pRegions-07280
  For each element of pRegions, srcOffset.z **must** be a multiple of the texel block extent depth of the VkFormat of srcImage
- VUID-VkCopyImageInfo2-pRegions-07281
  For each element of pRegions, dstOffset.x **must** be a multiple of the texel block extent width of the VkFormat of dstImage
- VUID-VkCopyImageInfo2-pRegions-07282
  For each element of pRegions, dstOffset.y **must** be a multiple of the texel block extent height of the VkFormat of dstImage
- VUID-VkCopyImageInfo2-pRegions-07283
  For each element of pRegions, dstOffset.z **must** be a multiple of the texel block extent depth of the VkFormat of dstImage

- VUID-VkCopyImageInfo2-srcImage-01728
  For each element of `pRegions`, if the sum of `srcOffset.x` and `extent.width` does not equal the width of the subresource specified by `srcSubresource`, `extent.width` **must** be a multiple of the texel block extent width of the VkFormat of `srcImage`
- VUID-VkCopyImageInfo2-srcImage-01729
  For each element of `pRegions`, if the sum of `srcOffset.y` and `extent.height` does not equal the height of the subresource specified by `srcSubresource`, `extent.height` **must** be a multiple of the texel block extent height of the VkFormat of `srcImage`
- VUID-VkCopyImageInfo2-srcImage-01730
  For each element of `pRegions`, if the sum of `srcOffset.z` and `extent.depth` does not equal the depth of the subresource specified by `srcSubresource`, `extent.depth` **must** be a multiple of the texel block extent depth of the VkFormat of `srcImage`
- VUID-VkCopyImageInfo2-dstImage-01732
  For each element of `pRegions`, if the sum of `dstOffset.x` and `extent.width` does not equal the width of the subresource specified by `dstSubresource`, `extent.width` **must** be a multiple of the texel block extent width of the VkFormat of `dstImage`
- VUID-VkCopyImageInfo2-dstImage-01733
  For each element of `pRegions`, if the sum of `dstOffset.y` and `extent.height` does not equal the height of the subresource specified by `dstSubresource`, `extent.height` **must** be a multiple of the texel block extent height of the VkFormat of `dstImage`
- VUID-VkCopyImageInfo2-dstImage-01734
  For each element of `pRegions`, if the sum of `dstOffset.z` and `extent.depth` does not equal the depth of the subresource specified by `dstSubresource`, `extent.depth` **must** be a multiple of the texel block extent depth of the VkFormat of `dstImage`
- VUID-VkCopyImageInfo2-aspect-06662
  If the `aspect` member of any element of `pRegions` includes any flag other than `VK_IMAGE_ASPECT_STENCIL_BIT` or `srcImage` was not created with separate stencil usage, `VK_IMAGE_USAGE_TRANSFER_SRC_BIT` **must** have been included in the VkImageCreateInfo::`usage` used to create `srcImage`
- VUID-VkCopyImageInfo2-aspect-06663
  If the `aspect` member of any element of `pRegions` includes any flag other than `VK_IMAGE_ASPECT_STENCIL_BIT` or `dstImage` was not created with separate stencil usage, `VK_IMAGE_USAGE_TRANSFER_DST_BIT` **must** have been included in the VkImageCreateInfo::`usage` used to create `dstImage`
- VUID-VkCopyImageInfo2-aspect-06664
  If the `aspect` member of any element of `pRegions` includes `VK_IMAGE_ASPECT_STENCIL_BIT`, and `srcImage` was created with separate stencil usage, `VK_IMAGE_USAGE_TRANSFER_SRC_BIT` **must** have been included in the VkImageStencilUsageCreateInfo::`stencilUsage` used to create `srcImage`
- VUID-VkCopyImageInfo2-aspect-06665
  If the `aspect` member of any element of `pRegions` includes `VK_IMAGE_ASPECT_STENCIL_BIT`, and `dstImage` was created with separate stencil usage, `VK_IMAGE_USAGE_TRANSFER_DST_BIT` **must** have been included in the VkImageStencilUsageCreateInfo::`stencilUsage` used to create `dstImage`

- VUID-VkCopyImageInfo2-srcImage-07966
  If srcImage is non-sparse then the image or the specified *disjoint* plane **must** be bound completely and contiguously to a single VkDeviceMemory object
- VUID-VkCopyImageInfo2-srcSubresource-07967
  The srcSubresource.mipLevel member of each element of pRegions **must** be less than the mipLevels specified in VkImageCreateInfo when srcImage was created
- VUID-VkCopyImageInfo2-srcSubresource-07968
  If srcSubresource.layerCount is not VK_REMAINING_ARRAY_LAYERS, srcSubresource.baseArrayLayer + srcSubresource.layerCount of each element of pRegions **must** be less than or equal to the arrayLayers specified in VkImageCreateInfo when srcImage was created
- VUID-VkCopyImageInfo2-srcImage-07969
  srcImage **must** not have been created with flags containing VK_IMAGE_CREATE_SUBSAMPLED_BIT_EXT
- VUID-VkCopyImageInfo2-dstImage-07966
  If dstImage is non-sparse then the image or the specified *disjoint* plane **must** be bound completely and contiguously to a single VkDeviceMemory object
- VUID-VkCopyImageInfo2-dstSubresource-07967
  The dstSubresource.mipLevel member of each element of pRegions **must** be less than the mipLevels specified in VkImageCreateInfo when dstImage was created
- VUID-VkCopyImageInfo2-dstSubresource-07968
  If dstSubresource.layerCount is not VK_REMAINING_ARRAY_LAYERS, dstSubresource.baseArrayLayer + dstSubresource.layerCount of each element of pRegions **must** be less than or equal to the arrayLayers specified in VkImageCreateInfo when dstImage was created
- VUID-VkCopyImageInfo2-dstImage-07969
  dstImage **must** not have been created with flags containing VK_IMAGE_CREATE_SUBSAMPLED_BIT_EXT

## Valid Usage (Implicit)

- VUID-VkCopyImageInfo2-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_COPY_IMAGE_INFO_2
- VUID-VkCopyImageInfo2-pNext-pNext
  pNext **must** be NULL
- VUID-VkCopyImageInfo2-srcImage-parameter
  srcImage **must** be a valid VkImage handle
- VUID-VkCopyImageInfo2-srcImageLayout-parameter
  srcImageLayout **must** be a valid VkImageLayout value
- VUID-VkCopyImageInfo2-dstImage-parameter
  dstImage **must** be a valid VkImage handle
- VUID-VkCopyImageInfo2-dstImageLayout-parameter

dstImageLayout **must** be a valid VkImageLayout value

- VUID-VkCopyImageInfo2-pRegions-parameter
  pRegions **must** be a valid pointer to an array of regionCount valid VkImageCopy2 structures
- VUID-VkCopyImageInfo2-regionCount-arraylength
  regionCount **must** be greater than 0
- VUID-VkCopyImageInfo2-commonparent
  Both of dstImage, and srcImage **must** have been created, allocated, or retrieved from the same VkDevice

The VkImageCopy2 structure is defined as:

```
typedef struct VkImageCopy2 {
    VkStructureType             sType;
    const void*                 pNext;
    VkImageSubresourceLayers    srcSubresource;
    VkOffset3D                  srcOffset;
    VkImageSubresourceLayers    dstSubresource;
    VkOffset3D                  dstOffset;
    VkExtent3D                  extent;
} VkImageCopy2;
```

or the equivalent

```
// Provided by VK_KHR_copy_commands2
typedef VkImageCopy2 VkImageCopy2KHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- srcSubresource and dstSubresource are VkImageSubresourceLayers structures specifying the image subresources of the images used for the source and destination image data, respectively.
- srcOffset and dstOffset select the initial x, y, and z offsets in texels of the sub-regions of the source and destination image data.
- extent is the size in texels of the image to copy in width, height and depth.

### Valid Usage

- VUID-VkImageCopy2-apiVersion-07940
  If the VK_KHR_sampler_ycbcr_conversion extension is not enabled, and VkPhysicalDeviceProperties::apiVersion is less than Vulkan 1.1, the aspectMask member of srcSubresource and dstSubresource **must** match
- VUID-VkImageCopy2-apiVersion-07941
  If the VK_KHR_maintenance1 extension is not enabled, and VkPhysicalDeviceProperties

::apiVersion is less than Vulkan 1.1, the layerCount member of srcSubresource and dstSubresource **must** match

- VUID-VkImageCopy2-extent-06668
  extent.width **must** not be 0
- VUID-VkImageCopy2-extent-06669
  extent.height **must** not be 0
- VUID-VkImageCopy2-extent-06670
  extent.depth **must** not be 0

**Valid Usage (Implicit)**

- VUID-VkImageCopy2-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_IMAGE_COPY_2
- VUID-VkImageCopy2-pNext-pNext
  pNext **must** be NULL
- VUID-VkImageCopy2-srcSubresource-parameter
  srcSubresource **must** be a valid VkImageSubresourceLayers structure
- VUID-VkImageCopy2-dstSubresource-parameter
  dstSubresource **must** be a valid VkImageSubresourceLayers structure

## 20.3. Copying Data Between Buffers and Images

Data **can** be copied between buffers and images, enabling applications to load and store data between images and application-defined offsets in buffer memory.

When copying between a buffer and an image, texels in the image and bytes in the buffer are accessed as follows.

Texels at each coordinate (x,y,z,layer) in the image subresource are accessed, where:

x is in the range [imageOffset.x, imageOffset.x + imageExtent.width),

y is in the range [imageOffset.y, imageOffset.y + imageExtent.height),

z is in the range [imageOffset.z, imageOffset.z + imageExtent.depth),

layer is in the range [eq]#[imageSubresource.baseArrayLayer, imageSubresource.baseArrayLayer + imageSubresource.layerCount)

For each (x,y,z,layer) coordinate in the image, bytes in the buffer are accessed at offsets in the range

[texelOffset, texelOffset + blockSize), where:

texelOffset = `bufferOffset` + (⌊x / blockWidth⌋ × blockSize) + (⌊y / blockHeight⌋ × rowExtent) + (⌊z / blockDepth⌋ × sliceExtent) + (layer × layerExtent)

rowExtent = ⌈ max(`bufferRowLength`, `imageExtent.width`) / blockWidth ⌉ × blockSize

sliceExtent = ⌈ max(`bufferImageHeight`, `imageExtent.height`) / blockHeight ⌉ × rowExtent

layerExtent = ⌈ `imageExtent.depth` / blockDepth ⌉ × sliceExtent

and where blockSize, blockWidth, blockHeight, and blockDepth are the texel block size and extents of the image's format.

If a rotation is specified by VkCopyCommandTransformInfoQCOM, the 2D region of the image being addressed is rotated around the offset, and texels at each coordinate (x',y',z',layer) are accessed in the image subresource instead, where:

- If `VK_SURFACE_TRANSFORM_IDENTITY_BIT_KHR` is specified, no rotation is performed:

  x' is in the same range as x

  y' is in the same range as y

- If `VK_SURFACE_TRANSFORM_ROTATE_90_BIT_KHR` is specified

  x' is in the range [`imageOffset.x` - `imageExtent.height`, `imageOffset.x`)

  y' is in the range [`imageOffset.y`, `imageOffset.y` + `imageExtent.width`)

- If `VK_SURFACE_TRANSFORM_ROTATE_180_BIT_KHR` is specified:

  x' is in the range [`imageOffset.x` - `imageExtent.width`, `imageOffset.x`)

  y' is in the range [`imageOffset.y` - `imageExtent.height`, `imageOffset.y`)

- If `VK_SURFACE_TRANSFORM_ROTATE_270_BIT_KHR` is specified:

x' is in the range [imageOffset.x, imageOffset.x + imageExtent.height)

y' is in the range [imageOffset.y - imageExtent.width, imageOffset.y)

Buffer addressing calculations are unaffected by this rotation.

When copying between a buffer and the depth or stencil aspect of an image, data in the buffer is assumed to be laid out as separate planes rather than interleaved. Addressing calculations are thus performed for a different format than the base image, according to the aspect, as described in the following table:

*Table 30. Depth/Stencil Aspect Copy Table*

| Base Format | Depth Aspect Format | Stencil Aspect Format |
|---|---|---|
| VK_FORMAT_D16_UNORM | VK_FORMAT_D16_UNORM | - |
| VK_FORMAT_X8_D24_UNORM_PACK32 | VK_FORMAT_X8_D24_UNORM_PACK32 | - |
| VK_FORMAT_D32_SFLOAT | VK_FORMAT_D32_SFLOAT | - |
| VK_FORMAT_S8_UINT | - | VK_FORMAT_S8_UINT |
| VK_FORMAT_D16_UNORM_S8_UINT | VK_FORMAT_D16_UNORM | VK_FORMAT_S8_UINT |
| VK_FORMAT_D24_UNORM_S8_UINT | VK_FORMAT_X8_D24_UNORM_PACK32 | VK_FORMAT_S8_UINT |
| VK_FORMAT_D32_SFLOAT_S8_UINT | VK_FORMAT_D32_SFLOAT | VK_FORMAT_S8_UINT |

When copying between a buffer and any plane of a multi-planar image, addressing calculations are performed using the compatible format for that plane, rather than the format of the multi-planar image.

Each texel block is copied from one resource to the other according to the above addressing equations.

To copy data from a buffer object to an image object, call:

```
// Provided by VK_VERSION_1_0
void vkCmdCopyBufferToImage(
    VkCommandBuffer                             commandBuffer,
    VkBuffer                                    srcBuffer,
    VkImage                                     dstImage,
    VkImageLayout                               dstImageLayout,
    uint32_t                                    regionCount,
    const VkBufferImageCopy*                    pRegions);
```

- commandBuffer is the command buffer into which the command will be recorded.
- srcBuffer is the source buffer.
- dstImage is the destination image.
- dstImageLayout is the layout of the destination image subresources for the copy.

- regionCount is the number of regions to copy.
- pRegions is a pointer to an array of VkBufferImageCopy structures specifying the regions to copy.

Each source region specified by pRegions is copied from the source buffer to the destination region of the destination image according to the addressing calculations for each resource. If any of the specified regions in srcBuffer overlaps in memory with any of the specified regions in dstImage, values read from those overlapping regions are undefined. If any region accesses a depth aspect in dstImage and the VK_EXT_depth_range_unrestricted extension is not enabled, values copied from srcBuffer outside of the range [0,1] will be written as undefined values to the destination image.

Copy regions for the image **must** be aligned to a multiple of the texel block extent in each dimension, except at the edges of the image, where region extents **must** match the edge of the image.

## Valid Usage

- VUID-vkCmdCopyBufferToImage-dstImage-07966
  If dstImage is non-sparse then the image or the specified *disjoint* plane **must** be bound completely and contiguously to a single VkDeviceMemory object
- VUID-vkCmdCopyBufferToImage-imageSubresource-07967
  The imageSubresource.mipLevel member of each element of pRegions **must** be less than the mipLevels specified in VkImageCreateInfo when dstImage was created
- VUID-vkCmdCopyBufferToImage-imageSubresource-07968
  If imageSubresource.layerCount is not VK_REMAINING_ARRAY_LAYERS, imageSubresource.baseArrayLayer + imageSubresource.layerCount of each element of pRegions **must** be less than or equal to the arrayLayers specified in VkImageCreateInfo when dstImage was created
- VUID-vkCmdCopyBufferToImage-dstImage-07969
  dstImage **must** not have been created with flags containing VK_IMAGE_CREATE_SUBSAMPLED_BIT_EXT
- VUID-vkCmdCopyBufferToImage-imageSubresource-07970
  The image region specified by each element of pRegions **must** be contained within the specified imageSubresource of dstImage
- VUID-vkCmdCopyBufferToImage-imageSubresource-07971
  For each element of pRegions, imageOffset.x and (imageExtent.width + imageOffset.x) **must** both be greater than or equal to 0 and less than or equal to the width of the specified imageSubresource of dstImage
- VUID-vkCmdCopyBufferToImage-imageSubresource-07972
  For each element of pRegions, imageOffset.y and (imageExtent.height + imageOffset.y) **must** both be greater than or equal to 0 and less than or equal to the height of the specified imageSubresource of dstImage
- VUID-vkCmdCopyBufferToImage-dstImage-07973
  dstImage **must** have a sample count equal to VK_SAMPLE_COUNT_1_BIT

- VUID-vkCmdCopyBufferToImage-commandBuffer-01828
  If `commandBuffer` is an unprotected command buffer and `protectedNoFault` is not supported, `srcBuffer` **must** not be a protected buffer
- VUID-vkCmdCopyBufferToImage-commandBuffer-01829
  If `commandBuffer` is an unprotected command buffer and `protectedNoFault` is not supported, `dstImage` **must** not be a protected image
- VUID-vkCmdCopyBufferToImage-commandBuffer-01830
  If `commandBuffer` is a protected command buffer and `protectedNoFault` is not supported, `dstImage` **must** not be an unprotected image
- VUID-vkCmdCopyBufferToImage-commandBuffer-07737
  If the queue family used to create the VkCommandPool which `commandBuffer` was allocated from does not support `VK_QUEUE_GRAPHICS_BIT` or `VK_QUEUE_COMPUTE_BIT`, the `bufferOffset` member of any element of `pRegions` **must** be a multiple of `4`
- VUID-vkCmdCopyBufferToImage-imageOffset-07738
  The `imageOffset` and `imageExtent` members of each element of `pRegions` **must** respect the image transfer granularity requirements of `commandBuffer`'s command pool's queue family, as described in VkQueueFamilyProperties
- VUID-vkCmdCopyBufferToImage-commandBuffer-07739
  If the queue family used to create the VkCommandPool which `commandBuffer` was allocated from does not support `VK_QUEUE_GRAPHICS_BIT`, for each element of `pRegions`, the `aspectMask` member of `imageSubresource` **must** not be `VK_IMAGE_ASPECT_DEPTH_BIT` or `VK_IMAGE_ASPECT_STENCIL_BIT`
- VUID-vkCmdCopyBufferToImage-pRegions-00171
  `srcBuffer` **must** be large enough to contain all buffer locations that are accessed according to Buffer and Image Addressing, for each element of `pRegions`
- VUID-vkCmdCopyBufferToImage-pRegions-00173
  The union of all source regions, and the union of all destination regions, specified by the elements of `pRegions`, **must** not overlap in memory
- VUID-vkCmdCopyBufferToImage-srcBuffer-00174
  `srcBuffer` **must** have been created with `VK_BUFFER_USAGE_TRANSFER_SRC_BIT` usage flag
- VUID-vkCmdCopyBufferToImage-dstImage-01997
  The format features of `dstImage` **must** contain `VK_FORMAT_FEATURE_TRANSFER_DST_BIT`
- VUID-vkCmdCopyBufferToImage-srcBuffer-00176
  If `srcBuffer` is non-sparse then it **must** be bound completely and contiguously to a single `VkDeviceMemory` object
- VUID-vkCmdCopyBufferToImage-dstImage-00177
  `dstImage` **must** have been created with `VK_IMAGE_USAGE_TRANSFER_DST_BIT` usage flag
- VUID-vkCmdCopyBufferToImage-dstImageLayout-00180
  `dstImageLayout` **must** specify the layout of the image subresources of `dstImage` specified in `pRegions` at the time this command is executed on a `VkDevice`
- VUID-vkCmdCopyBufferToImage-dstImageLayout-01396
  `dstImageLayout` **must** be `VK_IMAGE_LAYOUT_SHARED_PRESENT_KHR`,

VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL, or VK_IMAGE_LAYOUT_GENERAL

- VUID-vkCmdCopyBufferToImage-pRegions-07931
  If VK_EXT_depth_range_unrestricted is not enabled, for each element of pRegions whose imageSubresource contains a depth aspect, the data in srcBuffer **must** be in the range [0,1]

- VUID-vkCmdCopyBufferToImage-dstImage-07979
  If dstImage is of type VK_IMAGE_TYPE_1D, then for each element of pRegions, imageOffset.y **must** be 0 and imageExtent.height **must** be 1

- VUID-vkCmdCopyBufferToImage-imageOffset-09104
  For each element of pRegions, imageOffset.z and (imageExtent.depth + imageOffset.z) **must** both be greater than or equal to 0 and less than or equal to the depth of the specified imageSubresource of dstImage

- VUID-vkCmdCopyBufferToImage-dstImage-07980
  If dstImage is of type VK_IMAGE_TYPE_1D or VK_IMAGE_TYPE_2D, then for each element of pRegions, imageOffset.z **must** be 0 and imageExtent.depth **must** be 1

- VUID-vkCmdCopyBufferToImage-dstImage-07274
  For each element of pRegions, if VkCopyCommandTransformInfoQCOM::transform is equal to VK_SURFACE_TRANSFORM_IDENTITY_BIT_KHR or VK_SURFACE_TRANSFORM_ROTATE_270_BIT_KHR, imageOffset.x **must** be a multiple of the texel block extent width of the VkFormat of dstImage

- VUID-vkCmdCopyBufferToImage-imageOffset-10051
  For each element of pRegions, if VkCopyCommandTransformInfoQCOM::transform is equal to VK_SURFACE_TRANSFORM_ROTATE_180_BIT_KHR or VK_SURFACE_TRANSFORM_ROTATE_90_BIT_KHR, and imageOffset.x does not equal the width of the subresource specified by imageSubresource, imageOffset.x **must** be a multiple of the texel block extent width of the VkFormat of dstImage

- VUID-vkCmdCopyBufferToImage-dstImage-07275
  For each element of pRegions, if VkCopyCommandTransformInfoQCOM::transform is equal to VK_SURFACE_TRANSFORM_IDENTITY_BIT_KHR or VK_SURFACE_TRANSFORM_ROTATE_90_BIT_KHR, imageOffset.y **must** be a multiple of the texel block extent height of the VkFormat of dstImage

- VUID-vkCmdCopyBufferToImage-imageOffset-10052
  For each element of pRegions, if VkCopyCommandTransformInfoQCOM::transform is equal to VK_SURFACE_TRANSFORM_ROTATE_270_BIT_KHR or VK_SURFACE_TRANSFORM_ROTATE_180_BIT_KHR, and imageOffset.y does not equal the height of the subresource specified by imageSubresource, imageOffset.y **must** be a multiple of the texel block extent height of the VkFormat of dstImage

- VUID-vkCmdCopyBufferToImage-dstImage-07276
  For each element of pRegions, imageOffset.z **must** be a multiple of the texel block extent depth of the VkFormat of dstImage

- VUID-vkCmdCopyBufferToImage-dstImage-00207
  For each element of pRegions, if VkCopyCommandTransformInfoQCOM::transform is equal to VK_SURFACE_TRANSFORM_IDENTITY_BIT_KHR, the sum of imageOffset.x and extent.width does not equal the width of the subresource specified by imageSubresource, extent.width **must**

be a multiple of the texel block extent width of the VkFormat of dstImage

- VUID-vkCmdCopyBufferToImage-imageOffset-10053
  For each element of pRegions, if VkCopyCommandTransformInfoQCOM::transform is equal to VK_SURFACE_TRANSFORM_ROTATE_90_BIT_KHR, the difference of imageOffset.x and extent.height **must** be a multiple of the texel block extent width of the VkFormat of dstImage
- VUID-vkCmdCopyBufferToImage-imageOffset-10054
  For each element of pRegions, if VkCopyCommandTransformInfoQCOM::transform is equal to VK_SURFACE_TRANSFORM_ROTATE_180_BIT_KHR, the difference of imageOffset.x and extent.width **must** be a multiple of the texel block extent width of the VkFormat of dstImage
- VUID-vkCmdCopyBufferToImage-imageOffset-10055
  For each element of pRegions, if VkCopyCommandTransformInfoQCOM::transform is equal to VK_SURFACE_TRANSFORM_ROTATE_270_BIT_KHR, the sum of imageOffset.x and extent.height does not equal the width of the subresource specified by imageSubresource, extent.height **must** be a multiple of the texel block extent width of the VkFormat of dstImage
- VUID-vkCmdCopyBufferToImage-dstImage-00208
  For each element of pRegions, if VkCopyCommandTransformInfoQCOM::transform is equal to VK_SURFACE_TRANSFORM_IDENTITY_BIT_KHR, and the sum of imageOffset.y and extent.height does not equal the height of the subresource specified by imageSubresource, extent.height **must** be a multiple of the texel block extent height of the VkFormat of dstImage
- VUID-vkCmdCopyBufferToImage-imageOffset-10056
  For each element of pRegions, if VkCopyCommandTransformInfoQCOM::transform is equal to VK_SURFACE_TRANSFORM_ROTATE_90_BIT_KHR, the sum of imageOffset.y and extent.width does not equal the height of the subresource specified by imageSubresource, extent.width **must** be a multiple of the texel block extent height of the VkFormat of dstImage
- VUID-vkCmdCopyBufferToImage-imageOffset-10057
  For each element of pRegions, if VkCopyCommandTransformInfoQCOM::transform is equal to VK_SURFACE_TRANSFORM_ROTATE_180_BIT_KHR, the difference of imageOffset.y and extent.height **must** be a multiple of the texel block extent height of the VkFormat of dstImage
- VUID-vkCmdCopyBufferToImage-imageOffset-10058
  For each element of pRegions, if VkCopyCommandTransformInfoQCOM::transform is equal to VK_SURFACE_TRANSFORM_ROTATE_270_BIT_KHR, the difference of imageOffset.y and extent.width **must** be a multiple of the texel block extent height of the VkFormat of dstImage
- VUID-vkCmdCopyBufferToImage-dstImage-00209
  For each element of pRegions, if the sum of imageOffset.z and extent.depth does not equal the depth of the subresource specified by srcSubresource, extent.depth **must** be a multiple of the texel block extent depth of the VkFormat of dstImage
- VUID-vkCmdCopyBufferToImage-imageSubresource-09105
  For each element of pRegions, imageSubresource.aspectMask **must** specify aspects present in dstImage
- VUID-vkCmdCopyBufferToImage-dstImage-07981

If dstImage has a multi-planar image format, then for each element of pRegions, imageSubresource.aspectMask **must** be a single valid multi-planar aspect mask bit

- VUID-vkCmdCopyBufferToImage-dstImage-07983
  If dstImage is of type VK_IMAGE_TYPE_3D, for each element of pRegions, imageSubresource.baseArrayLayer **must** be 0 and imageSubresource.layerCount **must** be 1
- VUID-vkCmdCopyBufferToImage-bufferRowLength-09106
  For each element of pRegions, bufferRowLength **must** be a multiple of the texel block extent width of the VkFormat of dstImage
- VUID-vkCmdCopyBufferToImage-bufferImageHeight-09107
  For each element of pRegions, bufferImageHeight **must** be a multiple of the texel block extent height of the VkFormat of dstImage
- VUID-vkCmdCopyBufferToImage-bufferRowLength-09108
  For each element of pRegions, bufferRowLength divided by the texel block extent width and then multiplied by the texel block size of dstImage **must** be less than or equal to $2^{31}$-1
- VUID-vkCmdCopyBufferToImage-dstImage-07975
  If dstImage does not have either a depth/stencil format or a multi-planar format, then for each element of pRegions, bufferOffset **must** be a multiple of the texel block size
- VUID-vkCmdCopyBufferToImage-dstImage-07976
  If dstImage has a multi-planar format, then for each element of pRegions, bufferOffset **must** be a multiple of the element size of the compatible format for the format and the aspectMask of the imageSubresource as defined in Compatible Formats of Planes of Multi-Planar Formats
- VUID-vkCmdCopyBufferToImage-dstImage-07978
  If dstImage has a depth/stencil format, the bufferOffset member of any element of pRegions **must** be a multiple of 4

## Valid Usage (Implicit)

- VUID-vkCmdCopyBufferToImage-commandBuffer-parameter
  commandBuffer **must** be a valid VkCommandBuffer handle
- VUID-vkCmdCopyBufferToImage-srcBuffer-parameter
  srcBuffer **must** be a valid VkBuffer handle
- VUID-vkCmdCopyBufferToImage-dstImage-parameter
  dstImage **must** be a valid VkImage handle
- VUID-vkCmdCopyBufferToImage-dstImageLayout-parameter
  dstImageLayout **must** be a valid VkImageLayout value
- VUID-vkCmdCopyBufferToImage-pRegions-parameter
  pRegions **must** be a valid pointer to an array of regionCount valid VkBufferImageCopy structures
- VUID-vkCmdCopyBufferToImage-commandBuffer-recording
  commandBuffer **must** be in the recording state

- VUID-vkCmdCopyBufferToImage-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support transfer, graphics, or compute operations
- VUID-vkCmdCopyBufferToImage-renderpass
  This command **must** only be called outside of a render pass instance
- VUID-vkCmdCopyBufferToImage-videocoding
  This command **must** only be called outside of a video coding scope
- VUID-vkCmdCopyBufferToImage-regionCount-arraylength
  `regionCount` **must** be greater than `0`
- VUID-vkCmdCopyBufferToImage-commonparent
  Each of `commandBuffer`, `dstImage`, and `srcBuffer` **must** have been created, allocated, or retrieved from the same VkDevice

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Outside | Outside | Transfer<br>Graphics<br>Compute | Action |

To copy data from an image object to a buffer object, call:

```
// Provided by VK_VERSION_1_0
void vkCmdCopyImageToBuffer(
    VkCommandBuffer                             commandBuffer,
    VkImage                                     srcImage,
    VkImageLayout                               srcImageLayout,
    VkBuffer                                    dstBuffer,
    uint32_t                                    regionCount,
    const VkBufferImageCopy*                    pRegions);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `srcImage` is the source image.
- `srcImageLayout` is the layout of the source image subresources for the copy.

- `dstBuffer` is the destination buffer.
- `regionCount` is the number of regions to copy.
- `pRegions` is a pointer to an array of VkBufferImageCopy structures specifying the regions to copy.

Each source region specified by `pRegions` is copied from the source image to the destination region of the destination buffer according to the addressing calculations for each resource. If any of the specified regions in `srcImage` overlaps in memory with any of the specified regions in `dstBuffer`, values read from those overlapping regions are undefined.

Copy regions for the image **must** be aligned to a multiple of the texel block extent in each dimension, except at the edges of the image, where region extents **must** match the edge of the image.

## Valid Usage

- VUID-vkCmdCopyImageToBuffer-srcImage-07966
  If `srcImage` is non-sparse then the image or the specified *disjoint* plane **must** be bound completely and contiguously to a single `VkDeviceMemory` object
- VUID-vkCmdCopyImageToBuffer-imageSubresource-07967
  The `imageSubresource.mipLevel` member of each element of `pRegions` **must** be less than the `mipLevels` specified in VkImageCreateInfo when `srcImage` was created
- VUID-vkCmdCopyImageToBuffer-imageSubresource-07968
  If `imageSubresource.layerCount` is not `VK_REMAINING_ARRAY_LAYERS`, `imageSubresource.baseArrayLayer` + `imageSubresource.layerCount` of each element of `pRegions` **must** be less than or equal to the `arrayLayers` specified in VkImageCreateInfo when `srcImage` was created
- VUID-vkCmdCopyImageToBuffer-srcImage-07969
  `srcImage` **must** not have been created with `flags` containing `VK_IMAGE_CREATE_SUBSAMPLED_BIT_EXT`
- VUID-vkCmdCopyImageToBuffer-imageSubresource-07970
  The image region specified by each element of `pRegions` **must** be contained within the specified `imageSubresource` of `srcImage`
- VUID-vkCmdCopyImageToBuffer-imageSubresource-07971
  For each element of `pRegions`, `imageOffset.x` and (`imageExtent.width` + `imageOffset.x`) **must** both be greater than or equal to `0` and less than or equal to the width of the specified `imageSubresource` of `srcImage`
- VUID-vkCmdCopyImageToBuffer-imageSubresource-07972
  For each element of `pRegions`, `imageOffset.y` and (`imageExtent.height` + `imageOffset.y`) **must** both be greater than or equal to `0` and less than or equal to the height of the specified `imageSubresource` of `srcImage`
- VUID-vkCmdCopyImageToBuffer-srcImage-07973
  `srcImage` **must** have a sample count equal to `VK_SAMPLE_COUNT_1_BIT`
- VUID-vkCmdCopyImageToBuffer-commandBuffer-01831

If commandBuffer is an unprotected command buffer and protectedNoFault is not supported, srcImage **must** not be a protected image

- VUID-vkCmdCopyImageToBuffer-commandBuffer-01832
  If commandBuffer is an unprotected command buffer and protectedNoFault is not supported, dstBuffer **must** not be a protected buffer
- VUID-vkCmdCopyImageToBuffer-commandBuffer-01833
  If commandBuffer is a protected command buffer and protectedNoFault is not supported, dstBuffer **must** not be an unprotected buffer
- VUID-vkCmdCopyImageToBuffer-commandBuffer-07746
  If the queue family used to create the VkCommandPool which commandBuffer was allocated from does not support VK_QUEUE_GRAPHICS_BIT or VK_QUEUE_COMPUTE_BIT, the bufferOffset member of any element of pRegions **must** be a multiple of 4
- VUID-vkCmdCopyImageToBuffer-imageOffset-07747
  The imageOffset and imageExtent members of each element of pRegions **must** respect the image transfer granularity requirements of commandBuffer's command pool's queue family, as described in VkQueueFamilyProperties
- VUID-vkCmdCopyImageToBuffer-pRegions-00183
  dstBuffer **must** be large enough to contain all buffer locations that are accessed according to Buffer and Image Addressing, for each element of pRegions
- VUID-vkCmdCopyImageToBuffer-pRegions-00184
  The union of all source regions, and the union of all destination regions, specified by the elements of pRegions, **must** not overlap in memory
- VUID-vkCmdCopyImageToBuffer-srcImage-00186
  srcImage **must** have been created with VK_IMAGE_USAGE_TRANSFER_SRC_BIT usage flag
- VUID-vkCmdCopyImageToBuffer-srcImage-01998
  The format features of srcImage **must** contain VK_FORMAT_FEATURE_TRANSFER_SRC_BIT
- VUID-vkCmdCopyImageToBuffer-dstBuffer-00191
  dstBuffer **must** have been created with VK_BUFFER_USAGE_TRANSFER_DST_BIT usage flag
- VUID-vkCmdCopyImageToBuffer-dstBuffer-00192
  If dstBuffer is non-sparse then it **must** be bound completely and contiguously to a single VkDeviceMemory object
- VUID-vkCmdCopyImageToBuffer-srcImageLayout-00189
  srcImageLayout **must** specify the layout of the image subresources of srcImage specified in pRegions at the time this command is executed on a VkDevice
- VUID-vkCmdCopyImageToBuffer-srcImageLayout-01397
  srcImageLayout **must** be VK_IMAGE_LAYOUT_SHARED_PRESENT_KHR, VK_IMAGE_LAYOUT_TRANSFER_SRC_OPTIMAL, or VK_IMAGE_LAYOUT_GENERAL
- VUID-vkCmdCopyImageToBuffer-srcImage-07979
  If srcImage is of type VK_IMAGE_TYPE_1D, then for each element of pRegions, imageOffset.y **must** be 0 and imageExtent.height **must** be 1
- VUID-vkCmdCopyImageToBuffer-imageOffset-09104

For each element of pRegions, imageOffset.z and (imageExtent.depth + imageOffset.z) **must** both be greater than or equal to 0 and less than or equal to the depth of the specified imageSubresource of srcImage

- VUID-vkCmdCopyImageToBuffer-srcImage-07980
  If srcImage is of type VK_IMAGE_TYPE_1D or VK_IMAGE_TYPE_2D, then for each element of pRegions, imageOffset.z **must** be 0 and imageExtent.depth **must** be 1
- VUID-vkCmdCopyImageToBuffer-srcImage-07274
  For each element of pRegions, if VkCopyCommandTransformInfoQCOM::transform is equal to VK_SURFACE_TRANSFORM_IDENTITY_BIT_KHR or VK_SURFACE_TRANSFORM_ROTATE_270_BIT_KHR, imageOffset.x **must** be a multiple of the texel block extent width of the VkFormat of srcImage
- VUID-vkCmdCopyImageToBuffer-imageOffset-10051
  For each element of pRegions, if VkCopyCommandTransformInfoQCOM::transform is equal to VK_SURFACE_TRANSFORM_ROTATE_180_BIT_KHR or VK_SURFACE_TRANSFORM_ROTATE_90_BIT_KHR, and imageOffset.x does not equal the width of the subresource specified by imageSubresource, imageOffset.x **must** be a multiple of the texel block extent width of the VkFormat of srcImage
- VUID-vkCmdCopyImageToBuffer-srcImage-07275
  For each element of pRegions, if VkCopyCommandTransformInfoQCOM::transform is equal to VK_SURFACE_TRANSFORM_IDENTITY_BIT_KHR or VK_SURFACE_TRANSFORM_ROTATE_90_BIT_KHR, imageOffset.y **must** be a multiple of the texel block extent height of the VkFormat of srcImage
- VUID-vkCmdCopyImageToBuffer-imageOffset-10052
  For each element of pRegions, if VkCopyCommandTransformInfoQCOM::transform is equal to VK_SURFACE_TRANSFORM_ROTATE_270_BIT_KHR or VK_SURFACE_TRANSFORM_ROTATE_180_BIT_KHR, and imageOffset.y does not equal the height of the subresource specified by imageSubresource, imageOffset.y **must** be a multiple of the texel block extent height of the VkFormat of srcImage
- VUID-vkCmdCopyImageToBuffer-srcImage-07276
  For each element of pRegions, imageOffset.z **must** be a multiple of the texel block extent depth of the VkFormat of srcImage
- VUID-vkCmdCopyImageToBuffer-srcImage-00207
  For each element of pRegions, if VkCopyCommandTransformInfoQCOM::transform is equal to VK_SURFACE_TRANSFORM_IDENTITY_BIT_KHR, the sum of imageOffset.x and extent.width does not equal the width of the subresource specified by imageSubresource, extent.width **must** be a multiple of the texel block extent width of the VkFormat of srcImage
- VUID-vkCmdCopyImageToBuffer-imageOffset-10053
  For each element of pRegions, if VkCopyCommandTransformInfoQCOM::transform is equal to VK_SURFACE_TRANSFORM_ROTATE_90_BIT_KHR, the difference of imageOffset.x and extent.height **must** be a multiple of the texel block extent width of the VkFormat of srcImage
- VUID-vkCmdCopyImageToBuffer-imageOffset-10054
  For each element of pRegions, if VkCopyCommandTransformInfoQCOM::transform is equal to VK_SURFACE_TRANSFORM_ROTATE_180_BIT_KHR, the difference of imageOffset.x and

extent.width **must** be a multiple of the texel block extent width of the VkFormat of srcImage

- VUID-vkCmdCopyImageToBuffer-imageOffset-10055
  For each element of pRegions, if VkCopyCommandTransformInfoQCOM::transform is equal to VK_SURFACE_TRANSFORM_ROTATE_270_BIT_KHR, the sum of imageOffset.x and extent.height does not equal the width of the subresource specified by imageSubresource, extent.height **must** be a multiple of the texel block extent width of the VkFormat of srcImage
- VUID-vkCmdCopyImageToBuffer-srcImage-00208
  For each element of pRegions, if VkCopyCommandTransformInfoQCOM::transform is equal to VK_SURFACE_TRANSFORM_IDENTITY_BIT_KHR, and the sum of imageOffset.y and extent.height does not equal the height of the subresource specified by imageSubresource, extent.height **must** be a multiple of the texel block extent height of the VkFormat of srcImage
- VUID-vkCmdCopyImageToBuffer-imageOffset-10056
  For each element of pRegions, if VkCopyCommandTransformInfoQCOM::transform is equal to VK_SURFACE_TRANSFORM_ROTATE_90_BIT_KHR, the sum of imageOffset.y and extent.width does not equal the height of the subresource specified by imageSubresource, extent.width **must** be a multiple of the texel block extent height of the VkFormat of srcImage
- VUID-vkCmdCopyImageToBuffer-imageOffset-10057
  For each element of pRegions, if VkCopyCommandTransformInfoQCOM::transform is equal to VK_SURFACE_TRANSFORM_ROTATE_180_BIT_KHR, the difference of imageOffset.y and extent.height **must** be a multiple of the texel block extent height of the VkFormat of srcImage
- VUID-vkCmdCopyImageToBuffer-imageOffset-10058
  For each element of pRegions, if VkCopyCommandTransformInfoQCOM::transform is equal to VK_SURFACE_TRANSFORM_ROTATE_270_BIT_KHR, the difference of imageOffset.y and extent.width **must** be a multiple of the texel block extent height of the VkFormat of srcImage
- VUID-vkCmdCopyImageToBuffer-srcImage-00209
  For each element of pRegions, if the sum of imageOffset.z and extent.depth does not equal the depth of the subresource specified by srcSubresource, extent.depth **must** be a multiple of the texel block extent depth of the VkFormat of srcImage
- VUID-vkCmdCopyImageToBuffer-imageSubresource-09105
  For each element of pRegions, imageSubresource.aspectMask **must** specify aspects present in srcImage
- VUID-vkCmdCopyImageToBuffer-srcImage-07981
  If srcImage has a multi-planar image format, then for each element of pRegions, imageSubresource.aspectMask **must** be a single valid multi-planar aspect mask bit
- VUID-vkCmdCopyImageToBuffer-srcImage-07983
  If srcImage is of type VK_IMAGE_TYPE_3D, for each element of pRegions, imageSubresource.baseArrayLayer **must** be 0 and imageSubresource.layerCount **must** be 1
- VUID-vkCmdCopyImageToBuffer-bufferRowLength-09106
  For each element of pRegions, bufferRowLength **must** be a multiple of the texel block extent width of the VkFormat of srcImage

- VUID-vkCmdCopyImageToBuffer-bufferImageHeight-09107
  For each element of `pRegions`, `bufferImageHeight` **must** be a multiple of the texel block extent height of the VkFormat of `srcImage`
- VUID-vkCmdCopyImageToBuffer-bufferRowLength-09108
  For each element of `pRegions`, `bufferRowLength` divided by the texel block extent width and then multiplied by the texel block size of `srcImage` **must** be less than or equal to $2^{31}$-1
- VUID-vkCmdCopyImageToBuffer-srcImage-07975
  If `srcImage` does not have either a depth/stencil format or a multi-planar format, then for each element of `pRegions`, `bufferOffset` **must** be a multiple of the texel block size
- VUID-vkCmdCopyImageToBuffer-srcImage-07976
  If `srcImage` has a multi-planar format, then for each element of `pRegions`, `bufferOffset` **must** be a multiple of the element size of the compatible format for the format and the `aspectMask` of the `imageSubresource` as defined in Compatible Formats of Planes of Multi-Planar Formats
- VUID-vkCmdCopyImageToBuffer-srcImage-07978
  If `srcImage` has a depth/stencil format, the `bufferOffset` member of any element of `pRegions` **must** be a multiple of `4`

## Valid Usage (Implicit)

- VUID-vkCmdCopyImageToBuffer-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdCopyImageToBuffer-srcImage-parameter
  `srcImage` **must** be a valid VkImage handle
- VUID-vkCmdCopyImageToBuffer-srcImageLayout-parameter
  `srcImageLayout` **must** be a valid VkImageLayout value
- VUID-vkCmdCopyImageToBuffer-dstBuffer-parameter
  `dstBuffer` **must** be a valid VkBuffer handle
- VUID-vkCmdCopyImageToBuffer-pRegions-parameter
  `pRegions` **must** be a valid pointer to an array of `regionCount` valid VkBufferImageCopy structures
- VUID-vkCmdCopyImageToBuffer-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdCopyImageToBuffer-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support transfer, graphics, or compute operations
- VUID-vkCmdCopyImageToBuffer-renderpass
  This command **must** only be called outside of a render pass instance
- VUID-vkCmdCopyImageToBuffer-videocoding
  This command **must** only be called outside of a video coding scope
- VUID-vkCmdCopyImageToBuffer-regionCount-arraylength

regionCount **must** be greater than 0

- VUID-vkCmdCopyImageToBuffer-commonparent
  Each of commandBuffer, dstBuffer, and srcImage **must** have been created, allocated, or retrieved from the same VkDevice

## Host Synchronization

- Host access to commandBuffer **must** be externally synchronized
- Host access to the VkCommandPool that commandBuffer was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Outside | Outside | Transfer<br>Graphics<br>Compute | Action |

For both vkCmdCopyBufferToImage and vkCmdCopyImageToBuffer, each element of pRegions is a structure defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkBufferImageCopy {
    VkDeviceSize                bufferOffset;
    uint32_t                    bufferRowLength;
    uint32_t                    bufferImageHeight;
    VkImageSubresourceLayers    imageSubresource;
    VkOffset3D                  imageOffset;
    VkExtent3D                  imageExtent;
} VkBufferImageCopy;
```

- bufferOffset is the offset in bytes from the start of the buffer object where the image data is copied from or to.
- bufferRowLength and bufferImageHeight specify in texels a subregion of a larger two- or three-dimensional image in buffer memory, and control the addressing calculations. If either of these values is zero, that aspect of the buffer memory is considered to be tightly packed according to the imageExtent.
- imageSubresource is a VkImageSubresourceLayers used to specify the specific image subresources of the image used for the source or destination image data.
- imageOffset selects the initial x, y, z offsets in texels of the sub-region of the source or destination image data.

- `imageExtent` is the size in texels of the image to copy in `width`, `height` and `depth`.

### Valid Usage

- VUID-VkBufferImageCopy-bufferRowLength-09101
  `bufferRowLength` **must** be `0`, or greater than or equal to the `width` member of `imageExtent`
- VUID-VkBufferImageCopy-bufferImageHeight-09102
  `bufferImageHeight` **must** be `0`, or greater than or equal to the `height` member of `imageExtent`
- VUID-VkBufferImageCopy-aspectMask-09103
  The `aspectMask` member of `imageSubresource` **must** only have a single bit set
- VUID-VkBufferImageCopy-imageExtent-06659
  `imageExtent.width` **must** not be 0
- VUID-VkBufferImageCopy-imageExtent-06660
  `imageExtent.height` **must** not be 0
- VUID-VkBufferImageCopy-imageExtent-06661
  `imageExtent.depth` **must** not be 0

### Valid Usage (Implicit)

- VUID-VkBufferImageCopy-imageSubresource-parameter
  `imageSubresource` **must** be a valid VkImageSubresourceLayers structure

More extensible versions of the commands to copy between buffers and images are defined below.

To copy data from a buffer object to an image object, call:

```
// Provided by VK_KHR_copy_commands2
void vkCmdCopyBufferToImage2KHR(
    VkCommandBuffer                             commandBuffer,
    const VkCopyBufferToImageInfo2*             pCopyBufferToImageInfo);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `pCopyBufferToImageInfo` is a pointer to a VkCopyBufferToImageInfo2 structure describing the copy parameters.

This command is functionally identical to vkCmdCopyBufferToImage, but includes extensible sub-structures that include `sType` and `pNext` parameters, allowing them to be more easily extended.

### Valid Usage

- VUID-vkCmdCopyBufferToImage2-commandBuffer-01828
  If `commandBuffer` is an unprotected command buffer and `protectedNoFault` is not supported, `srcBuffer` **must** not be a protected buffer

- VUID-vkCmdCopyBufferToImage2-commandBuffer-01829
  If `commandBuffer` is an unprotected command buffer and `protectedNoFault` is not supported, `dstImage` **must** not be a protected image
- VUID-vkCmdCopyBufferToImage2-commandBuffer-01830
  If `commandBuffer` is a protected command buffer and `protectedNoFault` is not supported, `dstImage` **must** not be an unprotected image
- VUID-vkCmdCopyBufferToImage2-commandBuffer-07737
  If the queue family used to create the VkCommandPool which `commandBuffer` was allocated from does not support `VK_QUEUE_GRAPHICS_BIT` or `VK_QUEUE_COMPUTE_BIT`, the `bufferOffset` member of any element of `pCopyBufferToImageInfo->pRegions` **must** be a multiple of `4`
- VUID-vkCmdCopyBufferToImage2-imageOffset-07738
  The `imageOffset` and `imageExtent` members of each element of `pCopyBufferToImageInfo->pRegions` **must** respect the image transfer granularity requirements of `commandBuffer`'s command pool's queue family, as described in VkQueueFamilyProperties
- VUID-vkCmdCopyBufferToImage2-commandBuffer-07739
  If the queue family used to create the VkCommandPool which `commandBuffer` was allocated from does not support `VK_QUEUE_GRAPHICS_BIT`, for each element of `pCopyBufferToImageInfo->pRegions`, the `aspectMask` member of `imageSubresource` **must** not be `VK_IMAGE_ASPECT_DEPTH_BIT` or `VK_IMAGE_ASPECT_STENCIL_BIT`

## Valid Usage (Implicit)

- VUID-vkCmdCopyBufferToImage2-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdCopyBufferToImage2-pCopyBufferToImageInfo-parameter
  `pCopyBufferToImageInfo` **must** be a valid pointer to a valid VkCopyBufferToImageInfo2 structure
- VUID-vkCmdCopyBufferToImage2-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdCopyBufferToImage2-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support transfer, graphics, or compute operations
- VUID-vkCmdCopyBufferToImage2-renderpass
  This command **must** only be called outside of a render pass instance
- VUID-vkCmdCopyBufferToImage2-videocoding
  This command **must** only be called outside of a video coding scope

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Outside | Outside | Transfer<br>Graphics<br>Compute | Action |

The `VkCopyBufferToImageInfo2` structure is defined as:

```
typedef struct VkCopyBufferToImageInfo2 {
    VkStructureType              sType;
    const void*                  pNext;
    VkBuffer                     srcBuffer;
    VkImage                      dstImage;
    VkImageLayout                dstImageLayout;
    uint32_t                     regionCount;
    const VkBufferImageCopy2*    pRegions;
} VkCopyBufferToImageInfo2;
```

or the equivalent

```
// Provided by VK_KHR_copy_commands2
typedef VkCopyBufferToImageInfo2 VkCopyBufferToImageInfo2KHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `srcBuffer` is the source buffer.
- `dstImage` is the destination image.
- `dstImageLayout` is the layout of the destination image subresources for the copy.
- `regionCount` is the number of regions to copy.
- `pRegions` is a pointer to an array of VkBufferImageCopy2 structures specifying the regions to copy.

## Valid Usage

- VUID-VkCopyBufferToImageInfo2-pRegions-04565
  The image region specified by each element of `pRegions` that does not contain VkCopyCommandTransformInfoQCOM in its `pNext` chain **must** be contained within the specified `imageSubresource` of `dstImage`
- VUID-VkCopyBufferToImageInfo2KHR-pRegions-04554
  If the image region specified by each element of `pRegions` contains

VkCopyCommandTransformInfoQCOM in its pNext chain, the rotated destination region as described in [copies-buffers-images-rotation-addressing] **must** be contained within dstImage

- VUID-VkCopyBufferToImageInfo2KHR-pRegions-04555
  If any element of pRegions contains VkCopyCommandTransformInfoQCOM in its pNext chain, then dstImage **must** have a 1x1x1 texel block extent
- VUID-VkCopyBufferToImageInfo2KHR-pRegions-06203
  If any element of pRegions contains VkCopyCommandTransformInfoQCOM in its pNext chain, then dstImage **must** be of type VK_IMAGE_TYPE_2D
- VUID-VkCopyBufferToImageInfo2KHR-pRegions-06204
  If any element of pRegions contains VkCopyCommandTransformInfoQCOM in its pNext chain, then dstImage **must** not have a multi-planar format
- VUID-VkCopyBufferToImageInfo2-pRegions-00171
  srcBuffer **must** be large enough to contain all buffer locations that are accessed according to Buffer and Image Addressing, for each element of pRegions
- VUID-VkCopyBufferToImageInfo2-pRegions-00173
  The union of all source regions, and the union of all destination regions, specified by the elements of pRegions, **must** not overlap in memory
- VUID-VkCopyBufferToImageInfo2-srcBuffer-00174
  srcBuffer **must** have been created with VK_BUFFER_USAGE_TRANSFER_SRC_BIT usage flag
- VUID-VkCopyBufferToImageInfo2-dstImage-01997
  The format features of dstImage **must** contain VK_FORMAT_FEATURE_TRANSFER_DST_BIT
- VUID-VkCopyBufferToImageInfo2-srcBuffer-00176
  If srcBuffer is non-sparse then it **must** be bound completely and contiguously to a single VkDeviceMemory object
- VUID-VkCopyBufferToImageInfo2-dstImage-00177
  dstImage **must** have been created with VK_IMAGE_USAGE_TRANSFER_DST_BIT usage flag
- VUID-VkCopyBufferToImageInfo2-dstImageLayout-00180
  dstImageLayout **must** specify the layout of the image subresources of dstImage specified in pRegions at the time this command is executed on a VkDevice
- VUID-VkCopyBufferToImageInfo2-dstImageLayout-01396
  dstImageLayout **must** be VK_IMAGE_LAYOUT_SHARED_PRESENT_KHR, VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL, or VK_IMAGE_LAYOUT_GENERAL
- VUID-VkCopyBufferToImageInfo2-pRegions-07931
  If VK_EXT_depth_range_unrestricted is not enabled, for each element of pRegions whose imageSubresource contains a depth aspect, the data in srcBuffer **must** be in the range [0,1]
- VUID-VkCopyBufferToImageInfo2-dstImage-07966
  If dstImage is non-sparse then the image or the specified *disjoint* plane **must** be bound completely and contiguously to a single VkDeviceMemory object
- VUID-VkCopyBufferToImageInfo2-imageSubresource-07967
  The imageSubresource.mipLevel member of each element of pRegions **must** be less than the

mipLevels specified in VkImageCreateInfo when dstImage was created

- VUID-VkCopyBufferToImageInfo2-imageSubresource-07968
  If imageSubresource.layerCount is not VK_REMAINING_ARRAY_LAYERS, imageSubresource.baseArrayLayer + imageSubresource.layerCount of each element of pRegions **must** be less than or equal to the arrayLayers specified in VkImageCreateInfo when dstImage was created
- VUID-VkCopyBufferToImageInfo2-dstImage-07969
  dstImage **must** not have been created with flags containing VK_IMAGE_CREATE_SUBSAMPLED_BIT_EXT
- VUID-VkCopyBufferToImageInfo2-dstImage-07973
  dstImage **must** have a sample count equal to VK_SAMPLE_COUNT_1_BIT
- VUID-VkCopyBufferToImageInfo2-dstImage-07979
  If dstImage is of type VK_IMAGE_TYPE_1D, then for each element of pRegions, imageOffset.y **must** be 0 and imageExtent.height **must** be 1
- VUID-VkCopyBufferToImageInfo2-imageOffset-09104
  For each element of pRegions, imageOffset.z and (imageExtent.depth + imageOffset.z) **must** both be greater than or equal to 0 and less than or equal to the depth of the specified imageSubresource of dstImage
- VUID-VkCopyBufferToImageInfo2-dstImage-07980
  If dstImage is of type VK_IMAGE_TYPE_1D or VK_IMAGE_TYPE_2D, then for each element of pRegions, imageOffset.z **must** be 0 and imageExtent.depth **must** be 1
- VUID-VkCopyBufferToImageInfo2-dstImage-07274
  For each element of pRegions, if VkCopyCommandTransformInfoQCOM::transform is equal to VK_SURFACE_TRANSFORM_IDENTITY_BIT_KHR or VK_SURFACE_TRANSFORM_ROTATE_270_BIT_KHR, imageOffset.x **must** be a multiple of the texel block extent width of the VkFormat of dstImage
- VUID-VkCopyBufferToImageInfo2-imageOffset-10051
  For each element of pRegions, if VkCopyCommandTransformInfoQCOM::transform is equal to VK_SURFACE_TRANSFORM_ROTATE_180_BIT_KHR or VK_SURFACE_TRANSFORM_ROTATE_90_BIT_KHR, and imageOffset.x does not equal the width of the subresource specified by imageSubresource, imageOffset.x **must** be a multiple of the texel block extent width of the VkFormat of dstImage
- VUID-VkCopyBufferToImageInfo2-dstImage-07275
  For each element of pRegions, if VkCopyCommandTransformInfoQCOM::transform is equal to VK_SURFACE_TRANSFORM_IDENTITY_BIT_KHR or VK_SURFACE_TRANSFORM_ROTATE_90_BIT_KHR, imageOffset.y **must** be a multiple of the texel block extent height of the VkFormat of dstImage
- VUID-VkCopyBufferToImageInfo2-imageOffset-10052
  For each element of pRegions, if VkCopyCommandTransformInfoQCOM::transform is equal to VK_SURFACE_TRANSFORM_ROTATE_270_BIT_KHR or VK_SURFACE_TRANSFORM_ROTATE_180_BIT_KHR, and imageOffset.y does not equal the height of the subresource specified by imageSubresource, imageOffset.y **must** be a multiple of the texel block extent height of the VkFormat of dstImage

- VUID-VkCopyBufferToImageInfo2-dstImage-07276
  For each element of `pRegions`, `imageOffset.z` **must** be a multiple of the texel block extent depth of the VkFormat of `dstImage`
- VUID-VkCopyBufferToImageInfo2-dstImage-00207
  For each element of `pRegions`, if VkCopyCommandTransformInfoQCOM::`transform` is equal to `VK_SURFACE_TRANSFORM_IDENTITY_BIT_KHR`, the sum of `imageOffset.x` and `extent.width` does not equal the width of the subresource specified by `imageSubresource`, `extent.width` **must** be a multiple of the texel block extent width of the VkFormat of `dstImage`
- VUID-VkCopyBufferToImageInfo2-imageOffset-10053
  For each element of `pRegions`, if VkCopyCommandTransformInfoQCOM::`transform` is equal to `VK_SURFACE_TRANSFORM_ROTATE_90_BIT_KHR`, the difference of `imageOffset.x` and `extent.height` **must** be a multiple of the texel block extent width of the VkFormat of `dstImage`
- VUID-VkCopyBufferToImageInfo2-imageOffset-10054
  For each element of `pRegions`, if VkCopyCommandTransformInfoQCOM::`transform` is equal to `VK_SURFACE_TRANSFORM_ROTATE_180_BIT_KHR`, the difference of `imageOffset.x` and `extent.width` **must** be a multiple of the texel block extent width of the VkFormat of `dstImage`
- VUID-VkCopyBufferToImageInfo2-imageOffset-10055
  For each element of `pRegions`, if VkCopyCommandTransformInfoQCOM::`transform` is equal to `VK_SURFACE_TRANSFORM_ROTATE_270_BIT_KHR`, the sum of `imageOffset.x` and `extent.height` does not equal the width of the subresource specified by `imageSubresource`, `extent.height` **must** be a multiple of the texel block extent width of the VkFormat of `dstImage`
- VUID-VkCopyBufferToImageInfo2-dstImage-00208
  For each element of `pRegions`, if VkCopyCommandTransformInfoQCOM::`transform` is equal to `VK_SURFACE_TRANSFORM_IDENTITY_BIT_KHR`, and the sum of `imageOffset.y` and `extent.height` does not equal the height of the subresource specified by `imageSubresource`, `extent.height` **must** be a multiple of the texel block extent height of the VkFormat of `dstImage`
- VUID-VkCopyBufferToImageInfo2-imageOffset-10056
  For each element of `pRegions`, if VkCopyCommandTransformInfoQCOM::`transform` is equal to `VK_SURFACE_TRANSFORM_ROTATE_90_BIT_KHR`, the sum of `imageOffset.y` and `extent.width` does not equal the height of the subresource specified by `imageSubresource`, `extent.width` **must** be a multiple of the texel block extent height of the VkFormat of `dstImage`
- VUID-VkCopyBufferToImageInfo2-imageOffset-10057
  For each element of `pRegions`, if VkCopyCommandTransformInfoQCOM::`transform` is equal to `VK_SURFACE_TRANSFORM_ROTATE_180_BIT_KHR`, the difference of `imageOffset.y` and `extent.height` **must** be a multiple of the texel block extent height of the VkFormat of `dstImage`
- VUID-VkCopyBufferToImageInfo2-imageOffset-10058
  For each element of `pRegions`, if VkCopyCommandTransformInfoQCOM::`transform` is equal to `VK_SURFACE_TRANSFORM_ROTATE_270_BIT_KHR`, the difference of `imageOffset.y` and `extent.width` **must** be a multiple of the texel block extent height of the VkFormat of `dstImage`
- VUID-VkCopyBufferToImageInfo2-dstImage-00209

For each element of pRegions, if the sum of imageOffset.z and extent.depth does not equal the depth of the subresource specified by srcSubresource, extent.depth **must** be a multiple of the texel block extent depth of the VkFormat of dstImage

- VUID-VkCopyBufferToImageInfo2-imageSubresource-09105
  For each element of pRegions, imageSubresource.aspectMask **must** specify aspects present in dstImage
- VUID-VkCopyBufferToImageInfo2-dstImage-07981
  If dstImage has a multi-planar image format, then for each element of pRegions, imageSubresource.aspectMask **must** be a single valid multi-planar aspect mask bit
- VUID-VkCopyBufferToImageInfo2-dstImage-07983
  If dstImage is of type VK_IMAGE_TYPE_3D, for each element of pRegions, imageSubresource.baseArrayLayer **must** be 0 and imageSubresource.layerCount **must** be 1
- VUID-VkCopyBufferToImageInfo2-bufferRowLength-09106
  For each element of pRegions, bufferRowLength **must** be a multiple of the texel block extent width of the VkFormat of dstImage
- VUID-VkCopyBufferToImageInfo2-bufferImageHeight-09107
  For each element of pRegions, bufferImageHeight **must** be a multiple of the texel block extent height of the VkFormat of dstImage
- VUID-VkCopyBufferToImageInfo2-bufferRowLength-09108
  For each element of pRegions, bufferRowLength divided by the texel block extent width and then multiplied by the texel block size of dstImage **must** be less than or equal to $2^{31}$-1
- VUID-VkCopyBufferToImageInfo2-dstImage-07975
  If dstImage does not have either a depth/stencil format or a multi-planar format, then for each element of pRegions, bufferOffset **must** be a multiple of the texel block size
- VUID-VkCopyBufferToImageInfo2-dstImage-07976
  If dstImage has a multi-planar format, then for each element of pRegions, bufferOffset **must** be a multiple of the element size of the compatible format for the format and the aspectMask of the imageSubresource as defined in Compatible Formats of Planes of Multi-Planar Formats
- VUID-VkCopyBufferToImageInfo2-dstImage-07978
  If dstImage has a depth/stencil format, the bufferOffset member of any element of pRegions **must** be a multiple of 4
- VUID-VkCopyBufferToImageInfo2-pRegions-06223
  For each element of pRegions not containing VkCopyCommandTransformInfoQCOM in its pNext chain, imageOffset.x and (imageExtent.width + imageOffset.x) **must** both be greater than or equal to 0 and less than or equal to the width of the specified imageSubresource of dstImage
- VUID-VkCopyBufferToImageInfo2-pRegions-06224
  For each element of pRegions not containing VkCopyCommandTransformInfoQCOM in its pNext chain, imageOffset.y and (imageExtent.height + imageOffset.y) **must** both be greater than or equal to 0 and less than or equal to the height of the specified imageSubresource of dstImage

### Valid Usage (Implicit)

- VUID-VkCopyBufferToImageInfo2-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_COPY_BUFFER_TO_IMAGE_INFO_2`
- VUID-VkCopyBufferToImageInfo2-pNext-pNext
  `pNext` **must** be `NULL`
- VUID-VkCopyBufferToImageInfo2-srcBuffer-parameter
  `srcBuffer` **must** be a valid VkBuffer handle
- VUID-VkCopyBufferToImageInfo2-dstImage-parameter
  `dstImage` **must** be a valid VkImage handle
- VUID-VkCopyBufferToImageInfo2-dstImageLayout-parameter
  `dstImageLayout` **must** be a valid VkImageLayout value
- VUID-VkCopyBufferToImageInfo2-pRegions-parameter
  `pRegions` **must** be a valid pointer to an array of `regionCount` valid VkBufferImageCopy2 structures
- VUID-VkCopyBufferToImageInfo2-regionCount-arraylength
  `regionCount` **must** be greater than `0`
- VUID-VkCopyBufferToImageInfo2-commonparent
  Both of `dstImage`, and `srcBuffer` **must** have been created, allocated, or retrieved from the same VkDevice

To copy data from an image object to a buffer object, call:

```
// Provided by VK_KHR_copy_commands2
void vkCmdCopyImageToBuffer2KHR(
    VkCommandBuffer                             commandBuffer,
    const VkCopyImageToBufferInfo2*             pCopyImageToBufferInfo);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `pCopyImageToBufferInfo` is a pointer to a VkCopyImageToBufferInfo2 structure describing the copy parameters.

This command is functionally identical to vkCmdCopyImageToBuffer, but includes extensible sub-structures that include `sType` and `pNext` parameters, allowing them to be more easily extended.

### Valid Usage

- VUID-vkCmdCopyImageToBuffer2-commandBuffer-01831
  If `commandBuffer` is an unprotected command buffer and `protectedNoFault` is not supported, `srcImage` **must** not be a protected image
- VUID-vkCmdCopyImageToBuffer2-commandBuffer-01832
  If `commandBuffer` is an unprotected command buffer and `protectedNoFault` is not supported,

dstBuffer **must** not be a protected buffer

- VUID-vkCmdCopyImageToBuffer2-commandBuffer-01833
  If commandBuffer is a protected command buffer and protectedNoFault is not supported, dstBuffer **must** not be an unprotected buffer
- VUID-vkCmdCopyImageToBuffer2-commandBuffer-07746
  If the queue family used to create the VkCommandPool which commandBuffer was allocated from does not support VK_QUEUE_GRAPHICS_BIT or VK_QUEUE_COMPUTE_BIT, the bufferOffset member of any element of pCopyImageToBufferInfo->pRegions **must** be a multiple of 4
- VUID-vkCmdCopyImageToBuffer2-imageOffset-07747
  The imageOffset and imageExtent members of each element of pCopyImageToBufferInfo->pRegions **must** respect the image transfer granularity requirements of commandBuffer's command pool's queue family, as described in VkQueueFamilyProperties

## Valid Usage (Implicit)

- VUID-vkCmdCopyImageToBuffer2-commandBuffer-parameter
  commandBuffer **must** be a valid VkCommandBuffer handle
- VUID-vkCmdCopyImageToBuffer2-pCopyImageToBufferInfo-parameter
  pCopyImageToBufferInfo **must** be a valid pointer to a valid VkCopyImageToBufferInfo2 structure
- VUID-vkCmdCopyImageToBuffer2-commandBuffer-recording
  commandBuffer **must** be in the recording state
- VUID-vkCmdCopyImageToBuffer2-commandBuffer-cmdpool
  The VkCommandPool that commandBuffer was allocated from **must** support transfer, graphics, or compute operations
- VUID-vkCmdCopyImageToBuffer2-renderpass
  This command **must** only be called outside of a render pass instance
- VUID-vkCmdCopyImageToBuffer2-videocoding
  This command **must** only be called outside of a video coding scope

## Host Synchronization

- Host access to commandBuffer **must** be externally synchronized
- Host access to the VkCommandPool that commandBuffer was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Outside | Outside | Transfer<br>Graphics<br>Compute | Action |

The `VkCopyImageToBufferInfo2` structure is defined as:

```
typedef struct VkCopyImageToBufferInfo2 {
    VkStructureType              sType;
    const void*                  pNext;
    VkImage                      srcImage;
    VkImageLayout                srcImageLayout;
    VkBuffer                     dstBuffer;
    uint32_t                     regionCount;
    const VkBufferImageCopy2*    pRegions;
} VkCopyImageToBufferInfo2;
```

or the equivalent

```
// Provided by VK_KHR_copy_commands2
typedef VkCopyImageToBufferInfo2 VkCopyImageToBufferInfo2KHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `srcImage` is the source image.
- `srcImageLayout` is the layout of the source image subresources for the copy.
- `dstBuffer` is the destination buffer.
- `regionCount` is the number of regions to copy.
- `pRegions` is a pointer to an array of VkBufferImageCopy2 structures specifying the regions to copy.

## Valid Usage

- VUID-VkCopyImageToBufferInfo2-pRegions-04566
  The image region specified by each element of `pRegions` that does not contain VkCopyCommandTransformInfoQCOM in its `pNext` chain **must** be contained within the specified `imageSubresource` of `srcImage`
- VUID-VkCopyImageToBufferInfo2KHR-pRegions-04557
  If the image region specified by each element of `pRegions` contains

VkCopyCommandTransformInfoQCOM in its pNext chain, the rotated source region as described in [copies-buffers-images-rotation-addressing] **must** be contained within srcImage

- VUID-VkCopyImageToBufferInfo2KHR-pRegions-04558
  If any element of pRegions contains VkCopyCommandTransformInfoQCOM in its pNext chain, then srcImage **must** have a 1x1x1 texel block extent
- VUID-VkCopyImageToBufferInfo2KHR-pRegions-06205
  If any element of pRegions contains VkCopyCommandTransformInfoQCOM in its pNext chain, then srcImage **must** be of type VK_IMAGE_TYPE_2D
- VUID-VkCopyImageToBufferInfo2KHR-pRegions-06206
  If any element of pRegions contains VkCopyCommandTransformInfoQCOM in its pNext chain, then srcImage **must** not have a multi-planar format
- VUID-VkCopyImageToBufferInfo2-pRegions-00183
  dstBuffer **must** be large enough to contain all buffer locations that are accessed according to Buffer and Image Addressing, for each element of pRegions
- VUID-VkCopyImageToBufferInfo2-pRegions-00184
  The union of all source regions, and the union of all destination regions, specified by the elements of pRegions, **must** not overlap in memory
- VUID-VkCopyImageToBufferInfo2-srcImage-00186
  srcImage **must** have been created with VK_IMAGE_USAGE_TRANSFER_SRC_BIT usage flag
- VUID-VkCopyImageToBufferInfo2-srcImage-01998
  The format features of srcImage **must** contain VK_FORMAT_FEATURE_TRANSFER_SRC_BIT
- VUID-VkCopyImageToBufferInfo2-dstBuffer-00191
  dstBuffer **must** have been created with VK_BUFFER_USAGE_TRANSFER_DST_BIT usage flag
- VUID-VkCopyImageToBufferInfo2-dstBuffer-00192
  If dstBuffer is non-sparse then it **must** be bound completely and contiguously to a single VkDeviceMemory object
- VUID-VkCopyImageToBufferInfo2-srcImageLayout-00189
  srcImageLayout **must** specify the layout of the image subresources of srcImage specified in pRegions at the time this command is executed on a VkDevice
- VUID-VkCopyImageToBufferInfo2-srcImageLayout-01397
  srcImageLayout **must** be VK_IMAGE_LAYOUT_SHARED_PRESENT_KHR, VK_IMAGE_LAYOUT_TRANSFER_SRC_OPTIMAL, or VK_IMAGE_LAYOUT_GENERAL
- VUID-VkCopyImageToBufferInfo2-srcImage-07966
  If srcImage is non-sparse then the image or the specified *disjoint* plane **must** be bound completely and contiguously to a single VkDeviceMemory object
- VUID-VkCopyImageToBufferInfo2-imageSubresource-07967
  The imageSubresource.mipLevel member of each element of pRegions **must** be less than the mipLevels specified in VkImageCreateInfo when srcImage was created
- VUID-VkCopyImageToBufferInfo2-imageSubresource-07968
  If imageSubresource.layerCount is not VK_REMAINING_ARRAY_LAYERS,

imageSubresource.baseArrayLayer + imageSubresource.layerCount of each element of pRegions **must** be less than or equal to the arrayLayers specified in VkImageCreateInfo when srcImage was created

- VUID-VkCopyImageToBufferInfo2-srcImage-07969
  srcImage **must** not have been created with flags containing VK_IMAGE_CREATE_SUBSAMPLED_BIT_EXT

- VUID-VkCopyImageToBufferInfo2-srcImage-07973
  srcImage **must** have a sample count equal to VK_SAMPLE_COUNT_1_BIT

- VUID-VkCopyImageToBufferInfo2-srcImage-07979
  If srcImage is of type VK_IMAGE_TYPE_1D, then for each element of pRegions, imageOffset.y **must** be 0 and imageExtent.height **must** be 1

- VUID-VkCopyImageToBufferInfo2-imageOffset-09104
  For each element of pRegions, imageOffset.z and (imageExtent.depth + imageOffset.z) **must** both be greater than or equal to 0 and less than or equal to the depth of the specified imageSubresource of srcImage

- VUID-VkCopyImageToBufferInfo2-srcImage-07980
  If srcImage is of type VK_IMAGE_TYPE_1D or VK_IMAGE_TYPE_2D, then for each element of pRegions, imageOffset.z **must** be 0 and imageExtent.depth **must** be 1

- VUID-VkCopyImageToBufferInfo2-srcImage-07274
  For each element of pRegions, if VkCopyCommandTransformInfoQCOM::transform is equal to VK_SURFACE_TRANSFORM_IDENTITY_BIT_KHR or VK_SURFACE_TRANSFORM_ROTATE_270_BIT_KHR, imageOffset.x **must** be a multiple of the texel block extent width of the VkFormat of srcImage

- VUID-VkCopyImageToBufferInfo2-imageOffset-10051
  For each element of pRegions, if VkCopyCommandTransformInfoQCOM::transform is equal to VK_SURFACE_TRANSFORM_ROTATE_180_BIT_KHR or VK_SURFACE_TRANSFORM_ROTATE_90_BIT_KHR, and imageOffset.x does not equal the width of the subresource specified by imageSubresource, imageOffset.x **must** be a multiple of the texel block extent width of the VkFormat of srcImage

- VUID-VkCopyImageToBufferInfo2-srcImage-07275
  For each element of pRegions, if VkCopyCommandTransformInfoQCOM::transform is equal to VK_SURFACE_TRANSFORM_IDENTITY_BIT_KHR or VK_SURFACE_TRANSFORM_ROTATE_90_BIT_KHR, imageOffset.y **must** be a multiple of the texel block extent height of the VkFormat of srcImage

- VUID-VkCopyImageToBufferInfo2-imageOffset-10052
  For each element of pRegions, if VkCopyCommandTransformInfoQCOM::transform is equal to VK_SURFACE_TRANSFORM_ROTATE_270_BIT_KHR or VK_SURFACE_TRANSFORM_ROTATE_180_BIT_KHR, and imageOffset.y does not equal the height of the subresource specified by imageSubresource, imageOffset.y **must** be a multiple of the texel block extent height of the VkFormat of srcImage

- VUID-VkCopyImageToBufferInfo2-srcImage-07276
  For each element of pRegions, imageOffset.z **must** be a multiple of the texel block extent depth of the VkFormat of srcImage

- VUID-VkCopyImageToBufferInfo2-srcImage-00207
  For each element of `pRegions`, if VkCopyCommandTransformInfoQCOM::`transform` is equal to `VK_SURFACE_TRANSFORM_IDENTITY_BIT_KHR`, the sum of `imageOffset.x` and `extent.width` does not equal the width of the subresource specified by `imageSubresource`, `extent.width` **must** be a multiple of the texel block extent width of the VkFormat of `srcImage`
- VUID-VkCopyImageToBufferInfo2-imageOffset-10053
  For each element of `pRegions`, if VkCopyCommandTransformInfoQCOM::`transform` is equal to `VK_SURFACE_TRANSFORM_ROTATE_90_BIT_KHR`, the difference of `imageOffset.x` and `extent.height` **must** be a multiple of the texel block extent width of the VkFormat of `srcImage`
- VUID-VkCopyImageToBufferInfo2-imageOffset-10054
  For each element of `pRegions`, if VkCopyCommandTransformInfoQCOM::`transform` is equal to `VK_SURFACE_TRANSFORM_ROTATE_180_BIT_KHR`, the difference of `imageOffset.x` and `extent.width` **must** be a multiple of the texel block extent width of the VkFormat of `srcImage`
- VUID-VkCopyImageToBufferInfo2-imageOffset-10055
  For each element of `pRegions`, if VkCopyCommandTransformInfoQCOM::`transform` is equal to `VK_SURFACE_TRANSFORM_ROTATE_270_BIT_KHR`, the sum of `imageOffset.x` and `extent.height` does not equal the width of the subresource specified by `imageSubresource`, `extent.height` **must** be a multiple of the texel block extent width of the VkFormat of `srcImage`
- VUID-VkCopyImageToBufferInfo2-srcImage-00208
  For each element of `pRegions`, if VkCopyCommandTransformInfoQCOM::`transform` is equal to `VK_SURFACE_TRANSFORM_IDENTITY_BIT_KHR`, and the sum of `imageOffset.y` and `extent.height` does not equal the height of the subresource specified by `imageSubresource`, `extent.height` **must** be a multiple of the texel block extent height of the VkFormat of `srcImage`
- VUID-VkCopyImageToBufferInfo2-imageOffset-10056
  For each element of `pRegions`, if VkCopyCommandTransformInfoQCOM::`transform` is equal to `VK_SURFACE_TRANSFORM_ROTATE_90_BIT_KHR`, the sum of `imageOffset.y` and `extent.width` does not equal the height of the subresource specified by `imageSubresource`, `extent.width` **must** be a multiple of the texel block extent height of the VkFormat of `srcImage`
- VUID-VkCopyImageToBufferInfo2-imageOffset-10057
  For each element of `pRegions`, if VkCopyCommandTransformInfoQCOM::`transform` is equal to `VK_SURFACE_TRANSFORM_ROTATE_180_BIT_KHR`, the difference of `imageOffset.y` and `extent.height` **must** be a multiple of the texel block extent height of the VkFormat of `srcImage`
- VUID-VkCopyImageToBufferInfo2-imageOffset-10058
  For each element of `pRegions`, if VkCopyCommandTransformInfoQCOM::`transform` is equal to `VK_SURFACE_TRANSFORM_ROTATE_270_BIT_KHR`, the difference of `imageOffset.y` and `extent.width` **must** be a multiple of the texel block extent height of the VkFormat of `srcImage`
- VUID-VkCopyImageToBufferInfo2-srcImage-00209
  For each element of `pRegions`, if the sum of `imageOffset.z` and `extent.depth` does not equal the depth of the subresource specified by `srcSubresource`, `extent.depth` **must** be a multiple of the texel block extent depth of the VkFormat of `srcImage`

- VUID-VkCopyImageToBufferInfo2-imageSubresource-09105
  For each element of `pRegions`, `imageSubresource.aspectMask` **must** specify aspects present in `srcImage`
- VUID-VkCopyImageToBufferInfo2-srcImage-07981
  If `srcImage` has a multi-planar image format, then for each element of `pRegions`, `imageSubresource.aspectMask` **must** be a single valid multi-planar aspect mask bit
- VUID-VkCopyImageToBufferInfo2-srcImage-07983
  If `srcImage` is of type `VK_IMAGE_TYPE_3D`, for each element of `pRegions`, `imageSubresource.baseArrayLayer` **must** be `0` and `imageSubresource.layerCount` **must** be `1`
- VUID-VkCopyImageToBufferInfo2-bufferRowLength-09106
  For each element of `pRegions`, `bufferRowLength` **must** be a multiple of the texel block extent width of the VkFormat of `srcImage`
- VUID-VkCopyImageToBufferInfo2-bufferImageHeight-09107
  For each element of `pRegions`, `bufferImageHeight` **must** be a multiple of the texel block extent height of the VkFormat of `srcImage`
- VUID-VkCopyImageToBufferInfo2-bufferRowLength-09108
  For each element of `pRegions`, `bufferRowLength` divided by the texel block extent width and then multiplied by the texel block size of `srcImage` **must** be less than or equal to $2^{31}$-1
- VUID-VkCopyImageToBufferInfo2-srcImage-07975
  If `srcImage` does not have either a depth/stencil format or a multi-planar format, then for each element of `pRegions`, `bufferOffset` **must** be a multiple of the texel block size
- VUID-VkCopyImageToBufferInfo2-srcImage-07976
  If `srcImage` has a multi-planar format, then for each element of `pRegions`, `bufferOffset` **must** be a multiple of the element size of the compatible format for the format and the `aspectMask` of the `imageSubresource` as defined in Compatible Formats of Planes of Multi-Planar Formats
- VUID-VkCopyImageToBufferInfo2-srcImage-07978
  If `srcImage` has a depth/stencil format, the `bufferOffset` member of any element of `pRegions` **must** be a multiple of `4`
- VUID-VkCopyImageToBufferInfo2-imageOffset-00197
  For each element of `pRegions` not containing `VkCopyCommandTransformInfoQCOM` in its `pNext` chain, `imageOffset.x` and (`imageExtent.width` + `imageOffset.x`) **must** both be greater than or equal to `0` and less than or equal to the width of the specified `imageSubresource` of `srcImage`
- VUID-VkCopyImageToBufferInfo2-imageOffset-00198
  For each element of `pRegions` not containing `VkCopyCommandTransformInfoQCOM` in its `pNext` chain, `imageOffset.y` and (`imageExtent.height` + `imageOffset.y`) **must** both be greater than or equal to `0` and less than or equal to the height of the specified `imageSubresource` of `srcImage`

## Valid Usage (Implicit)

- VUID-VkCopyImageToBufferInfo2-sType-sType

sType **must** be VK_STRUCTURE_TYPE_COPY_IMAGE_TO_BUFFER_INFO_2

- VUID-VkCopyImageToBufferInfo2-pNext-pNext
  pNext **must** be NULL
- VUID-VkCopyImageToBufferInfo2-srcImage-parameter
  srcImage **must** be a valid VkImage handle
- VUID-VkCopyImageToBufferInfo2-srcImageLayout-parameter
  srcImageLayout **must** be a valid VkImageLayout value
- VUID-VkCopyImageToBufferInfo2-dstBuffer-parameter
  dstBuffer **must** be a valid VkBuffer handle
- VUID-VkCopyImageToBufferInfo2-pRegions-parameter
  pRegions **must** be a valid pointer to an array of regionCount valid VkBufferImageCopy2 structures
- VUID-VkCopyImageToBufferInfo2-regionCount-arraylength
  regionCount **must** be greater than 0
- VUID-VkCopyImageToBufferInfo2-commonparent
  Both of dstBuffer, and srcImage **must** have been created, allocated, or retrieved from the same VkDevice

For both vkCmdCopyBufferToImage2KHR and vkCmdCopyImageToBuffer2KHR, each element of pRegions is a structure defined as:

```
typedef struct VkBufferImageCopy2 {
    VkStructureType             sType;
    const void*                 pNext;
    VkDeviceSize                bufferOffset;
    uint32_t                    bufferRowLength;
    uint32_t                    bufferImageHeight;
    VkImageSubresourceLayers    imageSubresource;
    VkOffset3D                  imageOffset;
    VkExtent3D                  imageExtent;
} VkBufferImageCopy2;
```

or the equivalent

```
// Provided by VK_KHR_copy_commands2
typedef VkBufferImageCopy2 VkBufferImageCopy2KHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- bufferOffset is the offset in bytes from the start of the buffer object where the image data is copied from or to.
- bufferRowLength and bufferImageHeight specify in texels a subregion of a larger two- or three-

dimensional image in buffer memory, and control the addressing calculations. If either of these values is zero, that aspect of the buffer memory is considered to be tightly packed according to the `imageExtent`.

- `imageSubresource` is a VkImageSubresourceLayers used to specify the specific image subresources of the image used for the source or destination image data.
- `imageOffset` selects the initial `x`, `y`, `z` offsets in texels of the sub-region of the source or destination image data.
- `imageExtent` is the size in texels of the image to copy in `width`, `height` and `depth`.

This structure is functionally identical to VkBufferImageCopy, but adds `sType` and `pNext` parameters, allowing it to be more easily extended.

### Valid Usage

- VUID-VkBufferImageCopy2-bufferRowLength-09101
  `bufferRowLength` **must** be `0`, or greater than or equal to the `width` member of `imageExtent`
- VUID-VkBufferImageCopy2-bufferImageHeight-09102
  `bufferImageHeight` **must** be `0`, or greater than or equal to the `height` member of `imageExtent`
- VUID-VkBufferImageCopy2-aspectMask-09103
  The `aspectMask` member of `imageSubresource` **must** only have a single bit set
- VUID-VkBufferImageCopy2-imageExtent-06659
  `imageExtent.width` **must** not be 0
- VUID-VkBufferImageCopy2-imageExtent-06660
  `imageExtent.height` **must** not be 0
- VUID-VkBufferImageCopy2-imageExtent-06661
  `imageExtent.depth` **must** not be 0

### Valid Usage (Implicit)

- VUID-VkBufferImageCopy2-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_BUFFER_IMAGE_COPY_2`
- VUID-VkBufferImageCopy2-pNext-pNext
  `pNext` **must** be `NULL` or a pointer to a valid instance of VkCopyCommandTransformInfoQCOM
- VUID-VkBufferImageCopy2-sType-unique
  The `sType` value of each struct in the `pNext` chain **must** be unique
- VUID-VkBufferImageCopy2-imageSubresource-parameter
  `imageSubresource` **must** be a valid VkImageSubresourceLayers structure

The `VkCopyCommandTransformInfoQCOM` structure is defined as:

```
// Provided by VK_QCOM_rotated_copy_commands
typedef struct VkCopyCommandTransformInfoQCOM {
    VkStructureType                 sType;
    const void*                     pNext;
    VkSurfaceTransformFlagBitsKHR   transform;
} VkCopyCommandTransformInfoQCOM;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `transform` is a VkSurfaceTransformFlagBitsKHR value describing the transform to be applied.

Including this structure in the `pNext` chain of VkBufferImageCopy2 defines a rotation to be performed when copying between an image and a buffer. Including this structure in the `pNext` chain of VkBlitImageInfo2 defines a rotation to be performed when blitting between two images. If this structure is not specified in either case, the implementation behaves as if it was specified with a `transform` equal to `VK_SURFACE_TRANSFORM_IDENTITY_BIT_KHR`.

Specifying a transform for a copy between an image and a buffer rotates the region accessed in the image around the offset. Specifying a transform for a blit performs a similar transform as described in Image Blits with Scaling and Rotation.

Rotations other than `VK_SURFACE_TRANSFORM_IDENTITY_BIT_KHR` **can** only be specified for single-plane 2D images with a 1x1x1 texel block extent.

### Valid Usage

- VUID-VkCopyCommandTransformInfoQCOM-transform-04560
  `transform` **must** be `VK_SURFACE_TRANSFORM_IDENTITY_BIT_KHR`, `VK_SURFACE_TRANSFORM_ROTATE_90_BIT_KHR`, `VK_SURFACE_TRANSFORM_ROTATE_180_BIT_KHR`, or `VK_SURFACE_TRANSFORM_ROTATE_270_BIT_KHR`

### Valid Usage (Implicit)

- VUID-VkCopyCommandTransformInfoQCOM-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_COPY_COMMAND_TRANSFORM_INFO_QCOM`

The following commands can be used to copy between host memory and images.

To copy data from host memory to an image object, call:

```
// Provided by VK_EXT_host_image_copy
VkResult vkCopyMemoryToImageEXT(
    VkDevice                                    device,
    const VkCopyMemoryToImageInfoEXT*           pCopyMemoryToImageInfo);
```

- `device` is the device which owns `pCopyMemoryToImageInfo->dstImage`.
- `pCopyMemoryToImageInfo` is a pointer to a VkCopyMemoryToImageInfoEXT structure describing the copy parameters.

This command is functionally similar to vkCmdCopyBufferToImage2KHR, except it is executed on the host and reads from host memory instead of a buffer. The memory of `pCopyMemoryToImageInfo->dstImage` is accessed by the host as if coherent.

| | |
|---|---|
| **NOTE** | Because queue submissions automatically make host memory visible to the device, there would not be a need for a memory barrier before using the results of this copy operation on the device. |

## Valid Usage

- VUID-vkCopyMemoryToImageEXT-hostImageCopy-09058
  The `hostImageCopy` feature **must** be enabled

## Valid Usage (Implicit)

- VUID-vkCopyMemoryToImageEXT-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkCopyMemoryToImageEXT-pCopyMemoryToImageInfo-parameter
  `pCopyMemoryToImageInfo` **must** be a valid pointer to a valid VkCopyMemoryToImageInfoEXT structure

## Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- `VK_ERROR_INITIALIZATION_FAILED`
- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_MEMORY_MAP_FAILED`

The `VkCopyMemoryToImageInfoEXT` structure is defined as:

```
// Provided by VK_EXT_host_image_copy
typedef struct VkCopyMemoryToImageInfoEXT {
    VkStructureType                 sType;
    const void*                     pNext;
    VkHostImageCopyFlagsEXT         flags;
    VkImage                         dstImage;
    VkImageLayout                   dstImageLayout;
    uint32_t                        regionCount;
    const VkMemoryToImageCopyEXT*   pRegions;
} VkCopyMemoryToImageInfoEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is a bitmask of VkHostImageCopyFlagBitsEXT values describing additional copy parameters.
- `dstImage` is the destination image.
- `dstImageLayout` is the layout of the destination image subresources for the copy.
- `regionCount` is the number of regions to copy.
- `pRegions` is a pointer to an array of VkMemoryToImageCopyEXT structures specifying the regions to copy.

`vkCopyMemoryToImageEXT` does not check whether the device memory associated with `dstImage` is currently in use before performing the copy. The application **must** guarantee that any previously submitted command that reads from or writes to the copy regions has completed before the host performs the copy.

Copy regions for the image **must** be aligned to a multiple of the texel block extent in each dimension, except at the edges of the image, where region extents **must** match the edge of the image.

### Valid Usage

- VUID-VkCopyMemoryToImageInfoEXT-dstImage-09109
  If `dstImage` is sparse then all memory ranges accessed by the copy command **must** be bound as described in Binding Resource Memory
- VUID-VkCopyMemoryToImageInfoEXT-dstImage-09111
  If the stencil aspect of `dstImage` is accessed, and `dstImage` was not created with separate stencil usage, `dstImage` **must** have been created with `VK_IMAGE_USAGE_HOST_TRANSFER_BIT_EXT` set in VkImageCreateInfo::`usage`
- VUID-VkCopyMemoryToImageInfoEXT-dstImage-09112
  If the stencil aspect of `dstImage` is accessed, and `dstImage` was created with separate stencil usage, `dstImage` **must** have been created with `VK_IMAGE_USAGE_HOST_TRANSFER_BIT_EXT` set in VkImageStencilUsageCreateInfo::`stencilUsage`
- VUID-VkCopyMemoryToImageInfoEXT-dstImage-09113

If non-stencil aspects of dstImage are accessed, dstImage **must** have been created with VK_IMAGE_USAGE_HOST_TRANSFER_BIT_EXT set in VkImageCreateInfo::usage

- VUID-VkCopyMemoryToImageInfoEXT-imageOffset-09114
  If flags contains VK_HOST_IMAGE_COPY_MEMCPY_EXT, the x, y, and z members of the imageOffset member of each element of pRegions **must** be 0
- VUID-VkCopyMemoryToImageInfoEXT-dstImage-09115
  If flags contains VK_HOST_IMAGE_COPY_MEMCPY_EXT, the imageExtent member of each element of pRegions **must** equal the extents of dstImage identified by imageSubresource
- VUID-VkCopyMemoryToImageInfoEXT-dstImage-07966
  If dstImage is non-sparse then the image or the specified *disjoint* plane **must** be bound completely and contiguously to a single VkDeviceMemory object
- VUID-VkCopyMemoryToImageInfoEXT-imageSubresource-07967
  The imageSubresource.mipLevel member of each element of pRegions **must** be less than the mipLevels specified in VkImageCreateInfo when dstImage was created
- VUID-VkCopyMemoryToImageInfoEXT-imageSubresource-07968
  If imageSubresource.layerCount is not VK_REMAINING_ARRAY_LAYERS, imageSubresource.baseArrayLayer + imageSubresource.layerCount of each element of pRegions **must** be less than or equal to the arrayLayers specified in VkImageCreateInfo when dstImage was created
- VUID-VkCopyMemoryToImageInfoEXT-dstImage-07969
  dstImage **must** not have been created with flags containing VK_IMAGE_CREATE_SUBSAMPLED_BIT_EXT
- VUID-VkCopyMemoryToImageInfoEXT-imageSubresource-07970
  The image region specified by each element of pRegions **must** be contained within the specified imageSubresource of dstImage
- VUID-VkCopyMemoryToImageInfoEXT-imageSubresource-07971
  For each element of pRegions, imageOffset.x and (imageExtent.width + imageOffset.x) **must** both be greater than or equal to 0 and less than or equal to the width of the specified imageSubresource of dstImage
- VUID-VkCopyMemoryToImageInfoEXT-imageSubresource-07972
  For each element of pRegions, imageOffset.y and (imageExtent.height + imageOffset.y) **must** both be greater than or equal to 0 and less than or equal to the height of the specified imageSubresource of dstImage
- VUID-VkCopyMemoryToImageInfoEXT-dstImage-07973
  dstImage **must** have a sample count equal to VK_SAMPLE_COUNT_1_BIT
- VUID-VkCopyMemoryToImageInfoEXT-dstImage-07979
  If dstImage is of type VK_IMAGE_TYPE_1D, then for each element of pRegions, imageOffset.y **must** be 0 and imageExtent.height **must** be 1
- VUID-VkCopyMemoryToImageInfoEXT-imageOffset-09104
  For each element of pRegions, imageOffset.z and (imageExtent.depth + imageOffset.z) **must** both be greater than or equal to 0 and less than or equal to the depth of the specified

imageSubresource of dstImage

- VUID-VkCopyMemoryToImageInfoEXT-dstImage-07980
  If dstImage is of type VK_IMAGE_TYPE_1D or VK_IMAGE_TYPE_2D, then for each element of pRegions, imageOffset.z **must** be 0 and imageExtent.depth **must** be 1
- VUID-VkCopyMemoryToImageInfoEXT-dstImage-07274
  For each element of pRegions, if VkCopyCommandTransformInfoQCOM::transform is equal to VK_SURFACE_TRANSFORM_IDENTITY_BIT_KHR or VK_SURFACE_TRANSFORM_ROTATE_270_BIT_KHR, imageOffset.x **must** be a multiple of the texel block extent width of the VkFormat of dstImage
- VUID-VkCopyMemoryToImageInfoEXT-imageOffset-10051
  For each element of pRegions, if VkCopyCommandTransformInfoQCOM::transform is equal to VK_SURFACE_TRANSFORM_ROTATE_180_BIT_KHR or VK_SURFACE_TRANSFORM_ROTATE_90_BIT_KHR, and imageOffset.x does not equal the width of the subresource specified by imageSubresource, imageOffset.x **must** be a multiple of the texel block extent width of the VkFormat of dstImage
- VUID-VkCopyMemoryToImageInfoEXT-dstImage-07275
  For each element of pRegions, if VkCopyCommandTransformInfoQCOM::transform is equal to VK_SURFACE_TRANSFORM_IDENTITY_BIT_KHR or VK_SURFACE_TRANSFORM_ROTATE_90_BIT_KHR, imageOffset.y **must** be a multiple of the texel block extent height of the VkFormat of dstImage
- VUID-VkCopyMemoryToImageInfoEXT-imageOffset-10052
  For each element of pRegions, if VkCopyCommandTransformInfoQCOM::transform is equal to VK_SURFACE_TRANSFORM_ROTATE_270_BIT_KHR or VK_SURFACE_TRANSFORM_ROTATE_180_BIT_KHR, and imageOffset.y does not equal the height of the subresource specified by imageSubresource, imageOffset.y **must** be a multiple of the texel block extent height of the VkFormat of dstImage
- VUID-VkCopyMemoryToImageInfoEXT-dstImage-07276
  For each element of pRegions, imageOffset.z **must** be a multiple of the texel block extent depth of the VkFormat of dstImage
- VUID-VkCopyMemoryToImageInfoEXT-dstImage-00207
  For each element of pRegions, if VkCopyCommandTransformInfoQCOM::transform is equal to VK_SURFACE_TRANSFORM_IDENTITY_BIT_KHR, the sum of imageOffset.x and extent.width does not equal the width of the subresource specified by imageSubresource, extent.width **must** be a multiple of the texel block extent width of the VkFormat of dstImage
- VUID-VkCopyMemoryToImageInfoEXT-imageOffset-10053
  For each element of pRegions, if VkCopyCommandTransformInfoQCOM::transform is equal to VK_SURFACE_TRANSFORM_ROTATE_90_BIT_KHR, the difference of imageOffset.x and extent.height **must** be a multiple of the texel block extent width of the VkFormat of dstImage
- VUID-VkCopyMemoryToImageInfoEXT-imageOffset-10054
  For each element of pRegions, if VkCopyCommandTransformInfoQCOM::transform is equal to VK_SURFACE_TRANSFORM_ROTATE_180_BIT_KHR, the difference of imageOffset.x and extent.width **must** be a multiple of the texel block extent width of the VkFormat of dstImage

- VUID-VkCopyMemoryToImageInfoEXT-imageOffset-10055
  For each element of `pRegions`, if VkCopyCommandTransformInfoQCOM::`transform` is equal to `VK_SURFACE_TRANSFORM_ROTATE_270_BIT_KHR`, the sum of `imageOffset.x` and `extent.height` does not equal the width of the subresource specified by `imageSubresource`, `extent.height` **must** be a multiple of the texel block extent width of the VkFormat of `dstImage`
- VUID-VkCopyMemoryToImageInfoEXT-dstImage-00208
  For each element of `pRegions`, if VkCopyCommandTransformInfoQCOM::`transform` is equal to `VK_SURFACE_TRANSFORM_IDENTITY_BIT_KHR`, and the sum of `imageOffset.y` and `extent.height` does not equal the height of the subresource specified by `imageSubresource`, `extent.height` **must** be a multiple of the texel block extent height of the VkFormat of `dstImage`
- VUID-VkCopyMemoryToImageInfoEXT-imageOffset-10056
  For each element of `pRegions`, if VkCopyCommandTransformInfoQCOM::`transform` is equal to `VK_SURFACE_TRANSFORM_ROTATE_90_BIT_KHR`, the sum of `imageOffset.y` and `extent.width` does not equal the height of the subresource specified by `imageSubresource`, `extent.width` **must** be a multiple of the texel block extent height of the VkFormat of `dstImage`
- VUID-VkCopyMemoryToImageInfoEXT-imageOffset-10057
  For each element of `pRegions`, if VkCopyCommandTransformInfoQCOM::`transform` is equal to `VK_SURFACE_TRANSFORM_ROTATE_180_BIT_KHR`, the difference of `imageOffset.y` and `extent.height` **must** be a multiple of the texel block extent height of the VkFormat of `dstImage`
- VUID-VkCopyMemoryToImageInfoEXT-imageOffset-10058
  For each element of `pRegions`, if VkCopyCommandTransformInfoQCOM::`transform` is equal to `VK_SURFACE_TRANSFORM_ROTATE_270_BIT_KHR`, the difference of `imageOffset.y` and `extent.width` **must** be a multiple of the texel block extent height of the VkFormat of `dstImage`
- VUID-VkCopyMemoryToImageInfoEXT-dstImage-00209
  For each element of `pRegions`, if the sum of `imageOffset.z` and `extent.depth` does not equal the depth of the subresource specified by `srcSubresource`, `extent.depth` **must** be a multiple of the texel block extent depth of the VkFormat of `dstImage`
- VUID-VkCopyMemoryToImageInfoEXT-imageSubresource-09105
  For each element of `pRegions`, `imageSubresource.aspectMask` **must** specify aspects present in `dstImage`
- VUID-VkCopyMemoryToImageInfoEXT-dstImage-07981
  If `dstImage` has a multi-planar image format, then for each element of `pRegions`, `imageSubresource.aspectMask` **must** be a single valid multi-planar aspect mask bit
- VUID-VkCopyMemoryToImageInfoEXT-dstImage-07983
  If `dstImage` is of type `VK_IMAGE_TYPE_3D`, for each element of `pRegions`, `imageSubresource.baseArrayLayer` **must** be `0` and `imageSubresource.layerCount` **must** be `1`
- VUID-VkCopyMemoryToImageInfoEXT-memoryRowLength-09106
  For each element of `pRegions`, `memoryRowLength` **must** be a multiple of the texel block extent width of the VkFormat of `dstImage`
- VUID-VkCopyMemoryToImageInfoEXT-memoryImageHeight-09107
  For each element of `pRegions`, `memoryImageHeight` **must** be a multiple of the texel block

extent height of the VkFormat of dstImage

- VUID-VkCopyMemoryToImageInfoEXT-memoryRowLength-09108
  For each element of pRegions, memoryRowLength divided by the texel block extent width and then multiplied by the texel block size of dstImage **must** be less than or equal to $2^{31}$-1
- VUID-VkCopyMemoryToImageInfoEXT-dstImageLayout-09059
  dstImageLayout **must** specify the current layout of the image subresources of dstImage specified in pRegions
- VUID-VkCopyMemoryToImageInfoEXT-dstImageLayout-09060
  dstImageLayout **must** be one of the image layouts returned in VkPhysicalDeviceHostImageCopyPropertiesEXT::pCopyDstLayouts
- VUID-VkCopyMemoryToImageInfoEXT-flags-09393
  If flags includes VK_HOST_IMAGE_COPY_MEMCPY_EXT, for each region in pRegions, memoryRowLength and memoryImageHeight **must** both be 0

## Valid Usage (Implicit)

- VUID-VkCopyMemoryToImageInfoEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_COPY_MEMORY_TO_IMAGE_INFO_EXT
- VUID-VkCopyMemoryToImageInfoEXT-pNext-pNext
  pNext **must** be NULL
- VUID-VkCopyMemoryToImageInfoEXT-flags-parameter
  flags **must** be a valid combination of VkHostImageCopyFlagBitsEXT values
- VUID-VkCopyMemoryToImageInfoEXT-dstImage-parameter
  dstImage **must** be a valid VkImage handle
- VUID-VkCopyMemoryToImageInfoEXT-dstImageLayout-parameter
  dstImageLayout **must** be a valid VkImageLayout value
- VUID-VkCopyMemoryToImageInfoEXT-pRegions-parameter
  pRegions **must** be a valid pointer to an array of regionCount valid VkMemoryToImageCopyEXT structures
- VUID-VkCopyMemoryToImageInfoEXT-regionCount-arraylength
  regionCount **must** be greater than 0

Each element of VkCopyMemoryToImageInfoEXT::pRegions is a structure defined as:

```
// Provided by VK_EXT_host_image_copy
typedef struct VkMemoryToImageCopyEXT {
    VkStructureType             sType;
    const void*                 pNext;
    const void*                 pHostPointer;
    uint32_t                    memoryRowLength;
    uint32_t                    memoryImageHeight;
    VkImageSubresourceLayers    imageSubresource;
    VkOffset3D                  imageOffset;
    VkExtent3D                  imageExtent;
} VkMemoryToImageCopyEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `pHostPointer` is the host memory address which is the source of the copy.
- `memoryRowLength` and `memoryImageHeight` specify in texels a subregion of a larger two- or three-dimensional image in host memory, and control the addressing calculations. If either of these values is zero, that aspect of the host memory is considered to be tightly packed according to the `imageExtent`.
- `imageSubresource` is a VkImageSubresourceLayers used to specify the specific image subresources of the image used for the source or destination image data.
- `imageOffset` selects the initial `x`, `y`, `z` offsets in texels of the sub-region of the destination image data.
- `imageExtent` is the size in texels of the image to copy in `width`, `height` and `depth`.

This structure is functionally similar to VkBufferImageCopy2, except it defines host memory as the source of copy instead of a buffer. In particular, the same data packing rules and restrictions as that structure apply here as well.

## Valid Usage

- VUID-VkMemoryToImageCopyEXT-pHostPointer-09061
  `pHostPointer` **must** point to memory that is large enough to contain all memory locations that are accessed according to Buffer and Image Addressing, for each element of `pRegions`
- VUID-VkMemoryToImageCopyEXT-pRegions-09062
  The union of all source regions, and the union of all destination regions, specified by the elements of `pRegions`, **must** not overlap in memory
- VUID-VkMemoryToImageCopyEXT-memoryRowLength-09101
  `memoryRowLength` **must** be `0`, or greater than or equal to the `width` member of `imageExtent`
- VUID-VkMemoryToImageCopyEXT-memoryImageHeight-09102
  `memoryImageHeight` **must** be `0`, or greater than or equal to the `height` member of `imageExtent`
- VUID-VkMemoryToImageCopyEXT-aspectMask-09103
  The `aspectMask` member of `imageSubresource` **must** only have a single bit set

- VUID-VkMemoryToImageCopyEXT-imageExtent-06659
  `imageExtent.width` **must** not be 0
- VUID-VkMemoryToImageCopyEXT-imageExtent-06660
  `imageExtent.height` **must** not be 0
- VUID-VkMemoryToImageCopyEXT-imageExtent-06661
  `imageExtent.depth` **must** not be 0

### Valid Usage (Implicit)

- VUID-VkMemoryToImageCopyEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_MEMORY_TO_IMAGE_COPY_EXT`
- VUID-VkMemoryToImageCopyEXT-pNext-pNext
  `pNext` **must** be `NULL`
- VUID-VkMemoryToImageCopyEXT-pHostPointer-parameter
  `pHostPointer` **must** be a pointer value
- VUID-VkMemoryToImageCopyEXT-imageSubresource-parameter
  `imageSubresource` **must** be a valid VkImageSubresourceLayers structure

To copy data from an image object to host memory, call:

```
// Provided by VK_EXT_host_image_copy
VkResult vkCopyImageToMemoryEXT(
    VkDevice                                    device,
    const VkCopyImageToMemoryInfoEXT*           pCopyImageToMemoryInfo);
```

- `device` is the device which owns `pCopyImageToMemoryInfo->srcImage`.
- `pCopyImageToMemoryInfo` is a pointer to a VkCopyImageToMemoryInfoEXT structure describing the copy parameters.

This command is functionally similar to vkCmdCopyImageToBuffer2KHR, except it is executed on the host and writes to host memory instead of a buffer. The memory of `pCopyImageToMemoryInfo->srcImage` is accessed by the host as if coherent.

**NOTE** If the device has written to the image memory, it is not automatically made available to the host. Before this copy command can be called, a memory barrier for this image **must** have been issued on the device with the second synchronization scope including `VK_PIPELINE_STAGE_HOST_BIT` and `VK_ACCESS_HOST_READ_BIT`.

### Valid Usage

- VUID-vkCopyImageToMemoryEXT-hostImageCopy-09063
  The `hostImageCopy` feature **must** be enabled

### Valid Usage (Implicit)

- VUID-vkCopyImageToMemoryEXT-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkCopyImageToMemoryEXT-pCopyImageToMemoryInfo-parameter
  `pCopyImageToMemoryInfo` **must** be a valid pointer to a valid VkCopyImageToMemoryInfoEXT structure

### Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- `VK_ERROR_INITIALIZATION_FAILED`
- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_MEMORY_MAP_FAILED`

The `VkCopyImageToMemoryInfoEXT` structure is defined as:

```
// Provided by VK_EXT_host_image_copy
typedef struct VkCopyImageToMemoryInfoEXT {
    VkStructureType                 sType;
    const void*                     pNext;
    VkHostImageCopyFlagsEXT         flags;
    VkImage                         srcImage;
    VkImageLayout                   srcImageLayout;
    uint32_t                        regionCount;
    const VkImageToMemoryCopyEXT*   pRegions;
} VkCopyImageToMemoryInfoEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is a bitmask of VkHostImageCopyFlagBitsEXT values describing additional copy parameters.
- `srcImage` is the source image.
- `srcImageLayout` is the layout of the source image subresources for the copy.
- `regionCount` is the number of regions to copy.
- `pRegions` is a pointer to an array of VkImageToMemoryCopyEXT structures specifying the regions to copy.

vkCopyImageToMemoryEXT does not check whether the device memory associated with srcImage is currently in use before performing the copy. The application **must** guarantee that any previously submitted command that writes to the copy regions has completed before the host performs the copy.

Copy regions for the image **must** be aligned to a multiple of the texel block extent in each dimension, except at the edges of the image, where region extents **must** match the edge of the image.

## Valid Usage

- VUID-VkCopyImageToMemoryInfoEXT-srcImage-09109
  If srcImage is sparse then all memory ranges accessed by the copy command **must** be bound as described in Binding Resource Memory
- VUID-VkCopyImageToMemoryInfoEXT-srcImage-09111
  If the stencil aspect of srcImage is accessed, and srcImage was not created with separate stencil usage, srcImage **must** have been created with VK_IMAGE_USAGE_HOST_TRANSFER_BIT_EXT set in VkImageCreateInfo::usage
- VUID-VkCopyImageToMemoryInfoEXT-srcImage-09112
  If the stencil aspect of srcImage is accessed, and srcImage was created with separate stencil usage, srcImage **must** have been created with VK_IMAGE_USAGE_HOST_TRANSFER_BIT_EXT set in VkImageStencilUsageCreateInfo::stencilUsage
- VUID-VkCopyImageToMemoryInfoEXT-srcImage-09113
  If non-stencil aspects of srcImage are accessed, srcImage **must** have been created with VK_IMAGE_USAGE_HOST_TRANSFER_BIT_EXT set in VkImageCreateInfo::usage
- VUID-VkCopyImageToMemoryInfoEXT-imageOffset-09114
  If flags contains VK_HOST_IMAGE_COPY_MEMCPY_EXT, the x, y, and z members of the imageOffset member of each element of pRegions **must** be 0
- VUID-VkCopyImageToMemoryInfoEXT-srcImage-09115
  If flags contains VK_HOST_IMAGE_COPY_MEMCPY_EXT, the imageExtent member of each element of pRegions **must** equal the extents of srcImage identified by imageSubresource
- VUID-VkCopyImageToMemoryInfoEXT-srcImage-07966
  If srcImage is non-sparse then the image or the specified *disjoint* plane **must** be bound completely and contiguously to a single VkDeviceMemory object
- VUID-VkCopyImageToMemoryInfoEXT-imageSubresource-07967
  The imageSubresource.mipLevel member of each element of pRegions **must** be less than the mipLevels specified in VkImageCreateInfo when srcImage was created
- VUID-VkCopyImageToMemoryInfoEXT-imageSubresource-07968
  If imageSubresource.layerCount is not VK_REMAINING_ARRAY_LAYERS, imageSubresource.baseArrayLayer + imageSubresource.layerCount of each element of pRegions **must** be less than or equal to the arrayLayers specified in VkImageCreateInfo when srcImage was created
- VUID-VkCopyImageToMemoryInfoEXT-srcImage-07969
  srcImage **must** not have been created with flags containing

VK_IMAGE_CREATE_SUBSAMPLED_BIT_EXT

- VUID-VkCopyImageToMemoryInfoEXT-imageSubresource-07970
  The image region specified by each element of pRegions **must** be contained within the specified imageSubresource of srcImage
- VUID-VkCopyImageToMemoryInfoEXT-imageSubresource-07971
  For each element of pRegions, imageOffset.x and (imageExtent.width + imageOffset.x) **must** both be greater than or equal to 0 and less than or equal to the width of the specified imageSubresource of srcImage
- VUID-VkCopyImageToMemoryInfoEXT-imageSubresource-07972
  For each element of pRegions, imageOffset.y and (imageExtent.height + imageOffset.y) **must** both be greater than or equal to 0 and less than or equal to the height of the specified imageSubresource of srcImage
- VUID-VkCopyImageToMemoryInfoEXT-srcImage-07973
  srcImage **must** have a sample count equal to VK_SAMPLE_COUNT_1_BIT
- VUID-VkCopyImageToMemoryInfoEXT-srcImage-07979
  If srcImage is of type VK_IMAGE_TYPE_1D, then for each element of pRegions, imageOffset.y **must** be 0 and imageExtent.height **must** be 1
- VUID-VkCopyImageToMemoryInfoEXT-imageOffset-09104
  For each element of pRegions, imageOffset.z and (imageExtent.depth + imageOffset.z) **must** both be greater than or equal to 0 and less than or equal to the depth of the specified imageSubresource of srcImage
- VUID-VkCopyImageToMemoryInfoEXT-srcImage-07980
  If srcImage is of type VK_IMAGE_TYPE_1D or VK_IMAGE_TYPE_2D, then for each element of pRegions, imageOffset.z **must** be 0 and imageExtent.depth **must** be 1
- VUID-VkCopyImageToMemoryInfoEXT-srcImage-07274
  For each element of pRegions, if VkCopyCommandTransformInfoQCOM::transform is equal to VK_SURFACE_TRANSFORM_IDENTITY_BIT_KHR or VK_SURFACE_TRANSFORM_ROTATE_270_BIT_KHR, imageOffset.x **must** be a multiple of the texel block extent width of the VkFormat of srcImage
- VUID-VkCopyImageToMemoryInfoEXT-imageOffset-10051
  For each element of pRegions, if VkCopyCommandTransformInfoQCOM::transform is equal to VK_SURFACE_TRANSFORM_ROTATE_180_BIT_KHR or VK_SURFACE_TRANSFORM_ROTATE_90_BIT_KHR, and imageOffset.x does not equal the width of the subresource specified by imageSubresource, imageOffset.x **must** be a multiple of the texel block extent width of the VkFormat of srcImage
- VUID-VkCopyImageToMemoryInfoEXT-srcImage-07275
  For each element of pRegions, if VkCopyCommandTransformInfoQCOM::transform is equal to VK_SURFACE_TRANSFORM_IDENTITY_BIT_KHR or VK_SURFACE_TRANSFORM_ROTATE_90_BIT_KHR, imageOffset.y **must** be a multiple of the texel block extent height of the VkFormat of srcImage
- VUID-VkCopyImageToMemoryInfoEXT-imageOffset-10052
  For each element of pRegions, if VkCopyCommandTransformInfoQCOM::transform is equal

to VK_SURFACE_TRANSFORM_ROTATE_270_BIT_KHR or VK_SURFACE_TRANSFORM_ROTATE_180_BIT_KHR, and imageOffset.y does not equal the height of the subresource specified by imageSubresource, imageOffset.y **must** be a multiple of the texel block extent height of the VkFormat of srcImage

- VUID-VkCopyImageToMemoryInfoEXT-srcImage-07276
  For each element of pRegions, imageOffset.z **must** be a multiple of the texel block extent depth of the VkFormat of srcImage
- VUID-VkCopyImageToMemoryInfoEXT-srcImage-00207
  For each element of pRegions, if VkCopyCommandTransformInfoQCOM::transform is equal to VK_SURFACE_TRANSFORM_IDENTITY_BIT_KHR, the sum of imageOffset.x and extent.width does not equal the width of the subresource specified by imageSubresource, extent.width **must** be a multiple of the texel block extent width of the VkFormat of srcImage
- VUID-VkCopyImageToMemoryInfoEXT-imageOffset-10053
  For each element of pRegions, if VkCopyCommandTransformInfoQCOM::transform is equal to VK_SURFACE_TRANSFORM_ROTATE_90_BIT_KHR, the difference of imageOffset.x and extent.height **must** be a multiple of the texel block extent width of the VkFormat of srcImage
- VUID-VkCopyImageToMemoryInfoEXT-imageOffset-10054
  For each element of pRegions, if VkCopyCommandTransformInfoQCOM::transform is equal to VK_SURFACE_TRANSFORM_ROTATE_180_BIT_KHR, the difference of imageOffset.x and extent.width **must** be a multiple of the texel block extent width of the VkFormat of srcImage
- VUID-VkCopyImageToMemoryInfoEXT-imageOffset-10055
  For each element of pRegions, if VkCopyCommandTransformInfoQCOM::transform is equal to VK_SURFACE_TRANSFORM_ROTATE_270_BIT_KHR, the sum of imageOffset.x and extent.height does not equal the width of the subresource specified by imageSubresource, extent.height **must** be a multiple of the texel block extent width of the VkFormat of srcImage
- VUID-VkCopyImageToMemoryInfoEXT-srcImage-00208
  For each element of pRegions, if VkCopyCommandTransformInfoQCOM::transform is equal to VK_SURFACE_TRANSFORM_IDENTITY_BIT_KHR, and the sum of imageOffset.y and extent.height does not equal the height of the subresource specified by imageSubresource, extent.height **must** be a multiple of the texel block extent height of the VkFormat of srcImage
- VUID-VkCopyImageToMemoryInfoEXT-imageOffset-10056
  For each element of pRegions, if VkCopyCommandTransformInfoQCOM::transform is equal to VK_SURFACE_TRANSFORM_ROTATE_90_BIT_KHR, the sum of imageOffset.y and extent.width does not equal the height of the subresource specified by imageSubresource, extent.width **must** be a multiple of the texel block extent height of the VkFormat of srcImage
- VUID-VkCopyImageToMemoryInfoEXT-imageOffset-10057
  For each element of pRegions, if VkCopyCommandTransformInfoQCOM::transform is equal to VK_SURFACE_TRANSFORM_ROTATE_180_BIT_KHR, the difference of imageOffset.y and extent.height **must** be a multiple of the texel block extent height of the VkFormat of srcImage
- VUID-VkCopyImageToMemoryInfoEXT-imageOffset-10058
  For each element of pRegions, if VkCopyCommandTransformInfoQCOM::transform is equal

to VK_SURFACE_TRANSFORM_ROTATE_270_BIT_KHR, the difference of imageOffset.y and extent.width **must** be a multiple of the texel block extent height of the VkFormat of srcImage

- VUID-VkCopyImageToMemoryInfoEXT-srcImage-00209
  For each element of pRegions, if the sum of imageOffset.z and extent.depth does not equal the depth of the subresource specified by srcSubresource, extent.depth **must** be a multiple of the texel block extent depth of the VkFormat of srcImage
- VUID-VkCopyImageToMemoryInfoEXT-imageSubresource-09105
  For each element of pRegions, imageSubresource.aspectMask **must** specify aspects present in srcImage
- VUID-VkCopyImageToMemoryInfoEXT-srcImage-07981
  If srcImage has a multi-planar image format, then for each element of pRegions, imageSubresource.aspectMask **must** be a single valid multi-planar aspect mask bit
- VUID-VkCopyImageToMemoryInfoEXT-srcImage-07983
  If srcImage is of type VK_IMAGE_TYPE_3D, for each element of pRegions, imageSubresource.baseArrayLayer **must** be 0 and imageSubresource.layerCount **must** be 1
- VUID-VkCopyImageToMemoryInfoEXT-memoryRowLength-09106
  For each element of pRegions, memoryRowLength **must** be a multiple of the texel block extent width of the VkFormat of srcImage
- VUID-VkCopyImageToMemoryInfoEXT-memoryImageHeight-09107
  For each element of pRegions, memoryImageHeight **must** be a multiple of the texel block extent height of the VkFormat of srcImage
- VUID-VkCopyImageToMemoryInfoEXT-memoryRowLength-09108
  For each element of pRegions, memoryRowLength divided by the texel block extent width and then multiplied by the texel block size of srcImage **must** be less than or equal to $2^{31}$-1
- VUID-VkCopyImageToMemoryInfoEXT-srcImageLayout-09064
  srcImageLayout **must** specify the current layout of the image subresources of srcImage specified in pRegions
- VUID-VkCopyImageToMemoryInfoEXT-srcImageLayout-09065
  srcImageLayout **must** be one of the image layouts returned in VkPhysicalDeviceHostImageCopyPropertiesEXT::pCopySrcLayouts
- VUID-VkCopyImageToMemoryInfoEXT-flags-09394
  If flags includes VK_HOST_IMAGE_COPY_MEMCPY_EXT, for each region in pRegions, memoryRowLength and memoryImageHeight **must** both be 0

## Valid Usage (Implicit)

- VUID-VkCopyImageToMemoryInfoEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_COPY_IMAGE_TO_MEMORY_INFO_EXT
- VUID-VkCopyImageToMemoryInfoEXT-pNext-pNext
  pNext **must** be NULL
- VUID-VkCopyImageToMemoryInfoEXT-flags-parameter

flags **must** be a valid combination of VkHostImageCopyFlagBitsEXT values

- VUID-VkCopyImageToMemoryInfoEXT-srcImage-parameter
  srcImage **must** be a valid VkImage handle
- VUID-VkCopyImageToMemoryInfoEXT-srcImageLayout-parameter
  srcImageLayout **must** be a valid VkImageLayout value
- VUID-VkCopyImageToMemoryInfoEXT-pRegions-parameter
  pRegions **must** be a valid pointer to an array of regionCount valid VkImageToMemoryCopyEXT structures
- VUID-VkCopyImageToMemoryInfoEXT-regionCount-arraylength
  regionCount **must** be greater than 0

Each element of VkCopyImageToMemoryInfoEXT::pRegions is a structure defined as:

```
// Provided by VK_EXT_host_image_copy
typedef struct VkImageToMemoryCopyEXT {
    VkStructureType             sType;
    const void*                 pNext;
    void*                       pHostPointer;
    uint32_t                    memoryRowLength;
    uint32_t                    memoryImageHeight;
    VkImageSubresourceLayers    imageSubresource;
    VkOffset3D                  imageOffset;
    VkExtent3D                  imageExtent;
} VkImageToMemoryCopyEXT;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- pHostPointer is the host memory address which is the destination of the copy.
- memoryRowLength and memoryImageHeight specify in texels a subregion of a larger two- or three-dimensional image in host memory, and control the addressing calculations. If either of these values is zero, that aspect of the host memory is considered to be tightly packed according to the imageExtent.
- imageSubresource is a VkImageSubresourceLayers used to specify the specific image subresources of the image used for the source or destination image data.
- imageOffset selects the initial x, y, z offsets in texels of the sub-region of the source image data.
- imageExtent is the size in texels of the image to copy in width, height and depth.

This structure is functionally similar to VkBufferImageCopy2, except it defines host memory as the target of copy instead of a buffer. In particular, the same data packing rules and restrictions as that structure apply here as well.

## Valid Usage

- VUID-VkImageToMemoryCopyEXT-pHostPointer-09066
  `pHostPointer` **must** point to memory that is large enough to contain all memory locations that are accessed according to Buffer and Image Addressing, for each element of `pRegions`
- VUID-VkImageToMemoryCopyEXT-pRegions-09067
  The union of all source regions, and the union of all destination regions, specified by the elements of `pRegions`, **must** not overlap in memory
- VUID-VkImageToMemoryCopyEXT-memoryRowLength-09101
  `memoryRowLength` **must** be `0`, or greater than or equal to the `width` member of `imageExtent`
- VUID-VkImageToMemoryCopyEXT-memoryImageHeight-09102
  `memoryImageHeight` **must** be `0`, or greater than or equal to the `height` member of `imageExtent`
- VUID-VkImageToMemoryCopyEXT-aspectMask-09103
  The `aspectMask` member of `imageSubresource` **must** only have a single bit set
- VUID-VkImageToMemoryCopyEXT-imageExtent-06659
  `imageExtent.width` **must** not be 0
- VUID-VkImageToMemoryCopyEXT-imageExtent-06660
  `imageExtent.height` **must** not be 0
- VUID-VkImageToMemoryCopyEXT-imageExtent-06661
  `imageExtent.depth` **must** not be 0

## Valid Usage (Implicit)

- VUID-VkImageToMemoryCopyEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_IMAGE_TO_MEMORY_COPY_EXT`
- VUID-VkImageToMemoryCopyEXT-pNext-pNext
  `pNext` **must** be `NULL`
- VUID-VkImageToMemoryCopyEXT-pHostPointer-parameter
  `pHostPointer` **must** be a pointer value
- VUID-VkImageToMemoryCopyEXT-imageSubresource-parameter
  `imageSubresource` **must** be a valid VkImageSubresourceLayers structure

Bits which **can** be set in VkCopyMemoryToImageInfoEXT::`flags`, VkCopyImageToMemoryInfoEXT::`flags`, and VkCopyImageToImageInfoEXT::`flags`, specifying additional copy parameters are:

```
// Provided by VK_EXT_host_image_copy
typedef enum VkHostImageCopyFlagBitsEXT {
    VK_HOST_IMAGE_COPY_MEMCPY_EXT = 0x00000001,
} VkHostImageCopyFlagBitsEXT;
```

- `VK_HOST_IMAGE_COPY_MEMCPY_EXT` specifies that no memory layout swizzling is to be applied during

data copy. For copies between memory and images, this flag indicates that image data in host memory is swizzled in exactly the same way as the image data on the device. Using this flag indicates that the implementations **may** use a simple memory copy to transfer the data between the host memory and the device memory. The format of the swizzled data in host memory is platform dependent and is not defined in this specification.

```
// Provided by VK_EXT_host_image_copy
typedef VkFlags VkHostImageCopyFlagsEXT;
```

`VkHostImageCopyFlagsEXT` is a bitmask type for setting a mask of zero or more VkHostImageCopyFlagBitsEXT.

To copy data from an image object to another image object using the host, call:

```
// Provided by VK_EXT_host_image_copy
VkResult vkCopyImageToImageEXT(
    VkDevice                                    device,
    const VkCopyImageToImageInfoEXT*            pCopyImageToImageInfo);
```

- `device` is the device which owns `pCopyImageToImageInfo->srcImage` and `pCopyImageToImageInfo->dstImage`.
- `pCopyImageToImageInfo` is a pointer to a VkCopyImageToImageInfoEXT structure describing the copy parameters.

This command is functionally similar to vkCmdCopyImage2KHR, except it is executed on the host. The memory of `pCopyImageToImageInfo->srcImage` and `pCopyImageToImageInfo->dstImage` is accessed by the host as if coherent.

**NOTE**

If the device has written to the memory of `pCopyImageToImageInfo->srcImage`, it is not automatically made available to the host. Before this copy command can be called, a memory barrier for this image **must** have been issued on the device with the second synchronization scope including `VK_PIPELINE_STAGE_HOST_BIT` and `VK_ACCESS_HOST_READ_BIT`.

Because queue submissions automatically make host memory visible to the device, there would not be a need for a memory barrier before using the results of this copy operation in `pCopyMemoryToImageInfo->dstImage` on the device.

### Valid Usage

- VUID-vkCopyImageToImageEXT-hostImageCopy-09068
  The `hostImageCopy` feature **must** be enabled

### Valid Usage (Implicit)

- VUID-vkCopyImageToImageEXT-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkCopyImageToImageEXT-pCopyImageToImageInfo-parameter
  `pCopyImageToImageInfo` **must** be a valid pointer to a valid VkCopyImageToImageInfoEXT structure

### Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- `VK_ERROR_INITIALIZATION_FAILED`
- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_MEMORY_MAP_FAILED`

The `VkCopyImageToImageInfoEXT` structure is defined as:

```
// Provided by VK_EXT_host_image_copy
typedef struct VkCopyImageToImageInfoEXT {
    VkStructureType            sType;
    const void*                pNext;
    VkHostImageCopyFlagsEXT    flags;
    VkImage                    srcImage;
    VkImageLayout              srcImageLayout;
    VkImage                    dstImage;
    VkImageLayout              dstImageLayout;
    uint32_t                   regionCount;
    const VkImageCopy2*        pRegions;
} VkCopyImageToImageInfoEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is a bitmask of VkHostImageCopyFlagBitsEXT values describing additional copy parameters.
- `srcImage` is the source image.
- `srcImageLayout` is the layout of the source image subresources for the copy.
- `dstImage` is the destination image.
- `dstImageLayout` is the layout of the destination image subresources for the copy.

- regionCount is the number of regions to copy.
- pRegions is a pointer to an array of VkImageCopy2 structures specifying the regions to copy.

vkCopyImageToImageEXT does not check whether the device memory associated with srcImage or dstImage is currently in use before performing the copy. The application **must** guarantee that any previously submitted command that writes to the copy regions has completed before the host performs the copy.

## Valid Usage

- VUID-VkCopyImageToImageInfoEXT-srcImage-09069
  srcImage and dstImage **must** have been created with identical image creation parameters
- VUID-VkCopyImageToImageInfoEXT-srcImage-09109
  If srcImage is sparse then all memory ranges accessed by the copy command **must** be bound as described in Binding Resource Memory
- VUID-VkCopyImageToImageInfoEXT-srcImage-09111
  If the stencil aspect of srcImage is accessed, and srcImage was not created with separate stencil usage, srcImage **must** have been created with VK_IMAGE_USAGE_HOST_TRANSFER_BIT_EXT set in VkImageCreateInfo::usage
- VUID-VkCopyImageToImageInfoEXT-srcImage-09112
  If the stencil aspect of srcImage is accessed, and srcImage was created with separate stencil usage, srcImage **must** have been created with VK_IMAGE_USAGE_HOST_TRANSFER_BIT_EXT set in VkImageStencilUsageCreateInfo::stencilUsage
- VUID-VkCopyImageToImageInfoEXT-srcImage-09113
  If non-stencil aspects of srcImage are accessed, srcImage **must** have been created with VK_IMAGE_USAGE_HOST_TRANSFER_BIT_EXT set in VkImageCreateInfo::usage
- VUID-VkCopyImageToImageInfoEXT-srcOffset-09114
  If flags contains VK_HOST_IMAGE_COPY_MEMCPY_EXT, the x, y, and z members of the srcOffset member of each element of pRegions **must** be 0
- VUID-VkCopyImageToImageInfoEXT-srcImage-09115
  If flags contains VK_HOST_IMAGE_COPY_MEMCPY_EXT, the extent member of each element of pRegions **must** equal the extents of srcImage identified by srcSubresource
- VUID-VkCopyImageToImageInfoEXT-srcImage-07966
  If srcImage is non-sparse then the image or the specified *disjoint* plane **must** be bound completely and contiguously to a single VkDeviceMemory object
- VUID-VkCopyImageToImageInfoEXT-srcSubresource-07967
  The srcSubresource.mipLevel member of each element of pRegions **must** be less than the mipLevels specified in VkImageCreateInfo when srcImage was created
- VUID-VkCopyImageToImageInfoEXT-srcSubresource-07968
  If srcSubresource.layerCount is not VK_REMAINING_ARRAY_LAYERS, srcSubresource.baseArrayLayer + srcSubresource.layerCount of each element of pRegions **must** be less than or equal to the arrayLayers specified in VkImageCreateInfo when srcImage was created

- VUID-VkCopyImageToImageInfoEXT-srcImage-07969
srcImage **must** not have been created with flags containing VK_IMAGE_CREATE_SUBSAMPLED_BIT_EXT

- VUID-VkCopyImageToImageInfoEXT-srcSubresource-07970
The image region specified by each element of pRegions **must** be contained within the specified srcSubresource of srcImage

- VUID-VkCopyImageToImageInfoEXT-srcSubresource-07971
For each element of pRegions, srcOffset.x and (extent.width + srcOffset.x) **must** both be greater than or equal to 0 and less than or equal to the width of the specified srcSubresource of srcImage

- VUID-VkCopyImageToImageInfoEXT-srcSubresource-07972
For each element of pRegions, srcOffset.y and (extent.height + srcOffset.y) **must** both be greater than or equal to 0 and less than or equal to the height of the specified srcSubresource of srcImage

- VUID-VkCopyImageToImageInfoEXT-srcImage-07979
If srcImage is of type VK_IMAGE_TYPE_1D, then for each element of pRegions, srcOffset.y **must** be 0 and extent.height **must** be 1

- VUID-VkCopyImageToImageInfoEXT-srcOffset-09104
For each element of pRegions, srcOffset.z and (extent.depth + srcOffset.z) **must** both be greater than or equal to 0 and less than or equal to the depth of the specified srcSubresource of srcImage

- VUID-VkCopyImageToImageInfoEXT-srcImage-07980
If srcImage is of type VK_IMAGE_TYPE_1D or VK_IMAGE_TYPE_2D, then for each element of pRegions, srcOffset.z **must** be 0 and extent.depth **must** be 1

- VUID-VkCopyImageToImageInfoEXT-srcImage-07274
For each element of pRegions, if VkCopyCommandTransformInfoQCOM::transform is equal to VK_SURFACE_TRANSFORM_IDENTITY_BIT_KHR or VK_SURFACE_TRANSFORM_ROTATE_270_BIT_KHR, srcOffset.x **must** be a multiple of the texel block extent width of the VkFormat of srcImage

- VUID-VkCopyImageToImageInfoEXT-srcOffset-10051
For each element of pRegions, if VkCopyCommandTransformInfoQCOM::transform is equal to VK_SURFACE_TRANSFORM_ROTATE_180_BIT_KHR or VK_SURFACE_TRANSFORM_ROTATE_90_BIT_KHR, and srcOffset.x does not equal the width of the subresource specified by srcSubresource, srcOffset.x **must** be a multiple of the texel block extent width of the VkFormat of srcImage

- VUID-VkCopyImageToImageInfoEXT-srcImage-07275
For each element of pRegions, if VkCopyCommandTransformInfoQCOM::transform is equal to VK_SURFACE_TRANSFORM_IDENTITY_BIT_KHR or VK_SURFACE_TRANSFORM_ROTATE_90_BIT_KHR, srcOffset.y **must** be a multiple of the texel block extent height of the VkFormat of srcImage

- VUID-VkCopyImageToImageInfoEXT-srcOffset-10052
For each element of pRegions, if VkCopyCommandTransformInfoQCOM::transform is equal to VK_SURFACE_TRANSFORM_ROTATE_270_BIT_KHR or VK_SURFACE_TRANSFORM_ROTATE_180_BIT_KHR, and srcOffset.y does not equal the height of the subresource specified by srcSubresource, srcOffset.y **must** be a multiple of the texel block extent height of the VkFormat of

srcImage

- VUID-VkCopyImageToImageInfoEXT-srcImage-07276
  For each element of pRegions, srcOffset.z **must** be a multiple of the texel block extent depth of the VkFormat of srcImage
- VUID-VkCopyImageToImageInfoEXT-srcImage-00207
  For each element of pRegions, if VkCopyCommandTransformInfoQCOM::transform is equal to VK_SURFACE_TRANSFORM_IDENTITY_BIT_KHR, the sum of srcOffset.x and extent.width does not equal the width of the subresource specified by srcSubresource, extent.width **must** be a multiple of the texel block extent width of the VkFormat of srcImage
- VUID-VkCopyImageToImageInfoEXT-srcOffset-10053
  For each element of pRegions, if VkCopyCommandTransformInfoQCOM::transform is equal to VK_SURFACE_TRANSFORM_ROTATE_90_BIT_KHR, the difference of srcOffset.x and extent.height **must** be a multiple of the texel block extent width of the VkFormat of srcImage
- VUID-VkCopyImageToImageInfoEXT-srcOffset-10054
  For each element of pRegions, if VkCopyCommandTransformInfoQCOM::transform is equal to VK_SURFACE_TRANSFORM_ROTATE_180_BIT_KHR, the difference of srcOffset.x and extent.width **must** be a multiple of the texel block extent width of the VkFormat of srcImage
- VUID-VkCopyImageToImageInfoEXT-srcOffset-10055
  For each element of pRegions, if VkCopyCommandTransformInfoQCOM::transform is equal to VK_SURFACE_TRANSFORM_ROTATE_270_BIT_KHR, the sum of srcOffset.x and extent.height does not equal the width of the subresource specified by srcSubresource, extent.height **must** be a multiple of the texel block extent width of the VkFormat of srcImage
- VUID-VkCopyImageToImageInfoEXT-srcImage-00208
  For each element of pRegions, if VkCopyCommandTransformInfoQCOM::transform is equal to VK_SURFACE_TRANSFORM_IDENTITY_BIT_KHR, and the sum of srcOffset.y and extent.height does not equal the height of the subresource specified by srcSubresource, extent.height **must** be a multiple of the texel block extent height of the VkFormat of srcImage
- VUID-VkCopyImageToImageInfoEXT-srcOffset-10056
  For each element of pRegions, if VkCopyCommandTransformInfoQCOM::transform is equal to VK_SURFACE_TRANSFORM_ROTATE_90_BIT_KHR, the sum of srcOffset.y and extent.width does not equal the height of the subresource specified by srcSubresource, extent.width **must** be a multiple of the texel block extent height of the VkFormat of srcImage
- VUID-VkCopyImageToImageInfoEXT-srcOffset-10057
  For each element of pRegions, if VkCopyCommandTransformInfoQCOM::transform is equal to VK_SURFACE_TRANSFORM_ROTATE_180_BIT_KHR, the difference of srcOffset.y and extent.height **must** be a multiple of the texel block extent height of the VkFormat of srcImage
- VUID-VkCopyImageToImageInfoEXT-srcOffset-10058
  For each element of pRegions, if VkCopyCommandTransformInfoQCOM::transform is equal to VK_SURFACE_TRANSFORM_ROTATE_270_BIT_KHR, the difference of srcOffset.y and extent.width **must** be a multiple of the texel block extent height of the VkFormat of srcImage

- VUID-VkCopyImageToImageInfoEXT-srcImage-00209
  For each element of `pRegions`, if the sum of `srcOffset.z` and `extent.depth` does not equal the depth of the subresource specified by `srcSubresource`, `extent.depth` **must** be a multiple of the texel block extent depth of the VkFormat of `srcImage`
- VUID-VkCopyImageToImageInfoEXT-srcSubresource-09105
  For each element of `pRegions`, `srcSubresource.aspectMask` **must** specify aspects present in `srcImage`
- VUID-VkCopyImageToImageInfoEXT-srcImage-07981
  If `srcImage` has a multi-planar image format, then for each element of `pRegions`, `srcSubresource.aspectMask` **must** be a single valid multi-planar aspect mask bit
- VUID-VkCopyImageToImageInfoEXT-srcImage-07983
  If `srcImage` is of type `VK_IMAGE_TYPE_3D`, for each element of `pRegions`, `srcSubresource.baseArrayLayer` **must** be `0` and `srcSubresource.layerCount` **must** be `1`
- VUID-VkCopyImageToImageInfoEXT-dstImage-09109
  If `dstImage` is sparse then all memory ranges accessed by the copy command **must** be bound as described in Binding Resource Memory
- VUID-VkCopyImageToImageInfoEXT-dstImage-09111
  If the stencil aspect of `dstImage` is accessed, and `dstImage` was not created with separate stencil usage, `dstImage` **must** have been created with `VK_IMAGE_USAGE_HOST_TRANSFER_BIT_EXT` set in VkImageCreateInfo::`usage`
- VUID-VkCopyImageToImageInfoEXT-dstImage-09112
  If the stencil aspect of `dstImage` is accessed, and `dstImage` was created with separate stencil usage, `dstImage` **must** have been created with `VK_IMAGE_USAGE_HOST_TRANSFER_BIT_EXT` set in VkImageStencilUsageCreateInfo::`stencilUsage`
- VUID-VkCopyImageToImageInfoEXT-dstImage-09113
  If non-stencil aspects of `dstImage` are accessed, `dstImage` **must** have been created with `VK_IMAGE_USAGE_HOST_TRANSFER_BIT_EXT` set in VkImageCreateInfo::`usage`
- VUID-VkCopyImageToImageInfoEXT-dstOffset-09114
  If `flags` contains `VK_HOST_IMAGE_COPY_MEMCPY_EXT`, the `x`, `y`, and `z` members of the `dstOffset` member of each element of `pRegions` **must** be `0`
- VUID-VkCopyImageToImageInfoEXT-dstImage-09115
  If `flags` contains `VK_HOST_IMAGE_COPY_MEMCPY_EXT`, the `extent` member of each element of `pRegions` **must** equal the extents of `dstImage` identified by `dstSubresource`
- VUID-VkCopyImageToImageInfoEXT-dstImage-07966
  If `dstImage` is non-sparse then the image or the specified *disjoint* plane **must** be bound completely and contiguously to a single `VkDeviceMemory` object
- VUID-VkCopyImageToImageInfoEXT-dstSubresource-07967
  The `dstSubresource.mipLevel` member of each element of `pRegions` **must** be less than the `mipLevels` specified in VkImageCreateInfo when `dstImage` was created
- VUID-VkCopyImageToImageInfoEXT-dstSubresource-07968
  If `dstSubresource.layerCount` is not `VK_REMAINING_ARRAY_LAYERS`, `dstSubresource.baseArrayLayer` + `dstSubresource.layerCount` of each element of `pRegions`

**must** be less than or equal to the `arrayLayers` specified in VkImageCreateInfo when `dstImage` was created

- VUID-VkCopyImageToImageInfoEXT-dstImage-07969
  `dstImage` **must** not have been created with `flags` containing `VK_IMAGE_CREATE_SUBSAMPLED_BIT_EXT`

- VUID-VkCopyImageToImageInfoEXT-dstSubresource-07970
  The image region specified by each element of `pRegions` **must** be contained within the specified `dstSubresource` of `dstImage`

- VUID-VkCopyImageToImageInfoEXT-dstSubresource-07971
  For each element of `pRegions`, `dstOffset.x` and (`extent.width` + `dstOffset.x`) **must** both be greater than or equal to `0` and less than or equal to the width of the specified `dstSubresource` of `dstImage`

- VUID-VkCopyImageToImageInfoEXT-dstSubresource-07972
  For each element of `pRegions`, `dstOffset.y` and (`extent.height` + `dstOffset.y`) **must** both be greater than or equal to `0` and less than or equal to the height of the specified `dstSubresource` of `dstImage`

- VUID-VkCopyImageToImageInfoEXT-dstImage-07979
  If `dstImage` is of type `VK_IMAGE_TYPE_1D`, then for each element of `pRegions`, `dstOffset.y` **must** be `0` and `extent.height` **must** be `1`

- VUID-VkCopyImageToImageInfoEXT-dstOffset-09104
  For each element of `pRegions`, `dstOffset.z` and (`extent.depth` + `dstOffset.z`) **must** both be greater than or equal to `0` and less than or equal to the depth of the specified `dstSubresource` of `dstImage`

- VUID-VkCopyImageToImageInfoEXT-dstImage-07980
  If `dstImage` is of type `VK_IMAGE_TYPE_1D` or `VK_IMAGE_TYPE_2D`, then for each element of `pRegions`, `dstOffset.z` **must** be `0` and `extent.depth` **must** be `1`

- VUID-VkCopyImageToImageInfoEXT-dstImage-07274
  For each element of `pRegions`, if VkCopyCommandTransformInfoQCOM::`transform` is equal to `VK_SURFACE_TRANSFORM_IDENTITY_BIT_KHR` or `VK_SURFACE_TRANSFORM_ROTATE_270_BIT_KHR`, `dstOffset.x` **must** be a multiple of the texel block extent width of the VkFormat of `dstImage`

- VUID-VkCopyImageToImageInfoEXT-dstOffset-10051
  For each element of `pRegions`, if VkCopyCommandTransformInfoQCOM::`transform` is equal to `VK_SURFACE_TRANSFORM_ROTATE_180_BIT_KHR` or `VK_SURFACE_TRANSFORM_ROTATE_90_BIT_KHR`, and `dstOffset.x` does not equal the width of the subresource specified by `dstSubresource`, `dstOffset.x` **must** be a multiple of the texel block extent width of the VkFormat of `dstImage`

- VUID-VkCopyImageToImageInfoEXT-dstImage-07275
  For each element of `pRegions`, if VkCopyCommandTransformInfoQCOM::`transform` is equal to `VK_SURFACE_TRANSFORM_IDENTITY_BIT_KHR` or `VK_SURFACE_TRANSFORM_ROTATE_90_BIT_KHR`, `dstOffset.y` **must** be a multiple of the texel block extent height of the VkFormat of `dstImage`

- VUID-VkCopyImageToImageInfoEXT-dstOffset-10052
  For each element of `pRegions`, if VkCopyCommandTransformInfoQCOM::`transform` is equal to `VK_SURFACE_TRANSFORM_ROTATE_270_BIT_KHR` or `VK_SURFACE_TRANSFORM_ROTATE_180_BIT_KHR`,

and dstOffset.y does not equal the height of the subresource specified by dstSubresource, dstOffset.y **must** be a multiple of the texel block extent height of the VkFormat of dstImage

- VUID-VkCopyImageToImageInfoEXT-dstImage-07276
  For each element of pRegions, dstOffset.z **must** be a multiple of the texel block extent depth of the VkFormat of dstImage
- VUID-VkCopyImageToImageInfoEXT-dstImage-00207
  For each element of pRegions, if VkCopyCommandTransformInfoQCOM::transform is equal to VK_SURFACE_TRANSFORM_IDENTITY_BIT_KHR, the sum of dstOffset.x and extent.width does not equal the width of the subresource specified by dstSubresource, extent.width **must** be a multiple of the texel block extent width of the VkFormat of dstImage
- VUID-VkCopyImageToImageInfoEXT-dstOffset-10053
  For each element of pRegions, if VkCopyCommandTransformInfoQCOM::transform is equal to VK_SURFACE_TRANSFORM_ROTATE_90_BIT_KHR, the difference of dstOffset.x and extent.height **must** be a multiple of the texel block extent width of the VkFormat of dstImage
- VUID-VkCopyImageToImageInfoEXT-dstOffset-10054
  For each element of pRegions, if VkCopyCommandTransformInfoQCOM::transform is equal to VK_SURFACE_TRANSFORM_ROTATE_180_BIT_KHR, the difference of dstOffset.x and extent.width **must** be a multiple of the texel block extent width of the VkFormat of dstImage
- VUID-VkCopyImageToImageInfoEXT-dstOffset-10055
  For each element of pRegions, if VkCopyCommandTransformInfoQCOM::transform is equal to VK_SURFACE_TRANSFORM_ROTATE_270_BIT_KHR, the sum of dstOffset.x and extent.height does not equal the width of the subresource specified by dstSubresource, extent.height **must** be a multiple of the texel block extent width of the VkFormat of dstImage
- VUID-VkCopyImageToImageInfoEXT-dstImage-00208
  For each element of pRegions, if VkCopyCommandTransformInfoQCOM::transform is equal to VK_SURFACE_TRANSFORM_IDENTITY_BIT_KHR, and the sum of dstOffset.y and extent.height does not equal the height of the subresource specified by dstSubresource, extent.height **must** be a multiple of the texel block extent height of the VkFormat of dstImage
- VUID-VkCopyImageToImageInfoEXT-dstOffset-10056
  For each element of pRegions, if VkCopyCommandTransformInfoQCOM::transform is equal to VK_SURFACE_TRANSFORM_ROTATE_90_BIT_KHR, the sum of dstOffset.y and extent.width does not equal the height of the subresource specified by dstSubresource, extent.width **must** be a multiple of the texel block extent height of the VkFormat of dstImage
- VUID-VkCopyImageToImageInfoEXT-dstOffset-10057
  For each element of pRegions, if VkCopyCommandTransformInfoQCOM::transform is equal to VK_SURFACE_TRANSFORM_ROTATE_180_BIT_KHR, the difference of dstOffset.y and extent.height **must** be a multiple of the texel block extent height of the VkFormat of dstImage
- VUID-VkCopyImageToImageInfoEXT-dstOffset-10058
  For each element of pRegions, if VkCopyCommandTransformInfoQCOM::transform is equal to VK_SURFACE_TRANSFORM_ROTATE_270_BIT_KHR, the difference of dstOffset.y and

extent.width **must** be a multiple of the texel block extent height of the VkFormat of dstImage

- VUID-VkCopyImageToImageInfoEXT-dstImage-00209
  For each element of pRegions, if the sum of dstOffset.z and extent.depth does not equal the depth of the subresource specified by srcSubresource, extent.depth **must** be a multiple of the texel block extent depth of the VkFormat of dstImage
- VUID-VkCopyImageToImageInfoEXT-dstSubresource-09105
  For each element of pRegions, dstSubresource.aspectMask **must** specify aspects present in dstImage
- VUID-VkCopyImageToImageInfoEXT-dstImage-07981
  If dstImage has a multi-planar image format, then for each element of pRegions, dstSubresource.aspectMask **must** be a single valid multi-planar aspect mask bit
- VUID-VkCopyImageToImageInfoEXT-dstImage-07983
  If dstImage is of type VK_IMAGE_TYPE_3D, for each element of pRegions, dstSubresource.baseArrayLayer **must** be 0 and dstSubresource.layerCount **must** be 1
- VUID-VkCopyImageToImageInfoEXT-srcImageLayout-09070
  srcImageLayout **must** specify the current layout of the image subresources of srcImage specified in pRegions
- VUID-VkCopyImageToImageInfoEXT-dstImageLayout-09071
  dstImageLayout **must** specify the current layout of the image subresources of dstImage specified in pRegions
- VUID-VkCopyImageToImageInfoEXT-srcImageLayout-09072
  srcImageLayout **must** be one of the image layouts returned in VkPhysicalDeviceHostImageCopyPropertiesEXT::pCopySrcLayouts
- VUID-VkCopyImageToImageInfoEXT-dstImageLayout-09073
  dstImageLayout **must** be one of the image layouts returned in VkPhysicalDeviceHostImageCopyPropertiesEXT::pCopyDstLayouts

## Valid Usage (Implicit)

- VUID-VkCopyImageToImageInfoEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_COPY_IMAGE_TO_IMAGE_INFO_EXT
- VUID-VkCopyImageToImageInfoEXT-pNext-pNext
  pNext **must** be NULL
- VUID-VkCopyImageToImageInfoEXT-flags-parameter
  flags **must** be a valid combination of VkHostImageCopyFlagBitsEXT values
- VUID-VkCopyImageToImageInfoEXT-srcImage-parameter
  srcImage **must** be a valid VkImage handle
- VUID-VkCopyImageToImageInfoEXT-srcImageLayout-parameter
  srcImageLayout **must** be a valid VkImageLayout value
- VUID-VkCopyImageToImageInfoEXT-dstImage-parameter
  dstImage **must** be a valid VkImage handle

- VUID-VkCopyImageToImageInfoEXT-dstImageLayout-parameter
  `dstImageLayout` **must** be a valid VkImageLayout value
- VUID-VkCopyImageToImageInfoEXT-pRegions-parameter
  `pRegions` **must** be a valid pointer to an array of `regionCount` valid VkImageCopy2 structures
- VUID-VkCopyImageToImageInfoEXT-regionCount-arraylength
  `regionCount` **must** be greater than `0`
- VUID-VkCopyImageToImageInfoEXT-commonparent
  Both of `dstImage`, and `srcImage` **must** have been created, allocated, or retrieved from the same VkDevice

## 20.4. Indirect Copies

An application can use indirect copies when the copy parameters are not known during the command buffer creation time.

To copy data between two memory regions by specifying copy parameters indirectly in a buffer, call:

```
// Provided by VK_NV_copy_memory_indirect
void vkCmdCopyMemoryIndirectNV(
    VkCommandBuffer                             commandBuffer,
    VkDeviceAddress                             copyBufferAddress,
    uint32_t                                    copyCount,
    uint32_t                                    stride);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `copyBufferAddress` is the buffer address specifying the copy parameters. This buffer is laid out in memory as an array of VkCopyMemoryIndirectCommandNV structures.
- `copyCount` is the number of copies to execute, and can be zero.
- `stride` is the stride in bytes between successive sets of copy parameters.

Each region read from `copyBufferAddress` is copied from the source region to the specified destination region. The results are undefined if any of the source and destination regions overlap in memory.

### Valid Usage

- VUID-vkCmdCopyMemoryIndirectNV-None-07653
  The `indirectCopy` feature **must** be enabled
- VUID-vkCmdCopyMemoryIndirectNV-copyBufferAddress-07654
  `copyBufferAddress` **must** be 4 byte aligned
- VUID-vkCmdCopyMemoryIndirectNV-stride-07655

stride **must** be a multiple of 4 and **must** be greater than or equal to sizeof(VkCopyMemoryIndirectCommandNV)

- VUID-vkCmdCopyMemoryIndirectNV-commandBuffer-07656
  The VkCommandPool that commandBuffer was allocated from **must** support at least one of the VkPhysicalDeviceCopyMemoryIndirectPropertiesNV::supportedQueues

## Valid Usage (Implicit)

- VUID-vkCmdCopyMemoryIndirectNV-commandBuffer-parameter
  commandBuffer **must** be a valid VkCommandBuffer handle
- VUID-vkCmdCopyMemoryIndirectNV-commandBuffer-recording
  commandBuffer **must** be in the recording state
- VUID-vkCmdCopyMemoryIndirectNV-commandBuffer-cmdpool
  The VkCommandPool that commandBuffer was allocated from **must** support transfer, graphics, or compute operations
- VUID-vkCmdCopyMemoryIndirectNV-renderpass
  This command **must** only be called outside of a render pass instance
- VUID-vkCmdCopyMemoryIndirectNV-videocoding
  This command **must** only be called outside of a video coding scope

## Host Synchronization

- Host access to commandBuffer **must** be externally synchronized
- Host access to the VkCommandPool that commandBuffer was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Outside | Outside | Transfer<br>Graphics<br>Compute | Action |

The structure describing source and destination memory regions, VkCopyMemoryIndirectCommandNV is defined as:

```
// Provided by VK_NV_copy_memory_indirect
typedef struct VkCopyMemoryIndirectCommandNV {
    VkDeviceAddress    srcAddress;
    VkDeviceAddress    dstAddress;
    VkDeviceSize       size;
} VkCopyMemoryIndirectCommandNV;
```

- srcAddress is the starting address of the source device memory to copy from.
- dstAddress is the starting address of the destination device memory to copy to.
- size is the size of the copy in bytes.

### Valid Usage

- VUID-VkCopyMemoryIndirectCommandNV-srcAddress-07657
  The srcAddress **must** be 4 byte aligned
- VUID-VkCopyMemoryIndirectCommandNV-dstAddress-07658
  The dstAddress **must** be 4 byte aligned
- VUID-VkCopyMemoryIndirectCommandNV-size-07659
  The size **must** be 4 byte aligned

To copy data from a memory region to an image object by specifying copy parameters in a buffer, call:

```
// Provided by VK_NV_copy_memory_indirect
void vkCmdCopyMemoryToImageIndirectNV(
    VkCommandBuffer                             commandBuffer,
    VkDeviceAddress                             copyBufferAddress,
    uint32_t                                    copyCount,
    uint32_t                                    stride,
    VkImage                                     dstImage,
    VkImageLayout                               dstImageLayout,
    const VkImageSubresourceLayers*             pImageSubresources);
```

- commandBuffer is the command buffer into which the command will be recorded.
- copyBufferAddress is the buffer address specifying the copy parameters. This buffer is laid out in memory as an array of VkCopyMemoryToImageIndirectCommandNV structures.
- copyCount is the number of copies to execute, and can be zero.
- stride is the byte stride between successive sets of copy parameters.
- dstImage is the destination image.
- dstImageLayout is the layout of the destination image subresources for the copy.
- pImageSubresources is a pointer to an array of size copyCount of VkImageSubresourceLayers used to specify the specific image subresource of the destination image data for that copy.

Each region in copyBufferAddress is copied from the source memory region to an image region in the destination image. If the destination image is of type VK_IMAGE_TYPE_3D, the starting slice and number of slices to copy are specified in pImageSubresources->baseArrayLayer and pImageSubresources->layerCount respectively. The copy **must** be performed on a queue that supports indirect copy operations, see VkPhysicalDeviceCopyMemoryIndirectPropertiesNV.

### Valid Usage

- VUID-vkCmdCopyMemoryToImageIndirectNV-None-07660
  The indirectCopy feature **must** be enabled
- VUID-vkCmdCopyMemoryToImageIndirectNV-dstImage-07661
  dstImage **must** not be a protected image
- VUID-vkCmdCopyMemoryToImageIndirectNV-aspectMask-07662
  The aspectMask member for every subresource in pImageSubresources **must** only have a single bit set
- VUID-vkCmdCopyMemoryToImageIndirectNV-dstImage-07663
  The image region specified by each element in copyBufferAddress **must** be a region that is contained within dstImage
- VUID-vkCmdCopyMemoryToImageIndirectNV-dstImage-07664
  dstImage **must** have been created with VK_IMAGE_USAGE_TRANSFER_DST_BIT usage flag
- VUID-vkCmdCopyMemoryToImageIndirectNV-dstImage-07665
  If dstImage is non-sparse then it **must** be bound completely and contiguously to a single VkDeviceMemory object
- VUID-vkCmdCopyMemoryToImageIndirectNV-dstImage-07973
  dstImage **must** have a sample count equal to VK_SAMPLE_COUNT_1_BIT
- VUID-vkCmdCopyMemoryToImageIndirectNV-dstImageLayout-07667
  dstImageLayout **must** specify the layout of the image subresources of dstImage at the time this command is executed on a VkDevice
- VUID-vkCmdCopyMemoryToImageIndirectNV-dstImageLayout-07669
  dstImageLayout **must** be VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL, VK_IMAGE_LAYOUT_SHARED_PRESENT_KHR, or VK_IMAGE_LAYOUT_GENERAL
- VUID-vkCmdCopyMemoryToImageIndirectNV-mipLevel-07670
  The specified mipLevel of each region **must** be less than the mipLevels specified in VkImageCreateInfo when dstImage was created
- VUID-vkCmdCopyMemoryToImageIndirectNV-layerCount-08764
  If layerCount is not VK_REMAINING_ARRAY_LAYERS, the specified baseArrayLayer + layerCount of each region **must** be less than or equal to the arrayLayers specified in VkImageCreateInfo when dstImage was created
- VUID-vkCmdCopyMemoryToImageIndirectNV-imageOffset-07672
  The imageOffset and imageExtent members of each region **must** respect the image transfer granularity requirements of commandBuffer's command pool's queue family, as described in VkQueueFamilyProperties

- VUID-vkCmdCopyMemoryToImageIndirectNV-dstImage-07673
  `dstImage` **must** not have been created with `flags` containing `VK_IMAGE_CREATE_SUBSAMPLED_BIT_EXT`
- VUID-vkCmdCopyMemoryToImageIndirectNV-commandBuffer-07674
  If the queue family used to create the VkCommandPool which `commandBuffer` was allocated from does not support `VK_QUEUE_GRAPHICS_BIT`, for each region, the `aspectMask` member of `pImageSubresources` **must** not be `VK_IMAGE_ASPECT_DEPTH_BIT` or `VK_IMAGE_ASPECT_STENCIL_BIT`
- VUID-vkCmdCopyMemoryToImageIndirectNV-imageOffset-07675
  For each region in `copyBufferAddress`, `imageOffset.y` and (`imageExtent.height` + `imageOffset.y`) **must** both be greater than or equal to `0` and less than or equal to the height of the specified subresource
- VUID-vkCmdCopyMemoryToImageIndirectNV-offset-07676
  `offset` **must** be 4 byte aligned
- VUID-vkCmdCopyMemoryToImageIndirectNV-stride-07677
  `stride` **must** be a multiple of `4` and **must** be greater than or equal to sizeof(`VkCopyMemoryToImageIndirectCommandNV`)

## Valid Usage (Implicit)

- VUID-vkCmdCopyMemoryToImageIndirectNV-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdCopyMemoryToImageIndirectNV-dstImage-parameter
  `dstImage` **must** be a valid VkImage handle
- VUID-vkCmdCopyMemoryToImageIndirectNV-dstImageLayout-parameter
  `dstImageLayout` **must** be a valid VkImageLayout value
- VUID-vkCmdCopyMemoryToImageIndirectNV-pImageSubresources-parameter
  `pImageSubresources` **must** be a valid pointer to an array of `copyCount` valid VkImageSubresourceLayers structures
- VUID-vkCmdCopyMemoryToImageIndirectNV-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdCopyMemoryToImageIndirectNV-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support transfer, graphics, or compute operations
- VUID-vkCmdCopyMemoryToImageIndirectNV-renderpass
  This command **must** only be called outside of a render pass instance
- VUID-vkCmdCopyMemoryToImageIndirectNV-videocoding
  This command **must** only be called outside of a video coding scope
- VUID-vkCmdCopyMemoryToImageIndirectNV-copyCount-arraylength
  `copyCount` **must** be greater than `0`
- VUID-vkCmdCopyMemoryToImageIndirectNV-commonparent
  Both of `commandBuffer`, and `dstImage` **must** have been created, allocated, or retrieved from the same VkDevice

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Outside | Outside | Transfer<br>Graphics<br>Compute | Action |

The `VkCopyMemoryToImageIndirectCommandNV` is defined as:

```
// Provided by VK_NV_copy_memory_indirect
typedef struct VkCopyMemoryToImageIndirectCommandNV {
    VkDeviceAddress             srcAddress;
    uint32_t                    bufferRowLength;
    uint32_t                    bufferImageHeight;
    VkImageSubresourceLayers    imageSubresource;
    VkOffset3D                  imageOffset;
    VkExtent3D                  imageExtent;
} VkCopyMemoryToImageIndirectCommandNV;
```

- `srcAddress` is the starting address of the source device memory to copy from.
- `bufferRowLength` and `bufferImageHeight` specify in texels a subregion of a larger two- or three-dimensional image in buffer memory, and control the addressing calculations. If either of these values is zero, that aspect of the buffer memory is considered to be tightly packed according to the `imageExtent`.
- `imageSubresource` is a VkImageSubresourceLayers used to specify the specific image subresources of the image used for the destination image data, which **must** match the values specified in `pImageSubresources` parameter of vkCmdCopyMemoryToImageIndirectNV during command recording.
- `imageOffset` selects the initial `x`, `y`, `z` offsets in texels of the sub-region of the destination image data.
- `imageExtent` is the size in texels of the destination image in `width`, `height` and `depth`.

## Valid Usage

- VUID-VkCopyMemoryToImageIndirectCommandNV-srcAddress-07678

The srcAddress **must** be 4 byte aligned

- VUID-VkCopyMemoryToImageIndirectCommandNV-bufferRowLength-07679
  bufferRowLength **must** be 0, or greater than or equal to the width member of imageExtent
- VUID-VkCopyMemoryToImageIndirectCommandNV-bufferImageHeight-07680
  bufferImageHeight **must** be 0, or greater than or equal to the height member of imageExtent
- VUID-VkCopyMemoryToImageIndirectCommandNV-imageOffset-07681
  imageOffset **must** specify a valid offset in the destination image
- VUID-VkCopyMemoryToImageIndirectCommandNV-imageExtent-07682
  imageExtent **must** specify a valid region in the destination image and can be 0

**Valid Usage (Implicit)**

- VUID-VkCopyMemoryToImageIndirectCommandNV-imageSubresource-parameter
  imageSubresource **must** be a valid VkImageSubresourceLayers structure

## 20.5. Image Copies With Scaling

To copy regions of a source image into a destination image, potentially performing format conversion, arbitrary scaling, and filtering, call:

```
// Provided by VK_VERSION_1_0
void vkCmdBlitImage(
    VkCommandBuffer                                 commandBuffer,
    VkImage                                         srcImage,
    VkImageLayout                                   srcImageLayout,
    VkImage                                         dstImage,
    VkImageLayout                                   dstImageLayout,
    uint32_t                                        regionCount,
    const VkImageBlit*                              pRegions,
    VkFilter                                        filter);
```

- commandBuffer is the command buffer into which the command will be recorded.
- srcImage is the source image.
- srcImageLayout is the layout of the source image subresources for the blit.
- dstImage is the destination image.
- dstImageLayout is the layout of the destination image subresources for the blit.
- regionCount is the number of regions to blit.
- pRegions is a pointer to an array of VkImageBlit structures specifying the regions to blit.
- filter is a VkFilter specifying the filter to apply if the blits require scaling.

vkCmdBlitImage **must** not be used for multisampled source or destination images. Use

vkCmdResolveImage for this purpose.

As the sizes of the source and destination extents **can** differ in any dimension, texels in the source extent are scaled and filtered to the destination extent. Scaling occurs via the following operations:

- For each destination texel, the integer coordinate of that texel is converted to an unnormalized texture coordinate, using the effective inverse of the equations described in unnormalized to integer conversion:

  $$u_{base} = i + ½$$

  $$v_{base} = j + ½$$

  $$w_{base} = k + ½$$

- These base coordinates are then offset by the first destination offset:

  $$u_{offset} = u_{base} - x_{dst0}$$

  $$v_{offset} = v_{base} - y_{dst0}$$

  $$w_{offset} = w_{base} - z_{dst0}$$

  $$a_{offset} = a - \texttt{baseArrayCount}_{dst}$$

- The scale is determined from the source and destination regions, and applied to the offset coordinates:

  $$scale_u = (x_{src1} - x_{src0}) / (x_{dst1} - x_{dst0})$$

  $$scale_v = (y_{src1} - y_{src0}) / (y_{dst1} - y_{dst0})$$

  $$scale_w = (z_{src1} - z_{src0}) / (z_{dst1} - z_{dst0})$$

  $$u_{scaled} = u_{offset} \times scale_u$$

  $$v_{scaled} = v_{offset} \times scale_v$$

$$w_{scaled} = w_{offset} \times scale_w$$

- Finally the source offset is added to the scaled coordinates, to determine the final unnormalized coordinates used to sample from `srcImage`:

$$u = u_{scaled} + x_{src0}$$

$$v = v_{scaled} + y_{src0}$$

$$w = w_{scaled} + z_{src0}$$

$$q = \texttt{mipLevel}$$

$$a = a_{offset} + \texttt{baseArrayCount}_{src}$$

These coordinates are used to sample from the source image, as described in Image Operations chapter, with the filter mode equal to that of `filter`, a mipmap mode of `VK_SAMPLER_MIPMAP_MODE_NEAREST` and an address mode of `VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_EDGE`. Implementations **must** clamp at the edge of the source image, and **may** additionally clamp to the edge of the source region.

| | |
|---|---|
| **NOTE** | Due to allowable rounding errors in the generation of the source texture coordinates, it is not always possible to guarantee exactly which source texels will be sampled for a given blit. As rounding errors are implementation-dependent, the exact results of a blitting operation are also implementation-dependent. |

Blits are done layer by layer starting with the `baseArrayLayer` member of `srcSubresource` for the source and `dstSubresource` for the destination. `layerCount` layers are blitted to the destination image.

When blitting 3D textures, slices in the destination region bounded by `dstOffsets`[0].z and `dstOffsets`[1].z are sampled from slices in the source region bounded by `srcOffsets`[0].z and `srcOffsets`[1].z. If the `filter` parameter is `VK_FILTER_LINEAR` then the value sampled from the source image is taken by doing linear filtering using the interpolated **z** coordinate represented by **w** in the previous equations. If the `filter` parameter is `VK_FILTER_NEAREST` then the value sampled from the source image is taken from the single nearest slice, with an implementation-dependent arithmetic rounding mode.

The following filtering and conversion rules apply:

- Integer formats **can** only be converted to other integer formats with the same signedness.
- No format conversion is supported between depth/stencil images. The formats **must** match.
- Format conversions on unorm, snorm, scaled and packed float formats of the copied aspect of

the image are performed by first converting the pixels to float values.

- For sRGB source formats, nonlinear RGB values are converted to linear representation prior to filtering.
- After filtering, the float values are first clamped and then cast to the destination image format. In case of sRGB destination format, linear RGB values are converted to nonlinear representation before writing the pixel to the image.

Signed and unsigned integers are converted by first clamping to the representable range of the destination format, then casting the value.

## Valid Usage

- VUID-vkCmdBlitImage-commandBuffer-01834
  If `commandBuffer` is an unprotected command buffer and `protectedNoFault` is not supported, `srcImage` **must** not be a protected image
- VUID-vkCmdBlitImage-commandBuffer-01835
  If `commandBuffer` is an unprotected command buffer and `protectedNoFault` is not supported, `dstImage` **must** not be a protected image
- VUID-vkCmdBlitImage-commandBuffer-01836
  If `commandBuffer` is a protected command buffer and `protectedNoFault` is not supported, `dstImage` **must** not be an unprotected image
- VUID-vkCmdBlitImage-pRegions-00215
  The source region specified by each element of `pRegions` **must** be a region that is contained within `srcImage`
- VUID-vkCmdBlitImage-pRegions-00216
  The destination region specified by each element of `pRegions` **must** be a region that is contained within `dstImage`
- VUID-vkCmdBlitImage-pRegions-00217
  The union of all destination regions, specified by the elements of `pRegions`, **must** not overlap in memory with any texel that **may** be sampled during the blit operation
- VUID-vkCmdBlitImage-srcImage-01999
  The format features of `srcImage` **must** contain `VK_FORMAT_FEATURE_BLIT_SRC_BIT`
- VUID-vkCmdBlitImage-srcImage-06421
  `srcImage` **must** not use a format that requires a sampler $Y'C_BC_R$ conversion
- VUID-vkCmdBlitImage-srcImage-00219
  `srcImage` **must** have been created with `VK_IMAGE_USAGE_TRANSFER_SRC_BIT` usage flag
- VUID-vkCmdBlitImage-srcImage-00220
  If `srcImage` is non-sparse then it **must** be bound completely and contiguously to a single `VkDeviceMemory` object
- VUID-vkCmdBlitImage-srcImageLayout-00221
  `srcImageLayout` **must** specify the layout of the image subresources of `srcImage` specified in `pRegions` at the time this command is executed on a `VkDevice`

- VUID-vkCmdBlitImage-srcImageLayout-01398
srcImageLayout **must** be VK_IMAGE_LAYOUT_SHARED_PRESENT_KHR, VK_IMAGE_LAYOUT_TRANSFER_SRC_OPTIMAL or VK_IMAGE_LAYOUT_GENERAL
- VUID-vkCmdBlitImage-srcImage-09459
If srcImage and dstImage are the same, and an elements of pRegions contains the srcSubresource and dstSubresource with matching mipLevel and overlapping array layers, then the srcImageLayout and dstImageLayout **must** be VK_IMAGE_LAYOUT_GENERAL or VK_IMAGE_LAYOUT_SHARED_PRESENT_KHR
- VUID-vkCmdBlitImage-dstImage-02000
The format features of dstImage **must** contain VK_FORMAT_FEATURE_BLIT_DST_BIT
- VUID-vkCmdBlitImage-dstImage-06422
dstImage **must** not use a format that requires a sampler $Y'C_BC_R$ conversion
- VUID-vkCmdBlitImage-dstImage-00224
dstImage **must** have been created with VK_IMAGE_USAGE_TRANSFER_DST_BIT usage flag
- VUID-vkCmdBlitImage-dstImage-00225
If dstImage is non-sparse then it **must** be bound completely and contiguously to a single VkDeviceMemory object
- VUID-vkCmdBlitImage-dstImageLayout-00226
dstImageLayout **must** specify the layout of the image subresources of dstImage specified in pRegions at the time this command is executed on a VkDevice
- VUID-vkCmdBlitImage-dstImageLayout-01399
dstImageLayout **must** be VK_IMAGE_LAYOUT_SHARED_PRESENT_KHR, VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL or VK_IMAGE_LAYOUT_GENERAL
- VUID-vkCmdBlitImage-srcImage-00229
If either of srcImage or dstImage was created with a signed integer VkFormat, the other **must** also have been created with a signed integer VkFormat
- VUID-vkCmdBlitImage-srcImage-00230
If either of srcImage or dstImage was created with an unsigned integer VkFormat, the other **must** also have been created with an unsigned integer VkFormat
- VUID-vkCmdBlitImage-srcImage-00231
If either of srcImage or dstImage was created with a depth/stencil format, the other **must** have exactly the same format
- VUID-vkCmdBlitImage-srcImage-00232
If srcImage was created with a depth/stencil format, filter **must** be VK_FILTER_NEAREST
- VUID-vkCmdBlitImage-srcImage-00233
srcImage **must** have been created with a samples value of VK_SAMPLE_COUNT_1_BIT
- VUID-vkCmdBlitImage-dstImage-00234
dstImage **must** have been created with a samples value of VK_SAMPLE_COUNT_1_BIT
- VUID-vkCmdBlitImage-filter-02001
If filter is VK_FILTER_LINEAR, then the format features of srcImage **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT
- VUID-vkCmdBlitImage-filter-02002

If filter is VK_FILTER_CUBIC_EXT, then the format features of srcImage **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_CUBIC_BIT_EXT

- VUID-vkCmdBlitImage-filter-00237
  If filter is VK_FILTER_CUBIC_EXT, srcImage **must** be of type VK_IMAGE_TYPE_2D
- VUID-vkCmdBlitImage-srcSubresource-01705
  The srcSubresource.mipLevel member of each element of pRegions **must** be less than the mipLevels specified in VkImageCreateInfo when srcImage was created
- VUID-vkCmdBlitImage-dstSubresource-01706
  The dstSubresource.mipLevel member of each element of pRegions **must** be less than the mipLevels specified in VkImageCreateInfo when dstImage was created
- VUID-vkCmdBlitImage-srcSubresource-01707
  If srcSubresource.layerCount is not VK_REMAINING_ARRAY_LAYERS, srcSubresource.baseArrayLayer + srcSubresource.layerCount of each element of pRegions **must** be less than or equal to the arrayLayers specified in VkImageCreateInfo when srcImage was created
- VUID-vkCmdBlitImage-dstSubresource-01708
  If srcSubresource.layerCount is not VK_REMAINING_ARRAY_LAYERS, dstSubresource.baseArrayLayer + dstSubresource.layerCount of each element of pRegions **must** be less than or equal to the arrayLayers specified in VkImageCreateInfo when dstImage was created
- VUID-vkCmdBlitImage-dstImage-02545
  dstImage and srcImage **must** not have been created with flags containing VK_IMAGE_CREATE_SUBSAMPLED_BIT_EXT
- VUID-vkCmdBlitImage-srcImage-00240
  If either srcImage or dstImage is of type VK_IMAGE_TYPE_3D, then for each element of pRegions, srcSubresource.baseArrayLayer and dstSubresource.baseArrayLayer **must** each be 0, and srcSubresource.layerCount and dstSubresource.layerCount **must** each be 1
- VUID-vkCmdBlitImage-aspectMask-00241
  For each element of pRegions, srcSubresource.aspectMask **must** specify aspects present in srcImage
- VUID-vkCmdBlitImage-aspectMask-00242
  For each element of pRegions, dstSubresource.aspectMask **must** specify aspects present in dstImage
- VUID-vkCmdBlitImage-srcOffset-00243
  For each element of pRegions, srcOffsets[0].x and srcOffsets[1].x **must** both be greater than or equal to 0 and less than or equal to the width of the specified srcSubresource of srcImage
- VUID-vkCmdBlitImage-srcOffset-00244
  For each element of pRegions, srcOffsets[0].y and srcOffsets[1].y **must** both be greater than or equal to 0 and less than or equal to the height of the specified srcSubresource of srcImage
- VUID-vkCmdBlitImage-srcImage-00245
  If srcImage is of type VK_IMAGE_TYPE_1D, then for each element of pRegions, srcOffsets[0].y

**must** be `0` and `srcOffsets`[1].y **must** be `1`

- VUID-vkCmdBlitImage-srcOffset-00246
  For each element of `pRegions`, `srcOffsets`[0].z and `srcOffsets`[1].z **must** both be greater than or equal to `0` and less than or equal to the depth of the specified `srcSubresource` of `srcImage`
- VUID-vkCmdBlitImage-srcImage-00247
  If `srcImage` is of type `VK_IMAGE_TYPE_1D` or `VK_IMAGE_TYPE_2D`, then for each element of `pRegions`, `srcOffsets`[0].z **must** be `0` and `srcOffsets`[1].z **must** be `1`
- VUID-vkCmdBlitImage-dstOffset-00248
  For each element of `pRegions`, `dstOffsets`[0].x and `dstOffsets`[1].x **must** both be greater than or equal to `0` and less than or equal to the width of the specified `dstSubresource` of `dstImage`
- VUID-vkCmdBlitImage-dstOffset-00249
  For each element of `pRegions`, `dstOffsets`[0].y and `dstOffsets`[1].y **must** both be greater than or equal to `0` and less than or equal to the height of the specified `dstSubresource` of `dstImage`
- VUID-vkCmdBlitImage-dstImage-00250
  If `dstImage` is of type `VK_IMAGE_TYPE_1D`, then for each element of `pRegions`, `dstOffsets`[0].y **must** be `0` and `dstOffsets`[1].y **must** be `1`
- VUID-vkCmdBlitImage-dstOffset-00251
  For each element of `pRegions`, `dstOffsets`[0].z and `dstOffsets`[1].z **must** both be greater than or equal to `0` and less than or equal to the depth of the specified `dstSubresource` of `dstImage`
- VUID-vkCmdBlitImage-dstImage-00252
  If `dstImage` is of type `VK_IMAGE_TYPE_1D` or `VK_IMAGE_TYPE_2D`, then for each element of `pRegions`, `dstOffsets`[0].z **must** be `0` and `dstOffsets`[1].z **must** be `1`

## Valid Usage (Implicit)

- VUID-vkCmdBlitImage-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdBlitImage-srcImage-parameter
  `srcImage` **must** be a valid VkImage handle
- VUID-vkCmdBlitImage-srcImageLayout-parameter
  `srcImageLayout` **must** be a valid VkImageLayout value
- VUID-vkCmdBlitImage-dstImage-parameter
  `dstImage` **must** be a valid VkImage handle
- VUID-vkCmdBlitImage-dstImageLayout-parameter
  `dstImageLayout` **must** be a valid VkImageLayout value
- VUID-vkCmdBlitImage-pRegions-parameter
  `pRegions` **must** be a valid pointer to an array of `regionCount` valid VkImageBlit structures
- VUID-vkCmdBlitImage-filter-parameter

filter **must** be a valid VkFilter value

- VUID-vkCmdBlitImage-commandBuffer-recording
  commandBuffer **must** be in the recording state
- VUID-vkCmdBlitImage-commandBuffer-cmdpool
  The VkCommandPool that commandBuffer was allocated from **must** support graphics operations
- VUID-vkCmdBlitImage-renderpass
  This command **must** only be called outside of a render pass instance
- VUID-vkCmdBlitImage-videocoding
  This command **must** only be called outside of a video coding scope
- VUID-vkCmdBlitImage-regionCount-arraylength
  regionCount **must** be greater than 0
- VUID-vkCmdBlitImage-commonparent
  Each of commandBuffer, dstImage, and srcImage **must** have been created, allocated, or retrieved from the same VkDevice

## Host Synchronization

- Host access to commandBuffer **must** be externally synchronized
- Host access to the VkCommandPool that commandBuffer was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Outside | Outside | Graphics | Action |

The VkImageBlit structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkImageBlit {
    VkImageSubresourceLayers    srcSubresource;
    VkOffset3D                  srcOffsets[2];
    VkImageSubresourceLayers    dstSubresource;
    VkOffset3D                  dstOffsets[2];
} VkImageBlit;
```

- srcSubresource is the subresource to blit from.
- srcOffsets is a pointer to an array of two VkOffset3D structures specifying the bounds of the

source region within srcSubresource.

- dstSubresource is the subresource to blit into.
- dstOffsets is a pointer to an array of two VkOffset3D structures specifying the bounds of the destination region within dstSubresource.

For each element of the pRegions array, a blit operation is performed for the specified source and destination regions.

### Valid Usage

- VUID-VkImageBlit-aspectMask-00238
  The aspectMask member of srcSubresource and dstSubresource **must** match
- VUID-VkImageBlit-layerCount-08800
  If neither of the layerCount members of srcSubresource or dstSubresource are VK_REMAINING_ARRAY_LAYERS, the layerCount members of srcSubresource or dstSubresource **must** match
- VUID-VkImageBlit-layerCount-08801
  If one of the layerCount members of srcSubresource or dstSubresource is VK_REMAINING_ARRAY_LAYERS, the other member **must** be either VK_REMAINING_ARRAY_LAYERS or equal to the arrayLayers member of the VkImageCreateInfo used to create the image minus baseArrayLayer

### Valid Usage (Implicit)

- VUID-VkImageBlit-srcSubresource-parameter
  srcSubresource **must** be a valid VkImageSubresourceLayers structure
- VUID-VkImageBlit-dstSubresource-parameter
  dstSubresource **must** be a valid VkImageSubresourceLayers structure

A more extensible version of the blit image command is defined below.

To copy regions of a source image into a destination image, potentially performing format conversion, arbitrary scaling, and filtering, call:

```
// Provided by VK_KHR_copy_commands2
void vkCmdBlitImage2KHR(
    VkCommandBuffer                             commandBuffer,
    const VkBlitImageInfo2*                     pBlitImageInfo);
```

- commandBuffer is the command buffer into which the command will be recorded.
- pBlitImageInfo is a pointer to a VkBlitImageInfo2 structure describing the blit parameters.

This command is functionally identical to vkCmdBlitImage, but includes extensible sub-structures

that include sType and pNext parameters, allowing them to be more easily extended.

## Valid Usage

- VUID-vkCmdBlitImage2-commandBuffer-01834
  If commandBuffer is an unprotected command buffer and protectedNoFault is not supported, srcImage **must** not be a protected image
- VUID-vkCmdBlitImage2-commandBuffer-01835
  If commandBuffer is an unprotected command buffer and protectedNoFault is not supported, dstImage **must** not be a protected image
- VUID-vkCmdBlitImage2-commandBuffer-01836
  If commandBuffer is a protected command buffer and protectedNoFault is not supported, dstImage **must** not be an unprotected image

## Valid Usage (Implicit)

- VUID-vkCmdBlitImage2-commandBuffer-parameter
  commandBuffer **must** be a valid VkCommandBuffer handle
- VUID-vkCmdBlitImage2-pBlitImageInfo-parameter
  pBlitImageInfo **must** be a valid pointer to a valid VkBlitImageInfo2 structure
- VUID-vkCmdBlitImage2-commandBuffer-recording
  commandBuffer **must** be in the recording state
- VUID-vkCmdBlitImage2-commandBuffer-cmdpool
  The VkCommandPool that commandBuffer was allocated from **must** support graphics operations
- VUID-vkCmdBlitImage2-renderpass
  This command **must** only be called outside of a render pass instance
- VUID-vkCmdBlitImage2-videocoding
  This command **must** only be called outside of a video coding scope

## Host Synchronization

- Host access to commandBuffer **must** be externally synchronized
- Host access to the VkCommandPool that commandBuffer was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Outside | Outside | Graphics | Action |

The `VkBlitImageInfo2` structure is defined as:

```
typedef struct VkBlitImageInfo2 {
    VkStructureType        sType;
    const void*            pNext;
    VkImage                srcImage;
    VkImageLayout          srcImageLayout;
    VkImage                dstImage;
    VkImageLayout          dstImageLayout;
    uint32_t               regionCount;
    const VkImageBlit2*    pRegions;
    VkFilter               filter;
} VkBlitImageInfo2;
```

or the equivalent

```
// Provided by VK_KHR_copy_commands2
typedef VkBlitImageInfo2 VkBlitImageInfo2KHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `srcImage` is the source image.
- `srcImageLayout` is the layout of the source image subresources for the blit.
- `dstImage` is the destination image.
- `dstImageLayout` is the layout of the destination image subresources for the blit.
- `regionCount` is the number of regions to blit.
- `pRegions` is a pointer to an array of VkImageBlit2 structures specifying the regions to blit.
- `filter` is a VkFilter specifying the filter to apply if the blits require scaling.

## Valid Usage

- VUID-VkBlitImageInfo2-pRegions-00215
  The source region specified by each element of `pRegions` **must** be a region that is contained within `srcImage`

- VUID-VkBlitImageInfo2-pRegions-00216
  The destination region specified by each element of `pRegions` **must** be a region that is contained within `dstImage`
- VUID-VkBlitImageInfo2-pRegions-00217
  The union of all destination regions, specified by the elements of `pRegions`, **must** not overlap in memory with any texel that **may** be sampled during the blit operation
- VUID-VkBlitImageInfo2-srcImage-01999
  The format features of `srcImage` **must** contain `VK_FORMAT_FEATURE_BLIT_SRC_BIT`
- VUID-VkBlitImageInfo2-srcImage-06421
  `srcImage` **must** not use a format that requires a sampler $Y'C_BC_R$ conversion
- VUID-VkBlitImageInfo2-srcImage-00219
  `srcImage` **must** have been created with `VK_IMAGE_USAGE_TRANSFER_SRC_BIT` usage flag
- VUID-VkBlitImageInfo2-srcImage-00220
  If `srcImage` is non-sparse then it **must** be bound completely and contiguously to a single `VkDeviceMemory` object
- VUID-VkBlitImageInfo2-srcImageLayout-00221
  `srcImageLayout` **must** specify the layout of the image subresources of `srcImage` specified in `pRegions` at the time this command is executed on a `VkDevice`
- VUID-VkBlitImageInfo2-srcImageLayout-01398
  `srcImageLayout` **must** be `VK_IMAGE_LAYOUT_SHARED_PRESENT_KHR`, `VK_IMAGE_LAYOUT_TRANSFER_SRC_OPTIMAL` or `VK_IMAGE_LAYOUT_GENERAL`
- VUID-VkBlitImageInfo2-srcImage-09459
  If `srcImage` and `dstImage` are the same, and an elements of `pRegions` contains the `srcSubresource` and `dstSubresource` with matching `mipLevel` and overlapping array layers, then the `srcImageLayout` and `dstImageLayout` **must** be `VK_IMAGE_LAYOUT_GENERAL` or `VK_IMAGE_LAYOUT_SHARED_PRESENT_KHR`
- VUID-VkBlitImageInfo2-dstImage-02000
  The format features of `dstImage` **must** contain `VK_FORMAT_FEATURE_BLIT_DST_BIT`
- VUID-VkBlitImageInfo2-dstImage-06422
  `dstImage` **must** not use a format that requires a sampler $Y'C_BC_R$ conversion
- VUID-VkBlitImageInfo2-dstImage-00224
  `dstImage` **must** have been created with `VK_IMAGE_USAGE_TRANSFER_DST_BIT` usage flag
- VUID-VkBlitImageInfo2-dstImage-00225
  If `dstImage` is non-sparse then it **must** be bound completely and contiguously to a single `VkDeviceMemory` object
- VUID-VkBlitImageInfo2-dstImageLayout-00226
  `dstImageLayout` **must** specify the layout of the image subresources of `dstImage` specified in `pRegions` at the time this command is executed on a `VkDevice`
- VUID-VkBlitImageInfo2-dstImageLayout-01399
  `dstImageLayout` **must** be `VK_IMAGE_LAYOUT_SHARED_PRESENT_KHR`, `VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL` or `VK_IMAGE_LAYOUT_GENERAL`
- VUID-VkBlitImageInfo2-srcImage-00229

If either of srcImage or dstImage was created with a signed integer VkFormat, the other **must** also have been created with a signed integer VkFormat

- VUID-VkBlitImageInfo2-srcImage-00230
  If either of srcImage or dstImage was created with an unsigned integer VkFormat, the other **must** also have been created with an unsigned integer VkFormat
- VUID-VkBlitImageInfo2-srcImage-00231
  If either of srcImage or dstImage was created with a depth/stencil format, the other **must** have exactly the same format
- VUID-VkBlitImageInfo2-srcImage-00232
  If srcImage was created with a depth/stencil format, filter **must** be VK_FILTER_NEAREST
- VUID-VkBlitImageInfo2-srcImage-00233
  srcImage **must** have been created with a samples value of VK_SAMPLE_COUNT_1_BIT
- VUID-VkBlitImageInfo2-dstImage-00234
  dstImage **must** have been created with a samples value of VK_SAMPLE_COUNT_1_BIT
- VUID-VkBlitImageInfo2-filter-02001
  If filter is VK_FILTER_LINEAR, then the format features of srcImage **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT
- VUID-VkBlitImageInfo2-filter-02002
  If filter is VK_FILTER_CUBIC_EXT, then the format features of srcImage **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_CUBIC_BIT_EXT
- VUID-VkBlitImageInfo2-filter-00237
  If filter is VK_FILTER_CUBIC_EXT, srcImage **must** be of type VK_IMAGE_TYPE_2D
- VUID-VkBlitImageInfo2-srcSubresource-01705
  The srcSubresource.mipLevel member of each element of pRegions **must** be less than the mipLevels specified in VkImageCreateInfo when srcImage was created
- VUID-VkBlitImageInfo2-dstSubresource-01706
  The dstSubresource.mipLevel member of each element of pRegions **must** be less than the mipLevels specified in VkImageCreateInfo when dstImage was created
- VUID-VkBlitImageInfo2-srcSubresource-01707
  If srcSubresource.layerCount is not VK_REMAINING_ARRAY_LAYERS, srcSubresource.baseArrayLayer + srcSubresource.layerCount of each element of pRegions **must** be less than or equal to the arrayLayers specified in VkImageCreateInfo when srcImage was created
- VUID-VkBlitImageInfo2-dstSubresource-01708
  If srcSubresource.layerCount is not VK_REMAINING_ARRAY_LAYERS, dstSubresource.baseArrayLayer + dstSubresource.layerCount of each element of pRegions **must** be less than or equal to the arrayLayers specified in VkImageCreateInfo when dstImage was created
- VUID-VkBlitImageInfo2-dstImage-02545
  dstImage and srcImage **must** not have been created with flags containing VK_IMAGE_CREATE_SUBSAMPLED_BIT_EXT
- VUID-VkBlitImageInfo2-srcImage-00240

If either srcImage or dstImage is of type VK_IMAGE_TYPE_3D, then for each element of pRegions, srcSubresource.baseArrayLayer and dstSubresource.baseArrayLayer **must** each be 0, and srcSubresource.layerCount and dstSubresource.layerCount **must** each be 1

- VUID-VkBlitImageInfo2-aspectMask-00241
  For each element of pRegions, srcSubresource.aspectMask **must** specify aspects present in srcImage
- VUID-VkBlitImageInfo2-aspectMask-00242
  For each element of pRegions, dstSubresource.aspectMask **must** specify aspects present in dstImage
- VUID-VkBlitImageInfo2-srcOffset-00243
  For each element of pRegions, srcOffsets[0].x and srcOffsets[1].x **must** both be greater than or equal to 0 and less than or equal to the width of the specified srcSubresource of srcImage
- VUID-VkBlitImageInfo2-srcOffset-00244
  For each element of pRegions, srcOffsets[0].y and srcOffsets[1].y **must** both be greater than or equal to 0 and less than or equal to the height of the specified srcSubresource of srcImage
- VUID-VkBlitImageInfo2-srcImage-00245
  If srcImage is of type VK_IMAGE_TYPE_1D, then for each element of pRegions, srcOffsets[0].y **must** be 0 and srcOffsets[1].y **must** be 1
- VUID-VkBlitImageInfo2-srcOffset-00246
  For each element of pRegions, srcOffsets[0].z and srcOffsets[1].z **must** both be greater than or equal to 0 and less than or equal to the depth of the specified srcSubresource of srcImage
- VUID-VkBlitImageInfo2-srcImage-00247
  If srcImage is of type VK_IMAGE_TYPE_1D or VK_IMAGE_TYPE_2D, then for each element of pRegions, srcOffsets[0].z **must** be 0 and srcOffsets[1].z **must** be 1
- VUID-VkBlitImageInfo2-dstOffset-00248
  For each element of pRegions, dstOffsets[0].x and dstOffsets[1].x **must** both be greater than or equal to 0 and less than or equal to the width of the specified dstSubresource of dstImage
- VUID-VkBlitImageInfo2-dstOffset-00249
  For each element of pRegions, dstOffsets[0].y and dstOffsets[1].y **must** both be greater than or equal to 0 and less than or equal to the height of the specified dstSubresource of dstImage
- VUID-VkBlitImageInfo2-dstImage-00250
  If dstImage is of type VK_IMAGE_TYPE_1D, then for each element of pRegions, dstOffsets[0].y **must** be 0 and dstOffsets[1].y **must** be 1
- VUID-VkBlitImageInfo2-dstOffset-00251
  For each element of pRegions, dstOffsets[0].z and dstOffsets[1].z **must** both be greater than or equal to 0 and less than or equal to the depth of the specified dstSubresource of dstImage
- VUID-VkBlitImageInfo2-dstImage-00252

If dstImage is of type VK_IMAGE_TYPE_1D or VK_IMAGE_TYPE_2D, then for each element of pRegions, dstOffsets[0].z **must** be 0 and dstOffsets[1].z **must** be 1

- VUID-VkBlitImageInfo2-pRegions-04561
  If any element of pRegions contains VkCopyCommandTransformInfoQCOM in its pNext chain, then srcImage and dstImage **must** not be block-compressed images
- VUID-VkBlitImageInfo2KHR-pRegions-06207
  If any element of pRegions contains VkCopyCommandTransformInfoQCOM in its pNext chain, then srcImage **must** be of type VK_IMAGE_TYPE_2D
- VUID-VkBlitImageInfo2KHR-pRegions-06208
  If any element of pRegions contains VkCopyCommandTransformInfoQCOM in its pNext chain, then srcImage **must** not have a multi-planar format
- VUID-VkBlitImageInfo2-filter-09204
  If filter is VK_FILTER_CUBIC_EXT and if the selectableCubicWeights feature is not enabled then the cubic weights **must** be VK_CUBIC_FILTER_WEIGHTS_CATMULL_ROM_QCOM

## Valid Usage (Implicit)

- VUID-VkBlitImageInfo2-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_BLIT_IMAGE_INFO_2
- VUID-VkBlitImageInfo2-pNext-pNext
  pNext **must** be NULL or a pointer to a valid instance of VkBlitImageCubicWeightsInfoQCOM
- VUID-VkBlitImageInfo2-sType-unique
  The sType value of each struct in the pNext chain **must** be unique
- VUID-VkBlitImageInfo2-srcImage-parameter
  srcImage **must** be a valid VkImage handle
- VUID-VkBlitImageInfo2-srcImageLayout-parameter
  srcImageLayout **must** be a valid VkImageLayout value
- VUID-VkBlitImageInfo2-dstImage-parameter
  dstImage **must** be a valid VkImage handle
- VUID-VkBlitImageInfo2-dstImageLayout-parameter
  dstImageLayout **must** be a valid VkImageLayout value
- VUID-VkBlitImageInfo2-pRegions-parameter
  pRegions **must** be a valid pointer to an array of regionCount valid VkImageBlit2 structures
- VUID-VkBlitImageInfo2-filter-parameter
  filter **must** be a valid VkFilter value
- VUID-VkBlitImageInfo2-regionCount-arraylength
  regionCount **must** be greater than 0
- VUID-VkBlitImageInfo2-commonparent
  Both of dstImage, and srcImage **must** have been created, allocated, or retrieved from the same VkDevice

If filter is VK_FILTER_CUBIC_EXT and if the pNext chain of VkBlitImageInfo2 includes a VkBlitImageCubicWeightsInfoQCOM structure, then that structure specifies cubic weights are used in the blit. If that structure is not present, then cubic weights are considered to be VK_CUBIC_FILTER_WEIGHTS_CATMULL_ROM_QCOM.

The VkBlitImageCubicWeightsInfoQCOM structure is defined as:

```
// Provided by VK_QCOM_filter_cubic_weights
typedef struct VkBlitImageCubicWeightsInfoQCOM {
    VkStructureType             sType;
    const void*                 pNext;
    VkCubicFilterWeightsQCOM    cubicWeights;
} VkBlitImageCubicWeightsInfoQCOM;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- cubicWeights is a VkCubicFilterWeightsQCOM value controlling cubic filter weights for the blit.

### Valid Usage (Implicit)

- VUID-VkBlitImageCubicWeightsInfoQCOM-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_BLIT_IMAGE_CUBIC_WEIGHTS_INFO_QCOM
- VUID-VkBlitImageCubicWeightsInfoQCOM-cubicWeights-parameter
  cubicWeights **must** be a valid VkCubicFilterWeightsQCOM value

The VkImageBlit2 structure is defined as:

```
typedef struct VkImageBlit2 {
    VkStructureType             sType;
    const void*                 pNext;
    VkImageSubresourceLayers    srcSubresource;
    VkOffset3D                  srcOffsets[2];
    VkImageSubresourceLayers    dstSubresource;
    VkOffset3D                  dstOffsets[2];
} VkImageBlit2;
```

or the equivalent

```
// Provided by VK_KHR_copy_commands2
typedef VkImageBlit2 VkImageBlit2KHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.

- srcSubresource is the subresource to blit from.
- srcOffsets is a pointer to an array of two VkOffset3D structures specifying the bounds of the source region within srcSubresource.
- dstSubresource is the subresource to blit into.
- dstOffsets is a pointer to an array of two VkOffset3D structures specifying the bounds of the destination region within dstSubresource.

For each element of the pRegions array, a blit operation is performed for the specified source and destination regions.

### Valid Usage

- VUID-VkImageBlit2-aspectMask-00238
  The aspectMask member of srcSubresource and dstSubresource **must** match
- VUID-VkImageBlit2-layerCount-08800
  If neither of the layerCount members of srcSubresource or dstSubresource are VK_REMAINING_ARRAY_LAYERS, the layerCount members of srcSubresource or dstSubresource **must** match
- VUID-VkImageBlit2-layerCount-08801
  If one of the layerCount members of srcSubresource or dstSubresource is VK_REMAINING_ARRAY_LAYERS, the other member **must** be either VK_REMAINING_ARRAY_LAYERS or equal to the arrayLayers member of the VkImageCreateInfo used to create the image minus baseArrayLayer

### Valid Usage (Implicit)

- VUID-VkImageBlit2-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_IMAGE_BLIT_2
- VUID-VkImageBlit2-pNext-pNext
  pNext **must** be NULL or a pointer to a valid instance of VkCopyCommandTransformInfoQCOM
- VUID-VkImageBlit2-sType-unique
  The sType value of each struct in the pNext chain **must** be unique
- VUID-VkImageBlit2-srcSubresource-parameter
  srcSubresource **must** be a valid VkImageSubresourceLayers structure
- VUID-VkImageBlit2-dstSubresource-parameter
  dstSubresource **must** be a valid VkImageSubresourceLayers structure

For vkCmdBlitImage2KHR, each region copied can include a rotation. To specify a rotated region, add VkCopyCommandTransformInfoQCOM to the pNext chain of VkImageBlit2. For each region with a rotation specified, Image Blits with Scaling and Rotation specifies how coordinates are rotated prior to sampling from the source image. When rotation is specified, the source and destination

images **must** each be 2D images, have a 1x1x1 texel block extent, and only one plane.

### 20.5.1. Image Blits With Scaling and Rotation

When VkCopyCommandTransformInfoQCOM is in the `pNext` chain of VkImageBlit2, the specified region is rotated during the blit. The following description of rotated addressing replaces the description in vkCmdBlitImage.

The following code computes rotation of normalized coordinates.

```
// rotation of normalized coordinates
VkOffset2D RotateNormUV(VkOffset2D in, VkSurfaceTransformFlagBitsKHR flags)
{
    VkOffset2D output;
    switch (flags)
    {
        case VK_SURFACE_TRANSFORM_IDENTITY_BIT_KHR:
            out.x = in.x;
            out.y = in.y;
            break;
        case VK_SURFACE_TRANSFORM_ROTATE_90_BIT_KHR:
            out.x = in.y;
            out.y = 1.0 - in.x;
            break;
        case VK_SURFACE_TRANSFORM_ROTATE_180_BIT_KHR:
            out.x = 1.0 - in.x;
            out.y = 1.0 - in.y;
            break;
        case VK_SURFACE_TRANSFORM_ROTATE_270_BIT_KHR:
            out.x = 1.0 - in.y;
            out.y = in.x;
            break;
    }
    return out;
}
```

- For each destination texel, the integer coordinate of that texel is converted to an unnormalized texture coordinate, using the effective inverse of the equations described in unnormalized to integer conversion:

$$u_{base} = i + ½$$

$$v_{base} = j + ½$$

$$w_{base} = k + ½$$

- These base coordinates are then offset by the first destination offset:

$$u_{offset} = u_{base} - x_{dst0}$$

$$v_{offset} = v_{base} - y_{dst0}$$

$$w_{offset} = w_{base} - z_{dst0}$$

$$a_{offset} = a - \texttt{baseArrayCount}_{dst}$$

- The UV destination coordinates are scaled by the destination region, rotated, and scaled by the source region.

$$u_{dest_scaled} = u_{offset} / (x_{dst1} - x_{dst0})$$

$$v_{dest_scaled} = v_{offset} / (y_{dst1} - y_{dst0})$$

$$(u_{src_scaled}, v_{src_scaled}) = \texttt{RotateNormUV}(u_{dest_scaled}, v_{dest_scaled}, \texttt{transform})$$

$$u_{scaled} = u_{src_scaled} \times (x_{Src1} - x_{Src0})$$

$$v_{scaled} = v_{src_scaled} \times (y_{Src1} - y_{Src0})$$

- The W coordinate is unaffected by rotation. The scale is determined from the ratio of source and destination regions, and applied to the offset coordinate:

$$scale_w = (z_{Src1} - z_{Src0}) / (z_{dst1} - z_{dst0})$$

$$w_{scaled} = w_{offset} \times scale_w$$

- Finally the source offset is added to the scaled source coordinates, to determine the final unnormalized coordinates used to sample from `srcImage`:

$$u = u_{scaled} + x_{Src0}$$

$$v = v_{scaled} + y_{Src0}$$

$w = w_{scaled} + z_{Src0}$

$q = \texttt{mipLevel}$

$a = a_{offset} + \texttt{baseArrayCount}_{src}$

These coordinates are used to sample from the source image as described for Image Operations, with the filter mode equal to that of `filter`; a mipmap mode of `VK_SAMPLER_MIPMAP_MODE_NEAREST`; and an address mode of `VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_EDGE`. Implementations **must** clamp at the edge of the source image, and **may** additionally clamp to the edge of the source region.

## 20.6. Resolving Multisample Images

To resolve a multisample color image to a non-multisample color image, call:

```
// Provided by VK_VERSION_1_0
void vkCmdResolveImage(
    VkCommandBuffer                             commandBuffer,
    VkImage                                     srcImage,
    VkImageLayout                               srcImageLayout,
    VkImage                                     dstImage,
    VkImageLayout                               dstImageLayout,
    uint32_t                                    regionCount,
    const VkImageResolve*                       pRegions);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `srcImage` is the source image.
- `srcImageLayout` is the layout of the source image subresources for the resolve.
- `dstImage` is the destination image.
- `dstImageLayout` is the layout of the destination image subresources for the resolve.
- `regionCount` is the number of regions to resolve.
- `pRegions` is a pointer to an array of VkImageResolve structures specifying the regions to resolve.

During the resolve the samples corresponding to each pixel location in the source are converted to a single sample before being written to the destination. If the source formats are floating-point or normalized types, the sample values for each pixel are resolved in an implementation-dependent manner. If the source formats are integer types, a single sample's value is selected for each pixel.

`srcOffset` and `dstOffset` select the initial `x`, `y`, and `z` offsets in texels of the sub-regions of the source and destination image data. `extent` is the size in texels of the source image to resolve in `width`, `height` and `depth`. Each element of `pRegions` **must** be a region that is contained within its corresponding image.

Resolves are done layer by layer starting with `baseArrayLayer` member of `srcSubresource` for the source and `dstSubresource` for the destination. `layerCount` layers are resolved to the destination image.

### Valid Usage

- VUID-vkCmdResolveImage-commandBuffer-01837
  If `commandBuffer` is an unprotected command buffer and `protectedNoFault` is not supported, `srcImage` **must** not be a protected image
- VUID-vkCmdResolveImage-commandBuffer-01838
  If `commandBuffer` is an unprotected command buffer and `protectedNoFault` is not supported, `dstImage` **must** not be a protected image
- VUID-vkCmdResolveImage-commandBuffer-01839
  If `commandBuffer` is a protected command buffer and `protectedNoFault` is not supported, `dstImage` **must** not be an unprotected image
- VUID-vkCmdResolveImage-pRegions-00255
  The union of all source regions, and the union of all destination regions, specified by the elements of `pRegions`, **must** not overlap in memory
- VUID-vkCmdResolveImage-srcImage-00256
  If `srcImage` is non-sparse then it **must** be bound completely and contiguously to a single `VkDeviceMemory` object
- VUID-vkCmdResolveImage-srcImage-00257
  `srcImage` **must** have a sample count equal to any valid sample count value other than `VK_SAMPLE_COUNT_1_BIT`
- VUID-vkCmdResolveImage-dstImage-00258
  If `dstImage` is non-sparse then it **must** be bound completely and contiguously to a single `VkDeviceMemory` object
- VUID-vkCmdResolveImage-dstImage-00259
  `dstImage` **must** have a sample count equal to `VK_SAMPLE_COUNT_1_BIT`
- VUID-vkCmdResolveImage-srcImageLayout-00260
  `srcImageLayout` **must** specify the layout of the image subresources of `srcImage` specified in `pRegions` at the time this command is executed on a `VkDevice`
- VUID-vkCmdResolveImage-srcImageLayout-01400
  `srcImageLayout` **must** be `VK_IMAGE_LAYOUT_SHARED_PRESENT_KHR`, `VK_IMAGE_LAYOUT_TRANSFER_SRC_OPTIMAL` or `VK_IMAGE_LAYOUT_GENERAL`
- VUID-vkCmdResolveImage-dstImageLayout-00262
  `dstImageLayout` **must** specify the layout of the image subresources of `dstImage` specified in `pRegions` at the time this command is executed on a `VkDevice`
- VUID-vkCmdResolveImage-dstImageLayout-01401
  `dstImageLayout` **must** be `VK_IMAGE_LAYOUT_SHARED_PRESENT_KHR`, `VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL` or `VK_IMAGE_LAYOUT_GENERAL`
- VUID-vkCmdResolveImage-dstImage-02003

The format features of dstImage **must** contain VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BIT

- VUID-vkCmdResolveImage-linearColorAttachment-06519
  If the linearColorAttachment feature is enabled and the image is created with VK_IMAGE_TILING_LINEAR, the format features of dstImage **must** contain VK_FORMAT_FEATURE_2_LINEAR_COLOR_ATTACHMENT_BIT_NV
- VUID-vkCmdResolveImage-srcImage-01386
  srcImage and dstImage **must** have been created with the same image format
- VUID-vkCmdResolveImage-srcSubresource-01709
  The srcSubresource.mipLevel member of each element of pRegions **must** be less than the mipLevels specified in VkImageCreateInfo when srcImage was created
- VUID-vkCmdResolveImage-dstSubresource-01710
  The dstSubresource.mipLevel member of each element of pRegions **must** be less than the mipLevels specified in VkImageCreateInfo when dstImage was created
- VUID-vkCmdResolveImage-srcSubresource-01711
  If srcSubresource.layerCount is not VK_REMAINING_ARRAY_LAYERS, srcSubresource.baseArrayLayer + srcSubresource.layerCount of each element of pRegions **must** be less than or equal to the arrayLayers specified in VkImageCreateInfo when srcImage was created
- VUID-vkCmdResolveImage-dstSubresource-01712
  If dstSubresource.layerCount is not VK_REMAINING_ARRAY_LAYERS, dstSubresource.baseArrayLayer + dstSubresource.layerCount of each element of pRegions **must** be less than or equal to the arrayLayers specified in VkImageCreateInfo when dstImage was created
- VUID-vkCmdResolveImage-dstImage-02546
  dstImage and srcImage **must** not have been created with flags containing VK_IMAGE_CREATE_SUBSAMPLED_BIT_EXT
- VUID-vkCmdResolveImage-srcImage-04446
  If dstImage is of type VK_IMAGE_TYPE_3D, then for each element of pRegions, srcSubresource.layerCount **must** be 1
- VUID-vkCmdResolveImage-srcImage-04447
  If dstImage is of type VK_IMAGE_TYPE_3D, then for each element of pRegions, dstSubresource.baseArrayLayer **must** be 0 and dstSubresource.layerCount **must** be 1
- VUID-vkCmdResolveImage-srcOffset-00269
  For each element of pRegions, srcOffset.x and (extent.width + srcOffset.x) **must** both be greater than or equal to 0 and less than or equal to the width of the specified srcSubresource of srcImage
- VUID-vkCmdResolveImage-srcOffset-00270
  For each element of pRegions, srcOffset.y and (extent.height + srcOffset.y) **must** both be greater than or equal to 0 and less than or equal to the height of the specified srcSubresource of srcImage
- VUID-vkCmdResolveImage-srcImage-00271
  If srcImage is of type VK_IMAGE_TYPE_1D, then for each element of pRegions, srcOffset.y **must** be 0 and extent.height **must** be 1

- VUID-vkCmdResolveImage-srcOffset-00272
  For each element of `pRegions`, `srcOffset.z` and (`extent.depth` + `srcOffset.z`) **must** both be greater than or equal to `0` and less than or equal to the depth of the specified `srcSubresource` of `srcImage`
- VUID-vkCmdResolveImage-srcImage-00273
  If `srcImage` is of type `VK_IMAGE_TYPE_1D` or `VK_IMAGE_TYPE_2D`, then for each element of `pRegions`, `srcOffset.z` **must** be `0` and `extent.depth` **must** be `1`
- VUID-vkCmdResolveImage-dstOffset-00274
  For each element of `pRegions`, `dstOffset.x` and (`extent.width` + `dstOffset.x`) **must** both be greater than or equal to `0` and less than or equal to the width of the specified `dstSubresource` of `dstImage`
- VUID-vkCmdResolveImage-dstOffset-00275
  For each element of `pRegions`, `dstOffset.y` and (`extent.height` + `dstOffset.y`) **must** both be greater than or equal to `0` and less than or equal to the height of the specified `dstSubresource` of `dstImage`
- VUID-vkCmdResolveImage-dstImage-00276
  If `dstImage` is of type `VK_IMAGE_TYPE_1D`, then for each element of `pRegions`, `dstOffset.y` **must** be `0` and `extent.height` **must** be `1`
- VUID-vkCmdResolveImage-dstOffset-00277
  For each element of `pRegions`, `dstOffset.z` and (`extent.depth` + `dstOffset.z`) **must** both be greater than or equal to `0` and less than or equal to the depth of the specified `dstSubresource` of `dstImage`
- VUID-vkCmdResolveImage-dstImage-00278
  If `dstImage` is of type `VK_IMAGE_TYPE_1D` or `VK_IMAGE_TYPE_2D`, then for each element of `pRegions`, `dstOffset.z` **must** be `0` and `extent.depth` **must** be `1`
- VUID-vkCmdResolveImage-srcImage-06762
  `srcImage` **must** have been created with `VK_IMAGE_USAGE_TRANSFER_SRC_BIT` usage flag
- VUID-vkCmdResolveImage-srcImage-06763
  The format features of `srcImage` **must** contain `VK_FORMAT_FEATURE_TRANSFER_SRC_BIT`
- VUID-vkCmdResolveImage-dstImage-06764
  `dstImage` **must** have been created with `VK_IMAGE_USAGE_TRANSFER_DST_BIT` usage flag
- VUID-vkCmdResolveImage-dstImage-06765
  The format features of `dstImage` **must** contain `VK_FORMAT_FEATURE_TRANSFER_DST_BIT`

## Valid Usage (Implicit)

- VUID-vkCmdResolveImage-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdResolveImage-srcImage-parameter
  `srcImage` **must** be a valid VkImage handle
- VUID-vkCmdResolveImage-srcImageLayout-parameter
  `srcImageLayout` **must** be a valid VkImageLayout value

- VUID-vkCmdResolveImage-dstImage-parameter
  `dstImage` **must** be a valid VkImage handle
- VUID-vkCmdResolveImage-dstImageLayout-parameter
  `dstImageLayout` **must** be a valid VkImageLayout value
- VUID-vkCmdResolveImage-pRegions-parameter
  `pRegions` **must** be a valid pointer to an array of `regionCount` valid VkImageResolve structures
- VUID-vkCmdResolveImage-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdResolveImage-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- VUID-vkCmdResolveImage-renderpass
  This command **must** only be called outside of a render pass instance
- VUID-vkCmdResolveImage-videocoding
  This command **must** only be called outside of a video coding scope
- VUID-vkCmdResolveImage-regionCount-arraylength
  `regionCount` **must** be greater than `0`
- VUID-vkCmdResolveImage-commonparent
  Each of `commandBuffer`, `dstImage`, and `srcImage` **must** have been created, allocated, or retrieved from the same VkDevice

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Outside | Outside | Graphics | Action |

The `VkImageResolve` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkImageResolve {
    VkImageSubresourceLayers    srcSubresource;
    VkOffset3D                  srcOffset;
    VkImageSubresourceLayers    dstSubresource;
    VkOffset3D                  dstOffset;
    VkExtent3D                  extent;
} VkImageResolve;
```

- `srcSubresource` and `dstSubresource` are VkImageSubresourceLayers structures specifying the image subresources of the images used for the source and destination image data, respectively. Resolve of depth/stencil images is not supported.
- `srcOffset` and `dstOffset` select the initial `x`, `y`, and `z` offsets in texels of the sub-regions of the source and destination image data.
- `extent` is the size in texels of the source image to resolve in `width`, `height` and `depth`.

### Valid Usage

- VUID-VkImageResolve-aspectMask-00266
  The `aspectMask` member of `srcSubresource` and `dstSubresource` **must** only contain `VK_IMAGE_ASPECT_COLOR_BIT`
- VUID-VkImageResolve-layerCount-08803
  If neither of the `layerCount` members of `srcSubresource` or `dstSubresource` are `VK_REMAINING_ARRAY_LAYERS`, the `layerCount` member of `srcSubresource` and `dstSubresource` **must** match
- VUID-VkImageResolve-layerCount-08804
  If one of the `layerCount` members of `srcSubresource` or `dstSubresource` is `VK_REMAINING_ARRAY_LAYERS`, the other member **must** be either `VK_REMAINING_ARRAY_LAYERS` or equal to the `arrayLayers` member of the VkImageCreateInfo used to create the image minus `baseArrayLayer`

### Valid Usage (Implicit)

- VUID-VkImageResolve-srcSubresource-parameter
  `srcSubresource` **must** be a valid VkImageSubresourceLayers structure
- VUID-VkImageResolve-dstSubresource-parameter
  `dstSubresource` **must** be a valid VkImageSubresourceLayers structure

A more extensible version of the resolve image command is defined below.

To resolve a multisample image to a non-multisample image, call:

```
// Provided by VK_KHR_copy_commands2
void vkCmdResolveImage2KHR(
    VkCommandBuffer                             commandBuffer,
    const VkResolveImageInfo2*                  pResolveImageInfo);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `pResolveImageInfo` is a pointer to a VkResolveImageInfo2 structure describing the resolve parameters.

This command is functionally identical to vkCmdResolveImage, but includes extensible sub-structures that include `sType` and `pNext` parameters, allowing them to be more easily extended.

## Valid Usage

- VUID-vkCmdResolveImage2-commandBuffer-01837
  If `commandBuffer` is an unprotected command buffer and `protectedNoFault` is not supported, `srcImage` **must** not be a protected image
- VUID-vkCmdResolveImage2-commandBuffer-01838
  If `commandBuffer` is an unprotected command buffer and `protectedNoFault` is not supported, `dstImage` **must** not be a protected image
- VUID-vkCmdResolveImage2-commandBuffer-01839
  If `commandBuffer` is a protected command buffer and `protectedNoFault` is not supported, `dstImage` **must** not be an unprotected image

## Valid Usage (Implicit)

- VUID-vkCmdResolveImage2-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdResolveImage2-pResolveImageInfo-parameter
  `pResolveImageInfo` **must** be a valid pointer to a valid VkResolveImageInfo2 structure
- VUID-vkCmdResolveImage2-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdResolveImage2-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- VUID-vkCmdResolveImage2-renderpass
  This command **must** only be called outside of a render pass instance
- VUID-vkCmdResolveImage2-videocoding
  This command **must** only be called outside of a video coding scope

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Outside | Outside | Graphics | Action |

The `VkResolveImageInfo2` structure is defined as:

```
typedef struct VkResolveImageInfo2 {
    VkStructureType           sType;
    const void*               pNext;
    VkImage                   srcImage;
    VkImageLayout             srcImageLayout;
    VkImage                   dstImage;
    VkImageLayout             dstImageLayout;
    uint32_t                  regionCount;
    const VkImageResolve2*    pRegions;
} VkResolveImageInfo2;
```

or the equivalent

```
// Provided by VK_KHR_copy_commands2
typedef VkResolveImageInfo2 VkResolveImageInfo2KHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `srcImage` is the source image.
- `srcImageLayout` is the layout of the source image subresources for the resolve.
- `dstImage` is the destination image.
- `dstImageLayout` is the layout of the destination image subresources for the resolve.
- `regionCount` is the number of regions to resolve.
- `pRegions` is a pointer to an array of VkImageResolve2 structures specifying the regions to resolve.

## Valid Usage

- VUID-VkResolveImageInfo2-pRegions-00255
  The union of all source regions, and the union of all destination regions, specified by the elements of `pRegions`, **must** not overlap in memory
- VUID-VkResolveImageInfo2-srcImage-00256
  If `srcImage` is non-sparse then it **must** be bound completely and contiguously to a single `VkDeviceMemory` object
- VUID-VkResolveImageInfo2-srcImage-00257
  `srcImage` **must** have a sample count equal to any valid sample count value other than `VK_SAMPLE_COUNT_1_BIT`
- VUID-VkResolveImageInfo2-dstImage-00258
  If `dstImage` is non-sparse then it **must** be bound completely and contiguously to a single `VkDeviceMemory` object
- VUID-VkResolveImageInfo2-dstImage-00259
  `dstImage` **must** have a sample count equal to `VK_SAMPLE_COUNT_1_BIT`
- VUID-VkResolveImageInfo2-srcImageLayout-00260
  `srcImageLayout` **must** specify the layout of the image subresources of `srcImage` specified in `pRegions` at the time this command is executed on a `VkDevice`
- VUID-VkResolveImageInfo2-srcImageLayout-01400
  `srcImageLayout` **must** be `VK_IMAGE_LAYOUT_SHARED_PRESENT_KHR`, `VK_IMAGE_LAYOUT_TRANSFER_SRC_OPTIMAL` or `VK_IMAGE_LAYOUT_GENERAL`
- VUID-VkResolveImageInfo2-dstImageLayout-00262
  `dstImageLayout` **must** specify the layout of the image subresources of `dstImage` specified in `pRegions` at the time this command is executed on a `VkDevice`
- VUID-VkResolveImageInfo2-dstImageLayout-01401
  `dstImageLayout` **must** be `VK_IMAGE_LAYOUT_SHARED_PRESENT_KHR`, `VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL` or `VK_IMAGE_LAYOUT_GENERAL`
- VUID-VkResolveImageInfo2-dstImage-02003
  The format features of `dstImage` **must** contain `VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BIT`
- VUID-VkResolveImageInfo2-linearColorAttachment-06519
  If the `linearColorAttachment` feature is enabled and the image is created with `VK_IMAGE_TILING_LINEAR`, the format features of `dstImage` **must** contain `VK_FORMAT_FEATURE_2_LINEAR_COLOR_ATTACHMENT_BIT_NV`
- VUID-VkResolveImageInfo2-srcImage-01386
  `srcImage` and `dstImage` **must** have been created with the same image format
- VUID-VkResolveImageInfo2-srcSubresource-01709
  The `srcSubresource.mipLevel` member of each element of `pRegions` **must** be less than the `mipLevels` specified in VkImageCreateInfo when `srcImage` was created
- VUID-VkResolveImageInfo2-dstSubresource-01710
  The `dstSubresource.mipLevel` member of each element of `pRegions` **must** be less than the `mipLevels` specified in VkImageCreateInfo when `dstImage` was created

- VUID-VkResolveImageInfo2-srcSubresource-01711
  If srcSubresource.layerCount is not VK_REMAINING_ARRAY_LAYERS, srcSubresource.baseArrayLayer + srcSubresource.layerCount of each element of pRegions **must** be less than or equal to the arrayLayers specified in VkImageCreateInfo when srcImage was created
- VUID-VkResolveImageInfo2-dstSubresource-01712
  If dstSubresource.layerCount is not VK_REMAINING_ARRAY_LAYERS, dstSubresource.baseArrayLayer + dstSubresource.layerCount of each element of pRegions **must** be less than or equal to the arrayLayers specified in VkImageCreateInfo when dstImage was created
- VUID-VkResolveImageInfo2-dstImage-02546
  dstImage and srcImage **must** not have been created with flags containing VK_IMAGE_CREATE_SUBSAMPLED_BIT_EXT
- VUID-VkResolveImageInfo2-srcImage-04446
  If dstImage is of type VK_IMAGE_TYPE_3D, then for each element of pRegions, srcSubresource.layerCount **must** be 1
- VUID-VkResolveImageInfo2-srcImage-04447
  If dstImage is of type VK_IMAGE_TYPE_3D, then for each element of pRegions, dstSubresource.baseArrayLayer **must** be 0 and dstSubresource.layerCount **must** be 1
- VUID-VkResolveImageInfo2-srcOffset-00269
  For each element of pRegions, srcOffset.x and (extent.width + srcOffset.x) **must** both be greater than or equal to 0 and less than or equal to the width of the specified srcSubresource of srcImage
- VUID-VkResolveImageInfo2-srcOffset-00270
  For each element of pRegions, srcOffset.y and (extent.height + srcOffset.y) **must** both be greater than or equal to 0 and less than or equal to the height of the specified srcSubresource of srcImage
- VUID-VkResolveImageInfo2-srcImage-00271
  If srcImage is of type VK_IMAGE_TYPE_1D, then for each element of pRegions, srcOffset.y **must** be 0 and extent.height **must** be 1
- VUID-VkResolveImageInfo2-srcOffset-00272
  For each element of pRegions, srcOffset.z and (extent.depth + srcOffset.z) **must** both be greater than or equal to 0 and less than or equal to the depth of the specified srcSubresource of srcImage
- VUID-VkResolveImageInfo2-srcImage-00273
  If srcImage is of type VK_IMAGE_TYPE_1D or VK_IMAGE_TYPE_2D, then for each element of pRegions, srcOffset.z **must** be 0 and extent.depth **must** be 1
- VUID-VkResolveImageInfo2-dstOffset-00274
  For each element of pRegions, dstOffset.x and (extent.width + dstOffset.x) **must** both be greater than or equal to 0 and less than or equal to the width of the specified dstSubresource of dstImage
- VUID-VkResolveImageInfo2-dstOffset-00275
  For each element of pRegions, dstOffset.y and (extent.height + dstOffset.y) **must** both be

greater than or equal to `0` and less than or equal to the height of the specified `dstSubresource` of `dstImage`

- VUID-VkResolveImageInfo2-dstImage-00276
  If `dstImage` is of type `VK_IMAGE_TYPE_1D`, then for each element of `pRegions`, `dstOffset.y` **must** be `0` and `extent.height` **must** be `1`
- VUID-VkResolveImageInfo2-dstOffset-00277
  For each element of `pRegions`, `dstOffset.z` and (`extent.depth` + `dstOffset.z`) **must** both be greater than or equal to `0` and less than or equal to the depth of the specified `dstSubresource` of `dstImage`
- VUID-VkResolveImageInfo2-dstImage-00278
  If `dstImage` is of type `VK_IMAGE_TYPE_1D` or `VK_IMAGE_TYPE_2D`, then for each element of `pRegions`, `dstOffset.z` **must** be `0` and `extent.depth` **must** be `1`
- VUID-VkResolveImageInfo2-srcImage-06762
  `srcImage` **must** have been created with `VK_IMAGE_USAGE_TRANSFER_SRC_BIT` usage flag
- VUID-VkResolveImageInfo2-srcImage-06763
  The format features of `srcImage` **must** contain `VK_FORMAT_FEATURE_TRANSFER_SRC_BIT`
- VUID-VkResolveImageInfo2-dstImage-06764
  `dstImage` **must** have been created with `VK_IMAGE_USAGE_TRANSFER_DST_BIT` usage flag
- VUID-VkResolveImageInfo2-dstImage-06765
  The format features of `dstImage` **must** contain `VK_FORMAT_FEATURE_TRANSFER_DST_BIT`

## Valid Usage (Implicit)

- VUID-VkResolveImageInfo2-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_RESOLVE_IMAGE_INFO_2`
- VUID-VkResolveImageInfo2-pNext-pNext
  `pNext` **must** be `NULL`
- VUID-VkResolveImageInfo2-srcImage-parameter
  `srcImage` **must** be a valid VkImage handle
- VUID-VkResolveImageInfo2-srcImageLayout-parameter
  `srcImageLayout` **must** be a valid VkImageLayout value
- VUID-VkResolveImageInfo2-dstImage-parameter
  `dstImage` **must** be a valid VkImage handle
- VUID-VkResolveImageInfo2-dstImageLayout-parameter
  `dstImageLayout` **must** be a valid VkImageLayout value
- VUID-VkResolveImageInfo2-pRegions-parameter
  `pRegions` **must** be a valid pointer to an array of `regionCount` valid VkImageResolve2 structures
- VUID-VkResolveImageInfo2-regionCount-arraylength
  `regionCount` **must** be greater than `0`
- VUID-VkResolveImageInfo2-commonparent

Both of dstImage, and srcImage **must** have been created, allocated, or retrieved from the same VkDevice

The VkImageResolve2 structure is defined as:

```
typedef struct VkImageResolve2 {
    VkStructureType             sType;
    const void*                 pNext;
    VkImageSubresourceLayers    srcSubresource;
    VkOffset3D                  srcOffset;
    VkImageSubresourceLayers    dstSubresource;
    VkOffset3D                  dstOffset;
    VkExtent3D                  extent;
} VkImageResolve2;
```

or the equivalent

```
// Provided by VK_KHR_copy_commands2
typedef VkImageResolve2 VkImageResolve2KHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- srcSubresource and dstSubresource are VkImageSubresourceLayers structures specifying the image subresources of the images used for the source and destination image data, respectively. Resolve of depth/stencil images is not supported.
- srcOffset and dstOffset select the initial x, y, and z offsets in texels of the sub-regions of the source and destination image data.
- extent is the size in texels of the source image to resolve in width, height and depth.

### Valid Usage

- VUID-VkImageResolve2-aspectMask-00266
  The aspectMask member of srcSubresource and dstSubresource **must** only contain VK_IMAGE_ASPECT_COLOR_BIT
- VUID-VkImageResolve2-layerCount-08803
  If neither of the layerCount members of srcSubresource or dstSubresource are VK_REMAINING_ARRAY_LAYERS, the layerCount member of srcSubresource and dstSubresource **must** match
- VUID-VkImageResolve2-layerCount-08804
  If one of the layerCount members of srcSubresource or dstSubresource is VK_REMAINING_ARRAY_LAYERS, the other member **must** be either VK_REMAINING_ARRAY_LAYERS or equal to the arrayLayers member of the VkImageCreateInfo used to create the image minus baseArrayLayer

**Valid Usage (Implicit)**

- VUID-VkImageResolve2-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_IMAGE_RESOLVE_2`
- VUID-VkImageResolve2-pNext-pNext
  `pNext` **must** be `NULL`
- VUID-VkImageResolve2-srcSubresource-parameter
  `srcSubresource` **must** be a valid VkImageSubresourceLayers structure
- VUID-VkImageResolve2-dstSubresource-parameter
  `dstSubresource` **must** be a valid VkImageSubresourceLayers structure

## 20.7. Buffer Markers

To write a 32-bit marker value into a buffer as a pipelined operation, call:

```
// Provided by VK_KHR_synchronization2 with VK_AMD_buffer_marker
void vkCmdWriteBufferMarker2AMD(
    VkCommandBuffer                             commandBuffer,
    VkPipelineStageFlags2                       stage,
    VkBuffer                                    dstBuffer,
    VkDeviceSize                                dstOffset,
    uint32_t                                    marker);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `stage` specifies the pipeline stage whose completion triggers the marker write.
- `dstBuffer` is the buffer where the marker will be written.
- `dstOffset` is the byte offset into the buffer where the marker will be written.
- `marker` is the 32-bit value of the marker.

The command will write the 32-bit marker value into the buffer only after all preceding commands have finished executing up to at least the specified pipeline stage. This includes the completion of other preceding `vkCmdWriteBufferMarker2AMD` commands so long as their specified pipeline stages occur either at the same time or earlier than this command's specified `stage`.

While consecutive buffer marker writes with the same `stage` parameter implicitly complete in submission order, memory and execution dependencies between buffer marker writes and other operations **must** still be explicitly ordered using synchronization commands. The access scope for buffer marker writes falls under the `VK_ACCESS_TRANSFER_WRITE_BIT`, and the pipeline stages for identifying the synchronization scope **must** include both `stage` and `VK_PIPELINE_STAGE_TRANSFER_BIT`.

| | |
|---|---|
| **NOTE** | Similar to `vkCmdWriteTimestamp2KHR`, if an implementation is unable to write a marker at any specific pipeline stage, it **may** instead do so at any logically later stage. |

**NOTE** Implementations **may** only support a limited number of pipelined marker write operations in flight at a given time. Thus an excessive number of marker write operations **may** degrade command execution performance.

## Valid Usage

- VUID-vkCmdWriteBufferMarker2AMD-stage-03929
  If the `geometryShader` feature is not enabled, `stage` **must** not contain `VK_PIPELINE_STAGE_2_GEOMETRY_SHADER_BIT`
- VUID-vkCmdWriteBufferMarker2AMD-stage-03930
  If the `tessellationShader` feature is not enabled, `stage` **must** not contain `VK_PIPELINE_STAGE_2_TESSELLATION_CONTROL_SHADER_BIT` or `VK_PIPELINE_STAGE_2_TESSELLATION_EVALUATION_SHADER_BIT`
- VUID-vkCmdWriteBufferMarker2AMD-stage-03931
  If the `conditionalRendering` feature is not enabled, `stage` **must** not contain `VK_PIPELINE_STAGE_2_CONDITIONAL_RENDERING_BIT_EXT`
- VUID-vkCmdWriteBufferMarker2AMD-stage-03932
  If the `fragmentDensityMap` feature is not enabled, `stage` **must** not contain `VK_PIPELINE_STAGE_2_FRAGMENT_DENSITY_PROCESS_BIT_EXT`
- VUID-vkCmdWriteBufferMarker2AMD-stage-03933
  If the `transformFeedback` feature is not enabled, `stage` **must** not contain `VK_PIPELINE_STAGE_2_TRANSFORM_FEEDBACK_BIT_EXT`
- VUID-vkCmdWriteBufferMarker2AMD-stage-03934
  If the `meshShader` feature is not enabled, `stage` **must** not contain `VK_PIPELINE_STAGE_2_MESH_SHADER_BIT_EXT`
- VUID-vkCmdWriteBufferMarker2AMD-stage-03935
  If the `taskShader` feature is not enabled, `stage` **must** not contain `VK_PIPELINE_STAGE_2_TASK_SHADER_BIT_EXT`
- VUID-vkCmdWriteBufferMarker2AMD-stage-07316
  If neither the `shadingRateImage` or `attachmentFragmentShadingRate` are enabled, `stage` **must** not contain `VK_PIPELINE_STAGE_2_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR`
- VUID-vkCmdWriteBufferMarker2AMD-stage-04957
  If the `subpassShading` feature is not enabled, `stage` **must** not contain `VK_PIPELINE_STAGE_2_SUBPASS_SHADER_BIT_HUAWEI`
- VUID-vkCmdWriteBufferMarker2AMD-stage-04995
  If the `invocationMask` feature is not enabled, `stage` **must** not contain `VK_PIPELINE_STAGE_2_INVOCATION_MASK_BIT_HUAWEI`
- VUID-vkCmdWriteBufferMarker2AMD-stage-07946
  If neither the VK_NV_ray_tracing extension or `rayTracingPipeline` feature are enabled, `stage` **must** not contain `VK_PIPELINE_STAGE_2_RAY_TRACING_SHADER_BIT_KHR`
- VUID-vkCmdWriteBufferMarker2AMD-synchronization2-03893
  The `synchronization2` feature **must** be enabled

- VUID-vkCmdWriteBufferMarker2AMD-stage-03894
  `stage` **must** include only a single pipeline stage
- VUID-vkCmdWriteBufferMarker2AMD-stage-03895
  `stage` **must** include only stages that are valid for the queue family that was used to create the command pool that `commandBuffer` was allocated from
- VUID-vkCmdWriteBufferMarker2AMD-dstOffset-03896
  `dstOffset` **must** be less than or equal to the size of `dstBuffer` minus `4`
- VUID-vkCmdWriteBufferMarker2AMD-dstBuffer-03897
  `dstBuffer` **must** have been created with the `VK_BUFFER_USAGE_TRANSFER_DST_BIT` usage flag
- VUID-vkCmdWriteBufferMarker2AMD-dstBuffer-03898
  If `dstBuffer` is non-sparse then it **must** be bound completely and contiguously to a single `VkDeviceMemory` object
- VUID-vkCmdWriteBufferMarker2AMD-dstOffset-03899
  `dstOffset` **must** be a multiple of `4`

## Valid Usage (Implicit)

- VUID-vkCmdWriteBufferMarker2AMD-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdWriteBufferMarker2AMD-stage-parameter
  `stage` **must** be a valid combination of VkPipelineStageFlagBits2 values
- VUID-vkCmdWriteBufferMarker2AMD-dstBuffer-parameter
  `dstBuffer` **must** be a valid VkBuffer handle
- VUID-vkCmdWriteBufferMarker2AMD-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdWriteBufferMarker2AMD-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support transfer, graphics, or compute operations
- VUID-vkCmdWriteBufferMarker2AMD-videocoding
  This command **must** only be called outside of a video coding scope
- VUID-vkCmdWriteBufferMarker2AMD-commonparent
  Both of `commandBuffer`, and `dstBuffer` **must** have been created, allocated, or retrieved from the same VkDevice

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

### Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Transfer<br>Graphics<br>Compute | Action |

To write a 32-bit marker value into a buffer as a pipelined operation, call:

```
// Provided by VK_AMD_buffer_marker
void vkCmdWriteBufferMarkerAMD(
    VkCommandBuffer                             commandBuffer,
    VkPipelineStageFlagBits                     pipelineStage,
    VkBuffer                                    dstBuffer,
    VkDeviceSize                                dstOffset,
    uint32_t                                    marker);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `pipelineStage` is a VkPipelineStageFlagBits value specifying the pipeline stage whose completion triggers the marker write.
- `dstBuffer` is the buffer where the marker will be written to.
- `dstOffset` is the byte offset into the buffer where the marker will be written to.
- `marker` is the 32-bit value of the marker.

The command will write the 32-bit marker value into the buffer only after all preceding commands have finished executing up to at least the specified pipeline stage. This includes the completion of other preceding `vkCmdWriteBufferMarkerAMD` commands so long as their specified pipeline stages occur either at the same time or earlier than this command's specified `pipelineStage`.

While consecutive buffer marker writes with the same `pipelineStage` parameter are implicitly complete in submission order, memory and execution dependencies between buffer marker writes and other operations **must** still be explicitly ordered using synchronization commands. The access scope for buffer marker writes falls under the `VK_ACCESS_TRANSFER_WRITE_BIT`, and the pipeline stages for identifying the synchronization scope **must** include both `pipelineStage` and `VK_PIPELINE_STAGE_TRANSFER_BIT`.

**NOTE** Similar to `vkCmdWriteTimestamp`, if an implementation is unable to write a marker at any specific pipeline stage, it **may** instead do so at any logically later stage.

**NOTE** Implementations **may** only support a limited number of pipelined marker write operations in flight at a given time, thus excessive number of marker write operations **may** degrade command execution performance.

## Valid Usage

- VUID-vkCmdWriteBufferMarkerAMD-pipelineStage-04074
  `pipelineStage` **must** be a valid stage for the queue family that was used to create the command pool that `commandBuffer` was allocated from
- VUID-vkCmdWriteBufferMarkerAMD-pipelineStage-04075
  If the `geometryShader` feature is not enabled, `pipelineStage` **must** not be `VK_PIPELINE_STAGE_GEOMETRY_SHADER_BIT`
- VUID-vkCmdWriteBufferMarkerAMD-pipelineStage-04076
  If the `tessellationShader` feature is not enabled, `pipelineStage` **must** not be `VK_PIPELINE_STAGE_TESSELLATION_CONTROL_SHADER_BIT` or `VK_PIPELINE_STAGE_TESSELLATION_EVALUATION_SHADER_BIT`
- VUID-vkCmdWriteBufferMarkerAMD-pipelineStage-04077
  If the `conditionalRendering` feature is not enabled, `pipelineStage` **must** not be `VK_PIPELINE_STAGE_CONDITIONAL_RENDERING_BIT_EXT`
- VUID-vkCmdWriteBufferMarkerAMD-pipelineStage-04078
  If the `fragmentDensityMap` feature is not enabled, `pipelineStage` **must** not be `VK_PIPELINE_STAGE_FRAGMENT_DENSITY_PROCESS_BIT_EXT`
- VUID-vkCmdWriteBufferMarkerAMD-pipelineStage-04079
  If the `transformFeedback` feature is not enabled, `pipelineStage` **must** not be `VK_PIPELINE_STAGE_TRANSFORM_FEEDBACK_BIT_EXT`
- VUID-vkCmdWriteBufferMarkerAMD-pipelineStage-04080
  If the `meshShader` feature is not enabled, `pipelineStage` **must** not be `VK_PIPELINE_STAGE_MESH_SHADER_BIT_EXT`
- VUID-vkCmdWriteBufferMarkerAMD-pipelineStage-07077
  If the `taskShader` feature is not enabled, `pipelineStage` **must** not be `VK_PIPELINE_STAGE_TASK_SHADER_BIT_EXT`
- VUID-vkCmdWriteBufferMarkerAMD-shadingRateImage-07314
  If neither the `shadingRateImage` or `attachmentFragmentShadingRate` are enabled, `pipelineStage` **must** not be `VK_PIPELINE_STAGE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR`
- VUID-vkCmdWriteBufferMarkerAMD-synchronization2-06489
  If the `synchronization2` feature is not enabled, `pipelineStage` **must** not be `VK_PIPELINE_STAGE_NONE`
- VUID-vkCmdWriteBufferMarkerAMD-rayTracingPipeline-07943
  If neither the VK_NV_ray_tracing extension or `rayTracingPipeline` feature are enabled, `pipelineStage` **must** not be `VK_PIPELINE_STAGE_RAY_TRACING_SHADER_BIT_KHR`
- VUID-vkCmdWriteBufferMarkerAMD-dstOffset-01798
  `dstOffset` **must** be less than or equal to the size of `dstBuffer` minus `4`
- VUID-vkCmdWriteBufferMarkerAMD-dstBuffer-01799
  `dstBuffer` **must** have been created with `VK_BUFFER_USAGE_TRANSFER_DST_BIT` usage flag
- VUID-vkCmdWriteBufferMarkerAMD-dstBuffer-01800
  If `dstBuffer` is non-sparse then it **must** be bound completely and contiguously to a single

VkDeviceMemory object

- VUID-vkCmdWriteBufferMarkerAMD-dstOffset-01801
  dstOffset **must** be a multiple of 4

## Valid Usage (Implicit)

- VUID-vkCmdWriteBufferMarkerAMD-commandBuffer-parameter
  commandBuffer **must** be a valid VkCommandBuffer handle
- VUID-vkCmdWriteBufferMarkerAMD-pipelineStage-parameter
  If pipelineStage is not 0, pipelineStage **must** be a valid VkPipelineStageFlagBits value
- VUID-vkCmdWriteBufferMarkerAMD-dstBuffer-parameter
  dstBuffer **must** be a valid VkBuffer handle
- VUID-vkCmdWriteBufferMarkerAMD-commandBuffer-recording
  commandBuffer **must** be in the recording state
- VUID-vkCmdWriteBufferMarkerAMD-commandBuffer-cmdpool
  The VkCommandPool that commandBuffer was allocated from **must** support transfer, graphics, or compute operations
- VUID-vkCmdWriteBufferMarkerAMD-videocoding
  This command **must** only be called outside of a video coding scope
- VUID-vkCmdWriteBufferMarkerAMD-commonparent
  Both of commandBuffer, and dstBuffer **must** have been created, allocated, or retrieved from the same VkDevice

## Host Synchronization

- Host access to commandBuffer **must** be externally synchronized
- Host access to the VkCommandPool that commandBuffer was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Transfer<br>Graphics<br>Compute | Action |

# Chapter 21. Drawing Commands

*Drawing commands* (commands with `Draw` in the name) provoke work in a graphics pipeline. Drawing commands are recorded into a command buffer and when executed by a queue, will produce work which executes according to the bound graphics pipeline, or if the `shaderObject` feature is enabled, any shader objects bound to graphics stages. A graphics pipeline or a combination of one or more graphics shader objects **must** be bound to a command buffer before any drawing commands are recorded in that command buffer.

Drawing can be achieved in two modes:

- Programmable Mesh Shading, the mesh shader assembles primitives, or
- Programmable Primitive Shading, the input primitives are assembled as follows.

Each draw is made up of zero or more vertices and zero or more instances, which are processed by the device and result in the assembly of primitives. Primitives are assembled according to the `pInputAssemblyState` member of the VkGraphicsPipelineCreateInfo structure, which is of type `VkPipelineInputAssemblyStateCreateInfo`:

```
// Provided by VK_VERSION_1_0
typedef struct VkPipelineInputAssemblyStateCreateInfo {
    VkStructureType                             sType;
    const void*                                 pNext;
    VkPipelineInputAssemblyStateCreateFlags     flags;
    VkPrimitiveTopology                         topology;
    VkBool32                                    primitiveRestartEnable;
} VkPipelineInputAssemblyStateCreateInfo;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is reserved for future use.
- `topology` is a VkPrimitiveTopology defining the primitive topology, as described below.
- `primitiveRestartEnable` controls whether a special vertex index value is treated as restarting the assembly of primitives. This enable only applies to indexed draws (vkCmdDrawIndexed, vkCmdDrawMultiIndexedEXT, and vkCmdDrawIndexedIndirect), and the special index value is either 0xFFFFFFFF when the `indexType` parameter of `vkCmdBindIndexBuffer2KHR` or `vkCmdBindIndexBuffer` is equal to `VK_INDEX_TYPE_UINT32`, 0xFF when `indexType` is equal to `VK_INDEX_TYPE_UINT8_KHR`, or 0xFFFF when `indexType` is equal to `VK_INDEX_TYPE_UINT16`. Primitive restart is not allowed for “list” topologies, unless one of the features `primitiveTopologyPatchListRestart` (for `VK_PRIMITIVE_TOPOLOGY_PATCH_LIST`) or `primitiveTopologyListRestart` (for all other list topologies) is enabled.

Restarting the assembly of primitives discards the most recent index values if those elements formed an incomplete primitive, and restarts the primitive assembly using the subsequent indices, but only assembling the immediately following element through the end of the originally specified elements. The primitive restart index value comparison is performed before adding the

vertexOffset value to the index value.

### Valid Usage

- VUID-VkPipelineInputAssemblyStateCreateInfo-topology-06252
  If the primitiveTopologyListRestart feature is not enabled, and topology is VK_PRIMITIVE_TOPOLOGY_POINT_LIST, VK_PRIMITIVE_TOPOLOGY_LINE_LIST, VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST, VK_PRIMITIVE_TOPOLOGY_LINE_LIST_WITH_ADJACENCY, or VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST_WITH_ADJACENCY, primitiveRestartEnable **must** be VK_FALSE
- VUID-VkPipelineInputAssemblyStateCreateInfo-topology-06253
  If the primitiveTopologyPatchListRestart feature is not enabled, and topology is VK_PRIMITIVE_TOPOLOGY_PATCH_LIST, primitiveRestartEnable **must** be VK_FALSE
- VUID-VkPipelineInputAssemblyStateCreateInfo-topology-00429
  If the geometryShader feature is not enabled, topology **must** not be any of VK_PRIMITIVE_TOPOLOGY_LINE_LIST_WITH_ADJACENCY, VK_PRIMITIVE_TOPOLOGY_LINE_STRIP_WITH_ADJACENCY, VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST_WITH_ADJACENCY or VK_PRIMITIVE_TOPOLOGY_TRIANGLE_STRIP_WITH_ADJACENCY
- VUID-VkPipelineInputAssemblyStateCreateInfo-topology-00430
  If the tessellationShader feature is not enabled, topology **must** not be VK_PRIMITIVE_TOPOLOGY_PATCH_LIST
- VUID-VkPipelineInputAssemblyStateCreateInfo-triangleFans-04452
  If the VK_KHR_portability_subset extension is enabled, and VkPhysicalDevicePortabilitySubsetFeaturesKHR::triangleFans is VK_FALSE, topology **must** not be VK_PRIMITIVE_TOPOLOGY_TRIANGLE_FAN

### Valid Usage (Implicit)

- VUID-VkPipelineInputAssemblyStateCreateInfo-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PIPELINE_INPUT_ASSEMBLY_STATE_CREATE_INFO
- VUID-VkPipelineInputAssemblyStateCreateInfo-pNext-pNext
  pNext **must** be NULL
- VUID-VkPipelineInputAssemblyStateCreateInfo-flags-zerobitmask
  flags **must** be 0
- VUID-VkPipelineInputAssemblyStateCreateInfo-topology-parameter
  topology **must** be a valid VkPrimitiveTopology value

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkPipelineInputAssemblyStateCreateFlags;
```

VkPipelineInputAssemblyStateCreateFlags is a bitmask type for setting a mask, but is currently

reserved for future use.

To dynamically control whether a special vertex index value is treated as restarting the assembly of primitives, call:

```
// Provided by VK_EXT_extended_dynamic_state2, VK_EXT_shader_object
void vkCmdSetPrimitiveRestartEnableEXT(
    VkCommandBuffer                             commandBuffer,
    VkBool32                                    primitiveRestartEnable);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `primitiveRestartEnable` controls whether a special vertex index value is treated as restarting the assembly of primitives. It behaves in the same way as `VkPipelineInputAssemblyStateCreateInfo::primitiveRestartEnable`

This command sets the primitive restart enable for subsequent drawing commands when drawing using shader objects, or when the graphics pipeline is created with `VK_DYNAMIC_STATE_PRIMITIVE_RESTART_ENABLE` set in VkPipelineDynamicStateCreateInfo::`pDynamicStates`. Otherwise, this state is specified by the VkPipelineInputAssemblyStateCreateInfo::`primitiveRestartEnable` value used to create the currently active pipeline.

## Valid Usage

- VUID-vkCmdSetPrimitiveRestartEnable-None-08970
  At least one of the following **must** be true:
  - the `extendedDynamicState2` feature is enabled
  - the `shaderObject` feature is enabled

## Valid Usage (Implicit)

- VUID-vkCmdSetPrimitiveRestartEnable-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdSetPrimitiveRestartEnable-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdSetPrimitiveRestartEnable-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- VUID-vkCmdSetPrimitiveRestartEnable-videocoding
  This command **must** only be called outside of a video coding scope

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized

- Host access to the VkCommandPool that commandBuffer was allocated from **must** be externally synchronized

**Command Properties**

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics | State |

## 21.1. Primitive Topologies

*Primitive topology* determines how consecutive vertices are organized into primitives, and determines the type of primitive that is used at the beginning of the graphics pipeline. The effective topology for later stages of the pipeline is altered by tessellation or geometry shading (if either is in use) and depends on the execution modes of those shaders. In the case of mesh shading the only effective topology is defined by the execution mode of the mesh shader.

The primitive topologies defined by VkPrimitiveTopology are:

```
// Provided by VK_VERSION_1_0
typedef enum VkPrimitiveTopology {
    VK_PRIMITIVE_TOPOLOGY_POINT_LIST = 0,
    VK_PRIMITIVE_TOPOLOGY_LINE_LIST = 1,
    VK_PRIMITIVE_TOPOLOGY_LINE_STRIP = 2,
    VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST = 3,
    VK_PRIMITIVE_TOPOLOGY_TRIANGLE_STRIP = 4,
    VK_PRIMITIVE_TOPOLOGY_TRIANGLE_FAN = 5,
    VK_PRIMITIVE_TOPOLOGY_LINE_LIST_WITH_ADJACENCY = 6,
    VK_PRIMITIVE_TOPOLOGY_LINE_STRIP_WITH_ADJACENCY = 7,
    VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST_WITH_ADJACENCY = 8,
    VK_PRIMITIVE_TOPOLOGY_TRIANGLE_STRIP_WITH_ADJACENCY = 9,
    VK_PRIMITIVE_TOPOLOGY_PATCH_LIST = 10,
} VkPrimitiveTopology;
```

- VK_PRIMITIVE_TOPOLOGY_POINT_LIST specifies a series of separate point primitives.
- VK_PRIMITIVE_TOPOLOGY_LINE_LIST specifies a series of separate line primitives.
- VK_PRIMITIVE_TOPOLOGY_LINE_STRIP specifies a series of connected line primitives with consecutive lines sharing a vertex.
- VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST specifies a series of separate triangle primitives.
- VK_PRIMITIVE_TOPOLOGY_TRIANGLE_STRIP specifies a series of connected triangle primitives with consecutive triangles sharing an edge.
- VK_PRIMITIVE_TOPOLOGY_TRIANGLE_FAN specifies a series of connected triangle primitives with all

triangles sharing a common vertex. If the VK_KHR_portability_subset extension is enabled, and VkPhysicalDevicePortabilitySubsetFeaturesKHR::triangleFans is VK_FALSE, then triangle fans are not supported by the implementation, and VK_PRIMITIVE_TOPOLOGY_TRIANGLE_FAN **must** not be used.

- VK_PRIMITIVE_TOPOLOGY_LINE_LIST_WITH_ADJACENCY specifies a series of separate line primitives with adjacency.
- VK_PRIMITIVE_TOPOLOGY_LINE_STRIP_WITH_ADJACENCY specifies a series of connected line primitives with adjacency, with consecutive primitives sharing three vertices.
- VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST_WITH_ADJACENCY specifies a series of separate triangle primitives with adjacency.
- VK_PRIMITIVE_TOPOLOGY_TRIANGLE_STRIP_WITH_ADJACENCY specifies connected triangle primitives with adjacency, with consecutive triangles sharing an edge.
- VK_PRIMITIVE_TOPOLOGY_PATCH_LIST specifies separate patch primitives.

Each primitive topology, and its construction from a list of vertices, is described in detail below with a supporting diagram, according to the following key:

| | | |
|---|---|---|
| • | Vertex | A point in 3-dimensional space. Positions chosen within the diagrams are arbitrary and for illustration only. |
| 5 | Vertex Number | Sequence position of a vertex within the provided vertex data. |
| | Provoking Vertex | Provoking vertex within the main primitive. The tail is angled towards the relevant primitive. Used in flat shading. |
| ——— | Primitive Edge | An edge connecting the points of a main primitive. |
| - - - - - - | Adjacency Edge | Points connected by these lines do not contribute to a main primitive, and are only accessible in a geometry shader. |
| | Winding Order | The relative order in which vertices are defined within a primitive, used in the facing determination. This ordering has no specific start or end point. |

The diagrams are supported with mathematical definitions where the vertices (v) and primitives (p) are numbered starting from 0; $v_0$ is the first vertex in the provided data and $p_0$ is the first primitive in the set of primitives defined by the vertices and topology.

To dynamically set primitive topology, call:

```
// Provided by VK_EXT_extended_dynamic_state, VK_EXT_shader_object
void vkCmdSetPrimitiveTopologyEXT(
    VkCommandBuffer                             commandBuffer,
    VkPrimitiveTopology                         primitiveTopology);
```

- commandBuffer is the command buffer into which the command will be recorded.
- primitiveTopology specifies the primitive topology to use for drawing.

This command sets the primitive topology for subsequent drawing commands when drawing using

shader objects, or when the graphics pipeline is created with `VK_DYNAMIC_STATE_PRIMITIVE_TOPOLOGY` set in VkPipelineDynamicStateCreateInfo::`pDynamicStates`. Otherwise, this state is specified by the VkPipelineInputAssemblyStateCreateInfo::`topology` value used to create the currently active pipeline.

## Valid Usage

- VUID-vkCmdSetPrimitiveTopology-None-08971
  At least one of the following **must** be true:
  - the `extendedDynamicState` feature is enabled
  - the `shaderObject` feature is enabled

## Valid Usage (Implicit)

- VUID-vkCmdSetPrimitiveTopology-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdSetPrimitiveTopology-primitiveTopology-parameter
  `primitiveTopology` **must** be a valid VkPrimitiveTopology value
- VUID-vkCmdSetPrimitiveTopology-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdSetPrimitiveTopology-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- VUID-vkCmdSetPrimitiveTopology-videocoding
  This command **must** only be called outside of a video coding scope

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics | State |

### 21.1.1. Topology Class

The primitive topologies are grouped into the following topology classes:

*Table 31. Topology classes*

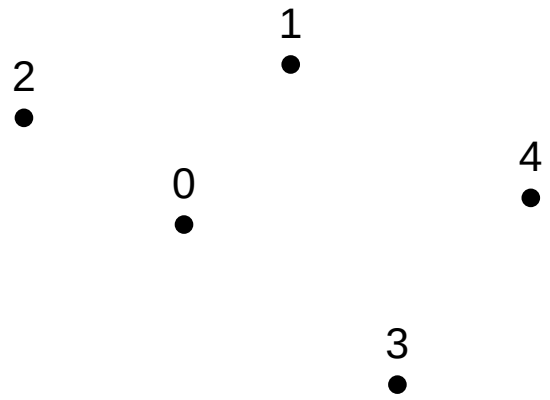
| Topology Class | Primitive Topology |
|---|---|
| Point | `VK_PRIMITIVE_TOPOLOGY_POINT_LIST` |
| Line | `VK_PRIMITIVE_TOPOLOGY_LINE_LIST`, `VK_PRIMITIVE_TOPOLOGY_LINE_STRIP`, `VK_PRIMITIVE_TOPOLOGY_LINE_LIST_WITH_ADJACENCY`, `VK_PRIMITIVE_TOPOLOGY_LINE_STRIP_WITH_ADJACENCY` |
| Triangle | `VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST`, `VK_PRIMITIVE_TOPOLOGY_TRIANGLE_STRIP`, `VK_PRIMITIVE_TOPOLOGY_TRIANGLE_FAN`, `VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST_WITH_ADJACENCY`, `VK_PRIMITIVE_TOPOLOGY_TRIANGLE_STRIP_WITH_ADJACENCY` |
| Patch | `VK_PRIMITIVE_TOPOLOGY_PATCH_LIST` |

### 21.1.2. Point Lists

When the topology is `VK_PRIMITIVE_TOPOLOGY_POINT_LIST`, each consecutive vertex defines a single point primitive, according to the equation:

$$p_i = \{v_i\}$$

As there is only one vertex, that vertex is the provoking vertex. The number of primitives generated is equal to `vertexCount`.



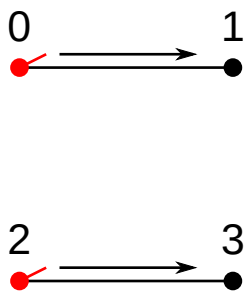

### 21.1.3. Line Lists

When the primitive topology is `VK_PRIMITIVE_TOPOLOGY_LINE_LIST`, each consecutive pair of vertices defines a single line primitive, according to the equation:

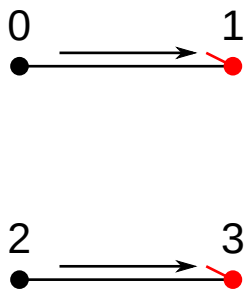
$$p_i = \{v_{2i}, v_{2i+1}\}$$

The number of primitives generated is equal to ⌊`vertexCount`/2⌋.

When the `provokingVertexMode` is `VK_PROVOKING_VERTEX_MODE_FIRST_VERTEX_EXT`, the provoking vertex for $p_i$ is $v_{2i}$.




When the `provokingVertexMode` is `VK_PROVOKING_VERTEX_MODE_LAST_VERTEX_EXT`, the provoking vertex for $p_i$ is $v_{2i+1}$.



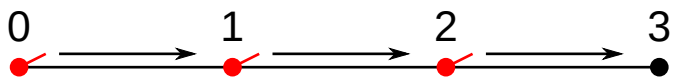

### 21.1.4. Line Strips

When the primitive topology is `VK_PRIMITIVE_TOPOLOGY_LINE_STRIP`, one line primitive is defined by each vertex and the following vertex, according to the equation:

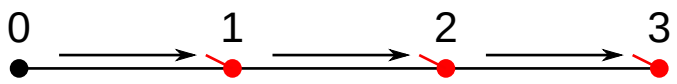
$$p_i = \{v_i, v_{i+1}\}$$

The number of primitives generated is equal to max(0,`vertexCount`-1).

When the `provokingVertexMode` is `VK_PROVOKING_VERTEX_MODE_FIRST_VERTEX_EXT`, the provoking vertex for $p_i$ is $v_i$.




When the `provokingVertexMode` is `VK_PROVOKING_VERTEX_MODE_LAST_VERTEX_EXT`, the provoking vertex for $p_i$ is $v_{i+1}$.



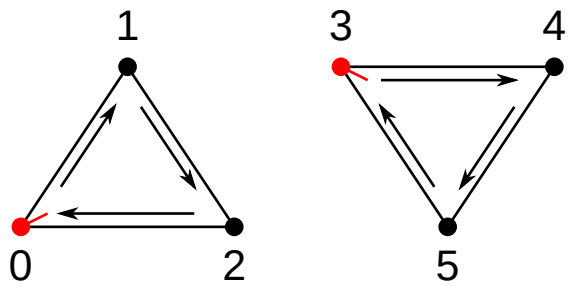
### 21.1.5. Triangle Lists

When the primitive topology is `VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST`, each consecutive set of three vertices defines a single triangle primitive, according to the equation:

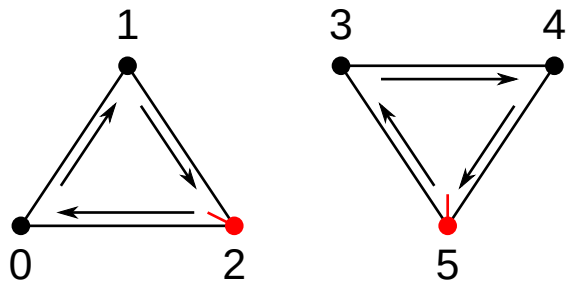
$$p_i = \{v_{3i}, v_{3i+1}, v_{3i+2}\}$$

The number of primitives generated is equal to ⌊`vertexCount`/3⌋.

When the `provokingVertexMode` is `VK_PROVOKING_VERTEX_MODE_FIRST_VERTEX_EXT`, the provoking vertex for $p_i$ is $v_{3i}$.




When the `provokingVertexMode` is `VK_PROVOKING_VERTEX_MODE_LAST_VERTEX_EXT`, the provoking vertex for $p_i$ is $v_{3i+2}$.



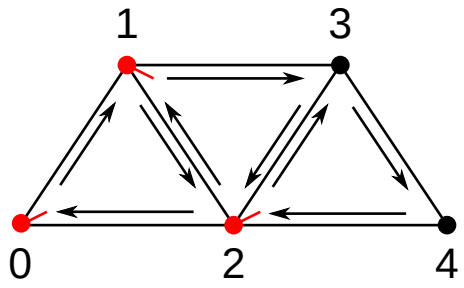

### 21.1.6. Triangle Strips

When the primitive topology is `VK_PRIMITIVE_TOPOLOGY_TRIANGLE_STRIP`, one triangle primitive is defined by each vertex and the two vertices that follow it, according to the equation:

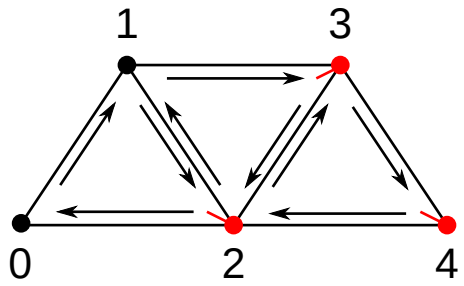
$$p_i = \{v_i, v_{i+(1+i\%2)}, v_{i+(2-i\%2)}\}$$

The number of primitives generated is equal to max(0,`vertexCount`-2).

When the `provokingVertexMode` is `VK_PROVOKING_VERTEX_MODE_FIRST_VERTEX_EXT`, the provoking vertex for $p_i$ is $v_i$.




When the `provokingVertexMode` is `VK_PROVOKING_VERTEX_MODE_LAST_VERTEX_EXT`, the provoking vertex for $p_i$ is $v_{i+2}$.




| | |
|---|---|
| **NOTE** | The ordering of the vertices in each successive triangle is reversed, so that the winding order is consistent throughout the strip. |

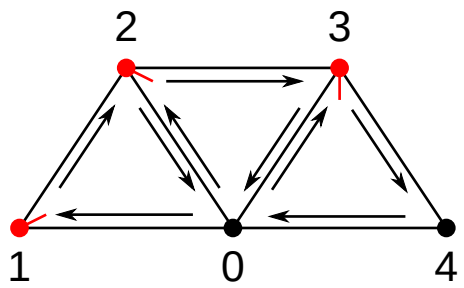
### 21.1.7. Triangle Fans

When the primitive topology is `VK_PRIMITIVE_TOPOLOGY_TRIANGLE_FAN`, triangle primitives are defined around a shared common vertex, according to the equation:

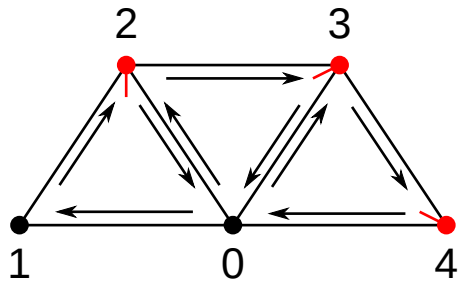
$$p_i = \{v_{i+1}, v_{i+2}, v_0\}$$

The number of primitives generated is equal to max(0,`vertexCount`-2).

When the `provokingVertexMode` is `VK_PROVOKING_VERTEX_MODE_FIRST_VERTEX_EXT`, the provoking vertex for $p_i$ is $v_{i+1}$.




When the `provokingVertexMode` is `VK_PROVOKING_VERTEX_MODE_LAST_VERTEX_EXT`, the provoking vertex for $p_i$ is $v_{i+2}$.




**NOTE** If the `VK_KHR_portability_subset` extension is enabled, and VkPhysicalDevicePortabilitySubsetFeaturesKHR::`triangleFans` is `VK_FALSE`, then triangle fans are not supported by the implementation, and `VK_PRIMITIVE_TOPOLOGY_TRIANGLE_FAN` **must** not be used.

### 21.1.8. Line Lists With Adjacency

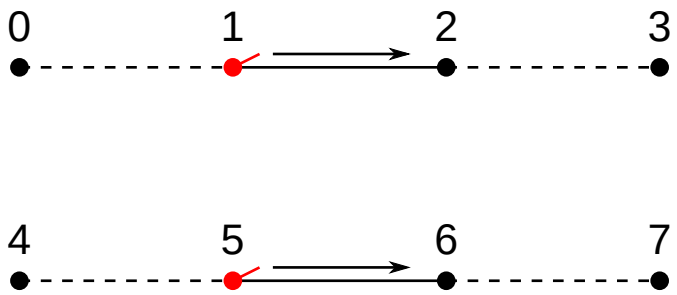
When the primitive topology is `VK_PRIMITIVE_TOPOLOGY_LINE_LIST_WITH_ADJACENCY`, each consecutive set of four vertices defines a single line primitive with adjacency, according to the equation:

$$p_i = \{v_{4i}, v_{4i+1}, v_{4i+2}, v_{4i+3}\}$$

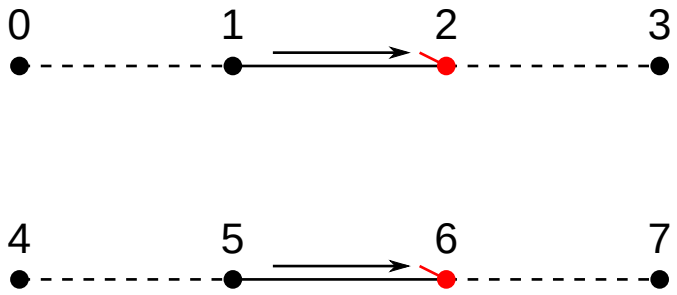
A line primitive is described by the second and third vertices of the total primitive, with the remaining two vertices only accessible in a geometry shader.

The number of primitives generated is equal to ⌊`vertexCount`/4⌋.

When the `provokingVertexMode` is `VK_PROVOKING_VERTEX_MODE_FIRST_VERTEX_EXT`, the provoking vertex for $p_i$ is $v_{4i+1}$.




When the `provokingVertexMode` is `VK_PROVOKING_VERTEX_MODE_LAST_VERTEX_EXT`, the provoking vertex for $p_i$ is $v_{4i+2}$.



### 21.1.9. Line Strips With Adjacency

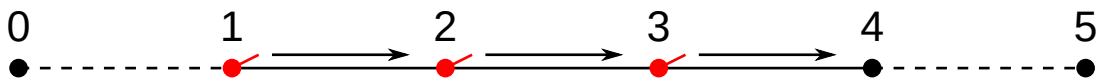
When the primitive topology is `VK_PRIMITIVE_TOPOLOGY_LINE_STRIP_WITH_ADJACENCY`, one line primitive with adjacency is defined by each vertex and the following vertex, according to the equation:

$$p_i = \{v_i, v_{i+1}, v_{i+2}, v_{i+3}\}$$

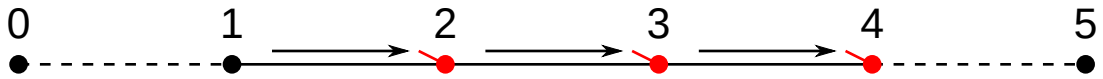
A line primitive is described by the second and third vertices of the total primitive, with the remaining two vertices only accessible in a geometry shader.

The number of primitives generated is equal to max(0,`vertexCount`-3).

When the `provokingVertexMode` is `VK_PROVOKING_VERTEX_MODE_FIRST_VERTEX_EXT`, the provoking vertex for $p_i$ is $v_{i+1}$.




When the `provokingVertexMode` is `VK_PROVOKING_VERTEX_MODE_LAST_VERTEX_EXT`, the provoking vertex for $p_i$ is $v_{i+2}$.




### 21.1.10. Triangle Lists With Adjacency

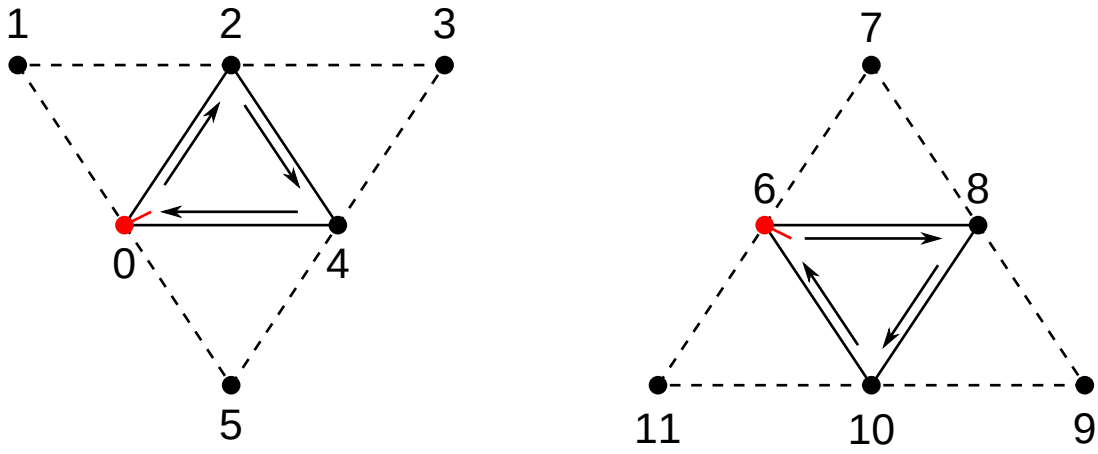
When the primitive topology is `VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST_WITH_ADJACENCY`, each consecutive set of six vertices defines a single triangle primitive with adjacency, according to the equations:

$$p_i = \{v_{6i}, v_{6i+1}, v_{6i+2}, v_{6i+3}, v_{6i+4}, v_{6i+5}\}$$

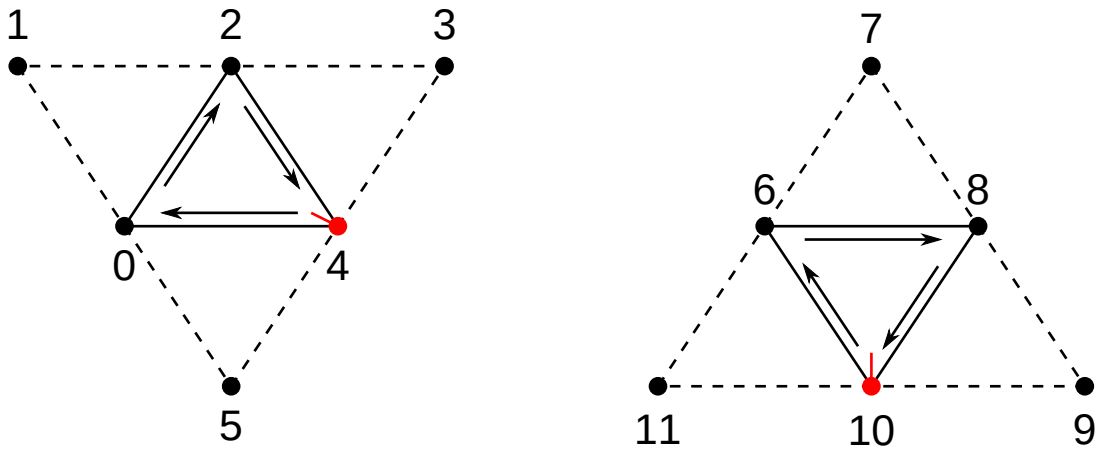
A triangle primitive is described by the first, third, and fifth vertices of the total primitive, with the remaining three vertices only accessible in a geometry shader.

The number of primitives generated is equal to ⌊`vertexCount`/6⌋.

When the `provokingVertexMode` is `VK_PROVOKING_VERTEX_MODE_FIRST_VERTEX_EXT`, the provoking vertex for $p_i$ is $v_{6i}$.




When the `provokingVertexMode` is `VK_PROVOKING_VERTEX_MODE_LAST_VERTEX_EXT`, the provoking vertex for $p_i$ is $v_{6i+4}$.




### 21.1.11. Triangle Strips With Adjacency

When the primitive topology is `VK_PRIMITIVE_TOPOLOGY_TRIANGLE_STRIP_WITH_ADJACENCY`, one triangle primitive with adjacency is defined by each vertex and the following 5 vertices.

The number of primitives generated, n, is equal to ⌊max(0, `vertexCount` - 4)/2⌋.

If n=1, the primitive is defined as:

$$p = \{v_0, v_1, v_2, v_5, v_4, v_3\}$$

If n>1, the total primitive consists of different vertices according to where it is in the strip:

$$p_i = \{v_{2i}, v_{2i+1}, v_{2i+2}, v_{2i+6}, v_{2i+4}, v_{2i+3}\} \text{ when } i=0$$

$$p_i = \{v_{2i}, v_{2i+3}, v_{2i+4}, v_{2i+6}, v_{2i+2}, v_{2i-2}\} \text{ when } i>0, i<n-1, \text{ and } i\%2=1$$

$$p_i = \{v_{2i}, v_{2i-2}, v_{2i+2}, v_{2i+6}, v_{2i+4}, v_{2i+3}\} \text{ when } i>0, i<n-1, \text{ and } i\%2=0$$

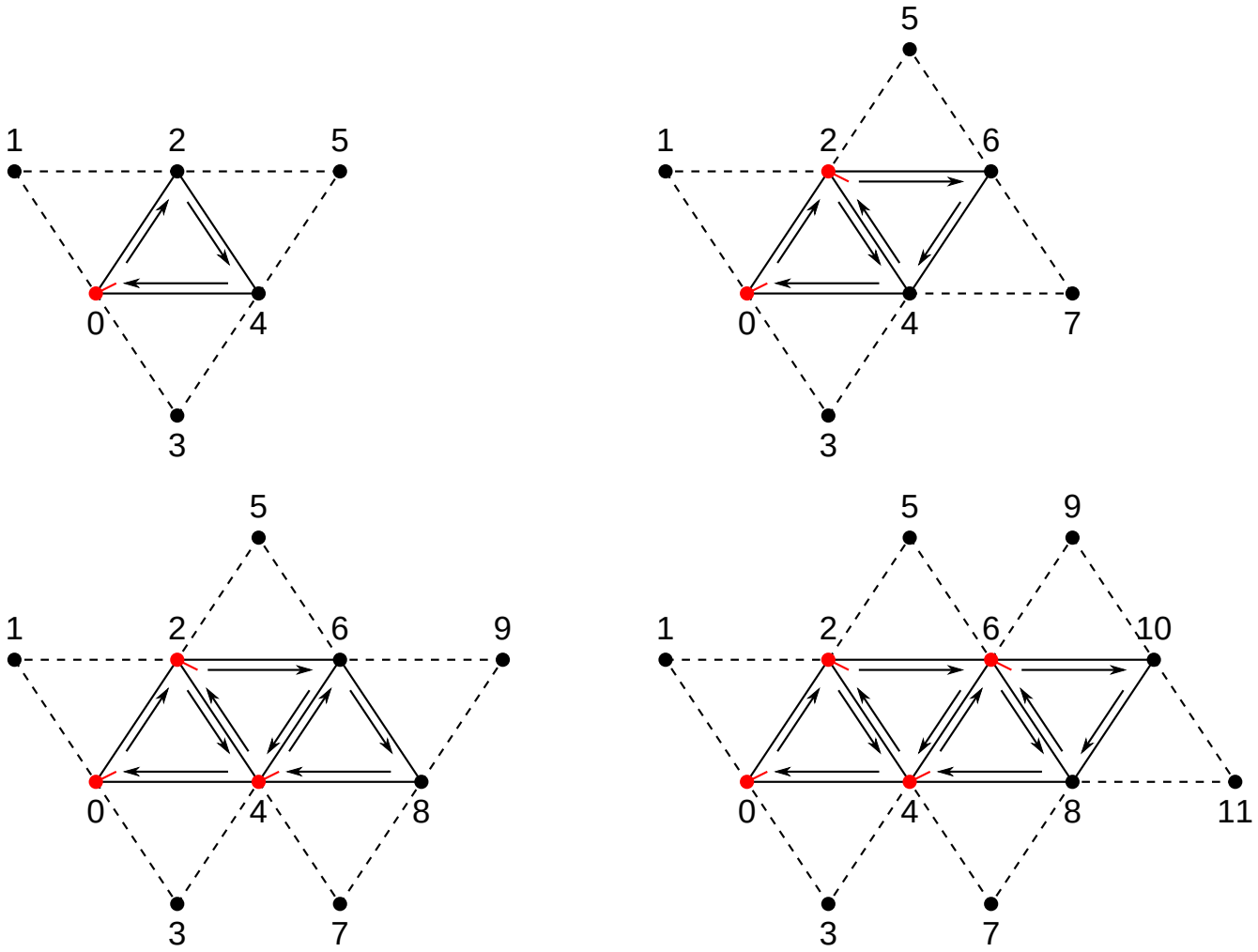
$p_i = \{v_{2i}, v_{2i+3}, v_{2i+4}, v_{2i+5}, v_{2i+2}, v_{2i-2}\}$ when i=n-1 and i%2=1

$p_i = \{v_{2i}, v_{2i-2}, v_{2i+2}, v_{2i+5}, v_{2i+4}, v_{2i+3}\}$ when i=n-1 and i%2=0

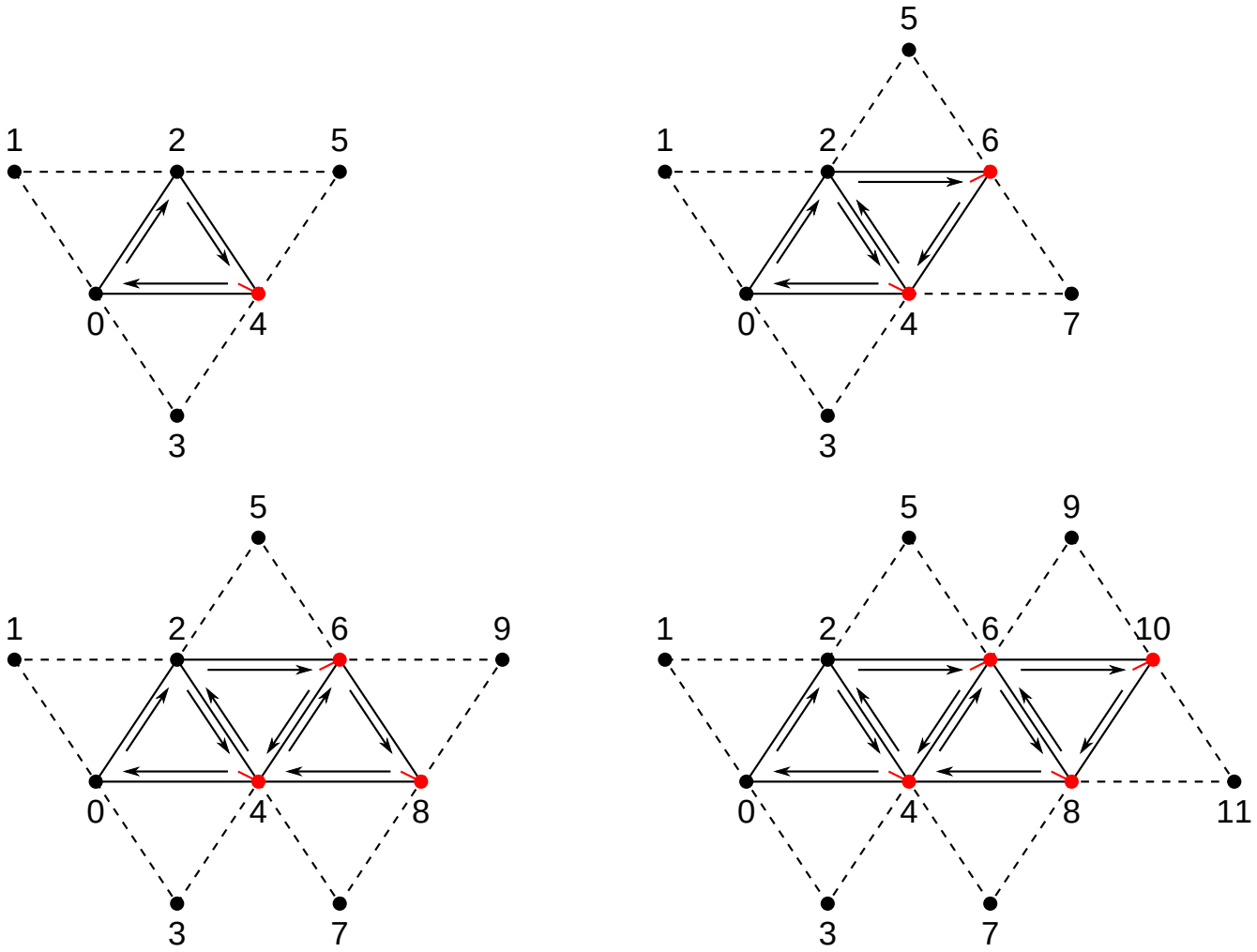
A triangle primitive is described by the first, third, and fifth vertices of the total primitive in all cases, with the remaining three vertices only accessible in a geometry shader.

| | |
|---|---|
| **NOTE** | The ordering of the vertices in each successive triangle is altered so that the winding order is consistent throughout the strip. |

When the `provokingVertexMode` is `VK_PROVOKING_VERTEX_MODE_FIRST_VERTEX_EXT`, the provoking vertex for $p_i$ is always $v_{2i}$.




When the `provokingVertexMode` is `VK_PROVOKING_VERTEX_MODE_LAST_VERTEX_EXT`, the provoking vertex for $p_i$ is always $v_{2i+4}$.




### 21.1.12. Patch Lists

When the primitive topology is `VK_PRIMITIVE_TOPOLOGY_PATCH_LIST`, each consecutive set of m vertices defines a single patch primitive, according to the equation:

$$p_i = \{v_{mi}, v_{mi+1}, \ldots, v_{mi+(m-2)}, v_{mi+(m-1)}\}$$

where m is equal to VkPipelineTessellationStateCreateInfo::`patchControlPoints`.

Patch lists are never passed to vertex post-processing, and as such no provoking vertex is defined for patch primitives. The number of primitives generated is equal to ⌊`vertexCount`/m⌋.

The vertices comprising a patch have no implied geometry, and are used as inputs to tessellation shaders and the fixed-function tessellator to generate new point, line, or triangle primitives.

## 21.2. Primitive Order

Primitives generated by drawing commands progress through the stages of the graphics pipeline in *primitive order*. Primitive order is initially determined in the following way:

1. Submission order determines the initial ordering
2. For indirect drawing commands, the order in which accessed instances of the

VkDrawIndirectCommand are stored in `buffer`, from lower indirect buffer addresses to higher addresses.

3. If a drawing command includes multiple instances, the order in which instances are executed, from lower numbered instances to higher.
4. The order in which primitives are specified by a drawing command:
    - For non-indexed draws, from vertices with a lower numbered `vertexIndex` to a higher numbered `vertexIndex`.
    - For indexed draws, vertices sourced from a lower index buffer addresses to higher addresses.
    - For draws using mesh shaders, the order is provided by mesh shading.
    - For draws using cluster culling shaders, the order is provided by cluster culling shading.

Within this order implementations further sort primitives:

5. If tessellation shading is active, by an implementation-dependent order of new primitives generated by tessellation.
6. If geometry shading is active, by the order new primitives are generated by geometry shading.
7. If the polygon mode is not `VK_POLYGON_MODE_FILL`, or `VK_POLYGON_MODE_FILL_RECTANGLE_NV`, by an implementation-dependent ordering of the new primitives generated within the original primitive.

Primitive order is later used to define rasterization order, which determines the order in which fragments output results to a framebuffer.

## 21.3. Programmable Primitive Shading

Once primitives are assembled, they proceed to the vertex shading stage of the pipeline. If the draw includes multiple instances, then the set of primitives is sent to the vertex shading stage multiple times, once for each instance.

It is implementation-dependent whether vertex shading occurs on vertices that are discarded as part of incomplete primitives, but if it does occur then it operates as if they were vertices in complete primitives and such invocations **can** have side effects.

Vertex shading receives two per-vertex inputs from the primitive assembly stage - the `vertexIndex` and the `instanceIndex`. How these values are generated is defined below, with each command.

Drawing commands fall roughly into two categories:

- Non-indexed drawing commands present a sequential `vertexIndex` to the vertex shader. The sequential index is generated automatically by the device (see Fixed-Function Vertex Processing for details on both specifying the vertex attributes indexed by `vertexIndex`, as well as binding vertex buffers containing those attributes to a command buffer). These commands are:
    - vkCmdDraw
    - vkCmdDrawIndirect

    - vkCmdDrawIndirectCountKHR
    - vkCmdDrawIndirectCountAMD
    - vkCmdDrawMultiEXT
- Indexed drawing commands read index values from an *index buffer* and use this to compute the `vertexIndex` value for the vertex shader. These commands are:
    - vkCmdDrawIndexed
    - vkCmdDrawIndexedIndirect
    - vkCmdDrawIndexedIndirectCountKHR
    - vkCmdDrawIndexedIndirectCountAMD
    - vkCmdDrawMultiIndexedEXT

To bind an index buffer to a command buffer, call:

```
// Provided by VK_VERSION_1_0
void vkCmdBindIndexBuffer(
    VkCommandBuffer                             commandBuffer,
    VkBuffer                                    buffer,
    VkDeviceSize                                offset,
    VkIndexType                                 indexType);
```

- `commandBuffer` is the command buffer into which the command is recorded.
- `buffer` is the buffer being bound.
- `offset` is the starting offset in bytes within `buffer` used in index buffer address calculations.
- `indexType` is a VkIndexType value specifying the size of the indices.

If the `maintenance6` feature is enabled, `buffer` **can** be VK_NULL_HANDLE. If `buffer` is VK_NULL_HANDLE and the `nullDescriptor` feature is enabled, every index fetched results in a value of zero.

### Valid Usage

- VUID-vkCmdBindIndexBuffer-offset-08782
  `offset` **must** be less than the size of `buffer`
- VUID-vkCmdBindIndexBuffer-offset-08783
  The sum of `offset` and the base address of the range of `VkDeviceMemory` object that is backing `buffer`, **must** be a multiple of the size of the type indicated by `indexType`
- VUID-vkCmdBindIndexBuffer-buffer-08784
  `buffer` **must** have been created with the `VK_BUFFER_USAGE_INDEX_BUFFER_BIT` flag
- VUID-vkCmdBindIndexBuffer-buffer-08785
  If `buffer` is non-sparse then it **must** be bound completely and contiguously to a single `VkDeviceMemory` object

- VUID-vkCmdBindIndexBuffer-indexType-08786
  `indexType` **must** not be `VK_INDEX_TYPE_NONE_KHR`
- VUID-vkCmdBindIndexBuffer-indexType-08787
  If `indexType` is `VK_INDEX_TYPE_UINT8_KHR`, the `indexTypeUint8` feature **must** be enabled
- VUID-vkCmdBindIndexBuffer-None-09493
  If `maintenance6` is not enabled, `buffer` **must** not be VK_NULL_HANDLE
- VUID-vkCmdBindIndexBuffer-buffer-09494
  If `buffer` is VK_NULL_HANDLE, offset **must** be zero

## Valid Usage (Implicit)

- VUID-vkCmdBindIndexBuffer-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdBindIndexBuffer-buffer-parameter
  If `buffer` is not VK_NULL_HANDLE, `buffer` **must** be a valid VkBuffer handle
- VUID-vkCmdBindIndexBuffer-indexType-parameter
  `indexType` **must** be a valid VkIndexType value
- VUID-vkCmdBindIndexBuffer-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdBindIndexBuffer-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- VUID-vkCmdBindIndexBuffer-videocoding
  This command **must** only be called outside of a video coding scope
- VUID-vkCmdBindIndexBuffer-commonparent
  Both of `buffer`, and `commandBuffer` that are valid handles of non-ignored parameters **must** have been created, allocated, or retrieved from the same VkDevice

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics | State |

To bind an index buffer, along with its size, to a command buffer, call:

```
// Provided by VK_KHR_maintenance5
void vkCmdBindIndexBuffer2KHR(
    VkCommandBuffer                             commandBuffer,
    VkBuffer                                    buffer,
    VkDeviceSize                                offset,
    VkDeviceSize                                size,
    VkIndexType                                 indexType);
```

- `commandBuffer` is the command buffer into which the command is recorded.
- `buffer` is the buffer being bound.
- `offset` is the starting offset in bytes within `buffer` used in index buffer address calculations.
- `size` is the size in bytes of index data bound from `buffer`.
- `indexType` is a VkIndexType value specifying the size of the indices.

`size` specifies the bound size of the index buffer starting from `offset`. If `size` is `VK_WHOLE_SIZE` then the bound size is from `offset` to the end of the `buffer`.

If the `maintenance6` feature is enabled, `buffer` **can** be VK_NULL_HANDLE. If `buffer` is VK_NULL_HANDLE and the `nullDescriptor` feature is enabled, every index fetched results in a value of zero.

### Valid Usage

- VUID-vkCmdBindIndexBuffer2KHR-offset-08782
  `offset` **must** be less than the size of `buffer`
- VUID-vkCmdBindIndexBuffer2KHR-offset-08783
  The sum of `offset` and the base address of the range of `VkDeviceMemory` object that is backing `buffer`, **must** be a multiple of the size of the type indicated by `indexType`
- VUID-vkCmdBindIndexBuffer2KHR-buffer-08784
  `buffer` **must** have been created with the `VK_BUFFER_USAGE_INDEX_BUFFER_BIT` flag
- VUID-vkCmdBindIndexBuffer2KHR-buffer-08785
  If `buffer` is non-sparse then it **must** be bound completely and contiguously to a single `VkDeviceMemory` object
- VUID-vkCmdBindIndexBuffer2KHR-indexType-08786
  `indexType` **must** not be `VK_INDEX_TYPE_NONE_KHR`
- VUID-vkCmdBindIndexBuffer2KHR-indexType-08787
  If `indexType` is `VK_INDEX_TYPE_UINT8_KHR`, the `indexTypeUint8` feature **must** be enabled
- VUID-vkCmdBindIndexBuffer2KHR-None-09493
  If `maintenance6` is not enabled, `buffer` **must** not be VK_NULL_HANDLE
- VUID-vkCmdBindIndexBuffer2KHR-buffer-09494
  If `buffer` is VK_NULL_HANDLE, offset **must** be zero

- VUID-vkCmdBindIndexBuffer2KHR-size-08767
  If `size` is not `VK_WHOLE_SIZE`, `size` **must** be a multiple of the size of the type indicated by `indexType`
- VUID-vkCmdBindIndexBuffer2KHR-size-08768
  If `size` is not `VK_WHOLE_SIZE`, the sum of `offset` and `size` **must** be less than or equal to the size of `buffer`

## Valid Usage (Implicit)

- VUID-vkCmdBindIndexBuffer2KHR-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdBindIndexBuffer2KHR-buffer-parameter
  If `buffer` is not VK_NULL_HANDLE, `buffer` **must** be a valid VkBuffer handle
- VUID-vkCmdBindIndexBuffer2KHR-indexType-parameter
  `indexType` **must** be a valid VkIndexType value
- VUID-vkCmdBindIndexBuffer2KHR-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdBindIndexBuffer2KHR-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- VUID-vkCmdBindIndexBuffer2KHR-videocoding
  This command **must** only be called outside of a video coding scope
- VUID-vkCmdBindIndexBuffer2KHR-commonparent
  Both of `buffer`, and `commandBuffer` that are valid handles of non-ignored parameters **must** have been created, allocated, or retrieved from the same VkDevice

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics | State |

Possible values of vkCmdBindIndexBuffer2KHR::`indexType` and vkCmdBindIndexBuffer::`indexType`,

specifying the size of indices, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkIndexType {
    VK_INDEX_TYPE_UINT16 = 0,
    VK_INDEX_TYPE_UINT32 = 1,
  // Provided by VK_KHR_acceleration_structure
    VK_INDEX_TYPE_NONE_KHR = 1000165000,
  // Provided by VK_KHR_index_type_uint8
    VK_INDEX_TYPE_UINT8_KHR = 1000265000,
  // Provided by VK_NV_ray_tracing
    VK_INDEX_TYPE_NONE_NV = VK_INDEX_TYPE_NONE_KHR,
  // Provided by VK_EXT_index_type_uint8
    VK_INDEX_TYPE_UINT8_EXT = VK_INDEX_TYPE_UINT8_KHR,
} VkIndexType;
```

- `VK_INDEX_TYPE_UINT16` specifies that indices are 16-bit unsigned integer values.
- `VK_INDEX_TYPE_UINT32` specifies that indices are 32-bit unsigned integer values.
- `VK_INDEX_TYPE_NONE_KHR` specifies that no indices are provided.
- `VK_INDEX_TYPE_UINT8_KHR` specifies that indices are 8-bit unsigned integer values.

The parameters for each drawing command are specified directly in the command or read from buffer memory, depending on the command. Drawing commands that source their parameters from buffer memory are known as *indirect* drawing commands.

All drawing commands interact with the `robustBufferAccess` feature.

To record a non-indexed draw, call:

```
// Provided by VK_VERSION_1_0
void vkCmdDraw(
    VkCommandBuffer                             commandBuffer,
    uint32_t                                    vertexCount,
    uint32_t                                    instanceCount,
    uint32_t                                    firstVertex,
    uint32_t                                    firstInstance);
```

- `commandBuffer` is the command buffer into which the command is recorded.
- `vertexCount` is the number of vertices to draw.
- `instanceCount` is the number of instances to draw.
- `firstVertex` is the index of the first vertex to draw.
- `firstInstance` is the instance ID of the first instance to draw.

When the command is executed, primitives are assembled using the current primitive topology and `vertexCount` consecutive vertex indices with the first `vertexIndex` value equal to `firstVertex`. The

primitives are drawn instanceCount times with instanceIndex starting with firstInstance and increasing sequentially for each instance. The assembled primitives execute the bound graphics pipeline.

### Valid Usage

- VUID-vkCmdDraw-magFilter-04553
  If a VkSampler created with magFilter or minFilter equal to VK_FILTER_LINEAR, reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE, and compareEnable equal to VK_FALSE is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT
- VUID-vkCmdDraw-magFilter-09598
  If a VkSampler created with magFilter or minFilter equal to VK_FILTER_LINEAR and reductionMode equal to either VK_SAMPLER_REDUCTION_MODE_MIN or VK_SAMPLER_REDUCTION_MODE_MAX is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT
- VUID-vkCmdDraw-mipmapMode-04770
  If a VkSampler created with mipmapMode equal to VK_SAMPLER_MIPMAP_MODE_LINEAR, reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE, and compareEnable equal to VK_FALSE is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT
- VUID-vkCmdDraw-mipmapMode-09599
  If a VkSampler created with mipmapMode equal to VK_SAMPLER_MIPMAP_MODE_LINEAR and reductionMode equal to either VK_SAMPLER_REDUCTION_MODE_MIN or VK_SAMPLER_REDUCTION_MODE_MAX is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT
- VUID-vkCmdDraw-unnormalizedCoordinates-09635
  If a VkSampler created with unnormalizedCoordinates equal to VK_TRUE is used to sample a VkImageView as a result of this command, then the image view's levelCount and layerCount **must** be 1
- VUID-vkCmdDraw-unnormalizedCoordinates-09636
  If a VkSampler created with unnormalizedCoordinates equal to VK_TRUE is used to sample a VkImageView as a result of this command, then the image view's viewType **must** be VK_IMAGE_VIEW_TYPE_1D or VK_IMAGE_VIEW_TYPE_2D
- VUID-vkCmdDraw-None-06479
  If a VkImageView is sampled with depth comparison, the image view's format features **must** contain VK_FORMAT_FEATURE_2_SAMPLED_IMAGE_DEPTH_COMPARISON_BIT
- VUID-vkCmdDraw-None-02691
  If a VkImageView is accessed using atomic operations as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_STORAGE_IMAGE_ATOMIC_BIT

- VUID-vkCmdDraw-None-07888
  If a VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER descriptor is accessed using atomic operations as a result of this command, then the storage texel buffer's format features **must** contain VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_ATOMIC_BIT
- VUID-vkCmdDraw-None-02692
  If a VkImageView is sampled with VK_FILTER_CUBIC_EXT as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_CUBIC_BIT_EXT
- VUID-vkCmdDraw-None-02693
  If the VK_EXT_filter_cubic extension is not enabled and any VkImageView is sampled with VK_FILTER_CUBIC_EXT as a result of this command, it **must** not have a VkImageViewType of VK_IMAGE_VIEW_TYPE_3D, VK_IMAGE_VIEW_TYPE_CUBE, or VK_IMAGE_VIEW_TYPE_CUBE_ARRAY
- VUID-vkCmdDraw-filterCubic-02694
  Any VkImageView being sampled with VK_FILTER_CUBIC_EXT as a result of this command **must** have a VkImageViewType and format that supports cubic filtering, as specified by VkFilterCubicImageViewImageFormatPropertiesEXT::filterCubic returned by vkGetPhysicalDeviceImageFormatProperties2
- VUID-vkCmdDraw-filterCubicMinmax-02695
  Any VkImageView being sampled with VK_FILTER_CUBIC_EXT with a reduction mode of either VK_SAMPLER_REDUCTION_MODE_MIN or VK_SAMPLER_REDUCTION_MODE_MAX as a result of this command **must** have a VkImageViewType and format that supports cubic filtering together with minmax filtering, as specified by VkFilterCubicImageViewImageFormatPropertiesEXT::filterCubicMinmax returned by vkGetPhysicalDeviceImageFormatProperties2
- VUID-vkCmdDraw-cubicRangeClamp-09212
  If the cubicRangeClamp feature is not enabled, then any VkImageView being sampled with VK_FILTER_CUBIC_EXT as a result of this command **must** not have a VkSamplerReductionModeCreateInfo::reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE_RANGECLAMP_QCOM
- VUID-vkCmdDraw-reductionMode-09213
  Any VkImageView being sampled with a VkSamplerReductionModeCreateInfo ::reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE_RANGECLAMP_QCOM as a result of this command **must** sample with VK_FILTER_CUBIC_EXT
- VUID-vkCmdDraw-selectableCubicWeights-09214
  If the selectableCubicWeights feature is not enabled, then any VkImageView being sampled with VK_FILTER_CUBIC_EXT as a result of this command **must** have VkSamplerCubicWeightsCreateInfoQCOM::cubicWeights equal to VK_CUBIC_FILTER_WEIGHTS_CATMULL_ROM_QCOM
- VUID-vkCmdDraw-flags-02696
  Any VkImage created with a VkImageCreateInfo::flags containing VK_IMAGE_CREATE_CORNER_SAMPLED_BIT_NV sampled as a result of this command **must** only be sampled using a VkSamplerAddressMode of VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_EDGE
- VUID-vkCmdDraw-OpTypeImage-07027
  For any VkImageView being written as a storage image where the image format field of

the OpTypeImage is Unknown, the view's format features **must** contain VK_FORMAT_FEATURE_2_STORAGE_WRITE_WITHOUT_FORMAT_BIT

- VUID-vkCmdDraw-OpTypeImage-07028
  For any VkImageView being read as a storage image where the image format field of the OpTypeImage is Unknown, the view's format features **must** contain VK_FORMAT_FEATURE_2_STORAGE_READ_WITHOUT_FORMAT_BIT
- VUID-vkCmdDraw-OpTypeImage-07029
  For any VkBufferView being written as a storage texel buffer where the image format field of the OpTypeImage is Unknown, the view's buffer features **must** contain VK_FORMAT_FEATURE_2_STORAGE_WRITE_WITHOUT_FORMAT_BIT
- VUID-vkCmdDraw-OpTypeImage-07030
  Any VkBufferView being read as a storage texel buffer where the image format field of the OpTypeImage is Unknown then the view's buffer features **must** contain VK_FORMAT_FEATURE_2_STORAGE_READ_WITHOUT_FORMAT_BIT
- VUID-vkCmdDraw-None-08600
  For each set *n* that is statically used by a bound shader, a descriptor set **must** have been bound to *n* at the same pipeline bind point, with a VkPipelineLayout that is compatible for set *n*, with the VkPipelineLayout used to create the current VkPipeline or the VkDescriptorSetLayout array used to create the current VkShaderEXT , as described in Pipeline Layout Compatibility
- VUID-vkCmdDraw-None-08601
  For each push constant that is statically used by a bound shader, a push constant value **must** have been set for the same pipeline bind point, with a VkPipelineLayout that is compatible for push constants, with the VkPipelineLayout used to create the current VkPipeline or the VkDescriptorSetLayout array used to create the current VkShaderEXT , as described in Pipeline Layout Compatibility
- VUID-vkCmdDraw-None-10068
  For each array of resources that is used by a bound shader, the indices used to access members of the array **must** be less than the descriptor count for the identified binding in the descriptor sets used by this command
- VUID-vkCmdDraw-maintenance4-08602
  If the maintenance4 feature is not enabled, then for each push constant that is statically used by a bound shader, a push constant value **must** have been set for the same pipeline bind point, with a VkPipelineLayout that is compatible for push constants, with the VkPipelineLayout used to create the current VkPipeline or the VkDescriptorSetLayout and VkPushConstantRange arrays used to create the current VkShaderEXT , as described in Pipeline Layout Compatibility
- VUID-vkCmdDraw-None-08114
  Descriptors in each bound descriptor set, specified via vkCmdBindDescriptorSets, **must** be valid as described by descriptor validity if they are statically used by the VkPipeline bound to the pipeline bind point used by this command and the bound VkPipeline was not created with VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT
- VUID-vkCmdDraw-None-08115
  If the descriptors used by the VkPipeline bound to the pipeline bind point were specified

via vkCmdBindDescriptorSets, the bound VkPipeline **must** have been created without VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT

- VUID-vkCmdDraw-None-08116
  Descriptors in bound descriptor buffers, specified via vkCmdSetDescriptorBufferOffsetsEXT, **must** be valid if they are dynamically used by the VkPipeline bound to the pipeline bind point used by this command and the bound VkPipeline was created with VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT
- VUID-vkCmdDraw-None-08604
  Descriptors in bound descriptor buffers, specified via vkCmdSetDescriptorBufferOffsetsEXT, **must** be valid if they are dynamically used by any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command
- VUID-vkCmdDraw-None-08117
  If the descriptors used by the VkPipeline bound to the pipeline bind point were specified via vkCmdSetDescriptorBufferOffsetsEXT, the bound VkPipeline **must** have been created with VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT
- VUID-vkCmdDraw-None-08119
  If a descriptor is dynamically used with a VkPipeline created with VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT, the descriptor memory **must** be resident
- VUID-vkCmdDraw-None-08605
  If a descriptor is dynamically used with a VkShaderEXT created with a VkDescriptorSetLayout that was created with VK_DESCRIPTOR_SET_LAYOUT_CREATE_DESCRIPTOR_BUFFER_BIT_EXT, the descriptor memory **must** be resident
- VUID-vkCmdDraw-None-08606
  If the shaderObject feature is not enabled, a valid pipeline **must** be bound to the pipeline bind point used by this command
- VUID-vkCmdDraw-None-08608
  If a pipeline is bound to the pipeline bind point used by this command, there **must** not have been any calls to dynamic state setting commands for any state specified statically in the VkPipeline object bound to the pipeline bind point used by this command, since that pipeline was bound
- VUID-vkCmdDraw-None-08609
  If the VkPipeline object bound to the pipeline bind point used by this command or any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a VkSampler object that uses unnormalized coordinates, that sampler **must** not be used to sample from any VkImage with a VkImageView of the type VK_IMAGE_VIEW_TYPE_3D, VK_IMAGE_VIEW_TYPE_CUBE, VK_IMAGE_VIEW_TYPE_1D_ARRAY, VK_IMAGE_VIEW_TYPE_2D_ARRAY or VK_IMAGE_VIEW_TYPE_CUBE_ARRAY, in any shader stage
- VUID-vkCmdDraw-None-08610
  If the VkPipeline object bound to the pipeline bind point used by this command or any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a VkSampler object that uses unnormalized coordinates, that sampler **must** not be used with any of the SPIR-V OpImageSample* or OpImageSparseSample*

instructions with `ImplicitLod`, `Dref` or `Proj` in their name, in any shader stage

- VUID-vkCmdDraw-None-08611
  If the VkPipeline object bound to the pipeline bind point used by this command or any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a VkSampler object that uses unnormalized coordinates, that sampler **must** not be used with any of the SPIR-V `OpImageSample*` or `OpImageSparseSample*` instructions that includes a LOD bias or any offset values, in any shader stage

- VUID-vkCmdDraw-None-08607
  If the `shaderObject` is enabled, either a valid pipeline **must** be bound to the pipeline bind point used by this command, or a valid combination of valid and VK_NULL_HANDLE shader objects **must** be bound to every supported shader stage corresponding to the pipeline bind point used by this command

- VUID-vkCmdDraw-uniformBuffers-06935
  If any stage of the VkPipeline object bound to the pipeline bind point used by this command accesses a uniform buffer, and that stage was created without enabling either `VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_EXT` or `VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_2_EXT` for `uniformBuffers`, and the `robustBufferAccess` feature is not enabled, that stage **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point

- VUID-vkCmdDraw-None-08612
  If the `robustBufferAccess` feature is not enabled, and any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a uniform buffer, it **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point

- VUID-vkCmdDraw-storageBuffers-06936
  If any stage of the VkPipeline object bound to the pipeline bind point used by this command accesses a storage buffer, and that stage was created without enabling either `VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_EXT` or `VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_2_EXT` for `storageBuffers`, and the `robustBufferAccess` feature is not enabled, that stage **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point

- VUID-vkCmdDraw-None-08613
  If the `robustBufferAccess` feature is not enabled, and any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a storage buffer, it **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point

- VUID-vkCmdDraw-commandBuffer-02707
  If `commandBuffer` is an unprotected command buffer and `protectedNoFault` is not supported, any resource accessed by bound shaders **must** not be a protected resource

- VUID-vkCmdDraw-None-06550
  If a bound shader accesses a VkSampler or VkImageView object that enables sampler $Y'C_BC_R$ conversion, that object **must** only be used with `OpImageSample*` or `OpImageSparseSample*` instructions

- VUID-vkCmdDraw-ConstOffset-06551
  If a bound shader accesses a VkSampler or VkImageView object that enables sampler $Y'C_BC_R$ conversion, that object **must** not use the `ConstOffset` and `Offset` operands
- VUID-vkCmdDraw-viewType-07752
  If a VkImageView is accessed as a result of this command, then the image view's `viewType` **must** match the `Dim` operand of the `OpTypeImage` as described in Compatibility Between SPIR-V Image Dimensions and Vulkan ImageView Types
- VUID-vkCmdDraw-format-07753
  If a VkImageView is accessed as a result of this command, then the numeric type of the image view's `format` and the `Sampled Type` operand of the `OpTypeImage` **must** match
- VUID-vkCmdDraw-OpImageWrite-08795
  If a VkImageView created with a format other than `VK_FORMAT_A8_UNORM_KHR` is accessed using `OpImageWrite` as a result of this command, then the `Type` of the `Texel` operand of that instruction **must** have at least as many components as the image view's format
- VUID-vkCmdDraw-OpImageWrite-08796
  If a VkImageView created with the format `VK_FORMAT_A8_UNORM_KHR` is accessed using `OpImageWrite` as a result of this command, then the `Type` of the `Texel` operand of that instruction **must** have four components
- VUID-vkCmdDraw-OpImageWrite-04469
  If a VkBufferView is accessed using `OpImageWrite` as a result of this command, then the `Type` of the `Texel` operand of that instruction **must** have at least as many components as the buffer view's format
- VUID-vkCmdDraw-SampledType-04470
  If a VkImageView with a VkFormat that has a 64-bit component width is accessed as a result of this command, the `SampledType` of the `OpTypeImage` operand of that instruction **must** have a `Width` of 64
- VUID-vkCmdDraw-SampledType-04471
  If a VkImageView with a VkFormat that has a component width less than 64-bit is accessed as a result of this command, the `SampledType` of the `OpTypeImage` operand of that instruction **must** have a `Width` of 32
- VUID-vkCmdDraw-SampledType-04472
  If a VkBufferView with a VkFormat that has a 64-bit component width is accessed as a result of this command, the `SampledType` of the `OpTypeImage` operand of that instruction **must** have a `Width` of 64
- VUID-vkCmdDraw-SampledType-04473
  If a VkBufferView with a VkFormat that has a component width less than 64-bit is accessed as a result of this command, the `SampledType` of the `OpTypeImage` operand of that instruction **must** have a `Width` of 32
- VUID-vkCmdDraw-sparseImageInt64Atomics-04474
  If the `sparseImageInt64Atomics` feature is not enabled, VkImage objects created with the `VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT` flag **must** not be accessed by atomic instructions through an `OpTypeImage` with a `SampledType` with a `Width` of 64 by this command
- VUID-vkCmdDraw-sparseImageInt64Atomics-04475

If the sparseImageInt64Atomics feature is not enabled, VkBuffer objects created with the VK_BUFFER_CREATE_SPARSE_RESIDENCY_BIT flag **must** not be accessed by atomic instructions through an OpTypeImage with a SampledType with a Width of 64 by this command

- VUID-vkCmdDraw-OpImageWeightedSampleQCOM-06971
  If OpImageWeightedSampleQCOM is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_WEIGHT_SAMPLED_IMAGE_BIT_QCOM
- VUID-vkCmdDraw-OpImageWeightedSampleQCOM-06972
  If OpImageWeightedSampleQCOM uses a VkImageView as a sample weight image as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_WEIGHT_IMAGE_BIT_QCOM
- VUID-vkCmdDraw-OpImageBoxFilterQCOM-06973
  If OpImageBoxFilterQCOM is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_BOX_FILTER_SAMPLED_BIT_QCOM
- VUID-vkCmdDraw-OpImageBlockMatchSSDQCOM-06974
  If OpImageBlockMatchSSDQCOM is used to read from an VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM
- VUID-vkCmdDraw-OpImageBlockMatchSADQCOM-06975
  If OpImageBlockMatchSADQCOM is used to read from an VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM
- VUID-vkCmdDraw-OpImageBlockMatchSADQCOM-06976
  If OpImageBlockMatchSADQCOM or OpImageBlockMatchSSDQCOM is used to read from a reference image as result of this command, then the specified reference coordinates **must** not fail integer texel coordinate validation
- VUID-vkCmdDraw-OpImageWeightedSampleQCOM-06977
  If OpImageWeightedSampleQCOM, OpImageBoxFilterQCOM, OpImageBlockMatchWindowSSDQCOM, OpImageBlockMatchWindowSADQCOM, OpImageBlockMatchGatherSSDQCOM, OpImageBlockMatchGatherSADQCOM, OpImageBlockMatchSSDQCOM, or OpImageBlockMatchSADQCOM uses a VkSampler as a result of this command, then the sampler **must** have been created with VK_SAMPLER_CREATE_IMAGE_PROCESSING_BIT_QCOM
- VUID-vkCmdDraw-OpImageWeightedSampleQCOM-06978
  If any command other than OpImageWeightedSampleQCOM, OpImageBoxFilterQCOM, OpImageBlockMatchWindowSSDQCOM, OpImageBlockMatchWindowSADQCOM, OpImageBlockMatchGatherSSDQCOM, OpImageBlockMatchGatherSADQCOM, OpImageBlockMatchSSDQCOM, or OpImageBlockMatchSADQCOM uses a VkSampler as a result of this command, then the sampler **must** not have been created with VK_SAMPLER_CREATE_IMAGE_PROCESSING_BIT_QCOM
- VUID-vkCmdDraw-OpImageBlockMatchWindow-09215
  If a OpImageBlockMatchWindow*QCOM or OpImageBlockMatchGather*QCOM instruction is used to read from an VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM

- VUID-vkCmdDraw-OpImageBlockMatchWindow-09216
  If a `OpImageBlockMatchWindow*QCOM` or `OpImageBlockMatchGather*QCOM` instruction is used to read from an VkImageView as a result of this command, then the image view's format **must** be a single-component format
- VUID-vkCmdDraw-OpImageBlockMatchWindow-09217
  If a `OpImageBlockMatchWindow*QCOM` or `OpImageBlockMatchGather*QCOM` read from a reference image as result of this command, then the specified reference coordinates **must** not fail integer texel coordinate validation
- VUID-vkCmdDraw-None-07288
  Any shader invocation executed by this command **must** terminate
- VUID-vkCmdDraw-None-09600
  If a descriptor with type equal to any of `VK_DESCRIPTOR_TYPE_SAMPLE_WEIGHT_IMAGE_QCOM`, `VK_DESCRIPTOR_TYPE_BLOCK_MATCH_IMAGE_QCOM`, `VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE`, `VK_DESCRIPTOR_TYPE_STORAGE_IMAGE`, or `VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT` is accessed as a result of this command, the image subresource identified by that descriptor **must** be in the image layout identified when the descriptor was written
- VUID-vkCmdDraw-renderPass-02684
  The current render pass **must** be compatible with the `renderPass` member of the `VkGraphicsPipelineCreateInfo` structure specified when creating the `VkPipeline` bound to `VK_PIPELINE_BIND_POINT_GRAPHICS`
- VUID-vkCmdDraw-subpass-02685
  The subpass index of the current render pass **must** be equal to the `subpass` member of the `VkGraphicsPipelineCreateInfo` structure specified when creating the `VkPipeline` bound to `VK_PIPELINE_BIND_POINT_GRAPHICS`
- VUID-vkCmdDraw-None-07748
  If any shader statically accesses an input attachment, a valid descriptor **must** be bound to the pipeline via a descriptor set
- VUID-vkCmdDraw-OpTypeImage-07468
  If any shader executed by this pipeline accesses an `OpTypeImage` variable with a `Dim` operand of `SubpassData`, it **must** be decorated with an `InputAttachmentIndex` that corresponds to a valid input attachment in the current subpass
- VUID-vkCmdDraw-None-07469
  Input attachment views accessed in a subpass **must** be created with the same VkFormat as the corresponding subpass definition, and be created with a VkImageView that is compatible with the attachment referenced by the subpass' `pInputAttachments` [`InputAttachmentIndex`] in the currently bound VkFramebuffer as specified by Fragment Input Attachment Compatibility
- VUID-vkCmdDraw-pDepthInputAttachmentIndex-09595
  Input attachment views accessed in a dynamic render pass with a `InputAttachmentIndex` referenced by VkRenderingInputAttachmentIndexInfoKHR, or no `InputAttachmentIndex` if VkRenderingInputAttachmentIndexInfoKHR:`pDepthInputAttachmentIndex` or VkRenderingInputAttachmentIndexInfoKHR:`pStencilInputAttachmentIndex` are `NULL`, **must** be created with a VkImageView that is compatible with the corresponding color, depth, or stencil attachment in VkRenderingInfo

- VUID-vkCmdDraw-pDepthInputAttachmentIndex-09596
  Input attachment views accessed in a dynamic render pass via a shader object **must** have an `InputAttachmentIndex` if both VkRenderingInputAttachmentIndexInfoKHR:`pDepthInputAttachmentIndex` and VkRenderingInputAttachmentIndexInfoKHR:`pStencilInputAttachmentIndex` are non-`NULL`
- VUID-vkCmdDraw-InputAttachmentIndex-09597
  If an input attachment view accessed in a dynamic render pass via a shader object has an `InputAttachmentIndex`, the `InputAttachmentIndex` **must** match an index in VkRenderingInputAttachmentIndexInfoKHR
- VUID-vkCmdDraw-None-06537
  Memory backing image subresources used as attachments in the current render pass **must** not be written in any way other than as an attachment by this command
- VUID-vkCmdDraw-None-09000
  If a color attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass, it is not in the `VK_IMAGE_LAYOUT_ATTACHMENT_FEEDBACK_LOOP_OPTIMAL_EXT` image layout, and either:
  - the `VK_PIPELINE_CREATE_COLOR_ATTACHMENT_FEEDBACK_LOOP_BIT_EXT` is set on the currently bound pipeline or
  - the last call to vkCmdSetAttachmentFeedbackLoopEnableEXT included `VK_IMAGE_ASPECT_COLOR_BIT` and
    - there is no currently bound graphics pipeline or
    - the currently bound graphics pipeline was created with `VK_DYNAMIC_STATE_ATTACHMENT_FEEDBACK_LOOP_ENABLE_EXT`

  it **must** not be accessed in any way other than as an attachment by this command
- VUID-vkCmdDraw-None-09001
  If a depth attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass, it is not in the `VK_IMAGE_LAYOUT_ATTACHMENT_FEEDBACK_LOOP_OPTIMAL_EXT` image layout, and either:
  - the `VK_PIPELINE_CREATE_DEPTH_STENCIL_ATTACHMENT_FEEDBACK_LOOP_BIT_EXT` is set on the currently bound pipeline or
  - the last call to vkCmdSetAttachmentFeedbackLoopEnableEXT included `VK_IMAGE_ASPECT_DEPTH_BIT` and
    - there is no currently bound graphics pipeline or
    - the currently bound graphics pipeline was created with `VK_DYNAMIC_STATE_ATTACHMENT_FEEDBACK_LOOP_ENABLE_EXT`

  it **must** not be accessed in any way other than as an attachment by this command
- VUID-vkCmdDraw-None-09002
  If a stencil attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass, it is not in the `VK_IMAGE_LAYOUT_ATTACHMENT_FEEDBACK_LOOP_OPTIMAL_EXT` image layout, and either:

- the VK_PIPELINE_CREATE_DEPTH_STENCIL_ATTACHMENT_FEEDBACK_LOOP_BIT_EXT is set on the currently bound pipeline or
- the last call to vkCmdSetAttachmentFeedbackLoopEnableEXT included VK_IMAGE_ASPECT_STENCIL_BIT and
  - there is no currently bound graphics pipeline or
  - the currently bound graphics pipeline was created with VK_DYNAMIC_STATE_ATTACHMENT_FEEDBACK_LOOP_ENABLE_EXT

it **must** not be accessed in any way other than as an attachment by this command

- VUID-vkCmdDraw-None-09003
  If an attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass, it **must** not be accessed in any way other than as an attachment, storage image, or sampled image by this command
- VUID-vkCmdDraw-None-06539
  If any previously recorded command in the current subpass accessed an image subresource used as an attachment in this subpass in any way other than as an attachment, this command **must** not write to that image subresource as an attachment
- VUID-vkCmdDraw-None-06886
  If the current render pass instance uses a depth/stencil attachment with a read-only layout for the depth aspect, depth writes **must** be disabled
- VUID-vkCmdDraw-None-06887
  If the current render pass instance uses a depth/stencil attachment with a read-only layout for the stencil aspect, both front and back writeMask are not zero, and stencil test is enabled, all stencil ops **must** be VK_STENCIL_OP_KEEP
- VUID-vkCmdDraw-None-07831
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT dynamic state enabled then vkCmdSetViewport **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDraw-None-07832
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SCISSOR dynamic state enabled then vkCmdSetScissor **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDraw-None-07833
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_WIDTH dynamic state enabled then vkCmdSetLineWidth **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDraw-None-08617
  If a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetPolygonModeEXT in the current command buffer set polygonMode to VK_POLYGON_MODE_LINE, vkCmdSetLineWidth **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDraw-None-08618
  If a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetPrimitiveTopologyEXT in the current command buffer set primitiveTopology to any line topology, vkCmdSetLineWidth **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDraw-None-08619
  If a shader object that outputs line primitives is bound to the VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT or VK_SHADER_STAGE_GEOMETRY_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, vkCmdSetLineWidth **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDraw-None-07834
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_BIAS dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of depthBiasEnable is VK_TRUE, then vkCmdSetDepthBounds or vkCmdSetDepthBias2EXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDraw-None-07835
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_BLEND_CONSTANTS dynamic state enabled then vkCmdSetBlendConstants **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDraw-None-08621
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetColorBlendEnableEXT in the current command buffer set any element of pColorBlendEnables to VK_TRUE, and the most recent call to vkCmdSetColorBlendEquationEXT in the current command buffer set the same element of pColorBlendEquations to a VkColorBlendEquationEXT structure with any VkBlendFactor member with a value of VK_BLEND_FACTOR_CONSTANT_COLOR, VK_BLEND_FACTOR_ONE_MINUS_CONSTANT_COLOR, VK_BLEND_FACTOR_CONSTANT_ALPHA, or VK_BLEND_FACTOR_ONE_MINUS_CONSTANT_ALPHA, vkCmdSetBlendConstants **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDraw-None-07836
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_BOUNDS dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of depthBoundsTestEnable is VK_TRUE, then vkCmdSetDepthBounds **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDraw-None-07837
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_STENCIL_COMPARE_MASK dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of stencilTestEnable is VK_TRUE, then vkCmdSetStencilCompareMask **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDraw-None-07838
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_STENCIL_WRITE_MASK dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of stencilTestEnable is VK_TRUE, then vkCmdSetStencilWriteMask **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDraw-None-07839
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_STENCIL_REFERENCE dynamic state enabled, the current value of and rasterizerDiscardEnable is VK_FALSE, the current value of stencilTestEnable is VK_TRUE, then vkCmdSetStencilReference **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDraw-maxMultiviewInstanceIndex-02688
  If the draw is recorded in a render pass instance with multiview enabled, the maximum instance index **must** be less than or equal to VkPhysicalDeviceMultiviewProperties::maxMultiviewInstanceIndex
- VUID-vkCmdDraw-sampleLocationsEnable-02689
  If the bound graphics pipeline was created with VkPipelineSampleLocationsStateCreateInfoEXT::sampleLocationsEnable set to VK_TRUE and the current subpass has a depth/stencil attachment, then that attachment **must** have been created with the VK_IMAGE_CREATE_SAMPLE_LOCATIONS_COMPATIBLE_DEPTH_BIT_EXT bit set
- VUID-vkCmdDraw-None-06666
  If the VK_EXT_sample_locations extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of sampleLocationsEnable is VK_TRUE, then vkCmdSetSampleLocationsEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDraw-None-07840
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_CULL_MODE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetCullModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDraw-None-07841
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which

was created with the VK_DYNAMIC_STATE_FRONT_FACE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetFrontFaceEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDraw-None-07843
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_TEST_ENABLE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, vkCmdSetDepthTestEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDraw-None-07844
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_WRITE_ENABLE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetDepthWriteEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDraw-None-07845
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_COMPARE_OP dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of depthTestEnable is VK_TRUE, then vkCmdSetDepthCompareOpEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDraw-None-07846
  If the depthBounds feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_BOUNDS_TEST_ENABLE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetDepthBoundsTestEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDraw-None-07847
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_STENCIL_TEST_ENABLE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetStencilTestEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDraw-None-07848
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_STENCIL_OP dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, the current value of stencilTestEnable is VK_TRUE, then vkCmdSetStencilOpEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDraw-viewportCount-03417
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, but not the

VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT dynamic state enabled, then vkCmdSetViewportWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the viewportCount parameter of vkCmdSetViewportWithCountEXT **must** match the VkPipelineViewportStateCreateInfo::scissorCount of the pipeline

- VUID-vkCmdDraw-scissorCount-03418
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT dynamic state enabled, but not the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, then vkCmdSetScissorWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the scissorCount parameter of vkCmdSetScissorWithCountEXT **must** match the VkPipelineViewportStateCreateInfo::viewportCount of the pipeline
- VUID-vkCmdDraw-viewportCount-03419
  If the bound graphics pipeline state was created with both the VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT and VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic states enabled then both vkCmdSetViewportWithCountEXT and vkCmdSetScissorWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the viewportCount parameter of vkCmdSetViewportWithCountEXT **must** match the scissorCount parameter of vkCmdSetScissorWithCountEXT
- VUID-vkCmdDraw-None-08635
  If a shader object is bound to any graphics stage, then both vkCmdSetViewportWithCountEXT and vkCmdSetScissorWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the viewportCount parameter of vkCmdSetViewportWithCountEXT **must** match the scissorCount parameter of vkCmdSetScissorWithCountEXT
- VUID-vkCmdDraw-viewportCount-04137
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, but not the VK_DYNAMIC_STATE_VIEWPORT_W_SCALING_NV dynamic state enabled, then the bound graphics pipeline **must** have been created with VkPipelineViewportWScalingStateCreateInfoNV::viewportCount greater or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDraw-viewportCount-04138
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT and VK_DYNAMIC_STATE_VIEWPORT_W_SCALING_NV dynamic states enabled then the viewportCount parameter in the last call to vkCmdSetViewportWScalingNV **must** be greater than or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDraw-None-09232
  If the VK_NV_clip_space_w_scaling extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetViewportWScalingEnableNV in the current command buffer set viewportWScalingEnable to VK_TRUE, then vkCmdSetViewportWScalingNV **must** have been called and not subsequently invalidated

in the current command buffer prior to this drawing command

- VUID-vkCmdDraw-None-08636
  If the VK_NV_clip_space_w_scaling extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetViewportWScalingEnableNV in the current command buffer set viewportWScalingEnable to VK_TRUE, then the viewportCount parameter in the last call to vkCmdSetViewportWScalingNV **must** be greater than or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT

- VUID-vkCmdDraw-viewportCount-04139
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, but not the VK_DYNAMIC_STATE_VIEWPORT_SHADING_RATE_PALETTE_NV dynamic state enabled, then the bound graphics pipeline **must** have been created with VkPipelineViewportShadingRateImageStateCreateInfoNV::viewportCount greater or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT

- VUID-vkCmdDraw-viewportCount-04140
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT and VK_DYNAMIC_STATE_VIEWPORT_SHADING_RATE_PALETTE_NV dynamic states enabled then the viewportCount parameter in the last call to vkCmdSetViewportShadingRatePaletteNV **must** be greater than or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT

- VUID-vkCmdDraw-shadingRateImage-09233
  If the shadingRateImage feature is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetCoarseSampleOrderNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDraw-shadingRateImage-09234
  If the shadingRateImage feature is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetShadingRateImageEnableNV in the current command buffer set shadingRateImageEnable to VK_TRUE, then vkCmdSetViewportShadingRatePaletteNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDraw-None-08637
  If the shadingRateImage feature is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetShadingRateImageEnableNV in the current command buffer set shadingRateImageEnable to VK_TRUE, then the viewportCount parameter in the last call to vkCmdSetViewportShadingRatePaletteNV **must** be greater than or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT

- VUID-vkCmdDraw-VkPipelineVieportCreateInfo-04141
  If the bound graphics pipeline state was created with the

VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled and a VkPipelineViewportSwizzleStateCreateInfoNV structure chained from VkPipelineViewportStateCreateInfo, then the bound graphics pipeline **must** have been created with VkPipelineViewportSwizzleStateCreateInfoNV::viewportCount greater or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT

- VUID-vkCmdDraw-VkPipelineVieportCreateInfo-04142
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled and a VkPipelineViewportExclusiveScissorStateCreateInfoNV structure chained from VkPipelineViewportStateCreateInfo, then the bound graphics pipeline **must** have been created with VkPipelineViewportExclusiveScissorStateCreateInfoNV::exclusiveScissorCount greater or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT

- VUID-vkCmdDraw-None-07878
  If the exclusiveScissor feature is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_EXCLUSIVE_SCISSOR_ENABLE_NV dynamic state enabled, then vkCmdSetExclusiveScissorEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDraw-None-07879
  If the exclusiveScissor feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_EXCLUSIVE_SCISSOR_NV dynamic state enabled, and the most recent call to vkCmdSetExclusiveScissorEnableNV in the current command buffer set any element of pExclusiveScissorEnables to VK_TRUE, then vkCmdSetExclusiveScissorNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDraw-None-04876
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_RASTERIZER_DISCARD_ENABLE dynamic state enabled, then vkCmdSetRasterizerDiscardEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDraw-None-04877
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_BIAS_ENABLE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetDepthBiasEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDraw-logicOp-04878
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_LOGIC_OP_EXT dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of logicOpEnable is VK_TRUE, then vkCmdSetLogicOpEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDraw-primitiveFragmentShadingRateWithMultipleViewports-04552

If the primitiveFragmentShadingRateWithMultipleViewports limit is not supported, the bound graphics pipeline was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, and any of the shader stages of the bound graphics pipeline write to the PrimitiveShadingRateKHR built-in, then vkCmdSetViewportWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the viewportCount parameter of vkCmdSetViewportWithCountEXT **must** be 1

- VUID-vkCmdDraw-primitiveFragmentShadingRateWithMultipleViewports-08642
  If the primitiveFragmentShadingRateWithMultipleViewports limit is not supported, and any shader object bound to a graphics stage writes to the PrimitiveShadingRateKHR built-in, then vkCmdSetViewportWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the viewportCount parameter of vkCmdSetViewportWithCountEXT **must** be 1

- VUID-vkCmdDraw-blendEnable-04727
  If rasterization is not disabled in the bound graphics pipeline, then for each color attachment in the subpass, if the corresponding image view's format features do not contain VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT, then the blendEnable member of the corresponding element of the pAttachments member of pColorBlendState **must** be VK_FALSE

- VUID-vkCmdDraw-None-08643
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then for each color attachment in the render pass, if the corresponding image view's format features do not contain VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT, then the corresponding member of pColorBlendEnables in the most recent call to vkCmdSetColorBlendEnableEXT in the current command buffer that affected that attachment index **must** have been VK_FALSE

- VUID-vkCmdDraw-multisampledRenderToSingleSampled-07284
  If rasterization is not disabled in the bound graphics pipeline, and none of the following is enabled:

  - the VK_AMD_mixed_attachment_samples extension
  - the VK_NV_framebuffer_mixed_samples extension
  - the multisampledRenderToSingleSampled feature

  then rasterizationSamples for the currently bound graphics pipeline **must** be the same as the current subpass color and/or depth/stencil attachments

- VUID-vkCmdDraw-None-08644
  If a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and none of the following is enabled:

  - the VK_AMD_mixed_attachment_samples extension
  - the VK_NV_framebuffer_mixed_samples extension
  - the multisampledRenderToSingleSampled feature

then the most recent call to vkCmdSetRasterizationSamplesEXT in the current command buffer **must** have set `rasterizationSamples` to be the same as the number of samples for the current render pass color and/or depth/stencil attachments

- VUID-vkCmdDraw-None-08876
  If a shader object is bound to any graphics stage, the current render pass instance **must** have been begun with vkCmdBeginRenderingKHR
- VUID-vkCmdDraw-imageView-06172
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the `imageView` member of `pDepthAttachment` is not VK_NULL_HANDLE, and the `layout` member of `pDepthAttachment` is `VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL`, this command **must** not write any values to the depth attachment
- VUID-vkCmdDraw-imageView-06173
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the `imageView` member of `pStencilAttachment` is not VK_NULL_HANDLE, and the `layout` member of `pStencilAttachment` is `VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL`, this command **must** not write any values to the stencil attachment
- VUID-vkCmdDraw-imageView-06174
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the `imageView` member of `pDepthAttachment` is not VK_NULL_HANDLE, and the `layout` member of `pDepthAttachment` is `VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_STENCIL_ATTACHMENT_OPTIMAL`, this command **must** not write any values to the depth attachment
- VUID-vkCmdDraw-imageView-06175
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the `imageView` member of `pStencilAttachment` is not VK_NULL_HANDLE, and the `layout` member of `pStencilAttachment` is `VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_STENCIL_READ_ONLY_OPTIMAL`, this command **must** not write any values to the stencil attachment
- VUID-vkCmdDraw-imageView-06176
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the `imageView` member of `pDepthAttachment` is not VK_NULL_HANDLE, and the `layout` member of `pDepthAttachment` is `VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL`, this command **must** not write any values to the depth attachment
- VUID-vkCmdDraw-imageView-06177
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the `imageView` member of `pStencilAttachment` is not VK_NULL_HANDLE, and the `layout` member of `pStencilAttachment` is `VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL`, this command **must** not write any values to the stencil attachment
- VUID-vkCmdDraw-viewMask-06178
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the currently bound graphics pipeline **must** have been created with a VkPipelineRenderingCreateInfo::`viewMask` equal to VkRenderingInfo::`viewMask`
- VUID-vkCmdDraw-colorAttachmentCount-06179
  If the `dynamicRenderingUnusedAttachments` feature is not enabled and the current render pass instance was begun with vkCmdBeginRenderingKHR, the currently bound graphics

pipeline **must** have been created with a VkPipelineRenderingCreateInfo::colorAttachmentCount equal to VkRenderingInfo::colorAttachmentCount

- VUID-vkCmdDraw-dynamicRenderingUnusedAttachments-08910
  If the dynamicRenderingUnusedAttachments feature is not enabled, and the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingInfo::colorAttachmentCount greater than 0, then each element of the VkRenderingInfo::pColorAttachments array with an imageView not equal to VK_NULL_HANDLE **must** have been created with a VkFormat equal to the corresponding element of VkPipelineRenderingCreateInfo::pColorAttachmentFormats used to create the currently bound graphics pipeline

- VUID-vkCmdDraw-dynamicRenderingUnusedAttachments-08912
  If the dynamicRenderingUnusedAttachments feature is not enabled, and the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingInfo::colorAttachmentCount greater than 0, then each element of the VkRenderingInfo::pColorAttachments array with an imageView equal to VK_NULL_HANDLE **must** have the corresponding element of VkPipelineRenderingCreateInfo::pColorAttachmentFormats used to create the currently bound pipeline equal to VK_FORMAT_UNDEFINED

- VUID-vkCmdDraw-dynamicRenderingUnusedAttachments-08911
  If the dynamicRenderingUnusedAttachments feature is enabled, and the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingInfo::colorAttachmentCount greater than 0, then each element of the VkRenderingInfo::pColorAttachments array with an imageView not equal to VK_NULL_HANDLE **must** have been created with a VkFormat equal to the corresponding element of VkPipelineRenderingCreateInfo::pColorAttachmentFormats used to create the currently bound graphics pipeline, or the corresponding element of VkPipelineRenderingCreateInfo::pColorAttachmentFormats, if it exists, **must** be VK_FORMAT_UNDEFINED

- VUID-vkCmdDraw-colorAttachmentCount-09362
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, with a VkRenderingInfo::colorAttachmentCount equal to 1, there is no shader object bound to any graphics stage, and a color attachment with a resolve mode of VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID, each element of the VkRenderingInfo::pColorAttachments array with a resolveImageView not equal to VK_NULL_HANDLE **must** have been created with an image created with a VkExternalFormatANDROID::externalFormat value equal to the VkExternalFormatANDROID::externalFormat value used to create the currently bound graphics pipeline

- VUID-vkCmdDraw-None-09363
  If there is no shader object bound to any graphics stage, the current render pass instance was begun with vkCmdBeginRenderingKHR and a VkRenderingInfo::colorAttachmentCount equal to 1, and a color attachment with a resolve mode of VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID, each element of the VkRenderingInfo::pColorAttachments array with a imageView not equal to VK_NULL_HANDLE **must** have been created with an image created with a VkExternalFormatANDROID::externalFormat value equal to the VkExternalFormatANDROID::externalFormat value used to create the currently bound

graphics pipeline

- VUID-vkCmdDraw-None-09364
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is no shader object bound to any graphics stage, and the currently bound graphics pipeline was created with a non-zero VkExternalFormatANDROID::externalFormat value and with the VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT dynamic state enabled, then vkCmdSetColorBlendEnableEXT **must** have set the blend enable to VK_FALSE prior to this drawing command

- VUID-vkCmdDraw-None-09365
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is no shader object bound to any graphics stage, and the currently bound graphics pipeline was created with a non-zero VkExternalFormatANDROID::externalFormat value and with the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT dynamic state enabled, then vkCmdSetRasterizationSamplesEXT **must** have set rasterizationSamples to VK_SAMPLE_COUNT_1_BIT prior to this drawing command

- VUID-vkCmdDraw-None-09366
  If there is a shader object bound to any graphics stage, and the current render pass includes a color attachment that uses the VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID resolve mode, then vkCmdSetColorBlendEnableEXT **must** have set blend enable to VK_FALSE prior to this drawing command

- VUID-vkCmdDraw-rasterizationSamples-09367
  If there is a shader object bound to any graphics stage, and the current render pass includes a color attachment that uses the VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID resolve mode, then vkCmdSetRasterizationSamplesEXT **must** have set rasterizationSamples to VK_SAMPLE_COUNT_1_BIT prior to this drawing command

- VUID-vkCmdDraw-None-09368
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is no shader object bound to any graphics stage, and the currently bound graphics pipeline was created with a non-zero VkExternalFormatANDROID::externalFormat value and with the VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR dynamic state enabled, then vkCmdSetFragmentShadingRateKHR **must** have set pFragmentSize->width to 1 prior to this drawing command

- VUID-vkCmdDraw-None-09369
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is no shader object bound to any graphics stage, and the currently bound graphics pipeline was created with a non-zero VkExternalFormatANDROID::externalFormat value and with the VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR dynamic state enabled, then vkCmdSetFragmentShadingRateKHR **must** have set pFragmentSize->height to 1 prior to this drawing command

- VUID-vkCmdDraw-pFragmentSize-09370
  If there is a shader object bound to any graphics stage, and the current render pass includes a color attachment that uses the VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID resolve mode, then

vkCmdSetFragmentShadingRateKHR **must** have set `pFragmentSize->width` to `1` prior to this drawing command

- VUID-vkCmdDraw-pFragmentSize-09371
  If there is a shader object bound to any graphics stage, and the current render pass includes a color attachment that uses the `VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID` resolve mode, then vkCmdSetFragmentShadingRateKHR **must** have set `pFragmentSize->height` to `1` prior to this drawing command

- VUID-vkCmdDraw-None-07749
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_COLOR_WRITE_ENABLE_EXT` dynamic state enabled then vkCmdSetColorWriteEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDraw-None-08646
  If the `colorWriteEnable` feature is enabled on the device, and a shader object is bound to the `VK_SHADER_STAGE_FRAGMENT_BIT` stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE`, then vkCmdSetColorWriteEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDraw-attachmentCount-07750
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_COLOR_WRITE_ENABLE_EXT` dynamic state enabled then the `attachmentCount` parameter of `vkCmdSetColorWriteEnableEXT` **must** be greater than or equal to the `VkPipelineColorBlendStateCreateInfo::attachmentCount` of the currently bound graphics pipeline

- VUID-vkCmdDraw-None-08647
  If the `colorWriteEnable` feature is enabled on the device, and a shader object is bound to the `VK_SHADER_STAGE_FRAGMENT_BIT` stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE`, then the `attachmentCount` parameter of most recent call to `vkCmdSetColorWriteEnableEXT` in the current command buffer **must** be greater than or equal to the number of color attachments in the current render pass instance

- VUID-vkCmdDraw-None-07751
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_DISCARD_RECTANGLE_EXT` dynamic state enabled then vkCmdSetDiscardRectangleEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command for each discard rectangle in VkPipelineDiscardRectangleStateCreateInfoEXT::`discardRectangleCount`

- VUID-vkCmdDraw-None-07880
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_DISCARD_RECTANGLE_ENABLE_EXT` dynamic state enabled then vkCmdSetDiscardRectangleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDraw-rasterizerDiscardEnable-09236

If the VK_EXT_discard_rectangles extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetDiscardRectangleEnableEXT in the current command buffer set discardRectangleEnable to VK_TRUE, then vkCmdSetDiscardRectangleEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDraw-None-08648
  If the VK_EXT_discard_rectangles extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetDiscardRectangleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDraw-None-07881
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_DISCARD_RECTANGLE_MODE_EXT dynamic state enabled then vkCmdSetDiscardRectangleModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDraw-None-08649
  If the VK_EXT_discard_rectangles extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetDiscardRectangleEnableEXT in the current command buffer set discardRectangleEnable to VK_TRUE, then vkCmdSetDiscardRectangleModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDraw-dynamicRenderingUnusedAttachments-08913
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is not enabled, and VkRenderingInfo::pDepthAttachment->imageView was VK_NULL_HANDLE, the value of VkPipelineRenderingCreateInfo::depthAttachmentFormat used to create the currently bound graphics pipeline **must** be equal to VK_FORMAT_UNDEFINED
- VUID-vkCmdDraw-dynamicRenderingUnusedAttachments-08914
  If current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is not enabled, and VkRenderingInfo::pDepthAttachment->imageView was not VK_NULL_HANDLE, the value of VkPipelineRenderingCreateInfo::depthAttachmentFormat used to create the currently bound graphics pipeline **must** be equal to the VkFormat used to create VkRenderingInfo::pDepthAttachment->imageView
- VUID-vkCmdDraw-dynamicRenderingUnusedAttachments-08915
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is enabled, VkRenderingInfo::pDepthAttachment->imageView was not VK_NULL_HANDLE, and the value of VkPipelineRenderingCreateInfo::depthAttachmentFormat used to create the currently bound graphics pipeline was not equal to the VkFormat used to create VkRenderingInfo

::pDepthAttachment->imageView, the value of the format **must** be VK_FORMAT_UNDEFINED

- VUID-vkCmdDraw-dynamicRenderingUnusedAttachments-08916
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is not enabled, and VkRenderingInfo::pStencilAttachment->imageView was VK_NULL_HANDLE, the value of VkPipelineRenderingCreateInfo::stencilAttachmentFormat used to create the currently bound graphics pipeline **must** be equal to VK_FORMAT_UNDEFINED
- VUID-vkCmdDraw-dynamicRenderingUnusedAttachments-08917
  If current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is not enabled, and VkRenderingInfo::pStencilAttachment->imageView was not VK_NULL_HANDLE, the value of VkPipelineRenderingCreateInfo::stencilAttachmentFormat used to create the currently bound graphics pipeline **must** be equal to the VkFormat used to create VkRenderingInfo::pStencilAttachment->imageView
- VUID-vkCmdDraw-dynamicRenderingUnusedAttachments-08918
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is enabled, VkRenderingInfo::pStencilAttachment->imageView was not VK_NULL_HANDLE, and the value of VkPipelineRenderingCreateInfo::stencilAttachmentFormat used to create the currently bound graphics pipeline was not equal to the VkFormat used to create VkRenderingInfo::pStencilAttachment->imageView, the value of the format **must** be VK_FORMAT_UNDEFINED
- VUID-vkCmdDraw-imageView-06183
  If the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingFragmentShadingRateAttachmentInfoKHR::imageView was not VK_NULL_HANDLE, the currently bound graphics pipeline **must** have been created with VK_PIPELINE_CREATE_RENDERING_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR
- VUID-vkCmdDraw-imageView-06184
  If the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingFragmentDensityMapAttachmentInfoEXT::imageView was not VK_NULL_HANDLE, the currently bound graphics pipeline **must** have been created with VK_PIPELINE_CREATE_RENDERING_FRAGMENT_DENSITY_MAP_ATTACHMENT_BIT_EXT
- VUID-vkCmdDraw-colorAttachmentCount-06185
  If the currently bound pipeline was created with a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and the current render pass instance was begun with vkCmdBeginRenderingKHR with a VkRenderingInfo::colorAttachmentCount parameter greater than 0, then each element of the VkRenderingInfo::pColorAttachments array with a imageView not equal to VK_NULL_HANDLE **must** have been created with a sample count equal to the corresponding element of the pColorAttachmentSamples member of VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV used to create the currently bound graphics pipeline
- VUID-vkCmdDraw-pDepthAttachment-06186
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the currently bound pipeline was created with a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and VkRenderingInfo::pDepthAttachment->imageView was not VK_NULL_HANDLE, the value of the depthStencilAttachmentSamples

member of VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV used to create the currently bound graphics pipeline **must** be equal to the sample count used to create VkRenderingInfo::pDepthAttachment->imageView

- VUID-vkCmdDraw-pStencilAttachment-06187
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the currently bound pipeline was created with a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and VkRenderingInfo::pStencilAttachment->imageView was not VK_NULL_HANDLE, the value of the depthStencilAttachmentSamples member of VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV used to create the currently bound graphics pipeline **must** be equal to the sample count used to create VkRenderingInfo::pStencilAttachment->imageView

- VUID-vkCmdDraw-multisampledRenderToSingleSampled-07285
  If the currently bound pipeline was created without a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and the multisampledRenderToSingleSampled feature is not enabled, and the current render pass instance was begun with vkCmdBeginRenderingKHR with a VkRenderingInfo::colorAttachmentCount parameter greater than 0, then each element of the VkRenderingInfo::pColorAttachments array with a imageView not equal to VK_NULL_HANDLE **must** have been created with a sample count equal to the value of rasterizationSamples for the currently bound graphics pipeline

- VUID-vkCmdDraw-multisampledRenderToSingleSampled-07286
  If the currently bound pipeline was created without a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and the multisampledRenderToSingleSampled feature is not enabled, and VkRenderingInfo::pDepthAttachment->imageView was not VK_NULL_HANDLE, the value of rasterizationSamples for the currently bound graphics pipeline **must** be equal to the sample count used to create VkRenderingInfo::pDepthAttachment->imageView

- VUID-vkCmdDraw-multisampledRenderToSingleSampled-07287
  If the currently bound pipeline was created without a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and the multisampledRenderToSingleSampled feature is not enabled, and VkRenderingInfo::pStencilAttachment->imageView was not VK_NULL_HANDLE, the value of rasterizationSamples for the currently bound graphics pipeline **must** be equal to the sample count used to create VkRenderingInfo::pStencilAttachment->imageView

- VUID-vkCmdDraw-pNext-07935
  If this command has been called inside a render pass instance started with vkCmdBeginRenderingKHR, and the pNext chain of VkRenderingInfo includes a VkMultisampledRenderToSingleSampledInfoEXT structure with multisampledRenderToSingleSampledEnable equal to VK_TRUE, then the value of rasterizationSamples for the currently bound graphics pipeline **must** be equal to VkMultisampledRenderToSingleSampledInfoEXT::rasterizationSamples

- VUID-vkCmdDraw-renderPass-06198
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the currently bound pipeline **must** have been created with a VkGraphicsPipelineCreateInfo::renderPass equal to VK_NULL_HANDLE

- VUID-vkCmdDraw-pColorAttachments-08963
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is a graphics pipeline bound with a fragment shader that statically writes to a color attachment, the color write mask is not zero, color writes are enabled, and the corresponding element of the VkRenderingInfo::pColorAttachments->imageView was not VK_NULL_HANDLE, then the corresponding element of VkPipelineRenderingCreateInfo::pColorAttachmentFormats used to create the pipeline **must** not be VK_FORMAT_UNDEFINED
- VUID-vkCmdDraw-pDepthAttachment-08964
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is a graphics pipeline bound, depth test is enabled, depth write is enabled, and the VkRenderingInfo::pDepthAttachment->imageView was not VK_NULL_HANDLE, then the VkPipelineRenderingCreateInfo::depthAttachmentFormat used to create the pipeline **must** not be VK_FORMAT_UNDEFINED
- VUID-vkCmdDraw-pStencilAttachment-08965
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is a graphics pipeline bound, stencil test is enabled and the VkRenderingInfo::pStencilAttachment->imageView was not VK_NULL_HANDLE, then the VkPipelineRenderingCreateInfo::stencilAttachmentFormat used to create the pipeline **must** not be VK_FORMAT_UNDEFINED
- VUID-vkCmdDraw-primitivesGeneratedQueryWithRasterizerDiscard-06708
  If the primitivesGeneratedQueryWithRasterizerDiscard feature is not enabled and the VK_QUERY_TYPE_PRIMITIVES_GENERATED_EXT query is active, rasterization discard **must** not be enabled
- VUID-vkCmdDraw-primitivesGeneratedQueryWithNonZeroStreams-06709
  If the primitivesGeneratedQueryWithNonZeroStreams feature is not enabled and the VK_QUERY_TYPE_PRIMITIVES_GENERATED_EXT query is active, the bound graphics pipeline **must** not have been created with a non-zero value in VkPipelineRasterizationStateStreamCreateInfoEXT::rasterizationStream
- VUID-vkCmdDraw-None-07619
  If a shader object is bound to the VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_TESSELLATION_DOMAIN_ORIGIN_EXT dynamic state enabled, then vkCmdSetTessellationDomainOriginEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDraw-None-07620
  If the depthClamp feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_CLAMP_ENABLE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetDepthClampEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDraw-None-07621
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_POLYGON_MODE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetPolygonModeEXT

**must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDraw-None-07622
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT` dynamic state enabled, and the current value of `rasterizerDiscardEnable` is `VK_FALSE`, then vkCmdSetRasterizationSamplesEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDraw-None-07623
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_SAMPLE_MASK_EXT` dynamic state enabled, and the current value of `rasterizerDiscardEnable` is `VK_FALSE`, then vkCmdSetSampleMaskEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDraw-alphaToCoverageEnable-08919
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_ALPHA_TO_COVERAGE_ENABLE_EXT` dynamic state enabled, and `alphaToCoverageEnable` was `VK_TRUE` in the last call to vkCmdSetAlphaToCoverageEnableEXT, then the Fragment Output Interface **must** contain a variable for the alpha `Component` word in `Location` 0 at `Index` 0

- VUID-vkCmdDraw-alphaToCoverageEnable-08920
  If a shader object is bound to any graphics stage, and the most recent call to vkCmdSetAlphaToCoverageEnableEXT in the current command buffer set `alphaToCoverageEnable` to `VK_TRUE`, then the Fragment Output Interface **must** contain a variable for the alpha `Component` word in `Location` 0 at `Index` 0

- VUID-vkCmdDraw-None-07624
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_ALPHA_TO_COVERAGE_ENABLE_EXT` dynamic state enabled, and the current value of `rasterizerDiscardEnable` is `VK_FALSE`, then vkCmdSetAlphaToCoverageEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDraw-None-07625
  If the `alphaToOne` feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_ALPHA_TO_ONE_ENABLE_EXT` dynamic state enabled, and the current value of `rasterizerDiscardEnable` is `VK_FALSE`, then vkCmdSetAlphaToOneEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDraw-None-07626
  If the `logicOp` feature is enabled, a shader object is bound to the `VK_SHADER_STAGE_FRAGMENT_BIT` stage or a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_LOGIC_OP_ENABLE_EXT` dynamic state enabled, and the current value of `rasterizerDiscardEnable` is `VK_FALSE`, then vkCmdSetLogicOpEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDraw-None-07627
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT dynamic state enabled then vkCmdSetColorBlendEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDraw-None-08657
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and both the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE and there are color attachments bound, then vkCmdSetColorBlendEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDraw-None-07628
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_EQUATION_EXT dynamic state enabled then vkCmdSetColorBlendEquationEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDraw-None-08658
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetColorBlendEnableEXT for any attachment set that attachment's value in pColorBlendEnables to VK_TRUE, then vkCmdSetColorBlendEquationEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDraw-None-07629
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_WRITE_MASK_EXT dynamic state enabled then vkCmdSetColorWriteMaskEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDraw-None-08659
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and both the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE and there are color attachments bound, then vkCmdSetColorWriteMaskEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDraw-None-07630
  If the geometryStreams feature is enabled, and a shader object is bound to the VK_SHADER_STAGE_GEOMETRY_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_RASTERIZATION_STREAM_EXT dynamic state enabled, then vkCmdSetRasterizationStreamEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDraw-None-07631
  If the VK_EXT_conservative_rasterization extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_CONSERVATIVE_RASTERIZATION_MODE_EXT dynamic state enabled, and the

current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetConservativeRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDraw-None-07632
  If the VK_EXT_conservative_rasterization extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_EXTRA_PRIMITIVE_OVERESTIMATION_SIZE_EXT dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of conservativeRasterizationMode is VK_CONSERVATIVE_RASTERIZATION_MODE_OVERESTIMATE_EXT, then vkCmdSetExtraPrimitiveOverestimationSizeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDraw-None-07633
  If the depthClipEnable feature is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_CLIP_ENABLE_EXT dynamic state, then vkCmdSetDepthClipEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDraw-None-07634
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT dynamic state enabled then vkCmdSetSampleLocationsEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDraw-None-08664
  If the VK_EXT_sample_locations extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetSampleLocationsEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDraw-None-07635
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_ADVANCED_EXT dynamic state enabled then vkCmdSetColorBlendAdvancedEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDraw-rasterizerDiscardEnable-09416
  If the VK_EXT_blend_operation_advanced extension is enabled, and a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then at least one of vkCmdSetColorBlendEquationEXT and vkCmdSetColorBlendAdvancedEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDraw-None-07636
  If the VK_EXT_provoking_vertex extension is enabled, a shader object is bound to the VK_SHADER_STAGE_VERTEX_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_PROVOKING_VERTEX_MODE_EXT dynamic state enabled, and the current

value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetProvokingVertexModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDraw-None-07637
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT dynamic state enabled then vkCmdSetLineRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDraw-None-08666
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetPolygonModeEXT in the current command buffer set polygonMode to VK_POLYGON_MODE_LINE, then vkCmdSetLineRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDraw-None-08667
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object is bound to the VK_SHADER_STAGE_VERTEX_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetPrimitiveTopologyEXT in the current command buffer set primitiveTopology to any line topology, then vkCmdSetLineRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDraw-None-08668
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object that outputs line primitives is bound to the VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT or VK_SHADER_STAGE_GEOMETRY_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetLineRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDraw-None-07638
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT dynamic state enabled then vkCmdSetLineStippleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDraw-None-08669
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetPolygonModeEXT in the current command buffer set polygonMode to

VK_POLYGON_MODE_LINE, then vkCmdSetLineStippleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDraw-None-08670
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object is bound to the VK_SHADER_STAGE_VERTEX_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetPrimitiveTopologyEXT in the current command buffer set primitiveTopology to any line topology, then vkCmdSetLineStippleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDraw-None-08671
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object that outputs line primitives is bound to the VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT or VK_SHADER_STAGE_GEOMETRY_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetLineStippleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDraw-None-07849
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_KHR dynamic state enabled then vkCmdSetLineStippleKHR **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDraw-None-08672
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetLineStippleEnableEXT in the current command buffer set stippledLineEnable to VK_TRUE, then vkCmdSetLineStippleEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDraw-None-07639
  If the depthClipControl feature is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_CLIP_NEGATIVE_ONE_TO_ONE_EXT dynamic state enabled, then vkCmdSetDepthClipNegativeOneToOneEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDraw-None-07640
  If the VK_NV_clip_space_w_scaling extension is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_VIEWPORT_W_SCALING_ENABLE_NV dynamic state enabled, then vkCmdSetViewportWScalingEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDraw-None-07641

If the VK_NV_viewport_swizzle extension is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_VIEWPORT_SWIZZLE_NV dynamic state enabled, then vkCmdSetViewportSwizzleNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDraw-None-07642
  If the VK_NV_fragment_coverage_to_color extension is enabled, a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_TO_COLOR_ENABLE_NV dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetCoverageToColorEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDraw-None-07643
  If the VK_NV_fragment_coverage_to_color extension is enabled, a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_TO_COLOR_LOCATION_NV dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of coverageToColorEnable is VK_TRUE, then vkCmdSetCoverageToColorLocationNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDraw-None-07644
  If the VK_NV_framebuffer_mixed_samples extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_MODULATION_MODE_NV dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetCoverageModulationModeNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDraw-None-07645
  If the VK_NV_framebuffer_mixed_samples extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_MODULATION_TABLE_ENABLE_NV dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of coverageModulationMode is any value other than VK_COVERAGE_MODULATION_MODE_NONE_NV, then vkCmdSetCoverageModulationTableEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDraw-None-07646
  If the VK_NV_framebuffer_mixed_samples extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_MODULATION_TABLE_NV dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of coverageModulationTableEnable is VK_TRUE, then vkCmdSetCoverageModulationTableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDraw-None-07647
  If the shadingRateImage feature is enabled, a shader object is bound to any graphics stage

or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_SHADING_RATE_IMAGE_ENABLE_NV dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetShadingRateImageEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDraw-pipelineFragmentShadingRate-09238
  If the pipelineFragmentShadingRate feature is enabled, a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetFragmentShadingRateKHR **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDraw-None-07648
  If the representativeFragmentTest feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_REPRESENTATIVE_FRAGMENT_TEST_ENABLE_NV dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetRepresentativeFragmentTestEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDraw-None-07649
  If the coverageReductionMode feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_REDUCTION_MODE_NV dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetCoverageReductionModeNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDraw-pColorBlendEnables-07470
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT state enabled and the last call to vkCmdSetColorBlendEnableEXT set pColorBlendEnables for any attachment to VK_TRUE, then for those attachments in the subpass the corresponding image view's format features **must** contain VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT

- VUID-vkCmdDraw-rasterizationSamples-07471
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, and the current subpass does not use any color and/or depth/stencil attachments, then the rasterizationSamples in the last call to vkCmdSetRasterizationSamplesEXT **must** follow the rules for a zero-attachment subpass

- VUID-vkCmdDraw-samples-07472
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_MASK_EXT state enabled and the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state disabled, then the samples parameter in the last call to vkCmdSetSampleMaskEXT **must** be greater or equal to the VkPipelineMultisampleStateCreateInfo::rasterizationSamples parameter used to create the bound graphics pipeline

- VUID-vkCmdDraw-samples-07473
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_SAMPLE_MASK_EXT` state and `VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT` states enabled, then the `samples` parameter in the last call to vkCmdSetSampleMaskEXT **must** be greater or equal to the `rasterizationSamples` parameter in the last call to vkCmdSetRasterizationSamplesEXT

- VUID-vkCmdDraw-rasterizationSamples-07474
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT` state enabled, and neither the `VK_AMD_mixed_attachment_samples` nor the `VK_NV_framebuffer_mixed_samples` extensions are enabled, then the `rasterizationSamples` in the last call to vkCmdSetRasterizationSamplesEXT **must** be the same as the current subpass color and/or depth/stencil attachments

- VUID-vkCmdDraw-None-09211
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT` state enabled, or a shader object is bound to any graphics stage, and the current render pass instance includes a VkMultisampledRenderToSingleSampledInfoEXT structure with `multisampledRenderToSingleSampledEnable` equal to `VK_TRUE`, then the `rasterizationSamples` in the last call to vkCmdSetRasterizationSamplesEXT **must** be the same as the `rasterizationSamples` member of that structure

- VUID-vkCmdDraw-firstAttachment-07476
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT` dynamic state enabled then vkCmdSetColorBlendEnableEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the `firstAttachment` and `attachmentCount` parameters of `vkCmdSetColorBlendEnableEXT` calls **must** specify an enable for all active color attachments in the current subpass

- VUID-vkCmdDraw-rasterizerDiscardEnable-09417
  If a shader object is bound to the `VK_SHADER_STAGE_FRAGMENT_BIT` stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE`, then vkCmdSetColorBlendEnableEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the `firstAttachment` and `attachmentCount` parameters of `vkCmdSetColorBlendEnableEXT` calls **must** specify an enable for all active color attachments in the current subpass

- VUID-vkCmdDraw-firstAttachment-07477
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_COLOR_BLEND_EQUATION_EXT` dynamic state enabled then vkCmdSetColorBlendEquationEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the `firstAttachment` and `attachmentCount` parameters of `vkCmdSetColorBlendEquationEXT` calls **must** specify the blend equations for all active color attachments in the current subpass where blending is enabled

- VUID-vkCmdDraw-rasterizerDiscardEnable-09418

If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and both the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE and there are color attachments bound, then vkCmdSetColorBlendEquationEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the firstAttachment and attachmentCount parameters of vkCmdSetColorBlendEquationEXT calls **must** specify the blend equations for all active color attachments in the current subpass where blending is enabled

- VUID-vkCmdDraw-firstAttachment-07478
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_WRITE_MASK_EXT dynamic state enabled then vkCmdSetColorWriteMaskEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the firstAttachment and attachmentCount parameters of vkCmdSetColorWriteMaskEXT calls **must** specify the color write mask for all active color attachments in the current subpass
- VUID-vkCmdDraw-rasterizerDiscardEnable-09419
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetColorWriteMaskEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the firstAttachment and attachmentCount parameters of vkCmdSetColorWriteMaskEXT calls **must** specify the color write mask for all active color attachments in the current subpass
- VUID-vkCmdDraw-firstAttachment-07479
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_ADVANCED_EXT dynamic state enabled then vkCmdSetColorBlendAdvancedEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the firstAttachment and attachmentCount parameters of vkCmdSetColorBlendAdvancedEXT calls **must** specify the advanced blend equations for all active color attachments in the current subpass where blending is enabled
- VUID-vkCmdDraw-advancedBlendMaxColorAttachments-07480
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_ADVANCED_EXT and VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT dynamic states enabled and the last calls to vkCmdSetColorBlendEnableEXT and vkCmdSetColorBlendAdvancedEXT have enabled advanced blending, then the number of active color attachments in the current subpass **must** not exceed advancedBlendMaxColorAttachments
- VUID-vkCmdDraw-primitivesGeneratedQueryWithNonZeroStreams-07481
  If the primitivesGeneratedQueryWithNonZeroStreams feature is not enabled and the VK_QUERY_TYPE_PRIMITIVES_GENERATED_EXT query is active, and the bound graphics pipeline was created with VK_DYNAMIC_STATE_RASTERIZATION_STREAM_EXT state enabled, the last call to vkCmdSetRasterizationStreamEXT **must** have set the rasterizationStream to zero
- VUID-vkCmdDraw-sampleLocationsPerPixel-07482
  If the bound graphics pipeline state was created with the

VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state enabled and the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state disabled, then the sampleLocationsPerPixel member of pSampleLocationsInfo in the last call to vkCmdSetSampleLocationsEXT **must** equal the rasterizationSamples member of the VkPipelineMultisampleStateCreateInfo structure the bound graphics pipeline has been created with

- VUID-vkCmdDraw-sampleLocationsPerPixel-07483
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state enabled and the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, then the sampleLocationsPerPixel member of pSampleLocationsInfo in the last call to vkCmdSetSampleLocationsEXT **must** equal the rasterizationSamples parameter of the last call to vkCmdSetRasterizationSamplesEXT

- VUID-vkCmdDraw-sampleLocationsEnable-07484
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, or the bound graphics pipeline was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, and sampleLocationsEnable was VK_TRUE in the last call to vkCmdSetSampleLocationsEnableEXT, and the current subpass has a depth/stencil attachment, then that attachment **must** have been created with the VK_IMAGE_CREATE_SAMPLE_LOCATIONS_COMPATIBLE_DEPTH_BIT_EXT bit set

- VUID-vkCmdDraw-sampleLocationsEnable-07485
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state enabled and the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, and if sampleLocationsEnable was VK_TRUE in the last call to vkCmdSetSampleLocationsEnableEXT, then the sampleLocationsInfo.sampleLocationGridSize.width in the last call to vkCmdSetSampleLocationsEXT **must** evenly divide VkMultisamplePropertiesEXT::sampleLocationGridSize.width as returned by vkGetPhysicalDeviceMultisamplePropertiesEXT with a samples parameter equaling rasterizationSamples

- VUID-vkCmdDraw-sampleLocationsEnable-07486
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state enabled and the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, and if sampleLocationsEnable was VK_TRUE in the last call to vkCmdSetSampleLocationsEnableEXT, then the sampleLocationsInfo.sampleLocationGridSize.height in the last call to vkCmdSetSampleLocationsEXT **must** evenly divide VkMultisamplePropertiesEXT::sampleLocationGridSize.height as returned by vkGetPhysicalDeviceMultisamplePropertiesEXT with a samples parameter equaling rasterizationSamples

- VUID-vkCmdDraw-sampleLocationsEnable-07487
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, and if sampleLocationsEnable

was VK_TRUE in the last call to vkCmdSetSampleLocationsEnableEXT, the fragment shader code **must** not statically use the extended instruction InterpolateAtSample

- VUID-vkCmdDraw-sampleLocationsEnable-07936
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state disabled and the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, the sampleLocationsEnable member of a VkPipelineSampleLocationsStateCreateInfoEXT::sampleLocationsEnable in the bound graphics pipeline is VK_TRUE or VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, then, sampleLocationsInfo.sampleLocationGridSize.width **must** evenly divide VkMultisamplePropertiesEXT::sampleLocationGridSize.width as returned by vkGetPhysicalDeviceMultisamplePropertiesEXT with a samples parameter equaling the value of rasterizationSamples in the last call to vkCmdSetRasterizationSamplesEXT

- VUID-vkCmdDraw-sampleLocationsEnable-07937
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state disabled and the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, the sampleLocationsEnable member of a VkPipelineSampleLocationsStateCreateInfoEXT::sampleLocationsEnable in the bound graphics pipeline is VK_TRUE or VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, then, sampleLocationsInfo.sampleLocationGridSize.height **must** evenly divide VkMultisamplePropertiesEXT::sampleLocationGridSize.height as returned by vkGetPhysicalDeviceMultisamplePropertiesEXT with a samples parameter equaling the value of rasterizationSamples in the last call to vkCmdSetRasterizationSamplesEXT

- VUID-vkCmdDraw-sampleLocationsEnable-07938
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state disabled and the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, the sampleLocationsEnable member of a VkPipelineSampleLocationsStateCreateInfoEXT::sampleLocationsEnable in the bound graphics pipeline is VK_TRUE or VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, then, sampleLocationsInfo.sampleLocationsPerPixel **must** equal rasterizationSamples in the last call to vkCmdSetRasterizationSamplesEXT

- VUID-vkCmdDraw-coverageModulationTableEnable-07488
  If a shader object is bound to any graphics stage or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COVERAGE_MODULATION_TABLE_ENABLE_NV state enabled, and the last call to vkCmdSetCoverageModulationTableEnableNV set coverageModulationTableEnable to VK_TRUE, then the coverageModulationTableCount parameter in the last call to vkCmdSetCoverageModulationTableNV **must** equal the current rasterizationSamples divided by the number of color samples in the current subpass

- VUID-vkCmdDraw-rasterizationSamples-07489
  If the VK_NV_framebuffer_mixed_samples extension is enabled, and if current subpass has a depth/stencil attachment and depth test, stencil test, or depth bounds test are enabled in the currently bound pipeline state, then the current rasterizationSamples **must** be the same as the sample count of the depth/stencil attachment

- VUID-vkCmdDraw-coverageToColorEnable-07490
  If the bound graphics pipeline state was created with the

VK_DYNAMIC_STATE_COVERAGE_TO_COLOR_ENABLE_NV state enabled and the last call to vkCmdSetCoverageToColorEnableNV set the coverageToColorEnable to VK_TRUE, then the current subpass **must** have a color attachment at the location selected by the last call to vkCmdSetCoverageToColorLocationNV coverageToColorLocation, with a VkFormat of VK_FORMAT_R8_UINT, VK_FORMAT_R8_SINT, VK_FORMAT_R16_UINT, VK_FORMAT_R16_SINT, VK_FORMAT_R32_UINT, or VK_FORMAT_R32_SINT

- VUID-vkCmdDraw-rasterizerDiscardEnable-09420
  If the VK_NV_fragment_coverage_to_color extension is enabled, and a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the last call to vkCmdSetCoverageToColorEnableNV set the coverageToColorEnable to VK_TRUE, then the current subpass **must** have a color attachment at the location selected by the last call to vkCmdSetCoverageToColorLocationNV coverageToColorLocation, with a VkFormat of VK_FORMAT_R8_UINT, VK_FORMAT_R8_SINT, VK_FORMAT_R16_UINT, VK_FORMAT_R16_SINT, VK_FORMAT_R32_UINT, or VK_FORMAT_R32_SINT

- VUID-vkCmdDraw-coverageReductionMode-07491
  If this VK_NV_coverage_reduction_mode extension is enabled, the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COVERAGE_TO_COLOR_ENABLE_NV and VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT states enabled, the current coverage reduction mode coverageReductionMode, then the current rasterizationSamples, and the sample counts for the color and depth/stencil attachments (if the subpass has them) **must** be a valid combination returned by vkGetPhysicalDeviceSupportedFramebufferMixedSamplesCombinationsNV

- VUID-vkCmdDraw-viewportCount-07492
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, but not the VK_DYNAMIC_STATE_VIEWPORT_SWIZZLE_NV dynamic state enabled, then the bound graphics pipeline **must** have been created with VkPipelineViewportSwizzleStateCreateInfoNV ::viewportCount greater or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT

- VUID-vkCmdDraw-viewportCount-07493
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT and VK_DYNAMIC_STATE_VIEWPORT_SWIZZLE_NV dynamic states enabled then the viewportCount parameter in the last call to vkCmdSetViewportSwizzleNV **must** be greater than or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT

- VUID-vkCmdDraw-viewportCount-09421
  If the VK_NV_viewport_swizzle extension is enabled, and a shader object is bound to any graphics stage, then the viewportCount parameter in the last call to vkCmdSetViewportSwizzleNV **must** be greater than or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT

- VUID-vkCmdDraw-rasterizationSamples-07494
  If the VK_NV_framebuffer_mixed_samples extension is enabled, and if the current subpass has any color attachments and rasterizationSamples of the last call to

vkCmdSetRasterizationSamplesEXT is greater than the number of color samples, then the pipeline sampleShadingEnable **must** be VK_FALSE

- VUID-vkCmdDraw-stippledLineEnable-07495
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT or VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT dynamic states enabled, and if the current stippledLineEnable state is VK_TRUE and the current lineRasterizationMode state is VK_LINE_RASTERIZATION_MODE_RECTANGULAR_KHR, then the stippledRectangularLines feature **must** be enabled
- VUID-vkCmdDraw-stippledLineEnable-07496
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT or VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT dynamic states enabled, and if the current stippledLineEnable state is VK_TRUE and the current lineRasterizationMode state is VK_LINE_RASTERIZATION_MODE_BRESENHAM_KHR, then the stippledBresenhamLines feature **must** be enabled
- VUID-vkCmdDraw-stippledLineEnable-07497
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT or VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT dynamic states enabled, and if the current stippledLineEnable state is VK_TRUE and the current lineRasterizationMode state is VK_LINE_RASTERIZATION_MODE_RECTANGULAR_SMOOTH_KHR, then the stippledSmoothLines feature **must** be enabled
- VUID-vkCmdDraw-stippledLineEnable-07498
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT or VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT dynamic states enabled, and if the current stippledLineEnable state is VK_TRUE and the current lineRasterizationMode state is VK_LINE_RASTERIZATION_MODE_DEFAULT_KHR, then the stippledRectangularLines feature **must** be enabled and VkPhysicalDeviceLimits::strictLines **must** be VK_TRUE
- VUID-vkCmdDraw-conservativePointAndLineRasterization-07499
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_CONSERVATIVE_RASTERIZATION_MODE_EXT dynamic state enabled, conservativePointAndLineRasterization is not supported, and the effective primitive topology output by the last pre-rasterization shader stage is a line or point, then the conservativeRasterizationMode set by the last call to vkCmdSetConservativeRasterizationModeEXT **must** be VK_CONSERVATIVE_RASTERIZATION_MODE_DISABLED_EXT
- VUID-vkCmdDraw-stage-07073
  If the currently bound pipeline was created with the VkPipelineShaderStageCreateInfo::stage member of an element of VkGraphicsPipelineCreateInfo::pStages set to VK_SHADER_STAGE_VERTEX_BIT, VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT, VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT or VK_SHADER_STAGE_GEOMETRY_BIT, then Mesh Shader Queries **must** not be active
- VUID-vkCmdDraw-None-08877

If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_ATTACHMENT_FEEDBACK_LOOP_ENABLE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetAttachmentFeedbackLoopEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDraw-None-07850
  If dynamic state was inherited from VkCommandBufferInheritanceViewportScissorInfoNV, it **must** be set in the current command buffer prior to this drawing command
- VUID-vkCmdDraw-None-08684
  If there is no bound graphics pipeline, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_VERTEX_BIT
- VUID-vkCmdDraw-None-08685
  If there is no bound graphics pipeline, and the tessellationShader feature is enabled, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT
- VUID-vkCmdDraw-None-08686
  If there is no bound graphics pipeline, and the tessellationShader feature is enabled, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT
- VUID-vkCmdDraw-None-08687
  If there is no bound graphics pipeline, and the geometryShader feature is enabled, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_GEOMETRY_BIT
- VUID-vkCmdDraw-None-08688
  If there is no bound graphics pipeline, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_FRAGMENT_BIT
- VUID-vkCmdDraw-None-08689
  If there is no bound graphics pipeline, and the taskShader feature is enabled, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_TASK_BIT_EXT
- VUID-vkCmdDraw-None-08690
  If there is no bound graphics pipeline, and the meshShader feature is enabled, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_MESH_BIT_EXT
- VUID-vkCmdDraw-None-08693
  If there is no bound graphics pipeline, and at least one of the taskShader and meshShader features is enabled, one of the VK_SHADER_STAGE_VERTEX_BIT or VK_SHADER_STAGE_MESH_BIT_EXT stages **must** have a valid VkShaderEXT bound, and the other **must** have no VkShaderEXT bound
- VUID-vkCmdDraw-None-08694
  If there is no bound graphics pipeline, and both the taskShader and meshShader features are enabled, and a valid VkShaderEXT is bound the to the VK_SHADER_STAGE_MESH_BIT_EXT

stage, and that VkShaderEXT was created without the VK_SHADER_CREATE_NO_TASK_SHADER_BIT_EXT flag, a valid VkShaderEXT **must** be bound to the VK_SHADER_STAGE_TASK_BIT_EXT stage

- VUID-vkCmdDraw-None-08695
  If there is no bound graphics pipeline, and both the taskShader and meshShader features are enabled, and a valid VkShaderEXT is bound the to the VK_SHADER_STAGE_MESH_BIT_EXT stage, and that VkShaderEXT was created with the VK_SHADER_CREATE_NO_TASK_SHADER_BIT_EXT flag, there **must** be no VkShaderEXT bound to the VK_SHADER_STAGE_TASK_BIT_EXT stage
- VUID-vkCmdDraw-None-08696
  If there is no bound graphics pipeline, and a valid VkShaderEXT is bound to the VK_SHADER_STAGE_VERTEX_BIT stage, there **must** be no VkShaderEXT bound to either the VK_SHADER_STAGE_TASK_BIT_EXT stage or the VK_SHADER_STAGE_MESH_BIT_EXT stage
- VUID-vkCmdDraw-None-08698
  If any graphics shader is bound which was created with the VK_SHADER_CREATE_LINK_STAGE_BIT_EXT flag, then all shaders created with the VK_SHADER_CREATE_LINK_STAGE_BIT_EXT flag in the same vkCreateShadersEXT call **must** also be bound
- VUID-vkCmdDraw-None-08699
  If any graphics shader is bound which was created with the VK_SHADER_CREATE_LINK_STAGE_BIT_EXT flag, any stages in between stages whose shaders which did not create a shader with the VK_SHADER_CREATE_LINK_STAGE_BIT_EXT flag as part of the same vkCreateShadersEXT call **must** not have any VkShaderEXT bound
- VUID-vkCmdDraw-None-08878
  All bound graphics shader objects **must** have been created with identical or identically defined push constant ranges
- VUID-vkCmdDraw-None-08879
  All bound graphics shader objects **must** have been created with identical or identically defined arrays of descriptor set layouts
- VUID-vkCmdDraw-colorAttachmentCount-09372
  If the current render pass instance was begun with vkCmdBeginRenderingKHR and a VkRenderingInfo::colorAttachmentCount equal to 1, a color attachment with a resolve mode of VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID, and a fragment shader is bound, it **must** not declare the DepthReplacing or StencilRefReplacingEXT execution modes
- VUID-vkCmdDraw-pDynamicStates-08715
  If the bound graphics pipeline state includes a fragment shader stage, was created with VK_DYNAMIC_STATE_DEPTH_WRITE_ENABLE set in VkPipelineDynamicStateCreateInfo::pDynamicStates, and the fragment shader declares the EarlyFragmentTests execution mode and uses OpDepthAttachmentReadEXT, the depthWriteEnable parameter in the last call to vkCmdSetDepthWriteEnableEXT **must** be VK_FALSE
- VUID-vkCmdDraw-pDynamicStates-08716
  If the bound graphics pipeline state includes a fragment shader stage, was created with VK_DYNAMIC_STATE_STENCIL_WRITE_MASK set in VkPipelineDynamicStateCreateInfo::pDynamicStates, and the fragment shader declares the EarlyFragmentTests execution mode and uses OpStencilAttachmentReadEXT, the writeMask parameter in the last call to

vkCmdSetStencilWriteMask **must** be `0`

- VUID-vkCmdDraw-None-09116
  If a shader object is bound to any graphics stage or the currently bound graphics pipeline was created with `VK_DYNAMIC_STATE_COLOR_WRITE_MASK_EXT`, and the format of any color attachment is `VK_FORMAT_E5B9G9R9_UFLOAT_PACK32`, the corresponding element of the `pColorWriteMasks` parameter of vkCmdSetColorWriteMaskEXT **must** either include all of `VK_COLOR_COMPONENT_R_BIT`, `VK_COLOR_COMPONENT_G_BIT`, and `VK_COLOR_COMPONENT_B_BIT`, or none of them

- VUID-vkCmdDraw-maxFragmentDualSrcAttachments-09239
  If blending is enabled for any attachment where either the source or destination blend factors for that attachment use the secondary color input, the maximum value of `Location` for any output attachment statically used in the `Fragment Execution Model` executed by this command **must** be less than `maxFragmentDualSrcAttachments`

- VUID-vkCmdDraw-None-09548
  If the current render pass was begun with vkCmdBeginRenderingKHR, and there is no shader object bound to any graphics stage, the value of each element of VkRenderingAttachmentLocationInfoKHR::`pColorAttachmentLocations` set by vkCmdSetRenderingAttachmentLocationsKHR **must** match the value set for the corresponding element in the currently bound pipeline

- VUID-vkCmdDraw-None-09549
  If the current render pass was begun with vkCmdBeginRenderingKHR, and there is no shader object bound to any graphics stage, input attachment index mappings in the currently bound pipeline **must** match those set for the current render pass instance via VkRenderingInputAttachmentIndexInfoKHR

- VUID-vkCmdDraw-None-09642
  If the current render pass was begun with vkCmdBeginRenderingKHR with the `VK_RENDERING_ENABLE_LEGACY_DITHERING_BIT_EXT` flag, the bound graphics pipeline **must** have been created with `VK_PIPELINE_CREATE_2_ENABLE_LEGACY_DITHERING_BIT_EXT`

- VUID-vkCmdDraw-None-09643
  If the bound graphics pipeline was created with `VK_PIPELINE_CREATE_2_ENABLE_LEGACY_DITHERING_BIT_EXT`, the current render pass **must** have begun with vkCmdBeginRenderingKHR with the `VK_RENDERING_ENABLE_LEGACY_DITHERING_BIT_EXT` flag

- VUID-vkCmdDraw-commandBuffer-02712
  If `commandBuffer` is a protected command buffer and `protectedNoFault` is not supported, any resource written to by the `VkPipeline` object bound to the pipeline bind point used by this command **must** not be an unprotected resource

- VUID-vkCmdDraw-commandBuffer-02713
  If `commandBuffer` is a protected command buffer and `protectedNoFault` is not supported, pipeline stages other than the framebuffer-space and compute stages in the `VkPipeline` object bound to the pipeline bind point used by this command **must** not write to any resource

- VUID-vkCmdDraw-commandBuffer-04617
  If any of the shader stages of the `VkPipeline` bound to the pipeline bind point used by this

command uses the RayQueryKHR capability, then commandBuffer **must** not be a protected command buffer

- VUID-vkCmdDraw-None-04007
  All vertex input bindings accessed via vertex input variables declared in the vertex shader entry point's interface **must** have either valid or VK_NULL_HANDLE buffers bound
- VUID-vkCmdDraw-None-04008
  If the nullDescriptor feature is not enabled, all vertex input bindings accessed via vertex input variables declared in the vertex shader entry point's interface **must** not be VK_NULL_HANDLE
- VUID-vkCmdDraw-None-02721
  If robustBufferAccess is not enabled, and that pipeline was created without enabling VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_EXT for vertexInputs, then for a given vertex buffer binding, any attribute data fetched **must** be entirely contained within the corresponding vertex buffer binding, as described in Vertex Input Description
- VUID-vkCmdDraw-None-07842
  If there is a shader object bound to the VK_SHADER_STAGE_VERTEX_BIT stage or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_PRIMITIVE_TOPOLOGY dynamic state enabled then vkCmdSetPrimitiveTopologyEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDraw-dynamicPrimitiveTopologyUnrestricted-07500
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_PRIMITIVE_TOPOLOGY dynamic state enabled and the dynamicPrimitiveTopologyUnrestricted is VK_FALSE, then the primitiveTopology parameter of vkCmdSetPrimitiveTopologyEXT **must** be of the same topology class as the pipeline VkPipelineInputAssemblyStateCreateInfo::topology state
- VUID-vkCmdDraw-pStrides-04913
  If the bound graphics pipeline was created with the VK_DYNAMIC_STATE_VERTEX_INPUT_BINDING_STRIDE_EXT dynamic state enabled, but without the VK_DYNAMIC_STATE_VERTEX_INPUT_EXT dynamic state enabled, then vkCmdBindVertexBuffers2EXT **must** have been called and not subsequently invalidated in the current command buffer prior to this draw command, and the pStrides parameter of vkCmdBindVertexBuffers2EXT **must** not be NULL
- VUID-vkCmdDraw-None-04914
  If there is a shader object bound to the VK_SHADER_STAGE_VERTEX_BIT stage or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VERTEX_INPUT_EXT dynamic state enabled then vkCmdSetVertexInputEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this draw command
- VUID-vkCmdDraw-Input-07939
  If there is a shader object bound to the VK_SHADER_STAGE_VERTEX_BIT stage or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VERTEX_INPUT_EXT dynamic state enabled then all variables with the Input storage class decorated with Location in the Vertex Execution Model OpEntryPoint **must** contain a location in

VkVertexInputAttributeDescription2EXT::location

- VUID-vkCmdDraw-Input-08734
  If there is a shader object bound to the VK_SHADER_STAGE_VERTEX_BIT stage or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VERTEX_INPUT_EXT dynamic state enabled and either legacyVertexAttributes is not enabled or the SPIR-V Type associated with a given Input variable of the corresponding Location in the Vertex Execution Model OpEntryPoint is 64-bit, then the numeric type associated with all Input variables of the corresponding Location in the Vertex Execution Model OpEntryPoint **must** be the same as VkVertexInputAttributeDescription2EXT::format
- VUID-vkCmdDraw-format-08936
  If there is a shader object bound to the VK_SHADER_STAGE_VERTEX_BIT stage or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VERTEX_INPUT_EXT dynamic state enabled and VkVertexInputAttributeDescription2EXT::format has a 64-bit component, then the scalar width associated with all Input variables of the corresponding Location in the Vertex Execution Model OpEntryPoint **must** be 64-bit
- VUID-vkCmdDraw-format-08937
  If there is a shader object bound to the VK_SHADER_STAGE_VERTEX_BIT stage or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VERTEX_INPUT_EXT dynamic state enabled and the scalar width associated with a Location decorated Input variable in the Vertex Execution Model OpEntryPoint is 64-bit, then the corresponding VkVertexInputAttributeDescription2EXT::format **must** have a 64-bit component
- VUID-vkCmdDraw-None-09203
  If there is a shader object bound to the VK_SHADER_STAGE_VERTEX_BIT stage or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VERTEX_INPUT_EXT dynamic state enabled and VkVertexInputAttributeDescription2EXT::format has a 64-bit component, then all Input variables at the corresponding Location in the Vertex Execution Model OpEntryPoint **must** not use components that are not present in the format
- VUID-vkCmdDraw-None-04875
  If there is a shader object bound to the VK_SHADER_STAGE_VERTEX_BIT stage and the most recent call to vkCmdSetPrimitiveTopologyEXT in the current command buffer set primitiveTopology to VK_PRIMITIVE_TOPOLOGY_PATCH_LIST, or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_PATCH_CONTROL_POINTS_EXT dynamic state enabled then vkCmdSetPatchControlPointsEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDraw-None-04879
  If there is a shader object bound to the VK_SHADER_STAGE_VERTEX_BIT stage or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_PRIMITIVE_RESTART_ENABLE dynamic state enabled then vkCmdSetPrimitiveRestartEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDraw-None-09637
  If the primitiveTopologyListRestart feature is not enabled, the topology is VK_PRIMITIVE_TOPOLOGY_POINT_LIST, VK_PRIMITIVE_TOPOLOGY_LINE_LIST, VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST, VK_PRIMITIVE_TOPOLOGY_LINE_LIST_WITH_ADJACENCY, or VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST_WITH_ADJACENCY, there is a shader object bound to the

VK_SHADER_STAGE_VERTEX_BIT stage or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_PRIMITIVE_RESTART_ENABLE dynamic state enabled then vkCmdSetPrimitiveRestartEnableEXT **must** be VK_FALSE

- VUID-vkCmdDraw-stage-06481
  The bound graphics pipeline **must** not have been created with the VkPipelineShaderStageCreateInfo::stage member of an element of VkGraphicsPipelineCreateInfo::pStages set to VK_SHADER_STAGE_TASK_BIT_EXT or VK_SHADER_STAGE_MESH_BIT_EXT

- VUID-vkCmdDraw-None-08885
  There **must** be no shader object bound to either of the VK_SHADER_STAGE_TASK_BIT_EXT or VK_SHADER_STAGE_MESH_BIT_EXT stages

- VUID-vkCmdDraw-pNext-09461
  If the bound graphics pipeline state was created with VkPipelineVertexInputDivisorStateCreateInfoKHR in the pNext chain of VkGraphicsPipelineCreateInfo::pVertexInputState, any member of VkPipelineVertexInputDivisorStateCreateInfoKHR::pVertexBindingDivisors has a value other than 1 in divisor, and VkPhysicalDeviceVertexAttributeDivisorPropertiesKHR::supportsNonZeroFirstInstance is VK_FALSE, then firstInstance **must** be 0

- VUID-vkCmdDraw-None-09462
  If shader objects are used for drawing or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VERTEX_INPUT_EXT dynamic state enabled, any member of the pVertexBindingDescriptions parameter to the vkCmdSetVertexInputEXT call that sets this dynamic state has a value other than 1 in divisor, and VkPhysicalDeviceVertexAttributeDivisorPropertiesKHR::supportsNonZeroFirstInstance is VK_FALSE, then firstInstance **must** be 0

## Valid Usage (Implicit)

- VUID-vkCmdDraw-commandBuffer-parameter
  commandBuffer **must** be a valid VkCommandBuffer handle

- VUID-vkCmdDraw-commandBuffer-recording
  commandBuffer **must** be in the recording state

- VUID-vkCmdDraw-commandBuffer-cmdpool
  The VkCommandPool that commandBuffer was allocated from **must** support graphics operations

- VUID-vkCmdDraw-renderpass
  This command **must** only be called inside of a render pass instance

- VUID-vkCmdDraw-videocoding
  This command **must** only be called outside of a video coding scope

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Inside | Outside | Graphics | Action |

To record an indexed draw, call:

```
// Provided by VK_VERSION_1_0
void vkCmdDrawIndexed(
    VkCommandBuffer                             commandBuffer,
    uint32_t                                    indexCount,
    uint32_t                                    instanceCount,
    uint32_t                                    firstIndex,
    int32_t                                     vertexOffset,
    uint32_t                                    firstInstance);
```

- `commandBuffer` is the command buffer into which the command is recorded.
- `indexCount` is the number of vertices to draw.
- `instanceCount` is the number of instances to draw.
- `firstIndex` is the base index within the index buffer.
- `vertexOffset` is the value added to the vertex index before indexing into the vertex buffer.
- `firstInstance` is the instance ID of the first instance to draw.

When the command is executed, primitives are assembled using the current primitive topology and `indexCount` vertices whose indices are retrieved from the index buffer. The index buffer is treated as an array of tightly packed unsigned integers of size defined by the vkCmdBindIndexBuffer2KHR::`indexType` or the vkCmdBindIndexBuffer::`indexType` parameter with which the buffer was bound.

The first vertex index is at an offset of `firstIndex` × `indexSize` + `offset` within the bound index buffer, where `offset` is the offset specified by `vkCmdBindIndexBuffer` or `vkCmdBindIndexBuffer2KHR`, and `indexSize` is the byte size of the type specified by `indexType`. Subsequent index values are retrieved from consecutive locations in the index buffer. Indices are first compared to the primitive restart value, then zero extended to 32 bits (if the `indexType` is `VK_INDEX_TYPE_UINT8_KHR` or `VK_INDEX_TYPE_UINT16`) and have `vertexOffset` added to them, before being supplied as the

vertexIndex value.

The primitives are drawn instanceCount times with instanceIndex starting with firstInstance and increasing sequentially for each instance. The assembled primitives execute the bound graphics pipeline.

### Valid Usage

- VUID-vkCmdDrawIndexed-magFilter-04553
  If a VkSampler created with magFilter or minFilter equal to VK_FILTER_LINEAR, reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE, and compareEnable equal to VK_FALSE is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT
- VUID-vkCmdDrawIndexed-magFilter-09598
  If a VkSampler created with magFilter or minFilter equal to VK_FILTER_LINEAR and reductionMode equal to either VK_SAMPLER_REDUCTION_MODE_MIN or VK_SAMPLER_REDUCTION_MODE_MAX is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT
- VUID-vkCmdDrawIndexed-mipmapMode-04770
  If a VkSampler created with mipmapMode equal to VK_SAMPLER_MIPMAP_MODE_LINEAR, reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE, and compareEnable equal to VK_FALSE is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT
- VUID-vkCmdDrawIndexed-mipmapMode-09599
  If a VkSampler created with mipmapMode equal to VK_SAMPLER_MIPMAP_MODE_LINEAR and reductionMode equal to either VK_SAMPLER_REDUCTION_MODE_MIN or VK_SAMPLER_REDUCTION_MODE_MAX is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT
- VUID-vkCmdDrawIndexed-unnormalizedCoordinates-09635
  If a VkSampler created with unnormalizedCoordinates equal to VK_TRUE is used to sample a VkImageView as a result of this command, then the image view's levelCount and layerCount **must** be 1
- VUID-vkCmdDrawIndexed-unnormalizedCoordinates-09636
  If a VkSampler created with unnormalizedCoordinates equal to VK_TRUE is used to sample a VkImageView as a result of this command, then the image view's viewType **must** be VK_IMAGE_VIEW_TYPE_1D or VK_IMAGE_VIEW_TYPE_2D
- VUID-vkCmdDrawIndexed-None-06479
  If a VkImageView is sampled with depth comparison, the image view's format features **must** contain VK_FORMAT_FEATURE_2_SAMPLED_IMAGE_DEPTH_COMPARISON_BIT
- VUID-vkCmdDrawIndexed-None-02691
  If a VkImageView is accessed using atomic operations as a result of this command, then

the image view's format features **must** contain VK_FORMAT_FEATURE_STORAGE_IMAGE_ATOMIC_BIT

- VUID-vkCmdDrawIndexed-None-07888
  If a VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER descriptor is accessed using atomic operations as a result of this command, then the storage texel buffer's format features **must** contain VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_ATOMIC_BIT

- VUID-vkCmdDrawIndexed-None-02692
  If a VkImageView is sampled with VK_FILTER_CUBIC_EXT as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_CUBIC_BIT_EXT

- VUID-vkCmdDrawIndexed-None-02693
  If the VK_EXT_filter_cubic extension is not enabled and any VkImageView is sampled with VK_FILTER_CUBIC_EXT as a result of this command, it **must** not have a VkImageViewType of VK_IMAGE_VIEW_TYPE_3D, VK_IMAGE_VIEW_TYPE_CUBE, or VK_IMAGE_VIEW_TYPE_CUBE_ARRAY

- VUID-vkCmdDrawIndexed-filterCubic-02694
  Any VkImageView being sampled with VK_FILTER_CUBIC_EXT as a result of this command **must** have a VkImageViewType and format that supports cubic filtering, as specified by VkFilterCubicImageViewImageFormatPropertiesEXT::filterCubic returned by vkGetPhysicalDeviceImageFormatProperties2

- VUID-vkCmdDrawIndexed-filterCubicMinmax-02695
  Any VkImageView being sampled with VK_FILTER_CUBIC_EXT with a reduction mode of either VK_SAMPLER_REDUCTION_MODE_MIN or VK_SAMPLER_REDUCTION_MODE_MAX as a result of this command **must** have a VkImageViewType and format that supports cubic filtering together with minmax filtering, as specified by VkFilterCubicImageViewImageFormatPropertiesEXT::filterCubicMinmax returned by vkGetPhysicalDeviceImageFormatProperties2

- VUID-vkCmdDrawIndexed-cubicRangeClamp-09212
  If the cubicRangeClamp feature is not enabled, then any VkImageView being sampled with VK_FILTER_CUBIC_EXT as a result of this command **must** not have a VkSamplerReductionModeCreateInfo::reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE_RANGECLAMP_QCOM

- VUID-vkCmdDrawIndexed-reductionMode-09213
  Any VkImageView being sampled with a VkSamplerReductionModeCreateInfo::reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE_RANGECLAMP_QCOM as a result of this command **must** sample with VK_FILTER_CUBIC_EXT

- VUID-vkCmdDrawIndexed-selectableCubicWeights-09214
  If the selectableCubicWeights feature is not enabled, then any VkImageView being sampled with VK_FILTER_CUBIC_EXT as a result of this command **must** have VkSamplerCubicWeightsCreateInfoQCOM::cubicWeights equal to VK_CUBIC_FILTER_WEIGHTS_CATMULL_ROM_QCOM

- VUID-vkCmdDrawIndexed-flags-02696
  Any VkImage created with a VkImageCreateInfo::flags containing VK_IMAGE_CREATE_CORNER_SAMPLED_BIT_NV sampled as a result of this command **must** only be sampled using a VkSamplerAddressMode of VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_EDGE

- VUID-vkCmdDrawIndexed-OpTypeImage-07027
  For any VkImageView being written as a storage image where the image format field of the `OpTypeImage` is `Unknown`, the view's format features **must** contain `VK_FORMAT_FEATURE_2_STORAGE_WRITE_WITHOUT_FORMAT_BIT`
- VUID-vkCmdDrawIndexed-OpTypeImage-07028
  For any VkImageView being read as a storage image where the image format field of the `OpTypeImage` is `Unknown`, the view's format features **must** contain `VK_FORMAT_FEATURE_2_STORAGE_READ_WITHOUT_FORMAT_BIT`
- VUID-vkCmdDrawIndexed-OpTypeImage-07029
  For any VkBufferView being written as a storage texel buffer where the image format field of the `OpTypeImage` is `Unknown`, the view's buffer features **must** contain `VK_FORMAT_FEATURE_2_STORAGE_WRITE_WITHOUT_FORMAT_BIT`
- VUID-vkCmdDrawIndexed-OpTypeImage-07030
  Any VkBufferView being read as a storage texel buffer where the image format field of the `OpTypeImage` is `Unknown` then the view's buffer features **must** contain `VK_FORMAT_FEATURE_2_STORAGE_READ_WITHOUT_FORMAT_BIT`
- VUID-vkCmdDrawIndexed-None-08600
  For each set *n* that is statically used by a bound shader, a descriptor set **must** have been bound to *n* at the same pipeline bind point, with a VkPipelineLayout that is compatible for set *n*, with the VkPipelineLayout used to create the current VkPipeline or the VkDescriptorSetLayout array used to create the current VkShaderEXT , as described in Pipeline Layout Compatibility
- VUID-vkCmdDrawIndexed-None-08601
  For each push constant that is statically used by a bound shader, a push constant value **must** have been set for the same pipeline bind point, with a VkPipelineLayout that is compatible for push constants, with the VkPipelineLayout used to create the current VkPipeline or the VkDescriptorSetLayout array used to create the current VkShaderEXT , as described in Pipeline Layout Compatibility
- VUID-vkCmdDrawIndexed-None-10068
  For each array of resources that is used by a bound shader, the indices used to access members of the array **must** be less than the descriptor count for the identified binding in the descriptor sets used by this command
- VUID-vkCmdDrawIndexed-maintenance4-08602
  If the `maintenance4` feature is not enabled, then for each push constant that is statically used by a bound shader, a push constant value **must** have been set for the same pipeline bind point, with a VkPipelineLayout that is compatible for push constants, with the VkPipelineLayout used to create the current VkPipeline or the VkDescriptorSetLayout and VkPushConstantRange arrays used to create the current VkShaderEXT , as described in Pipeline Layout Compatibility
- VUID-vkCmdDrawIndexed-None-08114
  Descriptors in each bound descriptor set, specified via vkCmdBindDescriptorSets, **must** be valid as described by descriptor validity if they are statically used by the VkPipeline bound to the pipeline bind point used by this command and the bound VkPipeline was not created with `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`

- VUID-vkCmdDrawIndexed-None-08115
  If the descriptors used by the VkPipeline bound to the pipeline bind point were specified via vkCmdBindDescriptorSets, the bound VkPipeline **must** have been created without VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT
- VUID-vkCmdDrawIndexed-None-08116
  Descriptors in bound descriptor buffers, specified via vkCmdSetDescriptorBufferOffsetsEXT, **must** be valid if they are dynamically used by the VkPipeline bound to the pipeline bind point used by this command and the bound VkPipeline was created with VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT
- VUID-vkCmdDrawIndexed-None-08604
  Descriptors in bound descriptor buffers, specified via vkCmdSetDescriptorBufferOffsetsEXT, **must** be valid if they are dynamically used by any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command
- VUID-vkCmdDrawIndexed-None-08117
  If the descriptors used by the VkPipeline bound to the pipeline bind point were specified via vkCmdSetDescriptorBufferOffsetsEXT, the bound VkPipeline **must** have been created with VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT
- VUID-vkCmdDrawIndexed-None-08119
  If a descriptor is dynamically used with a VkPipeline created with VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT, the descriptor memory **must** be resident
- VUID-vkCmdDrawIndexed-None-08605
  If a descriptor is dynamically used with a VkShaderEXT created with a VkDescriptorSetLayout that was created with VK_DESCRIPTOR_SET_LAYOUT_CREATE_DESCRIPTOR_BUFFER_BIT_EXT, the descriptor memory **must** be resident
- VUID-vkCmdDrawIndexed-None-08606
  If the shaderObject feature is not enabled, a valid pipeline **must** be bound to the pipeline bind point used by this command
- VUID-vkCmdDrawIndexed-None-08608
  If a pipeline is bound to the pipeline bind point used by this command, there **must** not have been any calls to dynamic state setting commands for any state specified statically in the VkPipeline object bound to the pipeline bind point used by this command, since that pipeline was bound
- VUID-vkCmdDrawIndexed-None-08609
  If the VkPipeline object bound to the pipeline bind point used by this command or any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a VkSampler object that uses unnormalized coordinates, that sampler **must** not be used to sample from any VkImage with a VkImageView of the type VK_IMAGE_VIEW_TYPE_3D, VK_IMAGE_VIEW_TYPE_CUBE, VK_IMAGE_VIEW_TYPE_1D_ARRAY, VK_IMAGE_VIEW_TYPE_2D_ARRAY or VK_IMAGE_VIEW_TYPE_CUBE_ARRAY, in any shader stage
- VUID-vkCmdDrawIndexed-None-08610
  If the VkPipeline object bound to the pipeline bind point used by this command or any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this

command accesses a VkSampler object that uses unnormalized coordinates, that sampler **must** not be used with any of the SPIR-V `OpImageSample*` or `OpImageSparseSample*` instructions with `ImplicitLod`, `Dref` or `Proj` in their name, in any shader stage

- VUID-vkCmdDrawIndexed-None-08611
  If the VkPipeline object bound to the pipeline bind point used by this command or any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a VkSampler object that uses unnormalized coordinates, that sampler **must** not be used with any of the SPIR-V `OpImageSample*` or `OpImageSparseSample*` instructions that includes a LOD bias or any offset values, in any shader stage

- VUID-vkCmdDrawIndexed-None-08607
  If the `shaderObject` is enabled, either a valid pipeline **must** be bound to the pipeline bind point used by this command, or a valid combination of valid and VK_NULL_HANDLE shader objects **must** be bound to every supported shader stage corresponding to the pipeline bind point used by this command

- VUID-vkCmdDrawIndexed-uniformBuffers-06935
  If any stage of the VkPipeline object bound to the pipeline bind point used by this command accesses a uniform buffer, and that stage was created without enabling either `VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_EXT` or `VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_2_EXT` for `uniformBuffers`, and the `robustBufferAccess` feature is not enabled, that stage **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point

- VUID-vkCmdDrawIndexed-None-08612
  If the `robustBufferAccess` feature is not enabled, and any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a uniform buffer, it **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point

- VUID-vkCmdDrawIndexed-storageBuffers-06936
  If any stage of the VkPipeline object bound to the pipeline bind point used by this command accesses a storage buffer, and that stage was created without enabling either `VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_EXT` or `VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_2_EXT` for `storageBuffers`, and the `robustBufferAccess` feature is not enabled, that stage **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point

- VUID-vkCmdDrawIndexed-None-08613
  If the `robustBufferAccess` feature is not enabled, and any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a storage buffer, it **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point

- VUID-vkCmdDrawIndexed-commandBuffer-02707
  If `commandBuffer` is an unprotected command buffer and `protectedNoFault` is not supported, any resource accessed by bound shaders **must** not be a protected resource

- VUID-vkCmdDrawIndexed-None-06550
  If a bound shader accesses a VkSampler or VkImageView object that enables sampler Y'C

$_{B}C_{R}$ conversion, that object **must** only be used with `OpImageSample*` or `OpImageSparseSample*` instructions

- VUID-vkCmdDrawIndexed-ConstOffset-06551
  If a bound shader accesses a VkSampler or VkImageView object that enables sampler Y'C$_{B}$C$_{R}$ conversion, that object **must** not use the `ConstOffset` and `Offset` operands
- VUID-vkCmdDrawIndexed-viewType-07752
  If a VkImageView is accessed as a result of this command, then the image view's `viewType` **must** match the `Dim` operand of the `OpTypeImage` as described in Compatibility Between SPIR-V Image Dimensions and Vulkan ImageView Types
- VUID-vkCmdDrawIndexed-format-07753
  If a VkImageView is accessed as a result of this command, then the numeric type of the image view's `format` and the `Sampled Type` operand of the `OpTypeImage` **must** match
- VUID-vkCmdDrawIndexed-OpImageWrite-08795
  If a VkImageView created with a format other than `VK_FORMAT_A8_UNORM_KHR` is accessed using `OpImageWrite` as a result of this command, then the `Type` of the `Texel` operand of that instruction **must** have at least as many components as the image view's format
- VUID-vkCmdDrawIndexed-OpImageWrite-08796
  If a VkImageView created with the format `VK_FORMAT_A8_UNORM_KHR` is accessed using `OpImageWrite` as a result of this command, then the `Type` of the `Texel` operand of that instruction **must** have four components
- VUID-vkCmdDrawIndexed-OpImageWrite-04469
  If a VkBufferView is accessed using `OpImageWrite` as a result of this command, then the `Type` of the `Texel` operand of that instruction **must** have at least as many components as the buffer view's format
- VUID-vkCmdDrawIndexed-SampledType-04470
  If a VkImageView with a VkFormat that has a 64-bit component width is accessed as a result of this command, the `SampledType` of the `OpTypeImage` operand of that instruction **must** have a `Width` of 64
- VUID-vkCmdDrawIndexed-SampledType-04471
  If a VkImageView with a VkFormat that has a component width less than 64-bit is accessed as a result of this command, the `SampledType` of the `OpTypeImage` operand of that instruction **must** have a `Width` of 32
- VUID-vkCmdDrawIndexed-SampledType-04472
  If a VkBufferView with a VkFormat that has a 64-bit component width is accessed as a result of this command, the `SampledType` of the `OpTypeImage` operand of that instruction **must** have a `Width` of 64
- VUID-vkCmdDrawIndexed-SampledType-04473
  If a VkBufferView with a VkFormat that has a component width less than 64-bit is accessed as a result of this command, the `SampledType` of the `OpTypeImage` operand of that instruction **must** have a `Width` of 32
- VUID-vkCmdDrawIndexed-sparseImageInt64Atomics-04474
  If the `sparseImageInt64Atomics` feature is not enabled, VkImage objects created with the `VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT` flag **must** not be accessed by atomic instructions

through an OpTypeImage with a SampledType with a Width of 64 by this command

- VUID-vkCmdDrawIndexed-sparseImageInt64Atomics-04475
  If the sparseImageInt64Atomics feature is not enabled, VkBuffer objects created with the VK_BUFFER_CREATE_SPARSE_RESIDENCY_BIT flag **must** not be accessed by atomic instructions through an OpTypeImage with a SampledType with a Width of 64 by this command

- VUID-vkCmdDrawIndexed-OpImageWeightedSampleQCOM-06971
  If OpImageWeightedSampleQCOM is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_WEIGHT_SAMPLED_IMAGE_BIT_QCOM

- VUID-vkCmdDrawIndexed-OpImageWeightedSampleQCOM-06972
  If OpImageWeightedSampleQCOM uses a VkImageView as a sample weight image as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_WEIGHT_IMAGE_BIT_QCOM

- VUID-vkCmdDrawIndexed-OpImageBoxFilterQCOM-06973
  If OpImageBoxFilterQCOM is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_BOX_FILTER_SAMPLED_BIT_QCOM

- VUID-vkCmdDrawIndexed-OpImageBlockMatchSSDQCOM-06974
  If OpImageBlockMatchSSDQCOM is used to read from an VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM

- VUID-vkCmdDrawIndexed-OpImageBlockMatchSADQCOM-06975
  If OpImageBlockMatchSADQCOM is used to read from an VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM

- VUID-vkCmdDrawIndexed-OpImageBlockMatchSADQCOM-06976
  If OpImageBlockMatchSADQCOM or OpImageBlockMatchSSDQCOM is used to read from a reference image as result of this command, then the specified reference coordinates **must** not fail integer texel coordinate validation

- VUID-vkCmdDrawIndexed-OpImageWeightedSampleQCOM-06977
  If OpImageWeightedSampleQCOM, OpImageBoxFilterQCOM, OpImageBlockMatchWindowSSDQCOM, OpImageBlockMatchWindowSADQCOM, OpImageBlockMatchGatherSSDQCOM, OpImageBlockMatchGatherSADQCOM, OpImageBlockMatchSSDQCOM, or OpImageBlockMatchSADQCOM uses a VkSampler as a result of this command, then the sampler **must** have been created with VK_SAMPLER_CREATE_IMAGE_PROCESSING_BIT_QCOM

- VUID-vkCmdDrawIndexed-OpImageWeightedSampleQCOM-06978
  If any command other than OpImageWeightedSampleQCOM, OpImageBoxFilterQCOM, OpImageBlockMatchWindowSSDQCOM, OpImageBlockMatchWindowSADQCOM, OpImageBlockMatchGatherSSDQCOM, OpImageBlockMatchGatherSADQCOM, OpImageBlockMatchSSDQCOM, or OpImageBlockMatchSADQCOM uses a VkSampler as a result of this command, then the sampler **must** not have been created with VK_SAMPLER_CREATE_IMAGE_PROCESSING_BIT_QCOM

- VUID-vkCmdDrawIndexed-OpImageBlockMatchWindow-09215
  If a OpImageBlockMatchWindow*QCOM or OpImageBlockMatchGather*QCOM instruction is used to

read from an VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM

- VUID-vkCmdDrawIndexed-OpImageBlockMatchWindow-09216
  If a OpImageBlockMatchWindow*QCOM or OpImageBlockMatchGather*QCOM instruction is used to read from an VkImageView as a result of this command, then the image view's format **must** be a single-component format
- VUID-vkCmdDrawIndexed-OpImageBlockMatchWindow-09217
  If a OpImageBlockMatchWindow*QCOM or OpImageBlockMatchGather*QCOM read from a reference image as result of this command, then the specified reference coordinates **must** not fail integer texel coordinate validation
- VUID-vkCmdDrawIndexed-None-07288
  Any shader invocation executed by this command **must** terminate
- VUID-vkCmdDrawIndexed-None-09600
  If a descriptor with type equal to any of VK_DESCRIPTOR_TYPE_SAMPLE_WEIGHT_IMAGE_QCOM, VK_DESCRIPTOR_TYPE_BLOCK_MATCH_IMAGE_QCOM, VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE, VK_DESCRIPTOR_TYPE_STORAGE_IMAGE, or VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT is accessed as a result of this command, the image subresource identified by that descriptor **must** be in the image layout identified when the descriptor was written
- VUID-vkCmdDrawIndexed-renderPass-02684
  The current render pass **must** be compatible with the renderPass member of the VkGraphicsPipelineCreateInfo structure specified when creating the VkPipeline bound to VK_PIPELINE_BIND_POINT_GRAPHICS
- VUID-vkCmdDrawIndexed-subpass-02685
  The subpass index of the current render pass **must** be equal to the subpass member of the VkGraphicsPipelineCreateInfo structure specified when creating the VkPipeline bound to VK_PIPELINE_BIND_POINT_GRAPHICS
- VUID-vkCmdDrawIndexed-None-07748
  If any shader statically accesses an input attachment, a valid descriptor **must** be bound to the pipeline via a descriptor set
- VUID-vkCmdDrawIndexed-OpTypeImage-07468
  If any shader executed by this pipeline accesses an OpTypeImage variable with a Dim operand of SubpassData, it **must** be decorated with an InputAttachmentIndex that corresponds to a valid input attachment in the current subpass
- VUID-vkCmdDrawIndexed-None-07469
  Input attachment views accessed in a subpass **must** be created with the same VkFormat as the corresponding subpass definition, and be created with a VkImageView that is compatible with the attachment referenced by the subpass' pInputAttachments [InputAttachmentIndex] in the currently bound VkFramebuffer as specified by Fragment Input Attachment Compatibility
- VUID-vkCmdDrawIndexed-pDepthInputAttachmentIndex-09595
  Input attachment views accessed in a dynamic render pass with a InputAttachmentIndex referenced by VkRenderingInputAttachmentIndexInfoKHR, or no InputAttachmentIndex if VkRenderingInputAttachmentIndexInfoKHR:pDepthInputAttachmentIndex or VkRenderingInputAttachmentIndexInfoKHR:pStencilInputAttachmentIndex are NULL, **must**

be created with a VkImageView that is compatible with the corresponding color, depth, or stencil attachment in VkRenderingInfo

- VUID-vkCmdDrawIndexed-pDepthInputAttachmentIndex-09596
  Input attachment views accessed in a dynamic render pass via a shader object **must** have an `InputAttachmentIndex` if both VkRenderingInputAttachmentIndexInfoKHR:`pDepthInputAttachmentIndex` and VkRenderingInputAttachmentIndexInfoKHR:`pStencilInputAttachmentIndex` are non-`NULL`

- VUID-vkCmdDrawIndexed-InputAttachmentIndex-09597
  If an input attachment view accessed in a dynamic render pass via a shader object has an `InputAttachmentIndex`, the `InputAttachmentIndex` **must** match an index in VkRenderingInputAttachmentIndexInfoKHR

- VUID-vkCmdDrawIndexed-None-06537
  Memory backing image subresources used as attachments in the current render pass **must** not be written in any way other than as an attachment by this command

- VUID-vkCmdDrawIndexed-None-09000
  If a color attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass, it is not in the `VK_IMAGE_LAYOUT_ATTACHMENT_FEEDBACK_LOOP_OPTIMAL_EXT` image layout, and either:

  - the `VK_PIPELINE_CREATE_COLOR_ATTACHMENT_FEEDBACK_LOOP_BIT_EXT` is set on the currently bound pipeline or
  - the last call to vkCmdSetAttachmentFeedbackLoopEnableEXT included `VK_IMAGE_ASPECT_COLOR_BIT` and
    - there is no currently bound graphics pipeline or
    - the currently bound graphics pipeline was created with `VK_DYNAMIC_STATE_ATTACHMENT_FEEDBACK_LOOP_ENABLE_EXT`

  it **must** not be accessed in any way other than as an attachment by this command

- VUID-vkCmdDrawIndexed-None-09001
  If a depth attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass, it is not in the `VK_IMAGE_LAYOUT_ATTACHMENT_FEEDBACK_LOOP_OPTIMAL_EXT` image layout, and either:

  - the `VK_PIPELINE_CREATE_DEPTH_STENCIL_ATTACHMENT_FEEDBACK_LOOP_BIT_EXT` is set on the currently bound pipeline or
  - the last call to vkCmdSetAttachmentFeedbackLoopEnableEXT included `VK_IMAGE_ASPECT_DEPTH_BIT` and
    - there is no currently bound graphics pipeline or
    - the currently bound graphics pipeline was created with `VK_DYNAMIC_STATE_ATTACHMENT_FEEDBACK_LOOP_ENABLE_EXT`

  it **must** not be accessed in any way other than as an attachment by this command

- VUID-vkCmdDrawIndexed-None-09002
  If a stencil attachment is written by any prior command in this subpass or by the load,

store, or resolve operations for this subpass, it is not in the VK_IMAGE_LAYOUT_ATTACHMENT_FEEDBACK_LOOP_OPTIMAL_EXT image layout, and either:

  - the VK_PIPELINE_CREATE_DEPTH_STENCIL_ATTACHMENT_FEEDBACK_LOOP_BIT_EXT is set on the currently bound pipeline or
  - the last call to vkCmdSetAttachmentFeedbackLoopEnableEXT included VK_IMAGE_ASPECT_STENCIL_BIT and
    - there is no currently bound graphics pipeline or
    - the currently bound graphics pipeline was created with VK_DYNAMIC_STATE_ATTACHMENT_FEEDBACK_LOOP_ENABLE_EXT

  it **must** not be accessed in any way other than as an attachment by this command

- VUID-vkCmdDrawIndexed-None-09003
  If an attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass, it **must** not be accessed in any way other than as an attachment, storage image, or sampled image by this command
- VUID-vkCmdDrawIndexed-None-06539
  If any previously recorded command in the current subpass accessed an image subresource used as an attachment in this subpass in any way other than as an attachment, this command **must** not write to that image subresource as an attachment
- VUID-vkCmdDrawIndexed-None-06886
  If the current render pass instance uses a depth/stencil attachment with a read-only layout for the depth aspect, depth writes **must** be disabled
- VUID-vkCmdDrawIndexed-None-06887
  If the current render pass instance uses a depth/stencil attachment with a read-only layout for the stencil aspect, both front and back writeMask are not zero, and stencil test is enabled, all stencil ops **must** be VK_STENCIL_OP_KEEP
- VUID-vkCmdDrawIndexed-None-07831
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT dynamic state enabled then vkCmdSetViewport **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexed-None-07832
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SCISSOR dynamic state enabled then vkCmdSetScissor **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexed-None-07833
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_WIDTH dynamic state enabled then vkCmdSetLineWidth **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexed-None-08617
  If a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetPolygonModeEXT in the current command buffer set polygonMode to

VK_POLYGON_MODE_LINE, vkCmdSetLineWidth **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexed-None-08618
  If a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetPrimitiveTopologyEXT in the current command buffer set primitiveTopology to any line topology, vkCmdSetLineWidth **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexed-None-08619
  If a shader object that outputs line primitives is bound to the VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT or VK_SHADER_STAGE_GEOMETRY_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, vkCmdSetLineWidth **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexed-None-07834
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_BIAS dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of depthBiasEnable is VK_TRUE, then vkCmdSetDepthBounds or vkCmdSetDepthBias2EXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexed-None-07835
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_BLEND_CONSTANTS dynamic state enabled then vkCmdSetBlendConstants **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexed-None-08621
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetColorBlendEnableEXT in the current command buffer set any element of pColorBlendEnables to VK_TRUE, and the most recent call to vkCmdSetColorBlendEquationEXT in the current command buffer set the same element of pColorBlendEquations to a VkColorBlendEquationEXT structure with any VkBlendFactor member with a value of VK_BLEND_FACTOR_CONSTANT_COLOR, VK_BLEND_FACTOR_ONE_MINUS_CONSTANT_COLOR, VK_BLEND_FACTOR_CONSTANT_ALPHA, or VK_BLEND_FACTOR_ONE_MINUS_CONSTANT_ALPHA, vkCmdSetBlendConstants **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexed-None-07836
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_BOUNDS dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of

depthBoundsTestEnable is VK_TRUE, then vkCmdSetDepthBounds **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexed-None-07837
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_STENCIL_COMPARE_MASK dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of stencilTestEnable is VK_TRUE, then vkCmdSetStencilCompareMask **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexed-None-07838
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_STENCIL_WRITE_MASK dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of stencilTestEnable is VK_TRUE, then vkCmdSetStencilWriteMask **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexed-None-07839
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_STENCIL_REFERENCE dynamic state enabled, the current value of and rasterizerDiscardEnable is VK_FALSE, the current value of stencilTestEnable is VK_TRUE, then vkCmdSetStencilReference **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexed-maxMultiviewInstanceIndex-02688
  If the draw is recorded in a render pass instance with multiview enabled, the maximum instance index **must** be less than or equal to VkPhysicalDeviceMultiviewProperties::maxMultiviewInstanceIndex
- VUID-vkCmdDrawIndexed-sampleLocationsEnable-02689
  If the bound graphics pipeline was created with VkPipelineSampleLocationsStateCreateInfoEXT::sampleLocationsEnable set to VK_TRUE and the current subpass has a depth/stencil attachment, then that attachment **must** have been created with the VK_IMAGE_CREATE_SAMPLE_LOCATIONS_COMPATIBLE_DEPTH_BIT_EXT bit set
- VUID-vkCmdDrawIndexed-None-06666
  If the VK_EXT_sample_locations extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of sampleLocationsEnable is VK_TRUE, then vkCmdSetSampleLocationsEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexed-None-07840
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_CULL_MODE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetCullModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this

drawing command

- VUID-vkCmdDrawIndexed-None-07841
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_FRONT_FACE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetFrontFaceEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexed-None-07843
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_TEST_ENABLE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, vkCmdSetDepthTestEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexed-None-07844
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_WRITE_ENABLE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetDepthWriteEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexed-None-07845
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_COMPARE_OP dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of depthTestEnable is VK_TRUE, then vkCmdSetDepthCompareOpEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexed-None-07846
  If the depthBounds feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_BOUNDS_TEST_ENABLE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetDepthBoundsTestEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexed-None-07847
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_STENCIL_TEST_ENABLE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetStencilTestEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexed-None-07848
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_STENCIL_OP dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, the current value of stencilTestEnable is VK_TRUE, then vkCmdSetStencilOpEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexed-viewportCount-03417
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT` dynamic state enabled, but not the `VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT` dynamic state enabled, then vkCmdSetViewportWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the `viewportCount` parameter of `vkCmdSetViewportWithCountEXT` **must** match the `VkPipelineViewportStateCreateInfo::scissorCount` of the pipeline
- VUID-vkCmdDrawIndexed-scissorCount-03418
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT` dynamic state enabled, but not the `VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT` dynamic state enabled, then vkCmdSetScissorWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the `scissorCount` parameter of `vkCmdSetScissorWithCountEXT` **must** match the `VkPipelineViewportStateCreateInfo::viewportCount` of the pipeline
- VUID-vkCmdDrawIndexed-viewportCount-03419
  If the bound graphics pipeline state was created with both the `VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT` and `VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT` dynamic states enabled then both vkCmdSetViewportWithCountEXT and vkCmdSetScissorWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the `viewportCount` parameter of `vkCmdSetViewportWithCountEXT` **must** match the `scissorCount` parameter of `vkCmdSetScissorWithCountEXT`
- VUID-vkCmdDrawIndexed-None-08635
  If a shader object is bound to any graphics stage, then both vkCmdSetViewportWithCountEXT and vkCmdSetScissorWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the `viewportCount` parameter of `vkCmdSetViewportWithCountEXT` **must** match the `scissorCount` parameter of `vkCmdSetScissorWithCountEXT`
- VUID-vkCmdDrawIndexed-viewportCount-04137
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT` dynamic state enabled, but not the `VK_DYNAMIC_STATE_VIEWPORT_W_SCALING_NV` dynamic state enabled, then the bound graphics pipeline **must** have been created with VkPipelineViewportWScalingStateCreateInfoNV `::viewportCount` greater or equal to the `viewportCount` parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawIndexed-viewportCount-04138
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT` and `VK_DYNAMIC_STATE_VIEWPORT_W_SCALING_NV` dynamic states enabled then the `viewportCount` parameter in the last call to vkCmdSetViewportWScalingNV **must** be greater than or equal to the `viewportCount` parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawIndexed-None-09232
  If the `VK_NV_clip_space_w_scaling` extension is enabled, and a shader object is bound to

any graphics stage, and the most recent call to vkCmdSetViewportWScalingEnableNV in the current command buffer set viewportWScalingEnable to VK_TRUE, then vkCmdSetViewportWScalingNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexed-None-08636
  If the VK_NV_clip_space_w_scaling extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetViewportWScalingEnableNV in the current command buffer set viewportWScalingEnable to VK_TRUE, then the viewportCount parameter in the last call to vkCmdSetViewportWScalingNV **must** be greater than or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT

- VUID-vkCmdDrawIndexed-viewportCount-04139
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, but not the VK_DYNAMIC_STATE_VIEWPORT_SHADING_RATE_PALETTE_NV dynamic state enabled, then the bound graphics pipeline **must** have been created with VkPipelineViewportShadingRateImageStateCreateInfoNV::viewportCount greater or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT

- VUID-vkCmdDrawIndexed-viewportCount-04140
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT and VK_DYNAMIC_STATE_VIEWPORT_SHADING_RATE_PALETTE_NV dynamic states enabled then the viewportCount parameter in the last call to vkCmdSetViewportShadingRatePaletteNV **must** be greater than or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT

- VUID-vkCmdDrawIndexed-shadingRateImage-09233
  If the shadingRateImage feature is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetCoarseSampleOrderNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexed-shadingRateImage-09234
  If the shadingRateImage feature is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetShadingRateImageEnableNV in the current command buffer set shadingRateImageEnable to VK_TRUE, then vkCmdSetViewportShadingRatePaletteNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexed-None-08637
  If the shadingRateImage feature is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetShadingRateImageEnableNV in the current command buffer set shadingRateImageEnable to VK_TRUE, then the viewportCount parameter in the last call to vkCmdSetViewportShadingRatePaletteNV **must** be greater than or equal to the

viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT

- VUID-vkCmdDrawIndexed-VkPipelineVieportCreateInfo-04141
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled and a VkPipelineViewportSwizzleStateCreateInfoNV structure chained from VkPipelineViewportStateCreateInfo, then the bound graphics pipeline **must** have been created with VkPipelineViewportSwizzleStateCreateInfoNV::viewportCount greater or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT

- VUID-vkCmdDrawIndexed-VkPipelineVieportCreateInfo-04142
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled and a VkPipelineViewportExclusiveScissorStateCreateInfoNV structure chained from VkPipelineViewportStateCreateInfo, then the bound graphics pipeline **must** have been created with VkPipelineViewportExclusiveScissorStateCreateInfoNV ::exclusiveScissorCount greater or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT

- VUID-vkCmdDrawIndexed-None-07878
  If the exclusiveScissor feature is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_EXCLUSIVE_SCISSOR_ENABLE_NV dynamic state enabled, then vkCmdSetExclusiveScissorEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexed-None-07879
  If the exclusiveScissor feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_EXCLUSIVE_SCISSOR_NV dynamic state enabled, and the most recent call to vkCmdSetExclusiveScissorEnableNV in the current command buffer set any element of pExclusiveScissorEnables to VK_TRUE, then vkCmdSetExclusiveScissorNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexed-None-04876
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_RASTERIZER_DISCARD_ENABLE dynamic state enabled, then vkCmdSetRasterizerDiscardEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexed-None-04877
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_BIAS_ENABLE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetDepthBiasEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexed-logicOp-04878
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_LOGIC_OP_EXT dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of

logicOpEnable is VK_TRUE, then vkCmdSetLogicOpEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexed-primitiveFragmentShadingRateWithMultipleViewports-04552
  If the primitiveFragmentShadingRateWithMultipleViewports limit is not supported, the bound graphics pipeline was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, and any of the shader stages of the bound graphics pipeline write to the PrimitiveShadingRateKHR built-in, then vkCmdSetViewportWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the viewportCount parameter of vkCmdSetViewportWithCountEXT **must** be 1
- VUID-vkCmdDrawIndexed-primitiveFragmentShadingRateWithMultipleViewports-08642
  If the primitiveFragmentShadingRateWithMultipleViewports limit is not supported, and any shader object bound to a graphics stage writes to the PrimitiveShadingRateKHR built-in, then vkCmdSetViewportWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the viewportCount parameter of vkCmdSetViewportWithCountEXT **must** be 1
- VUID-vkCmdDrawIndexed-blendEnable-04727
  If rasterization is not disabled in the bound graphics pipeline, then for each color attachment in the subpass, if the corresponding image view's format features do not contain VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT, then the blendEnable member of the corresponding element of the pAttachments member of pColorBlendState **must** be VK_FALSE
- VUID-vkCmdDrawIndexed-None-08643
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then for each color attachment in the render pass, if the corresponding image view's format features do not contain VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT, then the corresponding member of pColorBlendEnables in the most recent call to vkCmdSetColorBlendEnableEXT in the current command buffer that affected that attachment index **must** have been VK_FALSE
- VUID-vkCmdDrawIndexed-multisampledRenderToSingleSampled-07284
  If rasterization is not disabled in the bound graphics pipeline, and none of the following is enabled:
  - the VK_AMD_mixed_attachment_samples extension
  - the VK_NV_framebuffer_mixed_samples extension
  - the multisampledRenderToSingleSampled feature

  then rasterizationSamples for the currently bound graphics pipeline **must** be the same as the current subpass color and/or depth/stencil attachments
- VUID-vkCmdDrawIndexed-None-08644
  If a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and none of the following is enabled:
  - the VK_AMD_mixed_attachment_samples extension

- the VK_NV_framebuffer_mixed_samples extension
- the multisampledRenderToSingleSampled feature

then the most recent call to vkCmdSetRasterizationSamplesEXT in the current command buffer **must** have set rasterizationSamples to be the same as the number of samples for the current render pass color and/or depth/stencil attachments

- VUID-vkCmdDrawIndexed-None-08876
  If a shader object is bound to any graphics stage, the current render pass instance **must** have been begun with vkCmdBeginRenderingKHR
- VUID-vkCmdDrawIndexed-imageView-06172
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the imageView member of pDepthAttachment is not VK_NULL_HANDLE, and the layout member of pDepthAttachment is VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL, this command **must** not write any values to the depth attachment
- VUID-vkCmdDrawIndexed-imageView-06173
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the imageView member of pStencilAttachment is not VK_NULL_HANDLE, and the layout member of pStencilAttachment is VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL, this command **must** not write any values to the stencil attachment
- VUID-vkCmdDrawIndexed-imageView-06174
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the imageView member of pDepthAttachment is not VK_NULL_HANDLE, and the layout member of pDepthAttachment is VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_STENCIL_ATTACHMENT_OPTIMAL, this command **must** not write any values to the depth attachment
- VUID-vkCmdDrawIndexed-imageView-06175
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the imageView member of pStencilAttachment is not VK_NULL_HANDLE, and the layout member of pStencilAttachment is VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_STENCIL_READ_ONLY_OPTIMAL, this command **must** not write any values to the stencil attachment
- VUID-vkCmdDrawIndexed-imageView-06176
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the imageView member of pDepthAttachment is not VK_NULL_HANDLE, and the layout member of pDepthAttachment is VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL, this command **must** not write any values to the depth attachment
- VUID-vkCmdDrawIndexed-imageView-06177
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the imageView member of pStencilAttachment is not VK_NULL_HANDLE, and the layout member of pStencilAttachment is VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL, this command **must** not write any values to the stencil attachment
- VUID-vkCmdDrawIndexed-viewMask-06178
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the currently bound graphics pipeline **must** have been created with a VkPipelineRenderingCreateInfo::viewMask equal to VkRenderingInfo::viewMask

- VUID-vkCmdDrawIndexed-colorAttachmentCount-06179
  If the `dynamicRenderingUnusedAttachments` feature is not enabled and the current render pass instance was begun with vkCmdBeginRenderingKHR, the currently bound graphics pipeline **must** have been created with a VkPipelineRenderingCreateInfo::`colorAttachmentCount` equal to VkRenderingInfo::`colorAttachmentCount`
- VUID-vkCmdDrawIndexed-dynamicRenderingUnusedAttachments-08910
  If the `dynamicRenderingUnusedAttachments` feature is not enabled, and the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingInfo::`colorAttachmentCount` greater than `0`, then each element of the VkRenderingInfo::`pColorAttachments` array with an `imageView` not equal to VK_NULL_HANDLE **must** have been created with a VkFormat equal to the corresponding element of VkPipelineRenderingCreateInfo::`pColorAttachmentFormats` used to create the currently bound graphics pipeline
- VUID-vkCmdDrawIndexed-dynamicRenderingUnusedAttachments-08912
  If the `dynamicRenderingUnusedAttachments` feature is not enabled, and the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingInfo::`colorAttachmentCount` greater than `0`, then each element of the VkRenderingInfo::`pColorAttachments` array with an `imageView` equal to VK_NULL_HANDLE **must** have the corresponding element of VkPipelineRenderingCreateInfo::`pColorAttachmentFormats` used to create the currently bound pipeline equal to `VK_FORMAT_UNDEFINED`
- VUID-vkCmdDrawIndexed-dynamicRenderingUnusedAttachments-08911
  If the `dynamicRenderingUnusedAttachments` feature is enabled, and the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingInfo::`colorAttachmentCount` greater than `0`, then each element of the VkRenderingInfo::`pColorAttachments` array with an `imageView` not equal to VK_NULL_HANDLE **must** have been created with a VkFormat equal to the corresponding element of VkPipelineRenderingCreateInfo::`pColorAttachmentFormats` used to create the currently bound graphics pipeline, or the corresponding element of VkPipelineRenderingCreateInfo::`pColorAttachmentFormats`, if it exists, **must** be `VK_FORMAT_UNDEFINED`
- VUID-vkCmdDrawIndexed-colorAttachmentCount-09362
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, with a VkRenderingInfo::`colorAttachmentCount` equal to `1`, there is no shader object bound to any graphics stage, and a color attachment with a resolve mode of `VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID`, each element of the VkRenderingInfo::`pColorAttachments` array with a `resolveImageView` not equal to VK_NULL_HANDLE **must** have been created with an image created with a VkExternalFormatANDROID::`externalFormat` value equal to the VkExternalFormatANDROID::`externalFormat` value used to create the currently bound graphics pipeline
- VUID-vkCmdDrawIndexed-None-09363
  If there is no shader object bound to any graphics stage, the current render pass instance was begun with vkCmdBeginRenderingKHR and a VkRenderingInfo::`colorAttachmentCount` equal to `1`, and a color attachment with a resolve mode of `VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID`, each element of the VkRenderingInfo::`pColorAttachments` array with a `imageView` not equal to

VK_NULL_HANDLE **must** have been created with an image created with a VkExternalFormatANDROID::externalFormat value equal to the VkExternalFormatANDROID::externalFormat value used to create the currently bound graphics pipeline

- VUID-vkCmdDrawIndexed-None-09364
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is no shader object bound to any graphics stage, and the currently bound graphics pipeline was created with a non-zero VkExternalFormatANDROID::externalFormat value and with the VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT dynamic state enabled, then vkCmdSetColorBlendEnableEXT **must** have set the blend enable to VK_FALSE prior to this drawing command
- VUID-vkCmdDrawIndexed-None-09365
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is no shader object bound to any graphics stage, and the currently bound graphics pipeline was created with a non-zero VkExternalFormatANDROID::externalFormat value and with the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT dynamic state enabled, then vkCmdSetRasterizationSamplesEXT **must** have set rasterizationSamples to VK_SAMPLE_COUNT_1_BIT prior to this drawing command
- VUID-vkCmdDrawIndexed-None-09366
  If there is a shader object bound to any graphics stage, and the current render pass includes a color attachment that uses the VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID resolve mode, then vkCmdSetColorBlendEnableEXT **must** have set blend enable to VK_FALSE prior to this drawing command
- VUID-vkCmdDrawIndexed-rasterizationSamples-09367
  If there is a shader object bound to any graphics stage, and the current render pass includes a color attachment that uses the VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID resolve mode, then vkCmdSetRasterizationSamplesEXT **must** have set rasterizationSamples to VK_SAMPLE_COUNT_1_BIT prior to this drawing command
- VUID-vkCmdDrawIndexed-None-09368
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is no shader object bound to any graphics stage, and the currently bound graphics pipeline was created with a non-zero VkExternalFormatANDROID::externalFormat value and with the VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR dynamic state enabled, then vkCmdSetFragmentShadingRateKHR **must** have set pFragmentSize->width to 1 prior to this drawing command
- VUID-vkCmdDrawIndexed-None-09369
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is no shader object bound to any graphics stage, and the currently bound graphics pipeline was created with a non-zero VkExternalFormatANDROID::externalFormat value and with the VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR dynamic state enabled, then vkCmdSetFragmentShadingRateKHR **must** have set pFragmentSize->height to 1 prior to this drawing command
- VUID-vkCmdDrawIndexed-pFragmentSize-09370

If there is a shader object bound to any graphics stage, and the current render pass includes a color attachment that uses the VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID resolve mode, then vkCmdSetFragmentShadingRateKHR **must** have set pFragmentSize->width to 1 prior to this drawing command

- VUID-vkCmdDrawIndexed-pFragmentSize-09371
  If there is a shader object bound to any graphics stage, and the current render pass includes a color attachment that uses the VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID resolve mode, then vkCmdSetFragmentShadingRateKHR **must** have set pFragmentSize->height to 1 prior to this drawing command
- VUID-vkCmdDrawIndexed-None-07749
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_WRITE_ENABLE_EXT dynamic state enabled then vkCmdSetColorWriteEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexed-None-08646
  If the colorWriteEnable feature is enabled on the device, and a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetColorWriteEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexed-attachmentCount-07750
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_WRITE_ENABLE_EXT dynamic state enabled then the attachmentCount parameter of vkCmdSetColorWriteEnableEXT **must** be greater than or equal to the VkPipelineColorBlendStateCreateInfo::attachmentCount of the currently bound graphics pipeline
- VUID-vkCmdDrawIndexed-None-08647
  If the colorWriteEnable feature is enabled on the device, and a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then the attachmentCount parameter of most recent call to vkCmdSetColorWriteEnableEXT in the current command buffer **must** be greater than or equal to the number of color attachments in the current render pass instance
- VUID-vkCmdDrawIndexed-None-07751
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_DISCARD_RECTANGLE_EXT dynamic state enabled then vkCmdSetDiscardRectangleEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command for each discard rectangle in VkPipelineDiscardRectangleStateCreateInfoEXT::discardRectangleCount
- VUID-vkCmdDrawIndexed-None-07880
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_DISCARD_RECTANGLE_ENABLE_EXT dynamic state enabled then

vkCmdSetDiscardRectangleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexed-rasterizerDiscardEnable-09236
  If the VK_EXT_discard_rectangles extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetDiscardRectangleEnableEXT in the current command buffer set discardRectangleEnable to VK_TRUE, then vkCmdSetDiscardRectangleEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexed-None-08648
  If the VK_EXT_discard_rectangles extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetDiscardRectangleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexed-None-07881
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_DISCARD_RECTANGLE_MODE_EXT dynamic state enabled then vkCmdSetDiscardRectangleModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexed-None-08649
  If the VK_EXT_discard_rectangles extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetDiscardRectangleEnableEXT in the current command buffer set discardRectangleEnable to VK_TRUE, then vkCmdSetDiscardRectangleModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexed-dynamicRenderingUnusedAttachments-08913
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is not enabled, and VkRenderingInfo::pDepthAttachment->imageView was VK_NULL_HANDLE, the value of VkPipelineRenderingCreateInfo::depthAttachmentFormat used to create the currently bound graphics pipeline **must** be equal to VK_FORMAT_UNDEFINED

- VUID-vkCmdDrawIndexed-dynamicRenderingUnusedAttachments-08914
  If current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is not enabled, and VkRenderingInfo::pDepthAttachment->imageView was not VK_NULL_HANDLE, the value of VkPipelineRenderingCreateInfo::depthAttachmentFormat used to create the currently bound graphics pipeline **must** be equal to the VkFormat used to create VkRenderingInfo::pDepthAttachment->imageView

- VUID-vkCmdDrawIndexed-dynamicRenderingUnusedAttachments-08915
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is enabled, VkRenderingInfo

::pDepthAttachment->imageView was not VK_NULL_HANDLE, and the value of VkPipelineRenderingCreateInfo::depthAttachmentFormat used to create the currently bound graphics pipeline was not equal to the VkFormat used to create VkRenderingInfo::pDepthAttachment->imageView, the value of the format **must** be VK_FORMAT_UNDEFINED

- VUID-vkCmdDrawIndexed-dynamicRenderingUnusedAttachments-08916
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is not enabled, and VkRenderingInfo::pStencilAttachment->imageView was VK_NULL_HANDLE, the value of VkPipelineRenderingCreateInfo::stencilAttachmentFormat used to create the currently bound graphics pipeline **must** be equal to VK_FORMAT_UNDEFINED
- VUID-vkCmdDrawIndexed-dynamicRenderingUnusedAttachments-08917
  If current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is not enabled, and VkRenderingInfo::pStencilAttachment->imageView was not VK_NULL_HANDLE, the value of VkPipelineRenderingCreateInfo::stencilAttachmentFormat used to create the currently bound graphics pipeline **must** be equal to the VkFormat used to create VkRenderingInfo::pStencilAttachment->imageView
- VUID-vkCmdDrawIndexed-dynamicRenderingUnusedAttachments-08918
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is enabled, VkRenderingInfo::pStencilAttachment->imageView was not VK_NULL_HANDLE, and the value of VkPipelineRenderingCreateInfo::stencilAttachmentFormat used to create the currently bound graphics pipeline was not equal to the VkFormat used to create VkRenderingInfo::pStencilAttachment->imageView, the value of the format **must** be VK_FORMAT_UNDEFINED
- VUID-vkCmdDrawIndexed-imageView-06183
  If the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingFragmentShadingRateAttachmentInfoKHR::imageView was not VK_NULL_HANDLE, the currently bound graphics pipeline **must** have been created with VK_PIPELINE_CREATE_RENDERING_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR
- VUID-vkCmdDrawIndexed-imageView-06184
  If the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingFragmentDensityMapAttachmentInfoEXT::imageView was not VK_NULL_HANDLE, the currently bound graphics pipeline **must** have been created with VK_PIPELINE_CREATE_RENDERING_FRAGMENT_DENSITY_MAP_ATTACHMENT_BIT_EXT
- VUID-vkCmdDrawIndexed-colorAttachmentCount-06185
  If the currently bound pipeline was created with a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and the current render pass instance was begun with vkCmdBeginRenderingKHR with a VkRenderingInfo::colorAttachmentCount parameter greater than 0, then each element of the VkRenderingInfo::pColorAttachments array with a imageView not equal to VK_NULL_HANDLE **must** have been created with a sample count equal to the corresponding element of the pColorAttachmentSamples member of VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV used to create the currently bound graphics pipeline
- VUID-vkCmdDrawIndexed-pDepthAttachment-06186
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the

currently bound pipeline was created with a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and VkRenderingInfo::pDepthAttachment->imageView was not VK_NULL_HANDLE, the value of the depthStencilAttachmentSamples member of VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV used to create the currently bound graphics pipeline **must** be equal to the sample count used to create VkRenderingInfo::pDepthAttachment->imageView

- VUID-vkCmdDrawIndexed-pStencilAttachment-06187
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the currently bound pipeline was created with a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and VkRenderingInfo::pStencilAttachment->imageView was not VK_NULL_HANDLE, the value of the depthStencilAttachmentSamples member of VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV used to create the currently bound graphics pipeline **must** be equal to the sample count used to create VkRenderingInfo::pStencilAttachment->imageView
- VUID-vkCmdDrawIndexed-multisampledRenderToSingleSampled-07285
  If the currently bound pipeline was created without a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and the multisampledRenderToSingleSampled feature is not enabled, and the current render pass instance was begun with vkCmdBeginRenderingKHR with a VkRenderingInfo::colorAttachmentCount parameter greater than 0, then each element of the VkRenderingInfo::pColorAttachments array with a imageView not equal to VK_NULL_HANDLE **must** have been created with a sample count equal to the value of rasterizationSamples for the currently bound graphics pipeline
- VUID-vkCmdDrawIndexed-multisampledRenderToSingleSampled-07286
  If the currently bound pipeline was created without a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and the multisampledRenderToSingleSampled feature is not enabled, and VkRenderingInfo::pDepthAttachment->imageView was not VK_NULL_HANDLE, the value of rasterizationSamples for the currently bound graphics pipeline **must** be equal to the sample count used to create VkRenderingInfo::pDepthAttachment->imageView
- VUID-vkCmdDrawIndexed-multisampledRenderToSingleSampled-07287
  If the currently bound pipeline was created without a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and the multisampledRenderToSingleSampled feature is not enabled, and VkRenderingInfo::pStencilAttachment->imageView was not VK_NULL_HANDLE, the value of rasterizationSamples for the currently bound graphics pipeline **must** be equal to the sample count used to create VkRenderingInfo::pStencilAttachment->imageView
- VUID-vkCmdDrawIndexed-pNext-07935
  If this command has been called inside a render pass instance started with vkCmdBeginRenderingKHR, and the pNext chain of VkRenderingInfo includes a VkMultisampledRenderToSingleSampledInfoEXT structure with multisampledRenderToSingleSampledEnable equal to VK_TRUE, then the value of rasterizationSamples for the currently bound graphics pipeline **must** be equal to VkMultisampledRenderToSingleSampledInfoEXT::rasterizationSamples
- VUID-vkCmdDrawIndexed-renderPass-06198

If the current render pass instance was begun with vkCmdBeginRenderingKHR, the currently bound pipeline **must** have been created with a VkGraphicsPipelineCreateInfo::`renderPass` equal to VK_NULL_HANDLE

- VUID-vkCmdDrawIndexed-pColorAttachments-08963
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is a graphics pipeline bound with a fragment shader that statically writes to a color attachment, the color write mask is not zero, color writes are enabled, and the corresponding element of the VkRenderingInfo::`pColorAttachments->imageView` was not VK_NULL_HANDLE, then the corresponding element of VkPipelineRenderingCreateInfo::`pColorAttachmentFormats` used to create the pipeline **must** not be `VK_FORMAT_UNDEFINED`

- VUID-vkCmdDrawIndexed-pDepthAttachment-08964
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is a graphics pipeline bound, depth test is enabled, depth write is enabled, and the VkRenderingInfo::`pDepthAttachment->imageView` was not VK_NULL_HANDLE, then the VkPipelineRenderingCreateInfo::`depthAttachmentFormat` used to create the pipeline **must** not be `VK_FORMAT_UNDEFINED`

- VUID-vkCmdDrawIndexed-pStencilAttachment-08965
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is a graphics pipeline bound, stencil test is enabled and the VkRenderingInfo::`pStencilAttachment->imageView` was not VK_NULL_HANDLE, then the VkPipelineRenderingCreateInfo::`stencilAttachmentFormat` used to create the pipeline **must** not be `VK_FORMAT_UNDEFINED`

- VUID-vkCmdDrawIndexed-primitivesGeneratedQueryWithRasterizerDiscard-06708
  If the `primitivesGeneratedQueryWithRasterizerDiscard` feature is not enabled and the `VK_QUERY_TYPE_PRIMITIVES_GENERATED_EXT` query is active, rasterization discard **must** not be enabled

- VUID-vkCmdDrawIndexed-primitivesGeneratedQueryWithNonZeroStreams-06709
  If the `primitivesGeneratedQueryWithNonZeroStreams` feature is not enabled and the `VK_QUERY_TYPE_PRIMITIVES_GENERATED_EXT` query is active, the bound graphics pipeline **must** not have been created with a non-zero value in `VkPipelineRasterizationStateStreamCreateInfoEXT`::`rasterizationStream`

- VUID-vkCmdDrawIndexed-None-07619
  If a shader object is bound to the `VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT` stage or a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_TESSELLATION_DOMAIN_ORIGIN_EXT` dynamic state enabled, then vkCmdSetTessellationDomainOriginEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexed-None-07620
  If the `depthClamp` feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_DEPTH_CLAMP_ENABLE_EXT` dynamic state enabled, and the current value of `rasterizerDiscardEnable` is `VK_FALSE`, then vkCmdSetDepthClampEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexed-None-07621

If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_POLYGON_MODE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetPolygonModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexed-None-07622
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetRasterizationSamplesEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexed-None-07623
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_SAMPLE_MASK_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetSampleMaskEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexed-alphaToCoverageEnable-08919
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_ALPHA_TO_COVERAGE_ENABLE_EXT dynamic state enabled, and alphaToCoverageEnable was VK_TRUE in the last call to vkCmdSetAlphaToCoverageEnableEXT, then the Fragment Output Interface **must** contain a variable for the alpha Component word in Location 0 at Index 0
- VUID-vkCmdDrawIndexed-alphaToCoverageEnable-08920
  If a shader object is bound to any graphics stage, and the most recent call to vkCmdSetAlphaToCoverageEnableEXT in the current command buffer set alphaToCoverageEnable to VK_TRUE, then the Fragment Output Interface **must** contain a variable for the alpha Component word in Location 0 at Index 0
- VUID-vkCmdDrawIndexed-None-07624
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_ALPHA_TO_COVERAGE_ENABLE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetAlphaToCoverageEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexed-None-07625
  If the alphaToOne feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_ALPHA_TO_ONE_ENABLE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetAlphaToOneEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexed-None-07626
  If the logicOp feature is enabled, a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_LOGIC_OP_ENABLE_EXT dynamic state enabled, and the current

value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetLogicOpEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexed-None-07627
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT dynamic state enabled then vkCmdSetColorBlendEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexed-None-08657
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and both the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE and there are color attachments bound, then vkCmdSetColorBlendEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexed-None-07628
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_EQUATION_EXT dynamic state enabled then vkCmdSetColorBlendEquationEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexed-None-08658
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetColorBlendEnableEXT for any attachment set that attachment's value in pColorBlendEnables to VK_TRUE, then vkCmdSetColorBlendEquationEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexed-None-07629
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_WRITE_MASK_EXT dynamic state enabled then vkCmdSetColorWriteMaskEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexed-None-08659
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and both the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE and there are color attachments bound, then vkCmdSetColorWriteMaskEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexed-None-07630
  If the geometryStreams feature is enabled, and a shader object is bound to the VK_SHADER_STAGE_GEOMETRY_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_RASTERIZATION_STREAM_EXT dynamic state enabled, then vkCmdSetRasterizationStreamEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexed-None-07631

If the VK_EXT_conservative_rasterization extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_CONSERVATIVE_RASTERIZATION_MODE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetConservativeRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexed-None-07632
  If the VK_EXT_conservative_rasterization extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_EXTRA_PRIMITIVE_OVERESTIMATION_SIZE_EXT dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of conservativeRasterizationMode is VK_CONSERVATIVE_RASTERIZATION_MODE_OVERESTIMATE_EXT, then vkCmdSetExtraPrimitiveOverestimationSizeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexed-None-07633
  If the depthClipEnable feature is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_CLIP_ENABLE_EXT dynamic state, then vkCmdSetDepthClipEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexed-None-07634
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT dynamic state enabled then vkCmdSetSampleLocationsEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexed-None-08664
  If the VK_EXT_sample_locations extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetSampleLocationsEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexed-None-07635
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_ADVANCED_EXT dynamic state enabled then vkCmdSetColorBlendAdvancedEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexed-rasterizerDiscardEnable-09416
  If the VK_EXT_blend_operation_advanced extension is enabled, and a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then at least one of vkCmdSetColorBlendEquationEXT and vkCmdSetColorBlendAdvancedEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexed-None-07636

If the VK_EXT_provoking_vertex extension is enabled, a shader object is bound to the VK_SHADER_STAGE_VERTEX_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_PROVOKING_VERTEX_MODE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetProvokingVertexModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexed-None-07637
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT dynamic state enabled then vkCmdSetLineRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexed-None-08666
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetPolygonModeEXT in the current command buffer set polygonMode to VK_POLYGON_MODE_LINE, then vkCmdSetLineRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexed-None-08667
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object is bound to the VK_SHADER_STAGE_VERTEX_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetPrimitiveTopologyEXT in the current command buffer set primitiveTopology to any line topology, then vkCmdSetLineRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexed-None-08668
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object that outputs line primitives is bound to the VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT or VK_SHADER_STAGE_GEOMETRY_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetLineRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexed-None-07638
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT dynamic state enabled then vkCmdSetLineStippleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexed-None-08669
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to

vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetPolygonModeEXT in the current command buffer set polygonMode to VK_POLYGON_MODE_LINE, then vkCmdSetLineStippleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexed-None-08670
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object is bound to the VK_SHADER_STAGE_VERTEX_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetPrimitiveTopologyEXT in the current command buffer set primitiveTopology to any line topology, then vkCmdSetLineStippleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexed-None-08671
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object that outputs line primitives is bound to the VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT or VK_SHADER_STAGE_GEOMETRY_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetLineStippleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexed-None-07849
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_KHR dynamic state enabled then vkCmdSetLineStippleKHR **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexed-None-08672
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetLineStippleEnableEXT in the current command buffer set stippledLineEnable to VK_TRUE, then vkCmdSetLineStippleEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexed-None-07639
  If the depthClipControl feature is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_CLIP_NEGATIVE_ONE_TO_ONE_EXT dynamic state enabled, then vkCmdSetDepthClipNegativeOneToOneEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexed-None-07640
  If the VK_NV_clip_space_w_scaling extension is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_VIEWPORT_W_SCALING_ENABLE_NV dynamic state enabled, then

vkCmdSetViewportWScalingEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexed-None-07641
  If the VK_NV_viewport_swizzle extension is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_VIEWPORT_SWIZZLE_NV dynamic state enabled, then vkCmdSetViewportSwizzleNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexed-None-07642
  If the VK_NV_fragment_coverage_to_color extension is enabled, a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_TO_COLOR_ENABLE_NV dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetCoverageToColorEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexed-None-07643
  If the VK_NV_fragment_coverage_to_color extension is enabled, a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_TO_COLOR_LOCATION_NV dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of coverageToColorEnable is VK_TRUE, then vkCmdSetCoverageToColorLocationNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexed-None-07644
  If the VK_NV_framebuffer_mixed_samples extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_MODULATION_MODE_NV dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetCoverageModulationModeNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexed-None-07645
  If the VK_NV_framebuffer_mixed_samples extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_MODULATION_TABLE_ENABLE_NV dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of coverageModulationMode is any value other than VK_COVERAGE_MODULATION_MODE_NONE_NV, then vkCmdSetCoverageModulationTableEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexed-None-07646
  If the VK_NV_framebuffer_mixed_samples extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_MODULATION_TABLE_NV dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of coverageModulationTableEnable is VK_TRUE, then vkCmdSetCoverageModulationTableNV **must** have been called and not subsequently invalidated in the current command buffer

prior to this drawing command

- VUID-vkCmdDrawIndexed-None-07647
  If the `shadingRateImage` feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_SHADING_RATE_IMAGE_ENABLE_NV` dynamic state enabled, and the current value of `rasterizerDiscardEnable` is `VK_FALSE`, then vkCmdSetShadingRateImageEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexed-pipelineFragmentShadingRate-09238
  If the `pipelineFragmentShadingRate` feature is enabled, a shader object is bound to the `VK_SHADER_STAGE_FRAGMENT_BIT` stage or a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR` dynamic state enabled, and the current value of `rasterizerDiscardEnable` is `VK_FALSE`, then vkCmdSetFragmentShadingRateKHR **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexed-None-07648
  If the `representativeFragmentTest` feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_REPRESENTATIVE_FRAGMENT_TEST_ENABLE_NV` dynamic state enabled, and the current value of `rasterizerDiscardEnable` is `VK_FALSE`, then vkCmdSetRepresentativeFragmentTestEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexed-None-07649
  If the `coverageReductionMode` feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_COVERAGE_REDUCTION_MODE_NV` dynamic state enabled, and the current value of `rasterizerDiscardEnable` is `VK_FALSE`, then vkCmdSetCoverageReductionModeNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexed-pColorBlendEnables-07470
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT` state enabled and the last call to vkCmdSetColorBlendEnableEXT set `pColorBlendEnables` for any attachment to `VK_TRUE`, then for those attachments in the subpass the corresponding image view's format features **must** contain `VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT`

- VUID-vkCmdDrawIndexed-rasterizationSamples-07471
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT` state enabled, and the current subpass does not use any color and/or depth/stencil attachments, then the `rasterizationSamples` in the last call to vkCmdSetRasterizationSamplesEXT **must** follow the rules for a zero-attachment subpass

- VUID-vkCmdDrawIndexed-samples-07472
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_SAMPLE_MASK_EXT` state enabled and the `VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT` state disabled, then the `samples` parameter in

the last call to vkCmdSetSampleMaskEXT **must** be greater or equal to the VkPipelineMultisampleStateCreateInfo::rasterizationSamples parameter used to create the bound graphics pipeline

- VUID-vkCmdDrawIndexed-samples-07473
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_MASK_EXT state and VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT states enabled, then the samples parameter in the last call to vkCmdSetSampleMaskEXT **must** be greater or equal to the rasterizationSamples parameter in the last call to vkCmdSetRasterizationSamplesEXT

- VUID-vkCmdDrawIndexed-rasterizationSamples-07474
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, and neither the VK_AMD_mixed_attachment_samples nor the VK_NV_framebuffer_mixed_samples extensions are enabled, then the rasterizationSamples in the last call to vkCmdSetRasterizationSamplesEXT **must** be the same as the current subpass color and/or depth/stencil attachments

- VUID-vkCmdDrawIndexed-None-09211
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, or a shader object is bound to any graphics stage, and the current render pass instance includes a VkMultisampledRenderToSingleSampledInfoEXT structure with multisampledRenderToSingleSampledEnable equal to VK_TRUE, then the rasterizationSamples in the last call to vkCmdSetRasterizationSamplesEXT **must** be the same as the rasterizationSamples member of that structure

- VUID-vkCmdDrawIndexed-firstAttachment-07476
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT dynamic state enabled then vkCmdSetColorBlendEnableEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the firstAttachment and attachmentCount parameters of vkCmdSetColorBlendEnableEXT calls **must** specify an enable for all active color attachments in the current subpass

- VUID-vkCmdDrawIndexed-rasterizerDiscardEnable-09417
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetColorBlendEnableEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the firstAttachment and attachmentCount parameters of vkCmdSetColorBlendEnableEXT calls **must** specify an enable for all active color attachments in the current subpass

- VUID-vkCmdDrawIndexed-firstAttachment-07477
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_EQUATION_EXT dynamic state enabled then vkCmdSetColorBlendEquationEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the firstAttachment and attachmentCount parameters of vkCmdSetColorBlendEquationEXT calls **must** specify the blend

equations for all active color attachments in the current subpass where blending is enabled

- VUID-vkCmdDrawIndexed-rasterizerDiscardEnable-09418
  If a shader object is bound to the `VK_SHADER_STAGE_FRAGMENT_BIT` stage, and both the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE` and there are color attachments bound, then vkCmdSetColorBlendEquationEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the `firstAttachment` and `attachmentCount` parameters of `vkCmdSetColorBlendEquationEXT` calls **must** specify the blend equations for all active color attachments in the current subpass where blending is enabled
- VUID-vkCmdDrawIndexed-firstAttachment-07478
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_COLOR_WRITE_MASK_EXT` dynamic state enabled then vkCmdSetColorWriteMaskEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the `firstAttachment` and `attachmentCount` parameters of `vkCmdSetColorWriteMaskEXT` calls **must** specify the color write mask for all active color attachments in the current subpass
- VUID-vkCmdDrawIndexed-rasterizerDiscardEnable-09419
  If a shader object is bound to the `VK_SHADER_STAGE_FRAGMENT_BIT` stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE`, then vkCmdSetColorWriteMaskEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the `firstAttachment` and `attachmentCount` parameters of `vkCmdSetColorWriteMaskEXT` calls **must** specify the color write mask for all active color attachments in the current subpass
- VUID-vkCmdDrawIndexed-firstAttachment-07479
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_COLOR_BLEND_ADVANCED_EXT` dynamic state enabled then vkCmdSetColorBlendAdvancedEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the `firstAttachment` and `attachmentCount` parameters of `vkCmdSetColorBlendAdvancedEXT` calls **must** specify the advanced blend equations for all active color attachments in the current subpass where blending is enabled
- VUID-vkCmdDrawIndexed-advancedBlendMaxColorAttachments-07480
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_COLOR_BLEND_ADVANCED_EXT` and `VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT` dynamic states enabled and the last calls to vkCmdSetColorBlendEnableEXT and vkCmdSetColorBlendAdvancedEXT have enabled advanced blending, then the number of active color attachments in the current subpass **must** not exceed `advancedBlendMaxColorAttachments`
- VUID-vkCmdDrawIndexed-primitivesGeneratedQueryWithNonZeroStreams-07481
  If the `primitivesGeneratedQueryWithNonZeroStreams` feature is not enabled and the `VK_QUERY_TYPE_PRIMITIVES_GENERATED_EXT` query is active, and the bound graphics pipeline was created with `VK_DYNAMIC_STATE_RASTERIZATION_STREAM_EXT` state enabled, the last call to

vkCmdSetRasterizationStreamEXT **must** have set the `rasterizationStream` to zero

- VUID-vkCmdDrawIndexed-sampleLocationsPerPixel-07482
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT` state enabled and the `VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT` state disabled, then the `sampleLocationsPerPixel` member of `pSampleLocationsInfo` in the last call to vkCmdSetSampleLocationsEXT **must** equal the `rasterizationSamples` member of the VkPipelineMultisampleStateCreateInfo structure the bound graphics pipeline has been created with

- VUID-vkCmdDrawIndexed-sampleLocationsPerPixel-07483
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT` state enabled and the `VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT` state enabled, then the `sampleLocationsPerPixel` member of `pSampleLocationsInfo` in the last call to vkCmdSetSampleLocationsEXT **must** equal the `rasterizationSamples` parameter of the last call to vkCmdSetRasterizationSamplesEXT

- VUID-vkCmdDrawIndexed-sampleLocationsEnable-07484
  If a shader object is bound to the `VK_SHADER_STAGE_FRAGMENT_BIT` stage, or the bound graphics pipeline was created with the `VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT` state enabled, and `sampleLocationsEnable` was `VK_TRUE` in the last call to vkCmdSetSampleLocationsEnableEXT, and the current subpass has a depth/stencil attachment, then that attachment **must** have been created with the `VK_IMAGE_CREATE_SAMPLE_LOCATIONS_COMPATIBLE_DEPTH_BIT_EXT` bit set

- VUID-vkCmdDrawIndexed-sampleLocationsEnable-07485
  If a shader object is bound to the `VK_SHADER_STAGE_FRAGMENT_BIT` stage, or the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT` state enabled and the `VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT` state enabled, and if `sampleLocationsEnable` was `VK_TRUE` in the last call to vkCmdSetSampleLocationsEnableEXT, then the `sampleLocationsInfo.sampleLocationGridSize.width` in the last call to vkCmdSetSampleLocationsEXT **must** evenly divide VkMultisamplePropertiesEXT `::sampleLocationGridSize.width` as returned by vkGetPhysicalDeviceMultisamplePropertiesEXT with a `samples` parameter equaling `rasterizationSamples`

- VUID-vkCmdDrawIndexed-sampleLocationsEnable-07486
  If a shader object is bound to the `VK_SHADER_STAGE_FRAGMENT_BIT` stage, or the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT` state enabled and the `VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT` state enabled, and if `sampleLocationsEnable` was `VK_TRUE` in the last call to vkCmdSetSampleLocationsEnableEXT, then the `sampleLocationsInfo.sampleLocationGridSize.height` in the last call to vkCmdSetSampleLocationsEXT **must** evenly divide VkMultisamplePropertiesEXT `::sampleLocationGridSize.height` as returned by vkGetPhysicalDeviceMultisamplePropertiesEXT with a `samples` parameter equaling `rasterizationSamples`

- VUID-vkCmdDrawIndexed-sampleLocationsEnable-07487
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, and if sampleLocationsEnable was VK_TRUE in the last call to vkCmdSetSampleLocationsEnableEXT, the fragment shader code **must** not statically use the extended instruction InterpolateAtSample

- VUID-vkCmdDrawIndexed-sampleLocationsEnable-07936
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state disabled and the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, the sampleLocationsEnable member of a VkPipelineSampleLocationsStateCreateInfoEXT::sampleLocationsEnable in the bound graphics pipeline is VK_TRUE or VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, then, sampleLocationsInfo.sampleLocationGridSize.width **must** evenly divide VkMultisamplePropertiesEXT::sampleLocationGridSize.width as returned by vkGetPhysicalDeviceMultisamplePropertiesEXT with a samples parameter equaling the value of rasterizationSamples in the last call to vkCmdSetRasterizationSamplesEXT

- VUID-vkCmdDrawIndexed-sampleLocationsEnable-07937
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state disabled and the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, the sampleLocationsEnable member of a VkPipelineSampleLocationsStateCreateInfoEXT::sampleLocationsEnable in the bound graphics pipeline is VK_TRUE or VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, then, sampleLocationsInfo.sampleLocationGridSize.height **must** evenly divide VkMultisamplePropertiesEXT::sampleLocationGridSize.height as returned by vkGetPhysicalDeviceMultisamplePropertiesEXT with a samples parameter equaling the value of rasterizationSamples in the last call to vkCmdSetRasterizationSamplesEXT

- VUID-vkCmdDrawIndexed-sampleLocationsEnable-07938
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state disabled and the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, the sampleLocationsEnable member of a VkPipelineSampleLocationsStateCreateInfoEXT::sampleLocationsEnable in the bound graphics pipeline is VK_TRUE or VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, then, sampleLocationsInfo.sampleLocationsPerPixel **must** equal rasterizationSamples in the last call to vkCmdSetRasterizationSamplesEXT

- VUID-vkCmdDrawIndexed-coverageModulationTableEnable-07488
  If a shader object is bound to any graphics stage or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COVERAGE_MODULATION_TABLE_ENABLE_NV state enabled, and the last call to vkCmdSetCoverageModulationTableEnableNV set coverageModulationTableEnable to VK_TRUE, then the coverageModulationTableCount parameter in the last call to vkCmdSetCoverageModulationTableNV **must** equal the current rasterizationSamples divided by the number of color samples in the current subpass

- VUID-vkCmdDrawIndexed-rasterizationSamples-07489
  If the VK_NV_framebuffer_mixed_samples extension is enabled, and if current subpass has a depth/stencil attachment and depth test, stencil test, or depth bounds test are enabled in the currently bound pipeline state, then the current rasterizationSamples **must** be the

same as the sample count of the depth/stencil attachment

- VUID-vkCmdDrawIndexed-coverageToColorEnable-07490
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COVERAGE_TO_COLOR_ENABLE_NV state enabled and the last call to vkCmdSetCoverageToColorEnableNV set the coverageToColorEnable to VK_TRUE, then the current subpass **must** have a color attachment at the location selected by the last call to vkCmdSetCoverageToColorLocationNV coverageToColorLocation, with a VkFormat of VK_FORMAT_R8_UINT, VK_FORMAT_R8_SINT, VK_FORMAT_R16_UINT, VK_FORMAT_R16_SINT, VK_FORMAT_R32_UINT, or VK_FORMAT_R32_SINT

- VUID-vkCmdDrawIndexed-rasterizerDiscardEnable-09420
  If the VK_NV_fragment_coverage_to_color extension is enabled, and a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the last call to vkCmdSetCoverageToColorEnableNV set the coverageToColorEnable to VK_TRUE, then the current subpass **must** have a color attachment at the location selected by the last call to vkCmdSetCoverageToColorLocationNV coverageToColorLocation, with a VkFormat of VK_FORMAT_R8_UINT, VK_FORMAT_R8_SINT, VK_FORMAT_R16_UINT, VK_FORMAT_R16_SINT, VK_FORMAT_R32_UINT, or VK_FORMAT_R32_SINT

- VUID-vkCmdDrawIndexed-coverageReductionMode-07491
  If this VK_NV_coverage_reduction_mode extension is enabled, the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COVERAGE_TO_COLOR_ENABLE_NV and VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT states enabled, the current coverage reduction mode coverageReductionMode, then the current rasterizationSamples, and the sample counts for the color and depth/stencil attachments (if the subpass has them) **must** be a valid combination returned by vkGetPhysicalDeviceSupportedFramebufferMixedSamplesCombinationsNV

- VUID-vkCmdDrawIndexed-viewportCount-07492
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, but not the VK_DYNAMIC_STATE_VIEWPORT_SWIZZLE_NV dynamic state enabled, then the bound graphics pipeline **must** have been created with VkPipelineViewportSwizzleStateCreateInfoNV ::viewportCount greater or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT

- VUID-vkCmdDrawIndexed-viewportCount-07493
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT and VK_DYNAMIC_STATE_VIEWPORT_SWIZZLE_NV dynamic states enabled then the viewportCount parameter in the last call to vkCmdSetViewportSwizzleNV **must** be greater than or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT

- VUID-vkCmdDrawIndexed-viewportCount-09421
  If the VK_NV_viewport_swizzle extension is enabled, and a shader object is bound to any graphics stage, then the viewportCount parameter in the last call to vkCmdSetViewportSwizzleNV **must** be greater than or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT

- VUID-vkCmdDrawIndexed-rasterizationSamples-07494
  If the VK_NV_framebuffer_mixed_samples extension is enabled, and if the current subpass has any color attachments and rasterizationSamples of the last call to vkCmdSetRasterizationSamplesEXT is greater than the number of color samples, then the pipeline sampleShadingEnable **must** be VK_FALSE
- VUID-vkCmdDrawIndexed-stippledLineEnable-07495
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT or VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT dynamic states enabled, and if the current stippledLineEnable state is VK_TRUE and the current lineRasterizationMode state is VK_LINE_RASTERIZATION_MODE_RECTANGULAR_KHR, then the stippledRectangularLines feature **must** be enabled
- VUID-vkCmdDrawIndexed-stippledLineEnable-07496
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT or VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT dynamic states enabled, and if the current stippledLineEnable state is VK_TRUE and the current lineRasterizationMode state is VK_LINE_RASTERIZATION_MODE_BRESENHAM_KHR, then the stippledBresenhamLines feature **must** be enabled
- VUID-vkCmdDrawIndexed-stippledLineEnable-07497
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT or VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT dynamic states enabled, and if the current stippledLineEnable state is VK_TRUE and the current lineRasterizationMode state is VK_LINE_RASTERIZATION_MODE_RECTANGULAR_SMOOTH_KHR, then the stippledSmoothLines feature **must** be enabled
- VUID-vkCmdDrawIndexed-stippledLineEnable-07498
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT or VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT dynamic states enabled, and if the current stippledLineEnable state is VK_TRUE and the current lineRasterizationMode state is VK_LINE_RASTERIZATION_MODE_DEFAULT_KHR, then the stippledRectangularLines feature **must** be enabled and VkPhysicalDeviceLimits::strictLines **must** be VK_TRUE
- VUID-vkCmdDrawIndexed-conservativePointAndLineRasterization-07499
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_CONSERVATIVE_RASTERIZATION_MODE_EXT dynamic state enabled, conservativePointAndLineRasterization is not supported, and the effective primitive topology output by the last pre-rasterization shader stage is a line or point, then the conservativeRasterizationMode set by the last call to vkCmdSetConservativeRasterizationModeEXT **must** be VK_CONSERVATIVE_RASTERIZATION_MODE_DISABLED_EXT
- VUID-vkCmdDrawIndexed-stage-07073
  If the currently bound pipeline was created with the VkPipelineShaderStageCreateInfo::stage member of an element of VkGraphicsPipelineCreateInfo::pStages set to VK_SHADER_STAGE_VERTEX_BIT, VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT,

VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT or VK_SHADER_STAGE_GEOMETRY_BIT, then Mesh Shader Queries **must** not be active

- VUID-vkCmdDrawIndexed-None-08877
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_ATTACHMENT_FEEDBACK_LOOP_ENABLE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetAttachmentFeedbackLoopEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexed-None-07850
  If dynamic state was inherited from VkCommandBufferInheritanceViewportScissorInfoNV, it **must** be set in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexed-None-08684
  If there is no bound graphics pipeline, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_VERTEX_BIT
- VUID-vkCmdDrawIndexed-None-08685
  If there is no bound graphics pipeline, and the tessellationShader feature is enabled, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT
- VUID-vkCmdDrawIndexed-None-08686
  If there is no bound graphics pipeline, and the tessellationShader feature is enabled, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT
- VUID-vkCmdDrawIndexed-None-08687
  If there is no bound graphics pipeline, and the geometryShader feature is enabled, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_GEOMETRY_BIT
- VUID-vkCmdDrawIndexed-None-08688
  If there is no bound graphics pipeline, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_FRAGMENT_BIT
- VUID-vkCmdDrawIndexed-None-08689
  If there is no bound graphics pipeline, and the taskShader feature is enabled, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_TASK_BIT_EXT
- VUID-vkCmdDrawIndexed-None-08690
  If there is no bound graphics pipeline, and the meshShader feature is enabled, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_MESH_BIT_EXT
- VUID-vkCmdDrawIndexed-None-08693
  If there is no bound graphics pipeline, and at least one of the taskShader and meshShader features is enabled, one of the VK_SHADER_STAGE_VERTEX_BIT or VK_SHADER_STAGE_MESH_BIT_EXT stages **must** have a valid VkShaderEXT bound, and the other **must** have no VkShaderEXT bound

- VUID-vkCmdDrawIndexed-None-08694
  If there is no bound graphics pipeline, and both the `taskShader` and `meshShader` features are enabled, and a valid `VkShaderEXT` is bound the to the `VK_SHADER_STAGE_MESH_BIT_EXT` stage, and that `VkShaderEXT` was created without the `VK_SHADER_CREATE_NO_TASK_SHADER_BIT_EXT` flag, a valid `VkShaderEXT` **must** be bound to the `VK_SHADER_STAGE_TASK_BIT_EXT` stage
- VUID-vkCmdDrawIndexed-None-08695
  If there is no bound graphics pipeline, and both the `taskShader` and `meshShader` features are enabled, and a valid `VkShaderEXT` is bound the to the `VK_SHADER_STAGE_MESH_BIT_EXT` stage, and that `VkShaderEXT` was created with the `VK_SHADER_CREATE_NO_TASK_SHADER_BIT_EXT` flag, there **must** be no `VkShaderEXT` bound to the `VK_SHADER_STAGE_TASK_BIT_EXT` stage
- VUID-vkCmdDrawIndexed-None-08696
  If there is no bound graphics pipeline, and a valid `VkShaderEXT` is bound to the `VK_SHADER_STAGE_VERTEX_BIT` stage, there **must** be no `VkShaderEXT` bound to either the `VK_SHADER_STAGE_TASK_BIT_EXT` stage or the `VK_SHADER_STAGE_MESH_BIT_EXT` stage
- VUID-vkCmdDrawIndexed-None-08698
  If any graphics shader is bound which was created with the `VK_SHADER_CREATE_LINK_STAGE_BIT_EXT` flag, then all shaders created with the `VK_SHADER_CREATE_LINK_STAGE_BIT_EXT` flag in the same vkCreateShadersEXT call **must** also be bound
- VUID-vkCmdDrawIndexed-None-08699
  If any graphics shader is bound which was created with the `VK_SHADER_CREATE_LINK_STAGE_BIT_EXT` flag, any stages in between stages whose shaders which did not create a shader with the `VK_SHADER_CREATE_LINK_STAGE_BIT_EXT` flag as part of the same vkCreateShadersEXT call **must** not have any `VkShaderEXT` bound
- VUID-vkCmdDrawIndexed-None-08878
  All bound graphics shader objects **must** have been created with identical or identically defined push constant ranges
- VUID-vkCmdDrawIndexed-None-08879
  All bound graphics shader objects **must** have been created with identical or identically defined arrays of descriptor set layouts
- VUID-vkCmdDrawIndexed-colorAttachmentCount-09372
  If the current render pass instance was begun with vkCmdBeginRenderingKHR and a VkRenderingInfo::`colorAttachmentCount` equal to `1`, a color attachment with a resolve mode of `VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID`, and a fragment shader is bound, it **must** not declare the `DepthReplacing` or `StencilRefReplacingEXT` execution modes
- VUID-vkCmdDrawIndexed-pDynamicStates-08715
  If the bound graphics pipeline state includes a fragment shader stage, was created with `VK_DYNAMIC_STATE_DEPTH_WRITE_ENABLE` set in VkPipelineDynamicStateCreateInfo::`pDynamicStates`, and the fragment shader declares the `EarlyFragmentTests` execution mode and uses `OpDepthAttachmentReadEXT`, the `depthWriteEnable` parameter in the last call to vkCmdSetDepthWriteEnableEXT **must** be `VK_FALSE`
- VUID-vkCmdDrawIndexed-pDynamicStates-08716
  If the bound graphics pipeline state includes a fragment shader stage, was created with

VK_DYNAMIC_STATE_STENCIL_WRITE_MASK set in VkPipelineDynamicStateCreateInfo::pDynamicStates, and the fragment shader declares the EarlyFragmentTests execution mode and uses OpStencilAttachmentReadEXT, the writeMask parameter in the last call to vkCmdSetStencilWriteMask **must** be 0

- VUID-vkCmdDrawIndexed-None-09116
  If a shader object is bound to any graphics stage or the currently bound graphics pipeline was created with VK_DYNAMIC_STATE_COLOR_WRITE_MASK_EXT, and the format of any color attachment is VK_FORMAT_E5B9G9R9_UFLOAT_PACK32, the corresponding element of the pColorWriteMasks parameter of vkCmdSetColorWriteMaskEXT **must** either include all of VK_COLOR_COMPONENT_R_BIT, VK_COLOR_COMPONENT_G_BIT, and VK_COLOR_COMPONENT_B_BIT, or none of them

- VUID-vkCmdDrawIndexed-maxFragmentDualSrcAttachments-09239
  If blending is enabled for any attachment where either the source or destination blend factors for that attachment use the secondary color input, the maximum value of Location for any output attachment statically used in the Fragment Execution Model executed by this command **must** be less than maxFragmentDualSrcAttachments

- VUID-vkCmdDrawIndexed-None-09548
  If the current render pass was begun with vkCmdBeginRenderingKHR, and there is no shader object bound to any graphics stage, the value of each element of VkRenderingAttachmentLocationInfoKHR::pColorAttachmentLocations set by vkCmdSetRenderingAttachmentLocationsKHR **must** match the value set for the corresponding element in the currently bound pipeline

- VUID-vkCmdDrawIndexed-None-09549
  If the current render pass was begun with vkCmdBeginRenderingKHR, and there is no shader object bound to any graphics stage, input attachment index mappings in the currently bound pipeline **must** match those set for the current render pass instance via VkRenderingInputAttachmentIndexInfoKHR

- VUID-vkCmdDrawIndexed-None-09642
  If the current render pass was begun with vkCmdBeginRenderingKHR with the VK_RENDERING_ENABLE_LEGACY_DITHERING_BIT_EXT flag, the bound graphics pipeline **must** have been created with VK_PIPELINE_CREATE_2_ENABLE_LEGACY_DITHERING_BIT_EXT

- VUID-vkCmdDrawIndexed-None-09643
  If the bound graphics pipeline was created with VK_PIPELINE_CREATE_2_ENABLE_LEGACY_DITHERING_BIT_EXT, the current render pass **must** have begun with vkCmdBeginRenderingKHR with the VK_RENDERING_ENABLE_LEGACY_DITHERING_BIT_EXT flag

- VUID-vkCmdDrawIndexed-commandBuffer-02712
  If commandBuffer is a protected command buffer and protectedNoFault is not supported, any resource written to by the VkPipeline object bound to the pipeline bind point used by this command **must** not be an unprotected resource

- VUID-vkCmdDrawIndexed-commandBuffer-02713
  If commandBuffer is a protected command buffer and protectedNoFault is not supported, pipeline stages other than the framebuffer-space and compute stages in the VkPipeline object bound to the pipeline bind point used by this command **must** not write to any

resource

- VUID-vkCmdDrawIndexed-commandBuffer-04617
  If any of the shader stages of the `VkPipeline` bound to the pipeline bind point used by this command uses the `RayQueryKHR` capability, then `commandBuffer` **must** not be a protected command buffer

- VUID-vkCmdDrawIndexed-None-04007
  All vertex input bindings accessed via vertex input variables declared in the vertex shader entry point's interface **must** have either valid or VK_NULL_HANDLE buffers bound

- VUID-vkCmdDrawIndexed-None-04008
  If the `nullDescriptor` feature is not enabled, all vertex input bindings accessed via vertex input variables declared in the vertex shader entry point's interface **must** not be VK_NULL_HANDLE

- VUID-vkCmdDrawIndexed-None-02721
  If `robustBufferAccess` is not enabled, and that pipeline was created without enabling `VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_EXT` for `vertexInputs`, then for a given vertex buffer binding, any attribute data fetched **must** be entirely contained within the corresponding vertex buffer binding, as described in Vertex Input Description

- VUID-vkCmdDrawIndexed-None-07842
  If there is a shader object bound to the `VK_SHADER_STAGE_VERTEX_BIT` stage or the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_PRIMITIVE_TOPOLOGY` dynamic state enabled then vkCmdSetPrimitiveTopologyEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexed-dynamicPrimitiveTopologyUnrestricted-07500
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_PRIMITIVE_TOPOLOGY` dynamic state enabled and the `dynamicPrimitiveTopologyUnrestricted` is `VK_FALSE`, then the `primitiveTopology` parameter of `vkCmdSetPrimitiveTopologyEXT` **must** be of the same topology class as the pipeline VkPipelineInputAssemblyStateCreateInfo::`topology` state

- VUID-vkCmdDrawIndexed-pStrides-04913
  If the bound graphics pipeline was created with the `VK_DYNAMIC_STATE_VERTEX_INPUT_BINDING_STRIDE_EXT` dynamic state enabled, but without the `VK_DYNAMIC_STATE_VERTEX_INPUT_EXT` dynamic state enabled, then vkCmdBindVertexBuffers2EXT **must** have been called and not subsequently invalidated in the current command buffer prior to this draw command, and the `pStrides` parameter of vkCmdBindVertexBuffers2EXT **must** not be `NULL`

- VUID-vkCmdDrawIndexed-None-04914
  If there is a shader object bound to the `VK_SHADER_STAGE_VERTEX_BIT` stage or the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VERTEX_INPUT_EXT` dynamic state enabled then vkCmdSetVertexInputEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this draw command

- VUID-vkCmdDrawIndexed-Input-07939
  If there is a shader object bound to the `VK_SHADER_STAGE_VERTEX_BIT` stage or the bound

graphics pipeline state was created with the VK_DYNAMIC_STATE_VERTEX_INPUT_EXT dynamic state enabled then all variables with the Input storage class decorated with Location in the Vertex Execution Model OpEntryPoint **must** contain a location in VkVertexInputAttributeDescription2EXT::location

- VUID-vkCmdDrawIndexed-Input-08734
  If there is a shader object bound to the VK_SHADER_STAGE_VERTEX_BIT stage or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VERTEX_INPUT_EXT dynamic state enabled and either legacyVertexAttributes is not enabled or the SPIR-V Type associated with a given Input variable of the corresponding Location in the Vertex Execution Model OpEntryPoint is 64-bit, then the numeric type associated with all Input variables of the corresponding Location in the Vertex Execution Model OpEntryPoint **must** be the same as VkVertexInputAttributeDescription2EXT::format
- VUID-vkCmdDrawIndexed-format-08936
  If there is a shader object bound to the VK_SHADER_STAGE_VERTEX_BIT stage or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VERTEX_INPUT_EXT dynamic state enabled and VkVertexInputAttributeDescription2EXT::format has a 64-bit component, then the scalar width associated with all Input variables of the corresponding Location in the Vertex Execution Model OpEntryPoint **must** be 64-bit
- VUID-vkCmdDrawIndexed-format-08937
  If there is a shader object bound to the VK_SHADER_STAGE_VERTEX_BIT stage or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VERTEX_INPUT_EXT dynamic state enabled and the scalar width associated with a Location decorated Input variable in the Vertex Execution Model OpEntryPoint is 64-bit, then the corresponding VkVertexInputAttributeDescription2EXT::format **must** have a 64-bit component
- VUID-vkCmdDrawIndexed-None-09203
  If there is a shader object bound to the VK_SHADER_STAGE_VERTEX_BIT stage or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VERTEX_INPUT_EXT dynamic state enabled and VkVertexInputAttributeDescription2EXT::format has a 64-bit component, then all Input variables at the corresponding Location in the Vertex Execution Model OpEntryPoint **must** not use components that are not present in the format
- VUID-vkCmdDrawIndexed-None-04875
  If there is a shader object bound to the VK_SHADER_STAGE_VERTEX_BIT stage and the most recent call to vkCmdSetPrimitiveTopologyEXT in the current command buffer set primitiveTopology to VK_PRIMITIVE_TOPOLOGY_PATCH_LIST, or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_PATCH_CONTROL_POINTS_EXT dynamic state enabled then vkCmdSetPatchControlPointsEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexed-None-04879
  If there is a shader object bound to the VK_SHADER_STAGE_VERTEX_BIT stage or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_PRIMITIVE_RESTART_ENABLE dynamic state enabled then vkCmdSetPrimitiveRestartEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexed-None-09637
  If the primitiveTopologyListRestart feature is not enabled, the topology is

VK_PRIMITIVE_TOPOLOGY_POINT_LIST, VK_PRIMITIVE_TOPOLOGY_LINE_LIST, VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST, VK_PRIMITIVE_TOPOLOGY_LINE_LIST_WITH_ADJACENCY, or VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST_WITH_ADJACENCY, there is a shader object bound to the VK_SHADER_STAGE_VERTEX_BIT stage or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_PRIMITIVE_RESTART_ENABLE dynamic state enabled then vkCmdSetPrimitiveRestartEnableEXT **must** be VK_FALSE

- VUID-vkCmdDrawIndexed-stage-06481
  The bound graphics pipeline **must** not have been created with the VkPipelineShaderStageCreateInfo::stage member of an element of VkGraphicsPipelineCreateInfo::pStages set to VK_SHADER_STAGE_TASK_BIT_EXT or VK_SHADER_STAGE_MESH_BIT_EXT

- VUID-vkCmdDrawIndexed-None-08885
  There **must** be no shader object bound to either of the VK_SHADER_STAGE_TASK_BIT_EXT or VK_SHADER_STAGE_MESH_BIT_EXT stages

- VUID-vkCmdDrawIndexed-None-07312
  If maintenance6 is not enabled, a valid index buffer **must** be bound

- VUID-vkCmdDrawIndexed-pNext-09461
  If the bound graphics pipeline state was created with VkPipelineVertexInputDivisorStateCreateInfoKHR in the pNext chain of VkGraphicsPipelineCreateInfo::pVertexInputState, any member of VkPipelineVertexInputDivisorStateCreateInfoKHR::pVertexBindingDivisors has a value other than 1 in divisor, and VkPhysicalDeviceVertexAttributeDivisorPropertiesKHR::supportsNonZeroFirstInstance is VK_FALSE, then firstInstance **must** be 0

- VUID-vkCmdDrawIndexed-None-09462
  If shader objects are used for drawing or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VERTEX_INPUT_EXT dynamic state enabled, any member of the pVertexBindingDescriptions parameter to the vkCmdSetVertexInputEXT call that sets this dynamic state has a value other than 1 in divisor, and VkPhysicalDeviceVertexAttributeDivisorPropertiesKHR::supportsNonZeroFirstInstance is VK_FALSE, then firstInstance **must** be 0

- VUID-vkCmdDrawIndexed-robustBufferAccess2-08798
  If robustBufferAccess2 is not enabled, (indexSize × (firstIndex + indexCount) + offset) **must** be less than or equal to the size of the bound index buffer, with indexSize being based on the type specified by indexType, where the index buffer, indexType, and offset are specified via vkCmdBindIndexBuffer or vkCmdBindIndexBuffer2KHR. If vkCmdBindIndexBuffer2KHR is used to bind the index buffer, the size of the bound index buffer is vkCmdBindIndexBuffer2KHR::size

## Valid Usage (Implicit)

- VUID-vkCmdDrawIndexed-commandBuffer-parameter
  commandBuffer **must** be a valid VkCommandBuffer handle

- VUID-vkCmdDrawIndexed-commandBuffer-recording

commandBuffer **must** be in the recording state

- VUID-vkCmdDrawIndexed-commandBuffer-cmdpool
  The VkCommandPool that commandBuffer was allocated from **must** support graphics operations
- VUID-vkCmdDrawIndexed-renderpass
  This command **must** only be called inside of a render pass instance
- VUID-vkCmdDrawIndexed-videocoding
  This command **must** only be called outside of a video coding scope

## Host Synchronization

- Host access to commandBuffer **must** be externally synchronized
- Host access to the VkCommandPool that commandBuffer was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Inside | Outside | Graphics | Action |

To record an ordered sequence of draws which have no state changes between them, call:

```
// Provided by VK_EXT_multi_draw
void vkCmdDrawMultiEXT(
    VkCommandBuffer                             commandBuffer,
    uint32_t                                    drawCount,
    const VkMultiDrawInfoEXT*                   pVertexInfo,
    uint32_t                                    instanceCount,
    uint32_t                                    firstInstance,
    uint32_t                                    stride);
```

- commandBuffer is the command buffer into which the command is recorded.
- drawCount is the number of draws to execute, and **can** be zero.
- pVertexInfo is a pointer to an array of VkMultiDrawInfoEXT with vertex information to be drawn.
- instanceCount is the number of instances per draw.
- firstInstance is the instance ID of the first instance in each draw.
- stride is the byte stride between consecutive elements of pVertexInfo.

The number of draws recorded is drawCount, with each draw reading, sequentially, a firstVertex and a vertexCount from pVertexInfo. For each recorded draw, primitives are assembled as for vkCmdDraw, and drawn instanceCount times with instanceIndex starting with firstInstance and sequentially for each instance.

## Valid Usage

- VUID-vkCmdDrawMultiEXT-magFilter-04553
  If a VkSampler created with magFilter or minFilter equal to VK_FILTER_LINEAR, reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE, and compareEnable equal to VK_FALSE is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT
- VUID-vkCmdDrawMultiEXT-magFilter-09598
  If a VkSampler created with magFilter or minFilter equal to VK_FILTER_LINEAR and reductionMode equal to either VK_SAMPLER_REDUCTION_MODE_MIN or VK_SAMPLER_REDUCTION_MODE_MAX is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT
- VUID-vkCmdDrawMultiEXT-mipmapMode-04770
  If a VkSampler created with mipmapMode equal to VK_SAMPLER_MIPMAP_MODE_LINEAR, reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE, and compareEnable equal to VK_FALSE is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT
- VUID-vkCmdDrawMultiEXT-mipmapMode-09599
  If a VkSampler created with mipmapMode equal to VK_SAMPLER_MIPMAP_MODE_LINEAR and reductionMode equal to either VK_SAMPLER_REDUCTION_MODE_MIN or VK_SAMPLER_REDUCTION_MODE_MAX is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT
- VUID-vkCmdDrawMultiEXT-unnormalizedCoordinates-09635
  If a VkSampler created with unnormalizedCoordinates equal to VK_TRUE is used to sample a VkImageView as a result of this command, then the image view's levelCount and layerCount **must** be 1
- VUID-vkCmdDrawMultiEXT-unnormalizedCoordinates-09636
  If a VkSampler created with unnormalizedCoordinates equal to VK_TRUE is used to sample a VkImageView as a result of this command, then the image view's viewType **must** be VK_IMAGE_VIEW_TYPE_1D or VK_IMAGE_VIEW_TYPE_2D
- VUID-vkCmdDrawMultiEXT-None-06479
  If a VkImageView is sampled with depth comparison, the image view's format features **must** contain VK_FORMAT_FEATURE_2_SAMPLED_IMAGE_DEPTH_COMPARISON_BIT
- VUID-vkCmdDrawMultiEXT-None-02691
  If a VkImageView is accessed using atomic operations as a result of this command, then the image view's format features **must** contain

VK_FORMAT_FEATURE_STORAGE_IMAGE_ATOMIC_BIT

- VUID-vkCmdDrawMultiEXT-None-07888
  If a VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER descriptor is accessed using atomic operations as a result of this command, then the storage texel buffer's format features **must** contain VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_ATOMIC_BIT
- VUID-vkCmdDrawMultiEXT-None-02692
  If a VkImageView is sampled with VK_FILTER_CUBIC_EXT as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_CUBIC_BIT_EXT
- VUID-vkCmdDrawMultiEXT-None-02693
  If the VK_EXT_filter_cubic extension is not enabled and any VkImageView is sampled with VK_FILTER_CUBIC_EXT as a result of this command, it **must** not have a VkImageViewType of VK_IMAGE_VIEW_TYPE_3D, VK_IMAGE_VIEW_TYPE_CUBE, or VK_IMAGE_VIEW_TYPE_CUBE_ARRAY
- VUID-vkCmdDrawMultiEXT-filterCubic-02694
  Any VkImageView being sampled with VK_FILTER_CUBIC_EXT as a result of this command **must** have a VkImageViewType and format that supports cubic filtering, as specified by VkFilterCubicImageViewImageFormatPropertiesEXT::filterCubic returned by vkGetPhysicalDeviceImageFormatProperties2
- VUID-vkCmdDrawMultiEXT-filterCubicMinmax-02695
  Any VkImageView being sampled with VK_FILTER_CUBIC_EXT with a reduction mode of either VK_SAMPLER_REDUCTION_MODE_MIN or VK_SAMPLER_REDUCTION_MODE_MAX as a result of this command **must** have a VkImageViewType and format that supports cubic filtering together with minmax filtering, as specified by VkFilterCubicImageViewImageFormatPropertiesEXT::filterCubicMinmax returned by vkGetPhysicalDeviceImageFormatProperties2
- VUID-vkCmdDrawMultiEXT-cubicRangeClamp-09212
  If the cubicRangeClamp feature is not enabled, then any VkImageView being sampled with VK_FILTER_CUBIC_EXT as a result of this command **must** not have a VkSamplerReductionModeCreateInfo::reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE_RANGECLAMP_QCOM
- VUID-vkCmdDrawMultiEXT-reductionMode-09213
  Any VkImageView being sampled with a VkSamplerReductionModeCreateInfo::reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE_RANGECLAMP_QCOM as a result of this command **must** sample with VK_FILTER_CUBIC_EXT
- VUID-vkCmdDrawMultiEXT-selectableCubicWeights-09214
  If the selectableCubicWeights feature is not enabled, then any VkImageView being sampled with VK_FILTER_CUBIC_EXT as a result of this command **must** have VkSamplerCubicWeightsCreateInfoQCOM::cubicWeights equal to VK_CUBIC_FILTER_WEIGHTS_CATMULL_ROM_QCOM
- VUID-vkCmdDrawMultiEXT-flags-02696
  Any VkImage created with a VkImageCreateInfo::flags containing VK_IMAGE_CREATE_CORNER_SAMPLED_BIT_NV sampled as a result of this command **must** only be sampled using a VkSamplerAddressMode of VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_EDGE
- VUID-vkCmdDrawMultiEXT-OpTypeImage-07027

For any VkImageView being written as a storage image where the image format field of the OpTypeImage is Unknown, the view's format features **must** contain VK_FORMAT_FEATURE_2_STORAGE_WRITE_WITHOUT_FORMAT_BIT

- VUID-vkCmdDrawMultiEXT-OpTypeImage-07028
  For any VkImageView being read as a storage image where the image format field of the OpTypeImage is Unknown, the view's format features **must** contain VK_FORMAT_FEATURE_2_STORAGE_READ_WITHOUT_FORMAT_BIT
- VUID-vkCmdDrawMultiEXT-OpTypeImage-07029
  For any VkBufferView being written as a storage texel buffer where the image format field of the OpTypeImage is Unknown, the view's buffer features **must** contain VK_FORMAT_FEATURE_2_STORAGE_WRITE_WITHOUT_FORMAT_BIT
- VUID-vkCmdDrawMultiEXT-OpTypeImage-07030
  Any VkBufferView being read as a storage texel buffer where the image format field of the OpTypeImage is Unknown then the view's buffer features **must** contain VK_FORMAT_FEATURE_2_STORAGE_READ_WITHOUT_FORMAT_BIT
- VUID-vkCmdDrawMultiEXT-None-08600
  For each set *n* that is statically used by a bound shader, a descriptor set **must** have been bound to *n* at the same pipeline bind point, with a VkPipelineLayout that is compatible for set *n*, with the VkPipelineLayout used to create the current VkPipeline or the VkDescriptorSetLayout array used to create the current VkShaderEXT , as described in Pipeline Layout Compatibility
- VUID-vkCmdDrawMultiEXT-None-08601
  For each push constant that is statically used by a bound shader, a push constant value **must** have been set for the same pipeline bind point, with a VkPipelineLayout that is compatible for push constants, with the VkPipelineLayout used to create the current VkPipeline or the VkDescriptorSetLayout array used to create the current VkShaderEXT , as described in Pipeline Layout Compatibility
- VUID-vkCmdDrawMultiEXT-None-10068
  For each array of resources that is used by a bound shader, the indices used to access members of the array **must** be less than the descriptor count for the identified binding in the descriptor sets used by this command
- VUID-vkCmdDrawMultiEXT-maintenance4-08602
  If the maintenance4 feature is not enabled, then for each push constant that is statically used by a bound shader, a push constant value **must** have been set for the same pipeline bind point, with a VkPipelineLayout that is compatible for push constants, with the VkPipelineLayout used to create the current VkPipeline or the VkDescriptorSetLayout and VkPushConstantRange arrays used to create the current VkShaderEXT , as described in Pipeline Layout Compatibility
- VUID-vkCmdDrawMultiEXT-None-08114
  Descriptors in each bound descriptor set, specified via vkCmdBindDescriptorSets, **must** be valid as described by descriptor validity if they are statically used by the VkPipeline bound to the pipeline bind point used by this command and the bound VkPipeline was not created with VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT
- VUID-vkCmdDrawMultiEXT-None-08115

If the descriptors used by the VkPipeline bound to the pipeline bind point were specified via vkCmdBindDescriptorSets, the bound VkPipeline **must** have been created without VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT

- VUID-vkCmdDrawMultiEXT-None-08116
  Descriptors in bound descriptor buffers, specified via vkCmdSetDescriptorBufferOffsetsEXT, **must** be valid if they are dynamically used by the VkPipeline bound to the pipeline bind point used by this command and the bound VkPipeline was created with VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT
- VUID-vkCmdDrawMultiEXT-None-08604
  Descriptors in bound descriptor buffers, specified via vkCmdSetDescriptorBufferOffsetsEXT, **must** be valid if they are dynamically used by any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command
- VUID-vkCmdDrawMultiEXT-None-08117
  If the descriptors used by the VkPipeline bound to the pipeline bind point were specified via vkCmdSetDescriptorBufferOffsetsEXT, the bound VkPipeline **must** have been created with VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT
- VUID-vkCmdDrawMultiEXT-None-08119
  If a descriptor is dynamically used with a VkPipeline created with VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT, the descriptor memory **must** be resident
- VUID-vkCmdDrawMultiEXT-None-08605
  If a descriptor is dynamically used with a VkShaderEXT created with a VkDescriptorSetLayout that was created with VK_DESCRIPTOR_SET_LAYOUT_CREATE_DESCRIPTOR_BUFFER_BIT_EXT, the descriptor memory **must** be resident
- VUID-vkCmdDrawMultiEXT-None-08606
  If the shaderObject feature is not enabled, a valid pipeline **must** be bound to the pipeline bind point used by this command
- VUID-vkCmdDrawMultiEXT-None-08608
  If a pipeline is bound to the pipeline bind point used by this command, there **must** not have been any calls to dynamic state setting commands for any state specified statically in the VkPipeline object bound to the pipeline bind point used by this command, since that pipeline was bound
- VUID-vkCmdDrawMultiEXT-None-08609
  If the VkPipeline object bound to the pipeline bind point used by this command or any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a VkSampler object that uses unnormalized coordinates, that sampler **must** not be used to sample from any VkImage with a VkImageView of the type VK_IMAGE_VIEW_TYPE_3D, VK_IMAGE_VIEW_TYPE_CUBE, VK_IMAGE_VIEW_TYPE_1D_ARRAY, VK_IMAGE_VIEW_TYPE_2D_ARRAY or VK_IMAGE_VIEW_TYPE_CUBE_ARRAY, in any shader stage
- VUID-vkCmdDrawMultiEXT-None-08610
  If the VkPipeline object bound to the pipeline bind point used by this command or any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a VkSampler object that uses unnormalized coordinates, that sampler

**must** not be used with any of the SPIR-V `OpImageSample*` or `OpImageSparseSample*` instructions with `ImplicitLod`, `Dref` or `Proj` in their name, in any shader stage

- VUID-vkCmdDrawMultiEXT-None-08611
  If the VkPipeline object bound to the pipeline bind point used by this command or any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a VkSampler object that uses unnormalized coordinates, that sampler **must** not be used with any of the SPIR-V `OpImageSample*` or `OpImageSparseSample*` instructions that includes a LOD bias or any offset values, in any shader stage

- VUID-vkCmdDrawMultiEXT-None-08607
  If the `shaderObject` is enabled, either a valid pipeline **must** be bound to the pipeline bind point used by this command, or a valid combination of valid and VK_NULL_HANDLE shader objects **must** be bound to every supported shader stage corresponding to the pipeline bind point used by this command

- VUID-vkCmdDrawMultiEXT-uniformBuffers-06935
  If any stage of the VkPipeline object bound to the pipeline bind point used by this command accesses a uniform buffer, and that stage was created without enabling either `VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_EXT` or `VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_2_EXT` for `uniformBuffers`, and the `robustBufferAccess` feature is not enabled, that stage **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point

- VUID-vkCmdDrawMultiEXT-None-08612
  If the `robustBufferAccess` feature is not enabled, and any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a uniform buffer, it **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point

- VUID-vkCmdDrawMultiEXT-storageBuffers-06936
  If any stage of the VkPipeline object bound to the pipeline bind point used by this command accesses a storage buffer, and that stage was created without enabling either `VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_EXT` or `VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_2_EXT` for `storageBuffers`, and the `robustBufferAccess` feature is not enabled, that stage **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point

- VUID-vkCmdDrawMultiEXT-None-08613
  If the `robustBufferAccess` feature is not enabled, and any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a storage buffer, it **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point

- VUID-vkCmdDrawMultiEXT-commandBuffer-02707
  If `commandBuffer` is an unprotected command buffer and `protectedNoFault` is not supported, any resource accessed by bound shaders **must** not be a protected resource

- VUID-vkCmdDrawMultiEXT-None-06550
  If a bound shader accesses a VkSampler or VkImageView object that enables sampler $Y'C_BC_R$ conversion, that object **must** only be used with `OpImageSample*` or `OpImageSparseSample*`

instructions

- VUID-vkCmdDrawMultiEXT-ConstOffset-06551
  If a bound shader accesses a VkSampler or VkImageView object that enables sampler $Y'C_BC_R$ conversion, that object **must** not use the `ConstOffset` and `Offset` operands
- VUID-vkCmdDrawMultiEXT-viewType-07752
  If a VkImageView is accessed as a result of this command, then the image view's `viewType` **must** match the `Dim` operand of the `OpTypeImage` as described in Compatibility Between SPIR-V Image Dimensions and Vulkan ImageView Types
- VUID-vkCmdDrawMultiEXT-format-07753
  If a VkImageView is accessed as a result of this command, then the numeric type of the image view's `format` and the `Sampled Type` operand of the `OpTypeImage` **must** match
- VUID-vkCmdDrawMultiEXT-OpImageWrite-08795
  If a VkImageView created with a format other than `VK_FORMAT_A8_UNORM_KHR` is accessed using `OpImageWrite` as a result of this command, then the `Type` of the `Texel` operand of that instruction **must** have at least as many components as the image view's format
- VUID-vkCmdDrawMultiEXT-OpImageWrite-08796
  If a VkImageView created with the format `VK_FORMAT_A8_UNORM_KHR` is accessed using `OpImageWrite` as a result of this command, then the `Type` of the `Texel` operand of that instruction **must** have four components
- VUID-vkCmdDrawMultiEXT-OpImageWrite-04469
  If a VkBufferView is accessed using `OpImageWrite` as a result of this command, then the `Type` of the `Texel` operand of that instruction **must** have at least as many components as the buffer view's format
- VUID-vkCmdDrawMultiEXT-SampledType-04470
  If a VkImageView with a VkFormat that has a 64-bit component width is accessed as a result of this command, the `SampledType` of the `OpTypeImage` operand of that instruction **must** have a `Width` of 64
- VUID-vkCmdDrawMultiEXT-SampledType-04471
  If a VkImageView with a VkFormat that has a component width less than 64-bit is accessed as a result of this command, the `SampledType` of the `OpTypeImage` operand of that instruction **must** have a `Width` of 32
- VUID-vkCmdDrawMultiEXT-SampledType-04472
  If a VkBufferView with a VkFormat that has a 64-bit component width is accessed as a result of this command, the `SampledType` of the `OpTypeImage` operand of that instruction **must** have a `Width` of 64
- VUID-vkCmdDrawMultiEXT-SampledType-04473
  If a VkBufferView with a VkFormat that has a component width less than 64-bit is accessed as a result of this command, the `SampledType` of the `OpTypeImage` operand of that instruction **must** have a `Width` of 32
- VUID-vkCmdDrawMultiEXT-sparseImageInt64Atomics-04474
  If the `sparseImageInt64Atomics` feature is not enabled, VkImage objects created with the `VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT` flag **must** not be accessed by atomic instructions through an `OpTypeImage` with a `SampledType` with a `Width` of 64 by this command

- VUID-vkCmdDrawMultiEXT-sparseImageInt64Atomics-04475
  If the `sparseImageInt64Atomics` feature is not enabled, VkBuffer objects created with the `VK_BUFFER_CREATE_SPARSE_RESIDENCY_BIT` flag **must** not be accessed by atomic instructions through an `OpTypeImage` with a `SampledType` with a `Width` of 64 by this command
- VUID-vkCmdDrawMultiEXT-OpImageWeightedSampleQCOM-06971
  If `OpImageWeightedSampleQCOM` is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_2_WEIGHT_SAMPLED_IMAGE_BIT_QCOM`
- VUID-vkCmdDrawMultiEXT-OpImageWeightedSampleQCOM-06972
  If `OpImageWeightedSampleQCOM` uses a VkImageView as a sample weight image as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_2_WEIGHT_IMAGE_BIT_QCOM`
- VUID-vkCmdDrawMultiEXT-OpImageBoxFilterQCOM-06973
  If `OpImageBoxFilterQCOM` is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_2_BOX_FILTER_SAMPLED_BIT_QCOM`
- VUID-vkCmdDrawMultiEXT-OpImageBlockMatchSSDQCOM-06974
  If `OpImageBlockMatchSSDQCOM` is used to read from an VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM`
- VUID-vkCmdDrawMultiEXT-OpImageBlockMatchSADQCOM-06975
  If `OpImageBlockMatchSADQCOM` is used to read from an VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM`
- VUID-vkCmdDrawMultiEXT-OpImageBlockMatchSADQCOM-06976
  If `OpImageBlockMatchSADQCOM` or OpImageBlockMatchSSDQCOM is used to read from a reference image as result of this command, then the specified reference coordinates **must** not fail integer texel coordinate validation
- VUID-vkCmdDrawMultiEXT-OpImageWeightedSampleQCOM-06977
  If `OpImageWeightedSampleQCOM`, `OpImageBoxFilterQCOM`, `OpImageBlockMatchWindowSSDQCOM`, `OpImageBlockMatchWindowSADQCOM`, `OpImageBlockMatchGatherSSDQCOM`, `OpImageBlockMatchGatherSADQCOM`, `OpImageBlockMatchSSDQCOM`, or `OpImageBlockMatchSADQCOM` uses a VkSampler as a result of this command, then the sampler **must** have been created with `VK_SAMPLER_CREATE_IMAGE_PROCESSING_BIT_QCOM`
- VUID-vkCmdDrawMultiEXT-OpImageWeightedSampleQCOM-06978
  If any command other than `OpImageWeightedSampleQCOM`, `OpImageBoxFilterQCOM`, `OpImageBlockMatchWindowSSDQCOM`, `OpImageBlockMatchWindowSADQCOM`, `OpImageBlockMatchGatherSSDQCOM`, `OpImageBlockMatchGatherSADQCOM`, `OpImageBlockMatchSSDQCOM`, or `OpImageBlockMatchSADQCOM` uses a VkSampler as a result of this command, then the sampler **must** not have been created with `VK_SAMPLER_CREATE_IMAGE_PROCESSING_BIT_QCOM`
- VUID-vkCmdDrawMultiEXT-OpImageBlockMatchWindow-09215
  If a `OpImageBlockMatchWindow*QCOM` or `OpImageBlockMatchGather*QCOM` instruction is used to read from an VkImageView as a result of this command, then the image view's format

features **must** contain VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM

- VUID-vkCmdDrawMultiEXT-OpImageBlockMatchWindow-09216
  If a OpImageBlockMatchWindow*QCOM or OpImageBlockMatchGather*QCOM instruction is used to read from an VkImageView as a result of this command, then the image view's format **must** be a single-component format
- VUID-vkCmdDrawMultiEXT-OpImageBlockMatchWindow-09217
  If a OpImageBlockMatchWindow*QCOM or OpImageBlockMatchGather*QCOM read from a reference image as result of this command, then the specified reference coordinates **must** not fail integer texel coordinate validation
- VUID-vkCmdDrawMultiEXT-None-07288
  Any shader invocation executed by this command **must** terminate
- VUID-vkCmdDrawMultiEXT-None-09600
  If a descriptor with type equal to any of VK_DESCRIPTOR_TYPE_SAMPLE_WEIGHT_IMAGE_QCOM, VK_DESCRIPTOR_TYPE_BLOCK_MATCH_IMAGE_QCOM, VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE, VK_DESCRIPTOR_TYPE_STORAGE_IMAGE, or VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT is accessed as a result of this command, the image subresource identified by that descriptor **must** be in the image layout identified when the descriptor was written
- VUID-vkCmdDrawMultiEXT-renderPass-02684
  The current render pass **must** be compatible with the renderPass member of the VkGraphicsPipelineCreateInfo structure specified when creating the VkPipeline bound to VK_PIPELINE_BIND_POINT_GRAPHICS
- VUID-vkCmdDrawMultiEXT-subpass-02685
  The subpass index of the current render pass **must** be equal to the subpass member of the VkGraphicsPipelineCreateInfo structure specified when creating the VkPipeline bound to VK_PIPELINE_BIND_POINT_GRAPHICS
- VUID-vkCmdDrawMultiEXT-None-07748
  If any shader statically accesses an input attachment, a valid descriptor **must** be bound to the pipeline via a descriptor set
- VUID-vkCmdDrawMultiEXT-OpTypeImage-07468
  If any shader executed by this pipeline accesses an OpTypeImage variable with a Dim operand of SubpassData, it **must** be decorated with an InputAttachmentIndex that corresponds to a valid input attachment in the current subpass
- VUID-vkCmdDrawMultiEXT-None-07469
  Input attachment views accessed in a subpass **must** be created with the same VkFormat as the corresponding subpass definition, and be created with a VkImageView that is compatible with the attachment referenced by the subpass' pInputAttachments [InputAttachmentIndex] in the currently bound VkFramebuffer as specified by Fragment Input Attachment Compatibility
- VUID-vkCmdDrawMultiEXT-pDepthInputAttachmentIndex-09595
  Input attachment views accessed in a dynamic render pass with a InputAttachmentIndex referenced by VkRenderingInputAttachmentIndexInfoKHR, or no InputAttachmentIndex if VkRenderingInputAttachmentIndexInfoKHR:pDepthInputAttachmentIndex or VkRenderingInputAttachmentIndexInfoKHR:pStencilInputAttachmentIndex are NULL, **must** be created with a VkImageView that is compatible with the corresponding color, depth, or

stencil attachment in VkRenderingInfo

- VUID-vkCmdDrawMultiEXT-pDepthInputAttachmentIndex-09596
  Input attachment views accessed in a dynamic render pass via a shader object **must** have an `InputAttachmentIndex` if both VkRenderingInputAttachmentIndexInfoKHR :`pDepthInputAttachmentIndex` and VkRenderingInputAttachmentIndexInfoKHR :`pStencilInputAttachmentIndex` are non-`NULL`

- VUID-vkCmdDrawMultiEXT-InputAttachmentIndex-09597
  If an input attachment view accessed in a dynamic render pass via a shader object has an `InputAttachmentIndex`, the `InputAttachmentIndex` **must** match an index in VkRenderingInputAttachmentIndexInfoKHR

- VUID-vkCmdDrawMultiEXT-None-06537
  Memory backing image subresources used as attachments in the current render pass **must** not be written in any way other than as an attachment by this command

- VUID-vkCmdDrawMultiEXT-None-09000
  If a color attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass, it is not in the `VK_IMAGE_LAYOUT_ATTACHMENT_FEEDBACK_LOOP_OPTIMAL_EXT` image layout, and either:

  - the `VK_PIPELINE_CREATE_COLOR_ATTACHMENT_FEEDBACK_LOOP_BIT_EXT` is set on the currently bound pipeline or
  - the last call to vkCmdSetAttachmentFeedbackLoopEnableEXT included `VK_IMAGE_ASPECT_COLOR_BIT` and
    - there is no currently bound graphics pipeline or
    - the currently bound graphics pipeline was created with `VK_DYNAMIC_STATE_ATTACHMENT_FEEDBACK_LOOP_ENABLE_EXT`

  it **must** not be accessed in any way other than as an attachment by this command

- VUID-vkCmdDrawMultiEXT-None-09001
  If a depth attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass, it is not in the `VK_IMAGE_LAYOUT_ATTACHMENT_FEEDBACK_LOOP_OPTIMAL_EXT` image layout, and either:

  - the `VK_PIPELINE_CREATE_DEPTH_STENCIL_ATTACHMENT_FEEDBACK_LOOP_BIT_EXT` is set on the currently bound pipeline or
  - the last call to vkCmdSetAttachmentFeedbackLoopEnableEXT included `VK_IMAGE_ASPECT_DEPTH_BIT` and
    - there is no currently bound graphics pipeline or
    - the currently bound graphics pipeline was created with `VK_DYNAMIC_STATE_ATTACHMENT_FEEDBACK_LOOP_ENABLE_EXT`

  it **must** not be accessed in any way other than as an attachment by this command

- VUID-vkCmdDrawMultiEXT-None-09002
  If a stencil attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass, it is not in the

VK_IMAGE_LAYOUT_ATTACHMENT_FEEDBACK_LOOP_OPTIMAL_EXT image layout, and either:

  - the VK_PIPELINE_CREATE_DEPTH_STENCIL_ATTACHMENT_FEEDBACK_LOOP_BIT_EXT is set on the currently bound pipeline or
  - the last call to vkCmdSetAttachmentFeedbackLoopEnableEXT included VK_IMAGE_ASPECT_STENCIL_BIT and
    - there is no currently bound graphics pipeline or
    - the currently bound graphics pipeline was created with VK_DYNAMIC_STATE_ATTACHMENT_FEEDBACK_LOOP_ENABLE_EXT

  it **must** not be accessed in any way other than as an attachment by this command

- VUID-vkCmdDrawMultiEXT-None-09003
  If an attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass, it **must** not be accessed in any way other than as an attachment, storage image, or sampled image by this command
- VUID-vkCmdDrawMultiEXT-None-06539
  If any previously recorded command in the current subpass accessed an image subresource used as an attachment in this subpass in any way other than as an attachment, this command **must** not write to that image subresource as an attachment
- VUID-vkCmdDrawMultiEXT-None-06886
  If the current render pass instance uses a depth/stencil attachment with a read-only layout for the depth aspect, depth writes **must** be disabled
- VUID-vkCmdDrawMultiEXT-None-06887
  If the current render pass instance uses a depth/stencil attachment with a read-only layout for the stencil aspect, both front and back writeMask are not zero, and stencil test is enabled, all stencil ops **must** be VK_STENCIL_OP_KEEP
- VUID-vkCmdDrawMultiEXT-None-07831
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT dynamic state enabled then vkCmdSetViewport **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiEXT-None-07832
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SCISSOR dynamic state enabled then vkCmdSetScissor **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiEXT-None-07833
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_WIDTH dynamic state enabled then vkCmdSetLineWidth **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiEXT-None-08617
  If a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetPolygonModeEXT in the current command buffer set polygonMode to VK_POLYGON_MODE_LINE, vkCmdSetLineWidth **must** have been called and not subsequently

invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMultiEXT-None-08618
  If a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetPrimitiveTopologyEXT in the current command buffer set primitiveTopology to any line topology, vkCmdSetLineWidth **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMultiEXT-None-08619
  If a shader object that outputs line primitives is bound to the VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT or VK_SHADER_STAGE_GEOMETRY_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, vkCmdSetLineWidth **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMultiEXT-None-07834
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_BIAS dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of depthBiasEnable is VK_TRUE, then vkCmdSetDepthBounds or vkCmdSetDepthBias2EXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMultiEXT-None-07835
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_BLEND_CONSTANTS dynamic state enabled then vkCmdSetBlendConstants **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMultiEXT-None-08621
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetColorBlendEnableEXT in the current command buffer set any element of pColorBlendEnables to VK_TRUE, and the most recent call to vkCmdSetColorBlendEquationEXT in the current command buffer set the same element of pColorBlendEquations to a VkColorBlendEquationEXT structure with any VkBlendFactor member with a value of VK_BLEND_FACTOR_CONSTANT_COLOR, VK_BLEND_FACTOR_ONE_MINUS_CONSTANT_COLOR, VK_BLEND_FACTOR_CONSTANT_ALPHA, or VK_BLEND_FACTOR_ONE_MINUS_CONSTANT_ALPHA, vkCmdSetBlendConstants **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMultiEXT-None-07836
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_BOUNDS dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of depthBoundsTestEnable is VK_TRUE, then vkCmdSetDepthBounds **must** have been called and

not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMultiEXT-None-07837
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_STENCIL_COMPARE_MASK dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of stencilTestEnable is VK_TRUE, then vkCmdSetStencilCompareMask **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiEXT-None-07838
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_STENCIL_WRITE_MASK dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of stencilTestEnable is VK_TRUE, then vkCmdSetStencilWriteMask **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiEXT-None-07839
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_STENCIL_REFERENCE dynamic state enabled, the current value of and rasterizerDiscardEnable is VK_FALSE, the current value of stencilTestEnable is VK_TRUE, then vkCmdSetStencilReference **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiEXT-maxMultiviewInstanceIndex-02688
  If the draw is recorded in a render pass instance with multiview enabled, the maximum instance index **must** be less than or equal to VkPhysicalDeviceMultiviewProperties::maxMultiviewInstanceIndex
- VUID-vkCmdDrawMultiEXT-sampleLocationsEnable-02689
  If the bound graphics pipeline was created with VkPipelineSampleLocationsStateCreateInfoEXT::sampleLocationsEnable set to VK_TRUE and the current subpass has a depth/stencil attachment, then that attachment **must** have been created with the VK_IMAGE_CREATE_SAMPLE_LOCATIONS_COMPATIBLE_DEPTH_BIT_EXT bit set
- VUID-vkCmdDrawMultiEXT-None-06666
  If the VK_EXT_sample_locations extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of sampleLocationsEnable is VK_TRUE, then vkCmdSetSampleLocationsEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiEXT-None-07840
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_CULL_MODE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetCullModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMultiEXT-None-07841
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_FRONT_FACE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetFrontFaceEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiEXT-None-07843
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_TEST_ENABLE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, vkCmdSetDepthTestEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiEXT-None-07844
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_WRITE_ENABLE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetDepthWriteEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiEXT-None-07845
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_COMPARE_OP dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of depthTestEnable is VK_TRUE, then vkCmdSetDepthCompareOpEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiEXT-None-07846
  If the depthBounds feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_BOUNDS_TEST_ENABLE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetDepthBoundsTestEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiEXT-None-07847
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_STENCIL_TEST_ENABLE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetStencilTestEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiEXT-None-07848
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_STENCIL_OP dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, the current value of stencilTestEnable is VK_TRUE, then vkCmdSetStencilOpEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiEXT-viewportCount-03417

If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, but not the VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT dynamic state enabled, then vkCmdSetViewportWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the viewportCount parameter of vkCmdSetViewportWithCountEXT **must** match the VkPipelineViewportStateCreateInfo::scissorCount of the pipeline

- VUID-vkCmdDrawMultiEXT-scissorCount-03418
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT dynamic state enabled, but not the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, then vkCmdSetScissorWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the scissorCount parameter of vkCmdSetScissorWithCountEXT **must** match the VkPipelineViewportStateCreateInfo::viewportCount of the pipeline

- VUID-vkCmdDrawMultiEXT-viewportCount-03419
  If the bound graphics pipeline state was created with both the VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT and VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic states enabled then both vkCmdSetViewportWithCountEXT and vkCmdSetScissorWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the viewportCount parameter of vkCmdSetViewportWithCountEXT **must** match the scissorCount parameter of vkCmdSetScissorWithCountEXT

- VUID-vkCmdDrawMultiEXT-None-08635
  If a shader object is bound to any graphics stage, then both vkCmdSetViewportWithCountEXT and vkCmdSetScissorWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the viewportCount parameter of vkCmdSetViewportWithCountEXT **must** match the scissorCount parameter of vkCmdSetScissorWithCountEXT

- VUID-vkCmdDrawMultiEXT-viewportCount-04137
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, but not the VK_DYNAMIC_STATE_VIEWPORT_W_SCALING_NV dynamic state enabled, then the bound graphics pipeline **must** have been created with VkPipelineViewportWScalingStateCreateInfoNV::viewportCount greater or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT

- VUID-vkCmdDrawMultiEXT-viewportCount-04138
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT and VK_DYNAMIC_STATE_VIEWPORT_W_SCALING_NV dynamic states enabled then the viewportCount parameter in the last call to vkCmdSetViewportWScalingNV **must** be greater than or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT

- VUID-vkCmdDrawMultiEXT-None-09232
  If the VK_NV_clip_space_w_scaling extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetViewportWScalingEnableNV in

the current command buffer set viewportWScalingEnable to VK_TRUE, then vkCmdSetViewportWScalingNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMultiEXT-None-08636
  If the VK_NV_clip_space_w_scaling extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetViewportWScalingEnableNV in the current command buffer set viewportWScalingEnable to VK_TRUE, then the viewportCount parameter in the last call to vkCmdSetViewportWScalingNV **must** be greater than or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT

- VUID-vkCmdDrawMultiEXT-viewportCount-04139
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, but not the VK_DYNAMIC_STATE_VIEWPORT_SHADING_RATE_PALETTE_NV dynamic state enabled, then the bound graphics pipeline **must** have been created with VkPipelineViewportShadingRateImageStateCreateInfoNV::viewportCount greater or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT

- VUID-vkCmdDrawMultiEXT-viewportCount-04140
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT and VK_DYNAMIC_STATE_VIEWPORT_SHADING_RATE_PALETTE_NV dynamic states enabled then the viewportCount parameter in the last call to vkCmdSetViewportShadingRatePaletteNV **must** be greater than or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT

- VUID-vkCmdDrawMultiEXT-shadingRateImage-09233
  If the shadingRateImage feature is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetCoarseSampleOrderNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMultiEXT-shadingRateImage-09234
  If the shadingRateImage feature is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetShadingRateImageEnableNV in the current command buffer set shadingRateImageEnable to VK_TRUE, then vkCmdSetViewportShadingRatePaletteNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMultiEXT-None-08637
  If the shadingRateImage feature is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetShadingRateImageEnableNV in the current command buffer set shadingRateImageEnable to VK_TRUE, then the viewportCount parameter in the last call to vkCmdSetViewportShadingRatePaletteNV **must** be greater than or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT

- VUID-vkCmdDrawMultiEXT-VkPipelineVieportCreateInfo-04141
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT` dynamic state enabled and a VkPipelineViewportSwizzleStateCreateInfoNV structure chained from VkPipelineViewportStateCreateInfo, then the bound graphics pipeline **must** have been created with VkPipelineViewportSwizzleStateCreateInfoNV::`viewportCount` greater or equal to the `viewportCount` parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawMultiEXT-VkPipelineVieportCreateInfo-04142
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT` dynamic state enabled and a VkPipelineViewportExclusiveScissorStateCreateInfoNV structure chained from VkPipelineViewportStateCreateInfo, then the bound graphics pipeline **must** have been created with VkPipelineViewportExclusiveScissorStateCreateInfoNV::`exclusiveScissorCount` greater or equal to the `viewportCount` parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawMultiEXT-None-07878
  If the `exclusiveScissor` feature is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_EXCLUSIVE_SCISSOR_ENABLE_NV` dynamic state enabled, then vkCmdSetExclusiveScissorEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiEXT-None-07879
  If the `exclusiveScissor` feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_EXCLUSIVE_SCISSOR_NV` dynamic state enabled, and the most recent call to vkCmdSetExclusiveScissorEnableNV in the current command buffer set any element of `pExclusiveScissorEnables` to `VK_TRUE`, then vkCmdSetExclusiveScissorNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiEXT-None-04876
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_RASTERIZER_DISCARD_ENABLE` dynamic state enabled, then vkCmdSetRasterizerDiscardEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiEXT-None-04877
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_DEPTH_BIAS_ENABLE` dynamic state enabled, and the current value of `rasterizerDiscardEnable` is `VK_FALSE`, then vkCmdSetDepthBiasEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiEXT-logicOp-04878
  If a shader object is bound to the `VK_SHADER_STAGE_FRAGMENT_BIT` or a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_LOGIC_OP_EXT` dynamic state enabled, the current value of `rasterizerDiscardEnable` is `VK_FALSE`, and the current value of `logicOpEnable` is `VK_TRUE`, then vkCmdSetLogicOpEXT **must** have been called and not

subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMultiEXT-primitiveFragmentShadingRateWithMultipleViewports-04552
  If the `primitiveFragmentShadingRateWithMultipleViewports` limit is not supported, the bound graphics pipeline was created with the `VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT` dynamic state enabled, and any of the shader stages of the bound graphics pipeline write to the `PrimitiveShadingRateKHR` built-in, then vkCmdSetViewportWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the `viewportCount` parameter of `vkCmdSetViewportWithCountEXT` **must** be `1`
- VUID-vkCmdDrawMultiEXT-primitiveFragmentShadingRateWithMultipleViewports-08642
  If the `primitiveFragmentShadingRateWithMultipleViewports` limit is not supported, and any shader object bound to a graphics stage writes to the `PrimitiveShadingRateKHR` built-in, then vkCmdSetViewportWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the `viewportCount` parameter of `vkCmdSetViewportWithCountEXT` **must** be `1`
- VUID-vkCmdDrawMultiEXT-blendEnable-04727
  If rasterization is not disabled in the bound graphics pipeline, then for each color attachment in the subpass, if the corresponding image view's format features do not contain `VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT`, then the `blendEnable` member of the corresponding element of the `pAttachments` member of `pColorBlendState` **must** be `VK_FALSE`
- VUID-vkCmdDrawMultiEXT-None-08643
  If a shader object is bound to the `VK_SHADER_STAGE_FRAGMENT_BIT` stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE`, then for each color attachment in the render pass, if the corresponding image view's format features do not contain `VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT`, then the corresponding member of `pColorBlendEnables` in the most recent call to `vkCmdSetColorBlendEnableEXT` in the current command buffer that affected that attachment index **must** have been `VK_FALSE`
- VUID-vkCmdDrawMultiEXT-multisampledRenderToSingleSampled-07284
  If rasterization is not disabled in the bound graphics pipeline, and none of the following is enabled:
  - the `VK_AMD_mixed_attachment_samples` extension
  - the `VK_NV_framebuffer_mixed_samples` extension
  - the `multisampledRenderToSingleSampled` feature

  then `rasterizationSamples` for the currently bound graphics pipeline **must** be the same as the current subpass color and/or depth/stencil attachments
- VUID-vkCmdDrawMultiEXT-None-08644
  If a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE`, and none of the following is enabled:
  - the `VK_AMD_mixed_attachment_samples` extension
  - the `VK_NV_framebuffer_mixed_samples` extension

- the `multisampledRenderToSingleSampled` feature

then the most recent call to vkCmdSetRasterizationSamplesEXT in the current command buffer **must** have set `rasterizationSamples` to be the same as the number of samples for the current render pass color and/or depth/stencil attachments

- VUID-vkCmdDrawMultiEXT-None-08876
  If a shader object is bound to any graphics stage, the current render pass instance **must** have been begun with vkCmdBeginRenderingKHR
- VUID-vkCmdDrawMultiEXT-imageView-06172
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the `imageView` member of `pDepthAttachment` is not VK_NULL_HANDLE, and the `layout` member of `pDepthAttachment` is `VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL`, this command **must** not write any values to the depth attachment
- VUID-vkCmdDrawMultiEXT-imageView-06173
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the `imageView` member of `pStencilAttachment` is not VK_NULL_HANDLE, and the `layout` member of `pStencilAttachment` is `VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL`, this command **must** not write any values to the stencil attachment
- VUID-vkCmdDrawMultiEXT-imageView-06174
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the `imageView` member of `pDepthAttachment` is not VK_NULL_HANDLE, and the `layout` member of `pDepthAttachment` is `VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_STENCIL_ATTACHMENT_OPTIMAL`, this command **must** not write any values to the depth attachment
- VUID-vkCmdDrawMultiEXT-imageView-06175
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the `imageView` member of `pStencilAttachment` is not VK_NULL_HANDLE, and the `layout` member of `pStencilAttachment` is `VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_STENCIL_READ_ONLY_OPTIMAL`, this command **must** not write any values to the stencil attachment
- VUID-vkCmdDrawMultiEXT-imageView-06176
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the `imageView` member of `pDepthAttachment` is not VK_NULL_HANDLE, and the `layout` member of `pDepthAttachment` is `VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL`, this command **must** not write any values to the depth attachment
- VUID-vkCmdDrawMultiEXT-imageView-06177
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the `imageView` member of `pStencilAttachment` is not VK_NULL_HANDLE, and the `layout` member of `pStencilAttachment` is `VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL`, this command **must** not write any values to the stencil attachment
- VUID-vkCmdDrawMultiEXT-viewMask-06178
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the currently bound graphics pipeline **must** have been created with a VkPipelineRenderingCreateInfo::`viewMask` equal to VkRenderingInfo::`viewMask`
- VUID-vkCmdDrawMultiEXT-colorAttachmentCount-06179

If the dynamicRenderingUnusedAttachments feature is not enabled and the current render pass instance was begun with vkCmdBeginRenderingKHR, the currently bound graphics pipeline **must** have been created with a VkPipelineRenderingCreateInfo::colorAttachmentCount equal to VkRenderingInfo::colorAttachmentCount

- VUID-vkCmdDrawMultiEXT-dynamicRenderingUnusedAttachments-08910
  If the dynamicRenderingUnusedAttachments feature is not enabled, and the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingInfo::colorAttachmentCount greater than 0, then each element of the VkRenderingInfo::pColorAttachments array with an imageView not equal to VK_NULL_HANDLE **must** have been created with a VkFormat equal to the corresponding element of VkPipelineRenderingCreateInfo::pColorAttachmentFormats used to create the currently bound graphics pipeline
- VUID-vkCmdDrawMultiEXT-dynamicRenderingUnusedAttachments-08912
  If the dynamicRenderingUnusedAttachments feature is not enabled, and the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingInfo::colorAttachmentCount greater than 0, then each element of the VkRenderingInfo::pColorAttachments array with an imageView equal to VK_NULL_HANDLE **must** have the corresponding element of VkPipelineRenderingCreateInfo::pColorAttachmentFormats used to create the currently bound pipeline equal to VK_FORMAT_UNDEFINED
- VUID-vkCmdDrawMultiEXT-dynamicRenderingUnusedAttachments-08911
  If the dynamicRenderingUnusedAttachments feature is enabled, and the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingInfo::colorAttachmentCount greater than 0, then each element of the VkRenderingInfo::pColorAttachments array with an imageView not equal to VK_NULL_HANDLE **must** have been created with a VkFormat equal to the corresponding element of VkPipelineRenderingCreateInfo::pColorAttachmentFormats used to create the currently bound graphics pipeline, or the corresponding element of VkPipelineRenderingCreateInfo::pColorAttachmentFormats, if it exists, **must** be VK_FORMAT_UNDEFINED
- VUID-vkCmdDrawMultiEXT-colorAttachmentCount-09362
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, with a VkRenderingInfo::colorAttachmentCount equal to 1, there is no shader object bound to any graphics stage, and a color attachment with a resolve mode of VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID, each element of the VkRenderingInfo::pColorAttachments array with a resolveImageView not equal to VK_NULL_HANDLE **must** have been created with an image created with a VkExternalFormatANDROID::externalFormat value equal to the VkExternalFormatANDROID::externalFormat value used to create the currently bound graphics pipeline
- VUID-vkCmdDrawMultiEXT-None-09363
  If there is no shader object bound to any graphics stage, the current render pass instance was begun with vkCmdBeginRenderingKHR and a VkRenderingInfo::colorAttachmentCount equal to 1, and a color attachment with a resolve mode of VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID, each element of the VkRenderingInfo::pColorAttachments array with a imageView not equal to VK_NULL_HANDLE **must** have been created with an image created with a

VkExternalFormatANDROID::externalFormat value equal to the VkExternalFormatANDROID::externalFormat value used to create the currently bound graphics pipeline

- VUID-vkCmdDrawMultiEXT-None-09364
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is no shader object bound to any graphics stage, and the currently bound graphics pipeline was created with a non-zero VkExternalFormatANDROID::externalFormat value and with the VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT dynamic state enabled, then vkCmdSetColorBlendEnableEXT **must** have set the blend enable to VK_FALSE prior to this drawing command
- VUID-vkCmdDrawMultiEXT-None-09365
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is no shader object bound to any graphics stage, and the currently bound graphics pipeline was created with a non-zero VkExternalFormatANDROID::externalFormat value and with the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT dynamic state enabled, then vkCmdSetRasterizationSamplesEXT **must** have set rasterizationSamples to VK_SAMPLE_COUNT_1_BIT prior to this drawing command
- VUID-vkCmdDrawMultiEXT-None-09366
  If there is a shader object bound to any graphics stage, and the current render pass includes a color attachment that uses the VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID resolve mode, then vkCmdSetColorBlendEnableEXT **must** have set blend enable to VK_FALSE prior to this drawing command
- VUID-vkCmdDrawMultiEXT-rasterizationSamples-09367
  If there is a shader object bound to any graphics stage, and the current render pass includes a color attachment that uses the VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID resolve mode, then vkCmdSetRasterizationSamplesEXT **must** have set rasterizationSamples to VK_SAMPLE_COUNT_1_BIT prior to this drawing command
- VUID-vkCmdDrawMultiEXT-None-09368
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is no shader object bound to any graphics stage, and the currently bound graphics pipeline was created with a non-zero VkExternalFormatANDROID::externalFormat value and with the VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR dynamic state enabled, then vkCmdSetFragmentShadingRateKHR **must** have set pFragmentSize->width to 1 prior to this drawing command
- VUID-vkCmdDrawMultiEXT-None-09369
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is no shader object bound to any graphics stage, and the currently bound graphics pipeline was created with a non-zero VkExternalFormatANDROID::externalFormat value and with the VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR dynamic state enabled, then vkCmdSetFragmentShadingRateKHR **must** have set pFragmentSize->height to 1 prior to this drawing command
- VUID-vkCmdDrawMultiEXT-pFragmentSize-09370
  If there is a shader object bound to any graphics stage, and the current render pass

includes a color attachment that uses the VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID resolve mode, then vkCmdSetFragmentShadingRateKHR **must** have set pFragmentSize->width to 1 prior to this drawing command

- VUID-vkCmdDrawMultiEXT-pFragmentSize-09371
  If there is a shader object bound to any graphics stage, and the current render pass includes a color attachment that uses the VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID resolve mode, then vkCmdSetFragmentShadingRateKHR **must** have set pFragmentSize->height to 1 prior to this drawing command
- VUID-vkCmdDrawMultiEXT-None-07749
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_WRITE_ENABLE_EXT dynamic state enabled then vkCmdSetColorWriteEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiEXT-None-08646
  If the colorWriteEnable feature is enabled on the device, and a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetColorWriteEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiEXT-attachmentCount-07750
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_WRITE_ENABLE_EXT dynamic state enabled then the attachmentCount parameter of vkCmdSetColorWriteEnableEXT **must** be greater than or equal to the VkPipelineColorBlendStateCreateInfo::attachmentCount of the currently bound graphics pipeline
- VUID-vkCmdDrawMultiEXT-None-08647
  If the colorWriteEnable feature is enabled on the device, and a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then the attachmentCount parameter of most recent call to vkCmdSetColorWriteEnableEXT in the current command buffer **must** be greater than or equal to the number of color attachments in the current render pass instance
- VUID-vkCmdDrawMultiEXT-None-07751
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_DISCARD_RECTANGLE_EXT dynamic state enabled then vkCmdSetDiscardRectangleEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command for each discard rectangle in VkPipelineDiscardRectangleStateCreateInfoEXT::discardRectangleCount
- VUID-vkCmdDrawMultiEXT-None-07880
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_DISCARD_RECTANGLE_ENABLE_EXT dynamic state enabled then vkCmdSetDiscardRectangleEnableEXT **must** have been called and not subsequently

invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMultiEXT-rasterizerDiscardEnable-09236
  If the VK_EXT_discard_rectangles extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetDiscardRectangleEnableEXT in the current command buffer set discardRectangleEnable to VK_TRUE, then vkCmdSetDiscardRectangleEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMultiEXT-None-08648
  If the VK_EXT_discard_rectangles extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetDiscardRectangleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMultiEXT-None-07881
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_DISCARD_RECTANGLE_MODE_EXT dynamic state enabled then vkCmdSetDiscardRectangleModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMultiEXT-None-08649
  If the VK_EXT_discard_rectangles extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetDiscardRectangleEnableEXT in the current command buffer set discardRectangleEnable to VK_TRUE, then vkCmdSetDiscardRectangleModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMultiEXT-dynamicRenderingUnusedAttachments-08913
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is not enabled, and VkRenderingInfo::pDepthAttachment->imageView was VK_NULL_HANDLE, the value of VkPipelineRenderingCreateInfo::depthAttachmentFormat used to create the currently bound graphics pipeline **must** be equal to VK_FORMAT_UNDEFINED

- VUID-vkCmdDrawMultiEXT-dynamicRenderingUnusedAttachments-08914
  If current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is not enabled, and VkRenderingInfo::pDepthAttachment->imageView was not VK_NULL_HANDLE, the value of VkPipelineRenderingCreateInfo::depthAttachmentFormat used to create the currently bound graphics pipeline **must** be equal to the VkFormat used to create VkRenderingInfo::pDepthAttachment->imageView

- VUID-vkCmdDrawMultiEXT-dynamicRenderingUnusedAttachments-08915
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is enabled, VkRenderingInfo::pDepthAttachment->imageView was not VK_NULL_HANDLE, and the value of

VkPipelineRenderingCreateInfo::depthAttachmentFormat used to create the currently bound graphics pipeline was not equal to the VkFormat used to create VkRenderingInfo::pDepthAttachment->imageView, the value of the format **must** be VK_FORMAT_UNDEFINED

- VUID-vkCmdDrawMultiEXT-dynamicRenderingUnusedAttachments-08916
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is not enabled, and VkRenderingInfo::pStencilAttachment->imageView was VK_NULL_HANDLE, the value of VkPipelineRenderingCreateInfo::stencilAttachmentFormat used to create the currently bound graphics pipeline **must** be equal to VK_FORMAT_UNDEFINED
- VUID-vkCmdDrawMultiEXT-dynamicRenderingUnusedAttachments-08917
  If current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is not enabled, and VkRenderingInfo::pStencilAttachment->imageView was not VK_NULL_HANDLE, the value of VkPipelineRenderingCreateInfo::stencilAttachmentFormat used to create the currently bound graphics pipeline **must** be equal to the VkFormat used to create VkRenderingInfo::pStencilAttachment->imageView
- VUID-vkCmdDrawMultiEXT-dynamicRenderingUnusedAttachments-08918
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is enabled, VkRenderingInfo::pStencilAttachment->imageView was not VK_NULL_HANDLE, and the value of VkPipelineRenderingCreateInfo::stencilAttachmentFormat used to create the currently bound graphics pipeline was not equal to the VkFormat used to create VkRenderingInfo::pStencilAttachment->imageView, the value of the format **must** be VK_FORMAT_UNDEFINED
- VUID-vkCmdDrawMultiEXT-imageView-06183
  If the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingFragmentShadingRateAttachmentInfoKHR::imageView was not VK_NULL_HANDLE, the currently bound graphics pipeline **must** have been created with VK_PIPELINE_CREATE_RENDERING_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR
- VUID-vkCmdDrawMultiEXT-imageView-06184
  If the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingFragmentDensityMapAttachmentInfoEXT::imageView was not VK_NULL_HANDLE, the currently bound graphics pipeline **must** have been created with VK_PIPELINE_CREATE_RENDERING_FRAGMENT_DENSITY_MAP_ATTACHMENT_BIT_EXT
- VUID-vkCmdDrawMultiEXT-colorAttachmentCount-06185
  If the currently bound pipeline was created with a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and the current render pass instance was begun with vkCmdBeginRenderingKHR with a VkRenderingInfo::colorAttachmentCount parameter greater than 0, then each element of the VkRenderingInfo::pColorAttachments array with a imageView not equal to VK_NULL_HANDLE **must** have been created with a sample count equal to the corresponding element of the pColorAttachmentSamples member of VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV used to create the currently bound graphics pipeline
- VUID-vkCmdDrawMultiEXT-pDepthAttachment-06186
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the currently bound pipeline was created with a VkAttachmentSampleCountInfoAMD or

VkAttachmentSampleCountInfoNV structure, and VkRenderingInfo::pDepthAttachment->imageView was not VK_NULL_HANDLE, the value of the depthStencilAttachmentSamples member of VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV used to create the currently bound graphics pipeline **must** be equal to the sample count used to create VkRenderingInfo::pDepthAttachment->imageView

- VUID-vkCmdDrawMultiEXT-pStencilAttachment-06187
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the currently bound pipeline was created with a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and VkRenderingInfo::pStencilAttachment->imageView was not VK_NULL_HANDLE, the value of the depthStencilAttachmentSamples member of VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV used to create the currently bound graphics pipeline **must** be equal to the sample count used to create VkRenderingInfo::pStencilAttachment->imageView
- VUID-vkCmdDrawMultiEXT-multisampledRenderToSingleSampled-07285
  If the currently bound pipeline was created without a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and the multisampledRenderToSingleSampled feature is not enabled, and the current render pass instance was begun with vkCmdBeginRenderingKHR with a VkRenderingInfo::colorAttachmentCount parameter greater than 0, then each element of the VkRenderingInfo::pColorAttachments array with a imageView not equal to VK_NULL_HANDLE **must** have been created with a sample count equal to the value of rasterizationSamples for the currently bound graphics pipeline
- VUID-vkCmdDrawMultiEXT-multisampledRenderToSingleSampled-07286
  If the currently bound pipeline was created without a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and the multisampledRenderToSingleSampled feature is not enabled, and VkRenderingInfo::pDepthAttachment->imageView was not VK_NULL_HANDLE, the value of rasterizationSamples for the currently bound graphics pipeline **must** be equal to the sample count used to create VkRenderingInfo::pDepthAttachment->imageView
- VUID-vkCmdDrawMultiEXT-multisampledRenderToSingleSampled-07287
  If the currently bound pipeline was created without a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and the multisampledRenderToSingleSampled feature is not enabled, and VkRenderingInfo::pStencilAttachment->imageView was not VK_NULL_HANDLE, the value of rasterizationSamples for the currently bound graphics pipeline **must** be equal to the sample count used to create VkRenderingInfo::pStencilAttachment->imageView
- VUID-vkCmdDrawMultiEXT-pNext-07935
  If this command has been called inside a render pass instance started with vkCmdBeginRenderingKHR, and the pNext chain of VkRenderingInfo includes a VkMultisampledRenderToSingleSampledInfoEXT structure with multisampledRenderToSingleSampledEnable equal to VK_TRUE, then the value of rasterizationSamples for the currently bound graphics pipeline **must** be equal to VkMultisampledRenderToSingleSampledInfoEXT::rasterizationSamples
- VUID-vkCmdDrawMultiEXT-renderPass-06198
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the

currently bound pipeline **must** have been created with a VkGraphicsPipelineCreateInfo::renderPass equal to VK_NULL_HANDLE

- VUID-vkCmdDrawMultiEXT-pColorAttachments-08963
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is a graphics pipeline bound with a fragment shader that statically writes to a color attachment, the color write mask is not zero, color writes are enabled, and the corresponding element of the VkRenderingInfo::pColorAttachments->imageView was not VK_NULL_HANDLE, then the corresponding element of VkPipelineRenderingCreateInfo::pColorAttachmentFormats used to create the pipeline **must** not be VK_FORMAT_UNDEFINED
- VUID-vkCmdDrawMultiEXT-pDepthAttachment-08964
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is a graphics pipeline bound, depth test is enabled, depth write is enabled, and the VkRenderingInfo::pDepthAttachment->imageView was not VK_NULL_HANDLE, then the VkPipelineRenderingCreateInfo::depthAttachmentFormat used to create the pipeline **must** not be VK_FORMAT_UNDEFINED
- VUID-vkCmdDrawMultiEXT-pStencilAttachment-08965
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is a graphics pipeline bound, stencil test is enabled and the VkRenderingInfo::pStencilAttachment->imageView was not VK_NULL_HANDLE, then the VkPipelineRenderingCreateInfo::stencilAttachmentFormat used to create the pipeline **must** not be VK_FORMAT_UNDEFINED
- VUID-vkCmdDrawMultiEXT-primitivesGeneratedQueryWithRasterizerDiscard-06708
  If the primitivesGeneratedQueryWithRasterizerDiscard feature is not enabled and the VK_QUERY_TYPE_PRIMITIVES_GENERATED_EXT query is active, rasterization discard **must** not be enabled
- VUID-vkCmdDrawMultiEXT-primitivesGeneratedQueryWithNonZeroStreams-06709
  If the primitivesGeneratedQueryWithNonZeroStreams feature is not enabled and the VK_QUERY_TYPE_PRIMITIVES_GENERATED_EXT query is active, the bound graphics pipeline **must** not have been created with a non-zero value in VkPipelineRasterizationStateStreamCreateInfoEXT::rasterizationStream
- VUID-vkCmdDrawMultiEXT-None-07619
  If a shader object is bound to the VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_TESSELLATION_DOMAIN_ORIGIN_EXT dynamic state enabled, then vkCmdSetTessellationDomainOriginEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiEXT-None-07620
  If the depthClamp feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_CLAMP_ENABLE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetDepthClampEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiEXT-None-07621
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which

was created with the VK_DYNAMIC_STATE_POLYGON_MODE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetPolygonModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMultiEXT-None-07622
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetRasterizationSamplesEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMultiEXT-None-07623
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_SAMPLE_MASK_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetSampleMaskEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMultiEXT-alphaToCoverageEnable-08919
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_ALPHA_TO_COVERAGE_ENABLE_EXT dynamic state enabled, and alphaToCoverageEnable was VK_TRUE in the last call to vkCmdSetAlphaToCoverageEnableEXT, then the Fragment Output Interface **must** contain a variable for the alpha Component word in Location 0 at Index 0

- VUID-vkCmdDrawMultiEXT-alphaToCoverageEnable-08920
  If a shader object is bound to any graphics stage, and the most recent call to vkCmdSetAlphaToCoverageEnableEXT in the current command buffer set alphaToCoverageEnable to VK_TRUE, then the Fragment Output Interface **must** contain a variable for the alpha Component word in Location 0 at Index 0

- VUID-vkCmdDrawMultiEXT-None-07624
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_ALPHA_TO_COVERAGE_ENABLE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetAlphaToCoverageEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMultiEXT-None-07625
  If the alphaToOne feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_ALPHA_TO_ONE_ENABLE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetAlphaToOneEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMultiEXT-None-07626
  If the logicOp feature is enabled, a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_LOGIC_OP_ENABLE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetLogicOpEnableEXT **must**

have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMultiEXT-None-07627
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT dynamic state enabled then vkCmdSetColorBlendEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiEXT-None-08657
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and both the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE and there are color attachments bound, then vkCmdSetColorBlendEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiEXT-None-07628
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_EQUATION_EXT dynamic state enabled then vkCmdSetColorBlendEquationEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiEXT-None-08658
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetColorBlendEnableEXT for any attachment set that attachment's value in pColorBlendEnables to VK_TRUE, then vkCmdSetColorBlendEquationEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiEXT-None-07629
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_WRITE_MASK_EXT dynamic state enabled then vkCmdSetColorWriteMaskEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiEXT-None-08659
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and both the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE and there are color attachments bound, then vkCmdSetColorWriteMaskEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiEXT-None-07630
  If the geometryStreams feature is enabled, and a shader object is bound to the VK_SHADER_STAGE_GEOMETRY_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_RASTERIZATION_STREAM_EXT dynamic state enabled, then vkCmdSetRasterizationStreamEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiEXT-None-07631
  If the VK_EXT_conservative_rasterization extension is enabled, a shader object is bound to

any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_CONSERVATIVE_RASTERIZATION_MODE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetConservativeRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMultiEXT-None-07632
  If the VK_EXT_conservative_rasterization extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_EXTRA_PRIMITIVE_OVERESTIMATION_SIZE_EXT dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of conservativeRasterizationMode is VK_CONSERVATIVE_RASTERIZATION_MODE_OVERESTIMATE_EXT, then vkCmdSetExtraPrimitiveOverestimationSizeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiEXT-None-07633
  If the depthClipEnable feature is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_CLIP_ENABLE_EXT dynamic state, then vkCmdSetDepthClipEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiEXT-None-07634
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT dynamic state enabled then vkCmdSetSampleLocationsEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiEXT-None-08664
  If the VK_EXT_sample_locations extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetSampleLocationsEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiEXT-None-07635
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_ADVANCED_EXT dynamic state enabled then vkCmdSetColorBlendAdvancedEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiEXT-rasterizerDiscardEnable-09416
  If the VK_EXT_blend_operation_advanced extension is enabled, and a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then at least one of vkCmdSetColorBlendEquationEXT and vkCmdSetColorBlendAdvancedEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiEXT-None-07636
  If the VK_EXT_provoking_vertex extension is enabled, a shader object is bound to the

VK_SHADER_STAGE_VERTEX_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_PROVOKING_VERTEX_MODE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetProvokingVertexModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMultiEXT-None-07637
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT dynamic state enabled then vkCmdSetLineRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMultiEXT-None-08666
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetPolygonModeEXT in the current command buffer set polygonMode to VK_POLYGON_MODE_LINE, then vkCmdSetLineRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMultiEXT-None-08667
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object is bound to the VK_SHADER_STAGE_VERTEX_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetPrimitiveTopologyEXT in the current command buffer set primitiveTopology to any line topology, then vkCmdSetLineRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMultiEXT-None-08668
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object that outputs line primitives is bound to the VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT or VK_SHADER_STAGE_GEOMETRY_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetLineRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMultiEXT-None-07638
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT dynamic state enabled then vkCmdSetLineStippleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMultiEXT-None-08669
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set

rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetPolygonModeEXT in the current command buffer set polygonMode to VK_POLYGON_MODE_LINE, then vkCmdSetLineStippleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMultiEXT-None-08670
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object is bound to the VK_SHADER_STAGE_VERTEX_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetPrimitiveTopologyEXT in the current command buffer set primitiveTopology to any line topology, then vkCmdSetLineStippleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiEXT-None-08671
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object that outputs line primitives is bound to the VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT or VK_SHADER_STAGE_GEOMETRY_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetLineStippleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiEXT-None-07849
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_KHR dynamic state enabled then vkCmdSetLineStippleKHR **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiEXT-None-08672
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetLineStippleEnableEXT in the current command buffer set stippledLineEnable to VK_TRUE, then vkCmdSetLineStippleEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiEXT-None-07639
  If the depthClipControl feature is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_CLIP_NEGATIVE_ONE_TO_ONE_EXT dynamic state enabled, then vkCmdSetDepthClipNegativeOneToOneEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiEXT-None-07640
  If the VK_NV_clip_space_w_scaling extension is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_VIEWPORT_W_SCALING_ENABLE_NV dynamic state enabled, then vkCmdSetViewportWScalingEnableNV **must** have been called and not subsequently

invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMultiEXT-None-07641
  If the VK_NV_viewport_swizzle extension is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_VIEWPORT_SWIZZLE_NV dynamic state enabled, then vkCmdSetViewportSwizzleNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMultiEXT-None-07642
  If the VK_NV_fragment_coverage_to_color extension is enabled, a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_TO_COLOR_ENABLE_NV dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetCoverageToColorEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMultiEXT-None-07643
  If the VK_NV_fragment_coverage_to_color extension is enabled, a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_TO_COLOR_LOCATION_NV dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of coverageToColorEnable is VK_TRUE, then vkCmdSetCoverageToColorLocationNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMultiEXT-None-07644
  If the VK_NV_framebuffer_mixed_samples extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_MODULATION_MODE_NV dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetCoverageModulationModeNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMultiEXT-None-07645
  If the VK_NV_framebuffer_mixed_samples extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_MODULATION_TABLE_ENABLE_NV dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of coverageModulationMode is any value other than VK_COVERAGE_MODULATION_MODE_NONE_NV, then vkCmdSetCoverageModulationTableEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMultiEXT-None-07646
  If the VK_NV_framebuffer_mixed_samples extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_MODULATION_TABLE_NV dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of coverageModulationTableEnable is VK_TRUE, then vkCmdSetCoverageModulationTableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMultiEXT-None-07647
  If the `shadingRateImage` feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_SHADING_RATE_IMAGE_ENABLE_NV` dynamic state enabled, and the current value of `rasterizerDiscardEnable` is `VK_FALSE`, then vkCmdSetShadingRateImageEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiEXT-pipelineFragmentShadingRate-09238
  If the `pipelineFragmentShadingRate` feature is enabled, a shader object is bound to the `VK_SHADER_STAGE_FRAGMENT_BIT` stage or a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR` dynamic state enabled, and the current value of `rasterizerDiscardEnable` is `VK_FALSE`, then vkCmdSetFragmentShadingRateKHR **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiEXT-None-07648
  If the `representativeFragmentTest` feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_REPRESENTATIVE_FRAGMENT_TEST_ENABLE_NV` dynamic state enabled, and the current value of `rasterizerDiscardEnable` is `VK_FALSE`, then vkCmdSetRepresentativeFragmentTestEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiEXT-None-07649
  If the `coverageReductionMode` feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_COVERAGE_REDUCTION_MODE_NV` dynamic state enabled, and the current value of `rasterizerDiscardEnable` is `VK_FALSE`, then vkCmdSetCoverageReductionModeNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiEXT-pColorBlendEnables-07470
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT` state enabled and the last call to vkCmdSetColorBlendEnableEXT set `pColorBlendEnables` for any attachment to `VK_TRUE`, then for those attachments in the subpass the corresponding image view's format features **must** contain `VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT`
- VUID-vkCmdDrawMultiEXT-rasterizationSamples-07471
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT` state enabled, and the current subpass does not use any color and/or depth/stencil attachments, then the `rasterizationSamples` in the last call to vkCmdSetRasterizationSamplesEXT **must** follow the rules for a zero-attachment subpass
- VUID-vkCmdDrawMultiEXT-samples-07472
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_SAMPLE_MASK_EXT` state enabled and the `VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT` state disabled, then the `samples` parameter in the last call to vkCmdSetSampleMaskEXT **must** be greater or equal to the

VkPipelineMultisampleStateCreateInfo::rasterizationSamples parameter used to create the bound graphics pipeline

- VUID-vkCmdDrawMultiEXT-samples-07473
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_MASK_EXT state and VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT states enabled, then the samples parameter in the last call to vkCmdSetSampleMaskEXT **must** be greater or equal to the rasterizationSamples parameter in the last call to vkCmdSetRasterizationSamplesEXT

- VUID-vkCmdDrawMultiEXT-rasterizationSamples-07474
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, and neither the VK_AMD_mixed_attachment_samples nor the VK_NV_framebuffer_mixed_samples extensions are enabled, then the rasterizationSamples in the last call to vkCmdSetRasterizationSamplesEXT **must** be the same as the current subpass color and/or depth/stencil attachments

- VUID-vkCmdDrawMultiEXT-None-09211
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, or a shader object is bound to any graphics stage, and the current render pass instance includes a VkMultisampledRenderToSingleSampledInfoEXT structure with multisampledRenderToSingleSampledEnable equal to VK_TRUE, then the rasterizationSamples in the last call to vkCmdSetRasterizationSamplesEXT **must** be the same as the rasterizationSamples member of that structure

- VUID-vkCmdDrawMultiEXT-firstAttachment-07476
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT dynamic state enabled then vkCmdSetColorBlendEnableEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the firstAttachment and attachmentCount parameters of vkCmdSetColorBlendEnableEXT calls **must** specify an enable for all active color attachments in the current subpass

- VUID-vkCmdDrawMultiEXT-rasterizerDiscardEnable-09417
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetColorBlendEnableEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the firstAttachment and attachmentCount parameters of vkCmdSetColorBlendEnableEXT calls **must** specify an enable for all active color attachments in the current subpass

- VUID-vkCmdDrawMultiEXT-firstAttachment-07477
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_EQUATION_EXT dynamic state enabled then vkCmdSetColorBlendEquationEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the firstAttachment and attachmentCount parameters of vkCmdSetColorBlendEquationEXT calls **must** specify the blend equations for all active color attachments in the current subpass where blending is

enabled

- VUID-vkCmdDrawMultiEXT-rasterizerDiscardEnable-09418
  If a shader object is bound to the `VK_SHADER_STAGE_FRAGMENT_BIT` stage, and both the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE` and there are color attachments bound, then vkCmdSetColorBlendEquationEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the `firstAttachment` and `attachmentCount` parameters of `vkCmdSetColorBlendEquationEXT` calls **must** specify the blend equations for all active color attachments in the current subpass where blending is enabled

- VUID-vkCmdDrawMultiEXT-firstAttachment-07478
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_COLOR_WRITE_MASK_EXT` dynamic state enabled then vkCmdSetColorWriteMaskEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the `firstAttachment` and `attachmentCount` parameters of `vkCmdSetColorWriteMaskEXT` calls **must** specify the color write mask for all active color attachments in the current subpass

- VUID-vkCmdDrawMultiEXT-rasterizerDiscardEnable-09419
  If a shader object is bound to the `VK_SHADER_STAGE_FRAGMENT_BIT` stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE`, then vkCmdSetColorWriteMaskEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the `firstAttachment` and `attachmentCount` parameters of `vkCmdSetColorWriteMaskEXT` calls **must** specify the color write mask for all active color attachments in the current subpass

- VUID-vkCmdDrawMultiEXT-firstAttachment-07479
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_COLOR_BLEND_ADVANCED_EXT` dynamic state enabled then vkCmdSetColorBlendAdvancedEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the `firstAttachment` and `attachmentCount` parameters of `vkCmdSetColorBlendAdvancedEXT` calls **must** specify the advanced blend equations for all active color attachments in the current subpass where blending is enabled

- VUID-vkCmdDrawMultiEXT-advancedBlendMaxColorAttachments-07480
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_COLOR_BLEND_ADVANCED_EXT` and `VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT` dynamic states enabled and the last calls to vkCmdSetColorBlendEnableEXT and vkCmdSetColorBlendAdvancedEXT have enabled advanced blending, then the number of active color attachments in the current subpass **must** not exceed `advancedBlendMaxColorAttachments`

- VUID-vkCmdDrawMultiEXT-primitivesGeneratedQueryWithNonZeroStreams-07481
  If the `primitivesGeneratedQueryWithNonZeroStreams` feature is not enabled and the `VK_QUERY_TYPE_PRIMITIVES_GENERATED_EXT` query is active, and the bound graphics pipeline was created with `VK_DYNAMIC_STATE_RASTERIZATION_STREAM_EXT` state enabled, the last call to vkCmdSetRasterizationStreamEXT **must** have set the `rasterizationStream` to zero

- VUID-vkCmdDrawMultiEXT-sampleLocationsPerPixel-07482
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT` state enabled and the `VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT` state disabled, then the `sampleLocationsPerPixel` member of `pSampleLocationsInfo` in the last call to vkCmdSetSampleLocationsEXT **must** equal the `rasterizationSamples` member of the VkPipelineMultisampleStateCreateInfo structure the bound graphics pipeline has been created with
- VUID-vkCmdDrawMultiEXT-sampleLocationsPerPixel-07483
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT` state enabled and the `VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT` state enabled, then the `sampleLocationsPerPixel` member of `pSampleLocationsInfo` in the last call to vkCmdSetSampleLocationsEXT **must** equal the `rasterizationSamples` parameter of the last call to vkCmdSetRasterizationSamplesEXT
- VUID-vkCmdDrawMultiEXT-sampleLocationsEnable-07484
  If a shader object is bound to the `VK_SHADER_STAGE_FRAGMENT_BIT` stage, or the bound graphics pipeline was created with the `VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT` state enabled, and `sampleLocationsEnable` was `VK_TRUE` in the last call to vkCmdSetSampleLocationsEnableEXT, and the current subpass has a depth/stencil attachment, then that attachment **must** have been created with the `VK_IMAGE_CREATE_SAMPLE_LOCATIONS_COMPATIBLE_DEPTH_BIT_EXT` bit set
- VUID-vkCmdDrawMultiEXT-sampleLocationsEnable-07485
  If a shader object is bound to the `VK_SHADER_STAGE_FRAGMENT_BIT` stage, or the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT` state enabled and the `VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT` state enabled, and if `sampleLocationsEnable` was `VK_TRUE` in the last call to vkCmdSetSampleLocationsEnableEXT, then the `sampleLocationsInfo.sampleLocationGridSize.width` in the last call to vkCmdSetSampleLocationsEXT **must** evenly divide VkMultisamplePropertiesEXT `::sampleLocationGridSize.width` as returned by vkGetPhysicalDeviceMultisamplePropertiesEXT with a `samples` parameter equaling `rasterizationSamples`
- VUID-vkCmdDrawMultiEXT-sampleLocationsEnable-07486
  If a shader object is bound to the `VK_SHADER_STAGE_FRAGMENT_BIT` stage, or the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT` state enabled and the `VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT` state enabled, and if `sampleLocationsEnable` was `VK_TRUE` in the last call to vkCmdSetSampleLocationsEnableEXT, then the `sampleLocationsInfo.sampleLocationGridSize.height` in the last call to vkCmdSetSampleLocationsEXT **must** evenly divide VkMultisamplePropertiesEXT `::sampleLocationGridSize.height` as returned by vkGetPhysicalDeviceMultisamplePropertiesEXT with a `samples` parameter equaling `rasterizationSamples`
- VUID-vkCmdDrawMultiEXT-sampleLocationsEnable-07487
  If a shader object is bound to the `VK_SHADER_STAGE_FRAGMENT_BIT` stage, or the bound

graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, and if sampleLocationsEnable was VK_TRUE in the last call to vkCmdSetSampleLocationsEnableEXT, the fragment shader code **must** not statically use the extended instruction InterpolateAtSample

- VUID-vkCmdDrawMultiEXT-sampleLocationsEnable-07936
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state disabled and the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, the sampleLocationsEnable member of a VkPipelineSampleLocationsStateCreateInfoEXT::sampleLocationsEnable in the bound graphics pipeline is VK_TRUE or VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, then, sampleLocationsInfo.sampleLocationGridSize.width **must** evenly divide VkMultisamplePropertiesEXT::sampleLocationGridSize.width as returned by vkGetPhysicalDeviceMultisamplePropertiesEXT with a samples parameter equaling the value of rasterizationSamples in the last call to vkCmdSetRasterizationSamplesEXT

- VUID-vkCmdDrawMultiEXT-sampleLocationsEnable-07937
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state disabled and the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, the sampleLocationsEnable member of a VkPipelineSampleLocationsStateCreateInfoEXT::sampleLocationsEnable in the bound graphics pipeline is VK_TRUE or VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, then, sampleLocationsInfo.sampleLocationGridSize.height **must** evenly divide VkMultisamplePropertiesEXT::sampleLocationGridSize.height as returned by vkGetPhysicalDeviceMultisamplePropertiesEXT with a samples parameter equaling the value of rasterizationSamples in the last call to vkCmdSetRasterizationSamplesEXT

- VUID-vkCmdDrawMultiEXT-sampleLocationsEnable-07938
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state disabled and the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, the sampleLocationsEnable member of a VkPipelineSampleLocationsStateCreateInfoEXT::sampleLocationsEnable in the bound graphics pipeline is VK_TRUE or VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, then, sampleLocationsInfo.sampleLocationsPerPixel **must** equal rasterizationSamples in the last call to vkCmdSetRasterizationSamplesEXT

- VUID-vkCmdDrawMultiEXT-coverageModulationTableEnable-07488
  If a shader object is bound to any graphics stage or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COVERAGE_MODULATION_TABLE_ENABLE_NV state enabled, and the last call to vkCmdSetCoverageModulationTableEnableNV set coverageModulationTableEnable to VK_TRUE, then the coverageModulationTableCount parameter in the last call to vkCmdSetCoverageModulationTableNV **must** equal the current rasterizationSamples divided by the number of color samples in the current subpass

- VUID-vkCmdDrawMultiEXT-rasterizationSamples-07489
  If the VK_NV_framebuffer_mixed_samples extension is enabled, and if current subpass has a depth/stencil attachment and depth test, stencil test, or depth bounds test are enabled in the currently bound pipeline state, then the current rasterizationSamples **must** be the same as the sample count of the depth/stencil attachment

- VUID-vkCmdDrawMultiEXT-coverageToColorEnable-07490
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COVERAGE_TO_COLOR_ENABLE_NV state enabled and the last call to vkCmdSetCoverageToColorEnableNV set the coverageToColorEnable to VK_TRUE, then the current subpass **must** have a color attachment at the location selected by the last call to vkCmdSetCoverageToColorLocationNV coverageToColorLocation, with a VkFormat of VK_FORMAT_R8_UINT, VK_FORMAT_R8_SINT, VK_FORMAT_R16_UINT, VK_FORMAT_R16_SINT, VK_FORMAT_R32_UINT, or VK_FORMAT_R32_SINT
- VUID-vkCmdDrawMultiEXT-rasterizerDiscardEnable-09420
  If the VK_NV_fragment_coverage_to_color extension is enabled, and a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the last call to vkCmdSetCoverageToColorEnableNV set the coverageToColorEnable to VK_TRUE, then the current subpass **must** have a color attachment at the location selected by the last call to vkCmdSetCoverageToColorLocationNV coverageToColorLocation, with a VkFormat of VK_FORMAT_R8_UINT, VK_FORMAT_R8_SINT, VK_FORMAT_R16_UINT, VK_FORMAT_R16_SINT, VK_FORMAT_R32_UINT, or VK_FORMAT_R32_SINT
- VUID-vkCmdDrawMultiEXT-coverageReductionMode-07491
  If this VK_NV_coverage_reduction_mode extension is enabled, the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COVERAGE_TO_COLOR_ENABLE_NV and VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT states enabled, the current coverage reduction mode coverageReductionMode, then the current rasterizationSamples, and the sample counts for the color and depth/stencil attachments (if the subpass has them) **must** be a valid combination returned by vkGetPhysicalDeviceSupportedFramebufferMixedSamplesCombinationsNV
- VUID-vkCmdDrawMultiEXT-viewportCount-07492
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, but not the VK_DYNAMIC_STATE_VIEWPORT_SWIZZLE_NV dynamic state enabled, then the bound graphics pipeline **must** have been created with VkPipelineViewportSwizzleStateCreateInfoNV ::viewportCount greater or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawMultiEXT-viewportCount-07493
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT and VK_DYNAMIC_STATE_VIEWPORT_SWIZZLE_NV dynamic states enabled then the viewportCount parameter in the last call to vkCmdSetViewportSwizzleNV **must** be greater than or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawMultiEXT-viewportCount-09421
  If the VK_NV_viewport_swizzle extension is enabled, and a shader object is bound to any graphics stage, then the viewportCount parameter in the last call to vkCmdSetViewportSwizzleNV **must** be greater than or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawMultiEXT-rasterizationSamples-07494

If the VK_NV_framebuffer_mixed_samples extension is enabled, and if the current subpass has any color attachments and rasterizationSamples of the last call to vkCmdSetRasterizationSamplesEXT is greater than the number of color samples, then the pipeline sampleShadingEnable **must** be VK_FALSE

- VUID-vkCmdDrawMultiEXT-stippledLineEnable-07495
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT or VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT dynamic states enabled, and if the current stippledLineEnable state is VK_TRUE and the current lineRasterizationMode state is VK_LINE_RASTERIZATION_MODE_RECTANGULAR_KHR, then the stippledRectangularLines feature **must** be enabled

- VUID-vkCmdDrawMultiEXT-stippledLineEnable-07496
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT or VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT dynamic states enabled, and if the current stippledLineEnable state is VK_TRUE and the current lineRasterizationMode state is VK_LINE_RASTERIZATION_MODE_BRESENHAM_KHR, then the stippledBresenhamLines feature **must** be enabled

- VUID-vkCmdDrawMultiEXT-stippledLineEnable-07497
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT or VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT dynamic states enabled, and if the current stippledLineEnable state is VK_TRUE and the current lineRasterizationMode state is VK_LINE_RASTERIZATION_MODE_RECTANGULAR_SMOOTH_KHR, then the stippledSmoothLines feature **must** be enabled

- VUID-vkCmdDrawMultiEXT-stippledLineEnable-07498
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT or VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT dynamic states enabled, and if the current stippledLineEnable state is VK_TRUE and the current lineRasterizationMode state is VK_LINE_RASTERIZATION_MODE_DEFAULT_KHR, then the stippledRectangularLines feature **must** be enabled and VkPhysicalDeviceLimits::strictLines **must** be VK_TRUE

- VUID-vkCmdDrawMultiEXT-conservativePointAndLineRasterization-07499
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_CONSERVATIVE_RASTERIZATION_MODE_EXT dynamic state enabled, conservativePointAndLineRasterization is not supported, and the effective primitive topology output by the last pre-rasterization shader stage is a line or point, then the conservativeRasterizationMode set by the last call to vkCmdSetConservativeRasterizationModeEXT **must** be VK_CONSERVATIVE_RASTERIZATION_MODE_DISABLED_EXT

- VUID-vkCmdDrawMultiEXT-stage-07073
  If the currently bound pipeline was created with the VkPipelineShaderStageCreateInfo::stage member of an element of VkGraphicsPipelineCreateInfo::pStages set to VK_SHADER_STAGE_VERTEX_BIT, VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT, VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT or VK_SHADER_STAGE_GEOMETRY_BIT, then Mesh

Shader Queries **must** not be active

- VUID-vkCmdDrawMultiEXT-None-08877
  If a shader object is bound to the `VK_SHADER_STAGE_FRAGMENT_BIT` stage or a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_ATTACHMENT_FEEDBACK_LOOP_ENABLE_EXT` dynamic state enabled, and the current value of `rasterizerDiscardEnable` is `VK_FALSE`, then vkCmdSetAttachmentFeedbackLoopEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiEXT-None-07850
  If dynamic state was inherited from VkCommandBufferInheritanceViewportScissorInfoNV, it **must** be set in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiEXT-None-08684
  If there is no bound graphics pipeline, `vkCmdBindShadersEXT` **must** have been called in the current command buffer with `pStages` with an element of `VK_SHADER_STAGE_VERTEX_BIT`
- VUID-vkCmdDrawMultiEXT-None-08685
  If there is no bound graphics pipeline, and the `tessellationShader` feature is enabled, `vkCmdBindShadersEXT` **must** have been called in the current command buffer with `pStages` with an element of `VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT`
- VUID-vkCmdDrawMultiEXT-None-08686
  If there is no bound graphics pipeline, and the `tessellationShader` feature is enabled, `vkCmdBindShadersEXT` **must** have been called in the current command buffer with `pStages` with an element of `VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT`
- VUID-vkCmdDrawMultiEXT-None-08687
  If there is no bound graphics pipeline, and the `geometryShader` feature is enabled, `vkCmdBindShadersEXT` **must** have been called in the current command buffer with `pStages` with an element of `VK_SHADER_STAGE_GEOMETRY_BIT`
- VUID-vkCmdDrawMultiEXT-None-08688
  If there is no bound graphics pipeline, `vkCmdBindShadersEXT` **must** have been called in the current command buffer with `pStages` with an element of `VK_SHADER_STAGE_FRAGMENT_BIT`
- VUID-vkCmdDrawMultiEXT-None-08689
  If there is no bound graphics pipeline, and the `taskShader` feature is enabled, `vkCmdBindShadersEXT` **must** have been called in the current command buffer with `pStages` with an element of `VK_SHADER_STAGE_TASK_BIT_EXT`
- VUID-vkCmdDrawMultiEXT-None-08690
  If there is no bound graphics pipeline, and the `meshShader` feature is enabled, `vkCmdBindShadersEXT` **must** have been called in the current command buffer with `pStages` with an element of `VK_SHADER_STAGE_MESH_BIT_EXT`
- VUID-vkCmdDrawMultiEXT-None-08693
  If there is no bound graphics pipeline, and at least one of the `taskShader` and `meshShader` features is enabled, one of the `VK_SHADER_STAGE_VERTEX_BIT` or `VK_SHADER_STAGE_MESH_BIT_EXT` stages **must** have a valid `VkShaderEXT` bound, and the other **must** have no `VkShaderEXT` bound

- VUID-vkCmdDrawMultiEXT-None-08694
  If there is no bound graphics pipeline, and both the taskShader and meshShader features are enabled, and a valid VkShaderEXT is bound the to the VK_SHADER_STAGE_MESH_BIT_EXT stage, and that VkShaderEXT was created without the VK_SHADER_CREATE_NO_TASK_SHADER_BIT_EXT flag, a valid VkShaderEXT **must** be bound to the VK_SHADER_STAGE_TASK_BIT_EXT stage
- VUID-vkCmdDrawMultiEXT-None-08695
  If there is no bound graphics pipeline, and both the taskShader and meshShader features are enabled, and a valid VkShaderEXT is bound the to the VK_SHADER_STAGE_MESH_BIT_EXT stage, and that VkShaderEXT was created with the VK_SHADER_CREATE_NO_TASK_SHADER_BIT_EXT flag, there **must** be no VkShaderEXT bound to the VK_SHADER_STAGE_TASK_BIT_EXT stage
- VUID-vkCmdDrawMultiEXT-None-08696
  If there is no bound graphics pipeline, and a valid VkShaderEXT is bound to the VK_SHADER_STAGE_VERTEX_BIT stage, there **must** be no VkShaderEXT bound to either the VK_SHADER_STAGE_TASK_BIT_EXT stage or the VK_SHADER_STAGE_MESH_BIT_EXT stage
- VUID-vkCmdDrawMultiEXT-None-08698
  If any graphics shader is bound which was created with the VK_SHADER_CREATE_LINK_STAGE_BIT_EXT flag, then all shaders created with the VK_SHADER_CREATE_LINK_STAGE_BIT_EXT flag in the same vkCreateShadersEXT call **must** also be bound
- VUID-vkCmdDrawMultiEXT-None-08699
  If any graphics shader is bound which was created with the VK_SHADER_CREATE_LINK_STAGE_BIT_EXT flag, any stages in between stages whose shaders which did not create a shader with the VK_SHADER_CREATE_LINK_STAGE_BIT_EXT flag as part of the same vkCreateShadersEXT call **must** not have any VkShaderEXT bound
- VUID-vkCmdDrawMultiEXT-None-08878
  All bound graphics shader objects **must** have been created with identical or identically defined push constant ranges
- VUID-vkCmdDrawMultiEXT-None-08879
  All bound graphics shader objects **must** have been created with identical or identically defined arrays of descriptor set layouts
- VUID-vkCmdDrawMultiEXT-colorAttachmentCount-09372
  If the current render pass instance was begun with vkCmdBeginRenderingKHR and a VkRenderingInfo::colorAttachmentCount equal to 1, a color attachment with a resolve mode of VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID, and a fragment shader is bound, it **must** not declare the DepthReplacing or StencilRefReplacingEXT execution modes
- VUID-vkCmdDrawMultiEXT-pDynamicStates-08715
  If the bound graphics pipeline state includes a fragment shader stage, was created with VK_DYNAMIC_STATE_DEPTH_WRITE_ENABLE set in VkPipelineDynamicStateCreateInfo::pDynamicStates, and the fragment shader declares the EarlyFragmentTests execution mode and uses OpDepthAttachmentReadEXT, the depthWriteEnable parameter in the last call to vkCmdSetDepthWriteEnableEXT **must** be VK_FALSE
- VUID-vkCmdDrawMultiEXT-pDynamicStates-08716
  If the bound graphics pipeline state includes a fragment shader stage, was created with

VK_DYNAMIC_STATE_STENCIL_WRITE_MASK set in VkPipelineDynamicStateCreateInfo::pDynamicStates, and the fragment shader declares the EarlyFragmentTests execution mode and uses OpStencilAttachmentReadEXT, the writeMask parameter in the last call to vkCmdSetStencilWriteMask **must** be 0

- VUID-vkCmdDrawMultiEXT-None-09116
  If a shader object is bound to any graphics stage or the currently bound graphics pipeline was created with VK_DYNAMIC_STATE_COLOR_WRITE_MASK_EXT, and the format of any color attachment is VK_FORMAT_E5B9G9R9_UFLOAT_PACK32, the corresponding element of the pColorWriteMasks parameter of vkCmdSetColorWriteMaskEXT **must** either include all of VK_COLOR_COMPONENT_R_BIT, VK_COLOR_COMPONENT_G_BIT, and VK_COLOR_COMPONENT_B_BIT, or none of them

- VUID-vkCmdDrawMultiEXT-maxFragmentDualSrcAttachments-09239
  If blending is enabled for any attachment where either the source or destination blend factors for that attachment use the secondary color input, the maximum value of Location for any output attachment statically used in the Fragment Execution Model executed by this command **must** be less than maxFragmentDualSrcAttachments

- VUID-vkCmdDrawMultiEXT-None-09548
  If the current render pass was begun with vkCmdBeginRenderingKHR, and there is no shader object bound to any graphics stage, the value of each element of VkRenderingAttachmentLocationInfoKHR::pColorAttachmentLocations set by vkCmdSetRenderingAttachmentLocationsKHR **must** match the value set for the corresponding element in the currently bound pipeline

- VUID-vkCmdDrawMultiEXT-None-09549
  If the current render pass was begun with vkCmdBeginRenderingKHR, and there is no shader object bound to any graphics stage, input attachment index mappings in the currently bound pipeline **must** match those set for the current render pass instance via VkRenderingInputAttachmentIndexInfoKHR

- VUID-vkCmdDrawMultiEXT-None-09642
  If the current render pass was begun with vkCmdBeginRenderingKHR with the VK_RENDERING_ENABLE_LEGACY_DITHERING_BIT_EXT flag, the bound graphics pipeline **must** have been created with VK_PIPELINE_CREATE_2_ENABLE_LEGACY_DITHERING_BIT_EXT

- VUID-vkCmdDrawMultiEXT-None-09643
  If the bound graphics pipeline was created with VK_PIPELINE_CREATE_2_ENABLE_LEGACY_DITHERING_BIT_EXT, the current render pass **must** have begun with vkCmdBeginRenderingKHR with the VK_RENDERING_ENABLE_LEGACY_DITHERING_BIT_EXT flag

- VUID-vkCmdDrawMultiEXT-commandBuffer-02712
  If commandBuffer is a protected command buffer and protectedNoFault is not supported, any resource written to by the VkPipeline object bound to the pipeline bind point used by this command **must** not be an unprotected resource

- VUID-vkCmdDrawMultiEXT-commandBuffer-02713
  If commandBuffer is a protected command buffer and protectedNoFault is not supported, pipeline stages other than the framebuffer-space and compute stages in the VkPipeline object bound to the pipeline bind point used by this command **must** not write to any

resource

- VUID-vkCmdDrawMultiEXT-commandBuffer-04617
  If any of the shader stages of the `VkPipeline` bound to the pipeline bind point used by this command uses the `RayQueryKHR` capability, then `commandBuffer` **must** not be a protected command buffer

- VUID-vkCmdDrawMultiEXT-None-04007
  All vertex input bindings accessed via vertex input variables declared in the vertex shader entry point's interface **must** have either valid or VK_NULL_HANDLE buffers bound

- VUID-vkCmdDrawMultiEXT-None-04008
  If the `nullDescriptor` feature is not enabled, all vertex input bindings accessed via vertex input variables declared in the vertex shader entry point's interface **must** not be VK_NULL_HANDLE

- VUID-vkCmdDrawMultiEXT-None-02721
  If `robustBufferAccess` is not enabled, and that pipeline was created without enabling `VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_EXT` for `vertexInputs`, then for a given vertex buffer binding, any attribute data fetched **must** be entirely contained within the corresponding vertex buffer binding, as described in Vertex Input Description

- VUID-vkCmdDrawMultiEXT-None-07842
  If there is a shader object bound to the `VK_SHADER_STAGE_VERTEX_BIT` stage or the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_PRIMITIVE_TOPOLOGY` dynamic state enabled then vkCmdSetPrimitiveTopologyEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMultiEXT-dynamicPrimitiveTopologyUnrestricted-07500
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_PRIMITIVE_TOPOLOGY` dynamic state enabled and the `dynamicPrimitiveTopologyUnrestricted` is `VK_FALSE`, then the `primitiveTopology` parameter of `vkCmdSetPrimitiveTopologyEXT` **must** be of the same topology class as the pipeline VkPipelineInputAssemblyStateCreateInfo::`topology` state

- VUID-vkCmdDrawMultiEXT-pStrides-04913
  If the bound graphics pipeline was created with the `VK_DYNAMIC_STATE_VERTEX_INPUT_BINDING_STRIDE_EXT` dynamic state enabled, but without the `VK_DYNAMIC_STATE_VERTEX_INPUT_EXT` dynamic state enabled, then vkCmdBindVertexBuffers2EXT **must** have been called and not subsequently invalidated in the current command buffer prior to this draw command, and the `pStrides` parameter of vkCmdBindVertexBuffers2EXT **must** not be `NULL`

- VUID-vkCmdDrawMultiEXT-None-04914
  If there is a shader object bound to the `VK_SHADER_STAGE_VERTEX_BIT` stage or the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VERTEX_INPUT_EXT` dynamic state enabled then vkCmdSetVertexInputEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this draw command

- VUID-vkCmdDrawMultiEXT-Input-07939
  If there is a shader object bound to the `VK_SHADER_STAGE_VERTEX_BIT` stage or the bound

graphics pipeline state was created with the VK_DYNAMIC_STATE_VERTEX_INPUT_EXT dynamic state enabled then all variables with the Input storage class decorated with Location in the Vertex Execution Model OpEntryPoint **must** contain a location in VkVertexInputAttributeDescription2EXT::location

- VUID-vkCmdDrawMultiEXT-Input-08734
  If there is a shader object bound to the VK_SHADER_STAGE_VERTEX_BIT stage or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VERTEX_INPUT_EXT dynamic state enabled and either legacyVertexAttributes is not enabled or the SPIR-V Type associated with a given Input variable of the corresponding Location in the Vertex Execution Model OpEntryPoint is 64-bit, then the numeric type associated with all Input variables of the corresponding Location in the Vertex Execution Model OpEntryPoint **must** be the same as VkVertexInputAttributeDescription2EXT::format
- VUID-vkCmdDrawMultiEXT-format-08936
  If there is a shader object bound to the VK_SHADER_STAGE_VERTEX_BIT stage or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VERTEX_INPUT_EXT dynamic state enabled and VkVertexInputAttributeDescription2EXT::format has a 64-bit component, then the scalar width associated with all Input variables of the corresponding Location in the Vertex Execution Model OpEntryPoint **must** be 64-bit
- VUID-vkCmdDrawMultiEXT-format-08937
  If there is a shader object bound to the VK_SHADER_STAGE_VERTEX_BIT stage or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VERTEX_INPUT_EXT dynamic state enabled and the scalar width associated with a Location decorated Input variable in the Vertex Execution Model OpEntryPoint is 64-bit, then the corresponding VkVertexInputAttributeDescription2EXT::format **must** have a 64-bit component
- VUID-vkCmdDrawMultiEXT-None-09203
  If there is a shader object bound to the VK_SHADER_STAGE_VERTEX_BIT stage or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VERTEX_INPUT_EXT dynamic state enabled and VkVertexInputAttributeDescription2EXT::format has a 64-bit component, then all Input variables at the corresponding Location in the Vertex Execution Model OpEntryPoint **must** not use components that are not present in the format
- VUID-vkCmdDrawMultiEXT-None-04875
  If there is a shader object bound to the VK_SHADER_STAGE_VERTEX_BIT stage and the most recent call to vkCmdSetPrimitiveTopologyEXT in the current command buffer set primitiveTopology to VK_PRIMITIVE_TOPOLOGY_PATCH_LIST, or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_PATCH_CONTROL_POINTS_EXT dynamic state enabled then vkCmdSetPatchControlPointsEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiEXT-None-04879
  If there is a shader object bound to the VK_SHADER_STAGE_VERTEX_BIT stage or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_PRIMITIVE_RESTART_ENABLE dynamic state enabled then vkCmdSetPrimitiveRestartEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiEXT-None-09637
  If the primitiveTopologyListRestart feature is not enabled, the topology is

VK_PRIMITIVE_TOPOLOGY_POINT_LIST, VK_PRIMITIVE_TOPOLOGY_LINE_LIST, VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST, VK_PRIMITIVE_TOPOLOGY_LINE_LIST_WITH_ADJACENCY, or VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST_WITH_ADJACENCY, there is a shader object bound to the VK_SHADER_STAGE_VERTEX_BIT stage or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_PRIMITIVE_RESTART_ENABLE dynamic state enabled then vkCmdSetPrimitiveRestartEnableEXT **must** be VK_FALSE

- VUID-vkCmdDrawMultiEXT-stage-06481
  The bound graphics pipeline **must** not have been created with the VkPipelineShaderStageCreateInfo::stage member of an element of VkGraphicsPipelineCreateInfo::pStages set to VK_SHADER_STAGE_TASK_BIT_EXT or VK_SHADER_STAGE_MESH_BIT_EXT
- VUID-vkCmdDrawMultiEXT-None-08885
  There **must** be no shader object bound to either of the VK_SHADER_STAGE_TASK_BIT_EXT or VK_SHADER_STAGE_MESH_BIT_EXT stages
- VUID-vkCmdDrawMultiEXT-pNext-09461
  If the bound graphics pipeline state was created with VkPipelineVertexInputDivisorStateCreateInfoKHR in the pNext chain of VkGraphicsPipelineCreateInfo::pVertexInputState, any member of VkPipelineVertexInputDivisorStateCreateInfoKHR::pVertexBindingDivisors has a value other than 1 in divisor, and VkPhysicalDeviceVertexAttributeDivisorPropertiesKHR::supportsNonZeroFirstInstance is VK_FALSE, then firstInstance **must** be 0
- VUID-vkCmdDrawMultiEXT-None-09462
  If shader objects are used for drawing or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VERTEX_INPUT_EXT dynamic state enabled, any member of the pVertexBindingDescriptions parameter to the vkCmdSetVertexInputEXT call that sets this dynamic state has a value other than 1 in divisor, and VkPhysicalDeviceVertexAttributeDivisorPropertiesKHR::supportsNonZeroFirstInstance is VK_FALSE, then firstInstance **must** be 0
- VUID-vkCmdDrawMultiEXT-None-04933
  The multiDraw feature **must** be enabled
- VUID-vkCmdDrawMultiEXT-drawCount-04934
  drawCount **must** be less than VkPhysicalDeviceMultiDrawPropertiesEXT::maxMultiDrawCount
- VUID-vkCmdDrawMultiEXT-drawCount-04935
  If drawCount is greater than zero, pVertexInfo **must** be a valid pointer to memory containing one or more valid instances of VkMultiDrawInfoEXT structures
- VUID-vkCmdDrawMultiEXT-drawCount-09628
  If drawCount is greater than 1, stride **must** be a multiple of 4 and **must** be greater than or equal to sizeof(VkMultiDrawInfoEXT)

## Valid Usage (Implicit)

- VUID-vkCmdDrawMultiEXT-commandBuffer-parameter
  commandBuffer **must** be a valid VkCommandBuffer handle

- VUID-vkCmdDrawMultiEXT-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdDrawMultiEXT-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- VUID-vkCmdDrawMultiEXT-renderpass
  This command **must** only be called inside of a render pass instance
- VUID-vkCmdDrawMultiEXT-videocoding
  This command **must** only be called outside of a video coding scope

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Inside | Outside | Graphics | Action |

To record an ordered sequence of indexed draws which have no state changes between them, call:

```
// Provided by VK_EXT_multi_draw
void vkCmdDrawMultiIndexedEXT(
    VkCommandBuffer                             commandBuffer,
    uint32_t                                    drawCount,
    const VkMultiDrawIndexedInfoEXT*            pIndexInfo,
    uint32_t                                    instanceCount,
    uint32_t                                    firstInstance,
    uint32_t                                    stride,
    const int32_t*                              pVertexOffset);
```

- `commandBuffer` is the command buffer into which the command is recorded.
- `drawCount` is the number of draws to execute, and **can** be zero.
- `pIndexInfo` is a pointer to an array of VkMultiDrawIndexedInfoEXT with index information to be drawn.
- `instanceCount` is the number of instances per draw.
- `firstInstance` is the instance ID of the first instance in each draw.

- stride is the byte stride between consecutive elements of pIndexInfo.
- pVertexOffset is NULL or a pointer to the value added to the vertex index before indexing into the vertex buffer. When specified, VkMultiDrawIndexedInfoEXT::offset is ignored.

The number of draws recorded is drawCount, with each draw reading, sequentially, a firstIndex and an indexCount from pIndexInfo. For each recorded draw, primitives are assembled as for vkCmdDrawIndexed, and drawn instanceCount times with instanceIndex starting with firstInstance and sequentially for each instance. If pVertexOffset is NULL, a vertexOffset is also read from pIndexInfo, otherwise the value from dereferencing pVertexOffset is used.

## Valid Usage

- VUID-vkCmdDrawMultiIndexedEXT-magFilter-04553
  If a VkSampler created with magFilter or minFilter equal to VK_FILTER_LINEAR, reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE, and compareEnable equal to VK_FALSE is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT
- VUID-vkCmdDrawMultiIndexedEXT-magFilter-09598
  If a VkSampler created with magFilter or minFilter equal to VK_FILTER_LINEAR and reductionMode equal to either VK_SAMPLER_REDUCTION_MODE_MIN or VK_SAMPLER_REDUCTION_MODE_MAX is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT
- VUID-vkCmdDrawMultiIndexedEXT-mipmapMode-04770
  If a VkSampler created with mipmapMode equal to VK_SAMPLER_MIPMAP_MODE_LINEAR, reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE, and compareEnable equal to VK_FALSE is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT
- VUID-vkCmdDrawMultiIndexedEXT-mipmapMode-09599
  If a VkSampler created with mipmapMode equal to VK_SAMPLER_MIPMAP_MODE_LINEAR and reductionMode equal to either VK_SAMPLER_REDUCTION_MODE_MIN or VK_SAMPLER_REDUCTION_MODE_MAX is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT
- VUID-vkCmdDrawMultiIndexedEXT-unnormalizedCoordinates-09635
  If a VkSampler created with unnormalizedCoordinates equal to VK_TRUE is used to sample a VkImageView as a result of this command, then the image view's levelCount and layerCount **must** be 1
- VUID-vkCmdDrawMultiIndexedEXT-unnormalizedCoordinates-09636
  If a VkSampler created with unnormalizedCoordinates equal to VK_TRUE is used to sample a VkImageView as a result of this command, then the image view's viewType **must** be VK_IMAGE_VIEW_TYPE_1D or VK_IMAGE_VIEW_TYPE_2D
- VUID-vkCmdDrawMultiIndexedEXT-None-06479

If a VkImageView is sampled with depth comparison, the image view's format features **must** contain VK_FORMAT_FEATURE_2_SAMPLED_IMAGE_DEPTH_COMPARISON_BIT

- VUID-vkCmdDrawMultiIndexedEXT-None-02691
  If a VkImageView is accessed using atomic operations as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_STORAGE_IMAGE_ATOMIC_BIT
- VUID-vkCmdDrawMultiIndexedEXT-None-07888
  If a VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER descriptor is accessed using atomic operations as a result of this command, then the storage texel buffer's format features **must** contain VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_ATOMIC_BIT
- VUID-vkCmdDrawMultiIndexedEXT-None-02692
  If a VkImageView is sampled with VK_FILTER_CUBIC_EXT as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_CUBIC_BIT_EXT
- VUID-vkCmdDrawMultiIndexedEXT-None-02693
  If the VK_EXT_filter_cubic extension is not enabled and any VkImageView is sampled with VK_FILTER_CUBIC_EXT as a result of this command, it **must** not have a VkImageViewType of VK_IMAGE_VIEW_TYPE_3D, VK_IMAGE_VIEW_TYPE_CUBE, or VK_IMAGE_VIEW_TYPE_CUBE_ARRAY
- VUID-vkCmdDrawMultiIndexedEXT-filterCubic-02694
  Any VkImageView being sampled with VK_FILTER_CUBIC_EXT as a result of this command **must** have a VkImageViewType and format that supports cubic filtering, as specified by VkFilterCubicImageViewImageFormatPropertiesEXT::filterCubic returned by vkGetPhysicalDeviceImageFormatProperties2
- VUID-vkCmdDrawMultiIndexedEXT-filterCubicMinmax-02695
  Any VkImageView being sampled with VK_FILTER_CUBIC_EXT with a reduction mode of either VK_SAMPLER_REDUCTION_MODE_MIN or VK_SAMPLER_REDUCTION_MODE_MAX as a result of this command **must** have a VkImageViewType and format that supports cubic filtering together with minmax filtering, as specified by VkFilterCubicImageViewImageFormatPropertiesEXT::filterCubicMinmax returned by vkGetPhysicalDeviceImageFormatProperties2
- VUID-vkCmdDrawMultiIndexedEXT-cubicRangeClamp-09212
  If the cubicRangeClamp feature is not enabled, then any VkImageView being sampled with VK_FILTER_CUBIC_EXT as a result of this command **must** not have a VkSamplerReductionModeCreateInfo::reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE_RANGECLAMP_QCOM
- VUID-vkCmdDrawMultiIndexedEXT-reductionMode-09213
  Any VkImageView being sampled with a VkSamplerReductionModeCreateInfo::reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE_RANGECLAMP_QCOM as a result of this command **must** sample with VK_FILTER_CUBIC_EXT
- VUID-vkCmdDrawMultiIndexedEXT-selectableCubicWeights-09214
  If the selectableCubicWeights feature is not enabled, then any VkImageView being sampled with VK_FILTER_CUBIC_EXT as a result of this command **must** have VkSamplerCubicWeightsCreateInfoQCOM::cubicWeights equal to VK_CUBIC_FILTER_WEIGHTS_CATMULL_ROM_QCOM

- VUID-vkCmdDrawMultiIndexedEXT-flags-02696
  Any VkImage created with a VkImageCreateInfo::flags containing VK_IMAGE_CREATE_CORNER_SAMPLED_BIT_NV sampled as a result of this command **must** only be sampled using a VkSamplerAddressMode of VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_EDGE
- VUID-vkCmdDrawMultiIndexedEXT-OpTypeImage-07027
  For any VkImageView being written as a storage image where the image format field of the OpTypeImage is Unknown, the view's format features **must** contain VK_FORMAT_FEATURE_2_STORAGE_WRITE_WITHOUT_FORMAT_BIT
- VUID-vkCmdDrawMultiIndexedEXT-OpTypeImage-07028
  For any VkImageView being read as a storage image where the image format field of the OpTypeImage is Unknown, the view's format features **must** contain VK_FORMAT_FEATURE_2_STORAGE_READ_WITHOUT_FORMAT_BIT
- VUID-vkCmdDrawMultiIndexedEXT-OpTypeImage-07029
  For any VkBufferView being written as a storage texel buffer where the image format field of the OpTypeImage is Unknown, the view's buffer features **must** contain VK_FORMAT_FEATURE_2_STORAGE_WRITE_WITHOUT_FORMAT_BIT
- VUID-vkCmdDrawMultiIndexedEXT-OpTypeImage-07030
  Any VkBufferView being read as a storage texel buffer where the image format field of the OpTypeImage is Unknown then the view's buffer features **must** contain VK_FORMAT_FEATURE_2_STORAGE_READ_WITHOUT_FORMAT_BIT
- VUID-vkCmdDrawMultiIndexedEXT-None-08600
  For each set *n* that is statically used by a bound shader, a descriptor set **must** have been bound to *n* at the same pipeline bind point, with a VkPipelineLayout that is compatible for set *n*, with the VkPipelineLayout used to create the current VkPipeline or the VkDescriptorSetLayout array used to create the current VkShaderEXT , as described in Pipeline Layout Compatibility
- VUID-vkCmdDrawMultiIndexedEXT-None-08601
  For each push constant that is statically used by a bound shader, a push constant value **must** have been set for the same pipeline bind point, with a VkPipelineLayout that is compatible for push constants, with the VkPipelineLayout used to create the current VkPipeline or the VkDescriptorSetLayout array used to create the current VkShaderEXT , as described in Pipeline Layout Compatibility
- VUID-vkCmdDrawMultiIndexedEXT-None-10068
  For each array of resources that is used by a bound shader, the indices used to access members of the array **must** be less than the descriptor count for the identified binding in the descriptor sets used by this command
- VUID-vkCmdDrawMultiIndexedEXT-maintenance4-08602
  If the maintenance4 feature is not enabled, then for each push constant that is statically used by a bound shader, a push constant value **must** have been set for the same pipeline bind point, with a VkPipelineLayout that is compatible for push constants, with the VkPipelineLayout used to create the current VkPipeline or the VkDescriptorSetLayout and VkPushConstantRange arrays used to create the current VkShaderEXT , as described in Pipeline Layout Compatibility
- VUID-vkCmdDrawMultiIndexedEXT-None-08114

Descriptors in each bound descriptor set, specified via vkCmdBindDescriptorSets, **must** be valid as described by descriptor validity if they are statically used by the VkPipeline bound to the pipeline bind point used by this command and the bound VkPipeline was not created with `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`

- VUID-vkCmdDrawMultiIndexedEXT-None-08115
  If the descriptors used by the VkPipeline bound to the pipeline bind point were specified via vkCmdBindDescriptorSets, the bound VkPipeline **must** have been created without `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`
- VUID-vkCmdDrawMultiIndexedEXT-None-08116
  Descriptors in bound descriptor buffers, specified via vkCmdSetDescriptorBufferOffsetsEXT, **must** be valid if they are dynamically used by the VkPipeline bound to the pipeline bind point used by this command and the bound VkPipeline was created with `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`
- VUID-vkCmdDrawMultiIndexedEXT-None-08604
  Descriptors in bound descriptor buffers, specified via vkCmdSetDescriptorBufferOffsetsEXT, **must** be valid if they are dynamically used by any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command
- VUID-vkCmdDrawMultiIndexedEXT-None-08117
  If the descriptors used by the VkPipeline bound to the pipeline bind point were specified via vkCmdSetDescriptorBufferOffsetsEXT, the bound VkPipeline **must** have been created with `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`
- VUID-vkCmdDrawMultiIndexedEXT-None-08119
  If a descriptor is dynamically used with a VkPipeline created with `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`, the descriptor memory **must** be resident
- VUID-vkCmdDrawMultiIndexedEXT-None-08605
  If a descriptor is dynamically used with a VkShaderEXT created with a `VkDescriptorSetLayout` that was created with `VK_DESCRIPTOR_SET_LAYOUT_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`, the descriptor memory **must** be resident
- VUID-vkCmdDrawMultiIndexedEXT-None-08606
  If the `shaderObject` feature is not enabled, a valid pipeline **must** be bound to the pipeline bind point used by this command
- VUID-vkCmdDrawMultiIndexedEXT-None-08608
  If a pipeline is bound to the pipeline bind point used by this command, there **must** not have been any calls to dynamic state setting commands for any state specified statically in the VkPipeline object bound to the pipeline bind point used by this command, since that pipeline was bound
- VUID-vkCmdDrawMultiIndexedEXT-None-08609
  If the VkPipeline object bound to the pipeline bind point used by this command or any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a VkSampler object that uses unnormalized coordinates, that sampler **must** not be used to sample from any VkImage with a VkImageView of the type `VK_IMAGE_VIEW_TYPE_3D`, `VK_IMAGE_VIEW_TYPE_CUBE`, `VK_IMAGE_VIEW_TYPE_1D_ARRAY`,

VK_IMAGE_VIEW_TYPE_2D_ARRAY or VK_IMAGE_VIEW_TYPE_CUBE_ARRAY, in any shader stage

- VUID-vkCmdDrawMultiIndexedEXT-None-08610
  If the VkPipeline object bound to the pipeline bind point used by this command or any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a VkSampler object that uses unnormalized coordinates, that sampler **must** not be used with any of the SPIR-V OpImageSample* or OpImageSparseSample* instructions with ImplicitLod, Dref or Proj in their name, in any shader stage
- VUID-vkCmdDrawMultiIndexedEXT-None-08611
  If the VkPipeline object bound to the pipeline bind point used by this command or any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a VkSampler object that uses unnormalized coordinates, that sampler **must** not be used with any of the SPIR-V OpImageSample* or OpImageSparseSample* instructions that includes a LOD bias or any offset values, in any shader stage
- VUID-vkCmdDrawMultiIndexedEXT-None-08607
  If the shaderObject is enabled, either a valid pipeline **must** be bound to the pipeline bind point used by this command, or a valid combination of valid and VK_NULL_HANDLE shader objects **must** be bound to every supported shader stage corresponding to the pipeline bind point used by this command
- VUID-vkCmdDrawMultiIndexedEXT-uniformBuffers-06935
  If any stage of the VkPipeline object bound to the pipeline bind point used by this command accesses a uniform buffer, and that stage was created without enabling either VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_EXT or VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_2_EXT for uniformBuffers, and the robustBufferAccess feature is not enabled, that stage **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point
- VUID-vkCmdDrawMultiIndexedEXT-None-08612
  If the robustBufferAccess feature is not enabled, and any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a uniform buffer, it **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point
- VUID-vkCmdDrawMultiIndexedEXT-storageBuffers-06936
  If any stage of the VkPipeline object bound to the pipeline bind point used by this command accesses a storage buffer, and that stage was created without enabling either VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_EXT or VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_2_EXT for storageBuffers, and the robustBufferAccess feature is not enabled, that stage **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point
- VUID-vkCmdDrawMultiIndexedEXT-None-08613
  If the robustBufferAccess feature is not enabled, and any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a storage buffer, it **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point
- VUID-vkCmdDrawMultiIndexedEXT-commandBuffer-02707

If commandBuffer is an unprotected command buffer and protectedNoFault is not supported, any resource accessed by bound shaders **must** not be a protected resource

- VUID-vkCmdDrawMultiIndexedEXT-None-06550
  If a bound shader accesses a VkSampler or VkImageView object that enables sampler $Y'C_BC_R$ conversion, that object **must** only be used with OpImageSample* or OpImageSparseSample* instructions
- VUID-vkCmdDrawMultiIndexedEXT-ConstOffset-06551
  If a bound shader accesses a VkSampler or VkImageView object that enables sampler $Y'C_BC_R$ conversion, that object **must** not use the ConstOffset and Offset operands
- VUID-vkCmdDrawMultiIndexedEXT-viewType-07752
  If a VkImageView is accessed as a result of this command, then the image view's viewType **must** match the Dim operand of the OpTypeImage as described in Compatibility Between SPIR-V Image Dimensions and Vulkan ImageView Types
- VUID-vkCmdDrawMultiIndexedEXT-format-07753
  If a VkImageView is accessed as a result of this command, then the numeric type of the image view's format and the Sampled Type operand of the OpTypeImage **must** match
- VUID-vkCmdDrawMultiIndexedEXT-OpImageWrite-08795
  If a VkImageView created with a format other than VK_FORMAT_A8_UNORM_KHR is accessed using OpImageWrite as a result of this command, then the Type of the Texel operand of that instruction **must** have at least as many components as the image view's format
- VUID-vkCmdDrawMultiIndexedEXT-OpImageWrite-08796
  If a VkImageView created with the format VK_FORMAT_A8_UNORM_KHR is accessed using OpImageWrite as a result of this command, then the Type of the Texel operand of that instruction **must** have four components
- VUID-vkCmdDrawMultiIndexedEXT-OpImageWrite-04469
  If a VkBufferView is accessed using OpImageWrite as a result of this command, then the Type of the Texel operand of that instruction **must** have at least as many components as the buffer view's format
- VUID-vkCmdDrawMultiIndexedEXT-SampledType-04470
  If a VkImageView with a VkFormat that has a 64-bit component width is accessed as a result of this command, the SampledType of the OpTypeImage operand of that instruction **must** have a Width of 64
- VUID-vkCmdDrawMultiIndexedEXT-SampledType-04471
  If a VkImageView with a VkFormat that has a component width less than 64-bit is accessed as a result of this command, the SampledType of the OpTypeImage operand of that instruction **must** have a Width of 32
- VUID-vkCmdDrawMultiIndexedEXT-SampledType-04472
  If a VkBufferView with a VkFormat that has a 64-bit component width is accessed as a result of this command, the SampledType of the OpTypeImage operand of that instruction **must** have a Width of 64
- VUID-vkCmdDrawMultiIndexedEXT-SampledType-04473
  If a VkBufferView with a VkFormat that has a component width less than 64-bit is accessed as a result of this command, the SampledType of the OpTypeImage operand of that

instruction **must** have a `Width` of 32

- VUID-vkCmdDrawMultiIndexedEXT-sparseImageInt64Atomics-04474
  If the `sparseImageInt64Atomics` feature is not enabled, VkImage objects created with the `VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT` flag **must** not be accessed by atomic instructions through an `OpTypeImage` with a `SampledType` with a `Width` of 64 by this command
- VUID-vkCmdDrawMultiIndexedEXT-sparseImageInt64Atomics-04475
  If the `sparseImageInt64Atomics` feature is not enabled, VkBuffer objects created with the `VK_BUFFER_CREATE_SPARSE_RESIDENCY_BIT` flag **must** not be accessed by atomic instructions through an `OpTypeImage` with a `SampledType` with a `Width` of 64 by this command
- VUID-vkCmdDrawMultiIndexedEXT-OpImageWeightedSampleQCOM-06971
  If `OpImageWeightedSampleQCOM` is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_2_WEIGHT_SAMPLED_IMAGE_BIT_QCOM`
- VUID-vkCmdDrawMultiIndexedEXT-OpImageWeightedSampleQCOM-06972
  If `OpImageWeightedSampleQCOM` uses a VkImageView as a sample weight image as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_2_WEIGHT_IMAGE_BIT_QCOM`
- VUID-vkCmdDrawMultiIndexedEXT-OpImageBoxFilterQCOM-06973
  If `OpImageBoxFilterQCOM` is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_2_BOX_FILTER_SAMPLED_BIT_QCOM`
- VUID-vkCmdDrawMultiIndexedEXT-OpImageBlockMatchSSDQCOM-06974
  If `OpImageBlockMatchSSDQCOM` is used to read from an VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM`
- VUID-vkCmdDrawMultiIndexedEXT-OpImageBlockMatchSADQCOM-06975
  If `OpImageBlockMatchSADQCOM` is used to read from an VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM`
- VUID-vkCmdDrawMultiIndexedEXT-OpImageBlockMatchSADQCOM-06976
  If `OpImageBlockMatchSADQCOM` or OpImageBlockMatchSSDQCOM is used to read from a reference image as result of this command, then the specified reference coordinates **must** not fail integer texel coordinate validation
- VUID-vkCmdDrawMultiIndexedEXT-OpImageWeightedSampleQCOM-06977
  If `OpImageWeightedSampleQCOM`, `OpImageBoxFilterQCOM`, `OpImageBlockMatchWindowSSDQCOM`, `OpImageBlockMatchWindowSADQCOM`, `OpImageBlockMatchGatherSSDQCOM`, `OpImageBlockMatchGatherSADQCOM`, `OpImageBlockMatchSSDQCOM`, or `OpImageBlockMatchSADQCOM` uses a VkSampler as a result of this command, then the sampler **must** have been created with `VK_SAMPLER_CREATE_IMAGE_PROCESSING_BIT_QCOM`
- VUID-vkCmdDrawMultiIndexedEXT-OpImageWeightedSampleQCOM-06978
  If any command other than `OpImageWeightedSampleQCOM`, `OpImageBoxFilterQCOM`, `OpImageBlockMatchWindowSSDQCOM`, `OpImageBlockMatchWindowSADQCOM`, `OpImageBlockMatchGatherSSDQCOM`, `OpImageBlockMatchGatherSADQCOM`, `OpImageBlockMatchSSDQCOM`, or `OpImageBlockMatchSADQCOM` uses a VkSampler as a result of this

command, then the sampler **must** not have been created with VK_SAMPLER_CREATE_IMAGE_PROCESSING_BIT_QCOM

- VUID-vkCmdDrawMultiIndexedEXT-OpImageBlockMatchWindow-09215
  If a OpImageBlockMatchWindow*QCOM or OpImageBlockMatchGather*QCOM instruction is used to read from an VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM
- VUID-vkCmdDrawMultiIndexedEXT-OpImageBlockMatchWindow-09216
  If a OpImageBlockMatchWindow*QCOM or OpImageBlockMatchGather*QCOM instruction is used to read from an VkImageView as a result of this command, then the image view's format **must** be a single-component format
- VUID-vkCmdDrawMultiIndexedEXT-OpImageBlockMatchWindow-09217
  If a OpImageBlockMatchWindow*QCOM or OpImageBlockMatchGather*QCOM read from a reference image as result of this command, then the specified reference coordinates **must** not fail integer texel coordinate validation
- VUID-vkCmdDrawMultiIndexedEXT-None-07288
  Any shader invocation executed by this command **must** terminate
- VUID-vkCmdDrawMultiIndexedEXT-None-09600
  If a descriptor with type equal to any of VK_DESCRIPTOR_TYPE_SAMPLE_WEIGHT_IMAGE_QCOM, VK_DESCRIPTOR_TYPE_BLOCK_MATCH_IMAGE_QCOM, VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE, VK_DESCRIPTOR_TYPE_STORAGE_IMAGE, or VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT is accessed as a result of this command, the image subresource identified by that descriptor **must** be in the image layout identified when the descriptor was written
- VUID-vkCmdDrawMultiIndexedEXT-renderPass-02684
  The current render pass **must** be compatible with the renderPass member of the VkGraphicsPipelineCreateInfo structure specified when creating the VkPipeline bound to VK_PIPELINE_BIND_POINT_GRAPHICS
- VUID-vkCmdDrawMultiIndexedEXT-subpass-02685
  The subpass index of the current render pass **must** be equal to the subpass member of the VkGraphicsPipelineCreateInfo structure specified when creating the VkPipeline bound to VK_PIPELINE_BIND_POINT_GRAPHICS
- VUID-vkCmdDrawMultiIndexedEXT-None-07748
  If any shader statically accesses an input attachment, a valid descriptor **must** be bound to the pipeline via a descriptor set
- VUID-vkCmdDrawMultiIndexedEXT-OpTypeImage-07468
  If any shader executed by this pipeline accesses an OpTypeImage variable with a Dim operand of SubpassData, it **must** be decorated with an InputAttachmentIndex that corresponds to a valid input attachment in the current subpass
- VUID-vkCmdDrawMultiIndexedEXT-None-07469
  Input attachment views accessed in a subpass **must** be created with the same VkFormat as the corresponding subpass definition, and be created with a VkImageView that is compatible with the attachment referenced by the subpass' pInputAttachments [InputAttachmentIndex] in the currently bound VkFramebuffer as specified by Fragment Input Attachment Compatibility

- VUID-vkCmdDrawMultiIndexedEXT-pDepthInputAttachmentIndex-09595
  Input attachment views accessed in a dynamic render pass with a `InputAttachmentIndex` referenced by VkRenderingInputAttachmentIndexInfoKHR, or no `InputAttachmentIndex` if VkRenderingInputAttachmentIndexInfoKHR:`pDepthInputAttachmentIndex` or VkRenderingInputAttachmentIndexInfoKHR:`pStencilInputAttachmentIndex` are `NULL`, **must** be created with a VkImageView that is compatible with the corresponding color, depth, or stencil attachment in VkRenderingInfo
- VUID-vkCmdDrawMultiIndexedEXT-pDepthInputAttachmentIndex-09596
  Input attachment views accessed in a dynamic render pass via a shader object **must** have an `InputAttachmentIndex` if both VkRenderingInputAttachmentIndexInfoKHR:`pDepthInputAttachmentIndex` and VkRenderingInputAttachmentIndexInfoKHR:`pStencilInputAttachmentIndex` are non-`NULL`
- VUID-vkCmdDrawMultiIndexedEXT-InputAttachmentIndex-09597
  If an input attachment view accessed in a dynamic render pass via a shader object has an `InputAttachmentIndex`, the `InputAttachmentIndex` **must** match an index in VkRenderingInputAttachmentIndexInfoKHR
- VUID-vkCmdDrawMultiIndexedEXT-None-06537
  Memory backing image subresources used as attachments in the current render pass **must** not be written in any way other than as an attachment by this command
- VUID-vkCmdDrawMultiIndexedEXT-None-09000
  If a color attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass, it is not in the `VK_IMAGE_LAYOUT_ATTACHMENT_FEEDBACK_LOOP_OPTIMAL_EXT` image layout, and either:
  - the `VK_PIPELINE_CREATE_COLOR_ATTACHMENT_FEEDBACK_LOOP_BIT_EXT` is set on the currently bound pipeline or
  - the last call to vkCmdSetAttachmentFeedbackLoopEnableEXT included `VK_IMAGE_ASPECT_COLOR_BIT` and
    - there is no currently bound graphics pipeline or
    - the currently bound graphics pipeline was created with `VK_DYNAMIC_STATE_ATTACHMENT_FEEDBACK_LOOP_ENABLE_EXT`

  it **must** not be accessed in any way other than as an attachment by this command
- VUID-vkCmdDrawMultiIndexedEXT-None-09001
  If a depth attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass, it is not in the `VK_IMAGE_LAYOUT_ATTACHMENT_FEEDBACK_LOOP_OPTIMAL_EXT` image layout, and either:
  - the `VK_PIPELINE_CREATE_DEPTH_STENCIL_ATTACHMENT_FEEDBACK_LOOP_BIT_EXT` is set on the currently bound pipeline or
  - the last call to vkCmdSetAttachmentFeedbackLoopEnableEXT included `VK_IMAGE_ASPECT_DEPTH_BIT` and
    - there is no currently bound graphics pipeline or
    - the currently bound graphics pipeline was created with

VK_DYNAMIC_STATE_ATTACHMENT_FEEDBACK_LOOP_ENABLE_EXT

it **must** not be accessed in any way other than as an attachment by this command

- VUID-vkCmdDrawMultiIndexedEXT-None-09002
  If a stencil attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass, it is not in the VK_IMAGE_LAYOUT_ATTACHMENT_FEEDBACK_LOOP_OPTIMAL_EXT image layout, and either:
  - the VK_PIPELINE_CREATE_DEPTH_STENCIL_ATTACHMENT_FEEDBACK_LOOP_BIT_EXT is set on the currently bound pipeline or
  - the last call to vkCmdSetAttachmentFeedbackLoopEnableEXT included VK_IMAGE_ASPECT_STENCIL_BIT and
    - there is no currently bound graphics pipeline or
    - the currently bound graphics pipeline was created with VK_DYNAMIC_STATE_ATTACHMENT_FEEDBACK_LOOP_ENABLE_EXT

  it **must** not be accessed in any way other than as an attachment by this command

- VUID-vkCmdDrawMultiIndexedEXT-None-09003
  If an attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass, it **must** not be accessed in any way other than as an attachment, storage image, or sampled image by this command
- VUID-vkCmdDrawMultiIndexedEXT-None-06539
  If any previously recorded command in the current subpass accessed an image subresource used as an attachment in this subpass in any way other than as an attachment, this command **must** not write to that image subresource as an attachment
- VUID-vkCmdDrawMultiIndexedEXT-None-06886
  If the current render pass instance uses a depth/stencil attachment with a read-only layout for the depth aspect, depth writes **must** be disabled
- VUID-vkCmdDrawMultiIndexedEXT-None-06887
  If the current render pass instance uses a depth/stencil attachment with a read-only layout for the stencil aspect, both front and back writeMask are not zero, and stencil test is enabled, all stencil ops **must** be VK_STENCIL_OP_KEEP
- VUID-vkCmdDrawMultiIndexedEXT-None-07831
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT dynamic state enabled then vkCmdSetViewport **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiIndexedEXT-None-07832
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SCISSOR dynamic state enabled then vkCmdSetScissor **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiIndexedEXT-None-07833
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_WIDTH dynamic state enabled then vkCmdSetLineWidth **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMultiIndexedEXT-None-08617
  If a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE`, and the most recent call to vkCmdSetPolygonModeEXT in the current command buffer set `polygonMode` to `VK_POLYGON_MODE_LINE`, vkCmdSetLineWidth **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiIndexedEXT-None-08618
  If a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE`, and the most recent call to vkCmdSetPrimitiveTopologyEXT in the current command buffer set `primitiveTopology` to any line topology, vkCmdSetLineWidth **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiIndexedEXT-None-08619
  If a shader object that outputs line primitives is bound to the `VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT` or `VK_SHADER_STAGE_GEOMETRY_BIT` stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE`, vkCmdSetLineWidth **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiIndexedEXT-None-07834
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_DEPTH_BIAS` dynamic state enabled, the current value of `rasterizerDiscardEnable` is `VK_FALSE`, and the current value of `depthBiasEnable` is `VK_TRUE`, then vkCmdSetDepthBounds or vkCmdSetDepthBias2EXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiIndexedEXT-None-07835
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_BLEND_CONSTANTS` dynamic state enabled then vkCmdSetBlendConstants **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiIndexedEXT-None-08621
  If a shader object is bound to the `VK_SHADER_STAGE_FRAGMENT_BIT` stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE`, and the most recent call to vkCmdSetColorBlendEnableEXT in the current command buffer set any element of `pColorBlendEnables` to `VK_TRUE`, and the most recent call to vkCmdSetColorBlendEquationEXT in the current command buffer set the same element of `pColorBlendEquations` to a `VkColorBlendEquationEXT` structure with any VkBlendFactor member with a value of `VK_BLEND_FACTOR_CONSTANT_COLOR`, `VK_BLEND_FACTOR_ONE_MINUS_CONSTANT_COLOR`, `VK_BLEND_FACTOR_CONSTANT_ALPHA`, or `VK_BLEND_FACTOR_ONE_MINUS_CONSTANT_ALPHA`, vkCmdSetBlendConstants **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMultiIndexedEXT-None-07836
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_BOUNDS dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of depthBoundsTestEnable is VK_TRUE, then vkCmdSetDepthBounds **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiIndexedEXT-None-07837
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_STENCIL_COMPARE_MASK dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of stencilTestEnable is VK_TRUE, then vkCmdSetStencilCompareMask **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiIndexedEXT-None-07838
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_STENCIL_WRITE_MASK dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of stencilTestEnable is VK_TRUE, then vkCmdSetStencilWriteMask **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiIndexedEXT-None-07839
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_STENCIL_REFERENCE dynamic state enabled, the current value of and rasterizerDiscardEnable is VK_FALSE, the current value of stencilTestEnable is VK_TRUE, then vkCmdSetStencilReference **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiIndexedEXT-maxMultiviewInstanceIndex-02688
  If the draw is recorded in a render pass instance with multiview enabled, the maximum instance index **must** be less than or equal to VkPhysicalDeviceMultiviewProperties::maxMultiviewInstanceIndex
- VUID-vkCmdDrawMultiIndexedEXT-sampleLocationsEnable-02689
  If the bound graphics pipeline was created with VkPipelineSampleLocationsStateCreateInfoEXT::sampleLocationsEnable set to VK_TRUE and the current subpass has a depth/stencil attachment, then that attachment **must** have been created with the VK_IMAGE_CREATE_SAMPLE_LOCATIONS_COMPATIBLE_DEPTH_BIT_EXT bit set
- VUID-vkCmdDrawMultiIndexedEXT-None-06666
  If the VK_EXT_sample_locations extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of sampleLocationsEnable is VK_TRUE, then vkCmdSetSampleLocationsEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiIndexedEXT-None-07840

If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_CULL_MODE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetCullModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMultiIndexedEXT-None-07841
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_FRONT_FACE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetFrontFaceEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiIndexedEXT-None-07843
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_TEST_ENABLE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, vkCmdSetDepthTestEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiIndexedEXT-None-07844
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_WRITE_ENABLE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetDepthWriteEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiIndexedEXT-None-07845
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_COMPARE_OP dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of depthTestEnable is VK_TRUE, then vkCmdSetDepthCompareOpEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiIndexedEXT-None-07846
  If the depthBounds feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_BOUNDS_TEST_ENABLE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetDepthBoundsTestEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiIndexedEXT-None-07847
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_STENCIL_TEST_ENABLE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetStencilTestEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiIndexedEXT-None-07848
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which

was created with the VK_DYNAMIC_STATE_STENCIL_OP dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, the current value of stencilTestEnable is VK_TRUE, then vkCmdSetStencilOpEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMultiIndexedEXT-viewportCount-03417
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, but not the VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT dynamic state enabled, then vkCmdSetViewportWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the viewportCount parameter of vkCmdSetViewportWithCountEXT **must** match the VkPipelineViewportStateCreateInfo::scissorCount of the pipeline
- VUID-vkCmdDrawMultiIndexedEXT-scissorCount-03418
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT dynamic state enabled, but not the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, then vkCmdSetScissorWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the scissorCount parameter of vkCmdSetScissorWithCountEXT **must** match the VkPipelineViewportStateCreateInfo::viewportCount of the pipeline
- VUID-vkCmdDrawMultiIndexedEXT-viewportCount-03419
  If the bound graphics pipeline state was created with both the VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT and VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic states enabled then both vkCmdSetViewportWithCountEXT and vkCmdSetScissorWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the viewportCount parameter of vkCmdSetViewportWithCountEXT **must** match the scissorCount parameter of vkCmdSetScissorWithCountEXT
- VUID-vkCmdDrawMultiIndexedEXT-None-08635
  If a shader object is bound to any graphics stage, then both vkCmdSetViewportWithCountEXT and vkCmdSetScissorWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the viewportCount parameter of vkCmdSetViewportWithCountEXT **must** match the scissorCount parameter of vkCmdSetScissorWithCountEXT
- VUID-vkCmdDrawMultiIndexedEXT-viewportCount-04137
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, but not the VK_DYNAMIC_STATE_VIEWPORT_W_SCALING_NV dynamic state enabled, then the bound graphics pipeline **must** have been created with VkPipelineViewportWScalingStateCreateInfoNV::viewportCount greater or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawMultiIndexedEXT-viewportCount-04138
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT and VK_DYNAMIC_STATE_VIEWPORT_W_SCALING_NV dynamic states enabled then the viewportCount parameter in the last call to

vkCmdSetViewportWScalingNV **must** be greater than or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT

- VUID-vkCmdDrawMultiIndexedEXT-None-09232
  If the VK_NV_clip_space_w_scaling extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetViewportWScalingEnableNV in the current command buffer set viewportWScalingEnable to VK_TRUE, then vkCmdSetViewportWScalingNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiIndexedEXT-None-08636
  If the VK_NV_clip_space_w_scaling extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetViewportWScalingEnableNV in the current command buffer set viewportWScalingEnable to VK_TRUE, then the viewportCount parameter in the last call to vkCmdSetViewportWScalingNV **must** be greater than or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawMultiIndexedEXT-viewportCount-04139
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, but not the VK_DYNAMIC_STATE_VIEWPORT_SHADING_RATE_PALETTE_NV dynamic state enabled, then the bound graphics pipeline **must** have been created with VkPipelineViewportShadingRateImageStateCreateInfoNV::viewportCount greater or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawMultiIndexedEXT-viewportCount-04140
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT and VK_DYNAMIC_STATE_VIEWPORT_SHADING_RATE_PALETTE_NV dynamic states enabled then the viewportCount parameter in the last call to vkCmdSetViewportShadingRatePaletteNV **must** be greater than or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawMultiIndexedEXT-shadingRateImage-09233
  If the shadingRateImage feature is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetCoarseSampleOrderNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiIndexedEXT-shadingRateImage-09234
  If the shadingRateImage feature is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetShadingRateImageEnableNV in the current command buffer set shadingRateImageEnable to VK_TRUE, then vkCmdSetViewportShadingRatePaletteNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiIndexedEXT-None-08637
  If the shadingRateImage feature is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current

command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetShadingRateImageEnableNV in the current command buffer set shadingRateImageEnable to VK_TRUE, then the viewportCount parameter in the last call to vkCmdSetViewportShadingRatePaletteNV **must** be greater than or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT

- VUID-vkCmdDrawMultiIndexedEXT-VkPipelineVieportCreateInfo-04141
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled and a VkPipelineViewportSwizzleStateCreateInfoNV structure chained from VkPipelineViewportStateCreateInfo, then the bound graphics pipeline **must** have been created with VkPipelineViewportSwizzleStateCreateInfoNV::viewportCount greater or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawMultiIndexedEXT-VkPipelineVieportCreateInfo-04142
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled and a VkPipelineViewportExclusiveScissorStateCreateInfoNV structure chained from VkPipelineViewportStateCreateInfo, then the bound graphics pipeline **must** have been created with VkPipelineViewportExclusiveScissorStateCreateInfoNV::exclusiveScissorCount greater or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawMultiIndexedEXT-None-07878
  If the exclusiveScissor feature is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_EXCLUSIVE_SCISSOR_ENABLE_NV dynamic state enabled, then vkCmdSetExclusiveScissorEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiIndexedEXT-None-07879
  If the exclusiveScissor feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_EXCLUSIVE_SCISSOR_NV dynamic state enabled, and the most recent call to vkCmdSetExclusiveScissorEnableNV in the current command buffer set any element of pExclusiveScissorEnables to VK_TRUE, then vkCmdSetExclusiveScissorNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiIndexedEXT-None-04876
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_RASTERIZER_DISCARD_ENABLE dynamic state enabled, then vkCmdSetRasterizerDiscardEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiIndexedEXT-None-04877
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_BIAS_ENABLE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetDepthBiasEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMultiIndexedEXT-logicOp-04878
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_LOGIC_OP_EXT dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of logicOpEnable is VK_TRUE, then vkCmdSetLogicOpEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiIndexedEXT-primitiveFragmentShadingRateWithMultipleViewports-04552
  If the primitiveFragmentShadingRateWithMultipleViewports limit is not supported, the bound graphics pipeline was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, and any of the shader stages of the bound graphics pipeline write to the PrimitiveShadingRateKHR built-in, then vkCmdSetViewportWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the viewportCount parameter of vkCmdSetViewportWithCountEXT **must** be 1
- VUID-vkCmdDrawMultiIndexedEXT-primitiveFragmentShadingRateWithMultipleViewports-08642
  If the primitiveFragmentShadingRateWithMultipleViewports limit is not supported, and any shader object bound to a graphics stage writes to the PrimitiveShadingRateKHR built-in, then vkCmdSetViewportWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the viewportCount parameter of vkCmdSetViewportWithCountEXT **must** be 1
- VUID-vkCmdDrawMultiIndexedEXT-blendEnable-04727
  If rasterization is not disabled in the bound graphics pipeline, then for each color attachment in the subpass, if the corresponding image view's format features do not contain VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT, then the blendEnable member of the corresponding element of the pAttachments member of pColorBlendState **must** be VK_FALSE
- VUID-vkCmdDrawMultiIndexedEXT-None-08643
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then for each color attachment in the render pass, if the corresponding image view's format features do not contain VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT, then the corresponding member of pColorBlendEnables in the most recent call to vkCmdSetColorBlendEnableEXT in the current command buffer that affected that attachment index **must** have been VK_FALSE
- VUID-vkCmdDrawMultiIndexedEXT-multisampledRenderToSingleSampled-07284
  If rasterization is not disabled in the bound graphics pipeline, and none of the following is enabled:
  - the VK_AMD_mixed_attachment_samples extension
  - the VK_NV_framebuffer_mixed_samples extension
  - the multisampledRenderToSingleSampled feature

  then rasterizationSamples for the currently bound graphics pipeline **must** be the same as the current subpass color and/or depth/stencil attachments

- VUID-vkCmdDrawMultiIndexedEXT-None-08644
  If a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and none of the following is enabled:

  - the VK_AMD_mixed_attachment_samples extension
  - the VK_NV_framebuffer_mixed_samples extension
  - the multisampledRenderToSingleSampled feature

  then the most recent call to vkCmdSetRasterizationSamplesEXT in the current command buffer **must** have set rasterizationSamples to be the same as the number of samples for the current render pass color and/or depth/stencil attachments

- VUID-vkCmdDrawMultiIndexedEXT-None-08876
  If a shader object is bound to any graphics stage, the current render pass instance **must** have been begun with vkCmdBeginRenderingKHR
- VUID-vkCmdDrawMultiIndexedEXT-imageView-06172
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the imageView member of pDepthAttachment is not VK_NULL_HANDLE, and the layout member of pDepthAttachment is VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL, this command **must** not write any values to the depth attachment
- VUID-vkCmdDrawMultiIndexedEXT-imageView-06173
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the imageView member of pStencilAttachment is not VK_NULL_HANDLE, and the layout member of pStencilAttachment is VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL, this command **must** not write any values to the stencil attachment
- VUID-vkCmdDrawMultiIndexedEXT-imageView-06174
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the imageView member of pDepthAttachment is not VK_NULL_HANDLE, and the layout member of pDepthAttachment is VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_STENCIL_ATTACHMENT_OPTIMAL, this command **must** not write any values to the depth attachment
- VUID-vkCmdDrawMultiIndexedEXT-imageView-06175
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the imageView member of pStencilAttachment is not VK_NULL_HANDLE, and the layout member of pStencilAttachment is VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_STENCIL_READ_ONLY_OPTIMAL, this command **must** not write any values to the stencil attachment
- VUID-vkCmdDrawMultiIndexedEXT-imageView-06176
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the imageView member of pDepthAttachment is not VK_NULL_HANDLE, and the layout member of pDepthAttachment is VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL, this command **must** not write any values to the depth attachment
- VUID-vkCmdDrawMultiIndexedEXT-imageView-06177
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the imageView member of pStencilAttachment is not VK_NULL_HANDLE, and the layout member of pStencilAttachment is VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL, this

command **must** not write any values to the stencil attachment

- VUID-vkCmdDrawMultiIndexedEXT-viewMask-06178
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the currently bound graphics pipeline **must** have been created with a VkPipelineRenderingCreateInfo::viewMask equal to VkRenderingInfo::viewMask

- VUID-vkCmdDrawMultiIndexedEXT-colorAttachmentCount-06179
  If the dynamicRenderingUnusedAttachments feature is not enabled and the current render pass instance was begun with vkCmdBeginRenderingKHR, the currently bound graphics pipeline **must** have been created with a VkPipelineRenderingCreateInfo::colorAttachmentCount equal to VkRenderingInfo::colorAttachmentCount

- VUID-vkCmdDrawMultiIndexedEXT-dynamicRenderingUnusedAttachments-08910
  If the dynamicRenderingUnusedAttachments feature is not enabled, and the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingInfo::colorAttachmentCount greater than 0, then each element of the VkRenderingInfo::pColorAttachments array with an imageView not equal to VK_NULL_HANDLE **must** have been created with a VkFormat equal to the corresponding element of VkPipelineRenderingCreateInfo::pColorAttachmentFormats used to create the currently bound graphics pipeline

- VUID-vkCmdDrawMultiIndexedEXT-dynamicRenderingUnusedAttachments-08912
  If the dynamicRenderingUnusedAttachments feature is not enabled, and the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingInfo::colorAttachmentCount greater than 0, then each element of the VkRenderingInfo::pColorAttachments array with an imageView equal to VK_NULL_HANDLE **must** have the corresponding element of VkPipelineRenderingCreateInfo::pColorAttachmentFormats used to create the currently bound pipeline equal to VK_FORMAT_UNDEFINED

- VUID-vkCmdDrawMultiIndexedEXT-dynamicRenderingUnusedAttachments-08911
  If the dynamicRenderingUnusedAttachments feature is enabled, and the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingInfo::colorAttachmentCount greater than 0, then each element of the VkRenderingInfo::pColorAttachments array with an imageView not equal to VK_NULL_HANDLE **must** have been created with a VkFormat equal to the corresponding element of VkPipelineRenderingCreateInfo::pColorAttachmentFormats used to create the currently bound graphics pipeline, or the corresponding element of VkPipelineRenderingCreateInfo::pColorAttachmentFormats, if it exists, **must** be VK_FORMAT_UNDEFINED

- VUID-vkCmdDrawMultiIndexedEXT-colorAttachmentCount-09362
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, with a VkRenderingInfo::colorAttachmentCount equal to 1, there is no shader object bound to any graphics stage, and a color attachment with a resolve mode of VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID, each element of the VkRenderingInfo::pColorAttachments array with a resolveImageView not equal to VK_NULL_HANDLE **must** have been created with an image created with a VkExternalFormatANDROID::externalFormat value equal to the VkExternalFormatANDROID::externalFormat value used to create the currently bound graphics pipeline

- VUID-vkCmdDrawMultiIndexedEXT-None-09363
  If there is no shader object bound to any graphics stage, the current render pass instance was begun with vkCmdBeginRenderingKHR and a VkRenderingInfo::colorAttachmentCount equal to 1, and a color attachment with a resolve mode of VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID, each element of the VkRenderingInfo::pColorAttachments array with a imageView not equal to VK_NULL_HANDLE **must** have been created with an image created with a VkExternalFormatANDROID::externalFormat value equal to the VkExternalFormatANDROID::externalFormat value used to create the currently bound graphics pipeline
- VUID-vkCmdDrawMultiIndexedEXT-None-09364
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is no shader object bound to any graphics stage, and the currently bound graphics pipeline was created with a non-zero VkExternalFormatANDROID::externalFormat value and with the VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT dynamic state enabled, then vkCmdSetColorBlendEnableEXT **must** have set the blend enable to VK_FALSE prior to this drawing command
- VUID-vkCmdDrawMultiIndexedEXT-None-09365
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is no shader object bound to any graphics stage, and the currently bound graphics pipeline was created with a non-zero VkExternalFormatANDROID::externalFormat value and with the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT dynamic state enabled, then vkCmdSetRasterizationSamplesEXT **must** have set rasterizationSamples to VK_SAMPLE_COUNT_1_BIT prior to this drawing command
- VUID-vkCmdDrawMultiIndexedEXT-None-09366
  If there is a shader object bound to any graphics stage, and the current render pass includes a color attachment that uses the VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID resolve mode, then vkCmdSetColorBlendEnableEXT **must** have set blend enable to VK_FALSE prior to this drawing command
- VUID-vkCmdDrawMultiIndexedEXT-rasterizationSamples-09367
  If there is a shader object bound to any graphics stage, and the current render pass includes a color attachment that uses the VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID resolve mode, then vkCmdSetRasterizationSamplesEXT **must** have set rasterizationSamples to VK_SAMPLE_COUNT_1_BIT prior to this drawing command
- VUID-vkCmdDrawMultiIndexedEXT-None-09368
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is no shader object bound to any graphics stage, and the currently bound graphics pipeline was created with a non-zero VkExternalFormatANDROID::externalFormat value and with the VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR dynamic state enabled, then vkCmdSetFragmentShadingRateKHR **must** have set pFragmentSize->width to 1 prior to this drawing command
- VUID-vkCmdDrawMultiIndexedEXT-None-09369
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is

no shader object bound to any graphics stage, and the currently bound graphics pipeline was created with a non-zero VkExternalFormatANDROID::externalFormat value and with the VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR dynamic state enabled, then vkCmdSetFragmentShadingRateKHR **must** have set pFragmentSize->height to 1 prior to this drawing command

- VUID-vkCmdDrawMultiIndexedEXT-pFragmentSize-09370
  If there is a shader object bound to any graphics stage, and the current render pass includes a color attachment that uses the VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID resolve mode, then vkCmdSetFragmentShadingRateKHR **must** have set pFragmentSize->width to 1 prior to this drawing command
- VUID-vkCmdDrawMultiIndexedEXT-pFragmentSize-09371
  If there is a shader object bound to any graphics stage, and the current render pass includes a color attachment that uses the VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID resolve mode, then vkCmdSetFragmentShadingRateKHR **must** have set pFragmentSize->height to 1 prior to this drawing command
- VUID-vkCmdDrawMultiIndexedEXT-None-07749
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_WRITE_ENABLE_EXT dynamic state enabled then vkCmdSetColorWriteEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiIndexedEXT-None-08646
  If the colorWriteEnable feature is enabled on the device, and a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetColorWriteEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiIndexedEXT-attachmentCount-07750
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_WRITE_ENABLE_EXT dynamic state enabled then the attachmentCount parameter of vkCmdSetColorWriteEnableEXT **must** be greater than or equal to the VkPipelineColorBlendStateCreateInfo::attachmentCount of the currently bound graphics pipeline
- VUID-vkCmdDrawMultiIndexedEXT-None-08647
  If the colorWriteEnable feature is enabled on the device, and a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then the attachmentCount parameter of most recent call to vkCmdSetColorWriteEnableEXT in the current command buffer **must** be greater than or equal to the number of color attachments in the current render pass instance
- VUID-vkCmdDrawMultiIndexedEXT-None-07751
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_DISCARD_RECTANGLE_EXT dynamic state enabled then

vkCmdSetDiscardRectangleEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command for each discard rectangle in VkPipelineDiscardRectangleStateCreateInfoEXT::discardRectangleCount

- VUID-vkCmdDrawMultiIndexedEXT-None-07880
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_DISCARD_RECTANGLE_ENABLE_EXT dynamic state enabled then vkCmdSetDiscardRectangleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiIndexedEXT-rasterizerDiscardEnable-09236
  If the VK_EXT_discard_rectangles extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetDiscardRectangleEnableEXT in the current command buffer set discardRectangleEnable to VK_TRUE, then vkCmdSetDiscardRectangleEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiIndexedEXT-None-08648
  If the VK_EXT_discard_rectangles extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetDiscardRectangleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiIndexedEXT-None-07881
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_DISCARD_RECTANGLE_MODE_EXT dynamic state enabled then vkCmdSetDiscardRectangleModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiIndexedEXT-None-08649
  If the VK_EXT_discard_rectangles extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetDiscardRectangleEnableEXT in the current command buffer set discardRectangleEnable to VK_TRUE, then vkCmdSetDiscardRectangleModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiIndexedEXT-dynamicRenderingUnusedAttachments-08913
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is not enabled, and VkRenderingInfo::pDepthAttachment->imageView was VK_NULL_HANDLE, the value of VkPipelineRenderingCreateInfo::depthAttachmentFormat used to create the currently bound graphics pipeline **must** be equal to VK_FORMAT_UNDEFINED
- VUID-vkCmdDrawMultiIndexedEXT-dynamicRenderingUnusedAttachments-08914
  If current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is not enabled, and VkRenderingInfo::pDepthAttachment->imageView was not VK_NULL_HANDLE, the value of

VkPipelineRenderingCreateInfo::depthAttachmentFormat used to create the currently bound graphics pipeline **must** be equal to the VkFormat used to create VkRenderingInfo::pDepthAttachment->imageView

- VUID-vkCmdDrawMultiIndexedEXT-dynamicRenderingUnusedAttachments-08915
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is enabled, VkRenderingInfo::pDepthAttachment->imageView was not VK_NULL_HANDLE, and the value of VkPipelineRenderingCreateInfo::depthAttachmentFormat used to create the currently bound graphics pipeline was not equal to the VkFormat used to create VkRenderingInfo::pDepthAttachment->imageView, the value of the format **must** be VK_FORMAT_UNDEFINED
- VUID-vkCmdDrawMultiIndexedEXT-dynamicRenderingUnusedAttachments-08916
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is not enabled, and VkRenderingInfo::pStencilAttachment->imageView was VK_NULL_HANDLE, the value of VkPipelineRenderingCreateInfo::stencilAttachmentFormat used to create the currently bound graphics pipeline **must** be equal to VK_FORMAT_UNDEFINED
- VUID-vkCmdDrawMultiIndexedEXT-dynamicRenderingUnusedAttachments-08917
  If current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is not enabled, and VkRenderingInfo::pStencilAttachment->imageView was not VK_NULL_HANDLE, the value of VkPipelineRenderingCreateInfo::stencilAttachmentFormat used to create the currently bound graphics pipeline **must** be equal to the VkFormat used to create VkRenderingInfo::pStencilAttachment->imageView
- VUID-vkCmdDrawMultiIndexedEXT-dynamicRenderingUnusedAttachments-08918
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is enabled, VkRenderingInfo::pStencilAttachment->imageView was not VK_NULL_HANDLE, and the value of VkPipelineRenderingCreateInfo::stencilAttachmentFormat used to create the currently bound graphics pipeline was not equal to the VkFormat used to create VkRenderingInfo::pStencilAttachment->imageView, the value of the format **must** be VK_FORMAT_UNDEFINED
- VUID-vkCmdDrawMultiIndexedEXT-imageView-06183
  If the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingFragmentShadingRateAttachmentInfoKHR::imageView was not VK_NULL_HANDLE, the currently bound graphics pipeline **must** have been created with VK_PIPELINE_CREATE_RENDERING_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR
- VUID-vkCmdDrawMultiIndexedEXT-imageView-06184
  If the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingFragmentDensityMapAttachmentInfoEXT::imageView was not VK_NULL_HANDLE, the currently bound graphics pipeline **must** have been created with VK_PIPELINE_CREATE_RENDERING_FRAGMENT_DENSITY_MAP_ATTACHMENT_BIT_EXT
- VUID-vkCmdDrawMultiIndexedEXT-colorAttachmentCount-06185
  If the currently bound pipeline was created with a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and the current render pass instance was begun with vkCmdBeginRenderingKHR with a VkRenderingInfo::colorAttachmentCount parameter greater than 0, then each element of the VkRenderingInfo::pColorAttachments

array with a imageView not equal to VK_NULL_HANDLE **must** have been created with a sample count equal to the corresponding element of the pColorAttachmentSamples member of VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV used to create the currently bound graphics pipeline

- VUID-vkCmdDrawMultiIndexedEXT-pDepthAttachment-06186
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the currently bound pipeline was created with a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and VkRenderingInfo::pDepthAttachment->imageView was not VK_NULL_HANDLE, the value of the depthStencilAttachmentSamples member of VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV used to create the currently bound graphics pipeline **must** be equal to the sample count used to create VkRenderingInfo::pDepthAttachment->imageView
- VUID-vkCmdDrawMultiIndexedEXT-pStencilAttachment-06187
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the currently bound pipeline was created with a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and VkRenderingInfo::pStencilAttachment->imageView was not VK_NULL_HANDLE, the value of the depthStencilAttachmentSamples member of VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV used to create the currently bound graphics pipeline **must** be equal to the sample count used to create VkRenderingInfo::pStencilAttachment->imageView
- VUID-vkCmdDrawMultiIndexedEXT-multisampledRenderToSingleSampled-07285
  If the currently bound pipeline was created without a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and the multisampledRenderToSingleSampled feature is not enabled, and the current render pass instance was begun with vkCmdBeginRenderingKHR with a VkRenderingInfo::colorAttachmentCount parameter greater than 0, then each element of the VkRenderingInfo::pColorAttachments array with a imageView not equal to VK_NULL_HANDLE **must** have been created with a sample count equal to the value of rasterizationSamples for the currently bound graphics pipeline
- VUID-vkCmdDrawMultiIndexedEXT-multisampledRenderToSingleSampled-07286
  If the currently bound pipeline was created without a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and the multisampledRenderToSingleSampled feature is not enabled, and VkRenderingInfo::pDepthAttachment->imageView was not VK_NULL_HANDLE, the value of rasterizationSamples for the currently bound graphics pipeline **must** be equal to the sample count used to create VkRenderingInfo::pDepthAttachment->imageView
- VUID-vkCmdDrawMultiIndexedEXT-multisampledRenderToSingleSampled-07287
  If the currently bound pipeline was created without a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and the multisampledRenderToSingleSampled feature is not enabled, and VkRenderingInfo::pStencilAttachment->imageView was not VK_NULL_HANDLE, the value of rasterizationSamples for the currently bound graphics pipeline **must** be equal to the sample count used to create VkRenderingInfo::pStencilAttachment->imageView
- VUID-vkCmdDrawMultiIndexedEXT-pNext-07935
  If this command has been called inside a render pass instance started with

vkCmdBeginRenderingKHR, and the pNext chain of VkRenderingInfo includes a VkMultisampledRenderToSingleSampledInfoEXT structure with multisampledRenderToSingleSampledEnable equal to VK_TRUE, then the value of rasterizationSamples for the currently bound graphics pipeline **must** be equal to VkMultisampledRenderToSingleSampledInfoEXT::rasterizationSamples

- VUID-vkCmdDrawMultiIndexedEXT-renderPass-06198
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the currently bound pipeline **must** have been created with a VkGraphicsPipelineCreateInfo::renderPass equal to VK_NULL_HANDLE

- VUID-vkCmdDrawMultiIndexedEXT-pColorAttachments-08963
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is a graphics pipeline bound with a fragment shader that statically writes to a color attachment, the color write mask is not zero, color writes are enabled, and the corresponding element of the VkRenderingInfo::pColorAttachments->imageView was not VK_NULL_HANDLE, then the corresponding element of VkPipelineRenderingCreateInfo::pColorAttachmentFormats used to create the pipeline **must** not be VK_FORMAT_UNDEFINED

- VUID-vkCmdDrawMultiIndexedEXT-pDepthAttachment-08964
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is a graphics pipeline bound, depth test is enabled, depth write is enabled, and the VkRenderingInfo::pDepthAttachment->imageView was not VK_NULL_HANDLE, then the VkPipelineRenderingCreateInfo::depthAttachmentFormat used to create the pipeline **must** not be VK_FORMAT_UNDEFINED

- VUID-vkCmdDrawMultiIndexedEXT-pStencilAttachment-08965
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is a graphics pipeline bound, stencil test is enabled and the VkRenderingInfo::pStencilAttachment->imageView was not VK_NULL_HANDLE, then the VkPipelineRenderingCreateInfo::stencilAttachmentFormat used to create the pipeline **must** not be VK_FORMAT_UNDEFINED

- VUID-vkCmdDrawMultiIndexedEXT-primitivesGeneratedQueryWithRasterizerDiscard-06708
  If the primitivesGeneratedQueryWithRasterizerDiscard feature is not enabled and the VK_QUERY_TYPE_PRIMITIVES_GENERATED_EXT query is active, rasterization discard **must** not be enabled

- VUID-vkCmdDrawMultiIndexedEXT-primitivesGeneratedQueryWithNonZeroStreams-06709
  If the primitivesGeneratedQueryWithNonZeroStreams feature is not enabled and the VK_QUERY_TYPE_PRIMITIVES_GENERATED_EXT query is active, the bound graphics pipeline **must** not have been created with a non-zero value in VkPipelineRasterizationStateStreamCreateInfoEXT::rasterizationStream

- VUID-vkCmdDrawMultiIndexedEXT-None-07619
  If a shader object is bound to the VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_TESSELLATION_DOMAIN_ORIGIN_EXT dynamic state enabled, then vkCmdSetTessellationDomainOriginEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMultiIndexedEXT-None-07620
  If the depthClamp feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_CLAMP_ENABLE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetDepthClampEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiIndexedEXT-None-07621
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_POLYGON_MODE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetPolygonModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiIndexedEXT-None-07622
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetRasterizationSamplesEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiIndexedEXT-None-07623
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_SAMPLE_MASK_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetSampleMaskEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiIndexedEXT-alphaToCoverageEnable-08919
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_ALPHA_TO_COVERAGE_ENABLE_EXT dynamic state enabled, and alphaToCoverageEnable was VK_TRUE in the last call to vkCmdSetAlphaToCoverageEnableEXT, then the Fragment Output Interface **must** contain a variable for the alpha Component word in Location 0 at Index 0
- VUID-vkCmdDrawMultiIndexedEXT-alphaToCoverageEnable-08920
  If a shader object is bound to any graphics stage, and the most recent call to vkCmdSetAlphaToCoverageEnableEXT in the current command buffer set alphaToCoverageEnable to VK_TRUE, then the Fragment Output Interface **must** contain a variable for the alpha Component word in Location 0 at Index 0
- VUID-vkCmdDrawMultiIndexedEXT-None-07624
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_ALPHA_TO_COVERAGE_ENABLE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetAlphaToCoverageEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiIndexedEXT-None-07625
  If the alphaToOne feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the

VK_DYNAMIC_STATE_ALPHA_TO_ONE_ENABLE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetAlphaToOneEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMultiIndexedEXT-None-07626
  If the logicOp feature is enabled, a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_LOGIC_OP_ENABLE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetLogicOpEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiIndexedEXT-None-07627
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT dynamic state enabled then vkCmdSetColorBlendEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiIndexedEXT-None-08657
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and both the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE and there are color attachments bound, then vkCmdSetColorBlendEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiIndexedEXT-None-07628
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_EQUATION_EXT dynamic state enabled then vkCmdSetColorBlendEquationEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiIndexedEXT-None-08658
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetColorBlendEnableEXT for any attachment set that attachment's value in pColorBlendEnables to VK_TRUE, then vkCmdSetColorBlendEquationEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiIndexedEXT-None-07629
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_WRITE_MASK_EXT dynamic state enabled then vkCmdSetColorWriteMaskEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiIndexedEXT-None-08659
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and both the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE and there are color attachments bound, then vkCmdSetColorWriteMaskEXT **must** have been called and not subsequently invalidated in

the current command buffer prior to this drawing command

- VUID-vkCmdDrawMultiIndexedEXT-None-07630
  If the `geometryStreams` feature is enabled, and a shader object is bound to the `VK_SHADER_STAGE_GEOMETRY_BIT` stage or a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_RASTERIZATION_STREAM_EXT` dynamic state enabled, then vkCmdSetRasterizationStreamEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMultiIndexedEXT-None-07631
  If the `VK_EXT_conservative_rasterization` extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_CONSERVATIVE_RASTERIZATION_MODE_EXT` dynamic state enabled, and the current value of `rasterizerDiscardEnable` is `VK_FALSE`, then vkCmdSetConservativeRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMultiIndexedEXT-None-07632
  If the `VK_EXT_conservative_rasterization` extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_EXTRA_PRIMITIVE_OVERESTIMATION_SIZE_EXT` dynamic state enabled, the current value of `rasterizerDiscardEnable` is `VK_FALSE`, and the current value of `conservativeRasterizationMode` is `VK_CONSERVATIVE_RASTERIZATION_MODE_OVERESTIMATE_EXT`, then vkCmdSetExtraPrimitiveOverestimationSizeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMultiIndexedEXT-None-07633
  If the `depthClipEnable` feature is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_DEPTH_CLIP_ENABLE_EXT` dynamic state, then vkCmdSetDepthClipEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMultiIndexedEXT-None-07634
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT` dynamic state enabled then vkCmdSetSampleLocationsEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMultiIndexedEXT-None-08664
  If the `VK_EXT_sample_locations` extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE`, then vkCmdSetSampleLocationsEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMultiIndexedEXT-None-07635
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_COLOR_BLEND_ADVANCED_EXT` dynamic state enabled then vkCmdSetColorBlendAdvancedEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMultiIndexedEXT-rasterizerDiscardEnable-09416

If the VK_EXT_blend_operation_advanced extension is enabled, and a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then at least one of vkCmdSetColorBlendEquationEXT and vkCmdSetColorBlendAdvancedEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMultiIndexedEXT-None-07636
  If the VK_EXT_provoking_vertex extension is enabled, a shader object is bound to the VK_SHADER_STAGE_VERTEX_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_PROVOKING_VERTEX_MODE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetProvokingVertexModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiIndexedEXT-None-07637
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT dynamic state enabled then vkCmdSetLineRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiIndexedEXT-None-08666
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetPolygonModeEXT in the current command buffer set polygonMode to VK_POLYGON_MODE_LINE, then vkCmdSetLineRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiIndexedEXT-None-08667
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object is bound to the VK_SHADER_STAGE_VERTEX_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetPrimitiveTopologyEXT in the current command buffer set primitiveTopology to any line topology, then vkCmdSetLineRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiIndexedEXT-None-08668
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object that outputs line primitives is bound to the VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT or VK_SHADER_STAGE_GEOMETRY_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetLineRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMultiIndexedEXT-None-07638
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT dynamic state enabled then vkCmdSetLineStippleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiIndexedEXT-None-08669
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetPolygonModeEXT in the current command buffer set polygonMode to VK_POLYGON_MODE_LINE, then vkCmdSetLineStippleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiIndexedEXT-None-08670
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object is bound to the VK_SHADER_STAGE_VERTEX_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetPrimitiveTopologyEXT in the current command buffer set primitiveTopology to any line topology, then vkCmdSetLineStippleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiIndexedEXT-None-08671
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object that outputs line primitives is bound to the VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT or VK_SHADER_STAGE_GEOMETRY_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetLineStippleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiIndexedEXT-None-07849
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_KHR dynamic state enabled then vkCmdSetLineStippleKHR **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiIndexedEXT-None-08672
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetLineStippleEnableEXT in the current command buffer set stippledLineEnable to VK_TRUE, then vkCmdSetLineStippleEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiIndexedEXT-None-07639
  If the depthClipControl feature is enabled, and a shader object is bound to any graphics

stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_CLIP_NEGATIVE_ONE_TO_ONE_EXT dynamic state enabled, then vkCmdSetDepthClipNegativeOneToOneEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMultiIndexedEXT-None-07640
  If the VK_NV_clip_space_w_scaling extension is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_VIEWPORT_W_SCALING_ENABLE_NV dynamic state enabled, then vkCmdSetViewportWScalingEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiIndexedEXT-None-07641
  If the VK_NV_viewport_swizzle extension is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_VIEWPORT_SWIZZLE_NV dynamic state enabled, then vkCmdSetViewportSwizzleNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiIndexedEXT-None-07642
  If the VK_NV_fragment_coverage_to_color extension is enabled, a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_TO_COLOR_ENABLE_NV dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetCoverageToColorEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiIndexedEXT-None-07643
  If the VK_NV_fragment_coverage_to_color extension is enabled, a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_TO_COLOR_LOCATION_NV dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of coverageToColorEnable is VK_TRUE, then vkCmdSetCoverageToColorLocationNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiIndexedEXT-None-07644
  If the VK_NV_framebuffer_mixed_samples extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_MODULATION_MODE_NV dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetCoverageModulationModeNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiIndexedEXT-None-07645
  If the VK_NV_framebuffer_mixed_samples extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_MODULATION_TABLE_ENABLE_NV dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of coverageModulationMode is any value other than VK_COVERAGE_MODULATION_MODE_NONE_NV, then vkCmdSetCoverageModulationTableEnableNV **must** have been called and not

subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMultiIndexedEXT-None-07646
  If the VK_NV_framebuffer_mixed_samples extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_MODULATION_TABLE_NV dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of coverageModulationTableEnable is VK_TRUE, then vkCmdSetCoverageModulationTableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMultiIndexedEXT-None-07647
  If the shadingRateImage feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_SHADING_RATE_IMAGE_ENABLE_NV dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetShadingRateImageEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMultiIndexedEXT-pipelineFragmentShadingRate-09238
  If the pipelineFragmentShadingRate feature is enabled, a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetFragmentShadingRateKHR **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMultiIndexedEXT-None-07648
  If the representativeFragmentTest feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_REPRESENTATIVE_FRAGMENT_TEST_ENABLE_NV dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetRepresentativeFragmentTestEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMultiIndexedEXT-None-07649
  If the coverageReductionMode feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_REDUCTION_MODE_NV dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetCoverageReductionModeNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMultiIndexedEXT-pColorBlendEnables-07470
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT state enabled and the last call to vkCmdSetColorBlendEnableEXT set pColorBlendEnables for any attachment to VK_TRUE, then for those attachments in the subpass the corresponding image view's format features **must** contain VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT

- VUID-vkCmdDrawMultiIndexedEXT-rasterizationSamples-07471
  If the bound graphics pipeline state was created with the

VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, and the current subpass does not use any color and/or depth/stencil attachments, then the rasterizationSamples in the last call to vkCmdSetRasterizationSamplesEXT **must** follow the rules for a zero-attachment subpass

- VUID-vkCmdDrawMultiIndexedEXT-samples-07472
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_MASK_EXT state enabled and the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state disabled, then the samples parameter in the last call to vkCmdSetSampleMaskEXT **must** be greater or equal to the VkPipelineMultisampleStateCreateInfo::rasterizationSamples parameter used to create the bound graphics pipeline

- VUID-vkCmdDrawMultiIndexedEXT-samples-07473
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_MASK_EXT state and VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT states enabled, then the samples parameter in the last call to vkCmdSetSampleMaskEXT **must** be greater or equal to the rasterizationSamples parameter in the last call to vkCmdSetRasterizationSamplesEXT

- VUID-vkCmdDrawMultiIndexedEXT-rasterizationSamples-07474
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, and neither the VK_AMD_mixed_attachment_samples nor the VK_NV_framebuffer_mixed_samples extensions are enabled, then the rasterizationSamples in the last call to vkCmdSetRasterizationSamplesEXT **must** be the same as the current subpass color and/or depth/stencil attachments

- VUID-vkCmdDrawMultiIndexedEXT-None-09211
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, or a shader object is bound to any graphics stage, and the current render pass instance includes a VkMultisampledRenderToSingleSampledInfoEXT structure with multisampledRenderToSingleSampledEnable equal to VK_TRUE, then the rasterizationSamples in the last call to vkCmdSetRasterizationSamplesEXT **must** be the same as the rasterizationSamples member of that structure

- VUID-vkCmdDrawMultiIndexedEXT-firstAttachment-07476
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT dynamic state enabled then vkCmdSetColorBlendEnableEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the firstAttachment and attachmentCount parameters of vkCmdSetColorBlendEnableEXT calls **must** specify an enable for all active color attachments in the current subpass

- VUID-vkCmdDrawMultiIndexedEXT-rasterizerDiscardEnable-09417
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetColorBlendEnableEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the firstAttachment and attachmentCount parameters of

vkCmdSetColorBlendEnableEXT calls **must** specify an enable for all active color attachments in the current subpass

- VUID-vkCmdDrawMultiIndexedEXT-firstAttachment-07477
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_EQUATION_EXT dynamic state enabled then vkCmdSetColorBlendEquationEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the firstAttachment and attachmentCount parameters of vkCmdSetColorBlendEquationEXT calls **must** specify the blend equations for all active color attachments in the current subpass where blending is enabled

- VUID-vkCmdDrawMultiIndexedEXT-rasterizerDiscardEnable-09418
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and both the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE and there are color attachments bound, then vkCmdSetColorBlendEquationEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the firstAttachment and attachmentCount parameters of vkCmdSetColorBlendEquationEXT calls **must** specify the blend equations for all active color attachments in the current subpass where blending is enabled

- VUID-vkCmdDrawMultiIndexedEXT-firstAttachment-07478
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_WRITE_MASK_EXT dynamic state enabled then vkCmdSetColorWriteMaskEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the firstAttachment and attachmentCount parameters of vkCmdSetColorWriteMaskEXT calls **must** specify the color write mask for all active color attachments in the current subpass

- VUID-vkCmdDrawMultiIndexedEXT-rasterizerDiscardEnable-09419
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetColorWriteMaskEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the firstAttachment and attachmentCount parameters of vkCmdSetColorWriteMaskEXT calls **must** specify the color write mask for all active color attachments in the current subpass

- VUID-vkCmdDrawMultiIndexedEXT-firstAttachment-07479
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_ADVANCED_EXT dynamic state enabled then vkCmdSetColorBlendAdvancedEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the firstAttachment and attachmentCount parameters of vkCmdSetColorBlendAdvancedEXT calls **must** specify the advanced blend equations for all active color attachments in the current subpass where blending is enabled

- VUID-vkCmdDrawMultiIndexedEXT-advancedBlendMaxColorAttachments-07480
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_ADVANCED_EXT and VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT

dynamic states enabled and the last calls to vkCmdSetColorBlendEnableEXT and vkCmdSetColorBlendAdvancedEXT have enabled advanced blending, then the number of active color attachments in the current subpass **must** not exceed advancedBlendMaxColorAttachments

- VUID-vkCmdDrawMultiIndexedEXT-primitivesGeneratedQueryWithNonZeroStreams-07481
  If the primitivesGeneratedQueryWithNonZeroStreams feature is not enabled and the VK_QUERY_TYPE_PRIMITIVES_GENERATED_EXT query is active, and the bound graphics pipeline was created with VK_DYNAMIC_STATE_RASTERIZATION_STREAM_EXT state enabled, the last call to vkCmdSetRasterizationStreamEXT **must** have set the rasterizationStream to zero
- VUID-vkCmdDrawMultiIndexedEXT-sampleLocationsPerPixel-07482
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state enabled and the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state disabled, then the sampleLocationsPerPixel member of pSampleLocationsInfo in the last call to vkCmdSetSampleLocationsEXT **must** equal the rasterizationSamples member of the VkPipelineMultisampleStateCreateInfo structure the bound graphics pipeline has been created with
- VUID-vkCmdDrawMultiIndexedEXT-sampleLocationsPerPixel-07483
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state enabled and the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, then the sampleLocationsPerPixel member of pSampleLocationsInfo in the last call to vkCmdSetSampleLocationsEXT **must** equal the rasterizationSamples parameter of the last call to vkCmdSetRasterizationSamplesEXT
- VUID-vkCmdDrawMultiIndexedEXT-sampleLocationsEnable-07484
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, or the bound graphics pipeline was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, and sampleLocationsEnable was VK_TRUE in the last call to vkCmdSetSampleLocationsEnableEXT, and the current subpass has a depth/stencil attachment, then that attachment **must** have been created with the VK_IMAGE_CREATE_SAMPLE_LOCATIONS_COMPATIBLE_DEPTH_BIT_EXT bit set
- VUID-vkCmdDrawMultiIndexedEXT-sampleLocationsEnable-07485
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state enabled and the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, and if sampleLocationsEnable was VK_TRUE in the last call to vkCmdSetSampleLocationsEnableEXT, then the sampleLocationsInfo.sampleLocationGridSize.width in the last call to vkCmdSetSampleLocationsEXT **must** evenly divide VkMultisamplePropertiesEXT::sampleLocationGridSize.width as returned by vkGetPhysicalDeviceMultisamplePropertiesEXT with a samples parameter equaling rasterizationSamples
- VUID-vkCmdDrawMultiIndexedEXT-sampleLocationsEnable-07486
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, or the bound

graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state enabled and the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, and if sampleLocationsEnable was VK_TRUE in the last call to vkCmdSetSampleLocationsEnableEXT, then the sampleLocationsInfo.sampleLocationGridSize.height in the last call to vkCmdSetSampleLocationsEXT **must** evenly divide VkMultisamplePropertiesEXT::sampleLocationGridSize.height as returned by vkGetPhysicalDeviceMultisamplePropertiesEXT with a samples parameter equaling rasterizationSamples

- VUID-vkCmdDrawMultiIndexedEXT-sampleLocationsEnable-07487
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, and if sampleLocationsEnable was VK_TRUE in the last call to vkCmdSetSampleLocationsEnableEXT, the fragment shader code **must** not statically use the extended instruction InterpolateAtSample

- VUID-vkCmdDrawMultiIndexedEXT-sampleLocationsEnable-07936
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state disabled and the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, the sampleLocationsEnable member of a VkPipelineSampleLocationsStateCreateInfoEXT::sampleLocationsEnable in the bound graphics pipeline is VK_TRUE or VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, then, sampleLocationsInfo.sampleLocationGridSize.width **must** evenly divide VkMultisamplePropertiesEXT::sampleLocationGridSize.width as returned by vkGetPhysicalDeviceMultisamplePropertiesEXT with a samples parameter equaling the value of rasterizationSamples in the last call to vkCmdSetRasterizationSamplesEXT

- VUID-vkCmdDrawMultiIndexedEXT-sampleLocationsEnable-07937
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state disabled and the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, the sampleLocationsEnable member of a VkPipelineSampleLocationsStateCreateInfoEXT::sampleLocationsEnable in the bound graphics pipeline is VK_TRUE or VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, then, sampleLocationsInfo.sampleLocationGridSize.height **must** evenly divide VkMultisamplePropertiesEXT::sampleLocationGridSize.height as returned by vkGetPhysicalDeviceMultisamplePropertiesEXT with a samples parameter equaling the value of rasterizationSamples in the last call to vkCmdSetRasterizationSamplesEXT

- VUID-vkCmdDrawMultiIndexedEXT-sampleLocationsEnable-07938
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state disabled and the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, the sampleLocationsEnable member of a VkPipelineSampleLocationsStateCreateInfoEXT::sampleLocationsEnable in the bound graphics pipeline is VK_TRUE or VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, then, sampleLocationsInfo.sampleLocationsPerPixel **must** equal rasterizationSamples in the last call to vkCmdSetRasterizationSamplesEXT

- VUID-vkCmdDrawMultiIndexedEXT-coverageModulationTableEnable-07488
  If a shader object is bound to any graphics stage or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COVERAGE_MODULATION_TABLE_ENABLE_NV state enabled,

and the last call to vkCmdSetCoverageModulationTableEnableNV set coverageModulationTableEnable to VK_TRUE, then the coverageModulationTableCount parameter in the last call to vkCmdSetCoverageModulationTableNV **must** equal the current rasterizationSamples divided by the number of color samples in the current subpass

- VUID-vkCmdDrawMultiIndexedEXT-rasterizationSamples-07489
  If the VK_NV_framebuffer_mixed_samples extension is enabled, and if current subpass has a depth/stencil attachment and depth test, stencil test, or depth bounds test are enabled in the currently bound pipeline state, then the current rasterizationSamples **must** be the same as the sample count of the depth/stencil attachment

- VUID-vkCmdDrawMultiIndexedEXT-coverageToColorEnable-07490
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COVERAGE_TO_COLOR_ENABLE_NV state enabled and the last call to vkCmdSetCoverageToColorEnableNV set the coverageToColorEnable to VK_TRUE, then the current subpass **must** have a color attachment at the location selected by the last call to vkCmdSetCoverageToColorLocationNV coverageToColorLocation, with a VkFormat of VK_FORMAT_R8_UINT, VK_FORMAT_R8_SINT, VK_FORMAT_R16_UINT, VK_FORMAT_R16_SINT, VK_FORMAT_R32_UINT, or VK_FORMAT_R32_SINT

- VUID-vkCmdDrawMultiIndexedEXT-rasterizerDiscardEnable-09420
  If the VK_NV_fragment_coverage_to_color extension is enabled, and a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the last call to vkCmdSetCoverageToColorEnableNV set the coverageToColorEnable to VK_TRUE, then the current subpass **must** have a color attachment at the location selected by the last call to vkCmdSetCoverageToColorLocationNV coverageToColorLocation, with a VkFormat of VK_FORMAT_R8_UINT, VK_FORMAT_R8_SINT, VK_FORMAT_R16_UINT, VK_FORMAT_R16_SINT, VK_FORMAT_R32_UINT, or VK_FORMAT_R32_SINT

- VUID-vkCmdDrawMultiIndexedEXT-coverageReductionMode-07491
  If this VK_NV_coverage_reduction_mode extension is enabled, the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COVERAGE_TO_COLOR_ENABLE_NV and VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT states enabled, the current coverage reduction mode coverageReductionMode, then the current rasterizationSamples, and the sample counts for the color and depth/stencil attachments (if the subpass has them) **must** be a valid combination returned by vkGetPhysicalDeviceSupportedFramebufferMixedSamplesCombinationsNV

- VUID-vkCmdDrawMultiIndexedEXT-viewportCount-07492
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, but not the VK_DYNAMIC_STATE_VIEWPORT_SWIZZLE_NV dynamic state enabled, then the bound graphics pipeline **must** have been created with VkPipelineViewportSwizzleStateCreateInfoNV::viewportCount greater or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT

- VUID-vkCmdDrawMultiIndexedEXT-viewportCount-07493
  If the bound graphics pipeline state was created with the

VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT and VK_DYNAMIC_STATE_VIEWPORT_SWIZZLE_NV dynamic states enabled then the viewportCount parameter in the last call to vkCmdSetViewportSwizzleNV **must** be greater than or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT

- VUID-vkCmdDrawMultiIndexedEXT-viewportCount-09421
  If the VK_NV_viewport_swizzle extension is enabled, and a shader object is bound to any graphics stage, then the viewportCount parameter in the last call to vkCmdSetViewportSwizzleNV **must** be greater than or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT

- VUID-vkCmdDrawMultiIndexedEXT-rasterizationSamples-07494
  If the VK_NV_framebuffer_mixed_samples extension is enabled, and if the current subpass has any color attachments and rasterizationSamples of the last call to vkCmdSetRasterizationSamplesEXT is greater than the number of color samples, then the pipeline sampleShadingEnable **must** be VK_FALSE

- VUID-vkCmdDrawMultiIndexedEXT-stippledLineEnable-07495
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT or VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT dynamic states enabled, and if the current stippledLineEnable state is VK_TRUE and the current lineRasterizationMode state is VK_LINE_RASTERIZATION_MODE_RECTANGULAR_KHR, then the stippledRectangularLines feature **must** be enabled

- VUID-vkCmdDrawMultiIndexedEXT-stippledLineEnable-07496
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT or VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT dynamic states enabled, and if the current stippledLineEnable state is VK_TRUE and the current lineRasterizationMode state is VK_LINE_RASTERIZATION_MODE_BRESENHAM_KHR, then the stippledBresenhamLines feature **must** be enabled

- VUID-vkCmdDrawMultiIndexedEXT-stippledLineEnable-07497
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT or VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT dynamic states enabled, and if the current stippledLineEnable state is VK_TRUE and the current lineRasterizationMode state is VK_LINE_RASTERIZATION_MODE_RECTANGULAR_SMOOTH_KHR, then the stippledSmoothLines feature **must** be enabled

- VUID-vkCmdDrawMultiIndexedEXT-stippledLineEnable-07498
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT or VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT dynamic states enabled, and if the current stippledLineEnable state is VK_TRUE and the current lineRasterizationMode state is VK_LINE_RASTERIZATION_MODE_DEFAULT_KHR, then the stippledRectangularLines feature **must** be enabled and VkPhysicalDeviceLimits::strictLines **must** be VK_TRUE

- VUID-vkCmdDrawMultiIndexedEXT-conservativePointAndLineRasterization-07499
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_CONSERVATIVE_RASTERIZATION_MODE_EXT dynamic state enabled,

conservativePointAndLineRasterization is not supported, and the effective primitive topology output by the last pre-rasterization shader stage is a line or point, then the conservativeRasterizationMode set by the last call to vkCmdSetConservativeRasterizationModeEXT **must** be VK_CONSERVATIVE_RASTERIZATION_MODE_DISABLED_EXT

- VUID-vkCmdDrawMultiIndexedEXT-stage-07073
  If the currently bound pipeline was created with the VkPipelineShaderStageCreateInfo::stage member of an element of VkGraphicsPipelineCreateInfo::pStages set to VK_SHADER_STAGE_VERTEX_BIT, VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT, VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT or VK_SHADER_STAGE_GEOMETRY_BIT, then Mesh Shader Queries **must** not be active
- VUID-vkCmdDrawMultiIndexedEXT-None-08877
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_ATTACHMENT_FEEDBACK_LOOP_ENABLE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetAttachmentFeedbackLoopEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiIndexedEXT-None-07850
  If dynamic state was inherited from VkCommandBufferInheritanceViewportScissorInfoNV, it **must** be set in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiIndexedEXT-None-08684
  If there is no bound graphics pipeline, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_VERTEX_BIT
- VUID-vkCmdDrawMultiIndexedEXT-None-08685
  If there is no bound graphics pipeline, and the tessellationShader feature is enabled, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT
- VUID-vkCmdDrawMultiIndexedEXT-None-08686
  If there is no bound graphics pipeline, and the tessellationShader feature is enabled, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT
- VUID-vkCmdDrawMultiIndexedEXT-None-08687
  If there is no bound graphics pipeline, and the geometryShader feature is enabled, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_GEOMETRY_BIT
- VUID-vkCmdDrawMultiIndexedEXT-None-08688
  If there is no bound graphics pipeline, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_FRAGMENT_BIT
- VUID-vkCmdDrawMultiIndexedEXT-None-08689
  If there is no bound graphics pipeline, and the taskShader feature is enabled, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_TASK_BIT_EXT

- VUID-vkCmdDrawMultiIndexedEXT-None-08690
  If there is no bound graphics pipeline, and the `meshShader` feature is enabled, `vkCmdBindShadersEXT` **must** have been called in the current command buffer with `pStages` with an element of `VK_SHADER_STAGE_MESH_BIT_EXT`
- VUID-vkCmdDrawMultiIndexedEXT-None-08693
  If there is no bound graphics pipeline, and at least one of the `taskShader` and `meshShader` features is enabled, one of the `VK_SHADER_STAGE_VERTEX_BIT` or `VK_SHADER_STAGE_MESH_BIT_EXT` stages **must** have a valid `VkShaderEXT` bound, and the other **must** have no `VkShaderEXT` bound
- VUID-vkCmdDrawMultiIndexedEXT-None-08694
  If there is no bound graphics pipeline, and both the `taskShader` and `meshShader` features are enabled, and a valid `VkShaderEXT` is bound the to the `VK_SHADER_STAGE_MESH_BIT_EXT` stage, and that `VkShaderEXT` was created without the `VK_SHADER_CREATE_NO_TASK_SHADER_BIT_EXT` flag, a valid `VkShaderEXT` **must** be bound to the `VK_SHADER_STAGE_TASK_BIT_EXT` stage
- VUID-vkCmdDrawMultiIndexedEXT-None-08695
  If there is no bound graphics pipeline, and both the `taskShader` and `meshShader` features are enabled, and a valid `VkShaderEXT` is bound the to the `VK_SHADER_STAGE_MESH_BIT_EXT` stage, and that `VkShaderEXT` was created with the `VK_SHADER_CREATE_NO_TASK_SHADER_BIT_EXT` flag, there **must** be no `VkShaderEXT` bound to the `VK_SHADER_STAGE_TASK_BIT_EXT` stage
- VUID-vkCmdDrawMultiIndexedEXT-None-08696
  If there is no bound graphics pipeline, and a valid `VkShaderEXT` is bound to the `VK_SHADER_STAGE_VERTEX_BIT` stage, there **must** be no `VkShaderEXT` bound to either the `VK_SHADER_STAGE_TASK_BIT_EXT` stage or the `VK_SHADER_STAGE_MESH_BIT_EXT` stage
- VUID-vkCmdDrawMultiIndexedEXT-None-08698
  If any graphics shader is bound which was created with the `VK_SHADER_CREATE_LINK_STAGE_BIT_EXT` flag, then all shaders created with the `VK_SHADER_CREATE_LINK_STAGE_BIT_EXT` flag in the same vkCreateShadersEXT call **must** also be bound
- VUID-vkCmdDrawMultiIndexedEXT-None-08699
  If any graphics shader is bound which was created with the `VK_SHADER_CREATE_LINK_STAGE_BIT_EXT` flag, any stages in between stages whose shaders which did not create a shader with the `VK_SHADER_CREATE_LINK_STAGE_BIT_EXT` flag as part of the same vkCreateShadersEXT call **must** not have any `VkShaderEXT` bound
- VUID-vkCmdDrawMultiIndexedEXT-None-08878
  All bound graphics shader objects **must** have been created with identical or identically defined push constant ranges
- VUID-vkCmdDrawMultiIndexedEXT-None-08879
  All bound graphics shader objects **must** have been created with identical or identically defined arrays of descriptor set layouts
- VUID-vkCmdDrawMultiIndexedEXT-colorAttachmentCount-09372
  If the current render pass instance was begun with vkCmdBeginRenderingKHR and a VkRenderingInfo::`colorAttachmentCount` equal to `1`, a color attachment with a resolve mode of `VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID`, and a fragment shader is

bound, it **must** not declare the `DepthReplacing` or `StencilRefReplacingEXT` execution modes

- VUID-vkCmdDrawMultiIndexedEXT-pDynamicStates-08715
  If the bound graphics pipeline state includes a fragment shader stage, was created with `VK_DYNAMIC_STATE_DEPTH_WRITE_ENABLE` set in VkPipelineDynamicStateCreateInfo::`pDynamicStates`, and the fragment shader declares the `EarlyFragmentTests` execution mode and uses `OpDepthAttachmentReadEXT`, the `depthWriteEnable` parameter in the last call to vkCmdSetDepthWriteEnableEXT **must** be `VK_FALSE`

- VUID-vkCmdDrawMultiIndexedEXT-pDynamicStates-08716
  If the bound graphics pipeline state includes a fragment shader stage, was created with `VK_DYNAMIC_STATE_STENCIL_WRITE_MASK` set in VkPipelineDynamicStateCreateInfo::`pDynamicStates`, and the fragment shader declares the `EarlyFragmentTests` execution mode and uses `OpStencilAttachmentReadEXT`, the `writeMask` parameter in the last call to vkCmdSetStencilWriteMask **must** be `0`

- VUID-vkCmdDrawMultiIndexedEXT-None-09116
  If a shader object is bound to any graphics stage or the currently bound graphics pipeline was created with `VK_DYNAMIC_STATE_COLOR_WRITE_MASK_EXT`, and the format of any color attachment is `VK_FORMAT_E5B9G9R9_UFLOAT_PACK32`, the corresponding element of the `pColorWriteMasks` parameter of vkCmdSetColorWriteMaskEXT **must** either include all of `VK_COLOR_COMPONENT_R_BIT`, `VK_COLOR_COMPONENT_G_BIT`, and `VK_COLOR_COMPONENT_B_BIT`, or none of them

- VUID-vkCmdDrawMultiIndexedEXT-maxFragmentDualSrcAttachments-09239
  If blending is enabled for any attachment where either the source or destination blend factors for that attachment use the secondary color input, the maximum value of `Location` for any output attachment statically used in the `Fragment Execution Model` executed by this command **must** be less than `maxFragmentDualSrcAttachments`

- VUID-vkCmdDrawMultiIndexedEXT-None-09548
  If the current render pass was begun with vkCmdBeginRenderingKHR, and there is no shader object bound to any graphics stage, the value of each element of VkRenderingAttachmentLocationInfoKHR::`pColorAttachmentLocations` set by vkCmdSetRenderingAttachmentLocationsKHR **must** match the value set for the corresponding element in the currently bound pipeline

- VUID-vkCmdDrawMultiIndexedEXT-None-09549
  If the current render pass was begun with vkCmdBeginRenderingKHR, and there is no shader object bound to any graphics stage, input attachment index mappings in the currently bound pipeline **must** match those set for the current render pass instance via VkRenderingInputAttachmentIndexInfoKHR

- VUID-vkCmdDrawMultiIndexedEXT-None-09642
  If the current render pass was begun with vkCmdBeginRenderingKHR with the `VK_RENDERING_ENABLE_LEGACY_DITHERING_BIT_EXT` flag, the bound graphics pipeline **must** have been created with `VK_PIPELINE_CREATE_2_ENABLE_LEGACY_DITHERING_BIT_EXT`

- VUID-vkCmdDrawMultiIndexedEXT-None-09643
  If the bound graphics pipeline was created with `VK_PIPELINE_CREATE_2_ENABLE_LEGACY_DITHERING_BIT_EXT`, the current render pass **must** have begun with vkCmdBeginRenderingKHR with the `VK_RENDERING_ENABLE_LEGACY_DITHERING_BIT_EXT` flag

- VUID-vkCmdDrawMultiIndexedEXT-commandBuffer-02712
  If `commandBuffer` is a protected command buffer and `protectedNoFault` is not supported, any resource written to by the `VkPipeline` object bound to the pipeline bind point used by this command **must** not be an unprotected resource
- VUID-vkCmdDrawMultiIndexedEXT-commandBuffer-02713
  If `commandBuffer` is a protected command buffer and `protectedNoFault` is not supported, pipeline stages other than the framebuffer-space and compute stages in the `VkPipeline` object bound to the pipeline bind point used by this command **must** not write to any resource
- VUID-vkCmdDrawMultiIndexedEXT-commandBuffer-04617
  If any of the shader stages of the `VkPipeline` bound to the pipeline bind point used by this command uses the `RayQueryKHR` capability, then `commandBuffer` **must** not be a protected command buffer
- VUID-vkCmdDrawMultiIndexedEXT-None-04007
  All vertex input bindings accessed via vertex input variables declared in the vertex shader entry point's interface **must** have either valid or VK_NULL_HANDLE buffers bound
- VUID-vkCmdDrawMultiIndexedEXT-None-04008
  If the `nullDescriptor` feature is not enabled, all vertex input bindings accessed via vertex input variables declared in the vertex shader entry point's interface **must** not be VK_NULL_HANDLE
- VUID-vkCmdDrawMultiIndexedEXT-None-02721
  If `robustBufferAccess` is not enabled, and that pipeline was created without enabling `VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_EXT` for `vertexInputs`, then for a given vertex buffer binding, any attribute data fetched **must** be entirely contained within the corresponding vertex buffer binding, as described in Vertex Input Description
- VUID-vkCmdDrawMultiIndexedEXT-None-07842
  If there is a shader object bound to the `VK_SHADER_STAGE_VERTEX_BIT` stage or the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_PRIMITIVE_TOPOLOGY` dynamic state enabled then vkCmdSetPrimitiveTopologyEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMultiIndexedEXT-dynamicPrimitiveTopologyUnrestricted-07500
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_PRIMITIVE_TOPOLOGY` dynamic state enabled and the `dynamicPrimitiveTopologyUnrestricted` is `VK_FALSE`, then the `primitiveTopology` parameter of `vkCmdSetPrimitiveTopologyEXT` **must** be of the same topology class as the pipeline VkPipelineInputAssemblyStateCreateInfo::`topology` state
- VUID-vkCmdDrawMultiIndexedEXT-pStrides-04913
  If the bound graphics pipeline was created with the `VK_DYNAMIC_STATE_VERTEX_INPUT_BINDING_STRIDE_EXT` dynamic state enabled, but without the `VK_DYNAMIC_STATE_VERTEX_INPUT_EXT` dynamic state enabled, then vkCmdBindVertexBuffers2EXT **must** have been called and not subsequently invalidated in the current command buffer prior to this draw command, and the `pStrides` parameter

of vkCmdBindVertexBuffers2EXT **must** not be NULL

- VUID-vkCmdDrawMultiIndexedEXT-None-04914
  If there is a shader object bound to the VK_SHADER_STAGE_VERTEX_BIT stage or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VERTEX_INPUT_EXT dynamic state enabled then vkCmdSetVertexInputEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this draw command

- VUID-vkCmdDrawMultiIndexedEXT-Input-07939
  If there is a shader object bound to the VK_SHADER_STAGE_VERTEX_BIT stage or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VERTEX_INPUT_EXT dynamic state enabled then all variables with the Input storage class decorated with Location in the Vertex Execution Model OpEntryPoint **must** contain a location in VkVertexInputAttributeDescription2EXT::location

- VUID-vkCmdDrawMultiIndexedEXT-Input-08734
  If there is a shader object bound to the VK_SHADER_STAGE_VERTEX_BIT stage or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VERTEX_INPUT_EXT dynamic state enabled and either legacyVertexAttributes is not enabled or the SPIR-V Type associated with a given Input variable of the corresponding Location in the Vertex Execution Model OpEntryPoint is 64-bit, then the numeric type associated with all Input variables of the corresponding Location in the Vertex Execution Model OpEntryPoint **must** be the same as VkVertexInputAttributeDescription2EXT::format

- VUID-vkCmdDrawMultiIndexedEXT-format-08936
  If there is a shader object bound to the VK_SHADER_STAGE_VERTEX_BIT stage or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VERTEX_INPUT_EXT dynamic state enabled and VkVertexInputAttributeDescription2EXT::format has a 64-bit component, then the scalar width associated with all Input variables of the corresponding Location in the Vertex Execution Model OpEntryPoint **must** be 64-bit

- VUID-vkCmdDrawMultiIndexedEXT-format-08937
  If there is a shader object bound to the VK_SHADER_STAGE_VERTEX_BIT stage or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VERTEX_INPUT_EXT dynamic state enabled and the scalar width associated with a Location decorated Input variable in the Vertex Execution Model OpEntryPoint is 64-bit, then the corresponding VkVertexInputAttributeDescription2EXT::format **must** have a 64-bit component

- VUID-vkCmdDrawMultiIndexedEXT-None-09203
  If there is a shader object bound to the VK_SHADER_STAGE_VERTEX_BIT stage or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VERTEX_INPUT_EXT dynamic state enabled and VkVertexInputAttributeDescription2EXT::format has a 64-bit component, then all Input variables at the corresponding Location in the Vertex Execution Model OpEntryPoint **must** not use components that are not present in the format

- VUID-vkCmdDrawMultiIndexedEXT-None-04875
  If there is a shader object bound to the VK_SHADER_STAGE_VERTEX_BIT stage and the most recent call to vkCmdSetPrimitiveTopologyEXT in the current command buffer set primitiveTopology to VK_PRIMITIVE_TOPOLOGY_PATCH_LIST, or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_PATCH_CONTROL_POINTS_EXT dynamic state enabled then vkCmdSetPatchControlPointsEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMultiIndexedEXT-None-04879
  If there is a shader object bound to the VK_SHADER_STAGE_VERTEX_BIT stage or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_PRIMITIVE_RESTART_ENABLE dynamic state enabled then vkCmdSetPrimitiveRestartEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMultiIndexedEXT-None-09637
  If the primitiveTopologyListRestart feature is not enabled, the topology is VK_PRIMITIVE_TOPOLOGY_POINT_LIST, VK_PRIMITIVE_TOPOLOGY_LINE_LIST, VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST, VK_PRIMITIVE_TOPOLOGY_LINE_LIST_WITH_ADJACENCY, or VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST_WITH_ADJACENCY, there is a shader object bound to the VK_SHADER_STAGE_VERTEX_BIT stage or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_PRIMITIVE_RESTART_ENABLE dynamic state enabled then vkCmdSetPrimitiveRestartEnableEXT **must** be VK_FALSE

- VUID-vkCmdDrawMultiIndexedEXT-stage-06481
  The bound graphics pipeline **must** not have been created with the VkPipelineShaderStageCreateInfo::stage member of an element of VkGraphicsPipelineCreateInfo::pStages set to VK_SHADER_STAGE_TASK_BIT_EXT or VK_SHADER_STAGE_MESH_BIT_EXT

- VUID-vkCmdDrawMultiIndexedEXT-None-08885
  There **must** be no shader object bound to either of the VK_SHADER_STAGE_TASK_BIT_EXT or VK_SHADER_STAGE_MESH_BIT_EXT stages

- VUID-vkCmdDrawMultiIndexedEXT-None-07312
  If maintenance6 is not enabled, a valid index buffer **must** be bound

- VUID-vkCmdDrawMultiIndexedEXT-pNext-09461
  If the bound graphics pipeline state was created with VkPipelineVertexInputDivisorStateCreateInfoKHR in the pNext chain of VkGraphicsPipelineCreateInfo::pVertexInputState, any member of VkPipelineVertexInputDivisorStateCreateInfoKHR::pVertexBindingDivisors has a value other than 1 in divisor, and VkPhysicalDeviceVertexAttributeDivisorPropertiesKHR::supportsNonZeroFirstInstance is VK_FALSE, then firstInstance **must** be 0

- VUID-vkCmdDrawMultiIndexedEXT-None-09462
  If shader objects are used for drawing or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VERTEX_INPUT_EXT dynamic state enabled, any member of the pVertexBindingDescriptions parameter to the vkCmdSetVertexInputEXT call that sets this dynamic state has a value other than 1 in divisor, and VkPhysicalDeviceVertexAttributeDivisorPropertiesKHR::supportsNonZeroFirstInstance is VK_FALSE, then firstInstance **must** be 0

- VUID-vkCmdDrawMultiIndexedEXT-robustBufferAccess2-08798
  If robustBufferAccess2 is not enabled, (indexSize × (firstIndex + indexCount) + offset) **must** be less than or equal to the size of the bound index buffer, with indexSize being based on the type specified by indexType, where the index buffer, indexType, and offset are specified via vkCmdBindIndexBuffer or vkCmdBindIndexBuffer2KHR. If vkCmdBindIndexBuffer2KHR is used to bind the index buffer, the size of the bound index

buffer is vkCmdBindIndexBuffer2KHR::size

- VUID-vkCmdDrawMultiIndexedEXT-None-04937
  The multiDraw feature **must** be enabled
- VUID-vkCmdDrawMultiIndexedEXT-drawCount-04939
  drawCount **must** be less than VkPhysicalDeviceMultiDrawPropertiesEXT::maxMultiDrawCount
- VUID-vkCmdDrawMultiIndexedEXT-drawCount-04940
  If drawCount is greater than zero, pIndexInfo **must** be a valid pointer to memory containing one or more valid instances of VkMultiDrawIndexedInfoEXT structures
- VUID-vkCmdDrawMultiIndexedEXT-drawCount-09629
  If drawCount is greater than 1, stride **must** be a multiple of 4 and **must** be greater than or equal to sizeof(VkMultiDrawIndexedInfoEXT)

## Valid Usage (Implicit)

- VUID-vkCmdDrawMultiIndexedEXT-commandBuffer-parameter
  commandBuffer **must** be a valid VkCommandBuffer handle
- VUID-vkCmdDrawMultiIndexedEXT-pVertexOffset-parameter
  If pVertexOffset is not NULL, pVertexOffset **must** be a valid pointer to a valid int32_t value
- VUID-vkCmdDrawMultiIndexedEXT-commandBuffer-recording
  commandBuffer **must** be in the recording state
- VUID-vkCmdDrawMultiIndexedEXT-commandBuffer-cmdpool
  The VkCommandPool that commandBuffer was allocated from **must** support graphics operations
- VUID-vkCmdDrawMultiIndexedEXT-renderpass
  This command **must** only be called inside of a render pass instance
- VUID-vkCmdDrawMultiIndexedEXT-videocoding
  This command **must** only be called outside of a video coding scope

## Host Synchronization

- Host access to commandBuffer **must** be externally synchronized
- Host access to the VkCommandPool that commandBuffer was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Inside | Outside | Graphics | Action |

The VkMultiDrawInfoEXT structure is defined as:

```
// Provided by VK_EXT_multi_draw
typedef struct VkMultiDrawInfoEXT {
    uint32_t    firstVertex;
    uint32_t    vertexCount;
} VkMultiDrawInfoEXT;
```

- firstVertex is the first vertex to draw.
- vertexCount is the number of vertices to draw.

The members of VkMultiDrawInfoEXT have the same meaning as the firstVertex and vertexCount parameters in vkCmdDraw.

The VkMultiDrawIndexedInfoEXT structure is defined as:

```
// Provided by VK_EXT_multi_draw
typedef struct VkMultiDrawIndexedInfoEXT {
    uint32_t    firstIndex;
    uint32_t    indexCount;
    int32_t     vertexOffset;
} VkMultiDrawIndexedInfoEXT;
```

- firstIndex is the first index to draw.
- indexCount is the number of vertices to draw.
- vertexOffset is the value added to the vertex index before indexing into the vertex buffer for indexed multidraws.

The firstIndex, indexCount, and vertexOffset members of VkMultiDrawIndexedInfoEXT have the same meaning as the firstIndex, indexCount, and vertexOffset parameters, respectively, of vkCmdDrawIndexed.

To record a non-indexed indirect drawing command, call:

```
// Provided by VK_VERSION_1_0
void vkCmdDrawIndirect(
    VkCommandBuffer                             commandBuffer,
    VkBuffer                                    buffer,
    VkDeviceSize                                offset,
    uint32_t                                    drawCount,
    uint32_t                                    stride);
```

- commandBuffer is the command buffer into which the command is recorded.
- buffer is the buffer containing draw parameters.
- offset is the byte offset into buffer where parameters begin.

- drawCount is the number of draws to execute, and **can** be zero.
- stride is the byte stride between successive sets of draw parameters.

vkCmdDrawIndirect behaves similarly to vkCmdDraw except that the parameters are read by the device from a buffer during execution. drawCount draws are executed by the command, with parameters taken from buffer starting at offset and increasing by stride bytes for each successive draw. The parameters of each draw are encoded in an array of VkDrawIndirectCommand structures. If drawCount is less than or equal to one, stride is ignored.

### Valid Usage

- VUID-vkCmdDrawIndirect-magFilter-04553
  If a VkSampler created with magFilter or minFilter equal to VK_FILTER_LINEAR, reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE, and compareEnable equal to VK_FALSE is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT
- VUID-vkCmdDrawIndirect-magFilter-09598
  If a VkSampler created with magFilter or minFilter equal to VK_FILTER_LINEAR and reductionMode equal to either VK_SAMPLER_REDUCTION_MODE_MIN or VK_SAMPLER_REDUCTION_MODE_MAX is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT
- VUID-vkCmdDrawIndirect-mipmapMode-04770
  If a VkSampler created with mipmapMode equal to VK_SAMPLER_MIPMAP_MODE_LINEAR, reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE, and compareEnable equal to VK_FALSE is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT
- VUID-vkCmdDrawIndirect-mipmapMode-09599
  If a VkSampler created with mipmapMode equal to VK_SAMPLER_MIPMAP_MODE_LINEAR and reductionMode equal to either VK_SAMPLER_REDUCTION_MODE_MIN or VK_SAMPLER_REDUCTION_MODE_MAX is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT
- VUID-vkCmdDrawIndirect-unnormalizedCoordinates-09635
  If a VkSampler created with unnormalizedCoordinates equal to VK_TRUE is used to sample a VkImageView as a result of this command, then the image view's levelCount and layerCount **must** be 1
- VUID-vkCmdDrawIndirect-unnormalizedCoordinates-09636
  If a VkSampler created with unnormalizedCoordinates equal to VK_TRUE is used to sample a VkImageView as a result of this command, then the image view's viewType **must** be VK_IMAGE_VIEW_TYPE_1D or VK_IMAGE_VIEW_TYPE_2D
- VUID-vkCmdDrawIndirect-None-06479
  If a VkImageView is sampled with depth comparison, the image view's format features

**must** contain VK_FORMAT_FEATURE_2_SAMPLED_IMAGE_DEPTH_COMPARISON_BIT

- VUID-vkCmdDrawIndirect-None-02691
  If a VkImageView is accessed using atomic operations as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_STORAGE_IMAGE_ATOMIC_BIT
- VUID-vkCmdDrawIndirect-None-07888
  If a VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER descriptor is accessed using atomic operations as a result of this command, then the storage texel buffer's format features **must** contain VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_ATOMIC_BIT
- VUID-vkCmdDrawIndirect-None-02692
  If a VkImageView is sampled with VK_FILTER_CUBIC_EXT as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_CUBIC_BIT_EXT
- VUID-vkCmdDrawIndirect-None-02693
  If the VK_EXT_filter_cubic extension is not enabled and any VkImageView is sampled with VK_FILTER_CUBIC_EXT as a result of this command, it **must** not have a VkImageViewType of VK_IMAGE_VIEW_TYPE_3D, VK_IMAGE_VIEW_TYPE_CUBE, or VK_IMAGE_VIEW_TYPE_CUBE_ARRAY
- VUID-vkCmdDrawIndirect-filterCubic-02694
  Any VkImageView being sampled with VK_FILTER_CUBIC_EXT as a result of this command **must** have a VkImageViewType and format that supports cubic filtering, as specified by VkFilterCubicImageViewImageFormatPropertiesEXT::filterCubic returned by vkGetPhysicalDeviceImageFormatProperties2
- VUID-vkCmdDrawIndirect-filterCubicMinmax-02695
  Any VkImageView being sampled with VK_FILTER_CUBIC_EXT with a reduction mode of either VK_SAMPLER_REDUCTION_MODE_MIN or VK_SAMPLER_REDUCTION_MODE_MAX as a result of this command **must** have a VkImageViewType and format that supports cubic filtering together with minmax filtering, as specified by VkFilterCubicImageViewImageFormatPropertiesEXT::filterCubicMinmax returned by vkGetPhysicalDeviceImageFormatProperties2
- VUID-vkCmdDrawIndirect-cubicRangeClamp-09212
  If the cubicRangeClamp feature is not enabled, then any VkImageView being sampled with VK_FILTER_CUBIC_EXT as a result of this command **must** not have a VkSamplerReductionModeCreateInfo::reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE_RANGECLAMP_QCOM
- VUID-vkCmdDrawIndirect-reductionMode-09213
  Any VkImageView being sampled with a VkSamplerReductionModeCreateInfo::reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE_RANGECLAMP_QCOM as a result of this command **must** sample with VK_FILTER_CUBIC_EXT
- VUID-vkCmdDrawIndirect-selectableCubicWeights-09214
  If the selectableCubicWeights feature is not enabled, then any VkImageView being sampled with VK_FILTER_CUBIC_EXT as a result of this command **must** have VkSamplerCubicWeightsCreateInfoQCOM::cubicWeights equal to VK_CUBIC_FILTER_WEIGHTS_CATMULL_ROM_QCOM
- VUID-vkCmdDrawIndirect-flags-02696

Any VkImage created with a VkImageCreateInfo::flags containing VK_IMAGE_CREATE_CORNER_SAMPLED_BIT_NV sampled as a result of this command **must** only be sampled using a VkSamplerAddressMode of VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_EDGE

- VUID-vkCmdDrawIndirect-OpTypeImage-07027
  For any VkImageView being written as a storage image where the image format field of the OpTypeImage is Unknown, the view's format features **must** contain VK_FORMAT_FEATURE_2_STORAGE_WRITE_WITHOUT_FORMAT_BIT
- VUID-vkCmdDrawIndirect-OpTypeImage-07028
  For any VkImageView being read as a storage image where the image format field of the OpTypeImage is Unknown, the view's format features **must** contain VK_FORMAT_FEATURE_2_STORAGE_READ_WITHOUT_FORMAT_BIT
- VUID-vkCmdDrawIndirect-OpTypeImage-07029
  For any VkBufferView being written as a storage texel buffer where the image format field of the OpTypeImage is Unknown, the view's buffer features **must** contain VK_FORMAT_FEATURE_2_STORAGE_WRITE_WITHOUT_FORMAT_BIT
- VUID-vkCmdDrawIndirect-OpTypeImage-07030
  Any VkBufferView being read as a storage texel buffer where the image format field of the OpTypeImage is Unknown then the view's buffer features **must** contain VK_FORMAT_FEATURE_2_STORAGE_READ_WITHOUT_FORMAT_BIT
- VUID-vkCmdDrawIndirect-None-08600
  For each set *n* that is statically used by a bound shader, a descriptor set **must** have been bound to *n* at the same pipeline bind point, with a VkPipelineLayout that is compatible for set *n*, with the VkPipelineLayout used to create the current VkPipeline or the VkDescriptorSetLayout array used to create the current VkShaderEXT , as described in Pipeline Layout Compatibility
- VUID-vkCmdDrawIndirect-None-08601
  For each push constant that is statically used by a bound shader, a push constant value **must** have been set for the same pipeline bind point, with a VkPipelineLayout that is compatible for push constants, with the VkPipelineLayout used to create the current VkPipeline or the VkDescriptorSetLayout array used to create the current VkShaderEXT , as described in Pipeline Layout Compatibility
- VUID-vkCmdDrawIndirect-None-10068
  For each array of resources that is used by a bound shader, the indices used to access members of the array **must** be less than the descriptor count for the identified binding in the descriptor sets used by this command
- VUID-vkCmdDrawIndirect-maintenance4-08602
  If the maintenance4 feature is not enabled, then for each push constant that is statically used by a bound shader, a push constant value **must** have been set for the same pipeline bind point, with a VkPipelineLayout that is compatible for push constants, with the VkPipelineLayout used to create the current VkPipeline or the VkDescriptorSetLayout and VkPushConstantRange arrays used to create the current VkShaderEXT , as described in Pipeline Layout Compatibility
- VUID-vkCmdDrawIndirect-None-08114
  Descriptors in each bound descriptor set, specified via vkCmdBindDescriptorSets, **must** be

valid as described by descriptor validity if they are statically used by the VkPipeline bound to the pipeline bind point used by this command and the bound VkPipeline was not created with VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT

- VUID-vkCmdDrawIndirect-None-08115
  If the descriptors used by the VkPipeline bound to the pipeline bind point were specified via vkCmdBindDescriptorSets, the bound VkPipeline **must** have been created without VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT
- VUID-vkCmdDrawIndirect-None-08116
  Descriptors in bound descriptor buffers, specified via vkCmdSetDescriptorBufferOffsetsEXT, **must** be valid if they are dynamically used by the VkPipeline bound to the pipeline bind point used by this command and the bound VkPipeline was created with VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT
- VUID-vkCmdDrawIndirect-None-08604
  Descriptors in bound descriptor buffers, specified via vkCmdSetDescriptorBufferOffsetsEXT, **must** be valid if they are dynamically used by any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command
- VUID-vkCmdDrawIndirect-None-08117
  If the descriptors used by the VkPipeline bound to the pipeline bind point were specified via vkCmdSetDescriptorBufferOffsetsEXT, the bound VkPipeline **must** have been created with VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT
- VUID-vkCmdDrawIndirect-None-08119
  If a descriptor is dynamically used with a VkPipeline created with VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT, the descriptor memory **must** be resident
- VUID-vkCmdDrawIndirect-None-08605
  If a descriptor is dynamically used with a VkShaderEXT created with a VkDescriptorSetLayout that was created with VK_DESCRIPTOR_SET_LAYOUT_CREATE_DESCRIPTOR_BUFFER_BIT_EXT, the descriptor memory **must** be resident
- VUID-vkCmdDrawIndirect-None-08606
  If the shaderObject feature is not enabled, a valid pipeline **must** be bound to the pipeline bind point used by this command
- VUID-vkCmdDrawIndirect-None-08608
  If a pipeline is bound to the pipeline bind point used by this command, there **must** not have been any calls to dynamic state setting commands for any state specified statically in the VkPipeline object bound to the pipeline bind point used by this command, since that pipeline was bound
- VUID-vkCmdDrawIndirect-None-08609
  If the VkPipeline object bound to the pipeline bind point used by this command or any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a VkSampler object that uses unnormalized coordinates, that sampler **must** not be used to sample from any VkImage with a VkImageView of the type VK_IMAGE_VIEW_TYPE_3D, VK_IMAGE_VIEW_TYPE_CUBE, VK_IMAGE_VIEW_TYPE_1D_ARRAY, VK_IMAGE_VIEW_TYPE_2D_ARRAY or VK_IMAGE_VIEW_TYPE_CUBE_ARRAY, in any shader stage

- VUID-vkCmdDrawIndirect-None-08610
  If the VkPipeline object bound to the pipeline bind point used by this command or any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a VkSampler object that uses unnormalized coordinates, that sampler **must** not be used with any of the SPIR-V `OpImageSample*` or `OpImageSparseSample*` instructions with `ImplicitLod`, `Dref` or `Proj` in their name, in any shader stage
- VUID-vkCmdDrawIndirect-None-08611
  If the VkPipeline object bound to the pipeline bind point used by this command or any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a VkSampler object that uses unnormalized coordinates, that sampler **must** not be used with any of the SPIR-V `OpImageSample*` or `OpImageSparseSample*` instructions that includes a LOD bias or any offset values, in any shader stage
- VUID-vkCmdDrawIndirect-None-08607
  If the `shaderObject` is enabled, either a valid pipeline **must** be bound to the pipeline bind point used by this command, or a valid combination of valid and VK_NULL_HANDLE shader objects **must** be bound to every supported shader stage corresponding to the pipeline bind point used by this command
- VUID-vkCmdDrawIndirect-uniformBuffers-06935
  If any stage of the VkPipeline object bound to the pipeline bind point used by this command accesses a uniform buffer, and that stage was created without enabling either `VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_EXT` or `VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_2_EXT` for `uniformBuffers`, and the `robustBufferAccess` feature is not enabled, that stage **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point
- VUID-vkCmdDrawIndirect-None-08612
  If the `robustBufferAccess` feature is not enabled, and any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a uniform buffer, it **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point
- VUID-vkCmdDrawIndirect-storageBuffers-06936
  If any stage of the VkPipeline object bound to the pipeline bind point used by this command accesses a storage buffer, and that stage was created without enabling either `VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_EXT` or `VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_2_EXT` for `storageBuffers`, and the `robustBufferAccess` feature is not enabled, that stage **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point
- VUID-vkCmdDrawIndirect-None-08613
  If the `robustBufferAccess` feature is not enabled, and any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a storage buffer, it **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point
- VUID-vkCmdDrawIndirect-commandBuffer-02707
  If `commandBuffer` is an unprotected command buffer and `protectedNoFault` is not supported,

any resource accessed by bound shaders **must** not be a protected resource

- VUID-vkCmdDrawIndirect-None-06550
  If a bound shader accesses a VkSampler or VkImageView object that enables sampler $Y'C_BC_R$ conversion, that object **must** only be used with `OpImageSample*` or `OpImageSparseSample*` instructions
- VUID-vkCmdDrawIndirect-ConstOffset-06551
  If a bound shader accesses a VkSampler or VkImageView object that enables sampler $Y'C_BC_R$ conversion, that object **must** not use the `ConstOffset` and `Offset` operands
- VUID-vkCmdDrawIndirect-viewType-07752
  If a VkImageView is accessed as a result of this command, then the image view's `viewType` **must** match the `Dim` operand of the `OpTypeImage` as described in Compatibility Between SPIR-V Image Dimensions and Vulkan ImageView Types
- VUID-vkCmdDrawIndirect-format-07753
  If a VkImageView is accessed as a result of this command, then the numeric type of the image view's `format` and the `Sampled Type` operand of the `OpTypeImage` **must** match
- VUID-vkCmdDrawIndirect-OpImageWrite-08795
  If a VkImageView created with a format other than `VK_FORMAT_A8_UNORM_KHR` is accessed using `OpImageWrite` as a result of this command, then the `Type` of the `Texel` operand of that instruction **must** have at least as many components as the image view's format
- VUID-vkCmdDrawIndirect-OpImageWrite-08796
  If a VkImageView created with the format `VK_FORMAT_A8_UNORM_KHR` is accessed using `OpImageWrite` as a result of this command, then the `Type` of the `Texel` operand of that instruction **must** have four components
- VUID-vkCmdDrawIndirect-OpImageWrite-04469
  If a VkBufferView is accessed using `OpImageWrite` as a result of this command, then the `Type` of the `Texel` operand of that instruction **must** have at least as many components as the buffer view's format
- VUID-vkCmdDrawIndirect-SampledType-04470
  If a VkImageView with a VkFormat that has a 64-bit component width is accessed as a result of this command, the `SampledType` of the `OpTypeImage` operand of that instruction **must** have a `Width` of 64
- VUID-vkCmdDrawIndirect-SampledType-04471
  If a VkImageView with a VkFormat that has a component width less than 64-bit is accessed as a result of this command, the `SampledType` of the `OpTypeImage` operand of that instruction **must** have a `Width` of 32
- VUID-vkCmdDrawIndirect-SampledType-04472
  If a VkBufferView with a VkFormat that has a 64-bit component width is accessed as a result of this command, the `SampledType` of the `OpTypeImage` operand of that instruction **must** have a `Width` of 64
- VUID-vkCmdDrawIndirect-SampledType-04473
  If a VkBufferView with a VkFormat that has a component width less than 64-bit is accessed as a result of this command, the `SampledType` of the `OpTypeImage` operand of that instruction **must** have a `Width` of 32

- VUID-vkCmdDrawIndirect-sparseImageInt64Atomics-04474
  If the `sparseImageInt64Atomics` feature is not enabled, VkImage objects created with the `VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT` flag **must** not be accessed by atomic instructions through an `OpTypeImage` with a `SampledType` with a `Width` of 64 by this command
- VUID-vkCmdDrawIndirect-sparseImageInt64Atomics-04475
  If the `sparseImageInt64Atomics` feature is not enabled, VkBuffer objects created with the `VK_BUFFER_CREATE_SPARSE_RESIDENCY_BIT` flag **must** not be accessed by atomic instructions through an `OpTypeImage` with a `SampledType` with a `Width` of 64 by this command
- VUID-vkCmdDrawIndirect-OpImageWeightedSampleQCOM-06971
  If `OpImageWeightedSampleQCOM` is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_2_WEIGHT_SAMPLED_IMAGE_BIT_QCOM`
- VUID-vkCmdDrawIndirect-OpImageWeightedSampleQCOM-06972
  If `OpImageWeightedSampleQCOM` uses a VkImageView as a sample weight image as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_2_WEIGHT_IMAGE_BIT_QCOM`
- VUID-vkCmdDrawIndirect-OpImageBoxFilterQCOM-06973
  If `OpImageBoxFilterQCOM` is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_2_BOX_FILTER_SAMPLED_BIT_QCOM`
- VUID-vkCmdDrawIndirect-OpImageBlockMatchSSDQCOM-06974
  If `OpImageBlockMatchSSDQCOM` is used to read from an VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM`
- VUID-vkCmdDrawIndirect-OpImageBlockMatchSADQCOM-06975
  If `OpImageBlockMatchSADQCOM` is used to read from an VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM`
- VUID-vkCmdDrawIndirect-OpImageBlockMatchSADQCOM-06976
  If `OpImageBlockMatchSADQCOM` or OpImageBlockMatchSSDQCOM is used to read from a reference image as result of this command, then the specified reference coordinates **must** not fail integer texel coordinate validation
- VUID-vkCmdDrawIndirect-OpImageWeightedSampleQCOM-06977
  If `OpImageWeightedSampleQCOM`, `OpImageBoxFilterQCOM`, `OpImageBlockMatchWindowSSDQCOM`, `OpImageBlockMatchWindowSADQCOM`, `OpImageBlockMatchGatherSSDQCOM`, `OpImageBlockMatchGatherSADQCOM`, `OpImageBlockMatchSSDQCOM`, or `OpImageBlockMatchSADQCOM` uses a VkSampler as a result of this command, then the sampler **must** have been created with `VK_SAMPLER_CREATE_IMAGE_PROCESSING_BIT_QCOM`
- VUID-vkCmdDrawIndirect-OpImageWeightedSampleQCOM-06978
  If any command other than `OpImageWeightedSampleQCOM`, `OpImageBoxFilterQCOM`, `OpImageBlockMatchWindowSSDQCOM`, `OpImageBlockMatchWindowSADQCOM`, `OpImageBlockMatchGatherSSDQCOM`, `OpImageBlockMatchGatherSADQCOM`, `OpImageBlockMatchSSDQCOM`, or `OpImageBlockMatchSADQCOM` uses a VkSampler as a result of this command, then the sampler **must** not have been created with

VK_SAMPLER_CREATE_IMAGE_PROCESSING_BIT_QCOM

- VUID-vkCmdDrawIndirect-OpImageBlockMatchWindow-09215
  If a OpImageBlockMatchWindow*QCOM or OpImageBlockMatchGather*QCOM instruction is used to read from an VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM
- VUID-vkCmdDrawIndirect-OpImageBlockMatchWindow-09216
  If a OpImageBlockMatchWindow*QCOM or OpImageBlockMatchGather*QCOM instruction is used to read from an VkImageView as a result of this command, then the image view's format **must** be a single-component format
- VUID-vkCmdDrawIndirect-OpImageBlockMatchWindow-09217
  If a OpImageBlockMatchWindow*QCOM or OpImageBlockMatchGather*QCOM read from a reference image as result of this command, then the specified reference coordinates **must** not fail integer texel coordinate validation
- VUID-vkCmdDrawIndirect-None-07288
  Any shader invocation executed by this command **must** terminate
- VUID-vkCmdDrawIndirect-None-09600
  If a descriptor with type equal to any of VK_DESCRIPTOR_TYPE_SAMPLE_WEIGHT_IMAGE_QCOM, VK_DESCRIPTOR_TYPE_BLOCK_MATCH_IMAGE_QCOM, VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE, VK_DESCRIPTOR_TYPE_STORAGE_IMAGE, or VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT is accessed as a result of this command, the image subresource identified by that descriptor **must** be in the image layout identified when the descriptor was written
- VUID-vkCmdDrawIndirect-renderPass-02684
  The current render pass **must** be compatible with the renderPass member of the VkGraphicsPipelineCreateInfo structure specified when creating the VkPipeline bound to VK_PIPELINE_BIND_POINT_GRAPHICS
- VUID-vkCmdDrawIndirect-subpass-02685
  The subpass index of the current render pass **must** be equal to the subpass member of the VkGraphicsPipelineCreateInfo structure specified when creating the VkPipeline bound to VK_PIPELINE_BIND_POINT_GRAPHICS
- VUID-vkCmdDrawIndirect-None-07748
  If any shader statically accesses an input attachment, a valid descriptor **must** be bound to the pipeline via a descriptor set
- VUID-vkCmdDrawIndirect-OpTypeImage-07468
  If any shader executed by this pipeline accesses an OpTypeImage variable with a Dim operand of SubpassData, it **must** be decorated with an InputAttachmentIndex that corresponds to a valid input attachment in the current subpass
- VUID-vkCmdDrawIndirect-None-07469
  Input attachment views accessed in a subpass **must** be created with the same VkFormat as the corresponding subpass definition, and be created with a VkImageView that is compatible with the attachment referenced by the subpass' pInputAttachments [InputAttachmentIndex] in the currently bound VkFramebuffer as specified by Fragment Input Attachment Compatibility
- VUID-vkCmdDrawIndirect-pDepthInputAttachmentIndex-09595

Input attachment views accessed in a dynamic render pass with a `InputAttachmentIndex` referenced by VkRenderingInputAttachmentIndexInfoKHR, or no `InputAttachmentIndex` if VkRenderingInputAttachmentIndexInfoKHR:`pDepthInputAttachmentIndex` or VkRenderingInputAttachmentIndexInfoKHR:`pStencilInputAttachmentIndex` are `NULL`, **must** be created with a VkImageView that is compatible with the corresponding color, depth, or stencil attachment in VkRenderingInfo

- VUID-vkCmdDrawIndirect-pDepthInputAttachmentIndex-09596
  Input attachment views accessed in a dynamic render pass via a shader object **must** have an `InputAttachmentIndex` if both VkRenderingInputAttachmentIndexInfoKHR:`pDepthInputAttachmentIndex` and VkRenderingInputAttachmentIndexInfoKHR:`pStencilInputAttachmentIndex` are non-`NULL`
- VUID-vkCmdDrawIndirect-InputAttachmentIndex-09597
  If an input attachment view accessed in a dynamic render pass via a shader object has an `InputAttachmentIndex`, the `InputAttachmentIndex` **must** match an index in VkRenderingInputAttachmentIndexInfoKHR
- VUID-vkCmdDrawIndirect-None-06537
  Memory backing image subresources used as attachments in the current render pass **must** not be written in any way other than as an attachment by this command
- VUID-vkCmdDrawIndirect-None-09000
  If a color attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass, it is not in the `VK_IMAGE_LAYOUT_ATTACHMENT_FEEDBACK_LOOP_OPTIMAL_EXT` image layout, and either:
  - the `VK_PIPELINE_CREATE_COLOR_ATTACHMENT_FEEDBACK_LOOP_BIT_EXT` is set on the currently bound pipeline or
  - the last call to vkCmdSetAttachmentFeedbackLoopEnableEXT included `VK_IMAGE_ASPECT_COLOR_BIT` and
    - there is no currently bound graphics pipeline or
    - the currently bound graphics pipeline was created with `VK_DYNAMIC_STATE_ATTACHMENT_FEEDBACK_LOOP_ENABLE_EXT`

  it **must** not be accessed in any way other than as an attachment by this command
- VUID-vkCmdDrawIndirect-None-09001
  If a depth attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass, it is not in the `VK_IMAGE_LAYOUT_ATTACHMENT_FEEDBACK_LOOP_OPTIMAL_EXT` image layout, and either:
  - the `VK_PIPELINE_CREATE_DEPTH_STENCIL_ATTACHMENT_FEEDBACK_LOOP_BIT_EXT` is set on the currently bound pipeline or
  - the last call to vkCmdSetAttachmentFeedbackLoopEnableEXT included `VK_IMAGE_ASPECT_DEPTH_BIT` and
    - there is no currently bound graphics pipeline or
    - the currently bound graphics pipeline was created with `VK_DYNAMIC_STATE_ATTACHMENT_FEEDBACK_LOOP_ENABLE_EXT`

it **must** not be accessed in any way other than as an attachment by this command

- VUID-vkCmdDrawIndirect-None-09002
  If a stencil attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass, it is not in the `VK_IMAGE_LAYOUT_ATTACHMENT_FEEDBACK_LOOP_OPTIMAL_EXT` image layout, and either:
  - the `VK_PIPELINE_CREATE_DEPTH_STENCIL_ATTACHMENT_FEEDBACK_LOOP_BIT_EXT` is set on the currently bound pipeline or
  - the last call to vkCmdSetAttachmentFeedbackLoopEnableEXT included `VK_IMAGE_ASPECT_STENCIL_BIT` and
    - there is no currently bound graphics pipeline or
    - the currently bound graphics pipeline was created with `VK_DYNAMIC_STATE_ATTACHMENT_FEEDBACK_LOOP_ENABLE_EXT`

  it **must** not be accessed in any way other than as an attachment by this command

- VUID-vkCmdDrawIndirect-None-09003
  If an attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass, it **must** not be accessed in any way other than as an attachment, storage image, or sampled image by this command
- VUID-vkCmdDrawIndirect-None-06539
  If any previously recorded command in the current subpass accessed an image subresource used as an attachment in this subpass in any way other than as an attachment, this command **must** not write to that image subresource as an attachment
- VUID-vkCmdDrawIndirect-None-06886
  If the current render pass instance uses a depth/stencil attachment with a read-only layout for the depth aspect, depth writes **must** be disabled
- VUID-vkCmdDrawIndirect-None-06887
  If the current render pass instance uses a depth/stencil attachment with a read-only layout for the stencil aspect, both front and back `writeMask` are not zero, and stencil test is enabled, all stencil ops **must** be `VK_STENCIL_OP_KEEP`
- VUID-vkCmdDrawIndirect-None-07831
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VIEWPORT` dynamic state enabled then vkCmdSetViewport **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirect-None-07832
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_SCISSOR` dynamic state enabled then vkCmdSetScissor **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirect-None-07833
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_LINE_WIDTH` dynamic state enabled then vkCmdSetLineWidth **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirect-None-08617

If a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetPolygonModeEXT in the current command buffer set polygonMode to VK_POLYGON_MODE_LINE, vkCmdSetLineWidth **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirect-None-08618
  If a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetPrimitiveTopologyEXT in the current command buffer set primitiveTopology to any line topology, vkCmdSetLineWidth **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirect-None-08619
  If a shader object that outputs line primitives is bound to the VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT or VK_SHADER_STAGE_GEOMETRY_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, vkCmdSetLineWidth **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirect-None-07834
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_BIAS dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of depthBiasEnable is VK_TRUE, then vkCmdSetDepthBounds or vkCmdSetDepthBias2EXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirect-None-07835
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_BLEND_CONSTANTS dynamic state enabled then vkCmdSetBlendConstants **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirect-None-08621
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetColorBlendEnableEXT in the current command buffer set any element of pColorBlendEnables to VK_TRUE, and the most recent call to vkCmdSetColorBlendEquationEXT in the current command buffer set the same element of pColorBlendEquations to a VkColorBlendEquationEXT structure with any VkBlendFactor member with a value of VK_BLEND_FACTOR_CONSTANT_COLOR, VK_BLEND_FACTOR_ONE_MINUS_CONSTANT_COLOR, VK_BLEND_FACTOR_CONSTANT_ALPHA, or VK_BLEND_FACTOR_ONE_MINUS_CONSTANT_ALPHA, vkCmdSetBlendConstants **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirect-None-07836
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_BOUNDS dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of depthBoundsTestEnable is VK_TRUE, then vkCmdSetDepthBounds **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirect-None-07837
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_STENCIL_COMPARE_MASK dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of stencilTestEnable is VK_TRUE, then vkCmdSetStencilCompareMask **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirect-None-07838
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_STENCIL_WRITE_MASK dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of stencilTestEnable is VK_TRUE, then vkCmdSetStencilWriteMask **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirect-None-07839
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_STENCIL_REFERENCE dynamic state enabled, the current value of and rasterizerDiscardEnable is VK_FALSE, the current value of stencilTestEnable is VK_TRUE, then vkCmdSetStencilReference **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirect-maxMultiviewInstanceIndex-02688
  If the draw is recorded in a render pass instance with multiview enabled, the maximum instance index **must** be less than or equal to VkPhysicalDeviceMultiviewProperties::maxMultiviewInstanceIndex
- VUID-vkCmdDrawIndirect-sampleLocationsEnable-02689
  If the bound graphics pipeline was created with VkPipelineSampleLocationsStateCreateInfoEXT::sampleLocationsEnable set to VK_TRUE and the current subpass has a depth/stencil attachment, then that attachment **must** have been created with the VK_IMAGE_CREATE_SAMPLE_LOCATIONS_COMPATIBLE_DEPTH_BIT_EXT bit set
- VUID-vkCmdDrawIndirect-None-06666
  If the VK_EXT_sample_locations extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of sampleLocationsEnable is VK_TRUE, then vkCmdSetSampleLocationsEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirect-None-07840

If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_CULL_MODE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetCullModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirect-None-07841
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_FRONT_FACE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetFrontFaceEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirect-None-07843
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_TEST_ENABLE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, vkCmdSetDepthTestEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirect-None-07844
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_WRITE_ENABLE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetDepthWriteEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirect-None-07845
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_COMPARE_OP dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of depthTestEnable is VK_TRUE, then vkCmdSetDepthCompareOpEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirect-None-07846
  If the depthBounds feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_BOUNDS_TEST_ENABLE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetDepthBoundsTestEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirect-None-07847
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_STENCIL_TEST_ENABLE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetStencilTestEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirect-None-07848
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which

was created with the VK_DYNAMIC_STATE_STENCIL_OP dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, the current value of stencilTestEnable is VK_TRUE, then vkCmdSetStencilOpEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirect-viewportCount-03417
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, but not the VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT dynamic state enabled, then vkCmdSetViewportWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the viewportCount parameter of vkCmdSetViewportWithCountEXT **must** match the VkPipelineViewportStateCreateInfo::scissorCount of the pipeline
- VUID-vkCmdDrawIndirect-scissorCount-03418
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT dynamic state enabled, but not the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, then vkCmdSetScissorWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the scissorCount parameter of vkCmdSetScissorWithCountEXT **must** match the VkPipelineViewportStateCreateInfo::viewportCount of the pipeline
- VUID-vkCmdDrawIndirect-viewportCount-03419
  If the bound graphics pipeline state was created with both the VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT and VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic states enabled then both vkCmdSetViewportWithCountEXT and vkCmdSetScissorWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the viewportCount parameter of vkCmdSetViewportWithCountEXT **must** match the scissorCount parameter of vkCmdSetScissorWithCountEXT
- VUID-vkCmdDrawIndirect-None-08635
  If a shader object is bound to any graphics stage, then both vkCmdSetViewportWithCountEXT and vkCmdSetScissorWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the viewportCount parameter of vkCmdSetViewportWithCountEXT **must** match the scissorCount parameter of vkCmdSetScissorWithCountEXT
- VUID-vkCmdDrawIndirect-viewportCount-04137
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, but not the VK_DYNAMIC_STATE_VIEWPORT_W_SCALING_NV dynamic state enabled, then the bound graphics pipeline **must** have been created with VkPipelineViewportWScalingStateCreateInfoNV::viewportCount greater or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawIndirect-viewportCount-04138
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT and VK_DYNAMIC_STATE_VIEWPORT_W_SCALING_NV dynamic states enabled then the viewportCount parameter in the last call to

vkCmdSetViewportWScalingNV **must** be greater than or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT

- VUID-vkCmdDrawIndirect-None-09232
  If the VK_NV_clip_space_w_scaling extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetViewportWScalingEnableNV in the current command buffer set viewportWScalingEnable to VK_TRUE, then vkCmdSetViewportWScalingNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirect-None-08636
  If the VK_NV_clip_space_w_scaling extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetViewportWScalingEnableNV in the current command buffer set viewportWScalingEnable to VK_TRUE, then the viewportCount parameter in the last call to vkCmdSetViewportWScalingNV **must** be greater than or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawIndirect-viewportCount-04139
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, but not the VK_DYNAMIC_STATE_VIEWPORT_SHADING_RATE_PALETTE_NV dynamic state enabled, then the bound graphics pipeline **must** have been created with VkPipelineViewportShadingRateImageStateCreateInfoNV::viewportCount greater or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawIndirect-viewportCount-04140
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT and VK_DYNAMIC_STATE_VIEWPORT_SHADING_RATE_PALETTE_NV dynamic states enabled then the viewportCount parameter in the last call to vkCmdSetViewportShadingRatePaletteNV **must** be greater than or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawIndirect-shadingRateImage-09233
  If the shadingRateImage feature is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetCoarseSampleOrderNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirect-shadingRateImage-09234
  If the shadingRateImage feature is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetShadingRateImageEnableNV in the current command buffer set shadingRateImageEnable to VK_TRUE, then vkCmdSetViewportShadingRatePaletteNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirect-None-08637
  If the shadingRateImage feature is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current

command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetShadingRateImageEnableNV in the current command buffer set shadingRateImageEnable to VK_TRUE, then the viewportCount parameter in the last call to vkCmdSetViewportShadingRatePaletteNV **must** be greater than or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT

- VUID-vkCmdDrawIndirect-VkPipelineVieportCreateInfo-04141
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled and a VkPipelineViewportSwizzleStateCreateInfoNV structure chained from VkPipelineViewportStateCreateInfo, then the bound graphics pipeline **must** have been created with VkPipelineViewportSwizzleStateCreateInfoNV::viewportCount greater or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawIndirect-VkPipelineVieportCreateInfo-04142
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled and a VkPipelineViewportExclusiveScissorStateCreateInfoNV structure chained from VkPipelineViewportStateCreateInfo, then the bound graphics pipeline **must** have been created with VkPipelineViewportExclusiveScissorStateCreateInfoNV::exclusiveScissorCount greater or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawIndirect-None-07878
  If the exclusiveScissor feature is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_EXCLUSIVE_SCISSOR_ENABLE_NV dynamic state enabled, then vkCmdSetExclusiveScissorEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirect-None-07879
  If the exclusiveScissor feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_EXCLUSIVE_SCISSOR_NV dynamic state enabled, and the most recent call to vkCmdSetExclusiveScissorEnableNV in the current command buffer set any element of pExclusiveScissorEnables to VK_TRUE, then vkCmdSetExclusiveScissorNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirect-None-04876
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_RASTERIZER_DISCARD_ENABLE dynamic state enabled, then vkCmdSetRasterizerDiscardEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirect-None-04877
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_BIAS_ENABLE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetDepthBiasEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirect-logicOp-04878
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_LOGIC_OP_EXT dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of logicOpEnable is VK_TRUE, then vkCmdSetLogicOpEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirect-primitiveFragmentShadingRateWithMultipleViewports-04552
  If the primitiveFragmentShadingRateWithMultipleViewports limit is not supported, the bound graphics pipeline was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, and any of the shader stages of the bound graphics pipeline write to the PrimitiveShadingRateKHR built-in, then vkCmdSetViewportWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the viewportCount parameter of vkCmdSetViewportWithCountEXT **must** be 1
- VUID-vkCmdDrawIndirect-primitiveFragmentShadingRateWithMultipleViewports-08642
  If the primitiveFragmentShadingRateWithMultipleViewports limit is not supported, and any shader object bound to a graphics stage writes to the PrimitiveShadingRateKHR built-in, then vkCmdSetViewportWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the viewportCount parameter of vkCmdSetViewportWithCountEXT **must** be 1
- VUID-vkCmdDrawIndirect-blendEnable-04727
  If rasterization is not disabled in the bound graphics pipeline, then for each color attachment in the subpass, if the corresponding image view's format features do not contain VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT, then the blendEnable member of the corresponding element of the pAttachments member of pColorBlendState **must** be VK_FALSE
- VUID-vkCmdDrawIndirect-None-08643
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then for each color attachment in the render pass, if the corresponding image view's format features do not contain VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT, then the corresponding member of pColorBlendEnables in the most recent call to vkCmdSetColorBlendEnableEXT in the current command buffer that affected that attachment index **must** have been VK_FALSE
- VUID-vkCmdDrawIndirect-multisampledRenderToSingleSampled-07284
  If rasterization is not disabled in the bound graphics pipeline, and none of the following is enabled:
  - the VK_AMD_mixed_attachment_samples extension
  - the VK_NV_framebuffer_mixed_samples extension
  - the multisampledRenderToSingleSampled feature

  then rasterizationSamples for the currently bound graphics pipeline **must** be the same as the current subpass color and/or depth/stencil attachments
- VUID-vkCmdDrawIndirect-None-08644
  If a shader object is bound to any graphics stage, and the most recent call to

vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and none of the following is enabled:

- the VK_AMD_mixed_attachment_samples extension
- the VK_NV_framebuffer_mixed_samples extension
- the multisampledRenderToSingleSampled feature

then the most recent call to vkCmdSetRasterizationSamplesEXT in the current command buffer **must** have set rasterizationSamples to be the same as the number of samples for the current render pass color and/or depth/stencil attachments

- VUID-vkCmdDrawIndirect-None-08876
  If a shader object is bound to any graphics stage, the current render pass instance **must** have been begun with vkCmdBeginRenderingKHR
- VUID-vkCmdDrawIndirect-imageView-06172
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the imageView member of pDepthAttachment is not VK_NULL_HANDLE, and the layout member of pDepthAttachment is VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL, this command **must** not write any values to the depth attachment
- VUID-vkCmdDrawIndirect-imageView-06173
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the imageView member of pStencilAttachment is not VK_NULL_HANDLE, and the layout member of pStencilAttachment is VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL, this command **must** not write any values to the stencil attachment
- VUID-vkCmdDrawIndirect-imageView-06174
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the imageView member of pDepthAttachment is not VK_NULL_HANDLE, and the layout member of pDepthAttachment is VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_STENCIL_ATTACHMENT_OPTIMAL, this command **must** not write any values to the depth attachment
- VUID-vkCmdDrawIndirect-imageView-06175
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the imageView member of pStencilAttachment is not VK_NULL_HANDLE, and the layout member of pStencilAttachment is VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_STENCIL_READ_ONLY_OPTIMAL, this command **must** not write any values to the stencil attachment
- VUID-vkCmdDrawIndirect-imageView-06176
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the imageView member of pDepthAttachment is not VK_NULL_HANDLE, and the layout member of pDepthAttachment is VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL, this command **must** not write any values to the depth attachment
- VUID-vkCmdDrawIndirect-imageView-06177
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the imageView member of pStencilAttachment is not VK_NULL_HANDLE, and the layout member of pStencilAttachment is VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL, this command **must** not write any values to the stencil attachment

- VUID-vkCmdDrawIndirect-viewMask-06178
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the currently bound graphics pipeline **must** have been created with a VkPipelineRenderingCreateInfo::viewMask equal to VkRenderingInfo::viewMask
- VUID-vkCmdDrawIndirect-colorAttachmentCount-06179
  If the dynamicRenderingUnusedAttachments feature is not enabled and the current render pass instance was begun with vkCmdBeginRenderingKHR, the currently bound graphics pipeline **must** have been created with a VkPipelineRenderingCreateInfo::colorAttachmentCount equal to VkRenderingInfo::colorAttachmentCount
- VUID-vkCmdDrawIndirect-dynamicRenderingUnusedAttachments-08910
  If the dynamicRenderingUnusedAttachments feature is not enabled, and the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingInfo::colorAttachmentCount greater than 0, then each element of the VkRenderingInfo::pColorAttachments array with an imageView not equal to VK_NULL_HANDLE **must** have been created with a VkFormat equal to the corresponding element of VkPipelineRenderingCreateInfo::pColorAttachmentFormats used to create the currently bound graphics pipeline
- VUID-vkCmdDrawIndirect-dynamicRenderingUnusedAttachments-08912
  If the dynamicRenderingUnusedAttachments feature is not enabled, and the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingInfo::colorAttachmentCount greater than 0, then each element of the VkRenderingInfo::pColorAttachments array with an imageView equal to VK_NULL_HANDLE **must** have the corresponding element of VkPipelineRenderingCreateInfo::pColorAttachmentFormats used to create the currently bound pipeline equal to VK_FORMAT_UNDEFINED
- VUID-vkCmdDrawIndirect-dynamicRenderingUnusedAttachments-08911
  If the dynamicRenderingUnusedAttachments feature is enabled, and the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingInfo::colorAttachmentCount greater than 0, then each element of the VkRenderingInfo::pColorAttachments array with an imageView not equal to VK_NULL_HANDLE **must** have been created with a VkFormat equal to the corresponding element of VkPipelineRenderingCreateInfo::pColorAttachmentFormats used to create the currently bound graphics pipeline, or the corresponding element of VkPipelineRenderingCreateInfo::pColorAttachmentFormats, if it exists, **must** be VK_FORMAT_UNDEFINED
- VUID-vkCmdDrawIndirect-colorAttachmentCount-09362
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, with a VkRenderingInfo::colorAttachmentCount equal to 1, there is no shader object bound to any graphics stage, and a color attachment with a resolve mode of VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID, each element of the VkRenderingInfo::pColorAttachments array with a resolveImageView not equal to VK_NULL_HANDLE **must** have been created with an image created with a VkExternalFormatANDROID::externalFormat value equal to the VkExternalFormatANDROID::externalFormat value used to create the currently bound graphics pipeline
- VUID-vkCmdDrawIndirect-None-09363

If there is no shader object bound to any graphics stage, the current render pass instance was begun with vkCmdBeginRenderingKHR and a VkRenderingInfo::colorAttachmentCount equal to 1, and a color attachment with a resolve mode of VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID, each element of the VkRenderingInfo::pColorAttachments array with a imageView not equal to VK_NULL_HANDLE **must** have been created with an image created with a VkExternalFormatANDROID::externalFormat value equal to the VkExternalFormatANDROID::externalFormat value used to create the currently bound graphics pipeline

- VUID-vkCmdDrawIndirect-None-09364
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is no shader object bound to any graphics stage, and the currently bound graphics pipeline was created with a non-zero VkExternalFormatANDROID::externalFormat value and with the VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT dynamic state enabled, then vkCmdSetColorBlendEnableEXT **must** have set the blend enable to VK_FALSE prior to this drawing command
- VUID-vkCmdDrawIndirect-None-09365
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is no shader object bound to any graphics stage, and the currently bound graphics pipeline was created with a non-zero VkExternalFormatANDROID::externalFormat value and with the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT dynamic state enabled, then vkCmdSetRasterizationSamplesEXT **must** have set rasterizationSamples to VK_SAMPLE_COUNT_1_BIT prior to this drawing command
- VUID-vkCmdDrawIndirect-None-09366
  If there is a shader object bound to any graphics stage, and the current render pass includes a color attachment that uses the VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID resolve mode, then vkCmdSetColorBlendEnableEXT **must** have set blend enable to VK_FALSE prior to this drawing command
- VUID-vkCmdDrawIndirect-rasterizationSamples-09367
  If there is a shader object bound to any graphics stage, and the current render pass includes a color attachment that uses the VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID resolve mode, then vkCmdSetRasterizationSamplesEXT **must** have set rasterizationSamples to VK_SAMPLE_COUNT_1_BIT prior to this drawing command
- VUID-vkCmdDrawIndirect-None-09368
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is no shader object bound to any graphics stage, and the currently bound graphics pipeline was created with a non-zero VkExternalFormatANDROID::externalFormat value and with the VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR dynamic state enabled, then vkCmdSetFragmentShadingRateKHR **must** have set pFragmentSize->width to 1 prior to this drawing command
- VUID-vkCmdDrawIndirect-None-09369
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is no shader object bound to any graphics stage, and the currently bound graphics pipeline

was created with a non-zero VkExternalFormatANDROID::externalFormat value and with the VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR dynamic state enabled, then vkCmdSetFragmentShadingRateKHR **must** have set pFragmentSize->height to 1 prior to this drawing command

- VUID-vkCmdDrawIndirect-pFragmentSize-09370
  If there is a shader object bound to any graphics stage, and the current render pass includes a color attachment that uses the VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID resolve mode, then vkCmdSetFragmentShadingRateKHR **must** have set pFragmentSize->width to 1 prior to this drawing command

- VUID-vkCmdDrawIndirect-pFragmentSize-09371
  If there is a shader object bound to any graphics stage, and the current render pass includes a color attachment that uses the VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID resolve mode, then vkCmdSetFragmentShadingRateKHR **must** have set pFragmentSize->height to 1 prior to this drawing command

- VUID-vkCmdDrawIndirect-None-07749
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_WRITE_ENABLE_EXT dynamic state enabled then vkCmdSetColorWriteEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirect-None-08646
  If the colorWriteEnable feature is enabled on the device, and a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetColorWriteEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirect-attachmentCount-07750
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_WRITE_ENABLE_EXT dynamic state enabled then the attachmentCount parameter of vkCmdSetColorWriteEnableEXT **must** be greater than or equal to the VkPipelineColorBlendStateCreateInfo::attachmentCount of the currently bound graphics pipeline

- VUID-vkCmdDrawIndirect-None-08647
  If the colorWriteEnable feature is enabled on the device, and a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then the attachmentCount parameter of most recent call to vkCmdSetColorWriteEnableEXT in the current command buffer **must** be greater than or equal to the number of color attachments in the current render pass instance

- VUID-vkCmdDrawIndirect-None-07751
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_DISCARD_RECTANGLE_EXT dynamic state enabled then vkCmdSetDiscardRectangleEXT **must** have been called and not subsequently invalidated

in the current command buffer prior to this drawing command for each discard rectangle in VkPipelineDiscardRectangleStateCreateInfoEXT::discardRectangleCount

- VUID-vkCmdDrawIndirect-None-07880
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_DISCARD_RECTANGLE_ENABLE_EXT dynamic state enabled then vkCmdSetDiscardRectangleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirect-rasterizerDiscardEnable-09236
  If the VK_EXT_discard_rectangles extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetDiscardRectangleEnableEXT in the current command buffer set discardRectangleEnable to VK_TRUE, then vkCmdSetDiscardRectangleEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirect-None-08648
  If the VK_EXT_discard_rectangles extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetDiscardRectangleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirect-None-07881
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_DISCARD_RECTANGLE_MODE_EXT dynamic state enabled then vkCmdSetDiscardRectangleModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirect-None-08649
  If the VK_EXT_discard_rectangles extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetDiscardRectangleEnableEXT in the current command buffer set discardRectangleEnable to VK_TRUE, then vkCmdSetDiscardRectangleModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirect-dynamicRenderingUnusedAttachments-08913
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is not enabled, and VkRenderingInfo::pDepthAttachment->imageView was VK_NULL_HANDLE, the value of VkPipelineRenderingCreateInfo::depthAttachmentFormat used to create the currently bound graphics pipeline **must** be equal to VK_FORMAT_UNDEFINED
- VUID-vkCmdDrawIndirect-dynamicRenderingUnusedAttachments-08914
  If current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is not enabled, and VkRenderingInfo::pDepthAttachment->imageView was not VK_NULL_HANDLE, the value of VkPipelineRenderingCreateInfo::depthAttachmentFormat used to create the currently bound

graphics pipeline **must** be equal to the VkFormat used to create VkRenderingInfo::pDepthAttachment->imageView

- VUID-vkCmdDrawIndirect-dynamicRenderingUnusedAttachments-08915
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is enabled, VkRenderingInfo::pDepthAttachment->imageView was not VK_NULL_HANDLE, and the value of VkPipelineRenderingCreateInfo::depthAttachmentFormat used to create the currently bound graphics pipeline was not equal to the VkFormat used to create VkRenderingInfo::pDepthAttachment->imageView, the value of the format **must** be VK_FORMAT_UNDEFINED
- VUID-vkCmdDrawIndirect-dynamicRenderingUnusedAttachments-08916
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is not enabled, and VkRenderingInfo::pStencilAttachment->imageView was VK_NULL_HANDLE, the value of VkPipelineRenderingCreateInfo::stencilAttachmentFormat used to create the currently bound graphics pipeline **must** be equal to VK_FORMAT_UNDEFINED
- VUID-vkCmdDrawIndirect-dynamicRenderingUnusedAttachments-08917
  If current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is not enabled, and VkRenderingInfo::pStencilAttachment->imageView was not VK_NULL_HANDLE, the value of VkPipelineRenderingCreateInfo::stencilAttachmentFormat used to create the currently bound graphics pipeline **must** be equal to the VkFormat used to create VkRenderingInfo::pStencilAttachment->imageView
- VUID-vkCmdDrawIndirect-dynamicRenderingUnusedAttachments-08918
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is enabled, VkRenderingInfo::pStencilAttachment->imageView was not VK_NULL_HANDLE, and the value of VkPipelineRenderingCreateInfo::stencilAttachmentFormat used to create the currently bound graphics pipeline was not equal to the VkFormat used to create VkRenderingInfo::pStencilAttachment->imageView, the value of the format **must** be VK_FORMAT_UNDEFINED
- VUID-vkCmdDrawIndirect-imageView-06183
  If the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingFragmentShadingRateAttachmentInfoKHR::imageView was not VK_NULL_HANDLE, the currently bound graphics pipeline **must** have been created with VK_PIPELINE_CREATE_RENDERING_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR
- VUID-vkCmdDrawIndirect-imageView-06184
  If the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingFragmentDensityMapAttachmentInfoEXT::imageView was not VK_NULL_HANDLE, the currently bound graphics pipeline **must** have been created with VK_PIPELINE_CREATE_RENDERING_FRAGMENT_DENSITY_MAP_ATTACHMENT_BIT_EXT
- VUID-vkCmdDrawIndirect-colorAttachmentCount-06185
  If the currently bound pipeline was created with a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and the current render pass instance was begun with vkCmdBeginRenderingKHR with a VkRenderingInfo::colorAttachmentCount parameter greater than 0, then each element of the VkRenderingInfo::pColorAttachments array with a imageView not equal to VK_NULL_HANDLE **must** have been created with a

sample count equal to the corresponding element of the pColorAttachmentSamples member of VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV used to create the currently bound graphics pipeline

- VUID-vkCmdDrawIndirect-pDepthAttachment-06186
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the currently bound pipeline was created with a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and VkRenderingInfo::pDepthAttachment->imageView was not VK_NULL_HANDLE, the value of the depthStencilAttachmentSamples member of VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV used to create the currently bound graphics pipeline **must** be equal to the sample count used to create VkRenderingInfo::pDepthAttachment->imageView
- VUID-vkCmdDrawIndirect-pStencilAttachment-06187
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the currently bound pipeline was created with a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and VkRenderingInfo::pStencilAttachment->imageView was not VK_NULL_HANDLE, the value of the depthStencilAttachmentSamples member of VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV used to create the currently bound graphics pipeline **must** be equal to the sample count used to create VkRenderingInfo::pStencilAttachment->imageView
- VUID-vkCmdDrawIndirect-multisampledRenderToSingleSampled-07285
  If the currently bound pipeline was created without a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and the multisampledRenderToSingleSampled feature is not enabled, and the current render pass instance was begun with vkCmdBeginRenderingKHR with a VkRenderingInfo::colorAttachmentCount parameter greater than 0, then each element of the VkRenderingInfo::pColorAttachments array with a imageView not equal to VK_NULL_HANDLE **must** have been created with a sample count equal to the value of rasterizationSamples for the currently bound graphics pipeline
- VUID-vkCmdDrawIndirect-multisampledRenderToSingleSampled-07286
  If the currently bound pipeline was created without a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and the multisampledRenderToSingleSampled feature is not enabled, and VkRenderingInfo::pDepthAttachment->imageView was not VK_NULL_HANDLE, the value of rasterizationSamples for the currently bound graphics pipeline **must** be equal to the sample count used to create VkRenderingInfo::pDepthAttachment->imageView
- VUID-vkCmdDrawIndirect-multisampledRenderToSingleSampled-07287
  If the currently bound pipeline was created without a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and the multisampledRenderToSingleSampled feature is not enabled, and VkRenderingInfo::pStencilAttachment->imageView was not VK_NULL_HANDLE, the value of rasterizationSamples for the currently bound graphics pipeline **must** be equal to the sample count used to create VkRenderingInfo::pStencilAttachment->imageView
- VUID-vkCmdDrawIndirect-pNext-07935
  If this command has been called inside a render pass instance started with vkCmdBeginRenderingKHR, and the pNext chain of VkRenderingInfo includes a

VkMultisampledRenderToSingleSampledInfoEXT structure with multisampledRenderToSingleSampledEnable equal to VK_TRUE, then the value of rasterizationSamples for the currently bound graphics pipeline **must** be equal to VkMultisampledRenderToSingleSampledInfoEXT::rasterizationSamples

- VUID-vkCmdDrawIndirect-renderPass-06198
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the currently bound pipeline **must** have been created with a VkGraphicsPipelineCreateInfo::renderPass equal to VK_NULL_HANDLE
- VUID-vkCmdDrawIndirect-pColorAttachments-08963
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is a graphics pipeline bound with a fragment shader that statically writes to a color attachment, the color write mask is not zero, color writes are enabled, and the corresponding element of the VkRenderingInfo::pColorAttachments->imageView was not VK_NULL_HANDLE, then the corresponding element of VkPipelineRenderingCreateInfo::pColorAttachmentFormats used to create the pipeline **must** not be VK_FORMAT_UNDEFINED
- VUID-vkCmdDrawIndirect-pDepthAttachment-08964
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is a graphics pipeline bound, depth test is enabled, depth write is enabled, and the VkRenderingInfo::pDepthAttachment->imageView was not VK_NULL_HANDLE, then the VkPipelineRenderingCreateInfo::depthAttachmentFormat used to create the pipeline **must** not be VK_FORMAT_UNDEFINED
- VUID-vkCmdDrawIndirect-pStencilAttachment-08965
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is a graphics pipeline bound, stencil test is enabled and the VkRenderingInfo::pStencilAttachment->imageView was not VK_NULL_HANDLE, then the VkPipelineRenderingCreateInfo::stencilAttachmentFormat used to create the pipeline **must** not be VK_FORMAT_UNDEFINED
- VUID-vkCmdDrawIndirect-primitivesGeneratedQueryWithRasterizerDiscard-06708
  If the primitivesGeneratedQueryWithRasterizerDiscard feature is not enabled and the VK_QUERY_TYPE_PRIMITIVES_GENERATED_EXT query is active, rasterization discard **must** not be enabled
- VUID-vkCmdDrawIndirect-primitivesGeneratedQueryWithNonZeroStreams-06709
  If the primitivesGeneratedQueryWithNonZeroStreams feature is not enabled and the VK_QUERY_TYPE_PRIMITIVES_GENERATED_EXT query is active, the bound graphics pipeline **must** not have been created with a non-zero value in VkPipelineRasterizationStateStreamCreateInfoEXT::rasterizationStream
- VUID-vkCmdDrawIndirect-None-07619
  If a shader object is bound to the VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_TESSELLATION_DOMAIN_ORIGIN_EXT dynamic state enabled, then vkCmdSetTessellationDomainOriginEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirect-None-07620
  If the depthClamp feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the

VK_DYNAMIC_STATE_DEPTH_CLAMP_ENABLE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetDepthClampEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirect-None-07621
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_POLYGON_MODE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetPolygonModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirect-None-07622
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetRasterizationSamplesEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirect-None-07623
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_SAMPLE_MASK_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetSampleMaskEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirect-alphaToCoverageEnable-08919
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_ALPHA_TO_COVERAGE_ENABLE_EXT dynamic state enabled, and alphaToCoverageEnable was VK_TRUE in the last call to vkCmdSetAlphaToCoverageEnableEXT, then the Fragment Output Interface **must** contain a variable for the alpha Component word in Location 0 at Index 0
- VUID-vkCmdDrawIndirect-alphaToCoverageEnable-08920
  If a shader object is bound to any graphics stage, and the most recent call to vkCmdSetAlphaToCoverageEnableEXT in the current command buffer set alphaToCoverageEnable to VK_TRUE, then the Fragment Output Interface **must** contain a variable for the alpha Component word in Location 0 at Index 0
- VUID-vkCmdDrawIndirect-None-07624
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_ALPHA_TO_COVERAGE_ENABLE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetAlphaToCoverageEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirect-None-07625
  If the alphaToOne feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_ALPHA_TO_ONE_ENABLE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetAlphaToOneEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this

drawing command

- VUID-vkCmdDrawIndirect-None-07626
  If the `logicOp` feature is enabled, a shader object is bound to the `VK_SHADER_STAGE_FRAGMENT_BIT` stage or a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_LOGIC_OP_ENABLE_EXT` dynamic state enabled, and the current value of `rasterizerDiscardEnable` is `VK_FALSE`, then vkCmdSetLogicOpEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirect-None-07627
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT` dynamic state enabled then vkCmdSetColorBlendEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirect-None-08657
  If a shader object is bound to the `VK_SHADER_STAGE_FRAGMENT_BIT` stage, and both the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE` and there are color attachments bound, then vkCmdSetColorBlendEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirect-None-07628
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_COLOR_BLEND_EQUATION_EXT` dynamic state enabled then vkCmdSetColorBlendEquationEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirect-None-08658
  If a shader object is bound to the `VK_SHADER_STAGE_FRAGMENT_BIT` stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE`, and the most recent call to vkCmdSetColorBlendEnableEXT for any attachment set that attachment's value in `pColorBlendEnables` to `VK_TRUE`, then vkCmdSetColorBlendEquationEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirect-None-07629
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_COLOR_WRITE_MASK_EXT` dynamic state enabled then vkCmdSetColorWriteMaskEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirect-None-08659
  If a shader object is bound to the `VK_SHADER_STAGE_FRAGMENT_BIT` stage, and both the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE` and there are color attachments bound, then vkCmdSetColorWriteMaskEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirect-None-07630
  If the `geometryStreams` feature is enabled, and a shader object is bound to the

VK_SHADER_STAGE_GEOMETRY_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_RASTERIZATION_STREAM_EXT dynamic state enabled, then vkCmdSetRasterizationStreamEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirect-None-07631
  If the VK_EXT_conservative_rasterization extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_CONSERVATIVE_RASTERIZATION_MODE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetConservativeRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirect-None-07632
  If the VK_EXT_conservative_rasterization extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_EXTRA_PRIMITIVE_OVERESTIMATION_SIZE_EXT dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of conservativeRasterizationMode is VK_CONSERVATIVE_RASTERIZATION_MODE_OVERESTIMATE_EXT, then vkCmdSetExtraPrimitiveOverestimationSizeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirect-None-07633
  If the depthClipEnable feature is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_CLIP_ENABLE_EXT dynamic state, then vkCmdSetDepthClipEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirect-None-07634
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT dynamic state enabled then vkCmdSetSampleLocationsEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirect-None-08664
  If the VK_EXT_sample_locations extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetSampleLocationsEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirect-None-07635
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_ADVANCED_EXT dynamic state enabled then vkCmdSetColorBlendAdvancedEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirect-rasterizerDiscardEnable-09416
  If the VK_EXT_blend_operation_advanced extension is enabled, and a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set

rasterizerDiscardEnable to VK_FALSE, then at least one of vkCmdSetColorBlendEquationEXT and vkCmdSetColorBlendAdvancedEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirect-None-07636
  If the VK_EXT_provoking_vertex extension is enabled, a shader object is bound to the VK_SHADER_STAGE_VERTEX_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_PROVOKING_VERTEX_MODE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetProvokingVertexModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirect-None-07637
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT dynamic state enabled then vkCmdSetLineRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirect-None-08666
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetPolygonModeEXT in the current command buffer set polygonMode to VK_POLYGON_MODE_LINE, then vkCmdSetLineRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirect-None-08667
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object is bound to the VK_SHADER_STAGE_VERTEX_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetPrimitiveTopologyEXT in the current command buffer set primitiveTopology to any line topology, then vkCmdSetLineRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirect-None-08668
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object that outputs line primitives is bound to the VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT or VK_SHADER_STAGE_GEOMETRY_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetLineRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirect-None-07638
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT dynamic state enabled then

vkCmdSetLineStippleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirect-None-08669
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetPolygonModeEXT in the current command buffer set polygonMode to VK_POLYGON_MODE_LINE, then vkCmdSetLineStippleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirect-None-08670
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object is bound to the VK_SHADER_STAGE_VERTEX_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetPrimitiveTopologyEXT in the current command buffer set primitiveTopology to any line topology, then vkCmdSetLineStippleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirect-None-08671
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object that outputs line primitives is bound to the VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT or VK_SHADER_STAGE_GEOMETRY_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetLineStippleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirect-None-07849
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_KHR dynamic state enabled then vkCmdSetLineStippleKHR **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirect-None-08672
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetLineStippleEnableEXT in the current command buffer set stippledLineEnable to VK_TRUE, then vkCmdSetLineStippleEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirect-None-07639
  If the depthClipControl feature is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_CLIP_NEGATIVE_ONE_TO_ONE_EXT dynamic state enabled, then vkCmdSetDepthClipNegativeOneToOneEXT **must** have been called and not subsequently

invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirect-None-07640
  If the VK_NV_clip_space_w_scaling extension is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_VIEWPORT_W_SCALING_ENABLE_NV dynamic state enabled, then vkCmdSetViewportWScalingEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirect-None-07641
  If the VK_NV_viewport_swizzle extension is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_VIEWPORT_SWIZZLE_NV dynamic state enabled, then vkCmdSetViewportSwizzleNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirect-None-07642
  If the VK_NV_fragment_coverage_to_color extension is enabled, a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_TO_COLOR_ENABLE_NV dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetCoverageToColorEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirect-None-07643
  If the VK_NV_fragment_coverage_to_color extension is enabled, a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_TO_COLOR_LOCATION_NV dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of coverageToColorEnable is VK_TRUE, then vkCmdSetCoverageToColorLocationNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirect-None-07644
  If the VK_NV_framebuffer_mixed_samples extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_MODULATION_MODE_NV dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetCoverageModulationModeNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirect-None-07645
  If the VK_NV_framebuffer_mixed_samples extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_MODULATION_TABLE_ENABLE_NV dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of coverageModulationMode is any value other than VK_COVERAGE_MODULATION_MODE_NONE_NV, then vkCmdSetCoverageModulationTableEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirect-None-07646
  If the VK_NV_framebuffer_mixed_samples extension is enabled, a shader object is bound to

any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_MODULATION_TABLE_NV dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of coverageModulationTableEnable is VK_TRUE, then vkCmdSetCoverageModulationTableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirect-None-07647
  If the shadingRateImage feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_SHADING_RATE_IMAGE_ENABLE_NV dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetShadingRateImageEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirect-pipelineFragmentShadingRate-09238
  If the pipelineFragmentShadingRate feature is enabled, a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetFragmentShadingRateKHR **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirect-None-07648
  If the representativeFragmentTest feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_REPRESENTATIVE_FRAGMENT_TEST_ENABLE_NV dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetRepresentativeFragmentTestEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirect-None-07649
  If the coverageReductionMode feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_REDUCTION_MODE_NV dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetCoverageReductionModeNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirect-pColorBlendEnables-07470
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT state enabled and the last call to vkCmdSetColorBlendEnableEXT set pColorBlendEnables for any attachment to VK_TRUE, then for those attachments in the subpass the corresponding image view's format features **must** contain VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT

- VUID-vkCmdDrawIndirect-rasterizationSamples-07471
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, and the current subpass does not use any color and/or depth/stencil attachments, then the rasterizationSamples in the last call to vkCmdSetRasterizationSamplesEXT **must** follow the rules for a zero-

attachment subpass

- VUID-vkCmdDrawIndirect-samples-07472
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_SAMPLE_MASK_EXT` state enabled and the `VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT` state disabled, then the `samples` parameter in the last call to vkCmdSetSampleMaskEXT **must** be greater or equal to the VkPipelineMultisampleStateCreateInfo::`rasterizationSamples` parameter used to create the bound graphics pipeline
- VUID-vkCmdDrawIndirect-samples-07473
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_SAMPLE_MASK_EXT` state and `VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT` states enabled, then the `samples` parameter in the last call to vkCmdSetSampleMaskEXT **must** be greater or equal to the `rasterizationSamples` parameter in the last call to vkCmdSetRasterizationSamplesEXT
- VUID-vkCmdDrawIndirect-rasterizationSamples-07474
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT` state enabled, and neither the `VK_AMD_mixed_attachment_samples` nor the `VK_NV_framebuffer_mixed_samples` extensions are enabled, then the `rasterizationSamples` in the last call to vkCmdSetRasterizationSamplesEXT **must** be the same as the current subpass color and/or depth/stencil attachments
- VUID-vkCmdDrawIndirect-None-09211
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT` state enabled, or a shader object is bound to any graphics stage, and the current render pass instance includes a VkMultisampledRenderToSingleSampledInfoEXT structure with `multisampledRenderToSingleSampledEnable` equal to `VK_TRUE`, then the `rasterizationSamples` in the last call to vkCmdSetRasterizationSamplesEXT **must** be the same as the `rasterizationSamples` member of that structure
- VUID-vkCmdDrawIndirect-firstAttachment-07476
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT` dynamic state enabled then vkCmdSetColorBlendEnableEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the `firstAttachment` and `attachmentCount` parameters of `vkCmdSetColorBlendEnableEXT` calls **must** specify an enable for all active color attachments in the current subpass
- VUID-vkCmdDrawIndirect-rasterizerDiscardEnable-09417
  If a shader object is bound to the `VK_SHADER_STAGE_FRAGMENT_BIT` stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE`, then vkCmdSetColorBlendEnableEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the `firstAttachment` and `attachmentCount` parameters of `vkCmdSetColorBlendEnableEXT` calls **must** specify an enable for all active color attachments in the current subpass
- VUID-vkCmdDrawIndirect-firstAttachment-07477

If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_COLOR_BLEND_EQUATION_EXT` dynamic state enabled then vkCmdSetColorBlendEquationEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the `firstAttachment` and `attachmentCount` parameters of `vkCmdSetColorBlendEquationEXT` calls **must** specify the blend equations for all active color attachments in the current subpass where blending is enabled

- VUID-vkCmdDrawIndirect-rasterizerDiscardEnable-09418
  If a shader object is bound to the `VK_SHADER_STAGE_FRAGMENT_BIT` stage, and both the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE` and there are color attachments bound, then vkCmdSetColorBlendEquationEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the `firstAttachment` and `attachmentCount` parameters of `vkCmdSetColorBlendEquationEXT` calls **must** specify the blend equations for all active color attachments in the current subpass where blending is enabled

- VUID-vkCmdDrawIndirect-firstAttachment-07478
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_COLOR_WRITE_MASK_EXT` dynamic state enabled then vkCmdSetColorWriteMaskEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the `firstAttachment` and `attachmentCount` parameters of `vkCmdSetColorWriteMaskEXT` calls **must** specify the color write mask for all active color attachments in the current subpass

- VUID-vkCmdDrawIndirect-rasterizerDiscardEnable-09419
  If a shader object is bound to the `VK_SHADER_STAGE_FRAGMENT_BIT` stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE`, then vkCmdSetColorWriteMaskEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the `firstAttachment` and `attachmentCount` parameters of `vkCmdSetColorWriteMaskEXT` calls **must** specify the color write mask for all active color attachments in the current subpass

- VUID-vkCmdDrawIndirect-firstAttachment-07479
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_COLOR_BLEND_ADVANCED_EXT` dynamic state enabled then vkCmdSetColorBlendAdvancedEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the `firstAttachment` and `attachmentCount` parameters of `vkCmdSetColorBlendAdvancedEXT` calls **must** specify the advanced blend equations for all active color attachments in the current subpass where blending is enabled

- VUID-vkCmdDrawIndirect-advancedBlendMaxColorAttachments-07480
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_COLOR_BLEND_ADVANCED_EXT` and `VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT` dynamic states enabled and the last calls to vkCmdSetColorBlendEnableEXT and vkCmdSetColorBlendAdvancedEXT have enabled advanced blending, then the number of active color attachments in the current subpass **must** not exceed `advancedBlendMaxColorAttachments`

- VUID-vkCmdDrawIndirect-primitivesGeneratedQueryWithNonZeroStreams-07481
  If the `primitivesGeneratedQueryWithNonZeroStreams` feature is not enabled and the `VK_QUERY_TYPE_PRIMITIVES_GENERATED_EXT` query is active, and the bound graphics pipeline was created with `VK_DYNAMIC_STATE_RASTERIZATION_STREAM_EXT` state enabled, the last call to vkCmdSetRasterizationStreamEXT **must** have set the `rasterizationStream` to zero
- VUID-vkCmdDrawIndirect-sampleLocationsPerPixel-07482
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT` state enabled and the `VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT` state disabled, then the `sampleLocationsPerPixel` member of `pSampleLocationsInfo` in the last call to vkCmdSetSampleLocationsEXT **must** equal the `rasterizationSamples` member of the VkPipelineMultisampleStateCreateInfo structure the bound graphics pipeline has been created with
- VUID-vkCmdDrawIndirect-sampleLocationsPerPixel-07483
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT` state enabled and the `VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT` state enabled, then the `sampleLocationsPerPixel` member of `pSampleLocationsInfo` in the last call to vkCmdSetSampleLocationsEXT **must** equal the `rasterizationSamples` parameter of the last call to vkCmdSetRasterizationSamplesEXT
- VUID-vkCmdDrawIndirect-sampleLocationsEnable-07484
  If a shader object is bound to the `VK_SHADER_STAGE_FRAGMENT_BIT` stage, or the bound graphics pipeline was created with the `VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT` state enabled, and `sampleLocationsEnable` was `VK_TRUE` in the last call to vkCmdSetSampleLocationsEnableEXT, and the current subpass has a depth/stencil attachment, then that attachment **must** have been created with the `VK_IMAGE_CREATE_SAMPLE_LOCATIONS_COMPATIBLE_DEPTH_BIT_EXT` bit set
- VUID-vkCmdDrawIndirect-sampleLocationsEnable-07485
  If a shader object is bound to the `VK_SHADER_STAGE_FRAGMENT_BIT` stage, or the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT` state enabled and the `VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT` state enabled, and if `sampleLocationsEnable` was `VK_TRUE` in the last call to vkCmdSetSampleLocationsEnableEXT, then the `sampleLocationsInfo.sampleLocationGridSize.width` in the last call to vkCmdSetSampleLocationsEXT **must** evenly divide VkMultisamplePropertiesEXT ::`sampleLocationGridSize.width` as returned by vkGetPhysicalDeviceMultisamplePropertiesEXT with a `samples` parameter equaling `rasterizationSamples`
- VUID-vkCmdDrawIndirect-sampleLocationsEnable-07486
  If a shader object is bound to the `VK_SHADER_STAGE_FRAGMENT_BIT` stage, or the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT` state enabled and the `VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT` state enabled, and if `sampleLocationsEnable` was `VK_TRUE` in the last call to vkCmdSetSampleLocationsEnableEXT, then the `sampleLocationsInfo.sampleLocationGridSize.height` in the last call to vkCmdSetSampleLocationsEXT **must** evenly divide VkMultisamplePropertiesEXT

::sampleLocationGridSize.height as returned by vkGetPhysicalDeviceMultisamplePropertiesEXT with a samples parameter equaling rasterizationSamples

- VUID-vkCmdDrawIndirect-sampleLocationsEnable-07487
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, and if sampleLocationsEnable was VK_TRUE in the last call to vkCmdSetSampleLocationsEnableEXT, the fragment shader code **must** not statically use the extended instruction InterpolateAtSample

- VUID-vkCmdDrawIndirect-sampleLocationsEnable-07936
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state disabled and the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, the sampleLocationsEnable member of a VkPipelineSampleLocationsStateCreateInfoEXT::sampleLocationsEnable in the bound graphics pipeline is VK_TRUE or VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, then, sampleLocationsInfo.sampleLocationGridSize.width **must** evenly divide VkMultisamplePropertiesEXT::sampleLocationGridSize.width as returned by vkGetPhysicalDeviceMultisamplePropertiesEXT with a samples parameter equaling the value of rasterizationSamples in the last call to vkCmdSetRasterizationSamplesEXT

- VUID-vkCmdDrawIndirect-sampleLocationsEnable-07937
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state disabled and the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, the sampleLocationsEnable member of a VkPipelineSampleLocationsStateCreateInfoEXT::sampleLocationsEnable in the bound graphics pipeline is VK_TRUE or VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, then, sampleLocationsInfo.sampleLocationGridSize.height **must** evenly divide VkMultisamplePropertiesEXT::sampleLocationGridSize.height as returned by vkGetPhysicalDeviceMultisamplePropertiesEXT with a samples parameter equaling the value of rasterizationSamples in the last call to vkCmdSetRasterizationSamplesEXT

- VUID-vkCmdDrawIndirect-sampleLocationsEnable-07938
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state disabled and the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, the sampleLocationsEnable member of a VkPipelineSampleLocationsStateCreateInfoEXT::sampleLocationsEnable in the bound graphics pipeline is VK_TRUE or VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, then, sampleLocationsInfo.sampleLocationsPerPixel **must** equal rasterizationSamples in the last call to vkCmdSetRasterizationSamplesEXT

- VUID-vkCmdDrawIndirect-coverageModulationTableEnable-07488
  If a shader object is bound to any graphics stage or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COVERAGE_MODULATION_TABLE_ENABLE_NV state enabled, and the last call to vkCmdSetCoverageModulationTableEnableNV set coverageModulationTableEnable to VK_TRUE, then the coverageModulationTableCount parameter in the last call to vkCmdSetCoverageModulationTableNV **must** equal the current rasterizationSamples divided by the number of color samples in the current subpass

- VUID-vkCmdDrawIndirect-rasterizationSamples-07489
  If the VK_NV_framebuffer_mixed_samples extension is enabled, and if current subpass has a depth/stencil attachment and depth test, stencil test, or depth bounds test are enabled in the currently bound pipeline state, then the current rasterizationSamples **must** be the same as the sample count of the depth/stencil attachment
- VUID-vkCmdDrawIndirect-coverageToColorEnable-07490
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COVERAGE_TO_COLOR_ENABLE_NV state enabled and the last call to vkCmdSetCoverageToColorEnableNV set the coverageToColorEnable to VK_TRUE, then the current subpass **must** have a color attachment at the location selected by the last call to vkCmdSetCoverageToColorLocationNV coverageToColorLocation, with a VkFormat of VK_FORMAT_R8_UINT, VK_FORMAT_R8_SINT, VK_FORMAT_R16_UINT, VK_FORMAT_R16_SINT, VK_FORMAT_R32_UINT, or VK_FORMAT_R32_SINT
- VUID-vkCmdDrawIndirect-rasterizerDiscardEnable-09420
  If the VK_NV_fragment_coverage_to_color extension is enabled, and a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the last call to vkCmdSetCoverageToColorEnableNV set the coverageToColorEnable to VK_TRUE, then the current subpass **must** have a color attachment at the location selected by the last call to vkCmdSetCoverageToColorLocationNV coverageToColorLocation, with a VkFormat of VK_FORMAT_R8_UINT, VK_FORMAT_R8_SINT, VK_FORMAT_R16_UINT, VK_FORMAT_R16_SINT, VK_FORMAT_R32_UINT, or VK_FORMAT_R32_SINT
- VUID-vkCmdDrawIndirect-coverageReductionMode-07491
  If this VK_NV_coverage_reduction_mode extension is enabled, the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COVERAGE_TO_COLOR_ENABLE_NV and VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT states enabled, the current coverage reduction mode coverageReductionMode, then the current rasterizationSamples, and the sample counts for the color and depth/stencil attachments (if the subpass has them) **must** be a valid combination returned by vkGetPhysicalDeviceSupportedFramebufferMixedSamplesCombinationsNV
- VUID-vkCmdDrawIndirect-viewportCount-07492
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, but not the VK_DYNAMIC_STATE_VIEWPORT_SWIZZLE_NV dynamic state enabled, then the bound graphics pipeline **must** have been created with VkPipelineViewportSwizzleStateCreateInfoNV ::viewportCount greater or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawIndirect-viewportCount-07493
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT and VK_DYNAMIC_STATE_VIEWPORT_SWIZZLE_NV dynamic states enabled then the viewportCount parameter in the last call to vkCmdSetViewportSwizzleNV **must** be greater than or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawIndirect-viewportCount-09421

If the VK_NV_viewport_swizzle extension is enabled, and a shader object is bound to any graphics stage, then the viewportCount parameter in the last call to vkCmdSetViewportSwizzleNV **must** be greater than or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT

- VUID-vkCmdDrawIndirect-rasterizationSamples-07494
  If the VK_NV_framebuffer_mixed_samples extension is enabled, and if the current subpass has any color attachments and rasterizationSamples of the last call to vkCmdSetRasterizationSamplesEXT is greater than the number of color samples, then the pipeline sampleShadingEnable **must** be VK_FALSE

- VUID-vkCmdDrawIndirect-stippledLineEnable-07495
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT or VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT dynamic states enabled, and if the current stippledLineEnable state is VK_TRUE and the current lineRasterizationMode state is VK_LINE_RASTERIZATION_MODE_RECTANGULAR_KHR, then the stippledRectangularLines feature **must** be enabled

- VUID-vkCmdDrawIndirect-stippledLineEnable-07496
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT or VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT dynamic states enabled, and if the current stippledLineEnable state is VK_TRUE and the current lineRasterizationMode state is VK_LINE_RASTERIZATION_MODE_BRESENHAM_KHR, then the stippledBresenhamLines feature **must** be enabled

- VUID-vkCmdDrawIndirect-stippledLineEnable-07497
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT or VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT dynamic states enabled, and if the current stippledLineEnable state is VK_TRUE and the current lineRasterizationMode state is VK_LINE_RASTERIZATION_MODE_RECTANGULAR_SMOOTH_KHR, then the stippledSmoothLines feature **must** be enabled

- VUID-vkCmdDrawIndirect-stippledLineEnable-07498
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT or VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT dynamic states enabled, and if the current stippledLineEnable state is VK_TRUE and the current lineRasterizationMode state is VK_LINE_RASTERIZATION_MODE_DEFAULT_KHR, then the stippledRectangularLines feature **must** be enabled and VkPhysicalDeviceLimits::strictLines **must** be VK_TRUE

- VUID-vkCmdDrawIndirect-conservativePointAndLineRasterization-07499
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_CONSERVATIVE_RASTERIZATION_MODE_EXT dynamic state enabled, conservativePointAndLineRasterization is not supported, and the effective primitive topology output by the last pre-rasterization shader stage is a line or point, then the conservativeRasterizationMode set by the last call to vkCmdSetConservativeRasterizationModeEXT **must** be VK_CONSERVATIVE_RASTERIZATION_MODE_DISABLED_EXT

- VUID-vkCmdDrawIndirect-stage-07073
  If the currently bound pipeline was created with the VkPipelineShaderStageCreateInfo::stage member of an element of VkGraphicsPipelineCreateInfo::pStages set to VK_SHADER_STAGE_VERTEX_BIT, VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT, VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT or VK_SHADER_STAGE_GEOMETRY_BIT, then Mesh Shader Queries **must** not be active
- VUID-vkCmdDrawIndirect-None-08877
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_ATTACHMENT_FEEDBACK_LOOP_ENABLE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetAttachmentFeedbackLoopEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirect-None-07850
  If dynamic state was inherited from VkCommandBufferInheritanceViewportScissorInfoNV, it **must** be set in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirect-None-08684
  If there is no bound graphics pipeline, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_VERTEX_BIT
- VUID-vkCmdDrawIndirect-None-08685
  If there is no bound graphics pipeline, and the tessellationShader feature is enabled, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT
- VUID-vkCmdDrawIndirect-None-08686
  If there is no bound graphics pipeline, and the tessellationShader feature is enabled, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT
- VUID-vkCmdDrawIndirect-None-08687
  If there is no bound graphics pipeline, and the geometryShader feature is enabled, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_GEOMETRY_BIT
- VUID-vkCmdDrawIndirect-None-08688
  If there is no bound graphics pipeline, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_FRAGMENT_BIT
- VUID-vkCmdDrawIndirect-None-08689
  If there is no bound graphics pipeline, and the taskShader feature is enabled, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_TASK_BIT_EXT
- VUID-vkCmdDrawIndirect-None-08690
  If there is no bound graphics pipeline, and the meshShader feature is enabled, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_MESH_BIT_EXT
- VUID-vkCmdDrawIndirect-None-08693

If there is no bound graphics pipeline, and at least one of the taskShader and meshShader features is enabled, one of the VK_SHADER_STAGE_VERTEX_BIT or VK_SHADER_STAGE_MESH_BIT_EXT stages **must** have a valid VkShaderEXT bound, and the other **must** have no VkShaderEXT bound

- VUID-vkCmdDrawIndirect-None-08694
  If there is no bound graphics pipeline, and both the taskShader and meshShader features are enabled, and a valid VkShaderEXT is bound the to the VK_SHADER_STAGE_MESH_BIT_EXT stage, and that VkShaderEXT was created without the VK_SHADER_CREATE_NO_TASK_SHADER_BIT_EXT flag, a valid VkShaderEXT **must** be bound to the VK_SHADER_STAGE_TASK_BIT_EXT stage

- VUID-vkCmdDrawIndirect-None-08695
  If there is no bound graphics pipeline, and both the taskShader and meshShader features are enabled, and a valid VkShaderEXT is bound the to the VK_SHADER_STAGE_MESH_BIT_EXT stage, and that VkShaderEXT was created with the VK_SHADER_CREATE_NO_TASK_SHADER_BIT_EXT flag, there **must** be no VkShaderEXT bound to the VK_SHADER_STAGE_TASK_BIT_EXT stage

- VUID-vkCmdDrawIndirect-None-08696
  If there is no bound graphics pipeline, and a valid VkShaderEXT is bound to the VK_SHADER_STAGE_VERTEX_BIT stage, there **must** be no VkShaderEXT bound to either the VK_SHADER_STAGE_TASK_BIT_EXT stage or the VK_SHADER_STAGE_MESH_BIT_EXT stage

- VUID-vkCmdDrawIndirect-None-08698
  If any graphics shader is bound which was created with the VK_SHADER_CREATE_LINK_STAGE_BIT_EXT flag, then all shaders created with the VK_SHADER_CREATE_LINK_STAGE_BIT_EXT flag in the same vkCreateShadersEXT call **must** also be bound

- VUID-vkCmdDrawIndirect-None-08699
  If any graphics shader is bound which was created with the VK_SHADER_CREATE_LINK_STAGE_BIT_EXT flag, any stages in between stages whose shaders which did not create a shader with the VK_SHADER_CREATE_LINK_STAGE_BIT_EXT flag as part of the same vkCreateShadersEXT call **must** not have any VkShaderEXT bound

- VUID-vkCmdDrawIndirect-None-08878
  All bound graphics shader objects **must** have been created with identical or identically defined push constant ranges

- VUID-vkCmdDrawIndirect-None-08879
  All bound graphics shader objects **must** have been created with identical or identically defined arrays of descriptor set layouts

- VUID-vkCmdDrawIndirect-colorAttachmentCount-09372
  If the current render pass instance was begun with vkCmdBeginRenderingKHR and a VkRenderingInfo::colorAttachmentCount equal to 1, a color attachment with a resolve mode of VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID, and a fragment shader is bound, it **must** not declare the DepthReplacing or StencilRefReplacingEXT execution modes

- VUID-vkCmdDrawIndirect-pDynamicStates-08715
  If the bound graphics pipeline state includes a fragment shader stage, was created with VK_DYNAMIC_STATE_DEPTH_WRITE_ENABLE set in VkPipelineDynamicStateCreateInfo::pDynamicStates, and the fragment shader declares the EarlyFragmentTests execution

mode and uses OpDepthAttachmentReadEXT, the depthWriteEnable parameter in the last call to vkCmdSetDepthWriteEnableEXT **must** be VK_FALSE

- VUID-vkCmdDrawIndirect-pDynamicStates-08716
  If the bound graphics pipeline state includes a fragment shader stage, was created with VK_DYNAMIC_STATE_STENCIL_WRITE_MASK set in VkPipelineDynamicStateCreateInfo::pDynamicStates, and the fragment shader declares the EarlyFragmentTests execution mode and uses OpStencilAttachmentReadEXT, the writeMask parameter in the last call to vkCmdSetStencilWriteMask **must** be 0

- VUID-vkCmdDrawIndirect-None-09116
  If a shader object is bound to any graphics stage or the currently bound graphics pipeline was created with VK_DYNAMIC_STATE_COLOR_WRITE_MASK_EXT, and the format of any color attachment is VK_FORMAT_E5B9G9R9_UFLOAT_PACK32, the corresponding element of the pColorWriteMasks parameter of vkCmdSetColorWriteMaskEXT **must** either include all of VK_COLOR_COMPONENT_R_BIT, VK_COLOR_COMPONENT_G_BIT, and VK_COLOR_COMPONENT_B_BIT, or none of them

- VUID-vkCmdDrawIndirect-maxFragmentDualSrcAttachments-09239
  If blending is enabled for any attachment where either the source or destination blend factors for that attachment use the secondary color input, the maximum value of Location for any output attachment statically used in the Fragment Execution Model executed by this command **must** be less than maxFragmentDualSrcAttachments

- VUID-vkCmdDrawIndirect-None-09548
  If the current render pass was begun with vkCmdBeginRenderingKHR, and there is no shader object bound to any graphics stage, the value of each element of VkRenderingAttachmentLocationInfoKHR::pColorAttachmentLocations set by vkCmdSetRenderingAttachmentLocationsKHR **must** match the value set for the corresponding element in the currently bound pipeline

- VUID-vkCmdDrawIndirect-None-09549
  If the current render pass was begun with vkCmdBeginRenderingKHR, and there is no shader object bound to any graphics stage, input attachment index mappings in the currently bound pipeline **must** match those set for the current render pass instance via VkRenderingInputAttachmentIndexInfoKHR

- VUID-vkCmdDrawIndirect-None-09642
  If the current render pass was begun with vkCmdBeginRenderingKHR with the VK_RENDERING_ENABLE_LEGACY_DITHERING_BIT_EXT flag, the bound graphics pipeline **must** have been created with VK_PIPELINE_CREATE_2_ENABLE_LEGACY_DITHERING_BIT_EXT

- VUID-vkCmdDrawIndirect-None-09643
  If the bound graphics pipeline was created with VK_PIPELINE_CREATE_2_ENABLE_LEGACY_DITHERING_BIT_EXT, the current render pass **must** have begun with vkCmdBeginRenderingKHR with the VK_RENDERING_ENABLE_LEGACY_DITHERING_BIT_EXT flag

- VUID-vkCmdDrawIndirect-None-04007
  All vertex input bindings accessed via vertex input variables declared in the vertex shader entry point's interface **must** have either valid or VK_NULL_HANDLE buffers bound

- VUID-vkCmdDrawIndirect-None-04008
  If the `nullDescriptor` feature is not enabled, all vertex input bindings accessed via vertex input variables declared in the vertex shader entry point's interface **must** not be VK_NULL_HANDLE
- VUID-vkCmdDrawIndirect-None-02721
  If `robustBufferAccess` is not enabled, and that pipeline was created without enabling `VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_EXT` for `vertexInputs`, then for a given vertex buffer binding, any attribute data fetched **must** be entirely contained within the corresponding vertex buffer binding, as described in Vertex Input Description
- VUID-vkCmdDrawIndirect-None-07842
  If there is a shader object bound to the `VK_SHADER_STAGE_VERTEX_BIT` stage or the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_PRIMITIVE_TOPOLOGY` dynamic state enabled then vkCmdSetPrimitiveTopologyEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirect-dynamicPrimitiveTopologyUnrestricted-07500
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_PRIMITIVE_TOPOLOGY` dynamic state enabled and the `dynamicPrimitiveTopologyUnrestricted` is `VK_FALSE`, then the `primitiveTopology` parameter of `vkCmdSetPrimitiveTopologyEXT` **must** be of the same topology class as the pipeline VkPipelineInputAssemblyStateCreateInfo::`topology` state
- VUID-vkCmdDrawIndirect-pStrides-04913
  If the bound graphics pipeline was created with the `VK_DYNAMIC_STATE_VERTEX_INPUT_BINDING_STRIDE_EXT` dynamic state enabled, but without the `VK_DYNAMIC_STATE_VERTEX_INPUT_EXT` dynamic state enabled, then vkCmdBindVertexBuffers2EXT **must** have been called and not subsequently invalidated in the current command buffer prior to this draw command, and the `pStrides` parameter of vkCmdBindVertexBuffers2EXT **must** not be `NULL`
- VUID-vkCmdDrawIndirect-None-04914
  If there is a shader object bound to the `VK_SHADER_STAGE_VERTEX_BIT` stage or the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VERTEX_INPUT_EXT` dynamic state enabled then vkCmdSetVertexInputEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this draw command
- VUID-vkCmdDrawIndirect-Input-07939
  If there is a shader object bound to the `VK_SHADER_STAGE_VERTEX_BIT` stage or the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VERTEX_INPUT_EXT` dynamic state enabled then all variables with the `Input` storage class decorated with `Location` in the `Vertex Execution Model OpEntryPoint` **must** contain a location in VkVertexInputAttributeDescription2EXT::`location`
- VUID-vkCmdDrawIndirect-Input-08734
  If there is a shader object bound to the `VK_SHADER_STAGE_VERTEX_BIT` stage or the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VERTEX_INPUT_EXT` dynamic state enabled and either `legacyVertexAttributes` is not enabled or the SPIR-V Type associated with a given `Input` variable of the corresponding `Location` in the `Vertex Execution Model OpEntryPoint` is 64-bit, then the numeric type associated with all `Input`

variables of the corresponding Location in the Vertex Execution Model OpEntryPoint **must** be the same as VkVertexInputAttributeDescription2EXT::format

- VUID-vkCmdDrawIndirect-format-08936
  If there is a shader object bound to the VK_SHADER_STAGE_VERTEX_BIT stage or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VERTEX_INPUT_EXT dynamic state enabled and VkVertexInputAttributeDescription2EXT::format has a 64-bit component, then the scalar width associated with all Input variables of the corresponding Location in the Vertex Execution Model OpEntryPoint **must** be 64-bit

- VUID-vkCmdDrawIndirect-format-08937
  If there is a shader object bound to the VK_SHADER_STAGE_VERTEX_BIT stage or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VERTEX_INPUT_EXT dynamic state enabled and the scalar width associated with a Location decorated Input variable in the Vertex Execution Model OpEntryPoint is 64-bit, then the corresponding VkVertexInputAttributeDescription2EXT::format **must** have a 64-bit component

- VUID-vkCmdDrawIndirect-None-09203
  If there is a shader object bound to the VK_SHADER_STAGE_VERTEX_BIT stage or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VERTEX_INPUT_EXT dynamic state enabled and VkVertexInputAttributeDescription2EXT::format has a 64-bit component, then all Input variables at the corresponding Location in the Vertex Execution Model OpEntryPoint **must** not use components that are not present in the format

- VUID-vkCmdDrawIndirect-None-04875
  If there is a shader object bound to the VK_SHADER_STAGE_VERTEX_BIT stage and the most recent call to vkCmdSetPrimitiveTopologyEXT in the current command buffer set primitiveTopology to VK_PRIMITIVE_TOPOLOGY_PATCH_LIST, or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_PATCH_CONTROL_POINTS_EXT dynamic state enabled then vkCmdSetPatchControlPointsEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirect-None-04879
  If there is a shader object bound to the VK_SHADER_STAGE_VERTEX_BIT stage or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_PRIMITIVE_RESTART_ENABLE dynamic state enabled then vkCmdSetPrimitiveRestartEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirect-None-09637
  If the primitiveTopologyListRestart feature is not enabled, the topology is VK_PRIMITIVE_TOPOLOGY_POINT_LIST, VK_PRIMITIVE_TOPOLOGY_LINE_LIST, VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST, VK_PRIMITIVE_TOPOLOGY_LINE_LIST_WITH_ADJACENCY, or VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST_WITH_ADJACENCY, there is a shader object bound to the VK_SHADER_STAGE_VERTEX_BIT stage or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_PRIMITIVE_RESTART_ENABLE dynamic state enabled then vkCmdSetPrimitiveRestartEnableEXT **must** be VK_FALSE

- VUID-vkCmdDrawIndirect-stage-06481
  The bound graphics pipeline **must** not have been created with the VkPipelineShaderStageCreateInfo::stage member of an element of VkGraphicsPipelineCreateInfo::pStages set to VK_SHADER_STAGE_TASK_BIT_EXT or

VK_SHADER_STAGE_MESH_BIT_EXT

- VUID-vkCmdDrawIndirect-None-08885
  There **must** be no shader object bound to either of the VK_SHADER_STAGE_TASK_BIT_EXT or VK_SHADER_STAGE_MESH_BIT_EXT stages
- VUID-vkCmdDrawIndirect-buffer-02708
  If buffer is non-sparse then it **must** be bound completely and contiguously to a single VkDeviceMemory object
- VUID-vkCmdDrawIndirect-buffer-02709
  buffer **must** have been created with the VK_BUFFER_USAGE_INDIRECT_BUFFER_BIT bit set
- VUID-vkCmdDrawIndirect-offset-02710
  offset **must** be a multiple of 4
- VUID-vkCmdDrawIndirect-commandBuffer-02711
  commandBuffer **must** not be a protected command buffer
- VUID-vkCmdDrawIndirect-drawCount-02718
  If the multiDrawIndirect feature is not enabled, drawCount **must** be 0 or 1
- VUID-vkCmdDrawIndirect-drawCount-02719
  drawCount **must** be less than or equal to VkPhysicalDeviceLimits::maxDrawIndirectCount
- VUID-vkCmdDrawIndirect-drawCount-00476
  If drawCount is greater than 1, stride **must** be a multiple of 4 and **must** be greater than or equal to sizeof(VkDrawIndirectCommand)
- VUID-vkCmdDrawIndirect-drawCount-00487
  If drawCount is equal to 1, (offset + sizeof(VkDrawIndirectCommand)) **must** be less than or equal to the size of buffer
- VUID-vkCmdDrawIndirect-drawCount-00488
  If drawCount is greater than 1, (stride × (drawCount - 1) + offset + sizeof(VkDrawIndirectCommand)) **must** be less than or equal to the size of buffer

## Valid Usage (Implicit)

- VUID-vkCmdDrawIndirect-commandBuffer-parameter
  commandBuffer **must** be a valid VkCommandBuffer handle
- VUID-vkCmdDrawIndirect-buffer-parameter
  buffer **must** be a valid VkBuffer handle
- VUID-vkCmdDrawIndirect-commandBuffer-recording
  commandBuffer **must** be in the recording state
- VUID-vkCmdDrawIndirect-commandBuffer-cmdpool
  The VkCommandPool that commandBuffer was allocated from **must** support graphics operations
- VUID-vkCmdDrawIndirect-renderpass
  This command **must** only be called inside of a render pass instance

- VUID-vkCmdDrawIndirect-videocoding
  This command **must** only be called outside of a video coding scope
- VUID-vkCmdDrawIndirect-commonparent
  Both of `buffer`, and `commandBuffer` **must** have been created, allocated, or retrieved from the same VkDevice

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Inside | Outside | Graphics | Action |

The `VkDrawIndirectCommand` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkDrawIndirectCommand {
    uint32_t    vertexCount;
    uint32_t    instanceCount;
    uint32_t    firstVertex;
    uint32_t    firstInstance;
} VkDrawIndirectCommand;
```

- `vertexCount` is the number of vertices to draw.
- `instanceCount` is the number of instances to draw.
- `firstVertex` is the index of the first vertex to draw.
- `firstInstance` is the instance ID of the first instance to draw.

The members of `VkDrawIndirectCommand` have the same meaning as the similarly named parameters of vkCmdDraw.

## Valid Usage

- VUID-VkDrawIndirectCommand-pNext-09461
  If the bound graphics pipeline state was created with VkPipelineVertexInputDivisorStateCreateInfoKHR in the `pNext` chain of

VkGraphicsPipelineCreateInfo::pVertexInputState, any member of VkPipelineVertexInputDivisorStateCreateInfoKHR::pVertexBindingDivisors has a value other than 1 in divisor, and VkPhysicalDeviceVertexAttributeDivisorPropertiesKHR::supportsNonZeroFirstInstance is VK_FALSE, then firstInstance **must** be 0

- VUID-VkDrawIndirectCommand-None-09462
  If shader objects are used for drawing or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VERTEX_INPUT_EXT dynamic state enabled, any member of the pVertexBindingDescriptions parameter to the vkCmdSetVertexInputEXT call that sets this dynamic state has a value other than 1 in divisor, and VkPhysicalDeviceVertexAttributeDivisorPropertiesKHR::supportsNonZeroFirstInstance is VK_FALSE, then firstInstance **must** be 0

- VUID-VkDrawIndirectCommand-None-00500
  For a given vertex buffer binding, any attribute data fetched **must** be entirely contained within the corresponding vertex buffer binding, as described in Vertex Input Description

- VUID-VkDrawIndirectCommand-firstInstance-00501
  If the drawIndirectFirstInstance feature is not enabled, firstInstance **must** be 0

To record a non-indexed draw call with a draw call count sourced from a buffer, call:

```
// Provided by VK_KHR_draw_indirect_count
void vkCmdDrawIndirectCountKHR(
    VkCommandBuffer                             commandBuffer,
    VkBuffer                                    buffer,
    VkDeviceSize                                offset,
    VkBuffer                                    countBuffer,
    VkDeviceSize                                countBufferOffset,
    uint32_t                                    maxDrawCount,
    uint32_t                                    stride);
```

or the equivalent command

```
// Provided by VK_AMD_draw_indirect_count
void vkCmdDrawIndirectCountAMD(
    VkCommandBuffer                             commandBuffer,
    VkBuffer                                    buffer,
    VkDeviceSize                                offset,
    VkBuffer                                    countBuffer,
    VkDeviceSize                                countBufferOffset,
    uint32_t                                    maxDrawCount,
    uint32_t                                    stride);
```

- commandBuffer is the command buffer into which the command is recorded.
- buffer is the buffer containing draw parameters.
- offset is the byte offset into buffer where parameters begin.

- countBuffer is the buffer containing the draw count.
- countBufferOffset is the byte offset into countBuffer where the draw count begins.
- maxDrawCount specifies the maximum number of draws that will be executed. The actual number of executed draw calls is the minimum of the count specified in countBuffer and maxDrawCount.
- stride is the byte stride between successive sets of draw parameters.

vkCmdDrawIndirectCountKHR behaves similarly to vkCmdDrawIndirect except that the draw count is read by the device from a buffer during execution. The command will read an unsigned 32-bit integer from countBuffer located at countBufferOffset and use this as the draw count.

### Valid Usage

- VUID-vkCmdDrawIndirectCount-magFilter-04553
  If a VkSampler created with magFilter or minFilter equal to VK_FILTER_LINEAR, reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE, and compareEnable equal to VK_FALSE is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT
- VUID-vkCmdDrawIndirectCount-magFilter-09598
  If a VkSampler created with magFilter or minFilter equal to VK_FILTER_LINEAR and reductionMode equal to either VK_SAMPLER_REDUCTION_MODE_MIN or VK_SAMPLER_REDUCTION_MODE_MAX is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT
- VUID-vkCmdDrawIndirectCount-mipmapMode-04770
  If a VkSampler created with mipmapMode equal to VK_SAMPLER_MIPMAP_MODE_LINEAR, reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE, and compareEnable equal to VK_FALSE is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT
- VUID-vkCmdDrawIndirectCount-mipmapMode-09599
  If a VkSampler created with mipmapMode equal to VK_SAMPLER_MIPMAP_MODE_LINEAR and reductionMode equal to either VK_SAMPLER_REDUCTION_MODE_MIN or VK_SAMPLER_REDUCTION_MODE_MAX is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT
- VUID-vkCmdDrawIndirectCount-unnormalizedCoordinates-09635
  If a VkSampler created with unnormalizedCoordinates equal to VK_TRUE is used to sample a VkImageView as a result of this command, then the image view's levelCount and layerCount **must** be 1
- VUID-vkCmdDrawIndirectCount-unnormalizedCoordinates-09636
  If a VkSampler created with unnormalizedCoordinates equal to VK_TRUE is used to sample a VkImageView as a result of this command, then the image view's viewType **must** be VK_IMAGE_VIEW_TYPE_1D or VK_IMAGE_VIEW_TYPE_2D

- VUID-vkCmdDrawIndirectCount-None-06479
  If a VkImageView is sampled with depth comparison, the image view's format features **must** contain VK_FORMAT_FEATURE_2_SAMPLED_IMAGE_DEPTH_COMPARISON_BIT
- VUID-vkCmdDrawIndirectCount-None-02691
  If a VkImageView is accessed using atomic operations as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_STORAGE_IMAGE_ATOMIC_BIT
- VUID-vkCmdDrawIndirectCount-None-07888
  If a VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER descriptor is accessed using atomic operations as a result of this command, then the storage texel buffer's format features **must** contain VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_ATOMIC_BIT
- VUID-vkCmdDrawIndirectCount-None-02692
  If a VkImageView is sampled with VK_FILTER_CUBIC_EXT as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_CUBIC_BIT_EXT
- VUID-vkCmdDrawIndirectCount-None-02693
  If the VK_EXT_filter_cubic extension is not enabled and any VkImageView is sampled with VK_FILTER_CUBIC_EXT as a result of this command, it **must** not have a VkImageViewType of VK_IMAGE_VIEW_TYPE_3D, VK_IMAGE_VIEW_TYPE_CUBE, or VK_IMAGE_VIEW_TYPE_CUBE_ARRAY
- VUID-vkCmdDrawIndirectCount-filterCubic-02694
  Any VkImageView being sampled with VK_FILTER_CUBIC_EXT as a result of this command **must** have a VkImageViewType and format that supports cubic filtering, as specified by VkFilterCubicImageViewImageFormatPropertiesEXT::filterCubic returned by vkGetPhysicalDeviceImageFormatProperties2
- VUID-vkCmdDrawIndirectCount-filterCubicMinmax-02695
  Any VkImageView being sampled with VK_FILTER_CUBIC_EXT with a reduction mode of either VK_SAMPLER_REDUCTION_MODE_MIN or VK_SAMPLER_REDUCTION_MODE_MAX as a result of this command **must** have a VkImageViewType and format that supports cubic filtering together with minmax filtering, as specified by VkFilterCubicImageViewImageFormatPropertiesEXT::filterCubicMinmax returned by vkGetPhysicalDeviceImageFormatProperties2
- VUID-vkCmdDrawIndirectCount-cubicRangeClamp-09212
  If the cubicRangeClamp feature is not enabled, then any VkImageView being sampled with VK_FILTER_CUBIC_EXT as a result of this command **must** not have a VkSamplerReductionModeCreateInfo::reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE_RANGECLAMP_QCOM
- VUID-vkCmdDrawIndirectCount-reductionMode-09213
  Any VkImageView being sampled with a VkSamplerReductionModeCreateInfo::reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE_RANGECLAMP_QCOM as a result of this command **must** sample with VK_FILTER_CUBIC_EXT
- VUID-vkCmdDrawIndirectCount-selectableCubicWeights-09214
  If the selectableCubicWeights feature is not enabled, then any VkImageView being sampled with VK_FILTER_CUBIC_EXT as a result of this command **must** have VkSamplerCubicWeightsCreateInfoQCOM::cubicWeights equal to

VK_CUBIC_FILTER_WEIGHTS_CATMULL_ROM_QCOM

- VUID-vkCmdDrawIndirectCount-flags-02696
  Any VkImage created with a VkImageCreateInfo::flags containing VK_IMAGE_CREATE_CORNER_SAMPLED_BIT_NV sampled as a result of this command **must** only be sampled using a VkSamplerAddressMode of VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_EDGE
- VUID-vkCmdDrawIndirectCount-OpTypeImage-07027
  For any VkImageView being written as a storage image where the image format field of the OpTypeImage is Unknown, the view's format features **must** contain VK_FORMAT_FEATURE_2_STORAGE_WRITE_WITHOUT_FORMAT_BIT
- VUID-vkCmdDrawIndirectCount-OpTypeImage-07028
  For any VkImageView being read as a storage image where the image format field of the OpTypeImage is Unknown, the view's format features **must** contain VK_FORMAT_FEATURE_2_STORAGE_READ_WITHOUT_FORMAT_BIT
- VUID-vkCmdDrawIndirectCount-OpTypeImage-07029
  For any VkBufferView being written as a storage texel buffer where the image format field of the OpTypeImage is Unknown, the view's buffer features **must** contain VK_FORMAT_FEATURE_2_STORAGE_WRITE_WITHOUT_FORMAT_BIT
- VUID-vkCmdDrawIndirectCount-OpTypeImage-07030
  Any VkBufferView being read as a storage texel buffer where the image format field of the OpTypeImage is Unknown then the view's buffer features **must** contain VK_FORMAT_FEATURE_2_STORAGE_READ_WITHOUT_FORMAT_BIT
- VUID-vkCmdDrawIndirectCount-None-08600
  For each set *n* that is statically used by a bound shader, a descriptor set **must** have been bound to *n* at the same pipeline bind point, with a VkPipelineLayout that is compatible for set *n*, with the VkPipelineLayout used to create the current VkPipeline or the VkDescriptorSetLayout array used to create the current VkShaderEXT , as described in Pipeline Layout Compatibility
- VUID-vkCmdDrawIndirectCount-None-08601
  For each push constant that is statically used by a bound shader, a push constant value **must** have been set for the same pipeline bind point, with a VkPipelineLayout that is compatible for push constants, with the VkPipelineLayout used to create the current VkPipeline or the VkDescriptorSetLayout array used to create the current VkShaderEXT , as described in Pipeline Layout Compatibility
- VUID-vkCmdDrawIndirectCount-None-10068
  For each array of resources that is used by a bound shader, the indices used to access members of the array **must** be less than the descriptor count for the identified binding in the descriptor sets used by this command
- VUID-vkCmdDrawIndirectCount-maintenance4-08602
  If the maintenance4 feature is not enabled, then for each push constant that is statically used by a bound shader, a push constant value **must** have been set for the same pipeline bind point, with a VkPipelineLayout that is compatible for push constants, with the VkPipelineLayout used to create the current VkPipeline or the VkDescriptorSetLayout and VkPushConstantRange arrays used to create the current VkShaderEXT , as described in Pipeline Layout Compatibility

- VUID-vkCmdDrawIndirectCount-None-08114
  Descriptors in each bound descriptor set, specified via vkCmdBindDescriptorSets, **must** be valid as described by descriptor validity if they are statically used by the VkPipeline bound to the pipeline bind point used by this command and the bound VkPipeline was not created with `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`
- VUID-vkCmdDrawIndirectCount-None-08115
  If the descriptors used by the VkPipeline bound to the pipeline bind point were specified via vkCmdBindDescriptorSets, the bound VkPipeline **must** have been created without `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`
- VUID-vkCmdDrawIndirectCount-None-08116
  Descriptors in bound descriptor buffers, specified via vkCmdSetDescriptorBufferOffsetsEXT, **must** be valid if they are dynamically used by the VkPipeline bound to the pipeline bind point used by this command and the bound VkPipeline was created with `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`
- VUID-vkCmdDrawIndirectCount-None-08604
  Descriptors in bound descriptor buffers, specified via vkCmdSetDescriptorBufferOffsetsEXT, **must** be valid if they are dynamically used by any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command
- VUID-vkCmdDrawIndirectCount-None-08117
  If the descriptors used by the VkPipeline bound to the pipeline bind point were specified via vkCmdSetDescriptorBufferOffsetsEXT, the bound VkPipeline **must** have been created with `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`
- VUID-vkCmdDrawIndirectCount-None-08119
  If a descriptor is dynamically used with a VkPipeline created with `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`, the descriptor memory **must** be resident
- VUID-vkCmdDrawIndirectCount-None-08605
  If a descriptor is dynamically used with a VkShaderEXT created with a `VkDescriptorSetLayout` that was created with `VK_DESCRIPTOR_SET_LAYOUT_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`, the descriptor memory **must** be resident
- VUID-vkCmdDrawIndirectCount-None-08606
  If the `shaderObject` feature is not enabled, a valid pipeline **must** be bound to the pipeline bind point used by this command
- VUID-vkCmdDrawIndirectCount-None-08608
  If a pipeline is bound to the pipeline bind point used by this command, there **must** not have been any calls to dynamic state setting commands for any state specified statically in the VkPipeline object bound to the pipeline bind point used by this command, since that pipeline was bound
- VUID-vkCmdDrawIndirectCount-None-08609
  If the VkPipeline object bound to the pipeline bind point used by this command or any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a VkSampler object that uses unnormalized coordinates, that sampler **must** not be used to sample from any VkImage with a VkImageView of the type

VK_IMAGE_VIEW_TYPE_3D, VK_IMAGE_VIEW_TYPE_CUBE, VK_IMAGE_VIEW_TYPE_1D_ARRAY, VK_IMAGE_VIEW_TYPE_2D_ARRAY or VK_IMAGE_VIEW_TYPE_CUBE_ARRAY, in any shader stage

- VUID-vkCmdDrawIndirectCount-None-08610
  If the VkPipeline object bound to the pipeline bind point used by this command or any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a VkSampler object that uses unnormalized coordinates, that sampler **must** not be used with any of the SPIR-V OpImageSample* or OpImageSparseSample* instructions with ImplicitLod, Dref or Proj in their name, in any shader stage
- VUID-vkCmdDrawIndirectCount-None-08611
  If the VkPipeline object bound to the pipeline bind point used by this command or any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a VkSampler object that uses unnormalized coordinates, that sampler **must** not be used with any of the SPIR-V OpImageSample* or OpImageSparseSample* instructions that includes a LOD bias or any offset values, in any shader stage
- VUID-vkCmdDrawIndirectCount-None-08607
  If the shaderObject is enabled, either a valid pipeline **must** be bound to the pipeline bind point used by this command, or a valid combination of valid and VK_NULL_HANDLE shader objects **must** be bound to every supported shader stage corresponding to the pipeline bind point used by this command
- VUID-vkCmdDrawIndirectCount-uniformBuffers-06935
  If any stage of the VkPipeline object bound to the pipeline bind point used by this command accesses a uniform buffer, and that stage was created without enabling either VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_EXT or VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_2_EXT for uniformBuffers, and the robustBufferAccess feature is not enabled, that stage **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point
- VUID-vkCmdDrawIndirectCount-None-08612
  If the robustBufferAccess feature is not enabled, and any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a uniform buffer, it **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point
- VUID-vkCmdDrawIndirectCount-storageBuffers-06936
  If any stage of the VkPipeline object bound to the pipeline bind point used by this command accesses a storage buffer, and that stage was created without enabling either VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_EXT or VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_2_EXT for storageBuffers, and the robustBufferAccess feature is not enabled, that stage **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point
- VUID-vkCmdDrawIndirectCount-None-08613
  If the robustBufferAccess feature is not enabled, and any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a storage buffer, it **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point

- VUID-vkCmdDrawIndirectCount-commandBuffer-02707
  If `commandBuffer` is an unprotected command buffer and `protectedNoFault` is not supported, any resource accessed by bound shaders **must** not be a protected resource
- VUID-vkCmdDrawIndirectCount-None-06550
  If a bound shader accesses a VkSampler or VkImageView object that enables sampler $Y'C_BC_R$ conversion, that object **must** only be used with `OpImageSample*` or `OpImageSparseSample*` instructions
- VUID-vkCmdDrawIndirectCount-ConstOffset-06551
  If a bound shader accesses a VkSampler or VkImageView object that enables sampler $Y'C_BC_R$ conversion, that object **must** not use the `ConstOffset` and `Offset` operands
- VUID-vkCmdDrawIndirectCount-viewType-07752
  If a VkImageView is accessed as a result of this command, then the image view's `viewType` **must** match the `Dim` operand of the `OpTypeImage` as described in Compatibility Between SPIR-V Image Dimensions and Vulkan ImageView Types
- VUID-vkCmdDrawIndirectCount-format-07753
  If a VkImageView is accessed as a result of this command, then the numeric type of the image view's `format` and the `Sampled Type` operand of the `OpTypeImage` **must** match
- VUID-vkCmdDrawIndirectCount-OpImageWrite-08795
  If a VkImageView created with a format other than `VK_FORMAT_A8_UNORM_KHR` is accessed using `OpImageWrite` as a result of this command, then the `Type` of the `Texel` operand of that instruction **must** have at least as many components as the image view's format
- VUID-vkCmdDrawIndirectCount-OpImageWrite-08796
  If a VkImageView created with the format `VK_FORMAT_A8_UNORM_KHR` is accessed using `OpImageWrite` as a result of this command, then the `Type` of the `Texel` operand of that instruction **must** have four components
- VUID-vkCmdDrawIndirectCount-OpImageWrite-04469
  If a VkBufferView is accessed using `OpImageWrite` as a result of this command, then the `Type` of the `Texel` operand of that instruction **must** have at least as many components as the buffer view's format
- VUID-vkCmdDrawIndirectCount-SampledType-04470
  If a VkImageView with a VkFormat that has a 64-bit component width is accessed as a result of this command, the `SampledType` of the `OpTypeImage` operand of that instruction **must** have a `Width` of 64
- VUID-vkCmdDrawIndirectCount-SampledType-04471
  If a VkImageView with a VkFormat that has a component width less than 64-bit is accessed as a result of this command, the `SampledType` of the `OpTypeImage` operand of that instruction **must** have a `Width` of 32
- VUID-vkCmdDrawIndirectCount-SampledType-04472
  If a VkBufferView with a VkFormat that has a 64-bit component width is accessed as a result of this command, the `SampledType` of the `OpTypeImage` operand of that instruction **must** have a `Width` of 64
- VUID-vkCmdDrawIndirectCount-SampledType-04473
  If a VkBufferView with a VkFormat that has a component width less than 64-bit is

accessed as a result of this command, the SampledType of the OpTypeImage operand of that instruction **must** have a Width of 32

- VUID-vkCmdDrawIndirectCount-sparseImageInt64Atomics-04474
  If the sparseImageInt64Atomics feature is not enabled, VkImage objects created with the VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT flag **must** not be accessed by atomic instructions through an OpTypeImage with a SampledType with a Width of 64 by this command

- VUID-vkCmdDrawIndirectCount-sparseImageInt64Atomics-04475
  If the sparseImageInt64Atomics feature is not enabled, VkBuffer objects created with the VK_BUFFER_CREATE_SPARSE_RESIDENCY_BIT flag **must** not be accessed by atomic instructions through an OpTypeImage with a SampledType with a Width of 64 by this command

- VUID-vkCmdDrawIndirectCount-OpImageWeightedSampleQCOM-06971
  If OpImageWeightedSampleQCOM is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_WEIGHT_SAMPLED_IMAGE_BIT_QCOM

- VUID-vkCmdDrawIndirectCount-OpImageWeightedSampleQCOM-06972
  If OpImageWeightedSampleQCOM uses a VkImageView as a sample weight image as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_WEIGHT_IMAGE_BIT_QCOM

- VUID-vkCmdDrawIndirectCount-OpImageBoxFilterQCOM-06973
  If OpImageBoxFilterQCOM is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_BOX_FILTER_SAMPLED_BIT_QCOM

- VUID-vkCmdDrawIndirectCount-OpImageBlockMatchSSDQCOM-06974
  If OpImageBlockMatchSSDQCOM is used to read from an VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM

- VUID-vkCmdDrawIndirectCount-OpImageBlockMatchSADQCOM-06975
  If OpImageBlockMatchSADQCOM is used to read from an VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM

- VUID-vkCmdDrawIndirectCount-OpImageBlockMatchSADQCOM-06976
  If OpImageBlockMatchSADQCOM or OpImageBlockMatchSSDQCOM is used to read from a reference image as result of this command, then the specified reference coordinates **must** not fail integer texel coordinate validation

- VUID-vkCmdDrawIndirectCount-OpImageWeightedSampleQCOM-06977
  If OpImageWeightedSampleQCOM, OpImageBoxFilterQCOM, OpImageBlockMatchWindowSSDQCOM, OpImageBlockMatchWindowSADQCOM, OpImageBlockMatchGatherSSDQCOM, OpImageBlockMatchGatherSADQCOM, OpImageBlockMatchSSDQCOM, or OpImageBlockMatchSADQCOM uses a VkSampler as a result of this command, then the sampler **must** have been created with VK_SAMPLER_CREATE_IMAGE_PROCESSING_BIT_QCOM

- VUID-vkCmdDrawIndirectCount-OpImageWeightedSampleQCOM-06978
  If any command other than OpImageWeightedSampleQCOM, OpImageBoxFilterQCOM, OpImageBlockMatchWindowSSDQCOM, OpImageBlockMatchWindowSADQCOM, OpImageBlockMatchGatherSSDQCOM, OpImageBlockMatchGatherSADQCOM,

OpImageBlockMatchSSDQCOM, or OpImageBlockMatchSADQCOM uses a VkSampler as a result of this command, then the sampler **must** not have been created with VK_SAMPLER_CREATE_IMAGE_PROCESSING_BIT_QCOM

- VUID-vkCmdDrawIndirectCount-OpImageBlockMatchWindow-09215
  If a OpImageBlockMatchWindow*QCOM or OpImageBlockMatchGather*QCOM instruction is used to read from an VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM
- VUID-vkCmdDrawIndirectCount-OpImageBlockMatchWindow-09216
  If a OpImageBlockMatchWindow*QCOM or OpImageBlockMatchGather*QCOM instruction is used to read from an VkImageView as a result of this command, then the image view's format **must** be a single-component format
- VUID-vkCmdDrawIndirectCount-OpImageBlockMatchWindow-09217
  If a OpImageBlockMatchWindow*QCOM or OpImageBlockMatchGather*QCOM read from a reference image as result of this command, then the specified reference coordinates **must** not fail integer texel coordinate validation
- VUID-vkCmdDrawIndirectCount-None-07288
  Any shader invocation executed by this command **must** terminate
- VUID-vkCmdDrawIndirectCount-None-09600
  If a descriptor with type equal to any of VK_DESCRIPTOR_TYPE_SAMPLE_WEIGHT_IMAGE_QCOM, VK_DESCRIPTOR_TYPE_BLOCK_MATCH_IMAGE_QCOM, VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE, VK_DESCRIPTOR_TYPE_STORAGE_IMAGE, or VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT is accessed as a result of this command, the image subresource identified by that descriptor **must** be in the image layout identified when the descriptor was written
- VUID-vkCmdDrawIndirectCount-renderPass-02684
  The current render pass **must** be compatible with the renderPass member of the VkGraphicsPipelineCreateInfo structure specified when creating the VkPipeline bound to VK_PIPELINE_BIND_POINT_GRAPHICS
- VUID-vkCmdDrawIndirectCount-subpass-02685
  The subpass index of the current render pass **must** be equal to the subpass member of the VkGraphicsPipelineCreateInfo structure specified when creating the VkPipeline bound to VK_PIPELINE_BIND_POINT_GRAPHICS
- VUID-vkCmdDrawIndirectCount-None-07748
  If any shader statically accesses an input attachment, a valid descriptor **must** be bound to the pipeline via a descriptor set
- VUID-vkCmdDrawIndirectCount-OpTypeImage-07468
  If any shader executed by this pipeline accesses an OpTypeImage variable with a Dim operand of SubpassData, it **must** be decorated with an InputAttachmentIndex that corresponds to a valid input attachment in the current subpass
- VUID-vkCmdDrawIndirectCount-None-07469
  Input attachment views accessed in a subpass **must** be created with the same VkFormat as the corresponding subpass definition, and be created with a VkImageView that is compatible with the attachment referenced by the subpass' pInputAttachments [InputAttachmentIndex] in the currently bound VkFramebuffer as specified by Fragment Input Attachment Compatibility

- VUID-vkCmdDrawIndirectCount-pDepthInputAttachmentIndex-09595
  Input attachment views accessed in a dynamic render pass with a `InputAttachmentIndex` referenced by VkRenderingInputAttachmentIndexInfoKHR, or no `InputAttachmentIndex` if VkRenderingInputAttachmentIndexInfoKHR:`pDepthInputAttachmentIndex` or VkRenderingInputAttachmentIndexInfoKHR:`pStencilInputAttachmentIndex` are `NULL`, **must** be created with a VkImageView that is compatible with the corresponding color, depth, or stencil attachment in VkRenderingInfo
- VUID-vkCmdDrawIndirectCount-pDepthInputAttachmentIndex-09596
  Input attachment views accessed in a dynamic render pass via a shader object **must** have an `InputAttachmentIndex` if both VkRenderingInputAttachmentIndexInfoKHR:`pDepthInputAttachmentIndex` and VkRenderingInputAttachmentIndexInfoKHR:`pStencilInputAttachmentIndex` are non-`NULL`
- VUID-vkCmdDrawIndirectCount-InputAttachmentIndex-09597
  If an input attachment view accessed in a dynamic render pass via a shader object has an `InputAttachmentIndex`, the `InputAttachmentIndex` **must** match an index in VkRenderingInputAttachmentIndexInfoKHR
- VUID-vkCmdDrawIndirectCount-None-06537
  Memory backing image subresources used as attachments in the current render pass **must** not be written in any way other than as an attachment by this command
- VUID-vkCmdDrawIndirectCount-None-09000
  If a color attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass, it is not in the `VK_IMAGE_LAYOUT_ATTACHMENT_FEEDBACK_LOOP_OPTIMAL_EXT` image layout, and either:
  - the `VK_PIPELINE_CREATE_COLOR_ATTACHMENT_FEEDBACK_LOOP_BIT_EXT` is set on the currently bound pipeline or
  - the last call to vkCmdSetAttachmentFeedbackLoopEnableEXT included `VK_IMAGE_ASPECT_COLOR_BIT` and
    - there is no currently bound graphics pipeline or
    - the currently bound graphics pipeline was created with `VK_DYNAMIC_STATE_ATTACHMENT_FEEDBACK_LOOP_ENABLE_EXT`

  it **must** not be accessed in any way other than as an attachment by this command
- VUID-vkCmdDrawIndirectCount-None-09001
  If a depth attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass, it is not in the `VK_IMAGE_LAYOUT_ATTACHMENT_FEEDBACK_LOOP_OPTIMAL_EXT` image layout, and either:
  - the `VK_PIPELINE_CREATE_DEPTH_STENCIL_ATTACHMENT_FEEDBACK_LOOP_BIT_EXT` is set on the currently bound pipeline or
  - the last call to vkCmdSetAttachmentFeedbackLoopEnableEXT included `VK_IMAGE_ASPECT_DEPTH_BIT` and
    - there is no currently bound graphics pipeline or
    - the currently bound graphics pipeline was created with

VK_DYNAMIC_STATE_ATTACHMENT_FEEDBACK_LOOP_ENABLE_EXT

it **must** not be accessed in any way other than as an attachment by this command

- VUID-vkCmdDrawIndirectCount-None-09002
  If a stencil attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass, it is not in the VK_IMAGE_LAYOUT_ATTACHMENT_FEEDBACK_LOOP_OPTIMAL_EXT image layout, and either:
  - the VK_PIPELINE_CREATE_DEPTH_STENCIL_ATTACHMENT_FEEDBACK_LOOP_BIT_EXT is set on the currently bound pipeline or
  - the last call to vkCmdSetAttachmentFeedbackLoopEnableEXT included VK_IMAGE_ASPECT_STENCIL_BIT and
    - there is no currently bound graphics pipeline or
    - the currently bound graphics pipeline was created with VK_DYNAMIC_STATE_ATTACHMENT_FEEDBACK_LOOP_ENABLE_EXT

  it **must** not be accessed in any way other than as an attachment by this command

- VUID-vkCmdDrawIndirectCount-None-09003
  If an attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass, it **must** not be accessed in any way other than as an attachment, storage image, or sampled image by this command
- VUID-vkCmdDrawIndirectCount-None-06539
  If any previously recorded command in the current subpass accessed an image subresource used as an attachment in this subpass in any way other than as an attachment, this command **must** not write to that image subresource as an attachment
- VUID-vkCmdDrawIndirectCount-None-06886
  If the current render pass instance uses a depth/stencil attachment with a read-only layout for the depth aspect, depth writes **must** be disabled
- VUID-vkCmdDrawIndirectCount-None-06887
  If the current render pass instance uses a depth/stencil attachment with a read-only layout for the stencil aspect, both front and back writeMask are not zero, and stencil test is enabled, all stencil ops **must** be VK_STENCIL_OP_KEEP
- VUID-vkCmdDrawIndirectCount-None-07831
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT dynamic state enabled then vkCmdSetViewport **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectCount-None-07832
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SCISSOR dynamic state enabled then vkCmdSetScissor **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectCount-None-07833
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_WIDTH dynamic state enabled then vkCmdSetLineWidth **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirectCount-None-08617
  If a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetPolygonModeEXT in the current command buffer set polygonMode to VK_POLYGON_MODE_LINE, vkCmdSetLineWidth **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectCount-None-08618
  If a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetPrimitiveTopologyEXT in the current command buffer set primitiveTopology to any line topology, vkCmdSetLineWidth **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectCount-None-08619
  If a shader object that outputs line primitives is bound to the VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT or VK_SHADER_STAGE_GEOMETRY_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, vkCmdSetLineWidth **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectCount-None-07834
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_BIAS dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of depthBiasEnable is VK_TRUE, then vkCmdSetDepthBounds or vkCmdSetDepthBias2EXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectCount-None-07835
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_BLEND_CONSTANTS dynamic state enabled then vkCmdSetBlendConstants **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectCount-None-08621
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetColorBlendEnableEXT in the current command buffer set any element of pColorBlendEnables to VK_TRUE, and the most recent call to vkCmdSetColorBlendEquationEXT in the current command buffer set the same element of pColorBlendEquations to a VkColorBlendEquationEXT structure with any VkBlendFactor member with a value of VK_BLEND_FACTOR_CONSTANT_COLOR, VK_BLEND_FACTOR_ONE_MINUS_CONSTANT_COLOR, VK_BLEND_FACTOR_CONSTANT_ALPHA, or VK_BLEND_FACTOR_ONE_MINUS_CONSTANT_ALPHA, vkCmdSetBlendConstants **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirectCount-None-07836
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_BOUNDS dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of depthBoundsTestEnable is VK_TRUE, then vkCmdSetDepthBounds **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectCount-None-07837
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_STENCIL_COMPARE_MASK dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of stencilTestEnable is VK_TRUE, then vkCmdSetStencilCompareMask **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectCount-None-07838
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_STENCIL_WRITE_MASK dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of stencilTestEnable is VK_TRUE, then vkCmdSetStencilWriteMask **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectCount-None-07839
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_STENCIL_REFERENCE dynamic state enabled, the current value of and rasterizerDiscardEnable is VK_FALSE, the current value of stencilTestEnable is VK_TRUE, then vkCmdSetStencilReference **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectCount-maxMultiviewInstanceIndex-02688
  If the draw is recorded in a render pass instance with multiview enabled, the maximum instance index **must** be less than or equal to VkPhysicalDeviceMultiviewProperties::maxMultiviewInstanceIndex
- VUID-vkCmdDrawIndirectCount-sampleLocationsEnable-02689
  If the bound graphics pipeline was created with VkPipelineSampleLocationsStateCreateInfoEXT::sampleLocationsEnable set to VK_TRUE and the current subpass has a depth/stencil attachment, then that attachment **must** have been created with the VK_IMAGE_CREATE_SAMPLE_LOCATIONS_COMPATIBLE_DEPTH_BIT_EXT bit set
- VUID-vkCmdDrawIndirectCount-None-06666
  If the VK_EXT_sample_locations extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of sampleLocationsEnable is VK_TRUE, then vkCmdSetSampleLocationsEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectCount-None-07840

If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_CULL_MODE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetCullModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirectCount-None-07841
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_FRONT_FACE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetFrontFaceEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectCount-None-07843
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_TEST_ENABLE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, vkCmdSetDepthTestEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectCount-None-07844
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_WRITE_ENABLE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetDepthWriteEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectCount-None-07845
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_COMPARE_OP dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of depthTestEnable is VK_TRUE, then vkCmdSetDepthCompareOpEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectCount-None-07846
  If the depthBounds feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_BOUNDS_TEST_ENABLE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetDepthBoundsTestEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectCount-None-07847
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_STENCIL_TEST_ENABLE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetStencilTestEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectCount-None-07848
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which

was created with the VK_DYNAMIC_STATE_STENCIL_OP dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, the current value of stencilTestEnable is VK_TRUE, then vkCmdSetStencilOpEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirectCount-viewportCount-03417
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, but not the VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT dynamic state enabled, then vkCmdSetViewportWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the viewportCount parameter of vkCmdSetViewportWithCountEXT **must** match the VkPipelineViewportStateCreateInfo::scissorCount of the pipeline
- VUID-vkCmdDrawIndirectCount-scissorCount-03418
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT dynamic state enabled, but not the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, then vkCmdSetScissorWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the scissorCount parameter of vkCmdSetScissorWithCountEXT **must** match the VkPipelineViewportStateCreateInfo::viewportCount of the pipeline
- VUID-vkCmdDrawIndirectCount-viewportCount-03419
  If the bound graphics pipeline state was created with both the VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT and VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic states enabled then both vkCmdSetViewportWithCountEXT and vkCmdSetScissorWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the viewportCount parameter of vkCmdSetViewportWithCountEXT **must** match the scissorCount parameter of vkCmdSetScissorWithCountEXT
- VUID-vkCmdDrawIndirectCount-None-08635
  If a shader object is bound to any graphics stage, then both vkCmdSetViewportWithCountEXT and vkCmdSetScissorWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the viewportCount parameter of vkCmdSetViewportWithCountEXT **must** match the scissorCount parameter of vkCmdSetScissorWithCountEXT
- VUID-vkCmdDrawIndirectCount-viewportCount-04137
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, but not the VK_DYNAMIC_STATE_VIEWPORT_W_SCALING_NV dynamic state enabled, then the bound graphics pipeline **must** have been created with VkPipelineViewportWScalingStateCreateInfoNV::viewportCount greater or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawIndirectCount-viewportCount-04138
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT and VK_DYNAMIC_STATE_VIEWPORT_W_SCALING_NV dynamic states enabled then the viewportCount parameter in the last call to

vkCmdSetViewportWScalingNV **must** be greater than or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT

- VUID-vkCmdDrawIndirectCount-None-09232
  If the VK_NV_clip_space_w_scaling extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetViewportWScalingEnableNV in the current command buffer set viewportWScalingEnable to VK_TRUE, then vkCmdSetViewportWScalingNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectCount-None-08636
  If the VK_NV_clip_space_w_scaling extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetViewportWScalingEnableNV in the current command buffer set viewportWScalingEnable to VK_TRUE, then the viewportCount parameter in the last call to vkCmdSetViewportWScalingNV **must** be greater than or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawIndirectCount-viewportCount-04139
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, but not the VK_DYNAMIC_STATE_VIEWPORT_SHADING_RATE_PALETTE_NV dynamic state enabled, then the bound graphics pipeline **must** have been created with VkPipelineViewportShadingRateImageStateCreateInfoNV::viewportCount greater or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawIndirectCount-viewportCount-04140
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT and VK_DYNAMIC_STATE_VIEWPORT_SHADING_RATE_PALETTE_NV dynamic states enabled then the viewportCount parameter in the last call to vkCmdSetViewportShadingRatePaletteNV **must** be greater than or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawIndirectCount-shadingRateImage-09233
  If the shadingRateImage feature is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetCoarseSampleOrderNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectCount-shadingRateImage-09234
  If the shadingRateImage feature is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetShadingRateImageEnableNV in the current command buffer set shadingRateImageEnable to VK_TRUE, then vkCmdSetViewportShadingRatePaletteNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectCount-None-08637
  If the shadingRateImage feature is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current

command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetShadingRateImageEnableNV in the current command buffer set shadingRateImageEnable to VK_TRUE, then the viewportCount parameter in the last call to vkCmdSetViewportShadingRatePaletteNV **must** be greater than or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT

- VUID-vkCmdDrawIndirectCount-VkPipelineVieportCreateInfo-04141
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled and a VkPipelineViewportSwizzleStateCreateInfoNV structure chained from VkPipelineViewportStateCreateInfo, then the bound graphics pipeline **must** have been created with VkPipelineViewportSwizzleStateCreateInfoNV::viewportCount greater or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawIndirectCount-VkPipelineVieportCreateInfo-04142
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled and a VkPipelineViewportExclusiveScissorStateCreateInfoNV structure chained from VkPipelineViewportStateCreateInfo, then the bound graphics pipeline **must** have been created with VkPipelineViewportExclusiveScissorStateCreateInfoNV::exclusiveScissorCount greater or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawIndirectCount-None-07878
  If the exclusiveScissor feature is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_EXCLUSIVE_SCISSOR_ENABLE_NV dynamic state enabled, then vkCmdSetExclusiveScissorEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectCount-None-07879
  If the exclusiveScissor feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_EXCLUSIVE_SCISSOR_NV dynamic state enabled, and the most recent call to vkCmdSetExclusiveScissorEnableNV in the current command buffer set any element of pExclusiveScissorEnables to VK_TRUE, then vkCmdSetExclusiveScissorNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectCount-None-04876
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_RASTERIZER_DISCARD_ENABLE dynamic state enabled, then vkCmdSetRasterizerDiscardEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectCount-None-04877
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_BIAS_ENABLE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetDepthBiasEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirectCount-logicOp-04878
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_LOGIC_OP_EXT dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of logicOpEnable is VK_TRUE, then vkCmdSetLogicOpEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectCount-primitiveFragmentShadingRateWithMultipleViewports-04552
  If the primitiveFragmentShadingRateWithMultipleViewports limit is not supported, the bound graphics pipeline was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, and any of the shader stages of the bound graphics pipeline write to the PrimitiveShadingRateKHR built-in, then vkCmdSetViewportWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the viewportCount parameter of vkCmdSetViewportWithCountEXT **must** be 1
- VUID-vkCmdDrawIndirectCount-primitiveFragmentShadingRateWithMultipleViewports-08642
  If the primitiveFragmentShadingRateWithMultipleViewports limit is not supported, and any shader object bound to a graphics stage writes to the PrimitiveShadingRateKHR built-in, then vkCmdSetViewportWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the viewportCount parameter of vkCmdSetViewportWithCountEXT **must** be 1
- VUID-vkCmdDrawIndirectCount-blendEnable-04727
  If rasterization is not disabled in the bound graphics pipeline, then for each color attachment in the subpass, if the corresponding image view's format features do not contain VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT, then the blendEnable member of the corresponding element of the pAttachments member of pColorBlendState **must** be VK_FALSE
- VUID-vkCmdDrawIndirectCount-None-08643
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then for each color attachment in the render pass, if the corresponding image view's format features do not contain VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT, then the corresponding member of pColorBlendEnables in the most recent call to vkCmdSetColorBlendEnableEXT in the current command buffer that affected that attachment index **must** have been VK_FALSE
- VUID-vkCmdDrawIndirectCount-multisampledRenderToSingleSampled-07284
  If rasterization is not disabled in the bound graphics pipeline, and none of the following is enabled:
  - the VK_AMD_mixed_attachment_samples extension
  - the VK_NV_framebuffer_mixed_samples extension
  - the multisampledRenderToSingleSampled feature

  then rasterizationSamples for the currently bound graphics pipeline **must** be the same as the current subpass color and/or depth/stencil attachments

- VUID-vkCmdDrawIndirectCount-None-08644
  If a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE`, and none of the following is enabled:

  - the `VK_AMD_mixed_attachment_samples` extension
  - the `VK_NV_framebuffer_mixed_samples` extension
  - the `multisampledRenderToSingleSampled` feature

  then the most recent call to vkCmdSetRasterizationSamplesEXT in the current command buffer **must** have set `rasterizationSamples` to be the same as the number of samples for the current render pass color and/or depth/stencil attachments

- VUID-vkCmdDrawIndirectCount-None-08876
  If a shader object is bound to any graphics stage, the current render pass instance **must** have been begun with vkCmdBeginRenderingKHR

- VUID-vkCmdDrawIndirectCount-imageView-06172
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the `imageView` member of `pDepthAttachment` is not VK_NULL_HANDLE, and the `layout` member of `pDepthAttachment` is `VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL`, this command **must** not write any values to the depth attachment

- VUID-vkCmdDrawIndirectCount-imageView-06173
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the `imageView` member of `pStencilAttachment` is not VK_NULL_HANDLE, and the `layout` member of `pStencilAttachment` is `VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL`, this command **must** not write any values to the stencil attachment

- VUID-vkCmdDrawIndirectCount-imageView-06174
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the `imageView` member of `pDepthAttachment` is not VK_NULL_HANDLE, and the `layout` member of `pDepthAttachment` is `VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_STENCIL_ATTACHMENT_OPTIMAL`, this command **must** not write any values to the depth attachment

- VUID-vkCmdDrawIndirectCount-imageView-06175
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the `imageView` member of `pStencilAttachment` is not VK_NULL_HANDLE, and the `layout` member of `pStencilAttachment` is `VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_STENCIL_READ_ONLY_OPTIMAL`, this command **must** not write any values to the stencil attachment

- VUID-vkCmdDrawIndirectCount-imageView-06176
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the `imageView` member of `pDepthAttachment` is not VK_NULL_HANDLE, and the `layout` member of `pDepthAttachment` is `VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL`, this command **must** not write any values to the depth attachment

- VUID-vkCmdDrawIndirectCount-imageView-06177
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the `imageView` member of `pStencilAttachment` is not VK_NULL_HANDLE, and the `layout` member of `pStencilAttachment` is `VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL`, this

command **must** not write any values to the stencil attachment

- VUID-vkCmdDrawIndirectCount-viewMask-06178
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the currently bound graphics pipeline **must** have been created with a VkPipelineRenderingCreateInfo::viewMask equal to VkRenderingInfo::viewMask

- VUID-vkCmdDrawIndirectCount-colorAttachmentCount-06179
  If the dynamicRenderingUnusedAttachments feature is not enabled and the current render pass instance was begun with vkCmdBeginRenderingKHR, the currently bound graphics pipeline **must** have been created with a VkPipelineRenderingCreateInfo::colorAttachmentCount equal to VkRenderingInfo::colorAttachmentCount

- VUID-vkCmdDrawIndirectCount-dynamicRenderingUnusedAttachments-08910
  If the dynamicRenderingUnusedAttachments feature is not enabled, and the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingInfo::colorAttachmentCount greater than 0, then each element of the VkRenderingInfo::pColorAttachments array with an imageView not equal to VK_NULL_HANDLE **must** have been created with a VkFormat equal to the corresponding element of VkPipelineRenderingCreateInfo::pColorAttachmentFormats used to create the currently bound graphics pipeline

- VUID-vkCmdDrawIndirectCount-dynamicRenderingUnusedAttachments-08912
  If the dynamicRenderingUnusedAttachments feature is not enabled, and the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingInfo::colorAttachmentCount greater than 0, then each element of the VkRenderingInfo::pColorAttachments array with an imageView equal to VK_NULL_HANDLE **must** have the corresponding element of VkPipelineRenderingCreateInfo::pColorAttachmentFormats used to create the currently bound pipeline equal to VK_FORMAT_UNDEFINED

- VUID-vkCmdDrawIndirectCount-dynamicRenderingUnusedAttachments-08911
  If the dynamicRenderingUnusedAttachments feature is enabled, and the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingInfo::colorAttachmentCount greater than 0, then each element of the VkRenderingInfo::pColorAttachments array with an imageView not equal to VK_NULL_HANDLE **must** have been created with a VkFormat equal to the corresponding element of VkPipelineRenderingCreateInfo::pColorAttachmentFormats used to create the currently bound graphics pipeline, or the corresponding element of VkPipelineRenderingCreateInfo::pColorAttachmentFormats, if it exists, **must** be VK_FORMAT_UNDEFINED

- VUID-vkCmdDrawIndirectCount-colorAttachmentCount-09362
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, with a VkRenderingInfo::colorAttachmentCount equal to 1, there is no shader object bound to any graphics stage, and a color attachment with a resolve mode of VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID, each element of the VkRenderingInfo::pColorAttachments array with a resolveImageView not equal to VK_NULL_HANDLE **must** have been created with an image created with a VkExternalFormatANDROID::externalFormat value equal to the VkExternalFormatANDROID::externalFormat value used to create the currently bound graphics pipeline

- VUID-vkCmdDrawIndirectCount-None-09363
  If there is no shader object bound to any graphics stage, the current render pass instance was begun with vkCmdBeginRenderingKHR and a VkRenderingInfo::colorAttachmentCount equal to 1, and a color attachment with a resolve mode of VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID, each element of the VkRenderingInfo::pColorAttachments array with a imageView not equal to VK_NULL_HANDLE **must** have been created with an image created with a VkExternalFormatANDROID::externalFormat value equal to the VkExternalFormatANDROID::externalFormat value used to create the currently bound graphics pipeline
- VUID-vkCmdDrawIndirectCount-None-09364
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is no shader object bound to any graphics stage, and the currently bound graphics pipeline was created with a non-zero VkExternalFormatANDROID::externalFormat value and with the VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT dynamic state enabled, then vkCmdSetColorBlendEnableEXT **must** have set the blend enable to VK_FALSE prior to this drawing command
- VUID-vkCmdDrawIndirectCount-None-09365
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is no shader object bound to any graphics stage, and the currently bound graphics pipeline was created with a non-zero VkExternalFormatANDROID::externalFormat value and with the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT dynamic state enabled, then vkCmdSetRasterizationSamplesEXT **must** have set rasterizationSamples to VK_SAMPLE_COUNT_1_BIT prior to this drawing command
- VUID-vkCmdDrawIndirectCount-None-09366
  If there is a shader object bound to any graphics stage, and the current render pass includes a color attachment that uses the VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID resolve mode, then vkCmdSetColorBlendEnableEXT **must** have set blend enable to VK_FALSE prior to this drawing command
- VUID-vkCmdDrawIndirectCount-rasterizationSamples-09367
  If there is a shader object bound to any graphics stage, and the current render pass includes a color attachment that uses the VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID resolve mode, then vkCmdSetRasterizationSamplesEXT **must** have set rasterizationSamples to VK_SAMPLE_COUNT_1_BIT prior to this drawing command
- VUID-vkCmdDrawIndirectCount-None-09368
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is no shader object bound to any graphics stage, and the currently bound graphics pipeline was created with a non-zero VkExternalFormatANDROID::externalFormat value and with the VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR dynamic state enabled, then vkCmdSetFragmentShadingRateKHR **must** have set pFragmentSize->width to 1 prior to this drawing command
- VUID-vkCmdDrawIndirectCount-None-09369
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is

no shader object bound to any graphics stage, and the currently bound graphics pipeline was created with a non-zero VkExternalFormatANDROID::externalFormat value and with the VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR dynamic state enabled, then vkCmdSetFragmentShadingRateKHR **must** have set pFragmentSize->height to 1 prior to this drawing command

- VUID-vkCmdDrawIndirectCount-pFragmentSize-09370
  If there is a shader object bound to any graphics stage, and the current render pass includes a color attachment that uses the VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID resolve mode, then vkCmdSetFragmentShadingRateKHR **must** have set pFragmentSize->width to 1 prior to this drawing command
- VUID-vkCmdDrawIndirectCount-pFragmentSize-09371
  If there is a shader object bound to any graphics stage, and the current render pass includes a color attachment that uses the VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID resolve mode, then vkCmdSetFragmentShadingRateKHR **must** have set pFragmentSize->height to 1 prior to this drawing command
- VUID-vkCmdDrawIndirectCount-None-07749
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_WRITE_ENABLE_EXT dynamic state enabled then vkCmdSetColorWriteEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectCount-None-08646
  If the colorWriteEnable feature is enabled on the device, and a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetColorWriteEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectCount-attachmentCount-07750
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_WRITE_ENABLE_EXT dynamic state enabled then the attachmentCount parameter of vkCmdSetColorWriteEnableEXT **must** be greater than or equal to the VkPipelineColorBlendStateCreateInfo::attachmentCount of the currently bound graphics pipeline
- VUID-vkCmdDrawIndirectCount-None-08647
  If the colorWriteEnable feature is enabled on the device, and a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then the attachmentCount parameter of most recent call to vkCmdSetColorWriteEnableEXT in the current command buffer **must** be greater than or equal to the number of color attachments in the current render pass instance
- VUID-vkCmdDrawIndirectCount-None-07751
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_DISCARD_RECTANGLE_EXT dynamic state enabled then

vkCmdSetDiscardRectangleEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command for each discard rectangle in VkPipelineDiscardRectangleStateCreateInfoEXT::discardRectangleCount

- VUID-vkCmdDrawIndirectCount-None-07880
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_DISCARD_RECTANGLE_ENABLE_EXT dynamic state enabled then vkCmdSetDiscardRectangleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectCount-rasterizerDiscardEnable-09236
  If the VK_EXT_discard_rectangles extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetDiscardRectangleEnableEXT in the current command buffer set discardRectangleEnable to VK_TRUE, then vkCmdSetDiscardRectangleEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectCount-None-08648
  If the VK_EXT_discard_rectangles extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetDiscardRectangleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectCount-None-07881
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_DISCARD_RECTANGLE_MODE_EXT dynamic state enabled then vkCmdSetDiscardRectangleModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectCount-None-08649
  If the VK_EXT_discard_rectangles extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetDiscardRectangleEnableEXT in the current command buffer set discardRectangleEnable to VK_TRUE, then vkCmdSetDiscardRectangleModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectCount-dynamicRenderingUnusedAttachments-08913
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is not enabled, and VkRenderingInfo::pDepthAttachment->imageView was VK_NULL_HANDLE, the value of VkPipelineRenderingCreateInfo::depthAttachmentFormat used to create the currently bound graphics pipeline **must** be equal to VK_FORMAT_UNDEFINED
- VUID-vkCmdDrawIndirectCount-dynamicRenderingUnusedAttachments-08914
  If current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is not enabled, and VkRenderingInfo::pDepthAttachment->imageView was not VK_NULL_HANDLE, the value of

VkPipelineRenderingCreateInfo::depthAttachmentFormat used to create the currently bound graphics pipeline **must** be equal to the VkFormat used to create VkRenderingInfo::pDepthAttachment->imageView

- VUID-vkCmdDrawIndirectCount-dynamicRenderingUnusedAttachments-08915
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is enabled, VkRenderingInfo::pDepthAttachment->imageView was not VK_NULL_HANDLE, and the value of VkPipelineRenderingCreateInfo::depthAttachmentFormat used to create the currently bound graphics pipeline was not equal to the VkFormat used to create VkRenderingInfo::pDepthAttachment->imageView, the value of the format **must** be VK_FORMAT_UNDEFINED
- VUID-vkCmdDrawIndirectCount-dynamicRenderingUnusedAttachments-08916
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is not enabled, and VkRenderingInfo::pStencilAttachment->imageView was VK_NULL_HANDLE, the value of VkPipelineRenderingCreateInfo::stencilAttachmentFormat used to create the currently bound graphics pipeline **must** be equal to VK_FORMAT_UNDEFINED
- VUID-vkCmdDrawIndirectCount-dynamicRenderingUnusedAttachments-08917
  If current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is not enabled, and VkRenderingInfo::pStencilAttachment->imageView was not VK_NULL_HANDLE, the value of VkPipelineRenderingCreateInfo::stencilAttachmentFormat used to create the currently bound graphics pipeline **must** be equal to the VkFormat used to create VkRenderingInfo::pStencilAttachment->imageView
- VUID-vkCmdDrawIndirectCount-dynamicRenderingUnusedAttachments-08918
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is enabled, VkRenderingInfo::pStencilAttachment->imageView was not VK_NULL_HANDLE, and the value of VkPipelineRenderingCreateInfo::stencilAttachmentFormat used to create the currently bound graphics pipeline was not equal to the VkFormat used to create VkRenderingInfo::pStencilAttachment->imageView, the value of the format **must** be VK_FORMAT_UNDEFINED
- VUID-vkCmdDrawIndirectCount-imageView-06183
  If the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingFragmentShadingRateAttachmentInfoKHR::imageView was not VK_NULL_HANDLE, the currently bound graphics pipeline **must** have been created with VK_PIPELINE_CREATE_RENDERING_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR
- VUID-vkCmdDrawIndirectCount-imageView-06184
  If the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingFragmentDensityMapAttachmentInfoEXT::imageView was not VK_NULL_HANDLE, the currently bound graphics pipeline **must** have been created with VK_PIPELINE_CREATE_RENDERING_FRAGMENT_DENSITY_MAP_ATTACHMENT_BIT_EXT
- VUID-vkCmdDrawIndirectCount-colorAttachmentCount-06185
  If the currently bound pipeline was created with a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and the current render pass instance was begun with vkCmdBeginRenderingKHR with a VkRenderingInfo::colorAttachmentCount parameter greater than 0, then each element of the VkRenderingInfo::pColorAttachments

array with a imageView not equal to VK_NULL_HANDLE **must** have been created with a sample count equal to the corresponding element of the pColorAttachmentSamples member of VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV used to create the currently bound graphics pipeline

- VUID-vkCmdDrawIndirectCount-pDepthAttachment-06186
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the currently bound pipeline was created with a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and VkRenderingInfo::pDepthAttachment->imageView was not VK_NULL_HANDLE, the value of the depthStencilAttachmentSamples member of VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV used to create the currently bound graphics pipeline **must** be equal to the sample count used to create VkRenderingInfo::pDepthAttachment->imageView
- VUID-vkCmdDrawIndirectCount-pStencilAttachment-06187
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the currently bound pipeline was created with a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and VkRenderingInfo::pStencilAttachment->imageView was not VK_NULL_HANDLE, the value of the depthStencilAttachmentSamples member of VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV used to create the currently bound graphics pipeline **must** be equal to the sample count used to create VkRenderingInfo::pStencilAttachment->imageView
- VUID-vkCmdDrawIndirectCount-multisampledRenderToSingleSampled-07285
  If the currently bound pipeline was created without a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and the multisampledRenderToSingleSampled feature is not enabled, and the current render pass instance was begun with vkCmdBeginRenderingKHR with a VkRenderingInfo::colorAttachmentCount parameter greater than 0, then each element of the VkRenderingInfo::pColorAttachments array with a imageView not equal to VK_NULL_HANDLE **must** have been created with a sample count equal to the value of rasterizationSamples for the currently bound graphics pipeline
- VUID-vkCmdDrawIndirectCount-multisampledRenderToSingleSampled-07286
  If the currently bound pipeline was created without a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and the multisampledRenderToSingleSampled feature is not enabled, and VkRenderingInfo::pDepthAttachment->imageView was not VK_NULL_HANDLE, the value of rasterizationSamples for the currently bound graphics pipeline **must** be equal to the sample count used to create VkRenderingInfo::pDepthAttachment->imageView
- VUID-vkCmdDrawIndirectCount-multisampledRenderToSingleSampled-07287
  If the currently bound pipeline was created without a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and the multisampledRenderToSingleSampled feature is not enabled, and VkRenderingInfo::pStencilAttachment->imageView was not VK_NULL_HANDLE, the value of rasterizationSamples for the currently bound graphics pipeline **must** be equal to the sample count used to create VkRenderingInfo::pStencilAttachment->imageView
- VUID-vkCmdDrawIndirectCount-pNext-07935
  If this command has been called inside a render pass instance started with

vkCmdBeginRenderingKHR, and the pNext chain of VkRenderingInfo includes a VkMultisampledRenderToSingleSampledInfoEXT structure with multisampledRenderToSingleSampledEnable equal to VK_TRUE, then the value of rasterizationSamples for the currently bound graphics pipeline **must** be equal to VkMultisampledRenderToSingleSampledInfoEXT::rasterizationSamples

- VUID-vkCmdDrawIndirectCount-renderPass-06198
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the currently bound pipeline **must** have been created with a VkGraphicsPipelineCreateInfo::renderPass equal to VK_NULL_HANDLE
- VUID-vkCmdDrawIndirectCount-pColorAttachments-08963
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is a graphics pipeline bound with a fragment shader that statically writes to a color attachment, the color write mask is not zero, color writes are enabled, and the corresponding element of the VkRenderingInfo::pColorAttachments->imageView was not VK_NULL_HANDLE, then the corresponding element of VkPipelineRenderingCreateInfo::pColorAttachmentFormats used to create the pipeline **must** not be VK_FORMAT_UNDEFINED
- VUID-vkCmdDrawIndirectCount-pDepthAttachment-08964
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is a graphics pipeline bound, depth test is enabled, depth write is enabled, and the VkRenderingInfo::pDepthAttachment->imageView was not VK_NULL_HANDLE, then the VkPipelineRenderingCreateInfo::depthAttachmentFormat used to create the pipeline **must** not be VK_FORMAT_UNDEFINED
- VUID-vkCmdDrawIndirectCount-pStencilAttachment-08965
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is a graphics pipeline bound, stencil test is enabled and the VkRenderingInfo::pStencilAttachment->imageView was not VK_NULL_HANDLE, then the VkPipelineRenderingCreateInfo::stencilAttachmentFormat used to create the pipeline **must** not be VK_FORMAT_UNDEFINED
- VUID-vkCmdDrawIndirectCount-primitivesGeneratedQueryWithRasterizerDiscard-06708
  If the primitivesGeneratedQueryWithRasterizerDiscard feature is not enabled and the VK_QUERY_TYPE_PRIMITIVES_GENERATED_EXT query is active, rasterization discard **must** not be enabled
- VUID-vkCmdDrawIndirectCount-primitivesGeneratedQueryWithNonZeroStreams-06709
  If the primitivesGeneratedQueryWithNonZeroStreams feature is not enabled and the VK_QUERY_TYPE_PRIMITIVES_GENERATED_EXT query is active, the bound graphics pipeline **must** not have been created with a non-zero value in VkPipelineRasterizationStateStreamCreateInfoEXT::rasterizationStream
- VUID-vkCmdDrawIndirectCount-None-07619
  If a shader object is bound to the VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_TESSELLATION_DOMAIN_ORIGIN_EXT dynamic state enabled, then vkCmdSetTessellationDomainOriginEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectCount-None-07620
  If the depthClamp feature is enabled, a shader object is bound to any graphics stage or a

graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_CLAMP_ENABLE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetDepthClampEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirectCount-None-07621
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_POLYGON_MODE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetPolygonModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectCount-None-07622
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetRasterizationSamplesEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectCount-None-07623
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_SAMPLE_MASK_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetSampleMaskEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectCount-alphaToCoverageEnable-08919
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_ALPHA_TO_COVERAGE_ENABLE_EXT dynamic state enabled, and alphaToCoverageEnable was VK_TRUE in the last call to vkCmdSetAlphaToCoverageEnableEXT, then the Fragment Output Interface **must** contain a variable for the alpha Component word in Location 0 at Index 0
- VUID-vkCmdDrawIndirectCount-alphaToCoverageEnable-08920
  If a shader object is bound to any graphics stage, and the most recent call to vkCmdSetAlphaToCoverageEnableEXT in the current command buffer set alphaToCoverageEnable to VK_TRUE, then the Fragment Output Interface **must** contain a variable for the alpha Component word in Location 0 at Index 0
- VUID-vkCmdDrawIndirectCount-None-07624
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_ALPHA_TO_COVERAGE_ENABLE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetAlphaToCoverageEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectCount-None-07625
  If the alphaToOne feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_ALPHA_TO_ONE_ENABLE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetAlphaToOneEnableEXT **must** have

been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirectCount-None-07626
  If the logicOp feature is enabled, a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_LOGIC_OP_ENABLE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetLogicOpEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectCount-None-07627
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT dynamic state enabled then vkCmdSetColorBlendEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectCount-None-08657
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and both the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE and there are color attachments bound, then vkCmdSetColorBlendEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectCount-None-07628
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_EQUATION_EXT dynamic state enabled then vkCmdSetColorBlendEquationEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectCount-None-08658
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetColorBlendEnableEXT for any attachment set that attachment's value in pColorBlendEnables to VK_TRUE, then vkCmdSetColorBlendEquationEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectCount-None-07629
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_WRITE_MASK_EXT dynamic state enabled then vkCmdSetColorWriteMaskEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectCount-None-08659
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and both the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE and there are color attachments bound, then vkCmdSetColorWriteMaskEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectCount-None-07630

If the geometryStreams feature is enabled, and a shader object is bound to the VK_SHADER_STAGE_GEOMETRY_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_RASTERIZATION_STREAM_EXT dynamic state enabled, then vkCmdSetRasterizationStreamEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirectCount-None-07631
  If the VK_EXT_conservative_rasterization extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_CONSERVATIVE_RASTERIZATION_MODE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetConservativeRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectCount-None-07632
  If the VK_EXT_conservative_rasterization extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_EXTRA_PRIMITIVE_OVERESTIMATION_SIZE_EXT dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of conservativeRasterizationMode is VK_CONSERVATIVE_RASTERIZATION_MODE_OVERESTIMATE_EXT, then vkCmdSetExtraPrimitiveOverestimationSizeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectCount-None-07633
  If the depthClipEnable feature is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_CLIP_ENABLE_EXT dynamic state, then vkCmdSetDepthClipEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectCount-None-07634
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT dynamic state enabled then vkCmdSetSampleLocationsEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectCount-None-08664
  If the VK_EXT_sample_locations extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetSampleLocationsEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectCount-None-07635
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_ADVANCED_EXT dynamic state enabled then vkCmdSetColorBlendAdvancedEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectCount-rasterizerDiscardEnable-09416
  If the VK_EXT_blend_operation_advanced extension is enabled, and a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to

vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then at least one of vkCmdSetColorBlendEquationEXT and vkCmdSetColorBlendAdvancedEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirectCount-None-07636
  If the VK_EXT_provoking_vertex extension is enabled, a shader object is bound to the VK_SHADER_STAGE_VERTEX_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_PROVOKING_VERTEX_MODE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetProvokingVertexModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectCount-None-07637
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT dynamic state enabled then vkCmdSetLineRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectCount-None-08666
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetPolygonModeEXT in the current command buffer set polygonMode to VK_POLYGON_MODE_LINE, then vkCmdSetLineRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectCount-None-08667
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object is bound to the VK_SHADER_STAGE_VERTEX_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetPrimitiveTopologyEXT in the current command buffer set primitiveTopology to any line topology, then vkCmdSetLineRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectCount-None-08668
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object that outputs line primitives is bound to the VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT or VK_SHADER_STAGE_GEOMETRY_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetLineRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectCount-None-07638
  If the bound graphics pipeline state was created with the

VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT dynamic state enabled then vkCmdSetLineStippleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirectCount-None-08669
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetPolygonModeEXT in the current command buffer set polygonMode to VK_POLYGON_MODE_LINE, then vkCmdSetLineStippleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectCount-None-08670
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object is bound to the VK_SHADER_STAGE_VERTEX_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetPrimitiveTopologyEXT in the current command buffer set primitiveTopology to any line topology, then vkCmdSetLineStippleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectCount-None-08671
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object that outputs line primitives is bound to the VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT or VK_SHADER_STAGE_GEOMETRY_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetLineStippleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectCount-None-07849
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_KHR dynamic state enabled then vkCmdSetLineStippleKHR **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectCount-None-08672
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetLineStippleEnableEXT in the current command buffer set stippledLineEnable to VK_TRUE, then vkCmdSetLineStippleEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectCount-None-07639
  If the depthClipControl feature is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_CLIP_NEGATIVE_ONE_TO_ONE_EXT dynamic state enabled, then

vkCmdSetDepthClipNegativeOneToOneEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirectCount-None-07640
  If the VK_NV_clip_space_w_scaling extension is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_VIEWPORT_W_SCALING_ENABLE_NV dynamic state enabled, then vkCmdSetViewportWScalingEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectCount-None-07641
  If the VK_NV_viewport_swizzle extension is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_VIEWPORT_SWIZZLE_NV dynamic state enabled, then vkCmdSetViewportSwizzleNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectCount-None-07642
  If the VK_NV_fragment_coverage_to_color extension is enabled, a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_TO_COLOR_ENABLE_NV dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetCoverageToColorEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectCount-None-07643
  If the VK_NV_fragment_coverage_to_color extension is enabled, a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_TO_COLOR_LOCATION_NV dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of coverageToColorEnable is VK_TRUE, then vkCmdSetCoverageToColorLocationNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectCount-None-07644
  If the VK_NV_framebuffer_mixed_samples extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_MODULATION_MODE_NV dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetCoverageModulationModeNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectCount-None-07645
  If the VK_NV_framebuffer_mixed_samples extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_MODULATION_TABLE_ENABLE_NV dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of coverageModulationMode is any value other than VK_COVERAGE_MODULATION_MODE_NONE_NV, then vkCmdSetCoverageModulationTableEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectCount-None-07646

If the VK_NV_framebuffer_mixed_samples extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_MODULATION_TABLE_NV dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of coverageModulationTableEnable is VK_TRUE, then vkCmdSetCoverageModulationTableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirectCount-None-07647
  If the shadingRateImage feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_SHADING_RATE_IMAGE_ENABLE_NV dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetShadingRateImageEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectCount-pipelineFragmentShadingRate-09238
  If the pipelineFragmentShadingRate feature is enabled, a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetFragmentShadingRateKHR **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectCount-None-07648
  If the representativeFragmentTest feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_REPRESENTATIVE_FRAGMENT_TEST_ENABLE_NV dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetRepresentativeFragmentTestEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectCount-None-07649
  If the coverageReductionMode feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_REDUCTION_MODE_NV dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetCoverageReductionModeNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectCount-pColorBlendEnables-07470
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT state enabled and the last call to vkCmdSetColorBlendEnableEXT set pColorBlendEnables for any attachment to VK_TRUE, then for those attachments in the subpass the corresponding image view's format features **must** contain VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT
- VUID-vkCmdDrawIndirectCount-rasterizationSamples-07471
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, and the current subpass does not use any color and/or depth/stencil attachments, then the rasterizationSamples in the

last call to vkCmdSetRasterizationSamplesEXT **must** follow the rules for a zero-attachment subpass

- VUID-vkCmdDrawIndirectCount-samples-07472
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_MASK_EXT state enabled and the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state disabled, then the samples parameter in the last call to vkCmdSetSampleMaskEXT **must** be greater or equal to the VkPipelineMultisampleStateCreateInfo::rasterizationSamples parameter used to create the bound graphics pipeline

- VUID-vkCmdDrawIndirectCount-samples-07473
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_MASK_EXT state and VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT states enabled, then the samples parameter in the last call to vkCmdSetSampleMaskEXT **must** be greater or equal to the rasterizationSamples parameter in the last call to vkCmdSetRasterizationSamplesEXT

- VUID-vkCmdDrawIndirectCount-rasterizationSamples-07474
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, and neither the VK_AMD_mixed_attachment_samples nor the VK_NV_framebuffer_mixed_samples extensions are enabled, then the rasterizationSamples in the last call to vkCmdSetRasterizationSamplesEXT **must** be the same as the current subpass color and/or depth/stencil attachments

- VUID-vkCmdDrawIndirectCount-None-09211
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, or a shader object is bound to any graphics stage, and the current render pass instance includes a VkMultisampledRenderToSingleSampledInfoEXT structure with multisampledRenderToSingleSampledEnable equal to VK_TRUE, then the rasterizationSamples in the last call to vkCmdSetRasterizationSamplesEXT **must** be the same as the rasterizationSamples member of that structure

- VUID-vkCmdDrawIndirectCount-firstAttachment-07476
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT dynamic state enabled then vkCmdSetColorBlendEnableEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the firstAttachment and attachmentCount parameters of vkCmdSetColorBlendEnableEXT calls **must** specify an enable for all active color attachments in the current subpass

- VUID-vkCmdDrawIndirectCount-rasterizerDiscardEnable-09417
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetColorBlendEnableEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the firstAttachment and attachmentCount parameters of vkCmdSetColorBlendEnableEXT calls **must** specify an enable for all active color attachments in the current subpass

- VUID-vkCmdDrawIndirectCount-firstAttachment-07477
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_COLOR_BLEND_EQUATION_EXT` dynamic state enabled then vkCmdSetColorBlendEquationEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the `firstAttachment` and `attachmentCount` parameters of `vkCmdSetColorBlendEquationEXT` calls **must** specify the blend equations for all active color attachments in the current subpass where blending is enabled
- VUID-vkCmdDrawIndirectCount-rasterizerDiscardEnable-09418
  If a shader object is bound to the `VK_SHADER_STAGE_FRAGMENT_BIT` stage, and both the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE` and there are color attachments bound, then vkCmdSetColorBlendEquationEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the `firstAttachment` and `attachmentCount` parameters of `vkCmdSetColorBlendEquationEXT` calls **must** specify the blend equations for all active color attachments in the current subpass where blending is enabled
- VUID-vkCmdDrawIndirectCount-firstAttachment-07478
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_COLOR_WRITE_MASK_EXT` dynamic state enabled then vkCmdSetColorWriteMaskEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the `firstAttachment` and `attachmentCount` parameters of `vkCmdSetColorWriteMaskEXT` calls **must** specify the color write mask for all active color attachments in the current subpass
- VUID-vkCmdDrawIndirectCount-rasterizerDiscardEnable-09419
  If a shader object is bound to the `VK_SHADER_STAGE_FRAGMENT_BIT` stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE`, then vkCmdSetColorWriteMaskEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the `firstAttachment` and `attachmentCount` parameters of `vkCmdSetColorWriteMaskEXT` calls **must** specify the color write mask for all active color attachments in the current subpass
- VUID-vkCmdDrawIndirectCount-firstAttachment-07479
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_COLOR_BLEND_ADVANCED_EXT` dynamic state enabled then vkCmdSetColorBlendAdvancedEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the `firstAttachment` and `attachmentCount` parameters of `vkCmdSetColorBlendAdvancedEXT` calls **must** specify the advanced blend equations for all active color attachments in the current subpass where blending is enabled
- VUID-vkCmdDrawIndirectCount-advancedBlendMaxColorAttachments-07480
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_COLOR_BLEND_ADVANCED_EXT` and `VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT` dynamic states enabled and the last calls to vkCmdSetColorBlendEnableEXT and vkCmdSetColorBlendAdvancedEXT have enabled advanced blending, then the number of active color attachments in the current subpass **must** not exceed

advancedBlendMaxColorAttachments

- VUID-vkCmdDrawIndirectCount-primitivesGeneratedQueryWithNonZeroStreams-07481
  If the primitivesGeneratedQueryWithNonZeroStreams feature is not enabled and the VK_QUERY_TYPE_PRIMITIVES_GENERATED_EXT query is active, and the bound graphics pipeline was created with VK_DYNAMIC_STATE_RASTERIZATION_STREAM_EXT state enabled, the last call to vkCmdSetRasterizationStreamEXT **must** have set the rasterizationStream to zero
- VUID-vkCmdDrawIndirectCount-sampleLocationsPerPixel-07482
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state enabled and the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state disabled, then the sampleLocationsPerPixel member of pSampleLocationsInfo in the last call to vkCmdSetSampleLocationsEXT **must** equal the rasterizationSamples member of the VkPipelineMultisampleStateCreateInfo structure the bound graphics pipeline has been created with
- VUID-vkCmdDrawIndirectCount-sampleLocationsPerPixel-07483
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state enabled and the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, then the sampleLocationsPerPixel member of pSampleLocationsInfo in the last call to vkCmdSetSampleLocationsEXT **must** equal the rasterizationSamples parameter of the last call to vkCmdSetRasterizationSamplesEXT
- VUID-vkCmdDrawIndirectCount-sampleLocationsEnable-07484
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, or the bound graphics pipeline was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, and sampleLocationsEnable was VK_TRUE in the last call to vkCmdSetSampleLocationsEnableEXT, and the current subpass has a depth/stencil attachment, then that attachment **must** have been created with the VK_IMAGE_CREATE_SAMPLE_LOCATIONS_COMPATIBLE_DEPTH_BIT_EXT bit set
- VUID-vkCmdDrawIndirectCount-sampleLocationsEnable-07485
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state enabled and the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, and if sampleLocationsEnable was VK_TRUE in the last call to vkCmdSetSampleLocationsEnableEXT, then the sampleLocationsInfo.sampleLocationGridSize.width in the last call to vkCmdSetSampleLocationsEXT **must** evenly divide VkMultisamplePropertiesEXT::sampleLocationGridSize.width as returned by vkGetPhysicalDeviceMultisamplePropertiesEXT with a samples parameter equaling rasterizationSamples
- VUID-vkCmdDrawIndirectCount-sampleLocationsEnable-07486
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state enabled and the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, and if sampleLocationsEnable was VK_TRUE in the last call to vkCmdSetSampleLocationsEnableEXT, then the

sampleLocationsInfo.sampleLocationGridSize.height in the last call to vkCmdSetSampleLocationsEXT **must** evenly divide VkMultisamplePropertiesEXT::sampleLocationGridSize.height as returned by vkGetPhysicalDeviceMultisamplePropertiesEXT with a samples parameter equaling rasterizationSamples

- VUID-vkCmdDrawIndirectCount-sampleLocationsEnable-07487
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, and if sampleLocationsEnable was VK_TRUE in the last call to vkCmdSetSampleLocationsEnableEXT, the fragment shader code **must** not statically use the extended instruction InterpolateAtSample

- VUID-vkCmdDrawIndirectCount-sampleLocationsEnable-07936
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state disabled and the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, the sampleLocationsEnable member of a VkPipelineSampleLocationsStateCreateInfoEXT::sampleLocationsEnable in the bound graphics pipeline is VK_TRUE or VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, then, sampleLocationsInfo.sampleLocationGridSize.width **must** evenly divide VkMultisamplePropertiesEXT::sampleLocationGridSize.width as returned by vkGetPhysicalDeviceMultisamplePropertiesEXT with a samples parameter equaling the value of rasterizationSamples in the last call to vkCmdSetRasterizationSamplesEXT

- VUID-vkCmdDrawIndirectCount-sampleLocationsEnable-07937
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state disabled and the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, the sampleLocationsEnable member of a VkPipelineSampleLocationsStateCreateInfoEXT::sampleLocationsEnable in the bound graphics pipeline is VK_TRUE or VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, then, sampleLocationsInfo.sampleLocationGridSize.height **must** evenly divide VkMultisamplePropertiesEXT::sampleLocationGridSize.height as returned by vkGetPhysicalDeviceMultisamplePropertiesEXT with a samples parameter equaling the value of rasterizationSamples in the last call to vkCmdSetRasterizationSamplesEXT

- VUID-vkCmdDrawIndirectCount-sampleLocationsEnable-07938
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state disabled and the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, the sampleLocationsEnable member of a VkPipelineSampleLocationsStateCreateInfoEXT::sampleLocationsEnable in the bound graphics pipeline is VK_TRUE or VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, then, sampleLocationsInfo.sampleLocationsPerPixel **must** equal rasterizationSamples in the last call to vkCmdSetRasterizationSamplesEXT

- VUID-vkCmdDrawIndirectCount-coverageModulationTableEnable-07488
  If a shader object is bound to any graphics stage or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COVERAGE_MODULATION_TABLE_ENABLE_NV state enabled, and the last call to vkCmdSetCoverageModulationTableEnableNV set coverageModulationTableEnable to VK_TRUE, then the coverageModulationTableCount parameter in the last call to vkCmdSetCoverageModulationTableNV **must** equal the current rasterizationSamples divided by the number of color samples in the current

subpass

- VUID-vkCmdDrawIndirectCount-rasterizationSamples-07489
  If the VK_NV_framebuffer_mixed_samples extension is enabled, and if current subpass has a depth/stencil attachment and depth test, stencil test, or depth bounds test are enabled in the currently bound pipeline state, then the current rasterizationSamples **must** be the same as the sample count of the depth/stencil attachment

- VUID-vkCmdDrawIndirectCount-coverageToColorEnable-07490
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COVERAGE_TO_COLOR_ENABLE_NV state enabled and the last call to vkCmdSetCoverageToColorEnableNV set the coverageToColorEnable to VK_TRUE, then the current subpass **must** have a color attachment at the location selected by the last call to vkCmdSetCoverageToColorLocationNV coverageToColorLocation, with a VkFormat of VK_FORMAT_R8_UINT, VK_FORMAT_R8_SINT, VK_FORMAT_R16_UINT, VK_FORMAT_R16_SINT, VK_FORMAT_R32_UINT, or VK_FORMAT_R32_SINT

- VUID-vkCmdDrawIndirectCount-rasterizerDiscardEnable-09420
  If the VK_NV_fragment_coverage_to_color extension is enabled, and a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the last call to vkCmdSetCoverageToColorEnableNV set the coverageToColorEnable to VK_TRUE, then the current subpass **must** have a color attachment at the location selected by the last call to vkCmdSetCoverageToColorLocationNV coverageToColorLocation, with a VkFormat of VK_FORMAT_R8_UINT, VK_FORMAT_R8_SINT, VK_FORMAT_R16_UINT, VK_FORMAT_R16_SINT, VK_FORMAT_R32_UINT, or VK_FORMAT_R32_SINT

- VUID-vkCmdDrawIndirectCount-coverageReductionMode-07491
  If this VK_NV_coverage_reduction_mode extension is enabled, the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COVERAGE_TO_COLOR_ENABLE_NV and VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT states enabled, the current coverage reduction mode coverageReductionMode, then the current rasterizationSamples, and the sample counts for the color and depth/stencil attachments (if the subpass has them) **must** be a valid combination returned by vkGetPhysicalDeviceSupportedFramebufferMixedSamplesCombinationsNV

- VUID-vkCmdDrawIndirectCount-viewportCount-07492
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, but not the VK_DYNAMIC_STATE_VIEWPORT_SWIZZLE_NV dynamic state enabled, then the bound graphics pipeline **must** have been created with VkPipelineViewportSwizzleStateCreateInfoNV ::viewportCount greater or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT

- VUID-vkCmdDrawIndirectCount-viewportCount-07493
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT and VK_DYNAMIC_STATE_VIEWPORT_SWIZZLE_NV dynamic states enabled then the viewportCount parameter in the last call to vkCmdSetViewportSwizzleNV **must** be greater than or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT

- VUID-vkCmdDrawIndirectCount-viewportCount-09421
  If the VK_NV_viewport_swizzle extension is enabled, and a shader object is bound to any graphics stage, then the viewportCount parameter in the last call to vkCmdSetViewportSwizzleNV **must** be greater than or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawIndirectCount-rasterizationSamples-07494
  If the VK_NV_framebuffer_mixed_samples extension is enabled, and if the current subpass has any color attachments and rasterizationSamples of the last call to vkCmdSetRasterizationSamplesEXT is greater than the number of color samples, then the pipeline sampleShadingEnable **must** be VK_FALSE
- VUID-vkCmdDrawIndirectCount-stippledLineEnable-07495
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT or VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT dynamic states enabled, and if the current stippledLineEnable state is VK_TRUE and the current lineRasterizationMode state is VK_LINE_RASTERIZATION_MODE_RECTANGULAR_KHR, then the stippledRectangularLines feature **must** be enabled
- VUID-vkCmdDrawIndirectCount-stippledLineEnable-07496
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT or VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT dynamic states enabled, and if the current stippledLineEnable state is VK_TRUE and the current lineRasterizationMode state is VK_LINE_RASTERIZATION_MODE_BRESENHAM_KHR, then the stippledBresenhamLines feature **must** be enabled
- VUID-vkCmdDrawIndirectCount-stippledLineEnable-07497
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT or VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT dynamic states enabled, and if the current stippledLineEnable state is VK_TRUE and the current lineRasterizationMode state is VK_LINE_RASTERIZATION_MODE_RECTANGULAR_SMOOTH_KHR, then the stippledSmoothLines feature **must** be enabled
- VUID-vkCmdDrawIndirectCount-stippledLineEnable-07498
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT or VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT dynamic states enabled, and if the current stippledLineEnable state is VK_TRUE and the current lineRasterizationMode state is VK_LINE_RASTERIZATION_MODE_DEFAULT_KHR, then the stippledRectangularLines feature **must** be enabled and VkPhysicalDeviceLimits::strictLines **must** be VK_TRUE
- VUID-vkCmdDrawIndirectCount-conservativePointAndLineRasterization-07499
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_CONSERVATIVE_RASTERIZATION_MODE_EXT dynamic state enabled, conservativePointAndLineRasterization is not supported, and the effective primitive topology output by the last pre-rasterization shader stage is a line or point, then the conservativeRasterizationMode set by the last call to vkCmdSetConservativeRasterizationModeEXT **must** be

VK_CONSERVATIVE_RASTERIZATION_MODE_DISABLED_EXT

- VUID-vkCmdDrawIndirectCount-stage-07073
  If the currently bound pipeline was created with the VkPipelineShaderStageCreateInfo::stage member of an element of VkGraphicsPipelineCreateInfo::pStages set to VK_SHADER_STAGE_VERTEX_BIT, VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT, VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT or VK_SHADER_STAGE_GEOMETRY_BIT, then Mesh Shader Queries **must** not be active
- VUID-vkCmdDrawIndirectCount-None-08877
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_ATTACHMENT_FEEDBACK_LOOP_ENABLE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetAttachmentFeedbackLoopEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectCount-None-07850
  If dynamic state was inherited from VkCommandBufferInheritanceViewportScissorInfoNV, it **must** be set in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectCount-None-08684
  If there is no bound graphics pipeline, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_VERTEX_BIT
- VUID-vkCmdDrawIndirectCount-None-08685
  If there is no bound graphics pipeline, and the tessellationShader feature is enabled, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT
- VUID-vkCmdDrawIndirectCount-None-08686
  If there is no bound graphics pipeline, and the tessellationShader feature is enabled, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT
- VUID-vkCmdDrawIndirectCount-None-08687
  If there is no bound graphics pipeline, and the geometryShader feature is enabled, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_GEOMETRY_BIT
- VUID-vkCmdDrawIndirectCount-None-08688
  If there is no bound graphics pipeline, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_FRAGMENT_BIT
- VUID-vkCmdDrawIndirectCount-None-08689
  If there is no bound graphics pipeline, and the taskShader feature is enabled, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_TASK_BIT_EXT
- VUID-vkCmdDrawIndirectCount-None-08690
  If there is no bound graphics pipeline, and the meshShader feature is enabled, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_MESH_BIT_EXT

- VUID-vkCmdDrawIndirectCount-None-08693
  If there is no bound graphics pipeline, and at least one of the `taskShader` and `meshShader` features is enabled, one of the `VK_SHADER_STAGE_VERTEX_BIT` or `VK_SHADER_STAGE_MESH_BIT_EXT` stages **must** have a valid `VkShaderEXT` bound, and the other **must** have no `VkShaderEXT` bound
- VUID-vkCmdDrawIndirectCount-None-08694
  If there is no bound graphics pipeline, and both the `taskShader` and `meshShader` features are enabled, and a valid `VkShaderEXT` is bound the to the `VK_SHADER_STAGE_MESH_BIT_EXT` stage, and that `VkShaderEXT` was created without the `VK_SHADER_CREATE_NO_TASK_SHADER_BIT_EXT` flag, a valid `VkShaderEXT` **must** be bound to the `VK_SHADER_STAGE_TASK_BIT_EXT` stage
- VUID-vkCmdDrawIndirectCount-None-08695
  If there is no bound graphics pipeline, and both the `taskShader` and `meshShader` features are enabled, and a valid `VkShaderEXT` is bound the to the `VK_SHADER_STAGE_MESH_BIT_EXT` stage, and that `VkShaderEXT` was created with the `VK_SHADER_CREATE_NO_TASK_SHADER_BIT_EXT` flag, there **must** be no `VkShaderEXT` bound to the `VK_SHADER_STAGE_TASK_BIT_EXT` stage
- VUID-vkCmdDrawIndirectCount-None-08696
  If there is no bound graphics pipeline, and a valid `VkShaderEXT` is bound to the `VK_SHADER_STAGE_VERTEX_BIT` stage, there **must** be no `VkShaderEXT` bound to either the `VK_SHADER_STAGE_TASK_BIT_EXT` stage or the `VK_SHADER_STAGE_MESH_BIT_EXT` stage
- VUID-vkCmdDrawIndirectCount-None-08698
  If any graphics shader is bound which was created with the `VK_SHADER_CREATE_LINK_STAGE_BIT_EXT` flag, then all shaders created with the `VK_SHADER_CREATE_LINK_STAGE_BIT_EXT` flag in the same vkCreateShadersEXT call **must** also be bound
- VUID-vkCmdDrawIndirectCount-None-08699
  If any graphics shader is bound which was created with the `VK_SHADER_CREATE_LINK_STAGE_BIT_EXT` flag, any stages in between stages whose shaders which did not create a shader with the `VK_SHADER_CREATE_LINK_STAGE_BIT_EXT` flag as part of the same vkCreateShadersEXT call **must** not have any `VkShaderEXT` bound
- VUID-vkCmdDrawIndirectCount-None-08878
  All bound graphics shader objects **must** have been created with identical or identically defined push constant ranges
- VUID-vkCmdDrawIndirectCount-None-08879
  All bound graphics shader objects **must** have been created with identical or identically defined arrays of descriptor set layouts
- VUID-vkCmdDrawIndirectCount-colorAttachmentCount-09372
  If the current render pass instance was begun with vkCmdBeginRenderingKHR and a VkRenderingInfo::`colorAttachmentCount` equal to `1`, a color attachment with a resolve mode of `VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID`, and a fragment shader is bound, it **must** not declare the `DepthReplacing` or `StencilRefReplacingEXT` execution modes
- VUID-vkCmdDrawIndirectCount-pDynamicStates-08715
  If the bound graphics pipeline state includes a fragment shader stage, was created with `VK_DYNAMIC_STATE_DEPTH_WRITE_ENABLE` set in VkPipelineDynamicStateCreateInfo

::pDynamicStates, and the fragment shader declares the EarlyFragmentTests execution mode and uses OpDepthAttachmentReadEXT, the depthWriteEnable parameter in the last call to vkCmdSetDepthWriteEnableEXT **must** be VK_FALSE

- VUID-vkCmdDrawIndirectCount-pDynamicStates-08716
  If the bound graphics pipeline state includes a fragment shader stage, was created with VK_DYNAMIC_STATE_STENCIL_WRITE_MASK set in VkPipelineDynamicStateCreateInfo ::pDynamicStates, and the fragment shader declares the EarlyFragmentTests execution mode and uses OpStencilAttachmentReadEXT, the writeMask parameter in the last call to vkCmdSetStencilWriteMask **must** be 0

- VUID-vkCmdDrawIndirectCount-None-09116
  If a shader object is bound to any graphics stage or the currently bound graphics pipeline was created with VK_DYNAMIC_STATE_COLOR_WRITE_MASK_EXT, and the format of any color attachment is VK_FORMAT_E5B9G9R9_UFLOAT_PACK32, the corresponding element of the pColorWriteMasks parameter of vkCmdSetColorWriteMaskEXT **must** either include all of VK_COLOR_COMPONENT_R_BIT, VK_COLOR_COMPONENT_G_BIT, and VK_COLOR_COMPONENT_B_BIT, or none of them

- VUID-vkCmdDrawIndirectCount-maxFragmentDualSrcAttachments-09239
  If blending is enabled for any attachment where either the source or destination blend factors for that attachment use the secondary color input, the maximum value of Location for any output attachment statically used in the Fragment Execution Model executed by this command **must** be less than maxFragmentDualSrcAttachments

- VUID-vkCmdDrawIndirectCount-None-09548
  If the current render pass was begun with vkCmdBeginRenderingKHR, and there is no shader object bound to any graphics stage, the value of each element of VkRenderingAttachmentLocationInfoKHR::pColorAttachmentLocations set by vkCmdSetRenderingAttachmentLocationsKHR **must** match the value set for the corresponding element in the currently bound pipeline

- VUID-vkCmdDrawIndirectCount-None-09549
  If the current render pass was begun with vkCmdBeginRenderingKHR, and there is no shader object bound to any graphics stage, input attachment index mappings in the currently bound pipeline **must** match those set for the current render pass instance via VkRenderingInputAttachmentIndexInfoKHR

- VUID-vkCmdDrawIndirectCount-None-09642
  If the current render pass was begun with vkCmdBeginRenderingKHR with the VK_RENDERING_ENABLE_LEGACY_DITHERING_BIT_EXT flag, the bound graphics pipeline **must** have been created with VK_PIPELINE_CREATE_2_ENABLE_LEGACY_DITHERING_BIT_EXT

- VUID-vkCmdDrawIndirectCount-None-09643
  If the bound graphics pipeline was created with VK_PIPELINE_CREATE_2_ENABLE_LEGACY_DITHERING_BIT_EXT, the current render pass **must** have begun with vkCmdBeginRenderingKHR with the VK_RENDERING_ENABLE_LEGACY_DITHERING_BIT_EXT flag

- VUID-vkCmdDrawIndirectCount-None-04007
  All vertex input bindings accessed via vertex input variables declared in the vertex shader entry point's interface **must** have either valid or VK_NULL_HANDLE buffers

bound

- VUID-vkCmdDrawIndirectCount-None-04008
  If the `nullDescriptor` feature is not enabled, all vertex input bindings accessed via vertex input variables declared in the vertex shader entry point's interface **must** not be VK_NULL_HANDLE

- VUID-vkCmdDrawIndirectCount-None-02721
  If `robustBufferAccess` is not enabled, and that pipeline was created without enabling `VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_EXT` for `vertexInputs`, then for a given vertex buffer binding, any attribute data fetched **must** be entirely contained within the corresponding vertex buffer binding, as described in Vertex Input Description

- VUID-vkCmdDrawIndirectCount-None-07842
  If there is a shader object bound to the `VK_SHADER_STAGE_VERTEX_BIT` stage or the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_PRIMITIVE_TOPOLOGY` dynamic state enabled then vkCmdSetPrimitiveTopologyEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirectCount-dynamicPrimitiveTopologyUnrestricted-07500
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_PRIMITIVE_TOPOLOGY` dynamic state enabled and the `dynamicPrimitiveTopologyUnrestricted` is `VK_FALSE`, then the `primitiveTopology` parameter of `vkCmdSetPrimitiveTopologyEXT` **must** be of the same topology class as the pipeline VkPipelineInputAssemblyStateCreateInfo::`topology` state

- VUID-vkCmdDrawIndirectCount-pStrides-04913
  If the bound graphics pipeline was created with the `VK_DYNAMIC_STATE_VERTEX_INPUT_BINDING_STRIDE_EXT` dynamic state enabled, but without the `VK_DYNAMIC_STATE_VERTEX_INPUT_EXT` dynamic state enabled, then vkCmdBindVertexBuffers2EXT **must** have been called and not subsequently invalidated in the current command buffer prior to this draw command, and the `pStrides` parameter of vkCmdBindVertexBuffers2EXT **must** not be `NULL`

- VUID-vkCmdDrawIndirectCount-None-04914
  If there is a shader object bound to the `VK_SHADER_STAGE_VERTEX_BIT` stage or the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VERTEX_INPUT_EXT` dynamic state enabled then vkCmdSetVertexInputEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this draw command

- VUID-vkCmdDrawIndirectCount-Input-07939
  If there is a shader object bound to the `VK_SHADER_STAGE_VERTEX_BIT` stage or the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VERTEX_INPUT_EXT` dynamic state enabled then all variables with the `Input` storage class decorated with `Location` in the `Vertex Execution Model OpEntryPoint` **must** contain a location in VkVertexInputAttributeDescription2EXT::`location`

- VUID-vkCmdDrawIndirectCount-Input-08734
  If there is a shader object bound to the `VK_SHADER_STAGE_VERTEX_BIT` stage or the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VERTEX_INPUT_EXT` dynamic state enabled and either `legacyVertexAttributes` is not enabled or the SPIR-V Type associated with a given `Input` variable of the corresponding `Location` in the `Vertex`

Execution Model OpEntryPoint is 64-bit, then the numeric type associated with all Input variables of the corresponding Location in the Vertex Execution Model OpEntryPoint **must** be the same as VkVertexInputAttributeDescription2EXT::format

- VUID-vkCmdDrawIndirectCount-format-08936
  If there is a shader object bound to the VK_SHADER_STAGE_VERTEX_BIT stage or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VERTEX_INPUT_EXT dynamic state enabled and VkVertexInputAttributeDescription2EXT::format has a 64-bit component, then the scalar width associated with all Input variables of the corresponding Location in the Vertex Execution Model OpEntryPoint **must** be 64-bit

- VUID-vkCmdDrawIndirectCount-format-08937
  If there is a shader object bound to the VK_SHADER_STAGE_VERTEX_BIT stage or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VERTEX_INPUT_EXT dynamic state enabled and the scalar width associated with a Location decorated Input variable in the Vertex Execution Model OpEntryPoint is 64-bit, then the corresponding VkVertexInputAttributeDescription2EXT::format **must** have a 64-bit component

- VUID-vkCmdDrawIndirectCount-None-09203
  If there is a shader object bound to the VK_SHADER_STAGE_VERTEX_BIT stage or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VERTEX_INPUT_EXT dynamic state enabled and VkVertexInputAttributeDescription2EXT::format has a 64-bit component, then all Input variables at the corresponding Location in the Vertex Execution Model OpEntryPoint **must** not use components that are not present in the format

- VUID-vkCmdDrawIndirectCount-None-04875
  If there is a shader object bound to the VK_SHADER_STAGE_VERTEX_BIT stage and the most recent call to vkCmdSetPrimitiveTopologyEXT in the current command buffer set primitiveTopology to VK_PRIMITIVE_TOPOLOGY_PATCH_LIST, or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_PATCH_CONTROL_POINTS_EXT dynamic state enabled then vkCmdSetPatchControlPointsEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirectCount-None-04879
  If there is a shader object bound to the VK_SHADER_STAGE_VERTEX_BIT stage or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_PRIMITIVE_RESTART_ENABLE dynamic state enabled then vkCmdSetPrimitiveRestartEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirectCount-None-09637
  If the primitiveTopologyListRestart feature is not enabled, the topology is VK_PRIMITIVE_TOPOLOGY_POINT_LIST, VK_PRIMITIVE_TOPOLOGY_LINE_LIST, VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST, VK_PRIMITIVE_TOPOLOGY_LINE_LIST_WITH_ADJACENCY, or VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST_WITH_ADJACENCY, there is a shader object bound to the VK_SHADER_STAGE_VERTEX_BIT stage or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_PRIMITIVE_RESTART_ENABLE dynamic state enabled then vkCmdSetPrimitiveRestartEnableEXT **must** be VK_FALSE

- VUID-vkCmdDrawIndirectCount-stage-06481
  The bound graphics pipeline **must** not have been created with the VkPipelineShaderStageCreateInfo::stage member of an element of

VkGraphicsPipelineCreateInfo::pStages set to VK_SHADER_STAGE_TASK_BIT_EXT or VK_SHADER_STAGE_MESH_BIT_EXT

- VUID-vkCmdDrawIndirectCount-None-08885
  There **must** be no shader object bound to either of the VK_SHADER_STAGE_TASK_BIT_EXT or VK_SHADER_STAGE_MESH_BIT_EXT stages
- VUID-vkCmdDrawIndirectCount-buffer-02708
  If buffer is non-sparse then it **must** be bound completely and contiguously to a single VkDeviceMemory object
- VUID-vkCmdDrawIndirectCount-buffer-02709
  buffer **must** have been created with the VK_BUFFER_USAGE_INDIRECT_BUFFER_BIT bit set
- VUID-vkCmdDrawIndirectCount-offset-02710
  offset **must** be a multiple of 4
- VUID-vkCmdDrawIndirectCount-commandBuffer-02711
  commandBuffer **must** not be a protected command buffer
- VUID-vkCmdDrawIndirectCount-countBuffer-02714
  If countBuffer is non-sparse then it **must** be bound completely and contiguously to a single VkDeviceMemory object
- VUID-vkCmdDrawIndirectCount-countBuffer-02715
  countBuffer **must** have been created with the VK_BUFFER_USAGE_INDIRECT_BUFFER_BIT bit set
- VUID-vkCmdDrawIndirectCount-countBufferOffset-02716
  countBufferOffset **must** be a multiple of 4
- VUID-vkCmdDrawIndirectCount-countBuffer-02717
  The count stored in countBuffer **must** be less than or equal to VkPhysicalDeviceLimits::maxDrawIndirectCount
- VUID-vkCmdDrawIndirectCount-countBufferOffset-04129
  (countBufferOffset + sizeof(uint32_t)) **must** be less than or equal to the size of countBuffer
- VUID-vkCmdDrawIndirectCount-stride-03110
  stride **must** be a multiple of 4 and **must** be greater than or equal to sizeof(VkDrawIndirectCommand)
- VUID-vkCmdDrawIndirectCount-maxDrawCount-03111
  If maxDrawCount is greater than or equal to 1, (stride × (maxDrawCount - 1) + offset + sizeof(VkDrawIndirectCommand)) **must** be less than or equal to the size of buffer
- VUID-vkCmdDrawIndirectCount-countBuffer-03121
  If the count stored in countBuffer is equal to 1, (offset + sizeof(VkDrawIndirectCommand)) **must** be less than or equal to the size of buffer
- VUID-vkCmdDrawIndirectCount-countBuffer-03122
  If the count stored in countBuffer is greater than 1, (stride × (drawCount - 1) + offset + sizeof(VkDrawIndirectCommand)) **must** be less than or equal to the size of buffer

## Valid Usage (Implicit)

- VUID-vkCmdDrawIndirectCount-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdDrawIndirectCount-buffer-parameter
  `buffer` **must** be a valid VkBuffer handle
- VUID-vkCmdDrawIndirectCount-countBuffer-parameter
  `countBuffer` **must** be a valid VkBuffer handle
- VUID-vkCmdDrawIndirectCount-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdDrawIndirectCount-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- VUID-vkCmdDrawIndirectCount-renderpass
  This command **must** only be called inside of a render pass instance
- VUID-vkCmdDrawIndirectCount-videocoding
  This command **must** only be called outside of a video coding scope
- VUID-vkCmdDrawIndirectCount-commonparent
  Each of `buffer`, `commandBuffer`, and `countBuffer` **must** have been created, allocated, or retrieved from the same VkDevice

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Inside | Outside | Graphics | Action |

To record an indexed indirect drawing command, call:

```
// Provided by VK_VERSION_1_0
void vkCmdDrawIndexedIndirect(
    VkCommandBuffer                             commandBuffer,
    VkBuffer                                    buffer,
    VkDeviceSize                                offset,
    uint32_t                                    drawCount,
    uint32_t                                    stride);
```

- `commandBuffer` is the command buffer into which the command is recorded.
- `buffer` is the buffer containing draw parameters.
- `offset` is the byte offset into `buffer` where parameters begin.
- `drawCount` is the number of draws to execute, and **can** be zero.
- `stride` is the byte stride between successive sets of draw parameters.

`vkCmdDrawIndexedIndirect` behaves similarly to vkCmdDrawIndexed except that the parameters are read by the device from a buffer during execution. `drawCount` draws are executed by the command, with parameters taken from `buffer` starting at `offset` and increasing by `stride` bytes for each successive draw. The parameters of each draw are encoded in an array of VkDrawIndexedIndirectCommand structures. If `drawCount` is less than or equal to one, `stride` is ignored.

## Valid Usage

- VUID-vkCmdDrawIndexedIndirect-magFilter-04553
  If a VkSampler created with `magFilter` or `minFilter` equal to `VK_FILTER_LINEAR`, `reductionMode` equal to `VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE`, and `compareEnable` equal to `VK_FALSE` is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT`
- VUID-vkCmdDrawIndexedIndirect-magFilter-09598
  If a VkSampler created with `magFilter` or `minFilter` equal to `VK_FILTER_LINEAR` and `reductionMode` equal to either `VK_SAMPLER_REDUCTION_MODE_MIN` or `VK_SAMPLER_REDUCTION_MODE_MAX` is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT`
- VUID-vkCmdDrawIndexedIndirect-mipmapMode-04770
  If a VkSampler created with `mipmapMode` equal to `VK_SAMPLER_MIPMAP_MODE_LINEAR`, `reductionMode` equal to `VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE`, and `compareEnable` equal to `VK_FALSE` is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT`
- VUID-vkCmdDrawIndexedIndirect-mipmapMode-09599
  If a VkSampler created with `mipmapMode` equal to `VK_SAMPLER_MIPMAP_MODE_LINEAR` and `reductionMode` equal to either `VK_SAMPLER_REDUCTION_MODE_MIN` or

VK_SAMPLER_REDUCTION_MODE_MAX is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT

- VUID-vkCmdDrawIndexedIndirect-unnormalizedCoordinates-09635
  If a VkSampler created with unnormalizedCoordinates equal to VK_TRUE is used to sample a VkImageView as a result of this command, then the image view's levelCount and layerCount **must** be 1
- VUID-vkCmdDrawIndexedIndirect-unnormalizedCoordinates-09636
  If a VkSampler created with unnormalizedCoordinates equal to VK_TRUE is used to sample a VkImageView as a result of this command, then the image view's viewType **must** be VK_IMAGE_VIEW_TYPE_1D or VK_IMAGE_VIEW_TYPE_2D
- VUID-vkCmdDrawIndexedIndirect-None-06479
  If a VkImageView is sampled with depth comparison, the image view's format features **must** contain VK_FORMAT_FEATURE_2_SAMPLED_IMAGE_DEPTH_COMPARISON_BIT
- VUID-vkCmdDrawIndexedIndirect-None-02691
  If a VkImageView is accessed using atomic operations as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_STORAGE_IMAGE_ATOMIC_BIT
- VUID-vkCmdDrawIndexedIndirect-None-07888
  If a VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER descriptor is accessed using atomic operations as a result of this command, then the storage texel buffer's format features **must** contain VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_ATOMIC_BIT
- VUID-vkCmdDrawIndexedIndirect-None-02692
  If a VkImageView is sampled with VK_FILTER_CUBIC_EXT as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_CUBIC_BIT_EXT
- VUID-vkCmdDrawIndexedIndirect-None-02693
  If the VK_EXT_filter_cubic extension is not enabled and any VkImageView is sampled with VK_FILTER_CUBIC_EXT as a result of this command, it **must** not have a VkImageViewType of VK_IMAGE_VIEW_TYPE_3D, VK_IMAGE_VIEW_TYPE_CUBE, or VK_IMAGE_VIEW_TYPE_CUBE_ARRAY
- VUID-vkCmdDrawIndexedIndirect-filterCubic-02694
  Any VkImageView being sampled with VK_FILTER_CUBIC_EXT as a result of this command **must** have a VkImageViewType and format that supports cubic filtering, as specified by VkFilterCubicImageViewImageFormatPropertiesEXT::filterCubic returned by vkGetPhysicalDeviceImageFormatProperties2
- VUID-vkCmdDrawIndexedIndirect-filterCubicMinmax-02695
  Any VkImageView being sampled with VK_FILTER_CUBIC_EXT with a reduction mode of either VK_SAMPLER_REDUCTION_MODE_MIN or VK_SAMPLER_REDUCTION_MODE_MAX as a result of this command **must** have a VkImageViewType and format that supports cubic filtering together with minmax filtering, as specified by VkFilterCubicImageViewImageFormatPropertiesEXT::filterCubicMinmax returned by vkGetPhysicalDeviceImageFormatProperties2
- VUID-vkCmdDrawIndexedIndirect-cubicRangeClamp-09212
  If the cubicRangeClamp feature is not enabled, then any VkImageView being sampled with

VK_FILTER_CUBIC_EXT as a result of this command **must** not have a VkSamplerReductionModeCreateInfo::reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE_RANGECLAMP_QCOM

- VUID-vkCmdDrawIndexedIndirect-reductionMode-09213
  Any VkImageView being sampled with a VkSamplerReductionModeCreateInfo::reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE_RANGECLAMP_QCOM as a result of this command **must** sample with VK_FILTER_CUBIC_EXT
- VUID-vkCmdDrawIndexedIndirect-selectableCubicWeights-09214
  If the selectableCubicWeights feature is not enabled, then any VkImageView being sampled with VK_FILTER_CUBIC_EXT as a result of this command **must** have VkSamplerCubicWeightsCreateInfoQCOM::cubicWeights equal to VK_CUBIC_FILTER_WEIGHTS_CATMULL_ROM_QCOM
- VUID-vkCmdDrawIndexedIndirect-flags-02696
  Any VkImage created with a VkImageCreateInfo::flags containing VK_IMAGE_CREATE_CORNER_SAMPLED_BIT_NV sampled as a result of this command **must** only be sampled using a VkSamplerAddressMode of VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_EDGE
- VUID-vkCmdDrawIndexedIndirect-OpTypeImage-07027
  For any VkImageView being written as a storage image where the image format field of the OpTypeImage is Unknown, the view's format features **must** contain VK_FORMAT_FEATURE_2_STORAGE_WRITE_WITHOUT_FORMAT_BIT
- VUID-vkCmdDrawIndexedIndirect-OpTypeImage-07028
  For any VkImageView being read as a storage image where the image format field of the OpTypeImage is Unknown, the view's format features **must** contain VK_FORMAT_FEATURE_2_STORAGE_READ_WITHOUT_FORMAT_BIT
- VUID-vkCmdDrawIndexedIndirect-OpTypeImage-07029
  For any VkBufferView being written as a storage texel buffer where the image format field of the OpTypeImage is Unknown, the view's buffer features **must** contain VK_FORMAT_FEATURE_2_STORAGE_WRITE_WITHOUT_FORMAT_BIT
- VUID-vkCmdDrawIndexedIndirect-OpTypeImage-07030
  Any VkBufferView being read as a storage texel buffer where the image format field of the OpTypeImage is Unknown then the view's buffer features **must** contain VK_FORMAT_FEATURE_2_STORAGE_READ_WITHOUT_FORMAT_BIT
- VUID-vkCmdDrawIndexedIndirect-None-08600
  For each set *n* that is statically used by a bound shader, a descriptor set **must** have been bound to *n* at the same pipeline bind point, with a VkPipelineLayout that is compatible for set *n*, with the VkPipelineLayout used to create the current VkPipeline or the VkDescriptorSetLayout array used to create the current VkShaderEXT , as described in Pipeline Layout Compatibility
- VUID-vkCmdDrawIndexedIndirect-None-08601
  For each push constant that is statically used by a bound shader, a push constant value **must** have been set for the same pipeline bind point, with a VkPipelineLayout that is compatible for push constants, with the VkPipelineLayout used to create the current VkPipeline or the VkDescriptorSetLayout array used to create the current VkShaderEXT , as described in Pipeline Layout Compatibility

- VUID-vkCmdDrawIndexedIndirect-None-10068
  For each array of resources that is used by a bound shader, the indices used to access members of the array **must** be less than the descriptor count for the identified binding in the descriptor sets used by this command
- VUID-vkCmdDrawIndexedIndirect-maintenance4-08602
  If the `maintenance4` feature is not enabled, then for each push constant that is statically used by a bound shader, a push constant value **must** have been set for the same pipeline bind point, with a VkPipelineLayout that is compatible for push constants, with the VkPipelineLayout used to create the current VkPipeline or the VkDescriptorSetLayout and VkPushConstantRange arrays used to create the current VkShaderEXT , as described in Pipeline Layout Compatibility
- VUID-vkCmdDrawIndexedIndirect-None-08114
  Descriptors in each bound descriptor set, specified via vkCmdBindDescriptorSets, **must** be valid as described by descriptor validity if they are statically used by the VkPipeline bound to the pipeline bind point used by this command and the bound VkPipeline was not created with `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`
- VUID-vkCmdDrawIndexedIndirect-None-08115
  If the descriptors used by the VkPipeline bound to the pipeline bind point were specified via vkCmdBindDescriptorSets, the bound VkPipeline **must** have been created without `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`
- VUID-vkCmdDrawIndexedIndirect-None-08116
  Descriptors in bound descriptor buffers, specified via vkCmdSetDescriptorBufferOffsetsEXT, **must** be valid if they are dynamically used by the VkPipeline bound to the pipeline bind point used by this command and the bound VkPipeline was created with `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`
- VUID-vkCmdDrawIndexedIndirect-None-08604
  Descriptors in bound descriptor buffers, specified via vkCmdSetDescriptorBufferOffsetsEXT, **must** be valid if they are dynamically used by any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command
- VUID-vkCmdDrawIndexedIndirect-None-08117
  If the descriptors used by the VkPipeline bound to the pipeline bind point were specified via vkCmdSetDescriptorBufferOffsetsEXT, the bound VkPipeline **must** have been created with `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`
- VUID-vkCmdDrawIndexedIndirect-None-08119
  If a descriptor is dynamically used with a VkPipeline created with `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`, the descriptor memory **must** be resident
- VUID-vkCmdDrawIndexedIndirect-None-08605
  If a descriptor is dynamically used with a VkShaderEXT created with a `VkDescriptorSetLayout` that was created with `VK_DESCRIPTOR_SET_LAYOUT_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`, the descriptor memory **must** be resident
- VUID-vkCmdDrawIndexedIndirect-None-08606
  If the `shaderObject` feature is not enabled, a valid pipeline **must** be bound to the pipeline

bind point used by this command

- VUID-vkCmdDrawIndexedIndirect-None-08608
  If a pipeline is bound to the pipeline bind point used by this command, there **must** not have been any calls to dynamic state setting commands for any state specified statically in the VkPipeline object bound to the pipeline bind point used by this command, since that pipeline was bound
- VUID-vkCmdDrawIndexedIndirect-None-08609
  If the VkPipeline object bound to the pipeline bind point used by this command or any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a VkSampler object that uses unnormalized coordinates, that sampler **must** not be used to sample from any VkImage with a VkImageView of the type `VK_IMAGE_VIEW_TYPE_3D`, `VK_IMAGE_VIEW_TYPE_CUBE`, `VK_IMAGE_VIEW_TYPE_1D_ARRAY`, `VK_IMAGE_VIEW_TYPE_2D_ARRAY` or `VK_IMAGE_VIEW_TYPE_CUBE_ARRAY`, in any shader stage
- VUID-vkCmdDrawIndexedIndirect-None-08610
  If the VkPipeline object bound to the pipeline bind point used by this command or any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a VkSampler object that uses unnormalized coordinates, that sampler **must** not be used with any of the SPIR-V `OpImageSample*` or `OpImageSparseSample*` instructions with `ImplicitLod`, `Dref` or `Proj` in their name, in any shader stage
- VUID-vkCmdDrawIndexedIndirect-None-08611
  If the VkPipeline object bound to the pipeline bind point used by this command or any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a VkSampler object that uses unnormalized coordinates, that sampler **must** not be used with any of the SPIR-V `OpImageSample*` or `OpImageSparseSample*` instructions that includes a LOD bias or any offset values, in any shader stage
- VUID-vkCmdDrawIndexedIndirect-None-08607
  If the `shaderObject` is enabled, either a valid pipeline **must** be bound to the pipeline bind point used by this command, or a valid combination of valid and VK_NULL_HANDLE shader objects **must** be bound to every supported shader stage corresponding to the pipeline bind point used by this command
- VUID-vkCmdDrawIndexedIndirect-uniformBuffers-06935
  If any stage of the VkPipeline object bound to the pipeline bind point used by this command accesses a uniform buffer, and that stage was created without enabling either `VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_EXT` or `VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_2_EXT` for `uniformBuffers`, and the `robustBufferAccess` feature is not enabled, that stage **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point
- VUID-vkCmdDrawIndexedIndirect-None-08612
  If the `robustBufferAccess` feature is not enabled, and any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a uniform buffer, it **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point
- VUID-vkCmdDrawIndexedIndirect-storageBuffers-06936
  If any stage of the VkPipeline object bound to the pipeline bind point used by this

command accesses a storage buffer, and that stage was created without enabling either `VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_EXT` or `VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_2_EXT` for `storageBuffers`, and the `robustBufferAccess` feature is not enabled, that stage **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point

- VUID-vkCmdDrawIndexedIndirect-None-08613
  If the `robustBufferAccess` feature is not enabled, and any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a storage buffer, it **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point
- VUID-vkCmdDrawIndexedIndirect-commandBuffer-02707
  If `commandBuffer` is an unprotected command buffer and `protectedNoFault` is not supported, any resource accessed by bound shaders **must** not be a protected resource
- VUID-vkCmdDrawIndexedIndirect-None-06550
  If a bound shader accesses a VkSampler or VkImageView object that enables sampler $Y'C_BC_R$ conversion, that object **must** only be used with `OpImageSample*` or `OpImageSparseSample*` instructions
- VUID-vkCmdDrawIndexedIndirect-ConstOffset-06551
  If a bound shader accesses a VkSampler or VkImageView object that enables sampler $Y'C_BC_R$ conversion, that object **must** not use the `ConstOffset` and `Offset` operands
- VUID-vkCmdDrawIndexedIndirect-viewType-07752
  If a VkImageView is accessed as a result of this command, then the image view's `viewType` **must** match the `Dim` operand of the `OpTypeImage` as described in Compatibility Between SPIR-V Image Dimensions and Vulkan ImageView Types
- VUID-vkCmdDrawIndexedIndirect-format-07753
  If a VkImageView is accessed as a result of this command, then the numeric type of the image view's `format` and the `Sampled Type` operand of the `OpTypeImage` **must** match
- VUID-vkCmdDrawIndexedIndirect-OpImageWrite-08795
  If a VkImageView created with a format other than `VK_FORMAT_A8_UNORM_KHR` is accessed using `OpImageWrite` as a result of this command, then the `Type` of the `Texel` operand of that instruction **must** have at least as many components as the image view's format
- VUID-vkCmdDrawIndexedIndirect-OpImageWrite-08796
  If a VkImageView created with the format `VK_FORMAT_A8_UNORM_KHR` is accessed using `OpImageWrite` as a result of this command, then the `Type` of the `Texel` operand of that instruction **must** have four components
- VUID-vkCmdDrawIndexedIndirect-OpImageWrite-04469
  If a VkBufferView is accessed using `OpImageWrite` as a result of this command, then the `Type` of the `Texel` operand of that instruction **must** have at least as many components as the buffer view's format
- VUID-vkCmdDrawIndexedIndirect-SampledType-04470
  If a VkImageView with a VkFormat that has a 64-bit component width is accessed as a result of this command, the `SampledType` of the `OpTypeImage` operand of that instruction **must** have a `Width` of 64

- VUID-vkCmdDrawIndexedIndirect-SampledType-04471
  If a VkImageView with a VkFormat that has a component width less than 64-bit is accessed as a result of this command, the `SampledType` of the `OpTypeImage` operand of that instruction **must** have a `Width` of 32
- VUID-vkCmdDrawIndexedIndirect-SampledType-04472
  If a VkBufferView with a VkFormat that has a 64-bit component width is accessed as a result of this command, the `SampledType` of the `OpTypeImage` operand of that instruction **must** have a `Width` of 64
- VUID-vkCmdDrawIndexedIndirect-SampledType-04473
  If a VkBufferView with a VkFormat that has a component width less than 64-bit is accessed as a result of this command, the `SampledType` of the `OpTypeImage` operand of that instruction **must** have a `Width` of 32
- VUID-vkCmdDrawIndexedIndirect-sparseImageInt64Atomics-04474
  If the `sparseImageInt64Atomics` feature is not enabled, VkImage objects created with the `VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT` flag **must** not be accessed by atomic instructions through an `OpTypeImage` with a `SampledType` with a `Width` of 64 by this command
- VUID-vkCmdDrawIndexedIndirect-sparseImageInt64Atomics-04475
  If the `sparseImageInt64Atomics` feature is not enabled, VkBuffer objects created with the `VK_BUFFER_CREATE_SPARSE_RESIDENCY_BIT` flag **must** not be accessed by atomic instructions through an `OpTypeImage` with a `SampledType` with a `Width` of 64 by this command
- VUID-vkCmdDrawIndexedIndirect-OpImageWeightedSampleQCOM-06971
  If `OpImageWeightedSampleQCOM` is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_2_WEIGHT_SAMPLED_IMAGE_BIT_QCOM`
- VUID-vkCmdDrawIndexedIndirect-OpImageWeightedSampleQCOM-06972
  If `OpImageWeightedSampleQCOM` uses a VkImageView as a sample weight image as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_2_WEIGHT_IMAGE_BIT_QCOM`
- VUID-vkCmdDrawIndexedIndirect-OpImageBoxFilterQCOM-06973
  If `OpImageBoxFilterQCOM` is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_2_BOX_FILTER_SAMPLED_BIT_QCOM`
- VUID-vkCmdDrawIndexedIndirect-OpImageBlockMatchSSDQCOM-06974
  If `OpImageBlockMatchSSDQCOM` is used to read from an VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM`
- VUID-vkCmdDrawIndexedIndirect-OpImageBlockMatchSADQCOM-06975
  If `OpImageBlockMatchSADQCOM` is used to read from an VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM`
- VUID-vkCmdDrawIndexedIndirect-OpImageBlockMatchSADQCOM-06976
  If `OpImageBlockMatchSADQCOM` or OpImageBlockMatchSSDQCOM is used to read from a reference image as result of this command, then the specified reference coordinates **must** not fail integer texel coordinate validation

- VUID-vkCmdDrawIndexedIndirect-OpImageWeightedSampleQCOM-06977
  If `OpImageWeightedSampleQCOM`, `OpImageBoxFilterQCOM`, `OpImageBlockMatchWindowSSDQCOM`, `OpImageBlockMatchWindowSADQCOM`, `OpImageBlockMatchGatherSSDQCOM`, `OpImageBlockMatchGatherSADQCOM`, `OpImageBlockMatchSSDQCOM`, or `OpImageBlockMatchSADQCOM` uses a VkSampler as a result of this command, then the sampler **must** have been created with `VK_SAMPLER_CREATE_IMAGE_PROCESSING_BIT_QCOM`
- VUID-vkCmdDrawIndexedIndirect-OpImageWeightedSampleQCOM-06978
  If any command other than `OpImageWeightedSampleQCOM`, `OpImageBoxFilterQCOM`, `OpImageBlockMatchWindowSSDQCOM`, `OpImageBlockMatchWindowSADQCOM`, `OpImageBlockMatchGatherSSDQCOM`, `OpImageBlockMatchGatherSADQCOM`, `OpImageBlockMatchSSDQCOM`, or `OpImageBlockMatchSADQCOM` uses a VkSampler as a result of this command, then the sampler **must** not have been created with `VK_SAMPLER_CREATE_IMAGE_PROCESSING_BIT_QCOM`
- VUID-vkCmdDrawIndexedIndirect-OpImageBlockMatchWindow-09215
  If a `OpImageBlockMatchWindow*QCOM` or `OpImageBlockMatchGather*QCOM` instruction is used to read from an VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM`
- VUID-vkCmdDrawIndexedIndirect-OpImageBlockMatchWindow-09216
  If a `OpImageBlockMatchWindow*QCOM` or `OpImageBlockMatchGather*QCOM` instruction is used to read from an VkImageView as a result of this command, then the image view's format **must** be a single-component format
- VUID-vkCmdDrawIndexedIndirect-OpImageBlockMatchWindow-09217
  If a `OpImageBlockMatchWindow*QCOM` or `OpImageBlockMatchGather*QCOM` read from a reference image as result of this command, then the specified reference coordinates **must** not fail integer texel coordinate validation
- VUID-vkCmdDrawIndexedIndirect-None-07288
  Any shader invocation executed by this command **must** terminate
- VUID-vkCmdDrawIndexedIndirect-None-09600
  If a descriptor with type equal to any of `VK_DESCRIPTOR_TYPE_SAMPLE_WEIGHT_IMAGE_QCOM`, `VK_DESCRIPTOR_TYPE_BLOCK_MATCH_IMAGE_QCOM`, `VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE`, `VK_DESCRIPTOR_TYPE_STORAGE_IMAGE`, or `VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT` is accessed as a result of this command, the image subresource identified by that descriptor **must** be in the image layout identified when the descriptor was written
- VUID-vkCmdDrawIndexedIndirect-renderPass-02684
  The current render pass **must** be compatible with the `renderPass` member of the `VkGraphicsPipelineCreateInfo` structure specified when creating the `VkPipeline` bound to `VK_PIPELINE_BIND_POINT_GRAPHICS`
- VUID-vkCmdDrawIndexedIndirect-subpass-02685
  The subpass index of the current render pass **must** be equal to the `subpass` member of the `VkGraphicsPipelineCreateInfo` structure specified when creating the `VkPipeline` bound to `VK_PIPELINE_BIND_POINT_GRAPHICS`
- VUID-vkCmdDrawIndexedIndirect-None-07748
  If any shader statically accesses an input attachment, a valid descriptor **must** be bound to the pipeline via a descriptor set

- VUID-vkCmdDrawIndexedIndirect-OpTypeImage-07468
  If any shader executed by this pipeline accesses an `OpTypeImage` variable with a `Dim` operand of `SubpassData`, it **must** be decorated with an `InputAttachmentIndex` that corresponds to a valid input attachment in the current subpass
- VUID-vkCmdDrawIndexedIndirect-None-07469
  Input attachment views accessed in a subpass **must** be created with the same VkFormat as the corresponding subpass definition, and be created with a VkImageView that is compatible with the attachment referenced by the subpass' `pInputAttachments` [`InputAttachmentIndex`] in the currently bound VkFramebuffer as specified by Fragment Input Attachment Compatibility
- VUID-vkCmdDrawIndexedIndirect-pDepthInputAttachmentIndex-09595
  Input attachment views accessed in a dynamic render pass with a `InputAttachmentIndex` referenced by VkRenderingInputAttachmentIndexInfoKHR, or no `InputAttachmentIndex` if VkRenderingInputAttachmentIndexInfoKHR:`pDepthInputAttachmentIndex` or VkRenderingInputAttachmentIndexInfoKHR:`pStencilInputAttachmentIndex` are `NULL`, **must** be created with a VkImageView that is compatible with the corresponding color, depth, or stencil attachment in VkRenderingInfo
- VUID-vkCmdDrawIndexedIndirect-pDepthInputAttachmentIndex-09596
  Input attachment views accessed in a dynamic render pass via a shader object **must** have an `InputAttachmentIndex` if both VkRenderingInputAttachmentIndexInfoKHR:`pDepthInputAttachmentIndex` and VkRenderingInputAttachmentIndexInfoKHR:`pStencilInputAttachmentIndex` are non-`NULL`
- VUID-vkCmdDrawIndexedIndirect-InputAttachmentIndex-09597
  If an input attachment view accessed in a dynamic render pass via a shader object has an `InputAttachmentIndex`, the `InputAttachmentIndex` **must** match an index in VkRenderingInputAttachmentIndexInfoKHR
- VUID-vkCmdDrawIndexedIndirect-None-06537
  Memory backing image subresources used as attachments in the current render pass **must** not be written in any way other than as an attachment by this command
- VUID-vkCmdDrawIndexedIndirect-None-09000
  If a color attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass, it is not in the `VK_IMAGE_LAYOUT_ATTACHMENT_FEEDBACK_LOOP_OPTIMAL_EXT` image layout, and either:
  - the `VK_PIPELINE_CREATE_COLOR_ATTACHMENT_FEEDBACK_LOOP_BIT_EXT` is set on the currently bound pipeline or
  - the last call to vkCmdSetAttachmentFeedbackLoopEnableEXT included `VK_IMAGE_ASPECT_COLOR_BIT` and
    - there is no currently bound graphics pipeline or
    - the currently bound graphics pipeline was created with `VK_DYNAMIC_STATE_ATTACHMENT_FEEDBACK_LOOP_ENABLE_EXT`

  it **must** not be accessed in any way other than as an attachment by this command
- VUID-vkCmdDrawIndexedIndirect-None-09001

If a depth attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass, it is not in the VK_IMAGE_LAYOUT_ATTACHMENT_FEEDBACK_LOOP_OPTIMAL_EXT image layout, and either:

- the VK_PIPELINE_CREATE_DEPTH_STENCIL_ATTACHMENT_FEEDBACK_LOOP_BIT_EXT is set on the currently bound pipeline or
- the last call to vkCmdSetAttachmentFeedbackLoopEnableEXT included VK_IMAGE_ASPECT_DEPTH_BIT and
  - there is no currently bound graphics pipeline or
  - the currently bound graphics pipeline was created with VK_DYNAMIC_STATE_ATTACHMENT_FEEDBACK_LOOP_ENABLE_EXT

it **must** not be accessed in any way other than as an attachment by this command

- VUID-vkCmdDrawIndexedIndirect-None-09002
  If a stencil attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass, it is not in the VK_IMAGE_LAYOUT_ATTACHMENT_FEEDBACK_LOOP_OPTIMAL_EXT image layout, and either:
  - the VK_PIPELINE_CREATE_DEPTH_STENCIL_ATTACHMENT_FEEDBACK_LOOP_BIT_EXT is set on the currently bound pipeline or
  - the last call to vkCmdSetAttachmentFeedbackLoopEnableEXT included VK_IMAGE_ASPECT_STENCIL_BIT and
    - there is no currently bound graphics pipeline or
    - the currently bound graphics pipeline was created with VK_DYNAMIC_STATE_ATTACHMENT_FEEDBACK_LOOP_ENABLE_EXT

  it **must** not be accessed in any way other than as an attachment by this command
- VUID-vkCmdDrawIndexedIndirect-None-09003
  If an attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass, it **must** not be accessed in any way other than as an attachment, storage image, or sampled image by this command
- VUID-vkCmdDrawIndexedIndirect-None-06539
  If any previously recorded command in the current subpass accessed an image subresource used as an attachment in this subpass in any way other than as an attachment, this command **must** not write to that image subresource as an attachment
- VUID-vkCmdDrawIndexedIndirect-None-06886
  If the current render pass instance uses a depth/stencil attachment with a read-only layout for the depth aspect, depth writes **must** be disabled
- VUID-vkCmdDrawIndexedIndirect-None-06887
  If the current render pass instance uses a depth/stencil attachment with a read-only layout for the stencil aspect, both front and back writeMask are not zero, and stencil test is enabled, all stencil ops **must** be VK_STENCIL_OP_KEEP
- VUID-vkCmdDrawIndexedIndirect-None-07831
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT

dynamic state enabled then vkCmdSetViewport **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirect-None-07832
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SCISSOR dynamic state enabled then vkCmdSetScissor **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirect-None-07833
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_WIDTH dynamic state enabled then vkCmdSetLineWidth **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirect-None-08617
  If a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetPolygonModeEXT in the current command buffer set polygonMode to VK_POLYGON_MODE_LINE, vkCmdSetLineWidth **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirect-None-08618
  If a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetPrimitiveTopologyEXT in the current command buffer set primitiveTopology to any line topology, vkCmdSetLineWidth **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirect-None-08619
  If a shader object that outputs line primitives is bound to the VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT or VK_SHADER_STAGE_GEOMETRY_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, vkCmdSetLineWidth **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirect-None-07834
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_BIAS dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of depthBiasEnable is VK_TRUE, then vkCmdSetDepthBounds or vkCmdSetDepthBias2EXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirect-None-07835
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_BLEND_CONSTANTS dynamic state enabled then vkCmdSetBlendConstants **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirect-None-08621
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent

call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetColorBlendEnableEXT in the current command buffer set any element of pColorBlendEnables to VK_TRUE, and the most recent call to vkCmdSetColorBlendEquationEXT in the current command buffer set the same element of pColorBlendEquations to a VkColorBlendEquationEXT structure with any VkBlendFactor member with a value of VK_BLEND_FACTOR_CONSTANT_COLOR, VK_BLEND_FACTOR_ONE_MINUS_CONSTANT_COLOR, VK_BLEND_FACTOR_CONSTANT_ALPHA, or VK_BLEND_FACTOR_ONE_MINUS_CONSTANT_ALPHA, vkCmdSetBlendConstants **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirect-None-07836
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_BOUNDS dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of depthBoundsTestEnable is VK_TRUE, then vkCmdSetDepthBounds **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirect-None-07837
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_STENCIL_COMPARE_MASK dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of stencilTestEnable is VK_TRUE, then vkCmdSetStencilCompareMask **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirect-None-07838
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_STENCIL_WRITE_MASK dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of stencilTestEnable is VK_TRUE, then vkCmdSetStencilWriteMask **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirect-None-07839
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_STENCIL_REFERENCE dynamic state enabled, the current value of and rasterizerDiscardEnable is VK_FALSE, the current value of stencilTestEnable is VK_TRUE, then vkCmdSetStencilReference **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirect-maxMultiviewInstanceIndex-02688
  If the draw is recorded in a render pass instance with multiview enabled, the maximum instance index **must** be less than or equal to VkPhysicalDeviceMultiviewProperties::maxMultiviewInstanceIndex
- VUID-vkCmdDrawIndexedIndirect-sampleLocationsEnable-02689
  If the bound graphics pipeline was created with

VkPipelineSampleLocationsStateCreateInfoEXT::sampleLocationsEnable set to VK_TRUE and the current subpass has a depth/stencil attachment, then that attachment **must** have been created with the VK_IMAGE_CREATE_SAMPLE_LOCATIONS_COMPATIBLE_DEPTH_BIT_EXT bit set

- VUID-vkCmdDrawIndexedIndirect-None-06666
  If the VK_EXT_sample_locations extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of sampleLocationsEnable is VK_TRUE, then vkCmdSetSampleLocationsEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirect-None-07840
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_CULL_MODE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetCullModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirect-None-07841
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_FRONT_FACE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetFrontFaceEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirect-None-07843
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_TEST_ENABLE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, vkCmdSetDepthTestEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirect-None-07844
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_WRITE_ENABLE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetDepthWriteEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirect-None-07845
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_COMPARE_OP dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of depthTestEnable is VK_TRUE, then vkCmdSetDepthCompareOpEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirect-None-07846
  If the depthBounds feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_BOUNDS_TEST_ENABLE dynamic state enabled, and the current value

of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetDepthBoundsTestEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirect-None-07847
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_STENCIL_TEST_ENABLE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetStencilTestEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirect-None-07848
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_STENCIL_OP dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, the current value of stencilTestEnable is VK_TRUE, then vkCmdSetStencilOpEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirect-viewportCount-03417
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, but not the VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT dynamic state enabled, then vkCmdSetViewportWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the viewportCount parameter of vkCmdSetViewportWithCountEXT **must** match the VkPipelineViewportStateCreateInfo::scissorCount of the pipeline

- VUID-vkCmdDrawIndexedIndirect-scissorCount-03418
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT dynamic state enabled, but not the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, then vkCmdSetScissorWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the scissorCount parameter of vkCmdSetScissorWithCountEXT **must** match the VkPipelineViewportStateCreateInfo::viewportCount of the pipeline

- VUID-vkCmdDrawIndexedIndirect-viewportCount-03419
  If the bound graphics pipeline state was created with both the VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT and VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic states enabled then both vkCmdSetViewportWithCountEXT and vkCmdSetScissorWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the viewportCount parameter of vkCmdSetViewportWithCountEXT **must** match the scissorCount parameter of vkCmdSetScissorWithCountEXT

- VUID-vkCmdDrawIndexedIndirect-None-08635
  If a shader object is bound to any graphics stage, then both vkCmdSetViewportWithCountEXT and vkCmdSetScissorWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the viewportCount parameter of vkCmdSetViewportWithCountEXT **must** match the scissorCount parameter of vkCmdSetScissorWithCountEXT

- VUID-vkCmdDrawIndexedIndirect-viewportCount-04137
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT` dynamic state enabled, but not the `VK_DYNAMIC_STATE_VIEWPORT_W_SCALING_NV` dynamic state enabled, then the bound graphics pipeline **must** have been created with VkPipelineViewportWScalingStateCreateInfoNV::`viewportCount` greater or equal to the `viewportCount` parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawIndexedIndirect-viewportCount-04138
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT` and `VK_DYNAMIC_STATE_VIEWPORT_W_SCALING_NV` dynamic states enabled then the `viewportCount` parameter in the last call to vkCmdSetViewportWScalingNV **must** be greater than or equal to the `viewportCount` parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawIndexedIndirect-None-09232
  If the `VK_NV_clip_space_w_scaling` extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetViewportWScalingEnableNV in the current command buffer set `viewportWScalingEnable` to `VK_TRUE`, then vkCmdSetViewportWScalingNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirect-None-08636
  If the `VK_NV_clip_space_w_scaling` extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetViewportWScalingEnableNV in the current command buffer set `viewportWScalingEnable` to `VK_TRUE`, then the `viewportCount` parameter in the last call to vkCmdSetViewportWScalingNV **must** be greater than or equal to the `viewportCount` parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawIndexedIndirect-viewportCount-04139
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT` dynamic state enabled, but not the `VK_DYNAMIC_STATE_VIEWPORT_SHADING_RATE_PALETTE_NV` dynamic state enabled, then the bound graphics pipeline **must** have been created with VkPipelineViewportShadingRateImageStateCreateInfoNV::`viewportCount` greater or equal to the `viewportCount` parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawIndexedIndirect-viewportCount-04140
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT` and `VK_DYNAMIC_STATE_VIEWPORT_SHADING_RATE_PALETTE_NV` dynamic states enabled then the `viewportCount` parameter in the last call to vkCmdSetViewportShadingRatePaletteNV **must** be greater than or equal to the `viewportCount` parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawIndexedIndirect-shadingRateImage-09233
  If the `shadingRateImage` feature is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE`, then vkCmdSetCoarseSampleOrderNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirect-shadingRateImage-09234
  If the `shadingRateImage` feature is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE`, and the most recent call to vkCmdSetShadingRateImageEnableNV in the current command buffer set `shadingRateImageEnable` to `VK_TRUE`, then vkCmdSetViewportShadingRatePaletteNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirect-None-08637
  If the `shadingRateImage` feature is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE`, and the most recent call to vkCmdSetShadingRateImageEnableNV in the current command buffer set `shadingRateImageEnable` to `VK_TRUE`, then the `viewportCount` parameter in the last call to vkCmdSetViewportShadingRatePaletteNV **must** be greater than or equal to the `viewportCount` parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawIndexedIndirect-VkPipelineVieportCreateInfo-04141
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT` dynamic state enabled and a VkPipelineViewportSwizzleStateCreateInfoNV structure chained from VkPipelineViewportStateCreateInfo, then the bound graphics pipeline **must** have been created with VkPipelineViewportSwizzleStateCreateInfoNV::`viewportCount` greater or equal to the `viewportCount` parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawIndexedIndirect-VkPipelineVieportCreateInfo-04142
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT` dynamic state enabled and a VkPipelineViewportExclusiveScissorStateCreateInfoNV structure chained from VkPipelineViewportStateCreateInfo, then the bound graphics pipeline **must** have been created with VkPipelineViewportExclusiveScissorStateCreateInfoNV::`exclusiveScissorCount` greater or equal to the `viewportCount` parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawIndexedIndirect-None-07878
  If the `exclusiveScissor` feature is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_EXCLUSIVE_SCISSOR_ENABLE_NV` dynamic state enabled, then vkCmdSetExclusiveScissorEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirect-None-07879
  If the `exclusiveScissor` feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_EXCLUSIVE_SCISSOR_NV` dynamic state enabled, and the most recent call to vkCmdSetExclusiveScissorEnableNV in the current command buffer set any element of `pExclusiveScissorEnables` to `VK_TRUE`, then vkCmdSetExclusiveScissorNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirect-None-04876
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_RASTERIZER_DISCARD_ENABLE` dynamic state enabled, then vkCmdSetRasterizerDiscardEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirect-None-04877
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_DEPTH_BIAS_ENABLE` dynamic state enabled, and the current value of `rasterizerDiscardEnable` is `VK_FALSE`, then vkCmdSetDepthBiasEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirect-logicOp-04878
  If a shader object is bound to the `VK_SHADER_STAGE_FRAGMENT_BIT` or a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_LOGIC_OP_EXT` dynamic state enabled, the current value of `rasterizerDiscardEnable` is `VK_FALSE`, and the current value of `logicOpEnable` is `VK_TRUE`, then vkCmdSetLogicOpEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirect-primitiveFragmentShadingRateWithMultipleViewports-04552
  If the `primitiveFragmentShadingRateWithMultipleViewports` limit is not supported, the bound graphics pipeline was created with the `VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT` dynamic state enabled, and any of the shader stages of the bound graphics pipeline write to the `PrimitiveShadingRateKHR` built-in, then vkCmdSetViewportWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the `viewportCount` parameter of `vkCmdSetViewportWithCountEXT` **must** be `1`
- VUID-vkCmdDrawIndexedIndirect-primitiveFragmentShadingRateWithMultipleViewports-08642
  If the `primitiveFragmentShadingRateWithMultipleViewports` limit is not supported, and any shader object bound to a graphics stage writes to the `PrimitiveShadingRateKHR` built-in, then vkCmdSetViewportWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the `viewportCount` parameter of `vkCmdSetViewportWithCountEXT` **must** be `1`
- VUID-vkCmdDrawIndexedIndirect-blendEnable-04727
  If rasterization is not disabled in the bound graphics pipeline, then for each color attachment in the subpass, if the corresponding image view's format features do not contain `VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT`, then the `blendEnable` member of the corresponding element of the `pAttachments` member of `pColorBlendState` **must** be `VK_FALSE`
- VUID-vkCmdDrawIndexedIndirect-None-08643
  If a shader object is bound to the `VK_SHADER_STAGE_FRAGMENT_BIT` stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE`, then for each color attachment in the render pass, if the corresponding image view's format features do not contain `VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT`, then the corresponding member of `pColorBlendEnables` in the most recent call to `vkCmdSetColorBlendEnableEXT` in the current

command buffer that affected that attachment index **must** have been VK_FALSE

- VUID-vkCmdDrawIndexedIndirect-multisampledRenderToSingleSampled-07284
  If rasterization is not disabled in the bound graphics pipeline, and none of the following is enabled:

  - the VK_AMD_mixed_attachment_samples extension
  - the VK_NV_framebuffer_mixed_samples extension
  - the multisampledRenderToSingleSampled feature

  then rasterizationSamples for the currently bound graphics pipeline **must** be the same as the current subpass color and/or depth/stencil attachments

- VUID-vkCmdDrawIndexedIndirect-None-08644
  If a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and none of the following is enabled:

  - the VK_AMD_mixed_attachment_samples extension
  - the VK_NV_framebuffer_mixed_samples extension
  - the multisampledRenderToSingleSampled feature

  then the most recent call to vkCmdSetRasterizationSamplesEXT in the current command buffer **must** have set rasterizationSamples to be the same as the number of samples for the current render pass color and/or depth/stencil attachments

- VUID-vkCmdDrawIndexedIndirect-None-08876
  If a shader object is bound to any graphics stage, the current render pass instance **must** have been begun with vkCmdBeginRenderingKHR

- VUID-vkCmdDrawIndexedIndirect-imageView-06172
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the imageView member of pDepthAttachment is not VK_NULL_HANDLE, and the layout member of pDepthAttachment is VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL, this command **must** not write any values to the depth attachment

- VUID-vkCmdDrawIndexedIndirect-imageView-06173
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the imageView member of pStencilAttachment is not VK_NULL_HANDLE, and the layout member of pStencilAttachment is VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL, this command **must** not write any values to the stencil attachment

- VUID-vkCmdDrawIndexedIndirect-imageView-06174
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the imageView member of pDepthAttachment is not VK_NULL_HANDLE, and the layout member of pDepthAttachment is VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_STENCIL_ATTACHMENT_OPTIMAL, this command **must** not write any values to the depth attachment

- VUID-vkCmdDrawIndexedIndirect-imageView-06175
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the imageView member of pStencilAttachment is not VK_NULL_HANDLE, and the layout

member of pStencilAttachment is VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_STENCIL_READ_ONLY_OPTIMAL, this command **must** not write any values to the stencil attachment

- VUID-vkCmdDrawIndexedIndirect-imageView-06176
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the imageView member of pDepthAttachment is not VK_NULL_HANDLE, and the layout member of pDepthAttachment is VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL, this command **must** not write any values to the depth attachment
- VUID-vkCmdDrawIndexedIndirect-imageView-06177
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the imageView member of pStencilAttachment is not VK_NULL_HANDLE, and the layout member of pStencilAttachment is VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL, this command **must** not write any values to the stencil attachment
- VUID-vkCmdDrawIndexedIndirect-viewMask-06178
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the currently bound graphics pipeline **must** have been created with a VkPipelineRenderingCreateInfo::viewMask equal to VkRenderingInfo::viewMask
- VUID-vkCmdDrawIndexedIndirect-colorAttachmentCount-06179
  If the dynamicRenderingUnusedAttachments feature is not enabled and the current render pass instance was begun with vkCmdBeginRenderingKHR, the currently bound graphics pipeline **must** have been created with a VkPipelineRenderingCreateInfo::colorAttachmentCount equal to VkRenderingInfo::colorAttachmentCount
- VUID-vkCmdDrawIndexedIndirect-dynamicRenderingUnusedAttachments-08910
  If the dynamicRenderingUnusedAttachments feature is not enabled, and the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingInfo::colorAttachmentCount greater than 0, then each element of the VkRenderingInfo::pColorAttachments array with an imageView not equal to VK_NULL_HANDLE **must** have been created with a VkFormat equal to the corresponding element of VkPipelineRenderingCreateInfo::pColorAttachmentFormats used to create the currently bound graphics pipeline
- VUID-vkCmdDrawIndexedIndirect-dynamicRenderingUnusedAttachments-08912
  If the dynamicRenderingUnusedAttachments feature is not enabled, and the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingInfo::colorAttachmentCount greater than 0, then each element of the VkRenderingInfo::pColorAttachments array with an imageView equal to VK_NULL_HANDLE **must** have the corresponding element of VkPipelineRenderingCreateInfo::pColorAttachmentFormats used to create the currently bound pipeline equal to VK_FORMAT_UNDEFINED
- VUID-vkCmdDrawIndexedIndirect-dynamicRenderingUnusedAttachments-08911
  If the dynamicRenderingUnusedAttachments feature is enabled, and the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingInfo::colorAttachmentCount greater than 0, then each element of the VkRenderingInfo::pColorAttachments array with an imageView not equal to VK_NULL_HANDLE **must** have been created with a VkFormat equal to the corresponding element of VkPipelineRenderingCreateInfo::pColorAttachmentFormats used to create the currently bound graphics pipeline, or the corresponding element of

VkPipelineRenderingCreateInfo::pColorAttachmentFormats, if it exists, **must** be VK_FORMAT_UNDEFINED

- VUID-vkCmdDrawIndexedIndirect-colorAttachmentCount-09362
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, with a VkRenderingInfo::colorAttachmentCount equal to 1, there is no shader object bound to any graphics stage, and a color attachment with a resolve mode of VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID, each element of the VkRenderingInfo::pColorAttachments array with a resolveImageView not equal to VK_NULL_HANDLE **must** have been created with an image created with a VkExternalFormatANDROID::externalFormat value equal to the VkExternalFormatANDROID::externalFormat value used to create the currently bound graphics pipeline
- VUID-vkCmdDrawIndexedIndirect-None-09363
  If there is no shader object bound to any graphics stage, the current render pass instance was begun with vkCmdBeginRenderingKHR and a VkRenderingInfo::colorAttachmentCount equal to 1, and a color attachment with a resolve mode of VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID, each element of the VkRenderingInfo::pColorAttachments array with a imageView not equal to VK_NULL_HANDLE **must** have been created with an image created with a VkExternalFormatANDROID::externalFormat value equal to the VkExternalFormatANDROID::externalFormat value used to create the currently bound graphics pipeline
- VUID-vkCmdDrawIndexedIndirect-None-09364
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is no shader object bound to any graphics stage, and the currently bound graphics pipeline was created with a non-zero VkExternalFormatANDROID::externalFormat value and with the VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT dynamic state enabled, then vkCmdSetColorBlendEnableEXT **must** have set the blend enable to VK_FALSE prior to this drawing command
- VUID-vkCmdDrawIndexedIndirect-None-09365
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is no shader object bound to any graphics stage, and the currently bound graphics pipeline was created with a non-zero VkExternalFormatANDROID::externalFormat value and with the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT dynamic state enabled, then vkCmdSetRasterizationSamplesEXT **must** have set rasterizationSamples to VK_SAMPLE_COUNT_1_BIT prior to this drawing command
- VUID-vkCmdDrawIndexedIndirect-None-09366
  If there is a shader object bound to any graphics stage, and the current render pass includes a color attachment that uses the VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID resolve mode, then vkCmdSetColorBlendEnableEXT **must** have set blend enable to VK_FALSE prior to this drawing command
- VUID-vkCmdDrawIndexedIndirect-rasterizationSamples-09367
  If there is a shader object bound to any graphics stage, and the current render pass includes a color attachment that uses the

VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID resolve mode, then vkCmdSetRasterizationSamplesEXT **must** have set rasterizationSamples to VK_SAMPLE_COUNT_1_BIT prior to this drawing command

- VUID-vkCmdDrawIndexedIndirect-None-09368
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is no shader object bound to any graphics stage, and the currently bound graphics pipeline was created with a non-zero VkExternalFormatANDROID::externalFormat value and with the VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR dynamic state enabled, then vkCmdSetFragmentShadingRateKHR **must** have set pFragmentSize->width to 1 prior to this drawing command
- VUID-vkCmdDrawIndexedIndirect-None-09369
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is no shader object bound to any graphics stage, and the currently bound graphics pipeline was created with a non-zero VkExternalFormatANDROID::externalFormat value and with the VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR dynamic state enabled, then vkCmdSetFragmentShadingRateKHR **must** have set pFragmentSize->height to 1 prior to this drawing command
- VUID-vkCmdDrawIndexedIndirect-pFragmentSize-09370
  If there is a shader object bound to any graphics stage, and the current render pass includes a color attachment that uses the VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID resolve mode, then vkCmdSetFragmentShadingRateKHR **must** have set pFragmentSize->width to 1 prior to this drawing command
- VUID-vkCmdDrawIndexedIndirect-pFragmentSize-09371
  If there is a shader object bound to any graphics stage, and the current render pass includes a color attachment that uses the VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID resolve mode, then vkCmdSetFragmentShadingRateKHR **must** have set pFragmentSize->height to 1 prior to this drawing command
- VUID-vkCmdDrawIndexedIndirect-None-07749
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_WRITE_ENABLE_EXT dynamic state enabled then vkCmdSetColorWriteEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirect-None-08646
  If the colorWriteEnable feature is enabled on the device, and a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetColorWriteEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirect-attachmentCount-07750
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_WRITE_ENABLE_EXT dynamic state enabled then the attachmentCount parameter of vkCmdSetColorWriteEnableEXT **must** be greater than or equal to the

VkPipelineColorBlendStateCreateInfo::attachmentCount of the currently bound graphics pipeline

- VUID-vkCmdDrawIndexedIndirect-None-08647
  If the colorWriteEnable feature is enabled on the device, and a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then the attachmentCount parameter of most recent call to vkCmdSetColorWriteEnableEXT in the current command buffer **must** be greater than or equal to the number of color attachments in the current render pass instance

- VUID-vkCmdDrawIndexedIndirect-None-07751
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_DISCARD_RECTANGLE_EXT dynamic state enabled then vkCmdSetDiscardRectangleEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command for each discard rectangle in VkPipelineDiscardRectangleStateCreateInfoEXT::discardRectangleCount

- VUID-vkCmdDrawIndexedIndirect-None-07880
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_DISCARD_RECTANGLE_ENABLE_EXT dynamic state enabled then vkCmdSetDiscardRectangleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirect-rasterizerDiscardEnable-09236
  If the VK_EXT_discard_rectangles extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetDiscardRectangleEnableEXT in the current command buffer set discardRectangleEnable to VK_TRUE, then vkCmdSetDiscardRectangleEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirect-None-08648
  If the VK_EXT_discard_rectangles extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetDiscardRectangleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirect-None-07881
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_DISCARD_RECTANGLE_MODE_EXT dynamic state enabled then vkCmdSetDiscardRectangleModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirect-None-08649
  If the VK_EXT_discard_rectangles extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetDiscardRectangleEnableEXT in the current command buffer set discardRectangleEnable to VK_TRUE, then vkCmdSetDiscardRectangleModeEXT **must** have

been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirect-dynamicRenderingUnusedAttachments-08913
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is not enabled, and VkRenderingInfo::pDepthAttachment->imageView was VK_NULL_HANDLE, the value of VkPipelineRenderingCreateInfo::depthAttachmentFormat used to create the currently bound graphics pipeline **must** be equal to VK_FORMAT_UNDEFINED
- VUID-vkCmdDrawIndexedIndirect-dynamicRenderingUnusedAttachments-08914
  If current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is not enabled, and VkRenderingInfo::pDepthAttachment->imageView was not VK_NULL_HANDLE, the value of VkPipelineRenderingCreateInfo::depthAttachmentFormat used to create the currently bound graphics pipeline **must** be equal to the VkFormat used to create VkRenderingInfo::pDepthAttachment->imageView
- VUID-vkCmdDrawIndexedIndirect-dynamicRenderingUnusedAttachments-08915
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is enabled, VkRenderingInfo::pDepthAttachment->imageView was not VK_NULL_HANDLE, and the value of VkPipelineRenderingCreateInfo::depthAttachmentFormat used to create the currently bound graphics pipeline was not equal to the VkFormat used to create VkRenderingInfo::pDepthAttachment->imageView, the value of the format **must** be VK_FORMAT_UNDEFINED
- VUID-vkCmdDrawIndexedIndirect-dynamicRenderingUnusedAttachments-08916
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is not enabled, and VkRenderingInfo::pStencilAttachment->imageView was VK_NULL_HANDLE, the value of VkPipelineRenderingCreateInfo::stencilAttachmentFormat used to create the currently bound graphics pipeline **must** be equal to VK_FORMAT_UNDEFINED
- VUID-vkCmdDrawIndexedIndirect-dynamicRenderingUnusedAttachments-08917
  If current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is not enabled, and VkRenderingInfo::pStencilAttachment->imageView was not VK_NULL_HANDLE, the value of VkPipelineRenderingCreateInfo::stencilAttachmentFormat used to create the currently bound graphics pipeline **must** be equal to the VkFormat used to create VkRenderingInfo::pStencilAttachment->imageView
- VUID-vkCmdDrawIndexedIndirect-dynamicRenderingUnusedAttachments-08918
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is enabled, VkRenderingInfo::pStencilAttachment->imageView was not VK_NULL_HANDLE, and the value of VkPipelineRenderingCreateInfo::stencilAttachmentFormat used to create the currently bound graphics pipeline was not equal to the VkFormat used to create VkRenderingInfo::pStencilAttachment->imageView, the value of the format **must** be VK_FORMAT_UNDEFINED
- VUID-vkCmdDrawIndexedIndirect-imageView-06183
  If the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingFragmentShadingRateAttachmentInfoKHR::imageView was not

VK_NULL_HANDLE, the currently bound graphics pipeline **must** have been created with VK_PIPELINE_CREATE_RENDERING_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR

- VUID-vkCmdDrawIndexedIndirect-imageView-06184
  If the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingFragmentDensityMapAttachmentInfoEXT::imageView was not VK_NULL_HANDLE, the currently bound graphics pipeline **must** have been created with VK_PIPELINE_CREATE_RENDERING_FRAGMENT_DENSITY_MAP_ATTACHMENT_BIT_EXT

- VUID-vkCmdDrawIndexedIndirect-colorAttachmentCount-06185
  If the currently bound pipeline was created with a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and the current render pass instance was begun with vkCmdBeginRenderingKHR with a VkRenderingInfo::colorAttachmentCount parameter greater than 0, then each element of the VkRenderingInfo::pColorAttachments array with a imageView not equal to VK_NULL_HANDLE **must** have been created with a sample count equal to the corresponding element of the pColorAttachmentSamples member of VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV used to create the currently bound graphics pipeline

- VUID-vkCmdDrawIndexedIndirect-pDepthAttachment-06186
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the currently bound pipeline was created with a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and VkRenderingInfo::pDepthAttachment->imageView was not VK_NULL_HANDLE, the value of the depthStencilAttachmentSamples member of VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV used to create the currently bound graphics pipeline **must** be equal to the sample count used to create VkRenderingInfo::pDepthAttachment->imageView

- VUID-vkCmdDrawIndexedIndirect-pStencilAttachment-06187
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the currently bound pipeline was created with a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and VkRenderingInfo::pStencilAttachment->imageView was not VK_NULL_HANDLE, the value of the depthStencilAttachmentSamples member of VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV used to create the currently bound graphics pipeline **must** be equal to the sample count used to create VkRenderingInfo::pStencilAttachment->imageView

- VUID-vkCmdDrawIndexedIndirect-multisampledRenderToSingleSampled-07285
  If the currently bound pipeline was created without a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and the multisampledRenderToSingleSampled feature is not enabled, and the current render pass instance was begun with vkCmdBeginRenderingKHR with a VkRenderingInfo::colorAttachmentCount parameter greater than 0, then each element of the VkRenderingInfo::pColorAttachments array with a imageView not equal to VK_NULL_HANDLE **must** have been created with a sample count equal to the value of rasterizationSamples for the currently bound graphics pipeline

- VUID-vkCmdDrawIndexedIndirect-multisampledRenderToSingleSampled-07286
  If the currently bound pipeline was created without a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and the multisampledRenderToSingleSampled feature is not enabled, and VkRenderingInfo

::pDepthAttachment->imageView was not VK_NULL_HANDLE, the value of rasterizationSamples for the currently bound graphics pipeline **must** be equal to the sample count used to create VkRenderingInfo::pDepthAttachment->imageView

- VUID-vkCmdDrawIndexedIndirect-multisampledRenderToSingleSampled-07287
  If the currently bound pipeline was created without a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and the multisampledRenderToSingleSampled feature is not enabled, and VkRenderingInfo::pStencilAttachment->imageView was not VK_NULL_HANDLE, the value of rasterizationSamples for the currently bound graphics pipeline **must** be equal to the sample count used to create VkRenderingInfo::pStencilAttachment->imageView

- VUID-vkCmdDrawIndexedIndirect-pNext-07935
  If this command has been called inside a render pass instance started with vkCmdBeginRenderingKHR, and the pNext chain of VkRenderingInfo includes a VkMultisampledRenderToSingleSampledInfoEXT structure with multisampledRenderToSingleSampledEnable equal to VK_TRUE, then the value of rasterizationSamples for the currently bound graphics pipeline **must** be equal to VkMultisampledRenderToSingleSampledInfoEXT::rasterizationSamples

- VUID-vkCmdDrawIndexedIndirect-renderPass-06198
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the currently bound pipeline **must** have been created with a VkGraphicsPipelineCreateInfo::renderPass equal to VK_NULL_HANDLE

- VUID-vkCmdDrawIndexedIndirect-pColorAttachments-08963
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is a graphics pipeline bound with a fragment shader that statically writes to a color attachment, the color write mask is not zero, color writes are enabled, and the corresponding element of the VkRenderingInfo::pColorAttachments->imageView was not VK_NULL_HANDLE, then the corresponding element of VkPipelineRenderingCreateInfo::pColorAttachmentFormats used to create the pipeline **must** not be VK_FORMAT_UNDEFINED

- VUID-vkCmdDrawIndexedIndirect-pDepthAttachment-08964
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is a graphics pipeline bound, depth test is enabled, depth write is enabled, and the VkRenderingInfo::pDepthAttachment->imageView was not VK_NULL_HANDLE, then the VkPipelineRenderingCreateInfo::depthAttachmentFormat used to create the pipeline **must** not be VK_FORMAT_UNDEFINED

- VUID-vkCmdDrawIndexedIndirect-pStencilAttachment-08965
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is a graphics pipeline bound, stencil test is enabled and the VkRenderingInfo::pStencilAttachment->imageView was not VK_NULL_HANDLE, then the VkPipelineRenderingCreateInfo::stencilAttachmentFormat used to create the pipeline **must** not be VK_FORMAT_UNDEFINED

- VUID-vkCmdDrawIndexedIndirect-primitivesGeneratedQueryWithRasterizerDiscard-06708
  If the primitivesGeneratedQueryWithRasterizerDiscard feature is not enabled and the VK_QUERY_TYPE_PRIMITIVES_GENERATED_EXT query is active, rasterization discard **must** not be enabled

- VUID-vkCmdDrawIndexedIndirect-primitivesGeneratedQueryWithNonZeroStreams-06709
  If the `primitivesGeneratedQueryWithNonZeroStreams` feature is not enabled and the `VK_QUERY_TYPE_PRIMITIVES_GENERATED_EXT` query is active, the bound graphics pipeline **must** not have been created with a non-zero value in `VkPipelineRasterizationStateStreamCreateInfoEXT::rasterizationStream`
- VUID-vkCmdDrawIndexedIndirect-None-07619
  If a shader object is bound to the `VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT` stage or a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_TESSELLATION_DOMAIN_ORIGIN_EXT` dynamic state enabled, then vkCmdSetTessellationDomainOriginEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirect-None-07620
  If the `depthClamp` feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_DEPTH_CLAMP_ENABLE_EXT` dynamic state enabled, and the current value of `rasterizerDiscardEnable` is `VK_FALSE`, then vkCmdSetDepthClampEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirect-None-07621
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_POLYGON_MODE_EXT` dynamic state enabled, and the current value of `rasterizerDiscardEnable` is `VK_FALSE`, then vkCmdSetPolygonModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirect-None-07622
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT` dynamic state enabled, and the current value of `rasterizerDiscardEnable` is `VK_FALSE`, then vkCmdSetRasterizationSamplesEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirect-None-07623
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_SAMPLE_MASK_EXT` dynamic state enabled, and the current value of `rasterizerDiscardEnable` is `VK_FALSE`, then vkCmdSetSampleMaskEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirect-alphaToCoverageEnable-08919
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_ALPHA_TO_COVERAGE_ENABLE_EXT` dynamic state enabled, and `alphaToCoverageEnable` was `VK_TRUE` in the last call to vkCmdSetAlphaToCoverageEnableEXT, then the Fragment Output Interface **must** contain a variable for the alpha `Component` word in `Location` 0 at `Index` 0
- VUID-vkCmdDrawIndexedIndirect-alphaToCoverageEnable-08920
  If a shader object is bound to any graphics stage, and the most recent call to

vkCmdSetAlphaToCoverageEnableEXT in the current command buffer set alphaToCoverageEnable to VK_TRUE, then the Fragment Output Interface **must** contain a variable for the alpha Component word in Location 0 at Index 0

- VUID-vkCmdDrawIndexedIndirect-None-07624
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_ALPHA_TO_COVERAGE_ENABLE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetAlphaToCoverageEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirect-None-07625
  If the alphaToOne feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_ALPHA_TO_ONE_ENABLE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetAlphaToOneEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirect-None-07626
  If the logicOp feature is enabled, a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_LOGIC_OP_ENABLE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetLogicOpEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirect-None-07627
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT dynamic state enabled then vkCmdSetColorBlendEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirect-None-08657
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and both the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE and there are color attachments bound, then vkCmdSetColorBlendEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirect-None-07628
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_EQUATION_EXT dynamic state enabled then vkCmdSetColorBlendEquationEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirect-None-08658
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetColorBlendEnableEXT for any attachment set that attachment's value in pColorBlendEnables to VK_TRUE, then vkCmdSetColorBlendEquationEXT **must** have been

called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirect-None-07629
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_WRITE_MASK_EXT dynamic state enabled then vkCmdSetColorWriteMaskEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirect-None-08659
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and both the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE and there are color attachments bound, then vkCmdSetColorWriteMaskEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirect-None-07630
  If the geometryStreams feature is enabled, and a shader object is bound to the VK_SHADER_STAGE_GEOMETRY_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_RASTERIZATION_STREAM_EXT dynamic state enabled, then vkCmdSetRasterizationStreamEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirect-None-07631
  If the VK_EXT_conservative_rasterization extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_CONSERVATIVE_RASTERIZATION_MODE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetConservativeRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirect-None-07632
  If the VK_EXT_conservative_rasterization extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_EXTRA_PRIMITIVE_OVERESTIMATION_SIZE_EXT dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of conservativeRasterizationMode is VK_CONSERVATIVE_RASTERIZATION_MODE_OVERESTIMATE_EXT, then vkCmdSetExtraPrimitiveOverestimationSizeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirect-None-07633
  If the depthClipEnable feature is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_CLIP_ENABLE_EXT dynamic state, then vkCmdSetDepthClipEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirect-None-07634
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT dynamic state enabled then vkCmdSetSampleLocationsEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirect-None-08664
  If the VK_EXT_sample_locations extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetSampleLocationsEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirect-None-07635
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_ADVANCED_EXT dynamic state enabled then vkCmdSetColorBlendAdvancedEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirect-rasterizerDiscardEnable-09416
  If the VK_EXT_blend_operation_advanced extension is enabled, and a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then at least one of vkCmdSetColorBlendEquationEXT and vkCmdSetColorBlendAdvancedEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirect-None-07636
  If the VK_EXT_provoking_vertex extension is enabled, a shader object is bound to the VK_SHADER_STAGE_VERTEX_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_PROVOKING_VERTEX_MODE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetProvokingVertexModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirect-None-07637
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT dynamic state enabled then vkCmdSetLineRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirect-None-08666
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetPolygonModeEXT in the current command buffer set polygonMode to VK_POLYGON_MODE_LINE, then vkCmdSetLineRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirect-None-08667
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object is bound to the VK_SHADER_STAGE_VERTEX_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to

vkCmdSetPrimitiveTopologyEXT in the current command buffer set primitiveTopology to any line topology, then vkCmdSetLineRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirect-None-08668
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object that outputs line primitives is bound to the VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT or VK_SHADER_STAGE_GEOMETRY_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetLineRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirect-None-07638
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT dynamic state enabled then vkCmdSetLineStippleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirect-None-08669
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetPolygonModeEXT in the current command buffer set polygonMode to VK_POLYGON_MODE_LINE, then vkCmdSetLineStippleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirect-None-08670
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object is bound to the VK_SHADER_STAGE_VERTEX_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetPrimitiveTopologyEXT in the current command buffer set primitiveTopology to any line topology, then vkCmdSetLineStippleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirect-None-08671
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object that outputs line primitives is bound to the VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT or VK_SHADER_STAGE_GEOMETRY_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetLineStippleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirect-None-07849
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_KHR dynamic state enabled then vkCmdSetLineStippleKHR

**must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirect-None-08672
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetLineStippleEnableEXT in the current command buffer set stippledLineEnable to VK_TRUE, then vkCmdSetLineStippleEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirect-None-07639
  If the depthClipControl feature is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_CLIP_NEGATIVE_ONE_TO_ONE_EXT dynamic state enabled, then vkCmdSetDepthClipNegativeOneToOneEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirect-None-07640
  If the VK_NV_clip_space_w_scaling extension is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_VIEWPORT_W_SCALING_ENABLE_NV dynamic state enabled, then vkCmdSetViewportWScalingEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirect-None-07641
  If the VK_NV_viewport_swizzle extension is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_VIEWPORT_SWIZZLE_NV dynamic state enabled, then vkCmdSetViewportSwizzleNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirect-None-07642
  If the VK_NV_fragment_coverage_to_color extension is enabled, a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_TO_COLOR_ENABLE_NV dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetCoverageToColorEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirect-None-07643
  If the VK_NV_fragment_coverage_to_color extension is enabled, a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_TO_COLOR_LOCATION_NV dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of coverageToColorEnable is VK_TRUE, then vkCmdSetCoverageToColorLocationNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirect-None-07644
  If the VK_NV_framebuffer_mixed_samples extension is enabled, a shader object is bound to

any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_MODULATION_MODE_NV dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetCoverageModulationModeNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirect-None-07645
  If the VK_NV_framebuffer_mixed_samples extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_MODULATION_TABLE_ENABLE_NV dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of coverageModulationMode is any value other than VK_COVERAGE_MODULATION_MODE_NONE_NV, then vkCmdSetCoverageModulationTableEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirect-None-07646
  If the VK_NV_framebuffer_mixed_samples extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_MODULATION_TABLE_NV dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of coverageModulationTableEnable is VK_TRUE, then vkCmdSetCoverageModulationTableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirect-None-07647
  If the shadingRateImage feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_SHADING_RATE_IMAGE_ENABLE_NV dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetShadingRateImageEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirect-pipelineFragmentShadingRate-09238
  If the pipelineFragmentShadingRate feature is enabled, a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetFragmentShadingRateKHR **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirect-None-07648
  If the representativeFragmentTest feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_REPRESENTATIVE_FRAGMENT_TEST_ENABLE_NV dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetRepresentativeFragmentTestEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirect-None-07649
  If the coverageReductionMode feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the

VK_DYNAMIC_STATE_COVERAGE_REDUCTION_MODE_NV dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetCoverageReductionModeNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirect-pColorBlendEnables-07470
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT state enabled and the last call to vkCmdSetColorBlendEnableEXT set pColorBlendEnables for any attachment to VK_TRUE, then for those attachments in the subpass the corresponding image view's format features **must** contain VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT

- VUID-vkCmdDrawIndexedIndirect-rasterizationSamples-07471
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, and the current subpass does not use any color and/or depth/stencil attachments, then the rasterizationSamples in the last call to vkCmdSetRasterizationSamplesEXT **must** follow the rules for a zero-attachment subpass

- VUID-vkCmdDrawIndexedIndirect-samples-07472
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_MASK_EXT state enabled and the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state disabled, then the samples parameter in the last call to vkCmdSetSampleMaskEXT **must** be greater or equal to the VkPipelineMultisampleStateCreateInfo::rasterizationSamples parameter used to create the bound graphics pipeline

- VUID-vkCmdDrawIndexedIndirect-samples-07473
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_MASK_EXT state and VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT states enabled, then the samples parameter in the last call to vkCmdSetSampleMaskEXT **must** be greater or equal to the rasterizationSamples parameter in the last call to vkCmdSetRasterizationSamplesEXT

- VUID-vkCmdDrawIndexedIndirect-rasterizationSamples-07474
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, and neither the VK_AMD_mixed_attachment_samples nor the VK_NV_framebuffer_mixed_samples extensions are enabled, then the rasterizationSamples in the last call to vkCmdSetRasterizationSamplesEXT **must** be the same as the current subpass color and/or depth/stencil attachments

- VUID-vkCmdDrawIndexedIndirect-None-09211
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, or a shader object is bound to any graphics stage, and the current render pass instance includes a VkMultisampledRenderToSingleSampledInfoEXT structure with multisampledRenderToSingleSampledEnable equal to VK_TRUE, then the rasterizationSamples in the last call to vkCmdSetRasterizationSamplesEXT **must** be the same as the rasterizationSamples member of that structure

- VUID-vkCmdDrawIndexedIndirect-firstAttachment-07476

If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT dynamic state enabled then vkCmdSetColorBlendEnableEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the firstAttachment and attachmentCount parameters of vkCmdSetColorBlendEnableEXT calls **must** specify an enable for all active color attachments in the current subpass

- VUID-vkCmdDrawIndexedIndirect-rasterizerDiscardEnable-09417
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetColorBlendEnableEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the firstAttachment and attachmentCount parameters of vkCmdSetColorBlendEnableEXT calls **must** specify an enable for all active color attachments in the current subpass

- VUID-vkCmdDrawIndexedIndirect-firstAttachment-07477
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_EQUATION_EXT dynamic state enabled then vkCmdSetColorBlendEquationEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the firstAttachment and attachmentCount parameters of vkCmdSetColorBlendEquationEXT calls **must** specify the blend equations for all active color attachments in the current subpass where blending is enabled

- VUID-vkCmdDrawIndexedIndirect-rasterizerDiscardEnable-09418
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and both the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE and there are color attachments bound, then vkCmdSetColorBlendEquationEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the firstAttachment and attachmentCount parameters of vkCmdSetColorBlendEquationEXT calls **must** specify the blend equations for all active color attachments in the current subpass where blending is enabled

- VUID-vkCmdDrawIndexedIndirect-firstAttachment-07478
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_WRITE_MASK_EXT dynamic state enabled then vkCmdSetColorWriteMaskEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the firstAttachment and attachmentCount parameters of vkCmdSetColorWriteMaskEXT calls **must** specify the color write mask for all active color attachments in the current subpass

- VUID-vkCmdDrawIndexedIndirect-rasterizerDiscardEnable-09419
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetColorWriteMaskEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the firstAttachment and attachmentCount parameters of vkCmdSetColorWriteMaskEXT calls **must** specify the color write mask for all active color attachments in the current subpass

- VUID-vkCmdDrawIndexedIndirect-firstAttachment-07479
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_ADVANCED_EXT dynamic state enabled then vkCmdSetColorBlendAdvancedEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the firstAttachment and attachmentCount parameters of vkCmdSetColorBlendAdvancedEXT calls **must** specify the advanced blend equations for all active color attachments in the current subpass where blending is enabled
- VUID-vkCmdDrawIndexedIndirect-advancedBlendMaxColorAttachments-07480
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_ADVANCED_EXT and VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT dynamic states enabled and the last calls to vkCmdSetColorBlendEnableEXT and vkCmdSetColorBlendAdvancedEXT have enabled advanced blending, then the number of active color attachments in the current subpass **must** not exceed advancedBlendMaxColorAttachments
- VUID-vkCmdDrawIndexedIndirect-primitivesGeneratedQueryWithNonZeroStreams-07481
  If the primitivesGeneratedQueryWithNonZeroStreams feature is not enabled and the VK_QUERY_TYPE_PRIMITIVES_GENERATED_EXT query is active, and the bound graphics pipeline was created with VK_DYNAMIC_STATE_RASTERIZATION_STREAM_EXT state enabled, the last call to vkCmdSetRasterizationStreamEXT **must** have set the rasterizationStream to zero
- VUID-vkCmdDrawIndexedIndirect-sampleLocationsPerPixel-07482
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state enabled and the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state disabled, then the sampleLocationsPerPixel member of pSampleLocationsInfo in the last call to vkCmdSetSampleLocationsEXT **must** equal the rasterizationSamples member of the VkPipelineMultisampleStateCreateInfo structure the bound graphics pipeline has been created with
- VUID-vkCmdDrawIndexedIndirect-sampleLocationsPerPixel-07483
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state enabled and the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, then the sampleLocationsPerPixel member of pSampleLocationsInfo in the last call to vkCmdSetSampleLocationsEXT **must** equal the rasterizationSamples parameter of the last call to vkCmdSetRasterizationSamplesEXT
- VUID-vkCmdDrawIndexedIndirect-sampleLocationsEnable-07484
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, or the bound graphics pipeline was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, and sampleLocationsEnable was VK_TRUE in the last call to vkCmdSetSampleLocationsEnableEXT, and the current subpass has a depth/stencil attachment, then that attachment **must** have been created with the VK_IMAGE_CREATE_SAMPLE_LOCATIONS_COMPATIBLE_DEPTH_BIT_EXT bit set
- VUID-vkCmdDrawIndexedIndirect-sampleLocationsEnable-07485
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, or the bound

graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state enabled and the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, and if sampleLocationsEnable was VK_TRUE in the last call to vkCmdSetSampleLocationsEnableEXT, then the sampleLocationsInfo.sampleLocationGridSize.width in the last call to vkCmdSetSampleLocationsEXT **must** evenly divide VkMultisamplePropertiesEXT::sampleLocationGridSize.width as returned by vkGetPhysicalDeviceMultisamplePropertiesEXT with a samples parameter equaling rasterizationSamples

- VUID-vkCmdDrawIndexedIndirect-sampleLocationsEnable-07486
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state enabled and the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, and if sampleLocationsEnable was VK_TRUE in the last call to vkCmdSetSampleLocationsEnableEXT, then the sampleLocationsInfo.sampleLocationGridSize.height in the last call to vkCmdSetSampleLocationsEXT **must** evenly divide VkMultisamplePropertiesEXT::sampleLocationGridSize.height as returned by vkGetPhysicalDeviceMultisamplePropertiesEXT with a samples parameter equaling rasterizationSamples

- VUID-vkCmdDrawIndexedIndirect-sampleLocationsEnable-07487
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, and if sampleLocationsEnable was VK_TRUE in the last call to vkCmdSetSampleLocationsEnableEXT, the fragment shader code **must** not statically use the extended instruction InterpolateAtSample

- VUID-vkCmdDrawIndexedIndirect-sampleLocationsEnable-07936
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state disabled and the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, the sampleLocationsEnable member of a VkPipelineSampleLocationsStateCreateInfoEXT::sampleLocationsEnable in the bound graphics pipeline is VK_TRUE or VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, then, sampleLocationsInfo.sampleLocationGridSize.width **must** evenly divide VkMultisamplePropertiesEXT::sampleLocationGridSize.width as returned by vkGetPhysicalDeviceMultisamplePropertiesEXT with a samples parameter equaling the value of rasterizationSamples in the last call to vkCmdSetRasterizationSamplesEXT

- VUID-vkCmdDrawIndexedIndirect-sampleLocationsEnable-07937
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state disabled and the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, the sampleLocationsEnable member of a VkPipelineSampleLocationsStateCreateInfoEXT::sampleLocationsEnable in the bound graphics pipeline is VK_TRUE or VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, then, sampleLocationsInfo.sampleLocationGridSize.height **must** evenly divide VkMultisamplePropertiesEXT::sampleLocationGridSize.height as returned by vkGetPhysicalDeviceMultisamplePropertiesEXT with a samples parameter equaling the value of rasterizationSamples in the last call to vkCmdSetRasterizationSamplesEXT

- VUID-vkCmdDrawIndexedIndirect-sampleLocationsEnable-07938
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state disabled and the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, the sampleLocationsEnable member of a VkPipelineSampleLocationsStateCreateInfoEXT::sampleLocationsEnable in the bound graphics pipeline is VK_TRUE or VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, then, sampleLocationsInfo.sampleLocationsPerPixel **must** equal rasterizationSamples in the last call to vkCmdSetRasterizationSamplesEXT
- VUID-vkCmdDrawIndexedIndirect-coverageModulationTableEnable-07488
  If a shader object is bound to any graphics stage or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COVERAGE_MODULATION_TABLE_ENABLE_NV state enabled, and the last call to vkCmdSetCoverageModulationTableEnableNV set coverageModulationTableEnable to VK_TRUE, then the coverageModulationTableCount parameter in the last call to vkCmdSetCoverageModulationTableNV **must** equal the current rasterizationSamples divided by the number of color samples in the current subpass
- VUID-vkCmdDrawIndexedIndirect-rasterizationSamples-07489
  If the VK_NV_framebuffer_mixed_samples extension is enabled, and if current subpass has a depth/stencil attachment and depth test, stencil test, or depth bounds test are enabled in the currently bound pipeline state, then the current rasterizationSamples **must** be the same as the sample count of the depth/stencil attachment
- VUID-vkCmdDrawIndexedIndirect-coverageToColorEnable-07490
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COVERAGE_TO_COLOR_ENABLE_NV state enabled and the last call to vkCmdSetCoverageToColorEnableNV set the coverageToColorEnable to VK_TRUE, then the current subpass **must** have a color attachment at the location selected by the last call to vkCmdSetCoverageToColorLocationNV coverageToColorLocation, with a VkFormat of VK_FORMAT_R8_UINT, VK_FORMAT_R8_SINT, VK_FORMAT_R16_UINT, VK_FORMAT_R16_SINT, VK_FORMAT_R32_UINT, or VK_FORMAT_R32_SINT
- VUID-vkCmdDrawIndexedIndirect-rasterizerDiscardEnable-09420
  If the VK_NV_fragment_coverage_to_color extension is enabled, and a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the last call to vkCmdSetCoverageToColorEnableNV set the coverageToColorEnable to VK_TRUE, then the current subpass **must** have a color attachment at the location selected by the last call to vkCmdSetCoverageToColorLocationNV coverageToColorLocation, with a VkFormat of VK_FORMAT_R8_UINT, VK_FORMAT_R8_SINT, VK_FORMAT_R16_UINT, VK_FORMAT_R16_SINT, VK_FORMAT_R32_UINT, or VK_FORMAT_R32_SINT
- VUID-vkCmdDrawIndexedIndirect-coverageReductionMode-07491
  If this VK_NV_coverage_reduction_mode extension is enabled, the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COVERAGE_TO_COLOR_ENABLE_NV and VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT states enabled, the current coverage reduction mode coverageReductionMode, then the current rasterizationSamples, and the sample counts for the color and depth/stencil attachments (if the subpass has them) **must** be a valid combination returned by

vkGetPhysicalDeviceSupportedFramebufferMixedSamplesCombinationsNV

- VUID-vkCmdDrawIndexedIndirect-viewportCount-07492
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT` dynamic state enabled, but not the `VK_DYNAMIC_STATE_VIEWPORT_SWIZZLE_NV` dynamic state enabled, then the bound graphics pipeline **must** have been created with VkPipelineViewportSwizzleStateCreateInfoNV ::`viewportCount` greater or equal to the `viewportCount` parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawIndexedIndirect-viewportCount-07493
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT` and `VK_DYNAMIC_STATE_VIEWPORT_SWIZZLE_NV` dynamic states enabled then the `viewportCount` parameter in the last call to vkCmdSetViewportSwizzleNV **must** be greater than or equal to the `viewportCount` parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawIndexedIndirect-viewportCount-09421
  If the `VK_NV_viewport_swizzle` extension is enabled, and a shader object is bound to any graphics stage, then the `viewportCount` parameter in the last call to vkCmdSetViewportSwizzleNV **must** be greater than or equal to the `viewportCount` parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawIndexedIndirect-rasterizationSamples-07494
  If the `VK_NV_framebuffer_mixed_samples` extension is enabled, and if the current subpass has any color attachments and `rasterizationSamples` of the last call to vkCmdSetRasterizationSamplesEXT is greater than the number of color samples, then the pipeline `sampleShadingEnable` **must** be `VK_FALSE`
- VUID-vkCmdDrawIndexedIndirect-stippledLineEnable-07495
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT` or `VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT` dynamic states enabled, and if the current `stippledLineEnable` state is `VK_TRUE` and the current `lineRasterizationMode` state is `VK_LINE_RASTERIZATION_MODE_RECTANGULAR_KHR`, then the `stippledRectangularLines` feature **must** be enabled
- VUID-vkCmdDrawIndexedIndirect-stippledLineEnable-07496
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT` or `VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT` dynamic states enabled, and if the current `stippledLineEnable` state is `VK_TRUE` and the current `lineRasterizationMode` state is `VK_LINE_RASTERIZATION_MODE_BRESENHAM_KHR`, then the `stippledBresenhamLines` feature **must** be enabled
- VUID-vkCmdDrawIndexedIndirect-stippledLineEnable-07497
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT` or `VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT` dynamic states enabled, and if the current `stippledLineEnable` state is `VK_TRUE` and the current `lineRasterizationMode` state is `VK_LINE_RASTERIZATION_MODE_RECTANGULAR_SMOOTH_KHR`, then the `stippledSmoothLines` feature **must** be enabled

- VUID-vkCmdDrawIndexedIndirect-stippledLineEnable-07498
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT or VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT dynamic states enabled, and if the current stippledLineEnable state is VK_TRUE and the current lineRasterizationMode state is VK_LINE_RASTERIZATION_MODE_DEFAULT_KHR, then the stippledRectangularLines feature **must** be enabled and VkPhysicalDeviceLimits::strictLines **must** be VK_TRUE
- VUID-vkCmdDrawIndexedIndirect-conservativePointAndLineRasterization-07499
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_CONSERVATIVE_RASTERIZATION_MODE_EXT dynamic state enabled, conservativePointAndLineRasterization is not supported, and the effective primitive topology output by the last pre-rasterization shader stage is a line or point, then the conservativeRasterizationMode set by the last call to vkCmdSetConservativeRasterizationModeEXT **must** be VK_CONSERVATIVE_RASTERIZATION_MODE_DISABLED_EXT
- VUID-vkCmdDrawIndexedIndirect-stage-07073
  If the currently bound pipeline was created with the VkPipelineShaderStageCreateInfo::stage member of an element of VkGraphicsPipelineCreateInfo::pStages set to VK_SHADER_STAGE_VERTEX_BIT, VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT, VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT or VK_SHADER_STAGE_GEOMETRY_BIT, then Mesh Shader Queries **must** not be active
- VUID-vkCmdDrawIndexedIndirect-None-08877
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_ATTACHMENT_FEEDBACK_LOOP_ENABLE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetAttachmentFeedbackLoopEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirect-None-07850
  If dynamic state was inherited from VkCommandBufferInheritanceViewportScissorInfoNV, it **must** be set in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirect-None-08684
  If there is no bound graphics pipeline, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_VERTEX_BIT
- VUID-vkCmdDrawIndexedIndirect-None-08685
  If there is no bound graphics pipeline, and the tessellationShader feature is enabled, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT
- VUID-vkCmdDrawIndexedIndirect-None-08686
  If there is no bound graphics pipeline, and the tessellationShader feature is enabled, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT
- VUID-vkCmdDrawIndexedIndirect-None-08687
  If there is no bound graphics pipeline, and the geometryShader feature is enabled,

vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_GEOMETRY_BIT

- VUID-vkCmdDrawIndexedIndirect-None-08688
  If there is no bound graphics pipeline, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_FRAGMENT_BIT
- VUID-vkCmdDrawIndexedIndirect-None-08689
  If there is no bound graphics pipeline, and the taskShader feature is enabled, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_TASK_BIT_EXT
- VUID-vkCmdDrawIndexedIndirect-None-08690
  If there is no bound graphics pipeline, and the meshShader feature is enabled, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_MESH_BIT_EXT
- VUID-vkCmdDrawIndexedIndirect-None-08693
  If there is no bound graphics pipeline, and at least one of the taskShader and meshShader features is enabled, one of the VK_SHADER_STAGE_VERTEX_BIT or VK_SHADER_STAGE_MESH_BIT_EXT stages **must** have a valid VkShaderEXT bound, and the other **must** have no VkShaderEXT bound
- VUID-vkCmdDrawIndexedIndirect-None-08694
  If there is no bound graphics pipeline, and both the taskShader and meshShader features are enabled, and a valid VkShaderEXT is bound the to the VK_SHADER_STAGE_MESH_BIT_EXT stage, and that VkShaderEXT was created without the VK_SHADER_CREATE_NO_TASK_SHADER_BIT_EXT flag, a valid VkShaderEXT **must** be bound to the VK_SHADER_STAGE_TASK_BIT_EXT stage
- VUID-vkCmdDrawIndexedIndirect-None-08695
  If there is no bound graphics pipeline, and both the taskShader and meshShader features are enabled, and a valid VkShaderEXT is bound the to the VK_SHADER_STAGE_MESH_BIT_EXT stage, and that VkShaderEXT was created with the VK_SHADER_CREATE_NO_TASK_SHADER_BIT_EXT flag, there **must** be no VkShaderEXT bound to the VK_SHADER_STAGE_TASK_BIT_EXT stage
- VUID-vkCmdDrawIndexedIndirect-None-08696
  If there is no bound graphics pipeline, and a valid VkShaderEXT is bound to the VK_SHADER_STAGE_VERTEX_BIT stage, there **must** be no VkShaderEXT bound to either the VK_SHADER_STAGE_TASK_BIT_EXT stage or the VK_SHADER_STAGE_MESH_BIT_EXT stage
- VUID-vkCmdDrawIndexedIndirect-None-08698
  If any graphics shader is bound which was created with the VK_SHADER_CREATE_LINK_STAGE_BIT_EXT flag, then all shaders created with the VK_SHADER_CREATE_LINK_STAGE_BIT_EXT flag in the same vkCreateShadersEXT call **must** also be bound
- VUID-vkCmdDrawIndexedIndirect-None-08699
  If any graphics shader is bound which was created with the VK_SHADER_CREATE_LINK_STAGE_BIT_EXT flag, any stages in between stages whose shaders which did not create a shader with the VK_SHADER_CREATE_LINK_STAGE_BIT_EXT flag as part of the same vkCreateShadersEXT call **must** not have any VkShaderEXT bound
- VUID-vkCmdDrawIndexedIndirect-None-08878

All bound graphics shader objects **must** have been created with identical or identically defined push constant ranges

- VUID-vkCmdDrawIndexedIndirect-None-08879
  All bound graphics shader objects **must** have been created with identical or identically defined arrays of descriptor set layouts

- VUID-vkCmdDrawIndexedIndirect-colorAttachmentCount-09372
  If the current render pass instance was begun with vkCmdBeginRenderingKHR and a VkRenderingInfo::`colorAttachmentCount` equal to `1`, a color attachment with a resolve mode of `VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID`, and a fragment shader is bound, it **must** not declare the `DepthReplacing` or `StencilRefReplacingEXT` execution modes

- VUID-vkCmdDrawIndexedIndirect-pDynamicStates-08715
  If the bound graphics pipeline state includes a fragment shader stage, was created with `VK_DYNAMIC_STATE_DEPTH_WRITE_ENABLE` set in VkPipelineDynamicStateCreateInfo::`pDynamicStates`, and the fragment shader declares the `EarlyFragmentTests` execution mode and uses `OpDepthAttachmentReadEXT`, the `depthWriteEnable` parameter in the last call to vkCmdSetDepthWriteEnableEXT **must** be `VK_FALSE`

- VUID-vkCmdDrawIndexedIndirect-pDynamicStates-08716
  If the bound graphics pipeline state includes a fragment shader stage, was created with `VK_DYNAMIC_STATE_STENCIL_WRITE_MASK` set in VkPipelineDynamicStateCreateInfo::`pDynamicStates`, and the fragment shader declares the `EarlyFragmentTests` execution mode and uses `OpStencilAttachmentReadEXT`, the `writeMask` parameter in the last call to vkCmdSetStencilWriteMask **must** be `0`

- VUID-vkCmdDrawIndexedIndirect-None-09116
  If a shader object is bound to any graphics stage or the currently bound graphics pipeline was created with `VK_DYNAMIC_STATE_COLOR_WRITE_MASK_EXT`, and the format of any color attachment is `VK_FORMAT_E5B9G9R9_UFLOAT_PACK32`, the corresponding element of the `pColorWriteMasks` parameter of vkCmdSetColorWriteMaskEXT **must** either include all of `VK_COLOR_COMPONENT_R_BIT`, `VK_COLOR_COMPONENT_G_BIT`, and `VK_COLOR_COMPONENT_B_BIT`, or none of them

- VUID-vkCmdDrawIndexedIndirect-maxFragmentDualSrcAttachments-09239
  If blending is enabled for any attachment where either the source or destination blend factors for that attachment use the secondary color input, the maximum value of `Location` for any output attachment statically used in the `Fragment Execution Model` executed by this command **must** be less than `maxFragmentDualSrcAttachments`

- VUID-vkCmdDrawIndexedIndirect-None-09548
  If the current render pass was begun with vkCmdBeginRenderingKHR, and there is no shader object bound to any graphics stage, the value of each element of VkRenderingAttachmentLocationInfoKHR::`pColorAttachmentLocations` set by vkCmdSetRenderingAttachmentLocationsKHR **must** match the value set for the corresponding element in the currently bound pipeline

- VUID-vkCmdDrawIndexedIndirect-None-09549
  If the current render pass was begun with vkCmdBeginRenderingKHR, and there is no shader object bound to any graphics stage, input attachment index mappings in the currently bound pipeline **must** match those set for the current render pass instance via VkRenderingInputAttachmentIndexInfoKHR

- VUID-vkCmdDrawIndexedIndirect-None-09642
  If the current render pass was begun with vkCmdBeginRenderingKHR with the VK_RENDERING_ENABLE_LEGACY_DITHERING_BIT_EXT flag, the bound graphics pipeline **must** have been created with VK_PIPELINE_CREATE_2_ENABLE_LEGACY_DITHERING_BIT_EXT
- VUID-vkCmdDrawIndexedIndirect-None-09643
  If the bound graphics pipeline was created with VK_PIPELINE_CREATE_2_ENABLE_LEGACY_DITHERING_BIT_EXT, the current render pass **must** have begun with vkCmdBeginRenderingKHR with the VK_RENDERING_ENABLE_LEGACY_DITHERING_BIT_EXT flag
- VUID-vkCmdDrawIndexedIndirect-None-04007
  All vertex input bindings accessed via vertex input variables declared in the vertex shader entry point's interface **must** have either valid or VK_NULL_HANDLE buffers bound
- VUID-vkCmdDrawIndexedIndirect-None-04008
  If the nullDescriptor feature is not enabled, all vertex input bindings accessed via vertex input variables declared in the vertex shader entry point's interface **must** not be VK_NULL_HANDLE
- VUID-vkCmdDrawIndexedIndirect-None-02721
  If robustBufferAccess is not enabled, and that pipeline was created without enabling VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_EXT for vertexInputs, then for a given vertex buffer binding, any attribute data fetched **must** be entirely contained within the corresponding vertex buffer binding, as described in Vertex Input Description
- VUID-vkCmdDrawIndexedIndirect-None-07842
  If there is a shader object bound to the VK_SHADER_STAGE_VERTEX_BIT stage or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_PRIMITIVE_TOPOLOGY dynamic state enabled then vkCmdSetPrimitiveTopologyEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirect-dynamicPrimitiveTopologyUnrestricted-07500
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_PRIMITIVE_TOPOLOGY dynamic state enabled and the dynamicPrimitiveTopologyUnrestricted is VK_FALSE, then the primitiveTopology parameter of vkCmdSetPrimitiveTopologyEXT **must** be of the same topology class as the pipeline VkPipelineInputAssemblyStateCreateInfo::topology state
- VUID-vkCmdDrawIndexedIndirect-pStrides-04913
  If the bound graphics pipeline was created with the VK_DYNAMIC_STATE_VERTEX_INPUT_BINDING_STRIDE_EXT dynamic state enabled, but without the VK_DYNAMIC_STATE_VERTEX_INPUT_EXT dynamic state enabled, then vkCmdBindVertexBuffers2EXT **must** have been called and not subsequently invalidated in the current command buffer prior to this draw command, and the pStrides parameter of vkCmdBindVertexBuffers2EXT **must** not be NULL
- VUID-vkCmdDrawIndexedIndirect-None-04914
  If there is a shader object bound to the VK_SHADER_STAGE_VERTEX_BIT stage or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VERTEX_INPUT_EXT dynamic

state enabled then vkCmdSetVertexInputEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this draw command

- VUID-vkCmdDrawIndexedIndirect-Input-07939
  If there is a shader object bound to the VK_SHADER_STAGE_VERTEX_BIT stage or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VERTEX_INPUT_EXT dynamic state enabled then all variables with the Input storage class decorated with Location in the Vertex Execution Model OpEntryPoint **must** contain a location in VkVertexInputAttributeDescription2EXT::location

- VUID-vkCmdDrawIndexedIndirect-Input-08734
  If there is a shader object bound to the VK_SHADER_STAGE_VERTEX_BIT stage or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VERTEX_INPUT_EXT dynamic state enabled and either legacyVertexAttributes is not enabled or the SPIR-V Type associated with a given Input variable of the corresponding Location in the Vertex Execution Model OpEntryPoint is 64-bit, then the numeric type associated with all Input variables of the corresponding Location in the Vertex Execution Model OpEntryPoint **must** be the same as VkVertexInputAttributeDescription2EXT::format

- VUID-vkCmdDrawIndexedIndirect-format-08936
  If there is a shader object bound to the VK_SHADER_STAGE_VERTEX_BIT stage or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VERTEX_INPUT_EXT dynamic state enabled and VkVertexInputAttributeDescription2EXT::format has a 64-bit component, then the scalar width associated with all Input variables of the corresponding Location in the Vertex Execution Model OpEntryPoint **must** be 64-bit

- VUID-vkCmdDrawIndexedIndirect-format-08937
  If there is a shader object bound to the VK_SHADER_STAGE_VERTEX_BIT stage or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VERTEX_INPUT_EXT dynamic state enabled and the scalar width associated with a Location decorated Input variable in the Vertex Execution Model OpEntryPoint is 64-bit, then the corresponding VkVertexInputAttributeDescription2EXT::format **must** have a 64-bit component

- VUID-vkCmdDrawIndexedIndirect-None-09203
  If there is a shader object bound to the VK_SHADER_STAGE_VERTEX_BIT stage or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VERTEX_INPUT_EXT dynamic state enabled and VkVertexInputAttributeDescription2EXT::format has a 64-bit component, then all Input variables at the corresponding Location in the Vertex Execution Model OpEntryPoint **must** not use components that are not present in the format

- VUID-vkCmdDrawIndexedIndirect-None-04875
  If there is a shader object bound to the VK_SHADER_STAGE_VERTEX_BIT stage and the most recent call to vkCmdSetPrimitiveTopologyEXT in the current command buffer set primitiveTopology to VK_PRIMITIVE_TOPOLOGY_PATCH_LIST, or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_PATCH_CONTROL_POINTS_EXT dynamic state enabled then vkCmdSetPatchControlPointsEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirect-None-04879
  If there is a shader object bound to the VK_SHADER_STAGE_VERTEX_BIT stage or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_PRIMITIVE_RESTART_ENABLE dynamic state enabled then vkCmdSetPrimitiveRestartEnableEXT **must** have been called

and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirect-None-09637
  If the primitiveTopologyListRestart feature is not enabled, the topology is VK_PRIMITIVE_TOPOLOGY_POINT_LIST, VK_PRIMITIVE_TOPOLOGY_LINE_LIST, VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST, VK_PRIMITIVE_TOPOLOGY_LINE_LIST_WITH_ADJACENCY, or VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST_WITH_ADJACENCY, there is a shader object bound to the VK_SHADER_STAGE_VERTEX_BIT stage or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_PRIMITIVE_RESTART_ENABLE dynamic state enabled then vkCmdSetPrimitiveRestartEnableEXT **must** be VK_FALSE
- VUID-vkCmdDrawIndexedIndirect-stage-06481
  The bound graphics pipeline **must** not have been created with the VkPipelineShaderStageCreateInfo::stage member of an element of VkGraphicsPipelineCreateInfo::pStages set to VK_SHADER_STAGE_TASK_BIT_EXT or VK_SHADER_STAGE_MESH_BIT_EXT
- VUID-vkCmdDrawIndexedIndirect-None-08885
  There **must** be no shader object bound to either of the VK_SHADER_STAGE_TASK_BIT_EXT or VK_SHADER_STAGE_MESH_BIT_EXT stages
- VUID-vkCmdDrawIndexedIndirect-buffer-02708
  If buffer is non-sparse then it **must** be bound completely and contiguously to a single VkDeviceMemory object
- VUID-vkCmdDrawIndexedIndirect-buffer-02709
  buffer **must** have been created with the VK_BUFFER_USAGE_INDIRECT_BUFFER_BIT bit set
- VUID-vkCmdDrawIndexedIndirect-offset-02710
  offset **must** be a multiple of 4
- VUID-vkCmdDrawIndexedIndirect-commandBuffer-02711
  commandBuffer **must** not be a protected command buffer
- VUID-vkCmdDrawIndexedIndirect-drawCount-02718
  If the multiDrawIndirect feature is not enabled, drawCount **must** be 0 or 1
- VUID-vkCmdDrawIndexedIndirect-drawCount-02719
  drawCount **must** be less than or equal to VkPhysicalDeviceLimits::maxDrawIndirectCount
- VUID-vkCmdDrawIndexedIndirect-None-07312
  If maintenance6 is not enabled, a valid index buffer **must** be bound
- VUID-vkCmdDrawIndexedIndirect-drawCount-00528
  If drawCount is greater than 1, stride **must** be a multiple of 4 and **must** be greater than or equal to sizeof(VkDrawIndexedIndirectCommand)
- VUID-vkCmdDrawIndexedIndirect-drawCount-00539
  If drawCount is equal to 1, (offset + sizeof(VkDrawIndexedIndirectCommand)) **must** be less than or equal to the size of buffer
- VUID-vkCmdDrawIndexedIndirect-drawCount-00540
  If drawCount is greater than 1, (stride × (drawCount - 1) + offset + sizeof

(`VkDrawIndexedIndirectCommand`)) **must** be less than or equal to the size of `buffer`

## Valid Usage (Implicit)

- VUID-vkCmdDrawIndexedIndirect-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdDrawIndexedIndirect-buffer-parameter
  `buffer` **must** be a valid VkBuffer handle
- VUID-vkCmdDrawIndexedIndirect-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdDrawIndexedIndirect-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- VUID-vkCmdDrawIndexedIndirect-renderpass
  This command **must** only be called inside of a render pass instance
- VUID-vkCmdDrawIndexedIndirect-videocoding
  This command **must** only be called outside of a video coding scope
- VUID-vkCmdDrawIndexedIndirect-commonparent
  Both of `buffer`, and `commandBuffer` **must** have been created, allocated, or retrieved from the same VkDevice

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Inside | Outside | Graphics | Action |

The `VkDrawIndexedIndirectCommand` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkDrawIndexedIndirectCommand {
    uint32_t    indexCount;
    uint32_t    instanceCount;
    uint32_t    firstIndex;
    int32_t     vertexOffset;
    uint32_t    firstInstance;
} VkDrawIndexedIndirectCommand;
```

- `indexCount` is the number of vertices to draw.
- `instanceCount` is the number of instances to draw.
- `firstIndex` is the base index within the index buffer.
- `vertexOffset` is the value added to the vertex index before indexing into the vertex buffer.
- `firstInstance` is the instance ID of the first instance to draw.

The members of `VkDrawIndexedIndirectCommand` have the same meaning as the similarly named parameters of vkCmdDrawIndexed.

### Valid Usage

- VUID-VkDrawIndexedIndirectCommand-pNext-09461
  If the bound graphics pipeline state was created with VkPipelineVertexInputDivisorStateCreateInfoKHR in the `pNext` chain of VkGraphicsPipelineCreateInfo::`pVertexInputState`, any member of VkPipelineVertexInputDivisorStateCreateInfoKHR::`pVertexBindingDivisors` has a value other than `1` in `divisor`, and VkPhysicalDeviceVertexAttributeDivisorPropertiesKHR::`supportsNonZeroFirstInstance` is `VK_FALSE`, then `firstInstance` **must** be `0`
- VUID-VkDrawIndexedIndirectCommand-None-09462
  If shader objects are used for drawing or the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VERTEX_INPUT_EXT` dynamic state enabled, any member of the `pVertexBindingDescriptions` parameter to the vkCmdSetVertexInputEXT call that sets this dynamic state has a value other than `1` in `divisor`, and VkPhysicalDeviceVertexAttributeDivisorPropertiesKHR::`supportsNonZeroFirstInstance` is `VK_FALSE`, then `firstInstance` **must** be `0`
- VUID-VkDrawIndexedIndirectCommand-robustBufferAccess2-08798
  If `robustBufferAccess2` is not enabled, (`indexSize` × (`firstIndex` + `indexCount`) + `offset`) **must** be less than or equal to the size of the bound index buffer, with `indexSize` being based on the type specified by `indexType`, where the index buffer, `indexType`, and `offset` are specified via `vkCmdBindIndexBuffer` or `vkCmdBindIndexBuffer2KHR`. If `vkCmdBindIndexBuffer2KHR` is used to bind the index buffer, the size of the bound index buffer is vkCmdBindIndexBuffer2KHR::`size`
- VUID-VkDrawIndexedIndirectCommand-None-00552
  For a given vertex buffer binding, any attribute data fetched **must** be entirely contained within the corresponding vertex buffer binding, as described in Vertex Input Description

- VUID-VkDrawIndexedIndirectCommand-firstInstance-00554
  If the `drawIndirectFirstInstance` feature is not enabled, `firstInstance` **must** be `0`

To record an indexed draw call with a draw call count sourced from a buffer, call:

```
// Provided by VK_KHR_draw_indirect_count
void vkCmdDrawIndexedIndirectCountKHR(
    VkCommandBuffer                             commandBuffer,
    VkBuffer                                    buffer,
    VkDeviceSize                                offset,
    VkBuffer                                    countBuffer,
    VkDeviceSize                                countBufferOffset,
    uint32_t                                    maxDrawCount,
    uint32_t                                    stride);
```

or the equivalent command

```
// Provided by VK_AMD_draw_indirect_count
void vkCmdDrawIndexedIndirectCountAMD(
    VkCommandBuffer                             commandBuffer,
    VkBuffer                                    buffer,
    VkDeviceSize                                offset,
    VkBuffer                                    countBuffer,
    VkDeviceSize                                countBufferOffset,
    uint32_t                                    maxDrawCount,
    uint32_t                                    stride);
```

- `commandBuffer` is the command buffer into which the command is recorded.
- `buffer` is the buffer containing draw parameters.
- `offset` is the byte offset into `buffer` where parameters begin.
- `countBuffer` is the buffer containing the draw count.
- `countBufferOffset` is the byte offset into `countBuffer` where the draw count begins.
- `maxDrawCount` specifies the maximum number of draws that will be executed. The actual number of executed draw calls is the minimum of the count specified in `countBuffer` and `maxDrawCount`.
- `stride` is the byte stride between successive sets of draw parameters.

`vkCmdDrawIndexedIndirectCountKHR` behaves similarly to vkCmdDrawIndexedIndirect except that the draw count is read by the device from a buffer during execution. The command will read an unsigned 32-bit integer from `countBuffer` located at `countBufferOffset` and use this as the draw count.

### Valid Usage

- VUID-vkCmdDrawIndexedIndirectCount-magFilter-04553

If a VkSampler created with magFilter or minFilter equal to VK_FILTER_LINEAR, reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE, and compareEnable equal to VK_FALSE is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT

- VUID-vkCmdDrawIndexedIndirectCount-magFilter-09598
  If a VkSampler created with magFilter or minFilter equal to VK_FILTER_LINEAR and reductionMode equal to either VK_SAMPLER_REDUCTION_MODE_MIN or VK_SAMPLER_REDUCTION_MODE_MAX is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT
- VUID-vkCmdDrawIndexedIndirectCount-mipmapMode-04770
  If a VkSampler created with mipmapMode equal to VK_SAMPLER_MIPMAP_MODE_LINEAR, reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE, and compareEnable equal to VK_FALSE is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT
- VUID-vkCmdDrawIndexedIndirectCount-mipmapMode-09599
  If a VkSampler created with mipmapMode equal to VK_SAMPLER_MIPMAP_MODE_LINEAR and reductionMode equal to either VK_SAMPLER_REDUCTION_MODE_MIN or VK_SAMPLER_REDUCTION_MODE_MAX is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT
- VUID-vkCmdDrawIndexedIndirectCount-unnormalizedCoordinates-09635
  If a VkSampler created with unnormalizedCoordinates equal to VK_TRUE is used to sample a VkImageView as a result of this command, then the image view's levelCount and layerCount **must** be 1
- VUID-vkCmdDrawIndexedIndirectCount-unnormalizedCoordinates-09636
  If a VkSampler created with unnormalizedCoordinates equal to VK_TRUE is used to sample a VkImageView as a result of this command, then the image view's viewType **must** be VK_IMAGE_VIEW_TYPE_1D or VK_IMAGE_VIEW_TYPE_2D
- VUID-vkCmdDrawIndexedIndirectCount-None-06479
  If a VkImageView is sampled with depth comparison, the image view's format features **must** contain VK_FORMAT_FEATURE_2_SAMPLED_IMAGE_DEPTH_COMPARISON_BIT
- VUID-vkCmdDrawIndexedIndirectCount-None-02691
  If a VkImageView is accessed using atomic operations as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_STORAGE_IMAGE_ATOMIC_BIT
- VUID-vkCmdDrawIndexedIndirectCount-None-07888
  If a VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER descriptor is accessed using atomic operations as a result of this command, then the storage texel buffer's format features **must** contain VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_ATOMIC_BIT
- VUID-vkCmdDrawIndexedIndirectCount-None-02692
  If a VkImageView is sampled with VK_FILTER_CUBIC_EXT as a result of this command, then

the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_CUBIC_BIT_EXT

- VUID-vkCmdDrawIndexedIndirectCount-None-02693
  If the VK_EXT_filter_cubic extension is not enabled and any VkImageView is sampled with VK_FILTER_CUBIC_EXT as a result of this command, it **must** not have a VkImageViewType of VK_IMAGE_VIEW_TYPE_3D, VK_IMAGE_VIEW_TYPE_CUBE, or VK_IMAGE_VIEW_TYPE_CUBE_ARRAY

- VUID-vkCmdDrawIndexedIndirectCount-filterCubic-02694
  Any VkImageView being sampled with VK_FILTER_CUBIC_EXT as a result of this command **must** have a VkImageViewType and format that supports cubic filtering, as specified by VkFilterCubicImageViewImageFormatPropertiesEXT::filterCubic returned by vkGetPhysicalDeviceImageFormatProperties2

- VUID-vkCmdDrawIndexedIndirectCount-filterCubicMinmax-02695
  Any VkImageView being sampled with VK_FILTER_CUBIC_EXT with a reduction mode of either VK_SAMPLER_REDUCTION_MODE_MIN or VK_SAMPLER_REDUCTION_MODE_MAX as a result of this command **must** have a VkImageViewType and format that supports cubic filtering together with minmax filtering, as specified by VkFilterCubicImageViewImageFormatPropertiesEXT::filterCubicMinmax returned by vkGetPhysicalDeviceImageFormatProperties2

- VUID-vkCmdDrawIndexedIndirectCount-cubicRangeClamp-09212
  If the cubicRangeClamp feature is not enabled, then any VkImageView being sampled with VK_FILTER_CUBIC_EXT as a result of this command **must** not have a VkSamplerReductionModeCreateInfo::reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE_RANGECLAMP_QCOM

- VUID-vkCmdDrawIndexedIndirectCount-reductionMode-09213
  Any VkImageView being sampled with a VkSamplerReductionModeCreateInfo::reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE_RANGECLAMP_QCOM as a result of this command **must** sample with VK_FILTER_CUBIC_EXT

- VUID-vkCmdDrawIndexedIndirectCount-selectableCubicWeights-09214
  If the selectableCubicWeights feature is not enabled, then any VkImageView being sampled with VK_FILTER_CUBIC_EXT as a result of this command **must** have VkSamplerCubicWeightsCreateInfoQCOM::cubicWeights equal to VK_CUBIC_FILTER_WEIGHTS_CATMULL_ROM_QCOM

- VUID-vkCmdDrawIndexedIndirectCount-flags-02696
  Any VkImage created with a VkImageCreateInfo::flags containing VK_IMAGE_CREATE_CORNER_SAMPLED_BIT_NV sampled as a result of this command **must** only be sampled using a VkSamplerAddressMode of VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_EDGE

- VUID-vkCmdDrawIndexedIndirectCount-OpTypeImage-07027
  For any VkImageView being written as a storage image where the image format field of the OpTypeImage is Unknown, the view's format features **must** contain VK_FORMAT_FEATURE_2_STORAGE_WRITE_WITHOUT_FORMAT_BIT

- VUID-vkCmdDrawIndexedIndirectCount-OpTypeImage-07028
  For any VkImageView being read as a storage image where the image format field of the OpTypeImage is Unknown, the view's format features **must** contain VK_FORMAT_FEATURE_2_STORAGE_READ_WITHOUT_FORMAT_BIT

- VUID-vkCmdDrawIndexedIndirectCount-OpTypeImage-07029
  For any VkBufferView being written as a storage texel buffer where the image format field of the `OpTypeImage` is `Unknown`, the view's buffer features **must** contain `VK_FORMAT_FEATURE_2_STORAGE_WRITE_WITHOUT_FORMAT_BIT`
- VUID-vkCmdDrawIndexedIndirectCount-OpTypeImage-07030
  Any VkBufferView being read as a storage texel buffer where the image format field of the `OpTypeImage` is `Unknown` then the view's buffer features **must** contain `VK_FORMAT_FEATURE_2_STORAGE_READ_WITHOUT_FORMAT_BIT`
- VUID-vkCmdDrawIndexedIndirectCount-None-08600
  For each set *n* that is statically used by a bound shader, a descriptor set **must** have been bound to *n* at the same pipeline bind point, with a VkPipelineLayout that is compatible for set *n*, with the VkPipelineLayout used to create the current VkPipeline or the VkDescriptorSetLayout array used to create the current VkShaderEXT , as described in Pipeline Layout Compatibility
- VUID-vkCmdDrawIndexedIndirectCount-None-08601
  For each push constant that is statically used by a bound shader, a push constant value **must** have been set for the same pipeline bind point, with a VkPipelineLayout that is compatible for push constants, with the VkPipelineLayout used to create the current VkPipeline or the VkDescriptorSetLayout array used to create the current VkShaderEXT , as described in Pipeline Layout Compatibility
- VUID-vkCmdDrawIndexedIndirectCount-None-10068
  For each array of resources that is used by a bound shader, the indices used to access members of the array **must** be less than the descriptor count for the identified binding in the descriptor sets used by this command
- VUID-vkCmdDrawIndexedIndirectCount-maintenance4-08602
  If the `maintenance4` feature is not enabled, then for each push constant that is statically used by a bound shader, a push constant value **must** have been set for the same pipeline bind point, with a VkPipelineLayout that is compatible for push constants, with the VkPipelineLayout used to create the current VkPipeline or the VkDescriptorSetLayout and VkPushConstantRange arrays used to create the current VkShaderEXT , as described in Pipeline Layout Compatibility
- VUID-vkCmdDrawIndexedIndirectCount-None-08114
  Descriptors in each bound descriptor set, specified via vkCmdBindDescriptorSets, **must** be valid as described by descriptor validity if they are statically used by the VkPipeline bound to the pipeline bind point used by this command and the bound VkPipeline was not created with `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`
- VUID-vkCmdDrawIndexedIndirectCount-None-08115
  If the descriptors used by the VkPipeline bound to the pipeline bind point were specified via vkCmdBindDescriptorSets, the bound VkPipeline **must** have been created without `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`
- VUID-vkCmdDrawIndexedIndirectCount-None-08116
  Descriptors in bound descriptor buffers, specified via vkCmdSetDescriptorBufferOffsetsEXT, **must** be valid if they are dynamically used by the VkPipeline bound to the pipeline bind point used by this command and the bound VkPipeline was created with `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`

- VUID-vkCmdDrawIndexedIndirectCount-None-08604
  Descriptors in bound descriptor buffers, specified via vkCmdSetDescriptorBufferOffsetsEXT, **must** be valid if they are dynamically used by any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command
- VUID-vkCmdDrawIndexedIndirectCount-None-08117
  If the descriptors used by the VkPipeline bound to the pipeline bind point were specified via vkCmdSetDescriptorBufferOffsetsEXT, the bound VkPipeline **must** have been created with `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`
- VUID-vkCmdDrawIndexedIndirectCount-None-08119
  If a descriptor is dynamically used with a VkPipeline created with `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`, the descriptor memory **must** be resident
- VUID-vkCmdDrawIndexedIndirectCount-None-08605
  If a descriptor is dynamically used with a VkShaderEXT created with a `VkDescriptorSetLayout` that was created with `VK_DESCRIPTOR_SET_LAYOUT_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`, the descriptor memory **must** be resident
- VUID-vkCmdDrawIndexedIndirectCount-None-08606
  If the `shaderObject` feature is not enabled, a valid pipeline **must** be bound to the pipeline bind point used by this command
- VUID-vkCmdDrawIndexedIndirectCount-None-08608
  If a pipeline is bound to the pipeline bind point used by this command, there **must** not have been any calls to dynamic state setting commands for any state specified statically in the VkPipeline object bound to the pipeline bind point used by this command, since that pipeline was bound
- VUID-vkCmdDrawIndexedIndirectCount-None-08609
  If the VkPipeline object bound to the pipeline bind point used by this command or any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a VkSampler object that uses unnormalized coordinates, that sampler **must** not be used to sample from any VkImage with a VkImageView of the type `VK_IMAGE_VIEW_TYPE_3D`, `VK_IMAGE_VIEW_TYPE_CUBE`, `VK_IMAGE_VIEW_TYPE_1D_ARRAY`, `VK_IMAGE_VIEW_TYPE_2D_ARRAY` or `VK_IMAGE_VIEW_TYPE_CUBE_ARRAY`, in any shader stage
- VUID-vkCmdDrawIndexedIndirectCount-None-08610
  If the VkPipeline object bound to the pipeline bind point used by this command or any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a VkSampler object that uses unnormalized coordinates, that sampler **must** not be used with any of the SPIR-V `OpImageSample*` or `OpImageSparseSample*` instructions with `ImplicitLod`, `Dref` or `Proj` in their name, in any shader stage
- VUID-vkCmdDrawIndexedIndirectCount-None-08611
  If the VkPipeline object bound to the pipeline bind point used by this command or any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a VkSampler object that uses unnormalized coordinates, that sampler **must** not be used with any of the SPIR-V `OpImageSample*` or `OpImageSparseSample*` instructions that includes a LOD bias or any offset values, in any shader stage

- VUID-vkCmdDrawIndexedIndirectCount-None-08607
  If the `shaderObject` is enabled, either a valid pipeline **must** be bound to the pipeline bind point used by this command, or a valid combination of valid and VK_NULL_HANDLE shader objects **must** be bound to every supported shader stage corresponding to the pipeline bind point used by this command
- VUID-vkCmdDrawIndexedIndirectCount-uniformBuffers-06935
  If any stage of the VkPipeline object bound to the pipeline bind point used by this command accesses a uniform buffer, and that stage was created without enabling either `VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_EXT` or `VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_2_EXT` for `uniformBuffers`, and the `robustBufferAccess` feature is not enabled, that stage **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point
- VUID-vkCmdDrawIndexedIndirectCount-None-08612
  If the `robustBufferAccess` feature is not enabled, and any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a uniform buffer, it **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point
- VUID-vkCmdDrawIndexedIndirectCount-storageBuffers-06936
  If any stage of the VkPipeline object bound to the pipeline bind point used by this command accesses a storage buffer, and that stage was created without enabling either `VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_EXT` or `VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_2_EXT` for `storageBuffers`, and the `robustBufferAccess` feature is not enabled, that stage **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point
- VUID-vkCmdDrawIndexedIndirectCount-None-08613
  If the `robustBufferAccess` feature is not enabled, and any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a storage buffer, it **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point
- VUID-vkCmdDrawIndexedIndirectCount-commandBuffer-02707
  If `commandBuffer` is an unprotected command buffer and `protectedNoFault` is not supported, any resource accessed by bound shaders **must** not be a protected resource
- VUID-vkCmdDrawIndexedIndirectCount-None-06550
  If a bound shader accesses a VkSampler or VkImageView object that enables sampler $Y'C_BC_R$ conversion, that object **must** only be used with `OpImageSample*` or `OpImageSparseSample*` instructions
- VUID-vkCmdDrawIndexedIndirectCount-ConstOffset-06551
  If a bound shader accesses a VkSampler or VkImageView object that enables sampler $Y'C_BC_R$ conversion, that object **must** not use the `ConstOffset` and `Offset` operands
- VUID-vkCmdDrawIndexedIndirectCount-viewType-07752
  If a VkImageView is accessed as a result of this command, then the image view's `viewType` **must** match the `Dim` operand of the `OpTypeImage` as described in Compatibility Between SPIR-V Image Dimensions and Vulkan ImageView Types

- VUID-vkCmdDrawIndexedIndirectCount-format-07753
  If a VkImageView is accessed as a result of this command, then the numeric type of the image view's `format` and the `Sampled Type` operand of the `OpTypeImage` **must** match
- VUID-vkCmdDrawIndexedIndirectCount-OpImageWrite-08795
  If a VkImageView created with a format other than `VK_FORMAT_A8_UNORM_KHR` is accessed using `OpImageWrite` as a result of this command, then the `Type` of the `Texel` operand of that instruction **must** have at least as many components as the image view's format
- VUID-vkCmdDrawIndexedIndirectCount-OpImageWrite-08796
  If a VkImageView created with the format `VK_FORMAT_A8_UNORM_KHR` is accessed using `OpImageWrite` as a result of this command, then the `Type` of the `Texel` operand of that instruction **must** have four components
- VUID-vkCmdDrawIndexedIndirectCount-OpImageWrite-04469
  If a VkBufferView is accessed using `OpImageWrite` as a result of this command, then the `Type` of the `Texel` operand of that instruction **must** have at least as many components as the buffer view's format
- VUID-vkCmdDrawIndexedIndirectCount-SampledType-04470
  If a VkImageView with a VkFormat that has a 64-bit component width is accessed as a result of this command, the `SampledType` of the `OpTypeImage` operand of that instruction **must** have a `Width` of 64
- VUID-vkCmdDrawIndexedIndirectCount-SampledType-04471
  If a VkImageView with a VkFormat that has a component width less than 64-bit is accessed as a result of this command, the `SampledType` of the `OpTypeImage` operand of that instruction **must** have a `Width` of 32
- VUID-vkCmdDrawIndexedIndirectCount-SampledType-04472
  If a VkBufferView with a VkFormat that has a 64-bit component width is accessed as a result of this command, the `SampledType` of the `OpTypeImage` operand of that instruction **must** have a `Width` of 64
- VUID-vkCmdDrawIndexedIndirectCount-SampledType-04473
  If a VkBufferView with a VkFormat that has a component width less than 64-bit is accessed as a result of this command, the `SampledType` of the `OpTypeImage` operand of that instruction **must** have a `Width` of 32
- VUID-vkCmdDrawIndexedIndirectCount-sparseImageInt64Atomics-04474
  If the `sparseImageInt64Atomics` feature is not enabled, VkImage objects created with the `VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT` flag **must** not be accessed by atomic instructions through an `OpTypeImage` with a `SampledType` with a `Width` of 64 by this command
- VUID-vkCmdDrawIndexedIndirectCount-sparseImageInt64Atomics-04475
  If the `sparseImageInt64Atomics` feature is not enabled, VkBuffer objects created with the `VK_BUFFER_CREATE_SPARSE_RESIDENCY_BIT` flag **must** not be accessed by atomic instructions through an `OpTypeImage` with a `SampledType` with a `Width` of 64 by this command
- VUID-vkCmdDrawIndexedIndirectCount-OpImageWeightedSampleQCOM-06971
  If `OpImageWeightedSampleQCOM` is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_2_WEIGHT_SAMPLED_IMAGE_BIT_QCOM`

- VUID-vkCmdDrawIndexedIndirectCount-OpImageWeightedSampleQCOM-06972
  If `OpImageWeightedSampleQCOM` uses a VkImageView as a sample weight image as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_2_WEIGHT_IMAGE_BIT_QCOM`
- VUID-vkCmdDrawIndexedIndirectCount-OpImageBoxFilterQCOM-06973
  If `OpImageBoxFilterQCOM` is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_2_BOX_FILTER_SAMPLED_BIT_QCOM`
- VUID-vkCmdDrawIndexedIndirectCount-OpImageBlockMatchSSDQCOM-06974
  If `OpImageBlockMatchSSDQCOM` is used to read from an VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM`
- VUID-vkCmdDrawIndexedIndirectCount-OpImageBlockMatchSADQCOM-06975
  If `OpImageBlockMatchSADQCOM` is used to read from an VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM`
- VUID-vkCmdDrawIndexedIndirectCount-OpImageBlockMatchSADQCOM-06976
  If `OpImageBlockMatchSADQCOM` or OpImageBlockMatchSSDQCOM is used to read from a reference image as result of this command, then the specified reference coordinates **must** not fail integer texel coordinate validation
- VUID-vkCmdDrawIndexedIndirectCount-OpImageWeightedSampleQCOM-06977
  If `OpImageWeightedSampleQCOM`, `OpImageBoxFilterQCOM`, `OpImageBlockMatchWindowSSDQCOM`, `OpImageBlockMatchWindowSADQCOM`, `OpImageBlockMatchGatherSSDQCOM`, `OpImageBlockMatchGatherSADQCOM`, `OpImageBlockMatchSSDQCOM`, or `OpImageBlockMatchSADQCOM` uses a VkSampler as a result of this command, then the sampler **must** have been created with `VK_SAMPLER_CREATE_IMAGE_PROCESSING_BIT_QCOM`
- VUID-vkCmdDrawIndexedIndirectCount-OpImageWeightedSampleQCOM-06978
  If any command other than `OpImageWeightedSampleQCOM`, `OpImageBoxFilterQCOM`, `OpImageBlockMatchWindowSSDQCOM`, `OpImageBlockMatchWindowSADQCOM`, `OpImageBlockMatchGatherSSDQCOM`, `OpImageBlockMatchGatherSADQCOM`, `OpImageBlockMatchSSDQCOM`, or `OpImageBlockMatchSADQCOM` uses a VkSampler as a result of this command, then the sampler **must** not have been created with `VK_SAMPLER_CREATE_IMAGE_PROCESSING_BIT_QCOM`
- VUID-vkCmdDrawIndexedIndirectCount-OpImageBlockMatchWindow-09215
  If a `OpImageBlockMatchWindow*QCOM` or `OpImageBlockMatchGather*QCOM` instruction is used to read from an VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM`
- VUID-vkCmdDrawIndexedIndirectCount-OpImageBlockMatchWindow-09216
  If a `OpImageBlockMatchWindow*QCOM` or `OpImageBlockMatchGather*QCOM` instruction is used to read from an VkImageView as a result of this command, then the image view's format **must** be a single-component format
- VUID-vkCmdDrawIndexedIndirectCount-OpImageBlockMatchWindow-09217
  If a `OpImageBlockMatchWindow*QCOM` or `OpImageBlockMatchGather*QCOM` read from a reference image as result of this command, then the specified reference coordinates **must** not fail

integer texel coordinate validation

- VUID-vkCmdDrawIndexedIndirectCount-None-07288
  Any shader invocation executed by this command **must** terminate
- VUID-vkCmdDrawIndexedIndirectCount-None-09600
  If a descriptor with type equal to any of `VK_DESCRIPTOR_TYPE_SAMPLE_WEIGHT_IMAGE_QCOM`, `VK_DESCRIPTOR_TYPE_BLOCK_MATCH_IMAGE_QCOM`, `VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE`, `VK_DESCRIPTOR_TYPE_STORAGE_IMAGE`, or `VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT` is accessed as a result of this command, the image subresource identified by that descriptor **must** be in the image layout identified when the descriptor was written
- VUID-vkCmdDrawIndexedIndirectCount-renderPass-02684
  The current render pass **must** be compatible with the `renderPass` member of the `VkGraphicsPipelineCreateInfo` structure specified when creating the `VkPipeline` bound to `VK_PIPELINE_BIND_POINT_GRAPHICS`
- VUID-vkCmdDrawIndexedIndirectCount-subpass-02685
  The subpass index of the current render pass **must** be equal to the `subpass` member of the `VkGraphicsPipelineCreateInfo` structure specified when creating the `VkPipeline` bound to `VK_PIPELINE_BIND_POINT_GRAPHICS`
- VUID-vkCmdDrawIndexedIndirectCount-None-07748
  If any shader statically accesses an input attachment, a valid descriptor **must** be bound to the pipeline via a descriptor set
- VUID-vkCmdDrawIndexedIndirectCount-OpTypeImage-07468
  If any shader executed by this pipeline accesses an `OpTypeImage` variable with a `Dim` operand of `SubpassData`, it **must** be decorated with an `InputAttachmentIndex` that corresponds to a valid input attachment in the current subpass
- VUID-vkCmdDrawIndexedIndirectCount-None-07469
  Input attachment views accessed in a subpass **must** be created with the same VkFormat as the corresponding subpass definition, and be created with a VkImageView that is compatible with the attachment referenced by the subpass' `pInputAttachments`[`InputAttachmentIndex`] in the currently bound VkFramebuffer as specified by Fragment Input Attachment Compatibility
- VUID-vkCmdDrawIndexedIndirectCount-pDepthInputAttachmentIndex-09595
  Input attachment views accessed in a dynamic render pass with a `InputAttachmentIndex` referenced by VkRenderingInputAttachmentIndexInfoKHR, or no `InputAttachmentIndex` if VkRenderingInputAttachmentIndexInfoKHR:`pDepthInputAttachmentIndex` or VkRenderingInputAttachmentIndexInfoKHR:`pStencilInputAttachmentIndex` are `NULL`, **must** be created with a VkImageView that is compatible with the corresponding color, depth, or stencil attachment in VkRenderingInfo
- VUID-vkCmdDrawIndexedIndirectCount-pDepthInputAttachmentIndex-09596
  Input attachment views accessed in a dynamic render pass via a shader object **must** have an `InputAttachmentIndex` if both VkRenderingInputAttachmentIndexInfoKHR:`pDepthInputAttachmentIndex` and VkRenderingInputAttachmentIndexInfoKHR:`pStencilInputAttachmentIndex` are non-`NULL`
- VUID-vkCmdDrawIndexedIndirectCount-InputAttachmentIndex-09597
  If an input attachment view accessed in a dynamic render pass via a shader object has an

InputAttachmentIndex, the InputAttachmentIndex **must** match an index in VkRenderingInputAttachmentIndexInfoKHR

- VUID-vkCmdDrawIndexedIndirectCount-None-06537
  Memory backing image subresources used as attachments in the current render pass **must** not be written in any way other than as an attachment by this command
- VUID-vkCmdDrawIndexedIndirectCount-None-09000
  If a color attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass, it is not in the VK_IMAGE_LAYOUT_ATTACHMENT_FEEDBACK_LOOP_OPTIMAL_EXT image layout, and either:
  - the VK_PIPELINE_CREATE_COLOR_ATTACHMENT_FEEDBACK_LOOP_BIT_EXT is set on the currently bound pipeline or
  - the last call to vkCmdSetAttachmentFeedbackLoopEnableEXT included VK_IMAGE_ASPECT_COLOR_BIT and
    - there is no currently bound graphics pipeline or
    - the currently bound graphics pipeline was created with VK_DYNAMIC_STATE_ATTACHMENT_FEEDBACK_LOOP_ENABLE_EXT

  it **must** not be accessed in any way other than as an attachment by this command
- VUID-vkCmdDrawIndexedIndirectCount-None-09001
  If a depth attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass, it is not in the VK_IMAGE_LAYOUT_ATTACHMENT_FEEDBACK_LOOP_OPTIMAL_EXT image layout, and either:
  - the VK_PIPELINE_CREATE_DEPTH_STENCIL_ATTACHMENT_FEEDBACK_LOOP_BIT_EXT is set on the currently bound pipeline or
  - the last call to vkCmdSetAttachmentFeedbackLoopEnableEXT included VK_IMAGE_ASPECT_DEPTH_BIT and
    - there is no currently bound graphics pipeline or
    - the currently bound graphics pipeline was created with VK_DYNAMIC_STATE_ATTACHMENT_FEEDBACK_LOOP_ENABLE_EXT

  it **must** not be accessed in any way other than as an attachment by this command
- VUID-vkCmdDrawIndexedIndirectCount-None-09002
  If a stencil attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass, it is not in the VK_IMAGE_LAYOUT_ATTACHMENT_FEEDBACK_LOOP_OPTIMAL_EXT image layout, and either:
  - the VK_PIPELINE_CREATE_DEPTH_STENCIL_ATTACHMENT_FEEDBACK_LOOP_BIT_EXT is set on the currently bound pipeline or
  - the last call to vkCmdSetAttachmentFeedbackLoopEnableEXT included VK_IMAGE_ASPECT_STENCIL_BIT and
    - there is no currently bound graphics pipeline or
    - the currently bound graphics pipeline was created with

VK_DYNAMIC_STATE_ATTACHMENT_FEEDBACK_LOOP_ENABLE_EXT

it **must** not be accessed in any way other than as an attachment by this command

- VUID-vkCmdDrawIndexedIndirectCount-None-09003
  If an attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass, it **must** not be accessed in any way other than as an attachment, storage image, or sampled image by this command
- VUID-vkCmdDrawIndexedIndirectCount-None-06539
  If any previously recorded command in the current subpass accessed an image subresource used as an attachment in this subpass in any way other than as an attachment, this command **must** not write to that image subresource as an attachment
- VUID-vkCmdDrawIndexedIndirectCount-None-06886
  If the current render pass instance uses a depth/stencil attachment with a read-only layout for the depth aspect, depth writes **must** be disabled
- VUID-vkCmdDrawIndexedIndirectCount-None-06887
  If the current render pass instance uses a depth/stencil attachment with a read-only layout for the stencil aspect, both front and back `writeMask` are not zero, and stencil test is enabled, all stencil ops **must** be `VK_STENCIL_OP_KEEP`
- VUID-vkCmdDrawIndexedIndirectCount-None-07831
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VIEWPORT` dynamic state enabled then vkCmdSetViewport **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirectCount-None-07832
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_SCISSOR` dynamic state enabled then vkCmdSetScissor **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirectCount-None-07833
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_LINE_WIDTH` dynamic state enabled then vkCmdSetLineWidth **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirectCount-None-08617
  If a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE`, and the most recent call to vkCmdSetPolygonModeEXT in the current command buffer set `polygonMode` to `VK_POLYGON_MODE_LINE`, vkCmdSetLineWidth **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirectCount-None-08618
  If a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE`, and the most recent call to vkCmdSetPrimitiveTopologyEXT in the current command buffer set `primitiveTopology` to any line topology, vkCmdSetLineWidth **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirectCount-None-08619
  If a shader object that outputs line primitives is bound to the VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT or VK_SHADER_STAGE_GEOMETRY_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, vkCmdSetLineWidth **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirectCount-None-07834
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_BIAS dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of depthBiasEnable is VK_TRUE, then vkCmdSetDepthBounds or vkCmdSetDepthBias2EXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirectCount-None-07835
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_BLEND_CONSTANTS dynamic state enabled then vkCmdSetBlendConstants **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirectCount-None-08621
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetColorBlendEnableEXT in the current command buffer set any element of pColorBlendEnables to VK_TRUE, and the most recent call to vkCmdSetColorBlendEquationEXT in the current command buffer set the same element of pColorBlendEquations to a VkColorBlendEquationEXT structure with any VkBlendFactor member with a value of VK_BLEND_FACTOR_CONSTANT_COLOR, VK_BLEND_FACTOR_ONE_MINUS_CONSTANT_COLOR, VK_BLEND_FACTOR_CONSTANT_ALPHA, or VK_BLEND_FACTOR_ONE_MINUS_CONSTANT_ALPHA, vkCmdSetBlendConstants **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirectCount-None-07836
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_BOUNDS dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of depthBoundsTestEnable is VK_TRUE, then vkCmdSetDepthBounds **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirectCount-None-07837
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_STENCIL_COMPARE_MASK dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of stencilTestEnable is VK_TRUE, then vkCmdSetStencilCompareMask **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirectCount-None-07838
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_STENCIL_WRITE_MASK dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of stencilTestEnable is VK_TRUE, then vkCmdSetStencilWriteMask **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirectCount-None-07839
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_STENCIL_REFERENCE dynamic state enabled, the current value of and rasterizerDiscardEnable is VK_FALSE, the current value of stencilTestEnable is VK_TRUE, then vkCmdSetStencilReference **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirectCount-maxMultiviewInstanceIndex-02688
  If the draw is recorded in a render pass instance with multiview enabled, the maximum instance index **must** be less than or equal to VkPhysicalDeviceMultiviewProperties::maxMultiviewInstanceIndex
- VUID-vkCmdDrawIndexedIndirectCount-sampleLocationsEnable-02689
  If the bound graphics pipeline was created with VkPipelineSampleLocationsStateCreateInfoEXT::sampleLocationsEnable set to VK_TRUE and the current subpass has a depth/stencil attachment, then that attachment **must** have been created with the VK_IMAGE_CREATE_SAMPLE_LOCATIONS_COMPATIBLE_DEPTH_BIT_EXT bit set
- VUID-vkCmdDrawIndexedIndirectCount-None-06666
  If the VK_EXT_sample_locations extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of sampleLocationsEnable is VK_TRUE, then vkCmdSetSampleLocationsEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirectCount-None-07840
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_CULL_MODE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetCullModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirectCount-None-07841
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_FRONT_FACE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetFrontFaceEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirectCount-None-07843
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_TEST_ENABLE dynamic state enabled, and the

current value of rasterizerDiscardEnable is VK_FALSE, vkCmdSetDepthTestEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirectCount-None-07844
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_WRITE_ENABLE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetDepthWriteEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirectCount-None-07845
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_COMPARE_OP dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of depthTestEnable is VK_TRUE, then vkCmdSetDepthCompareOpEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirectCount-None-07846
  If the depthBounds feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_BOUNDS_TEST_ENABLE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetDepthBoundsTestEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirectCount-None-07847
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_STENCIL_TEST_ENABLE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetStencilTestEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirectCount-None-07848
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_STENCIL_OP dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, the current value of stencilTestEnable is VK_TRUE, then vkCmdSetStencilOpEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirectCount-viewportCount-03417
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, but not the VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT dynamic state enabled, then vkCmdSetViewportWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the viewportCount parameter of vkCmdSetViewportWithCountEXT **must** match the VkPipelineViewportStateCreateInfo::scissorCount of the pipeline
- VUID-vkCmdDrawIndexedIndirectCount-scissorCount-03418
  If the bound graphics pipeline state was created with the

VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT dynamic state enabled, but not the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, then vkCmdSetScissorWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the scissorCount parameter of vkCmdSetScissorWithCountEXT **must** match the VkPipelineViewportStateCreateInfo::viewportCount of the pipeline

- VUID-vkCmdDrawIndexedIndirectCount-viewportCount-03419
  If the bound graphics pipeline state was created with both the VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT and VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic states enabled then both vkCmdSetViewportWithCountEXT and vkCmdSetScissorWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the viewportCount parameter of vkCmdSetViewportWithCountEXT **must** match the scissorCount parameter of vkCmdSetScissorWithCountEXT

- VUID-vkCmdDrawIndexedIndirectCount-None-08635
  If a shader object is bound to any graphics stage, then both vkCmdSetViewportWithCountEXT and vkCmdSetScissorWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the viewportCount parameter of vkCmdSetViewportWithCountEXT **must** match the scissorCount parameter of vkCmdSetScissorWithCountEXT

- VUID-vkCmdDrawIndexedIndirectCount-viewportCount-04137
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, but not the VK_DYNAMIC_STATE_VIEWPORT_W_SCALING_NV dynamic state enabled, then the bound graphics pipeline **must** have been created with VkPipelineViewportWScalingStateCreateInfoNV::viewportCount greater or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT

- VUID-vkCmdDrawIndexedIndirectCount-viewportCount-04138
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT and VK_DYNAMIC_STATE_VIEWPORT_W_SCALING_NV dynamic states enabled then the viewportCount parameter in the last call to vkCmdSetViewportWScalingNV **must** be greater than or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT

- VUID-vkCmdDrawIndexedIndirectCount-None-09232
  If the VK_NV_clip_space_w_scaling extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetViewportWScalingEnableNV in the current command buffer set viewportWScalingEnable to VK_TRUE, then vkCmdSetViewportWScalingNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirectCount-None-08636
  If the VK_NV_clip_space_w_scaling extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetViewportWScalingEnableNV in the current command buffer set viewportWScalingEnable to VK_TRUE, then the viewportCount parameter in the last call to vkCmdSetViewportWScalingNV **must** be greater than or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT

- VUID-vkCmdDrawIndexedIndirectCount-viewportCount-04139
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT` dynamic state enabled, but not the `VK_DYNAMIC_STATE_VIEWPORT_SHADING_RATE_PALETTE_NV` dynamic state enabled, then the bound graphics pipeline **must** have been created with VkPipelineViewportShadingRateImageStateCreateInfoNV::`viewportCount` greater or equal to the `viewportCount` parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawIndexedIndirectCount-viewportCount-04140
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT` and `VK_DYNAMIC_STATE_VIEWPORT_SHADING_RATE_PALETTE_NV` dynamic states enabled then the `viewportCount` parameter in the last call to vkCmdSetViewportShadingRatePaletteNV **must** be greater than or equal to the `viewportCount` parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawIndexedIndirectCount-shadingRateImage-09233
  If the `shadingRateImage` feature is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE`, then vkCmdSetCoarseSampleOrderNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirectCount-shadingRateImage-09234
  If the `shadingRateImage` feature is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE`, and the most recent call to vkCmdSetShadingRateImageEnableNV in the current command buffer set `shadingRateImageEnable` to `VK_TRUE`, then vkCmdSetViewportShadingRatePaletteNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirectCount-None-08637
  If the `shadingRateImage` feature is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE`, and the most recent call to vkCmdSetShadingRateImageEnableNV in the current command buffer set `shadingRateImageEnable` to `VK_TRUE`, then the `viewportCount` parameter in the last call to vkCmdSetViewportShadingRatePaletteNV **must** be greater than or equal to the `viewportCount` parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawIndexedIndirectCount-VkPipelineVieportCreateInfo-04141
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT` dynamic state enabled and a VkPipelineViewportSwizzleStateCreateInfoNV structure chained from VkPipelineViewportStateCreateInfo, then the bound graphics pipeline **must** have been created with VkPipelineViewportSwizzleStateCreateInfoNV::`viewportCount` greater or equal to the `viewportCount` parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawIndexedIndirectCount-VkPipelineVieportCreateInfo-04142
  If the bound graphics pipeline state was created with the

VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled and a VkPipelineViewportExclusiveScissorStateCreateInfoNV structure chained from VkPipelineViewportStateCreateInfo, then the bound graphics pipeline **must** have been created with VkPipelineViewportExclusiveScissorStateCreateInfoNV::exclusiveScissorCount greater or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT

- VUID-vkCmdDrawIndexedIndirectCount-None-07878
  If the exclusiveScissor feature is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_EXCLUSIVE_SCISSOR_ENABLE_NV dynamic state enabled, then vkCmdSetExclusiveScissorEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirectCount-None-07879
  If the exclusiveScissor feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_EXCLUSIVE_SCISSOR_NV dynamic state enabled, and the most recent call to vkCmdSetExclusiveScissorEnableNV in the current command buffer set any element of pExclusiveScissorEnables to VK_TRUE, then vkCmdSetExclusiveScissorNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirectCount-None-04876
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_RASTERIZER_DISCARD_ENABLE dynamic state enabled, then vkCmdSetRasterizerDiscardEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirectCount-None-04877
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_BIAS_ENABLE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetDepthBiasEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirectCount-logicOp-04878
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_LOGIC_OP_EXT dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of logicOpEnable is VK_TRUE, then vkCmdSetLogicOpEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirectCount-primitiveFragmentShadingRateWithMultipleViewports-04552
  If the primitiveFragmentShadingRateWithMultipleViewports limit is not supported, the bound graphics pipeline was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, and any of the shader stages of the bound graphics pipeline write to the PrimitiveShadingRateKHR built-in, then vkCmdSetViewportWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the viewportCount parameter of vkCmdSetViewportWithCountEXT **must** be 1

- VUID-vkCmdDrawIndexedIndirectCount-primitiveFragmentShadingRateWithMultipleViewports-08642
  If the `primitiveFragmentShadingRateWithMultipleViewports` limit is not supported, and any shader object bound to a graphics stage writes to the `PrimitiveShadingRateKHR` built-in, then vkCmdSetViewportWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the `viewportCount` parameter of `vkCmdSetViewportWithCountEXT` **must** be `1`
- VUID-vkCmdDrawIndexedIndirectCount-blendEnable-04727
  If rasterization is not disabled in the bound graphics pipeline, then for each color attachment in the subpass, if the corresponding image view's format features do not contain `VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT`, then the `blendEnable` member of the corresponding element of the `pAttachments` member of `pColorBlendState` **must** be `VK_FALSE`
- VUID-vkCmdDrawIndexedIndirectCount-None-08643
  If a shader object is bound to the `VK_SHADER_STAGE_FRAGMENT_BIT` stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE`, then for each color attachment in the render pass, if the corresponding image view's format features do not contain `VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT`, then the corresponding member of `pColorBlendEnables` in the most recent call to `vkCmdSetColorBlendEnableEXT` in the current command buffer that affected that attachment index **must** have been `VK_FALSE`
- VUID-vkCmdDrawIndexedIndirectCount-multisampledRenderToSingleSampled-07284
  If rasterization is not disabled in the bound graphics pipeline, and none of the following is enabled:
  - the `VK_AMD_mixed_attachment_samples` extension
  - the `VK_NV_framebuffer_mixed_samples` extension
  - the `multisampledRenderToSingleSampled` feature

  then `rasterizationSamples` for the currently bound graphics pipeline **must** be the same as the current subpass color and/or depth/stencil attachments
- VUID-vkCmdDrawIndexedIndirectCount-None-08644
  If a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE`, and none of the following is enabled:
  - the `VK_AMD_mixed_attachment_samples` extension
  - the `VK_NV_framebuffer_mixed_samples` extension
  - the `multisampledRenderToSingleSampled` feature

  then the most recent call to vkCmdSetRasterizationSamplesEXT in the current command buffer **must** have set `rasterizationSamples` to be the same as the number of samples for the current render pass color and/or depth/stencil attachments
- VUID-vkCmdDrawIndexedIndirectCount-None-08876
  If a shader object is bound to any graphics stage, the current render pass instance **must**

have been begun with vkCmdBeginRenderingKHR

- VUID-vkCmdDrawIndexedIndirectCount-imageView-06172
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the imageView member of pDepthAttachment is not VK_NULL_HANDLE, and the layout member of pDepthAttachment is VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL, this command **must** not write any values to the depth attachment
- VUID-vkCmdDrawIndexedIndirectCount-imageView-06173
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the imageView member of pStencilAttachment is not VK_NULL_HANDLE, and the layout member of pStencilAttachment is VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL, this command **must** not write any values to the stencil attachment
- VUID-vkCmdDrawIndexedIndirectCount-imageView-06174
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the imageView member of pDepthAttachment is not VK_NULL_HANDLE, and the layout member of pDepthAttachment is VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_STENCIL_ATTACHMENT_OPTIMAL, this command **must** not write any values to the depth attachment
- VUID-vkCmdDrawIndexedIndirectCount-imageView-06175
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the imageView member of pStencilAttachment is not VK_NULL_HANDLE, and the layout member of pStencilAttachment is VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_STENCIL_READ_ONLY_OPTIMAL, this command **must** not write any values to the stencil attachment
- VUID-vkCmdDrawIndexedIndirectCount-imageView-06176
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the imageView member of pDepthAttachment is not VK_NULL_HANDLE, and the layout member of pDepthAttachment is VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL, this command **must** not write any values to the depth attachment
- VUID-vkCmdDrawIndexedIndirectCount-imageView-06177
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the imageView member of pStencilAttachment is not VK_NULL_HANDLE, and the layout member of pStencilAttachment is VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL, this command **must** not write any values to the stencil attachment
- VUID-vkCmdDrawIndexedIndirectCount-viewMask-06178
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the currently bound graphics pipeline **must** have been created with a VkPipelineRenderingCreateInfo::viewMask equal to VkRenderingInfo::viewMask
- VUID-vkCmdDrawIndexedIndirectCount-colorAttachmentCount-06179
  If the dynamicRenderingUnusedAttachments feature is not enabled and the current render pass instance was begun with vkCmdBeginRenderingKHR, the currently bound graphics pipeline **must** have been created with a VkPipelineRenderingCreateInfo::colorAttachmentCount equal to VkRenderingInfo::colorAttachmentCount
- VUID-vkCmdDrawIndexedIndirectCount-dynamicRenderingUnusedAttachments-08910
  If the dynamicRenderingUnusedAttachments feature is not enabled, and the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingInfo

::colorAttachmentCount greater than 0, then each element of the VkRenderingInfo::pColorAttachments array with an imageView not equal to VK_NULL_HANDLE **must** have been created with a VkFormat equal to the corresponding element of VkPipelineRenderingCreateInfo::pColorAttachmentFormats used to create the currently bound graphics pipeline

- VUID-vkCmdDrawIndexedIndirectCount-dynamicRenderingUnusedAttachments-08912
  If the dynamicRenderingUnusedAttachments feature is not enabled, and the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingInfo::colorAttachmentCount greater than 0, then each element of the VkRenderingInfo::pColorAttachments array with an imageView equal to VK_NULL_HANDLE **must** have the corresponding element of VkPipelineRenderingCreateInfo::pColorAttachmentFormats used to create the currently bound pipeline equal to VK_FORMAT_UNDEFINED

- VUID-vkCmdDrawIndexedIndirectCount-dynamicRenderingUnusedAttachments-08911
  If the dynamicRenderingUnusedAttachments feature is enabled, and the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingInfo::colorAttachmentCount greater than 0, then each element of the VkRenderingInfo::pColorAttachments array with an imageView not equal to VK_NULL_HANDLE **must** have been created with a VkFormat equal to the corresponding element of VkPipelineRenderingCreateInfo::pColorAttachmentFormats used to create the currently bound graphics pipeline, or the corresponding element of VkPipelineRenderingCreateInfo::pColorAttachmentFormats, if it exists, **must** be VK_FORMAT_UNDEFINED

- VUID-vkCmdDrawIndexedIndirectCount-colorAttachmentCount-09362
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, with a VkRenderingInfo::colorAttachmentCount equal to 1, there is no shader object bound to any graphics stage, and a color attachment with a resolve mode of VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID, each element of the VkRenderingInfo::pColorAttachments array with a resolveImageView not equal to VK_NULL_HANDLE **must** have been created with an image created with a VkExternalFormatANDROID::externalFormat value equal to the VkExternalFormatANDROID::externalFormat value used to create the currently bound graphics pipeline

- VUID-vkCmdDrawIndexedIndirectCount-None-09363
  If there is no shader object bound to any graphics stage, the current render pass instance was begun with vkCmdBeginRenderingKHR and a VkRenderingInfo::colorAttachmentCount equal to 1, and a color attachment with a resolve mode of VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID, each element of the VkRenderingInfo::pColorAttachments array with a imageView not equal to VK_NULL_HANDLE **must** have been created with an image created with a VkExternalFormatANDROID::externalFormat value equal to the VkExternalFormatANDROID::externalFormat value used to create the currently bound graphics pipeline

- VUID-vkCmdDrawIndexedIndirectCount-None-09364
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is no shader object bound to any graphics stage, and the currently bound graphics pipeline was created with a non-zero VkExternalFormatANDROID::externalFormat value and with

the VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT dynamic state enabled, then vkCmdSetColorBlendEnableEXT **must** have set the blend enable to VK_FALSE prior to this drawing command

- VUID-vkCmdDrawIndexedIndirectCount-None-09365
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is no shader object bound to any graphics stage, and the currently bound graphics pipeline was created with a non-zero VkExternalFormatANDROID::externalFormat value and with the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT dynamic state enabled, then vkCmdSetRasterizationSamplesEXT **must** have set rasterizationSamples to VK_SAMPLE_COUNT_1_BIT prior to this drawing command

- VUID-vkCmdDrawIndexedIndirectCount-None-09366
  If there is a shader object bound to any graphics stage, and the current render pass includes a color attachment that uses the VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID resolve mode, then vkCmdSetColorBlendEnableEXT **must** have set blend enable to VK_FALSE prior to this drawing command

- VUID-vkCmdDrawIndexedIndirectCount-rasterizationSamples-09367
  If there is a shader object bound to any graphics stage, and the current render pass includes a color attachment that uses the VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID resolve mode, then vkCmdSetRasterizationSamplesEXT **must** have set rasterizationSamples to VK_SAMPLE_COUNT_1_BIT prior to this drawing command

- VUID-vkCmdDrawIndexedIndirectCount-None-09368
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is no shader object bound to any graphics stage, and the currently bound graphics pipeline was created with a non-zero VkExternalFormatANDROID::externalFormat value and with the VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR dynamic state enabled, then vkCmdSetFragmentShadingRateKHR **must** have set pFragmentSize->width to 1 prior to this drawing command

- VUID-vkCmdDrawIndexedIndirectCount-None-09369
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is no shader object bound to any graphics stage, and the currently bound graphics pipeline was created with a non-zero VkExternalFormatANDROID::externalFormat value and with the VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR dynamic state enabled, then vkCmdSetFragmentShadingRateKHR **must** have set pFragmentSize->height to 1 prior to this drawing command

- VUID-vkCmdDrawIndexedIndirectCount-pFragmentSize-09370
  If there is a shader object bound to any graphics stage, and the current render pass includes a color attachment that uses the VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID resolve mode, then vkCmdSetFragmentShadingRateKHR **must** have set pFragmentSize->width to 1 prior to this drawing command

- VUID-vkCmdDrawIndexedIndirectCount-pFragmentSize-09371
  If there is a shader object bound to any graphics stage, and the current render pass includes a color attachment that uses the

VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID resolve mode, then vkCmdSetFragmentShadingRateKHR **must** have set pFragmentSize->height to 1 prior to this drawing command

- VUID-vkCmdDrawIndexedIndirectCount-None-07749
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_WRITE_ENABLE_EXT dynamic state enabled then vkCmdSetColorWriteEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirectCount-None-08646
  If the colorWriteEnable feature is enabled on the device, and a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetColorWriteEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirectCount-attachmentCount-07750
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_WRITE_ENABLE_EXT dynamic state enabled then the attachmentCount parameter of vkCmdSetColorWriteEnableEXT **must** be greater than or equal to the VkPipelineColorBlendStateCreateInfo::attachmentCount of the currently bound graphics pipeline

- VUID-vkCmdDrawIndexedIndirectCount-None-08647
  If the colorWriteEnable feature is enabled on the device, and a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then the attachmentCount parameter of most recent call to vkCmdSetColorWriteEnableEXT in the current command buffer **must** be greater than or equal to the number of color attachments in the current render pass instance

- VUID-vkCmdDrawIndexedIndirectCount-None-07751
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_DISCARD_RECTANGLE_EXT dynamic state enabled then vkCmdSetDiscardRectangleEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command for each discard rectangle in VkPipelineDiscardRectangleStateCreateInfoEXT::discardRectangleCount

- VUID-vkCmdDrawIndexedIndirectCount-None-07880
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_DISCARD_RECTANGLE_ENABLE_EXT dynamic state enabled then vkCmdSetDiscardRectangleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirectCount-rasterizerDiscardEnable-09236
  If the VK_EXT_discard_rectangles extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetDiscardRectangleEnableEXT in the current command buffer set discardRectangleEnable to VK_TRUE, then vkCmdSetDiscardRectangleEXT **must** have been

called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirectCount-None-08648
  If the VK_EXT_discard_rectangles extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetDiscardRectangleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirectCount-None-07881
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_DISCARD_RECTANGLE_MODE_EXT dynamic state enabled then vkCmdSetDiscardRectangleModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirectCount-None-08649
  If the VK_EXT_discard_rectangles extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetDiscardRectangleEnableEXT in the current command buffer set discardRectangleEnable to VK_TRUE, then vkCmdSetDiscardRectangleModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirectCount-dynamicRenderingUnusedAttachments-08913
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is not enabled, and VkRenderingInfo::pDepthAttachment->imageView was VK_NULL_HANDLE, the value of VkPipelineRenderingCreateInfo::depthAttachmentFormat used to create the currently bound graphics pipeline **must** be equal to VK_FORMAT_UNDEFINED

- VUID-vkCmdDrawIndexedIndirectCount-dynamicRenderingUnusedAttachments-08914
  If current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is not enabled, and VkRenderingInfo::pDepthAttachment->imageView was not VK_NULL_HANDLE, the value of VkPipelineRenderingCreateInfo::depthAttachmentFormat used to create the currently bound graphics pipeline **must** be equal to the VkFormat used to create VkRenderingInfo::pDepthAttachment->imageView

- VUID-vkCmdDrawIndexedIndirectCount-dynamicRenderingUnusedAttachments-08915
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is enabled, VkRenderingInfo::pDepthAttachment->imageView was not VK_NULL_HANDLE, and the value of VkPipelineRenderingCreateInfo::depthAttachmentFormat used to create the currently bound graphics pipeline was not equal to the VkFormat used to create VkRenderingInfo::pDepthAttachment->imageView, the value of the format **must** be VK_FORMAT_UNDEFINED

- VUID-vkCmdDrawIndexedIndirectCount-dynamicRenderingUnusedAttachments-08916
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is not enabled, and VkRenderingInfo::pStencilAttachment->imageView was VK_NULL_HANDLE, the value of

VkPipelineRenderingCreateInfo::stencilAttachmentFormat used to create the currently bound graphics pipeline **must** be equal to VK_FORMAT_UNDEFINED

- VUID-vkCmdDrawIndexedIndirectCount-dynamicRenderingUnusedAttachments-08917
  If current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is not enabled, and VkRenderingInfo::pStencilAttachment->imageView was not VK_NULL_HANDLE, the value of VkPipelineRenderingCreateInfo::stencilAttachmentFormat used to create the currently bound graphics pipeline **must** be equal to the VkFormat used to create VkRenderingInfo::pStencilAttachment->imageView
- VUID-vkCmdDrawIndexedIndirectCount-dynamicRenderingUnusedAttachments-08918
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is enabled, VkRenderingInfo::pStencilAttachment->imageView was not VK_NULL_HANDLE, and the value of VkPipelineRenderingCreateInfo::stencilAttachmentFormat used to create the currently bound graphics pipeline was not equal to the VkFormat used to create VkRenderingInfo::pStencilAttachment->imageView, the value of the format **must** be VK_FORMAT_UNDEFINED
- VUID-vkCmdDrawIndexedIndirectCount-imageView-06183
  If the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingFragmentShadingRateAttachmentInfoKHR::imageView was not VK_NULL_HANDLE, the currently bound graphics pipeline **must** have been created with VK_PIPELINE_CREATE_RENDERING_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR
- VUID-vkCmdDrawIndexedIndirectCount-imageView-06184
  If the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingFragmentDensityMapAttachmentInfoEXT::imageView was not VK_NULL_HANDLE, the currently bound graphics pipeline **must** have been created with VK_PIPELINE_CREATE_RENDERING_FRAGMENT_DENSITY_MAP_ATTACHMENT_BIT_EXT
- VUID-vkCmdDrawIndexedIndirectCount-colorAttachmentCount-06185
  If the currently bound pipeline was created with a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and the current render pass instance was begun with vkCmdBeginRenderingKHR with a VkRenderingInfo::colorAttachmentCount parameter greater than 0, then each element of the VkRenderingInfo::pColorAttachments array with a imageView not equal to VK_NULL_HANDLE **must** have been created with a sample count equal to the corresponding element of the pColorAttachmentSamples member of VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV used to create the currently bound graphics pipeline
- VUID-vkCmdDrawIndexedIndirectCount-pDepthAttachment-06186
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the currently bound pipeline was created with a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and VkRenderingInfo::pDepthAttachment->imageView was not VK_NULL_HANDLE, the value of the depthStencilAttachmentSamples member of VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV used to create the currently bound graphics pipeline **must** be equal to the sample count used to create VkRenderingInfo::pDepthAttachment->imageView
- VUID-vkCmdDrawIndexedIndirectCount-pStencilAttachment-06187
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the

currently bound pipeline was created with a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and VkRenderingInfo::pStencilAttachment->imageView was not VK_NULL_HANDLE, the value of the depthStencilAttachmentSamples member of VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV used to create the currently bound graphics pipeline **must** be equal to the sample count used to create VkRenderingInfo::pStencilAttachment->imageView

- VUID-vkCmdDrawIndexedIndirectCount-multisampledRenderToSingleSampled-07285
  If the currently bound pipeline was created without a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and the multisampledRenderToSingleSampled feature is not enabled, and the current render pass instance was begun with vkCmdBeginRenderingKHR with a VkRenderingInfo::colorAttachmentCount parameter greater than 0, then each element of the VkRenderingInfo::pColorAttachments array with a imageView not equal to VK_NULL_HANDLE **must** have been created with a sample count equal to the value of rasterizationSamples for the currently bound graphics pipeline
- VUID-vkCmdDrawIndexedIndirectCount-multisampledRenderToSingleSampled-07286
  If the currently bound pipeline was created without a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and the multisampledRenderToSingleSampled feature is not enabled, and VkRenderingInfo::pDepthAttachment->imageView was not VK_NULL_HANDLE, the value of rasterizationSamples for the currently bound graphics pipeline **must** be equal to the sample count used to create VkRenderingInfo::pDepthAttachment->imageView
- VUID-vkCmdDrawIndexedIndirectCount-multisampledRenderToSingleSampled-07287
  If the currently bound pipeline was created without a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and the multisampledRenderToSingleSampled feature is not enabled, and VkRenderingInfo::pStencilAttachment->imageView was not VK_NULL_HANDLE, the value of rasterizationSamples for the currently bound graphics pipeline **must** be equal to the sample count used to create VkRenderingInfo::pStencilAttachment->imageView
- VUID-vkCmdDrawIndexedIndirectCount-pNext-07935
  If this command has been called inside a render pass instance started with vkCmdBeginRenderingKHR, and the pNext chain of VkRenderingInfo includes a VkMultisampledRenderToSingleSampledInfoEXT structure with multisampledRenderToSingleSampledEnable equal to VK_TRUE, then the value of rasterizationSamples for the currently bound graphics pipeline **must** be equal to VkMultisampledRenderToSingleSampledInfoEXT::rasterizationSamples
- VUID-vkCmdDrawIndexedIndirectCount-renderPass-06198
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the currently bound pipeline **must** have been created with a VkGraphicsPipelineCreateInfo::renderPass equal to VK_NULL_HANDLE
- VUID-vkCmdDrawIndexedIndirectCount-pColorAttachments-08963
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is a graphics pipeline bound with a fragment shader that statically writes to a color attachment, the color write mask is not zero, color writes are enabled, and the corresponding element of the VkRenderingInfo::pColorAttachments->imageView was not

VK_NULL_HANDLE, then the corresponding element of VkPipelineRenderingCreateInfo::pColorAttachmentFormats used to create the pipeline **must** not be VK_FORMAT_UNDEFINED

- VUID-vkCmdDrawIndexedIndirectCount-pDepthAttachment-08964
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is a graphics pipeline bound, depth test is enabled, depth write is enabled, and the VkRenderingInfo::pDepthAttachment->imageView was not VK_NULL_HANDLE, then the VkPipelineRenderingCreateInfo::depthAttachmentFormat used to create the pipeline **must** not be VK_FORMAT_UNDEFINED

- VUID-vkCmdDrawIndexedIndirectCount-pStencilAttachment-08965
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is a graphics pipeline bound, stencil test is enabled and the VkRenderingInfo::pStencilAttachment->imageView was not VK_NULL_HANDLE, then the VkPipelineRenderingCreateInfo::stencilAttachmentFormat used to create the pipeline **must** not be VK_FORMAT_UNDEFINED

- VUID-vkCmdDrawIndexedIndirectCount-primitivesGeneratedQueryWithRasterizerDiscard-06708
  If the primitivesGeneratedQueryWithRasterizerDiscard feature is not enabled and the VK_QUERY_TYPE_PRIMITIVES_GENERATED_EXT query is active, rasterization discard **must** not be enabled

- VUID-vkCmdDrawIndexedIndirectCount-primitivesGeneratedQueryWithNonZeroStreams-06709
  If the primitivesGeneratedQueryWithNonZeroStreams feature is not enabled and the VK_QUERY_TYPE_PRIMITIVES_GENERATED_EXT query is active, the bound graphics pipeline **must** not have been created with a non-zero value in VkPipelineRasterizationStateStreamCreateInfoEXT::rasterizationStream

- VUID-vkCmdDrawIndexedIndirectCount-None-07619
  If a shader object is bound to the VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_TESSELLATION_DOMAIN_ORIGIN_EXT dynamic state enabled, then vkCmdSetTessellationDomainOriginEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirectCount-None-07620
  If the depthClamp feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_CLAMP_ENABLE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetDepthClampEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirectCount-None-07621
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_POLYGON_MODE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetPolygonModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirectCount-None-07622

If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetRasterizationSamplesEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirectCount-None-07623
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_SAMPLE_MASK_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetSampleMaskEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirectCount-alphaToCoverageEnable-08919
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_ALPHA_TO_COVERAGE_ENABLE_EXT dynamic state enabled, and alphaToCoverageEnable was VK_TRUE in the last call to vkCmdSetAlphaToCoverageEnableEXT, then the Fragment Output Interface **must** contain a variable for the alpha Component word in Location 0 at Index 0

- VUID-vkCmdDrawIndexedIndirectCount-alphaToCoverageEnable-08920
  If a shader object is bound to any graphics stage, and the most recent call to vkCmdSetAlphaToCoverageEnableEXT in the current command buffer set alphaToCoverageEnable to VK_TRUE, then the Fragment Output Interface **must** contain a variable for the alpha Component word in Location 0 at Index 0

- VUID-vkCmdDrawIndexedIndirectCount-None-07624
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_ALPHA_TO_COVERAGE_ENABLE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetAlphaToCoverageEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirectCount-None-07625
  If the alphaToOne feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_ALPHA_TO_ONE_ENABLE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetAlphaToOneEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirectCount-None-07626
  If the logicOp feature is enabled, a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_LOGIC_OP_ENABLE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetLogicOpEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirectCount-None-07627
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT dynamic state enabled then

vkCmdSetColorBlendEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirectCount-None-08657
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and both the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE and there are color attachments bound, then vkCmdSetColorBlendEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirectCount-None-07628
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_EQUATION_EXT dynamic state enabled then vkCmdSetColorBlendEquationEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirectCount-None-08658
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetColorBlendEnableEXT for any attachment set that attachment's value in pColorBlendEnables to VK_TRUE, then vkCmdSetColorBlendEquationEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirectCount-None-07629
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_WRITE_MASK_EXT dynamic state enabled then vkCmdSetColorWriteMaskEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirectCount-None-08659
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and both the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE and there are color attachments bound, then vkCmdSetColorWriteMaskEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirectCount-None-07630
  If the geometryStreams feature is enabled, and a shader object is bound to the VK_SHADER_STAGE_GEOMETRY_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_RASTERIZATION_STREAM_EXT dynamic state enabled, then vkCmdSetRasterizationStreamEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirectCount-None-07631
  If the VK_EXT_conservative_rasterization extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_CONSERVATIVE_RASTERIZATION_MODE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetConservativeRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirectCount-None-07632
  If the VK_EXT_conservative_rasterization extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_EXTRA_PRIMITIVE_OVERESTIMATION_SIZE_EXT dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of conservativeRasterizationMode is VK_CONSERVATIVE_RASTERIZATION_MODE_OVERESTIMATE_EXT, then vkCmdSetExtraPrimitiveOverestimationSizeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirectCount-None-07633
  If the depthClipEnable feature is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_CLIP_ENABLE_EXT dynamic state, then vkCmdSetDepthClipEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirectCount-None-07634
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT dynamic state enabled then vkCmdSetSampleLocationsEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirectCount-None-08664
  If the VK_EXT_sample_locations extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetSampleLocationsEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirectCount-None-07635
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_ADVANCED_EXT dynamic state enabled then vkCmdSetColorBlendAdvancedEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirectCount-rasterizerDiscardEnable-09416
  If the VK_EXT_blend_operation_advanced extension is enabled, and a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then at least one of vkCmdSetColorBlendEquationEXT and vkCmdSetColorBlendAdvancedEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirectCount-None-07636
  If the VK_EXT_provoking_vertex extension is enabled, a shader object is bound to the VK_SHADER_STAGE_VERTEX_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_PROVOKING_VERTEX_MODE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetProvokingVertexModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirectCount-None-07637
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT dynamic state enabled then vkCmdSetLineRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirectCount-None-08666
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetPolygonModeEXT in the current command buffer set polygonMode to VK_POLYGON_MODE_LINE, then vkCmdSetLineRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirectCount-None-08667
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object is bound to the VK_SHADER_STAGE_VERTEX_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetPrimitiveTopologyEXT in the current command buffer set primitiveTopology to any line topology, then vkCmdSetLineRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirectCount-None-08668
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object that outputs line primitives is bound to the VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT or VK_SHADER_STAGE_GEOMETRY_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetLineRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirectCount-None-07638
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT dynamic state enabled then vkCmdSetLineStippleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirectCount-None-08669
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetPolygonModeEXT in the current command buffer set polygonMode to VK_POLYGON_MODE_LINE, then vkCmdSetLineStippleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirectCount-None-08670
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object is bound to the VK_SHADER_STAGE_VERTEX_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetPrimitiveTopologyEXT in the current command buffer set primitiveTopology to any line topology, then vkCmdSetLineStippleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirectCount-None-08671
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object that outputs line primitives is bound to the VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT or VK_SHADER_STAGE_GEOMETRY_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetLineStippleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirectCount-None-07849
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_KHR dynamic state enabled then vkCmdSetLineStippleKHR **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirectCount-None-08672
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetLineStippleEnableEXT in the current command buffer set stippledLineEnable to VK_TRUE, then vkCmdSetLineStippleEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirectCount-None-07639
  If the depthClipControl feature is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_CLIP_NEGATIVE_ONE_TO_ONE_EXT dynamic state enabled, then vkCmdSetDepthClipNegativeOneToOneEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirectCount-None-07640
  If the VK_NV_clip_space_w_scaling extension is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_VIEWPORT_W_SCALING_ENABLE_NV dynamic state enabled, then vkCmdSetViewportWScalingEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirectCount-None-07641
  If the VK_NV_viewport_swizzle extension is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_VIEWPORT_SWIZZLE_NV dynamic state enabled, then

vkCmdSetViewportSwizzleNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirectCount-None-07642
  If the VK_NV_fragment_coverage_to_color extension is enabled, a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_TO_COLOR_ENABLE_NV dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetCoverageToColorEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirectCount-None-07643
  If the VK_NV_fragment_coverage_to_color extension is enabled, a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_TO_COLOR_LOCATION_NV dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of coverageToColorEnable is VK_TRUE, then vkCmdSetCoverageToColorLocationNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirectCount-None-07644
  If the VK_NV_framebuffer_mixed_samples extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_MODULATION_MODE_NV dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetCoverageModulationModeNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirectCount-None-07645
  If the VK_NV_framebuffer_mixed_samples extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_MODULATION_TABLE_ENABLE_NV dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of coverageModulationMode is any value other than VK_COVERAGE_MODULATION_MODE_NONE_NV, then vkCmdSetCoverageModulationTableEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirectCount-None-07646
  If the VK_NV_framebuffer_mixed_samples extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_MODULATION_TABLE_NV dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of coverageModulationTableEnable is VK_TRUE, then vkCmdSetCoverageModulationTableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirectCount-None-07647
  If the shadingRateImage feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_SHADING_RATE_IMAGE_ENABLE_NV dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetShadingRateImageEnableNV

**must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirectCount-pipelineFragmentShadingRate-09238
  If the `pipelineFragmentShadingRate` feature is enabled, a shader object is bound to the `VK_SHADER_STAGE_FRAGMENT_BIT` stage or a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR` dynamic state enabled, and the current value of `rasterizerDiscardEnable` is `VK_FALSE`, then vkCmdSetFragmentShadingRateKHR **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirectCount-None-07648
  If the `representativeFragmentTest` feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_REPRESENTATIVE_FRAGMENT_TEST_ENABLE_NV` dynamic state enabled, and the current value of `rasterizerDiscardEnable` is `VK_FALSE`, then vkCmdSetRepresentativeFragmentTestEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirectCount-None-07649
  If the `coverageReductionMode` feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_COVERAGE_REDUCTION_MODE_NV` dynamic state enabled, and the current value of `rasterizerDiscardEnable` is `VK_FALSE`, then vkCmdSetCoverageReductionModeNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirectCount-pColorBlendEnables-07470
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT` state enabled and the last call to vkCmdSetColorBlendEnableEXT set `pColorBlendEnables` for any attachment to `VK_TRUE`, then for those attachments in the subpass the corresponding image view's format features **must** contain `VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT`

- VUID-vkCmdDrawIndexedIndirectCount-rasterizationSamples-07471
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT` state enabled, and the current subpass does not use any color and/or depth/stencil attachments, then the `rasterizationSamples` in the last call to vkCmdSetRasterizationSamplesEXT **must** follow the rules for a zero-attachment subpass

- VUID-vkCmdDrawIndexedIndirectCount-samples-07472
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_SAMPLE_MASK_EXT` state enabled and the `VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT` state disabled, then the `samples` parameter in the last call to vkCmdSetSampleMaskEXT **must** be greater or equal to the VkPipelineMultisampleStateCreateInfo::`rasterizationSamples` parameter used to create the bound graphics pipeline

- VUID-vkCmdDrawIndexedIndirectCount-samples-07473
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_SAMPLE_MASK_EXT` state and `VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT`

states enabled, then the samples parameter in the last call to vkCmdSetSampleMaskEXT **must** be greater or equal to the rasterizationSamples parameter in the last call to vkCmdSetRasterizationSamplesEXT

- VUID-vkCmdDrawIndexedIndirectCount-rasterizationSamples-07474
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, and neither the VK_AMD_mixed_attachment_samples nor the VK_NV_framebuffer_mixed_samples extensions are enabled, then the rasterizationSamples in the last call to vkCmdSetRasterizationSamplesEXT **must** be the same as the current subpass color and/or depth/stencil attachments

- VUID-vkCmdDrawIndexedIndirectCount-None-09211
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, or a shader object is bound to any graphics stage, and the current render pass instance includes a VkMultisampledRenderToSingleSampledInfoEXT structure with multisampledRenderToSingleSampledEnable equal to VK_TRUE, then the rasterizationSamples in the last call to vkCmdSetRasterizationSamplesEXT **must** be the same as the rasterizationSamples member of that structure

- VUID-vkCmdDrawIndexedIndirectCount-firstAttachment-07476
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT dynamic state enabled then vkCmdSetColorBlendEnableEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the firstAttachment and attachmentCount parameters of vkCmdSetColorBlendEnableEXT calls **must** specify an enable for all active color attachments in the current subpass

- VUID-vkCmdDrawIndexedIndirectCount-rasterizerDiscardEnable-09417
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetColorBlendEnableEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the firstAttachment and attachmentCount parameters of vkCmdSetColorBlendEnableEXT calls **must** specify an enable for all active color attachments in the current subpass

- VUID-vkCmdDrawIndexedIndirectCount-firstAttachment-07477
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_EQUATION_EXT dynamic state enabled then vkCmdSetColorBlendEquationEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the firstAttachment and attachmentCount parameters of vkCmdSetColorBlendEquationEXT calls **must** specify the blend equations for all active color attachments in the current subpass where blending is enabled

- VUID-vkCmdDrawIndexedIndirectCount-rasterizerDiscardEnable-09418
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and both the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE and there are color attachments bound, then

vkCmdSetColorBlendEquationEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the `firstAttachment` and `attachmentCount` parameters of `vkCmdSetColorBlendEquationEXT` calls **must** specify the blend equations for all active color attachments in the current subpass where blending is enabled

- VUID-vkCmdDrawIndexedIndirectCount-firstAttachment-07478
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_COLOR_WRITE_MASK_EXT` dynamic state enabled then vkCmdSetColorWriteMaskEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the `firstAttachment` and `attachmentCount` parameters of `vkCmdSetColorWriteMaskEXT` calls **must** specify the color write mask for all active color attachments in the current subpass

- VUID-vkCmdDrawIndexedIndirectCount-rasterizerDiscardEnable-09419
  If a shader object is bound to the `VK_SHADER_STAGE_FRAGMENT_BIT` stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE`, then vkCmdSetColorWriteMaskEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the `firstAttachment` and `attachmentCount` parameters of `vkCmdSetColorWriteMaskEXT` calls **must** specify the color write mask for all active color attachments in the current subpass

- VUID-vkCmdDrawIndexedIndirectCount-firstAttachment-07479
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_COLOR_BLEND_ADVANCED_EXT` dynamic state enabled then vkCmdSetColorBlendAdvancedEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the `firstAttachment` and `attachmentCount` parameters of `vkCmdSetColorBlendAdvancedEXT` calls **must** specify the advanced blend equations for all active color attachments in the current subpass where blending is enabled

- VUID-vkCmdDrawIndexedIndirectCount-advancedBlendMaxColorAttachments-07480
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_COLOR_BLEND_ADVANCED_EXT` and `VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT` dynamic states enabled and the last calls to vkCmdSetColorBlendEnableEXT and vkCmdSetColorBlendAdvancedEXT have enabled advanced blending, then the number of active color attachments in the current subpass **must** not exceed `advancedBlendMaxColorAttachments`

- VUID-vkCmdDrawIndexedIndirectCount-primitivesGeneratedQueryWithNonZeroStreams-07481
  If the `primitivesGeneratedQueryWithNonZeroStreams` feature is not enabled and the `VK_QUERY_TYPE_PRIMITIVES_GENERATED_EXT` query is active, and the bound graphics pipeline was created with `VK_DYNAMIC_STATE_RASTERIZATION_STREAM_EXT` state enabled, the last call to vkCmdSetRasterizationStreamEXT **must** have set the `rasterizationStream` to zero

- VUID-vkCmdDrawIndexedIndirectCount-sampleLocationsPerPixel-07482
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT` state enabled and the `VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT` state disabled, then the

sampleLocationsPerPixel member of pSampleLocationsInfo in the last call to vkCmdSetSampleLocationsEXT **must** equal the rasterizationSamples member of the VkPipelineMultisampleStateCreateInfo structure the bound graphics pipeline has been created with

- VUID-vkCmdDrawIndexedIndirectCount-sampleLocationsPerPixel-07483
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state enabled and the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, then the sampleLocationsPerPixel member of pSampleLocationsInfo in the last call to vkCmdSetSampleLocationsEXT **must** equal the rasterizationSamples parameter of the last call to vkCmdSetRasterizationSamplesEXT

- VUID-vkCmdDrawIndexedIndirectCount-sampleLocationsEnable-07484
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, or the bound graphics pipeline was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, and sampleLocationsEnable was VK_TRUE in the last call to vkCmdSetSampleLocationsEnableEXT, and the current subpass has a depth/stencil attachment, then that attachment **must** have been created with the VK_IMAGE_CREATE_SAMPLE_LOCATIONS_COMPATIBLE_DEPTH_BIT_EXT bit set

- VUID-vkCmdDrawIndexedIndirectCount-sampleLocationsEnable-07485
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state enabled and the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, and if sampleLocationsEnable was VK_TRUE in the last call to vkCmdSetSampleLocationsEnableEXT, then the sampleLocationsInfo.sampleLocationGridSize.width in the last call to vkCmdSetSampleLocationsEXT **must** evenly divide VkMultisamplePropertiesEXT::sampleLocationGridSize.width as returned by vkGetPhysicalDeviceMultisamplePropertiesEXT with a samples parameter equaling rasterizationSamples

- VUID-vkCmdDrawIndexedIndirectCount-sampleLocationsEnable-07486
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state enabled and the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, and if sampleLocationsEnable was VK_TRUE in the last call to vkCmdSetSampleLocationsEnableEXT, then the sampleLocationsInfo.sampleLocationGridSize.height in the last call to vkCmdSetSampleLocationsEXT **must** evenly divide VkMultisamplePropertiesEXT::sampleLocationGridSize.height as returned by vkGetPhysicalDeviceMultisamplePropertiesEXT with a samples parameter equaling rasterizationSamples

- VUID-vkCmdDrawIndexedIndirectCount-sampleLocationsEnable-07487
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, and if sampleLocationsEnable was VK_TRUE in the last call to vkCmdSetSampleLocationsEnableEXT, the fragment shader code **must** not statically use the extended instruction InterpolateAtSample

- VUID-vkCmdDrawIndexedIndirectCount-sampleLocationsEnable-07936
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT` state disabled and the `VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT` state enabled, the `sampleLocationsEnable` member of a VkPipelineSampleLocationsStateCreateInfoEXT::`sampleLocationsEnable` in the bound graphics pipeline is `VK_TRUE` or `VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT` state enabled, then, `sampleLocationsInfo.sampleLocationGridSize.width` **must** evenly divide VkMultisamplePropertiesEXT::`sampleLocationGridSize.width` as returned by vkGetPhysicalDeviceMultisamplePropertiesEXT with a `samples` parameter equaling the value of `rasterizationSamples` in the last call to vkCmdSetRasterizationSamplesEXT
- VUID-vkCmdDrawIndexedIndirectCount-sampleLocationsEnable-07937
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT` state disabled and the `VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT` state enabled, the `sampleLocationsEnable` member of a VkPipelineSampleLocationsStateCreateInfoEXT::`sampleLocationsEnable` in the bound graphics pipeline is `VK_TRUE` or `VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT` state enabled, then, `sampleLocationsInfo.sampleLocationGridSize.height` **must** evenly divide VkMultisamplePropertiesEXT::`sampleLocationGridSize.height` as returned by vkGetPhysicalDeviceMultisamplePropertiesEXT with a `samples` parameter equaling the value of `rasterizationSamples` in the last call to vkCmdSetRasterizationSamplesEXT
- VUID-vkCmdDrawIndexedIndirectCount-sampleLocationsEnable-07938
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT` state disabled and the `VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT` state enabled, the `sampleLocationsEnable` member of a VkPipelineSampleLocationsStateCreateInfoEXT::`sampleLocationsEnable` in the bound graphics pipeline is `VK_TRUE` or `VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT` state enabled, then, `sampleLocationsInfo.sampleLocationsPerPixel` **must** equal `rasterizationSamples` in the last call to vkCmdSetRasterizationSamplesEXT
- VUID-vkCmdDrawIndexedIndirectCount-coverageModulationTableEnable-07488
  If a shader object is bound to any graphics stage or the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_COVERAGE_MODULATION_TABLE_ENABLE_NV` state enabled, and the last call to vkCmdSetCoverageModulationTableEnableNV set `coverageModulationTableEnable` to `VK_TRUE`, then the `coverageModulationTableCount` parameter in the last call to vkCmdSetCoverageModulationTableNV **must** equal the current `rasterizationSamples` divided by the number of color samples in the current subpass
- VUID-vkCmdDrawIndexedIndirectCount-rasterizationSamples-07489
  If the `VK_NV_framebuffer_mixed_samples` extension is enabled, and if current subpass has a depth/stencil attachment and depth test, stencil test, or depth bounds test are enabled in the currently bound pipeline state, then the current `rasterizationSamples` **must** be the same as the sample count of the depth/stencil attachment
- VUID-vkCmdDrawIndexedIndirectCount-coverageToColorEnable-07490
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_COVERAGE_TO_COLOR_ENABLE_NV` state enabled and the last call to vkCmdSetCoverageToColorEnableNV set the `coverageToColorEnable` to `VK_TRUE`, then the current subpass **must** have a color attachment at the location selected by the last call to

vkCmdSetCoverageToColorLocationNV coverageToColorLocation, with a VkFormat of VK_FORMAT_R8_UINT, VK_FORMAT_R8_SINT, VK_FORMAT_R16_UINT, VK_FORMAT_R16_SINT, VK_FORMAT_R32_UINT, or VK_FORMAT_R32_SINT

- VUID-vkCmdDrawIndexedIndirectCount-rasterizerDiscardEnable-09420
  If the VK_NV_fragment_coverage_to_color extension is enabled, and a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the last call to vkCmdSetCoverageToColorEnableNV set the coverageToColorEnable to VK_TRUE, then the current subpass **must** have a color attachment at the location selected by the last call to vkCmdSetCoverageToColorLocationNV coverageToColorLocation, with a VkFormat of VK_FORMAT_R8_UINT, VK_FORMAT_R8_SINT, VK_FORMAT_R16_UINT, VK_FORMAT_R16_SINT, VK_FORMAT_R32_UINT, or VK_FORMAT_R32_SINT

- VUID-vkCmdDrawIndexedIndirectCount-coverageReductionMode-07491
  If this VK_NV_coverage_reduction_mode extension is enabled, the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COVERAGE_TO_COLOR_ENABLE_NV and VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT states enabled, the current coverage reduction mode coverageReductionMode, then the current rasterizationSamples, and the sample counts for the color and depth/stencil attachments (if the subpass has them) **must** be a valid combination returned by vkGetPhysicalDeviceSupportedFramebufferMixedSamplesCombinationsNV

- VUID-vkCmdDrawIndexedIndirectCount-viewportCount-07492
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, but not the VK_DYNAMIC_STATE_VIEWPORT_SWIZZLE_NV dynamic state enabled, then the bound graphics pipeline **must** have been created with VkPipelineViewportSwizzleStateCreateInfoNV ::viewportCount greater or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT

- VUID-vkCmdDrawIndexedIndirectCount-viewportCount-07493
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT and VK_DYNAMIC_STATE_VIEWPORT_SWIZZLE_NV dynamic states enabled then the viewportCount parameter in the last call to vkCmdSetViewportSwizzleNV **must** be greater than or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT

- VUID-vkCmdDrawIndexedIndirectCount-viewportCount-09421
  If the VK_NV_viewport_swizzle extension is enabled, and a shader object is bound to any graphics stage, then the viewportCount parameter in the last call to vkCmdSetViewportSwizzleNV **must** be greater than or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT

- VUID-vkCmdDrawIndexedIndirectCount-rasterizationSamples-07494
  If the VK_NV_framebuffer_mixed_samples extension is enabled, and if the current subpass has any color attachments and rasterizationSamples of the last call to vkCmdSetRasterizationSamplesEXT is greater than the number of color samples, then the pipeline sampleShadingEnable **must** be VK_FALSE

- VUID-vkCmdDrawIndexedIndirectCount-stippledLineEnable-07495

If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT or VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT dynamic states enabled, and if the current stippledLineEnable state is VK_TRUE and the current lineRasterizationMode state is VK_LINE_RASTERIZATION_MODE_RECTANGULAR_KHR, then the stippledRectangularLines feature **must** be enabled

- VUID-vkCmdDrawIndexedIndirectCount-stippledLineEnable-07496
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT or VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT dynamic states enabled, and if the current stippledLineEnable state is VK_TRUE and the current lineRasterizationMode state is VK_LINE_RASTERIZATION_MODE_BRESENHAM_KHR, then the stippledBresenhamLines feature **must** be enabled

- VUID-vkCmdDrawIndexedIndirectCount-stippledLineEnable-07497
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT or VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT dynamic states enabled, and if the current stippledLineEnable state is VK_TRUE and the current lineRasterizationMode state is VK_LINE_RASTERIZATION_MODE_RECTANGULAR_SMOOTH_KHR, then the stippledSmoothLines feature **must** be enabled

- VUID-vkCmdDrawIndexedIndirectCount-stippledLineEnable-07498
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT or VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT dynamic states enabled, and if the current stippledLineEnable state is VK_TRUE and the current lineRasterizationMode state is VK_LINE_RASTERIZATION_MODE_DEFAULT_KHR, then the stippledRectangularLines feature **must** be enabled and VkPhysicalDeviceLimits::strictLines **must** be VK_TRUE

- VUID-vkCmdDrawIndexedIndirectCount-conservativePointAndLineRasterization-07499
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_CONSERVATIVE_RASTERIZATION_MODE_EXT dynamic state enabled, conservativePointAndLineRasterization is not supported, and the effective primitive topology output by the last pre-rasterization shader stage is a line or point, then the conservativeRasterizationMode set by the last call to vkCmdSetConservativeRasterizationModeEXT **must** be VK_CONSERVATIVE_RASTERIZATION_MODE_DISABLED_EXT

- VUID-vkCmdDrawIndexedIndirectCount-stage-07073
  If the currently bound pipeline was created with the VkPipelineShaderStageCreateInfo::stage member of an element of VkGraphicsPipelineCreateInfo::pStages set to VK_SHADER_STAGE_VERTEX_BIT, VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT, VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT or VK_SHADER_STAGE_GEOMETRY_BIT, then Mesh Shader Queries **must** not be active

- VUID-vkCmdDrawIndexedIndirectCount-None-08877
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_ATTACHMENT_FEEDBACK_LOOP_ENABLE_EXT dynamic state enabled, and the

current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetAttachmentFeedbackLoopEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirectCount-None-07850
  If dynamic state was inherited from VkCommandBufferInheritanceViewportScissorInfoNV, it **must** be set in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirectCount-None-08684
  If there is no bound graphics pipeline, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_VERTEX_BIT
- VUID-vkCmdDrawIndexedIndirectCount-None-08685
  If there is no bound graphics pipeline, and the tessellationShader feature is enabled, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT
- VUID-vkCmdDrawIndexedIndirectCount-None-08686
  If there is no bound graphics pipeline, and the tessellationShader feature is enabled, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT
- VUID-vkCmdDrawIndexedIndirectCount-None-08687
  If there is no bound graphics pipeline, and the geometryShader feature is enabled, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_GEOMETRY_BIT
- VUID-vkCmdDrawIndexedIndirectCount-None-08688
  If there is no bound graphics pipeline, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_FRAGMENT_BIT
- VUID-vkCmdDrawIndexedIndirectCount-None-08689
  If there is no bound graphics pipeline, and the taskShader feature is enabled, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_TASK_BIT_EXT
- VUID-vkCmdDrawIndexedIndirectCount-None-08690
  If there is no bound graphics pipeline, and the meshShader feature is enabled, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_MESH_BIT_EXT
- VUID-vkCmdDrawIndexedIndirectCount-None-08693
  If there is no bound graphics pipeline, and at least one of the taskShader and meshShader features is enabled, one of the VK_SHADER_STAGE_VERTEX_BIT or VK_SHADER_STAGE_MESH_BIT_EXT stages **must** have a valid VkShaderEXT bound, and the other **must** have no VkShaderEXT bound
- VUID-vkCmdDrawIndexedIndirectCount-None-08694
  If there is no bound graphics pipeline, and both the taskShader and meshShader features are enabled, and a valid VkShaderEXT is bound the to the VK_SHADER_STAGE_MESH_BIT_EXT stage, and that VkShaderEXT was created without the VK_SHADER_CREATE_NO_TASK_SHADER_BIT_EXT flag, a valid VkShaderEXT **must** be bound to the VK_SHADER_STAGE_TASK_BIT_EXT stage

- VUID-vkCmdDrawIndexedIndirectCount-None-08695
  If there is no bound graphics pipeline, and both the taskShader and meshShader features are enabled, and a valid VkShaderEXT is bound the to the VK_SHADER_STAGE_MESH_BIT_EXT stage, and that VkShaderEXT was created with the VK_SHADER_CREATE_NO_TASK_SHADER_BIT_EXT flag, there **must** be no VkShaderEXT bound to the VK_SHADER_STAGE_TASK_BIT_EXT stage
- VUID-vkCmdDrawIndexedIndirectCount-None-08696
  If there is no bound graphics pipeline, and a valid VkShaderEXT is bound to the VK_SHADER_STAGE_VERTEX_BIT stage, there **must** be no VkShaderEXT bound to either the VK_SHADER_STAGE_TASK_BIT_EXT stage or the VK_SHADER_STAGE_MESH_BIT_EXT stage
- VUID-vkCmdDrawIndexedIndirectCount-None-08698
  If any graphics shader is bound which was created with the VK_SHADER_CREATE_LINK_STAGE_BIT_EXT flag, then all shaders created with the VK_SHADER_CREATE_LINK_STAGE_BIT_EXT flag in the same vkCreateShadersEXT call **must** also be bound
- VUID-vkCmdDrawIndexedIndirectCount-None-08699
  If any graphics shader is bound which was created with the VK_SHADER_CREATE_LINK_STAGE_BIT_EXT flag, any stages in between stages whose shaders which did not create a shader with the VK_SHADER_CREATE_LINK_STAGE_BIT_EXT flag as part of the same vkCreateShadersEXT call **must** not have any VkShaderEXT bound
- VUID-vkCmdDrawIndexedIndirectCount-None-08878
  All bound graphics shader objects **must** have been created with identical or identically defined push constant ranges
- VUID-vkCmdDrawIndexedIndirectCount-None-08879
  All bound graphics shader objects **must** have been created with identical or identically defined arrays of descriptor set layouts
- VUID-vkCmdDrawIndexedIndirectCount-colorAttachmentCount-09372
  If the current render pass instance was begun with vkCmdBeginRenderingKHR and a VkRenderingInfo::colorAttachmentCount equal to 1, a color attachment with a resolve mode of VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID, and a fragment shader is bound, it **must** not declare the DepthReplacing or StencilRefReplacingEXT execution modes
- VUID-vkCmdDrawIndexedIndirectCount-pDynamicStates-08715
  If the bound graphics pipeline state includes a fragment shader stage, was created with VK_DYNAMIC_STATE_DEPTH_WRITE_ENABLE set in VkPipelineDynamicStateCreateInfo::pDynamicStates, and the fragment shader declares the EarlyFragmentTests execution mode and uses OpDepthAttachmentReadEXT, the depthWriteEnable parameter in the last call to vkCmdSetDepthWriteEnableEXT **must** be VK_FALSE
- VUID-vkCmdDrawIndexedIndirectCount-pDynamicStates-08716
  If the bound graphics pipeline state includes a fragment shader stage, was created with VK_DYNAMIC_STATE_STENCIL_WRITE_MASK set in VkPipelineDynamicStateCreateInfo::pDynamicStates, and the fragment shader declares the EarlyFragmentTests execution mode and uses OpStencilAttachmentReadEXT, the writeMask parameter in the last call to vkCmdSetStencilWriteMask **must** be 0
- VUID-vkCmdDrawIndexedIndirectCount-None-09116
  If a shader object is bound to any graphics stage or the currently bound graphics pipeline

was created with VK_DYNAMIC_STATE_COLOR_WRITE_MASK_EXT, and the format of any color attachment is VK_FORMAT_E5B9G9R9_UFLOAT_PACK32, the corresponding element of the pColorWriteMasks parameter of vkCmdSetColorWriteMaskEXT **must** either include all of VK_COLOR_COMPONENT_R_BIT, VK_COLOR_COMPONENT_G_BIT, and VK_COLOR_COMPONENT_B_BIT, or none of them

- VUID-vkCmdDrawIndexedIndirectCount-maxFragmentDualSrcAttachments-09239
  If blending is enabled for any attachment where either the source or destination blend factors for that attachment use the secondary color input, the maximum value of Location for any output attachment statically used in the Fragment Execution Model executed by this command **must** be less than maxFragmentDualSrcAttachments

- VUID-vkCmdDrawIndexedIndirectCount-None-09548
  If the current render pass was begun with vkCmdBeginRenderingKHR, and there is no shader object bound to any graphics stage, the value of each element of VkRenderingAttachmentLocationInfoKHR::pColorAttachmentLocations set by vkCmdSetRenderingAttachmentLocationsKHR **must** match the value set for the corresponding element in the currently bound pipeline

- VUID-vkCmdDrawIndexedIndirectCount-None-09549
  If the current render pass was begun with vkCmdBeginRenderingKHR, and there is no shader object bound to any graphics stage, input attachment index mappings in the currently bound pipeline **must** match those set for the current render pass instance via VkRenderingInputAttachmentIndexInfoKHR

- VUID-vkCmdDrawIndexedIndirectCount-None-09642
  If the current render pass was begun with vkCmdBeginRenderingKHR with the VK_RENDERING_ENABLE_LEGACY_DITHERING_BIT_EXT flag, the bound graphics pipeline **must** have been created with VK_PIPELINE_CREATE_2_ENABLE_LEGACY_DITHERING_BIT_EXT

- VUID-vkCmdDrawIndexedIndirectCount-None-09643
  If the bound graphics pipeline was created with VK_PIPELINE_CREATE_2_ENABLE_LEGACY_DITHERING_BIT_EXT, the current render pass **must** have begun with vkCmdBeginRenderingKHR with the VK_RENDERING_ENABLE_LEGACY_DITHERING_BIT_EXT flag

- VUID-vkCmdDrawIndexedIndirectCount-None-04007
  All vertex input bindings accessed via vertex input variables declared in the vertex shader entry point's interface **must** have either valid or VK_NULL_HANDLE buffers bound

- VUID-vkCmdDrawIndexedIndirectCount-None-04008
  If the nullDescriptor feature is not enabled, all vertex input bindings accessed via vertex input variables declared in the vertex shader entry point's interface **must** not be VK_NULL_HANDLE

- VUID-vkCmdDrawIndexedIndirectCount-None-02721
  If robustBufferAccess is not enabled, and that pipeline was created without enabling VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_EXT for vertexInputs, then for a given vertex buffer binding, any attribute data fetched **must** be entirely contained within the corresponding vertex buffer binding, as described in Vertex Input Description

- VUID-vkCmdDrawIndexedIndirectCount-None-07842

If there is a shader object bound to the VK_SHADER_STAGE_VERTEX_BIT stage or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_PRIMITIVE_TOPOLOGY dynamic state enabled then vkCmdSetPrimitiveTopologyEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirectCount-dynamicPrimitiveTopologyUnrestricted-07500
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_PRIMITIVE_TOPOLOGY dynamic state enabled and the dynamicPrimitiveTopologyUnrestricted is VK_FALSE, then the primitiveTopology parameter of vkCmdSetPrimitiveTopologyEXT **must** be of the same topology class as the pipeline VkPipelineInputAssemblyStateCreateInfo::topology state
- VUID-vkCmdDrawIndexedIndirectCount-pStrides-04913
  If the bound graphics pipeline was created with the VK_DYNAMIC_STATE_VERTEX_INPUT_BINDING_STRIDE_EXT dynamic state enabled, but without the VK_DYNAMIC_STATE_VERTEX_INPUT_EXT dynamic state enabled, then vkCmdBindVertexBuffers2EXT **must** have been called and not subsequently invalidated in the current command buffer prior to this draw command, and the pStrides parameter of vkCmdBindVertexBuffers2EXT **must** not be NULL
- VUID-vkCmdDrawIndexedIndirectCount-None-04914
  If there is a shader object bound to the VK_SHADER_STAGE_VERTEX_BIT stage or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VERTEX_INPUT_EXT dynamic state enabled then vkCmdSetVertexInputEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this draw command
- VUID-vkCmdDrawIndexedIndirectCount-Input-07939
  If there is a shader object bound to the VK_SHADER_STAGE_VERTEX_BIT stage or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VERTEX_INPUT_EXT dynamic state enabled then all variables with the Input storage class decorated with Location in the Vertex Execution Model OpEntryPoint **must** contain a location in VkVertexInputAttributeDescription2EXT::location
- VUID-vkCmdDrawIndexedIndirectCount-Input-08734
  If there is a shader object bound to the VK_SHADER_STAGE_VERTEX_BIT stage or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VERTEX_INPUT_EXT dynamic state enabled and either legacyVertexAttributes is not enabled or the SPIR-V Type associated with a given Input variable of the corresponding Location in the Vertex Execution Model OpEntryPoint is 64-bit, then the numeric type associated with all Input variables of the corresponding Location in the Vertex Execution Model OpEntryPoint **must** be the same as VkVertexInputAttributeDescription2EXT::format
- VUID-vkCmdDrawIndexedIndirectCount-format-08936
  If there is a shader object bound to the VK_SHADER_STAGE_VERTEX_BIT stage or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VERTEX_INPUT_EXT dynamic state enabled and VkVertexInputAttributeDescription2EXT::format has a 64-bit component, then the scalar width associated with all Input variables of the corresponding Location in the Vertex Execution Model OpEntryPoint **must** be 64-bit
- VUID-vkCmdDrawIndexedIndirectCount-format-08937
  If there is a shader object bound to the VK_SHADER_STAGE_VERTEX_BIT stage or the bound

graphics pipeline state was created with the VK_DYNAMIC_STATE_VERTEX_INPUT_EXT dynamic state enabled and the scalar width associated with a Location decorated Input variable in the Vertex Execution Model OpEntryPoint is 64-bit, then the corresponding VkVertexInputAttributeDescription2EXT::format **must** have a 64-bit component

- VUID-vkCmdDrawIndexedIndirectCount-None-09203
  If there is a shader object bound to the VK_SHADER_STAGE_VERTEX_BIT stage or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VERTEX_INPUT_EXT dynamic state enabled and VkVertexInputAttributeDescription2EXT::format has a 64-bit component, then all Input variables at the corresponding Location in the Vertex Execution Model OpEntryPoint **must** not use components that are not present in the format

- VUID-vkCmdDrawIndexedIndirectCount-None-04875
  If there is a shader object bound to the VK_SHADER_STAGE_VERTEX_BIT stage and the most recent call to vkCmdSetPrimitiveTopologyEXT in the current command buffer set primitiveTopology to VK_PRIMITIVE_TOPOLOGY_PATCH_LIST, or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_PATCH_CONTROL_POINTS_EXT dynamic state enabled then vkCmdSetPatchControlPointsEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirectCount-None-04879
  If there is a shader object bound to the VK_SHADER_STAGE_VERTEX_BIT stage or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_PRIMITIVE_RESTART_ENABLE dynamic state enabled then vkCmdSetPrimitiveRestartEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirectCount-None-09637
  If the primitiveTopologyListRestart feature is not enabled, the topology is VK_PRIMITIVE_TOPOLOGY_POINT_LIST, VK_PRIMITIVE_TOPOLOGY_LINE_LIST, VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST, VK_PRIMITIVE_TOPOLOGY_LINE_LIST_WITH_ADJACENCY, or VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST_WITH_ADJACENCY, there is a shader object bound to the VK_SHADER_STAGE_VERTEX_BIT stage or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_PRIMITIVE_RESTART_ENABLE dynamic state enabled then vkCmdSetPrimitiveRestartEnableEXT **must** be VK_FALSE

- VUID-vkCmdDrawIndexedIndirectCount-stage-06481
  The bound graphics pipeline **must** not have been created with the VkPipelineShaderStageCreateInfo::stage member of an element of VkGraphicsPipelineCreateInfo::pStages set to VK_SHADER_STAGE_TASK_BIT_EXT or VK_SHADER_STAGE_MESH_BIT_EXT

- VUID-vkCmdDrawIndexedIndirectCount-None-08885
  There **must** be no shader object bound to either of the VK_SHADER_STAGE_TASK_BIT_EXT or VK_SHADER_STAGE_MESH_BIT_EXT stages

- VUID-vkCmdDrawIndexedIndirectCount-buffer-02708
  If buffer is non-sparse then it **must** be bound completely and contiguously to a single VkDeviceMemory object

- VUID-vkCmdDrawIndexedIndirectCount-buffer-02709
  buffer **must** have been created with the VK_BUFFER_USAGE_INDIRECT_BUFFER_BIT bit set

- VUID-vkCmdDrawIndexedIndirectCount-offset-02710
  `offset` **must** be a multiple of `4`
- VUID-vkCmdDrawIndexedIndirectCount-commandBuffer-02711
  `commandBuffer` **must** not be a protected command buffer
- VUID-vkCmdDrawIndexedIndirectCount-countBuffer-02714
  If `countBuffer` is non-sparse then it **must** be bound completely and contiguously to a single `VkDeviceMemory` object
- VUID-vkCmdDrawIndexedIndirectCount-countBuffer-02715
  `countBuffer` **must** have been created with the `VK_BUFFER_USAGE_INDIRECT_BUFFER_BIT` bit set
- VUID-vkCmdDrawIndexedIndirectCount-countBufferOffset-02716
  `countBufferOffset` **must** be a multiple of `4`
- VUID-vkCmdDrawIndexedIndirectCount-countBuffer-02717
  The count stored in `countBuffer` **must** be less than or equal to `VkPhysicalDeviceLimits::maxDrawIndirectCount`
- VUID-vkCmdDrawIndexedIndirectCount-countBufferOffset-04129
  (`countBufferOffset` + `sizeof`(uint32_t)) **must** be less than or equal to the size of `countBuffer`
- VUID-vkCmdDrawIndexedIndirectCount-None-07312
  If `maintenance6` is not enabled, a valid index buffer **must** be bound
- VUID-vkCmdDrawIndexedIndirectCount-stride-03142
  `stride` **must** be a multiple of `4` and **must** be greater than or equal to sizeof(`VkDrawIndexedIndirectCommand`)
- VUID-vkCmdDrawIndexedIndirectCount-maxDrawCount-03143
  If `maxDrawCount` is greater than or equal to `1`, (`stride` × (`maxDrawCount` - 1) + `offset` + sizeof(`VkDrawIndexedIndirectCommand`)) **must** be less than or equal to the size of `buffer`
- VUID-vkCmdDrawIndexedIndirectCount-countBuffer-03153
  If count stored in `countBuffer` is equal to `1`, (`offset` + sizeof(`VkDrawIndexedIndirectCommand`)) **must** be less than or equal to the size of `buffer`
- VUID-vkCmdDrawIndexedIndirectCount-countBuffer-03154
  If count stored in `countBuffer` is greater than `1`, (`stride` × (`drawCount` - 1) + `offset` + sizeof(`VkDrawIndexedIndirectCommand`)) **must** be less than or equal to the size of `buffer`

## Valid Usage (Implicit)

- VUID-vkCmdDrawIndexedIndirectCount-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdDrawIndexedIndirectCount-buffer-parameter
  `buffer` **must** be a valid VkBuffer handle
- VUID-vkCmdDrawIndexedIndirectCount-countBuffer-parameter
  `countBuffer` **must** be a valid VkBuffer handle
- VUID-vkCmdDrawIndexedIndirectCount-commandBuffer-recording

commandBuffer **must** be in the recording state

- VUID-vkCmdDrawIndexedIndirectCount-commandBuffer-cmdpool
  The VkCommandPool that commandBuffer was allocated from **must** support graphics operations
- VUID-vkCmdDrawIndexedIndirectCount-renderpass
  This command **must** only be called inside of a render pass instance
- VUID-vkCmdDrawIndexedIndirectCount-videocoding
  This command **must** only be called outside of a video coding scope
- VUID-vkCmdDrawIndexedIndirectCount-commonparent
  Each of buffer, commandBuffer, and countBuffer **must** have been created, allocated, or retrieved from the same VkDevice

### Host Synchronization

- Host access to commandBuffer **must** be externally synchronized
- Host access to the VkCommandPool that commandBuffer was allocated from **must** be externally synchronized

### Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Inside | Outside | Graphics | Action |

## 21.3.1. Drawing Transform Feedback

It is possible to draw vertex data that was previously captured during active transform feedback by binding one or more of the transform feedback buffers as vertex buffers. A pipeline barrier is required between using the buffers as transform feedback buffers and vertex buffers to ensure all writes to the transform feedback buffers are visible when the data is read as vertex attributes. The source access is VK_ACCESS_TRANSFORM_FEEDBACK_WRITE_BIT_EXT and the destination access is VK_ACCESS_VERTEX_ATTRIBUTE_READ_BIT for the pipeline stages VK_PIPELINE_STAGE_TRANSFORM_FEEDBACK_BIT_EXT and VK_PIPELINE_STAGE_VERTEX_INPUT_BIT respectively. The value written to the counter buffer by vkCmdEndTransformFeedbackEXT **can** be used to determine the vertex count for the draw. A pipeline barrier is required between using the counter buffer for vkCmdEndTransformFeedbackEXT and vkCmdDrawIndirectByteCountEXT where the source access is VK_ACCESS_TRANSFORM_FEEDBACK_COUNTER_WRITE_BIT_EXT and the destination access is VK_ACCESS_INDIRECT_COMMAND_READ_BIT for the pipeline stages VK_PIPELINE_STAGE_TRANSFORM_FEEDBACK_BIT_EXT and VK_PIPELINE_STAGE_DRAW_INDIRECT_BIT respectively.

To record a non-indexed draw call, where the vertex count is based on a byte count read from a buffer and the passed in vertex stride parameter, call:

```
// Provided by VK_EXT_transform_feedback
void vkCmdDrawIndirectByteCountEXT(
    VkCommandBuffer                             commandBuffer,
    uint32_t                                    instanceCount,
    uint32_t                                    firstInstance,
    VkBuffer                                    counterBuffer,
    VkDeviceSize                                counterBufferOffset,
    uint32_t                                    counterOffset,
    uint32_t                                    vertexStride);
```

- `commandBuffer` is the command buffer into which the command is recorded.
- `instanceCount` is the number of instances to draw.
- `firstInstance` is the instance ID of the first instance to draw.
- `counterBuffer` is the buffer handle from where the byte count is read.
- `counterBufferOffset` is the offset into the buffer used to read the byte count, which is used to calculate the vertex count for this draw call.
- `counterOffset` is subtracted from the byte count read from the `counterBuffer` at the `counterBufferOffset`
- `vertexStride` is the stride in bytes between each element of the vertex data that is used to calculate the vertex count from the counter value. This value is typically the same value that was used in the graphics pipeline state when the transform feedback was captured as the `XfbStride`.

When the command is executed, primitives are assembled in the same way as done with vkCmdDraw except the `vertexCount` is calculated based on the byte count read from `counterBuffer` at offset `counterBufferOffset`. The assembled primitives execute the bound graphics pipeline.

The effective `vertexCount` is calculated as follows:

```
const uint32_t * counterBufferPtr = (const uint8_t *)counterBuffer.address +
counterBufferOffset;
vertexCount = floor(max(0, (*counterBufferPtr - counterOffset)) / vertexStride);
```

The effective `firstVertex` is zero.

### Valid Usage

- VUID-vkCmdDrawIndirectByteCountEXT-magFilter-04553
  If a VkSampler created with `magFilter` or `minFilter` equal to `VK_FILTER_LINEAR`, `reductionMode` equal to `VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE`, and `compareEnable` equal to `VK_FALSE` is used to sample a VkImageView as a result of this command, then the

image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT

- VUID-vkCmdDrawIndirectByteCountEXT-magFilter-09598
  If a VkSampler created with magFilter or minFilter equal to VK_FILTER_LINEAR and reductionMode equal to either VK_SAMPLER_REDUCTION_MODE_MIN or VK_SAMPLER_REDUCTION_MODE_MAX is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT
- VUID-vkCmdDrawIndirectByteCountEXT-mipmapMode-04770
  If a VkSampler created with mipmapMode equal to VK_SAMPLER_MIPMAP_MODE_LINEAR, reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE, and compareEnable equal to VK_FALSE is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT
- VUID-vkCmdDrawIndirectByteCountEXT-mipmapMode-09599
  If a VkSampler created with mipmapMode equal to VK_SAMPLER_MIPMAP_MODE_LINEAR and reductionMode equal to either VK_SAMPLER_REDUCTION_MODE_MIN or VK_SAMPLER_REDUCTION_MODE_MAX is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT
- VUID-vkCmdDrawIndirectByteCountEXT-unnormalizedCoordinates-09635
  If a VkSampler created with unnormalizedCoordinates equal to VK_TRUE is used to sample a VkImageView as a result of this command, then the image view's levelCount and layerCount **must** be 1
- VUID-vkCmdDrawIndirectByteCountEXT-unnormalizedCoordinates-09636
  If a VkSampler created with unnormalizedCoordinates equal to VK_TRUE is used to sample a VkImageView as a result of this command, then the image view's viewType **must** be VK_IMAGE_VIEW_TYPE_1D or VK_IMAGE_VIEW_TYPE_2D
- VUID-vkCmdDrawIndirectByteCountEXT-None-06479
  If a VkImageView is sampled with depth comparison, the image view's format features **must** contain VK_FORMAT_FEATURE_2_SAMPLED_IMAGE_DEPTH_COMPARISON_BIT
- VUID-vkCmdDrawIndirectByteCountEXT-None-02691
  If a VkImageView is accessed using atomic operations as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_STORAGE_IMAGE_ATOMIC_BIT
- VUID-vkCmdDrawIndirectByteCountEXT-None-07888
  If a VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER descriptor is accessed using atomic operations as a result of this command, then the storage texel buffer's format features **must** contain VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_ATOMIC_BIT
- VUID-vkCmdDrawIndirectByteCountEXT-None-02692
  If a VkImageView is sampled with VK_FILTER_CUBIC_EXT as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_CUBIC_BIT_EXT
- VUID-vkCmdDrawIndirectByteCountEXT-None-02693

If the VK_EXT_filter_cubic extension is not enabled and any VkImageView is sampled with VK_FILTER_CUBIC_EXT as a result of this command, it **must** not have a VkImageViewType of VK_IMAGE_VIEW_TYPE_3D, VK_IMAGE_VIEW_TYPE_CUBE, or VK_IMAGE_VIEW_TYPE_CUBE_ARRAY

- VUID-vkCmdDrawIndirectByteCountEXT-filterCubic-02694
  Any VkImageView being sampled with VK_FILTER_CUBIC_EXT as a result of this command **must** have a VkImageViewType and format that supports cubic filtering, as specified by VkFilterCubicImageViewImageFormatPropertiesEXT::filterCubic returned by vkGetPhysicalDeviceImageFormatProperties2

- VUID-vkCmdDrawIndirectByteCountEXT-filterCubicMinmax-02695
  Any VkImageView being sampled with VK_FILTER_CUBIC_EXT with a reduction mode of either VK_SAMPLER_REDUCTION_MODE_MIN or VK_SAMPLER_REDUCTION_MODE_MAX as a result of this command **must** have a VkImageViewType and format that supports cubic filtering together with minmax filtering, as specified by VkFilterCubicImageViewImageFormatPropertiesEXT::filterCubicMinmax returned by vkGetPhysicalDeviceImageFormatProperties2

- VUID-vkCmdDrawIndirectByteCountEXT-cubicRangeClamp-09212
  If the cubicRangeClamp feature is not enabled, then any VkImageView being sampled with VK_FILTER_CUBIC_EXT as a result of this command **must** not have a VkSamplerReductionModeCreateInfo::reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE_RANGECLAMP_QCOM

- VUID-vkCmdDrawIndirectByteCountEXT-reductionMode-09213
  Any VkImageView being sampled with a VkSamplerReductionModeCreateInfo::reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE_RANGECLAMP_QCOM as a result of this command **must** sample with VK_FILTER_CUBIC_EXT

- VUID-vkCmdDrawIndirectByteCountEXT-selectableCubicWeights-09214
  If the selectableCubicWeights feature is not enabled, then any VkImageView being sampled with VK_FILTER_CUBIC_EXT as a result of this command **must** have VkSamplerCubicWeightsCreateInfoQCOM::cubicWeights equal to VK_CUBIC_FILTER_WEIGHTS_CATMULL_ROM_QCOM

- VUID-vkCmdDrawIndirectByteCountEXT-flags-02696
  Any VkImage created with a VkImageCreateInfo::flags containing VK_IMAGE_CREATE_CORNER_SAMPLED_BIT_NV sampled as a result of this command **must** only be sampled using a VkSamplerAddressMode of VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_EDGE

- VUID-vkCmdDrawIndirectByteCountEXT-OpTypeImage-07027
  For any VkImageView being written as a storage image where the image format field of the OpTypeImage is Unknown, the view's format features **must** contain VK_FORMAT_FEATURE_2_STORAGE_WRITE_WITHOUT_FORMAT_BIT

- VUID-vkCmdDrawIndirectByteCountEXT-OpTypeImage-07028
  For any VkImageView being read as a storage image where the image format field of the OpTypeImage is Unknown, the view's format features **must** contain VK_FORMAT_FEATURE_2_STORAGE_READ_WITHOUT_FORMAT_BIT

- VUID-vkCmdDrawIndirectByteCountEXT-OpTypeImage-07029
  For any VkBufferView being written as a storage texel buffer where the image format field of the OpTypeImage is Unknown, the view's buffer features **must** contain

VK_FORMAT_FEATURE_2_STORAGE_WRITE_WITHOUT_FORMAT_BIT

- VUID-vkCmdDrawIndirectByteCountEXT-OpTypeImage-07030
  Any VkBufferView being read as a storage texel buffer where the image format field of the OpTypeImage is Unknown then the view's buffer features **must** contain VK_FORMAT_FEATURE_2_STORAGE_READ_WITHOUT_FORMAT_BIT
- VUID-vkCmdDrawIndirectByteCountEXT-None-08600
  For each set *n* that is statically used by a bound shader, a descriptor set **must** have been bound to *n* at the same pipeline bind point, with a VkPipelineLayout that is compatible for set *n*, with the VkPipelineLayout used to create the current VkPipeline or the VkDescriptorSetLayout array used to create the current VkShaderEXT , as described in Pipeline Layout Compatibility
- VUID-vkCmdDrawIndirectByteCountEXT-None-08601
  For each push constant that is statically used by a bound shader, a push constant value **must** have been set for the same pipeline bind point, with a VkPipelineLayout that is compatible for push constants, with the VkPipelineLayout used to create the current VkPipeline or the VkDescriptorSetLayout array used to create the current VkShaderEXT , as described in Pipeline Layout Compatibility
- VUID-vkCmdDrawIndirectByteCountEXT-None-10068
  For each array of resources that is used by a bound shader, the indices used to access members of the array **must** be less than the descriptor count for the identified binding in the descriptor sets used by this command
- VUID-vkCmdDrawIndirectByteCountEXT-maintenance4-08602
  If the maintenance4 feature is not enabled, then for each push constant that is statically used by a bound shader, a push constant value **must** have been set for the same pipeline bind point, with a VkPipelineLayout that is compatible for push constants, with the VkPipelineLayout used to create the current VkPipeline or the VkDescriptorSetLayout and VkPushConstantRange arrays used to create the current VkShaderEXT , as described in Pipeline Layout Compatibility
- VUID-vkCmdDrawIndirectByteCountEXT-None-08114
  Descriptors in each bound descriptor set, specified via vkCmdBindDescriptorSets, **must** be valid as described by descriptor validity if they are statically used by the VkPipeline bound to the pipeline bind point used by this command and the bound VkPipeline was not created with VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT
- VUID-vkCmdDrawIndirectByteCountEXT-None-08115
  If the descriptors used by the VkPipeline bound to the pipeline bind point were specified via vkCmdBindDescriptorSets, the bound VkPipeline **must** have been created without VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT
- VUID-vkCmdDrawIndirectByteCountEXT-None-08116
  Descriptors in bound descriptor buffers, specified via vkCmdSetDescriptorBufferOffsetsEXT, **must** be valid if they are dynamically used by the VkPipeline bound to the pipeline bind point used by this command and the bound VkPipeline was created with VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT
- VUID-vkCmdDrawIndirectByteCountEXT-None-08604
  Descriptors in bound descriptor buffers, specified via

vkCmdSetDescriptorBufferOffsetsEXT, **must** be valid if they are dynamically used by any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command

- VUID-vkCmdDrawIndirectByteCountEXT-None-08117
  If the descriptors used by the VkPipeline bound to the pipeline bind point were specified via vkCmdSetDescriptorBufferOffsetsEXT, the bound VkPipeline **must** have been created with `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`
- VUID-vkCmdDrawIndirectByteCountEXT-None-08119
  If a descriptor is dynamically used with a VkPipeline created with `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`, the descriptor memory **must** be resident
- VUID-vkCmdDrawIndirectByteCountEXT-None-08605
  If a descriptor is dynamically used with a VkShaderEXT created with a `VkDescriptorSetLayout` that was created with `VK_DESCRIPTOR_SET_LAYOUT_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`, the descriptor memory **must** be resident
- VUID-vkCmdDrawIndirectByteCountEXT-None-08606
  If the `shaderObject` feature is not enabled, a valid pipeline **must** be bound to the pipeline bind point used by this command
- VUID-vkCmdDrawIndirectByteCountEXT-None-08608
  If a pipeline is bound to the pipeline bind point used by this command, there **must** not have been any calls to dynamic state setting commands for any state specified statically in the VkPipeline object bound to the pipeline bind point used by this command, since that pipeline was bound
- VUID-vkCmdDrawIndirectByteCountEXT-None-08609
  If the VkPipeline object bound to the pipeline bind point used by this command or any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a VkSampler object that uses unnormalized coordinates, that sampler **must** not be used to sample from any VkImage with a VkImageView of the type `VK_IMAGE_VIEW_TYPE_3D`, `VK_IMAGE_VIEW_TYPE_CUBE`, `VK_IMAGE_VIEW_TYPE_1D_ARRAY`, `VK_IMAGE_VIEW_TYPE_2D_ARRAY` or `VK_IMAGE_VIEW_TYPE_CUBE_ARRAY`, in any shader stage
- VUID-vkCmdDrawIndirectByteCountEXT-None-08610
  If the VkPipeline object bound to the pipeline bind point used by this command or any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a VkSampler object that uses unnormalized coordinates, that sampler **must** not be used with any of the SPIR-V `OpImageSample*` or `OpImageSparseSample*` instructions with `ImplicitLod`, `Dref` or `Proj` in their name, in any shader stage
- VUID-vkCmdDrawIndirectByteCountEXT-None-08611
  If the VkPipeline object bound to the pipeline bind point used by this command or any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a VkSampler object that uses unnormalized coordinates, that sampler **must** not be used with any of the SPIR-V `OpImageSample*` or `OpImageSparseSample*` instructions that includes a LOD bias or any offset values, in any shader stage
- VUID-vkCmdDrawIndirectByteCountEXT-None-08607
  If the `shaderObject` is enabled, either a valid pipeline **must** be bound to the pipeline bind

point used by this command, or a valid combination of valid and VK_NULL_HANDLE shader objects **must** be bound to every supported shader stage corresponding to the pipeline bind point used by this command

- VUID-vkCmdDrawIndirectByteCountEXT-uniformBuffers-06935
  If any stage of the VkPipeline object bound to the pipeline bind point used by this command accesses a uniform buffer, and that stage was created without enabling either `VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_EXT` or `VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_2_EXT` for `uniformBuffers`, and the `robustBufferAccess` feature is not enabled, that stage **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point
- VUID-vkCmdDrawIndirectByteCountEXT-None-08612
  If the `robustBufferAccess` feature is not enabled, and any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a uniform buffer, it **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point
- VUID-vkCmdDrawIndirectByteCountEXT-storageBuffers-06936
  If any stage of the VkPipeline object bound to the pipeline bind point used by this command accesses a storage buffer, and that stage was created without enabling either `VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_EXT` or `VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_2_EXT` for `storageBuffers`, and the `robustBufferAccess` feature is not enabled, that stage **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point
- VUID-vkCmdDrawIndirectByteCountEXT-None-08613
  If the `robustBufferAccess` feature is not enabled, and any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a storage buffer, it **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point
- VUID-vkCmdDrawIndirectByteCountEXT-commandBuffer-02707
  If `commandBuffer` is an unprotected command buffer and `protectedNoFault` is not supported, any resource accessed by bound shaders **must** not be a protected resource
- VUID-vkCmdDrawIndirectByteCountEXT-None-06550
  If a bound shader accesses a VkSampler or VkImageView object that enables sampler $Y'C_BC_R$ conversion, that object **must** only be used with `OpImageSample*` or `OpImageSparseSample*` instructions
- VUID-vkCmdDrawIndirectByteCountEXT-ConstOffset-06551
  If a bound shader accesses a VkSampler or VkImageView object that enables sampler $Y'C_BC_R$ conversion, that object **must** not use the `ConstOffset` and `Offset` operands
- VUID-vkCmdDrawIndirectByteCountEXT-viewType-07752
  If a VkImageView is accessed as a result of this command, then the image view's `viewType` **must** match the `Dim` operand of the `OpTypeImage` as described in Compatibility Between SPIR-V Image Dimensions and Vulkan ImageView Types
- VUID-vkCmdDrawIndirectByteCountEXT-format-07753

If a VkImageView is accessed as a result of this command, then the numeric type of the image view's format and the Sampled Type operand of the OpTypeImage **must** match

- VUID-vkCmdDrawIndirectByteCountEXT-OpImageWrite-08795
  If a VkImageView created with a format other than VK_FORMAT_A8_UNORM_KHR is accessed using OpImageWrite as a result of this command, then the Type of the Texel operand of that instruction **must** have at least as many components as the image view's format
- VUID-vkCmdDrawIndirectByteCountEXT-OpImageWrite-08796
  If a VkImageView created with the format VK_FORMAT_A8_UNORM_KHR is accessed using OpImageWrite as a result of this command, then the Type of the Texel operand of that instruction **must** have four components
- VUID-vkCmdDrawIndirectByteCountEXT-OpImageWrite-04469
  If a VkBufferView is accessed using OpImageWrite as a result of this command, then the Type of the Texel operand of that instruction **must** have at least as many components as the buffer view's format
- VUID-vkCmdDrawIndirectByteCountEXT-SampledType-04470
  If a VkImageView with a VkFormat that has a 64-bit component width is accessed as a result of this command, the SampledType of the OpTypeImage operand of that instruction **must** have a Width of 64
- VUID-vkCmdDrawIndirectByteCountEXT-SampledType-04471
  If a VkImageView with a VkFormat that has a component width less than 64-bit is accessed as a result of this command, the SampledType of the OpTypeImage operand of that instruction **must** have a Width of 32
- VUID-vkCmdDrawIndirectByteCountEXT-SampledType-04472
  If a VkBufferView with a VkFormat that has a 64-bit component width is accessed as a result of this command, the SampledType of the OpTypeImage operand of that instruction **must** have a Width of 64
- VUID-vkCmdDrawIndirectByteCountEXT-SampledType-04473
  If a VkBufferView with a VkFormat that has a component width less than 64-bit is accessed as a result of this command, the SampledType of the OpTypeImage operand of that instruction **must** have a Width of 32
- VUID-vkCmdDrawIndirectByteCountEXT-sparseImageInt64Atomics-04474
  If the sparseImageInt64Atomics feature is not enabled, VkImage objects created with the VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT flag **must** not be accessed by atomic instructions through an OpTypeImage with a SampledType with a Width of 64 by this command
- VUID-vkCmdDrawIndirectByteCountEXT-sparseImageInt64Atomics-04475
  If the sparseImageInt64Atomics feature is not enabled, VkBuffer objects created with the VK_BUFFER_CREATE_SPARSE_RESIDENCY_BIT flag **must** not be accessed by atomic instructions through an OpTypeImage with a SampledType with a Width of 64 by this command
- VUID-vkCmdDrawIndirectByteCountEXT-OpImageWeightedSampleQCOM-06971
  If OpImageWeightedSampleQCOM is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_WEIGHT_SAMPLED_IMAGE_BIT_QCOM
- VUID-vkCmdDrawIndirectByteCountEXT-OpImageWeightedSampleQCOM-06972

If OpImageWeightedSampleQCOM uses a VkImageView as a sample weight image as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_WEIGHT_IMAGE_BIT_QCOM

- VUID-vkCmdDrawIndirectByteCountEXT-OpImageBoxFilterQCOM-06973
  If OpImageBoxFilterQCOM is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_BOX_FILTER_SAMPLED_BIT_QCOM
- VUID-vkCmdDrawIndirectByteCountEXT-OpImageBlockMatchSSDQCOM-06974
  If OpImageBlockMatchSSDQCOM is used to read from an VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM
- VUID-vkCmdDrawIndirectByteCountEXT-OpImageBlockMatchSADQCOM-06975
  If OpImageBlockMatchSADQCOM is used to read from an VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM
- VUID-vkCmdDrawIndirectByteCountEXT-OpImageBlockMatchSADQCOM-06976
  If OpImageBlockMatchSADQCOM or OpImageBlockMatchSSDQCOM is used to read from a reference image as result of this command, then the specified reference coordinates **must** not fail integer texel coordinate validation
- VUID-vkCmdDrawIndirectByteCountEXT-OpImageWeightedSampleQCOM-06977
  If OpImageWeightedSampleQCOM, OpImageBoxFilterQCOM, OpImageBlockMatchWindowSSDQCOM, OpImageBlockMatchWindowSADQCOM, OpImageBlockMatchGatherSSDQCOM, OpImageBlockMatchGatherSADQCOM, OpImageBlockMatchSSDQCOM, or OpImageBlockMatchSADQCOM uses a VkSampler as a result of this command, then the sampler **must** have been created with VK_SAMPLER_CREATE_IMAGE_PROCESSING_BIT_QCOM
- VUID-vkCmdDrawIndirectByteCountEXT-OpImageWeightedSampleQCOM-06978
  If any command other than OpImageWeightedSampleQCOM, OpImageBoxFilterQCOM, OpImageBlockMatchWindowSSDQCOM, OpImageBlockMatchWindowSADQCOM, OpImageBlockMatchGatherSSDQCOM, OpImageBlockMatchGatherSADQCOM, OpImageBlockMatchSSDQCOM, or OpImageBlockMatchSADQCOM uses a VkSampler as a result of this command, then the sampler **must** not have been created with VK_SAMPLER_CREATE_IMAGE_PROCESSING_BIT_QCOM
- VUID-vkCmdDrawIndirectByteCountEXT-OpImageBlockMatchWindow-09215
  If a OpImageBlockMatchWindow*QCOM or OpImageBlockMatchGather*QCOM instruction is used to read from an VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM
- VUID-vkCmdDrawIndirectByteCountEXT-OpImageBlockMatchWindow-09216
  If a OpImageBlockMatchWindow*QCOM or OpImageBlockMatchGather*QCOM instruction is used to read from an VkImageView as a result of this command, then the image view's format **must** be a single-component format
- VUID-vkCmdDrawIndirectByteCountEXT-OpImageBlockMatchWindow-09217
  If a OpImageBlockMatchWindow*QCOM or OpImageBlockMatchGather*QCOM read from a reference image as result of this command, then the specified reference coordinates **must** not fail integer texel coordinate validation

- VUID-vkCmdDrawIndirectByteCountEXT-None-07288
  Any shader invocation executed by this command **must** terminate
- VUID-vkCmdDrawIndirectByteCountEXT-None-09600
  If a descriptor with type equal to any of `VK_DESCRIPTOR_TYPE_SAMPLE_WEIGHT_IMAGE_QCOM`, `VK_DESCRIPTOR_TYPE_BLOCK_MATCH_IMAGE_QCOM`, `VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE`, `VK_DESCRIPTOR_TYPE_STORAGE_IMAGE`, or `VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT` is accessed as a result of this command, the image subresource identified by that descriptor **must** be in the image layout identified when the descriptor was written
- VUID-vkCmdDrawIndirectByteCountEXT-renderPass-02684
  The current render pass **must** be compatible with the `renderPass` member of the `VkGraphicsPipelineCreateInfo` structure specified when creating the `VkPipeline` bound to `VK_PIPELINE_BIND_POINT_GRAPHICS`
- VUID-vkCmdDrawIndirectByteCountEXT-subpass-02685
  The subpass index of the current render pass **must** be equal to the `subpass` member of the `VkGraphicsPipelineCreateInfo` structure specified when creating the `VkPipeline` bound to `VK_PIPELINE_BIND_POINT_GRAPHICS`
- VUID-vkCmdDrawIndirectByteCountEXT-None-07748
  If any shader statically accesses an input attachment, a valid descriptor **must** be bound to the pipeline via a descriptor set
- VUID-vkCmdDrawIndirectByteCountEXT-OpTypeImage-07468
  If any shader executed by this pipeline accesses an `OpTypeImage` variable with a `Dim` operand of `SubpassData`, it **must** be decorated with an `InputAttachmentIndex` that corresponds to a valid input attachment in the current subpass
- VUID-vkCmdDrawIndirectByteCountEXT-None-07469
  Input attachment views accessed in a subpass **must** be created with the same VkFormat as the corresponding subpass definition, and be created with a VkImageView that is compatible with the attachment referenced by the subpass' `pInputAttachments` [`InputAttachmentIndex`] in the currently bound VkFramebuffer as specified by Fragment Input Attachment Compatibility
- VUID-vkCmdDrawIndirectByteCountEXT-pDepthInputAttachmentIndex-09595
  Input attachment views accessed in a dynamic render pass with a `InputAttachmentIndex` referenced by VkRenderingInputAttachmentIndexInfoKHR, or no `InputAttachmentIndex` if VkRenderingInputAttachmentIndexInfoKHR:`pDepthInputAttachmentIndex` or VkRenderingInputAttachmentIndexInfoKHR:`pStencilInputAttachmentIndex` are `NULL`, **must** be created with a VkImageView that is compatible with the corresponding color, depth, or stencil attachment in VkRenderingInfo
- VUID-vkCmdDrawIndirectByteCountEXT-pDepthInputAttachmentIndex-09596
  Input attachment views accessed in a dynamic render pass via a shader object **must** have an `InputAttachmentIndex` if both VkRenderingInputAttachmentIndexInfoKHR:`pDepthInputAttachmentIndex` and VkRenderingInputAttachmentIndexInfoKHR:`pStencilInputAttachmentIndex` are non-`NULL`
- VUID-vkCmdDrawIndirectByteCountEXT-InputAttachmentIndex-09597
  If an input attachment view accessed in a dynamic render pass via a shader object has an `InputAttachmentIndex`, the `InputAttachmentIndex` **must** match an index in

VkRenderingInputAttachmentIndexInfoKHR

- VUID-vkCmdDrawIndirectByteCountEXT-None-06537
  Memory backing image subresources used as attachments in the current render pass **must** not be written in any way other than as an attachment by this command
- VUID-vkCmdDrawIndirectByteCountEXT-None-09000
  If a color attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass, it is not in the `VK_IMAGE_LAYOUT_ATTACHMENT_FEEDBACK_LOOP_OPTIMAL_EXT` image layout, and either:
  - the `VK_PIPELINE_CREATE_COLOR_ATTACHMENT_FEEDBACK_LOOP_BIT_EXT` is set on the currently bound pipeline or
  - the last call to vkCmdSetAttachmentFeedbackLoopEnableEXT included `VK_IMAGE_ASPECT_COLOR_BIT` and
    - there is no currently bound graphics pipeline or
    - the currently bound graphics pipeline was created with `VK_DYNAMIC_STATE_ATTACHMENT_FEEDBACK_LOOP_ENABLE_EXT`

  it **must** not be accessed in any way other than as an attachment by this command
- VUID-vkCmdDrawIndirectByteCountEXT-None-09001
  If a depth attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass, it is not in the `VK_IMAGE_LAYOUT_ATTACHMENT_FEEDBACK_LOOP_OPTIMAL_EXT` image layout, and either:
  - the `VK_PIPELINE_CREATE_DEPTH_STENCIL_ATTACHMENT_FEEDBACK_LOOP_BIT_EXT` is set on the currently bound pipeline or
  - the last call to vkCmdSetAttachmentFeedbackLoopEnableEXT included `VK_IMAGE_ASPECT_DEPTH_BIT` and
    - there is no currently bound graphics pipeline or
    - the currently bound graphics pipeline was created with `VK_DYNAMIC_STATE_ATTACHMENT_FEEDBACK_LOOP_ENABLE_EXT`

  it **must** not be accessed in any way other than as an attachment by this command
- VUID-vkCmdDrawIndirectByteCountEXT-None-09002
  If a stencil attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass, it is not in the `VK_IMAGE_LAYOUT_ATTACHMENT_FEEDBACK_LOOP_OPTIMAL_EXT` image layout, and either:
  - the `VK_PIPELINE_CREATE_DEPTH_STENCIL_ATTACHMENT_FEEDBACK_LOOP_BIT_EXT` is set on the currently bound pipeline or
  - the last call to vkCmdSetAttachmentFeedbackLoopEnableEXT included `VK_IMAGE_ASPECT_STENCIL_BIT` and
    - there is no currently bound graphics pipeline or
    - the currently bound graphics pipeline was created with `VK_DYNAMIC_STATE_ATTACHMENT_FEEDBACK_LOOP_ENABLE_EXT`

it **must** not be accessed in any way other than as an attachment by this command

- VUID-vkCmdDrawIndirectByteCountEXT-None-09003
  If an attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass, it **must** not be accessed in any way other than as an attachment, storage image, or sampled image by this command
- VUID-vkCmdDrawIndirectByteCountEXT-None-06539
  If any previously recorded command in the current subpass accessed an image subresource used as an attachment in this subpass in any way other than as an attachment, this command **must** not write to that image subresource as an attachment
- VUID-vkCmdDrawIndirectByteCountEXT-None-06886
  If the current render pass instance uses a depth/stencil attachment with a read-only layout for the depth aspect, depth writes **must** be disabled
- VUID-vkCmdDrawIndirectByteCountEXT-None-06887
  If the current render pass instance uses a depth/stencil attachment with a read-only layout for the stencil aspect, both front and back `writeMask` are not zero, and stencil test is enabled, all stencil ops **must** be `VK_STENCIL_OP_KEEP`
- VUID-vkCmdDrawIndirectByteCountEXT-None-07831
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VIEWPORT` dynamic state enabled then vkCmdSetViewport **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectByteCountEXT-None-07832
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_SCISSOR` dynamic state enabled then vkCmdSetScissor **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectByteCountEXT-None-07833
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_LINE_WIDTH` dynamic state enabled then vkCmdSetLineWidth **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectByteCountEXT-None-08617
  If a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE`, and the most recent call to vkCmdSetPolygonModeEXT in the current command buffer set `polygonMode` to `VK_POLYGON_MODE_LINE`, vkCmdSetLineWidth **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectByteCountEXT-None-08618
  If a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE`, and the most recent call to vkCmdSetPrimitiveTopologyEXT in the current command buffer set `primitiveTopology` to any line topology, vkCmdSetLineWidth **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectByteCountEXT-None-08619

If a shader object that outputs line primitives is bound to the VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT or VK_SHADER_STAGE_GEOMETRY_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, vkCmdSetLineWidth **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirectByteCountEXT-None-07834
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_BIAS dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of depthBiasEnable is VK_TRUE, then vkCmdSetDepthBounds or vkCmdSetDepthBias2EXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectByteCountEXT-None-07835
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_BLEND_CONSTANTS dynamic state enabled then vkCmdSetBlendConstants **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectByteCountEXT-None-08621
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetColorBlendEnableEXT in the current command buffer set any element of pColorBlendEnables to VK_TRUE, and the most recent call to vkCmdSetColorBlendEquationEXT in the current command buffer set the same element of pColorBlendEquations to a VkColorBlendEquationEXT structure with any VkBlendFactor member with a value of VK_BLEND_FACTOR_CONSTANT_COLOR, VK_BLEND_FACTOR_ONE_MINUS_CONSTANT_COLOR, VK_BLEND_FACTOR_CONSTANT_ALPHA, or VK_BLEND_FACTOR_ONE_MINUS_CONSTANT_ALPHA, vkCmdSetBlendConstants **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectByteCountEXT-None-07836
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_BOUNDS dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of depthBoundsTestEnable is VK_TRUE, then vkCmdSetDepthBounds **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectByteCountEXT-None-07837
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_STENCIL_COMPARE_MASK dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of stencilTestEnable is VK_TRUE, then vkCmdSetStencilCompareMask **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirectByteCountEXT-None-07838
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_STENCIL_WRITE_MASK dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of stencilTestEnable is VK_TRUE, then vkCmdSetStencilWriteMask **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectByteCountEXT-None-07839
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_STENCIL_REFERENCE dynamic state enabled, the current value of and rasterizerDiscardEnable is VK_FALSE, the current value of stencilTestEnable is VK_TRUE, then vkCmdSetStencilReference **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectByteCountEXT-maxMultiviewInstanceIndex-02688
  If the draw is recorded in a render pass instance with multiview enabled, the maximum instance index **must** be less than or equal to VkPhysicalDeviceMultiviewProperties::maxMultiviewInstanceIndex
- VUID-vkCmdDrawIndirectByteCountEXT-sampleLocationsEnable-02689
  If the bound graphics pipeline was created with VkPipelineSampleLocationsStateCreateInfoEXT::sampleLocationsEnable set to VK_TRUE and the current subpass has a depth/stencil attachment, then that attachment **must** have been created with the VK_IMAGE_CREATE_SAMPLE_LOCATIONS_COMPATIBLE_DEPTH_BIT_EXT bit set
- VUID-vkCmdDrawIndirectByteCountEXT-None-06666
  If the VK_EXT_sample_locations extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of sampleLocationsEnable is VK_TRUE, then vkCmdSetSampleLocationsEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectByteCountEXT-None-07840
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_CULL_MODE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetCullModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectByteCountEXT-None-07841
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_FRONT_FACE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetFrontFaceEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectByteCountEXT-None-07843
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_TEST_ENABLE dynamic state enabled, and the

current value of rasterizerDiscardEnable is VK_FALSE, vkCmdSetDepthTestEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirectByteCountEXT-None-07844
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_WRITE_ENABLE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetDepthWriteEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectByteCountEXT-None-07845
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_COMPARE_OP dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of depthTestEnable is VK_TRUE, then vkCmdSetDepthCompareOpEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectByteCountEXT-None-07846
  If the depthBounds feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_BOUNDS_TEST_ENABLE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetDepthBoundsTestEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectByteCountEXT-None-07847
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_STENCIL_TEST_ENABLE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetStencilTestEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectByteCountEXT-None-07848
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_STENCIL_OP dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, the current value of stencilTestEnable is VK_TRUE, then vkCmdSetStencilOpEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectByteCountEXT-viewportCount-03417
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, but not the VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT dynamic state enabled, then vkCmdSetViewportWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the viewportCount parameter of vkCmdSetViewportWithCountEXT **must** match the VkPipelineViewportStateCreateInfo::scissorCount of the pipeline
- VUID-vkCmdDrawIndirectByteCountEXT-scissorCount-03418
  If the bound graphics pipeline state was created with the

VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT dynamic state enabled, but not the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, then vkCmdSetScissorWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the scissorCount parameter of vkCmdSetScissorWithCountEXT **must** match the VkPipelineViewportStateCreateInfo::viewportCount of the pipeline

- VUID-vkCmdDrawIndirectByteCountEXT-viewportCount-03419
  If the bound graphics pipeline state was created with both the VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT and VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic states enabled then both vkCmdSetViewportWithCountEXT and vkCmdSetScissorWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the viewportCount parameter of vkCmdSetViewportWithCountEXT **must** match the scissorCount parameter of vkCmdSetScissorWithCountEXT

- VUID-vkCmdDrawIndirectByteCountEXT-None-08635
  If a shader object is bound to any graphics stage, then both vkCmdSetViewportWithCountEXT and vkCmdSetScissorWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the viewportCount parameter of vkCmdSetViewportWithCountEXT **must** match the scissorCount parameter of vkCmdSetScissorWithCountEXT

- VUID-vkCmdDrawIndirectByteCountEXT-viewportCount-04137
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, but not the VK_DYNAMIC_STATE_VIEWPORT_W_SCALING_NV dynamic state enabled, then the bound graphics pipeline **must** have been created with VkPipelineViewportWScalingStateCreateInfoNV::viewportCount greater or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT

- VUID-vkCmdDrawIndirectByteCountEXT-viewportCount-04138
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT and VK_DYNAMIC_STATE_VIEWPORT_W_SCALING_NV dynamic states enabled then the viewportCount parameter in the last call to vkCmdSetViewportWScalingNV **must** be greater than or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT

- VUID-vkCmdDrawIndirectByteCountEXT-None-09232
  If the VK_NV_clip_space_w_scaling extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetViewportWScalingEnableNV in the current command buffer set viewportWScalingEnable to VK_TRUE, then vkCmdSetViewportWScalingNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirectByteCountEXT-None-08636
  If the VK_NV_clip_space_w_scaling extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetViewportWScalingEnableNV in the current command buffer set viewportWScalingEnable to VK_TRUE, then the viewportCount parameter in the last call to vkCmdSetViewportWScalingNV **must** be greater than or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT

- VUID-vkCmdDrawIndirectByteCountEXT-viewportCount-04139
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT` dynamic state enabled, but not the `VK_DYNAMIC_STATE_VIEWPORT_SHADING_RATE_PALETTE_NV` dynamic state enabled, then the bound graphics pipeline **must** have been created with VkPipelineViewportShadingRateImageStateCreateInfoNV::`viewportCount` greater or equal to the `viewportCount` parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawIndirectByteCountEXT-viewportCount-04140
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT` and `VK_DYNAMIC_STATE_VIEWPORT_SHADING_RATE_PALETTE_NV` dynamic states enabled then the `viewportCount` parameter in the last call to vkCmdSetViewportShadingRatePaletteNV **must** be greater than or equal to the `viewportCount` parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawIndirectByteCountEXT-shadingRateImage-09233
  If the `shadingRateImage` feature is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE`, then vkCmdSetCoarseSampleOrderNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectByteCountEXT-shadingRateImage-09234
  If the `shadingRateImage` feature is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE`, and the most recent call to vkCmdSetShadingRateImageEnableNV in the current command buffer set `shadingRateImageEnable` to `VK_TRUE`, then vkCmdSetViewportShadingRatePaletteNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectByteCountEXT-None-08637
  If the `shadingRateImage` feature is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE`, and the most recent call to vkCmdSetShadingRateImageEnableNV in the current command buffer set `shadingRateImageEnable` to `VK_TRUE`, then the `viewportCount` parameter in the last call to vkCmdSetViewportShadingRatePaletteNV **must** be greater than or equal to the `viewportCount` parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawIndirectByteCountEXT-VkPipelineVieportCreateInfo-04141
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT` dynamic state enabled and a VkPipelineViewportSwizzleStateCreateInfoNV structure chained from VkPipelineViewportStateCreateInfo, then the bound graphics pipeline **must** have been created with VkPipelineViewportSwizzleStateCreateInfoNV::`viewportCount` greater or equal to the `viewportCount` parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawIndirectByteCountEXT-VkPipelineVieportCreateInfo-04142
  If the bound graphics pipeline state was created with the

VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled and a VkPipelineViewportExclusiveScissorStateCreateInfoNV structure chained from VkPipelineViewportStateCreateInfo, then the bound graphics pipeline **must** have been created with VkPipelineViewportExclusiveScissorStateCreateInfoNV ::exclusiveScissorCount greater or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT

- VUID-vkCmdDrawIndirectByteCountEXT-None-07878
  If the exclusiveScissor feature is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_EXCLUSIVE_SCISSOR_ENABLE_NV dynamic state enabled, then vkCmdSetExclusiveScissorEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectByteCountEXT-None-07879
  If the exclusiveScissor feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_EXCLUSIVE_SCISSOR_NV dynamic state enabled, and the most recent call to vkCmdSetExclusiveScissorEnableNV in the current command buffer set any element of pExclusiveScissorEnables to VK_TRUE, then vkCmdSetExclusiveScissorNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectByteCountEXT-None-04876
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_RASTERIZER_DISCARD_ENABLE dynamic state enabled, then vkCmdSetRasterizerDiscardEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectByteCountEXT-None-04877
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_BIAS_ENABLE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetDepthBiasEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectByteCountEXT-logicOp-04878
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_LOGIC_OP_EXT dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of logicOpEnable is VK_TRUE, then vkCmdSetLogicOpEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectByteCountEXT-primitiveFragmentShadingRateWithMultipleViewports-04552
  If the primitiveFragmentShadingRateWithMultipleViewports limit is not supported, the bound graphics pipeline was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, and any of the shader stages of the bound graphics pipeline write to the PrimitiveShadingRateKHR built-in, then vkCmdSetViewportWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the viewportCount parameter of vkCmdSetViewportWithCountEXT **must** be 1

- VUID-vkCmdDrawIndirectByteCountEXT-primitiveFragmentShadingRateWithMultipleViewports-08642
  If the `primitiveFragmentShadingRateWithMultipleViewports` limit is not supported, and any shader object bound to a graphics stage writes to the `PrimitiveShadingRateKHR` built-in, then vkCmdSetViewportWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the `viewportCount` parameter of `vkCmdSetViewportWithCountEXT` **must** be `1`
- VUID-vkCmdDrawIndirectByteCountEXT-blendEnable-04727
  If rasterization is not disabled in the bound graphics pipeline, then for each color attachment in the subpass, if the corresponding image view's format features do not contain `VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT`, then the `blendEnable` member of the corresponding element of the `pAttachments` member of `pColorBlendState` **must** be `VK_FALSE`
- VUID-vkCmdDrawIndirectByteCountEXT-None-08643
  If a shader object is bound to the `VK_SHADER_STAGE_FRAGMENT_BIT` stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE`, then for each color attachment in the render pass, if the corresponding image view's format features do not contain `VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT`, then the corresponding member of `pColorBlendEnables` in the most recent call to `vkCmdSetColorBlendEnableEXT` in the current command buffer that affected that attachment index **must** have been `VK_FALSE`
- VUID-vkCmdDrawIndirectByteCountEXT-multisampledRenderToSingleSampled-07284
  If rasterization is not disabled in the bound graphics pipeline, and none of the following is enabled:
  - the `VK_AMD_mixed_attachment_samples` extension
  - the `VK_NV_framebuffer_mixed_samples` extension
  - the `multisampledRenderToSingleSampled` feature

  then `rasterizationSamples` for the currently bound graphics pipeline **must** be the same as the current subpass color and/or depth/stencil attachments
- VUID-vkCmdDrawIndirectByteCountEXT-None-08644
  If a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE`, and none of the following is enabled:
  - the `VK_AMD_mixed_attachment_samples` extension
  - the `VK_NV_framebuffer_mixed_samples` extension
  - the `multisampledRenderToSingleSampled` feature

  then the most recent call to vkCmdSetRasterizationSamplesEXT in the current command buffer **must** have set `rasterizationSamples` to be the same as the number of samples for the current render pass color and/or depth/stencil attachments
- VUID-vkCmdDrawIndirectByteCountEXT-None-08876
  If a shader object is bound to any graphics stage, the current render pass instance **must**

have been begun with vkCmdBeginRenderingKHR

- VUID-vkCmdDrawIndirectByteCountEXT-imageView-06172
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the `imageView` member of `pDepthAttachment` is not VK_NULL_HANDLE, and the `layout` member of `pDepthAttachment` is `VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL`, this command **must** not write any values to the depth attachment
- VUID-vkCmdDrawIndirectByteCountEXT-imageView-06173
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the `imageView` member of `pStencilAttachment` is not VK_NULL_HANDLE, and the `layout` member of `pStencilAttachment` is `VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL`, this command **must** not write any values to the stencil attachment
- VUID-vkCmdDrawIndirectByteCountEXT-imageView-06174
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the `imageView` member of `pDepthAttachment` is not VK_NULL_HANDLE, and the `layout` member of `pDepthAttachment` is `VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_STENCIL_ATTACHMENT_OPTIMAL`, this command **must** not write any values to the depth attachment
- VUID-vkCmdDrawIndirectByteCountEXT-imageView-06175
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the `imageView` member of `pStencilAttachment` is not VK_NULL_HANDLE, and the `layout` member of `pStencilAttachment` is `VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_STENCIL_READ_ONLY_OPTIMAL`, this command **must** not write any values to the stencil attachment
- VUID-vkCmdDrawIndirectByteCountEXT-imageView-06176
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the `imageView` member of `pDepthAttachment` is not VK_NULL_HANDLE, and the `layout` member of `pDepthAttachment` is `VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL`, this command **must** not write any values to the depth attachment
- VUID-vkCmdDrawIndirectByteCountEXT-imageView-06177
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the `imageView` member of `pStencilAttachment` is not VK_NULL_HANDLE, and the `layout` member of `pStencilAttachment` is `VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL`, this command **must** not write any values to the stencil attachment
- VUID-vkCmdDrawIndirectByteCountEXT-viewMask-06178
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the currently bound graphics pipeline **must** have been created with a VkPipelineRenderingCreateInfo::`viewMask` equal to VkRenderingInfo::`viewMask`
- VUID-vkCmdDrawIndirectByteCountEXT-colorAttachmentCount-06179
  If the `dynamicRenderingUnusedAttachments` feature is not enabled and the current render pass instance was begun with vkCmdBeginRenderingKHR, the currently bound graphics pipeline **must** have been created with a VkPipelineRenderingCreateInfo::`colorAttachmentCount` equal to VkRenderingInfo::`colorAttachmentCount`
- VUID-vkCmdDrawIndirectByteCountEXT-dynamicRenderingUnusedAttachments-08910
  If the `dynamicRenderingUnusedAttachments` feature is not enabled, and the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingInfo

::colorAttachmentCount greater than 0, then each element of the VkRenderingInfo::pColorAttachments array with an imageView not equal to VK_NULL_HANDLE **must** have been created with a VkFormat equal to the corresponding element of VkPipelineRenderingCreateInfo::pColorAttachmentFormats used to create the currently bound graphics pipeline

- VUID-vkCmdDrawIndirectByteCountEXT-dynamicRenderingUnusedAttachments-08912
  If the dynamicRenderingUnusedAttachments feature is not enabled, and the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingInfo::colorAttachmentCount greater than 0, then each element of the VkRenderingInfo::pColorAttachments array with an imageView equal to VK_NULL_HANDLE **must** have the corresponding element of VkPipelineRenderingCreateInfo::pColorAttachmentFormats used to create the currently bound pipeline equal to VK_FORMAT_UNDEFINED

- VUID-vkCmdDrawIndirectByteCountEXT-dynamicRenderingUnusedAttachments-08911
  If the dynamicRenderingUnusedAttachments feature is enabled, and the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingInfo::colorAttachmentCount greater than 0, then each element of the VkRenderingInfo::pColorAttachments array with an imageView not equal to VK_NULL_HANDLE **must** have been created with a VkFormat equal to the corresponding element of VkPipelineRenderingCreateInfo::pColorAttachmentFormats used to create the currently bound graphics pipeline, or the corresponding element of VkPipelineRenderingCreateInfo::pColorAttachmentFormats, if it exists, **must** be VK_FORMAT_UNDEFINED

- VUID-vkCmdDrawIndirectByteCountEXT-colorAttachmentCount-09362
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, with a VkRenderingInfo::colorAttachmentCount equal to 1, there is no shader object bound to any graphics stage, and a color attachment with a resolve mode of VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID, each element of the VkRenderingInfo::pColorAttachments array with a resolveImageView not equal to VK_NULL_HANDLE **must** have been created with an image created with a VkExternalFormatANDROID::externalFormat value equal to the VkExternalFormatANDROID::externalFormat value used to create the currently bound graphics pipeline

- VUID-vkCmdDrawIndirectByteCountEXT-None-09363
  If there is no shader object bound to any graphics stage, the current render pass instance was begun with vkCmdBeginRenderingKHR and a VkRenderingInfo::colorAttachmentCount equal to 1, and a color attachment with a resolve mode of VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID, each element of the VkRenderingInfo::pColorAttachments array with a imageView not equal to VK_NULL_HANDLE **must** have been created with an image created with a VkExternalFormatANDROID::externalFormat value equal to the VkExternalFormatANDROID::externalFormat value used to create the currently bound graphics pipeline

- VUID-vkCmdDrawIndirectByteCountEXT-None-09364
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is no shader object bound to any graphics stage, and the currently bound graphics pipeline was created with a non-zero VkExternalFormatANDROID::externalFormat value and with

the VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT dynamic state enabled, then vkCmdSetColorBlendEnableEXT **must** have set the blend enable to VK_FALSE prior to this drawing command

- VUID-vkCmdDrawIndirectByteCountEXT-None-09365
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is no shader object bound to any graphics stage, and the currently bound graphics pipeline was created with a non-zero VkExternalFormatANDROID::externalFormat value and with the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT dynamic state enabled, then vkCmdSetRasterizationSamplesEXT **must** have set rasterizationSamples to VK_SAMPLE_COUNT_1_BIT prior to this drawing command
- VUID-vkCmdDrawIndirectByteCountEXT-None-09366
  If there is a shader object bound to any graphics stage, and the current render pass includes a color attachment that uses the VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID resolve mode, then vkCmdSetColorBlendEnableEXT **must** have set blend enable to VK_FALSE prior to this drawing command
- VUID-vkCmdDrawIndirectByteCountEXT-rasterizationSamples-09367
  If there is a shader object bound to any graphics stage, and the current render pass includes a color attachment that uses the VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID resolve mode, then vkCmdSetRasterizationSamplesEXT **must** have set rasterizationSamples to VK_SAMPLE_COUNT_1_BIT prior to this drawing command
- VUID-vkCmdDrawIndirectByteCountEXT-None-09368
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is no shader object bound to any graphics stage, and the currently bound graphics pipeline was created with a non-zero VkExternalFormatANDROID::externalFormat value and with the VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR dynamic state enabled, then vkCmdSetFragmentShadingRateKHR **must** have set pFragmentSize->width to 1 prior to this drawing command
- VUID-vkCmdDrawIndirectByteCountEXT-None-09369
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is no shader object bound to any graphics stage, and the currently bound graphics pipeline was created with a non-zero VkExternalFormatANDROID::externalFormat value and with the VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR dynamic state enabled, then vkCmdSetFragmentShadingRateKHR **must** have set pFragmentSize->height to 1 prior to this drawing command
- VUID-vkCmdDrawIndirectByteCountEXT-pFragmentSize-09370
  If there is a shader object bound to any graphics stage, and the current render pass includes a color attachment that uses the VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID resolve mode, then vkCmdSetFragmentShadingRateKHR **must** have set pFragmentSize->width to 1 prior to this drawing command
- VUID-vkCmdDrawIndirectByteCountEXT-pFragmentSize-09371
  If there is a shader object bound to any graphics stage, and the current render pass includes a color attachment that uses the

VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID resolve mode, then vkCmdSetFragmentShadingRateKHR **must** have set pFragmentSize->height to 1 prior to this drawing command

- VUID-vkCmdDrawIndirectByteCountEXT-None-07749
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_WRITE_ENABLE_EXT dynamic state enabled then vkCmdSetColorWriteEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirectByteCountEXT-None-08646
  If the colorWriteEnable feature is enabled on the device, and a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetColorWriteEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirectByteCountEXT-attachmentCount-07750
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_WRITE_ENABLE_EXT dynamic state enabled then the attachmentCount parameter of vkCmdSetColorWriteEnableEXT **must** be greater than or equal to the VkPipelineColorBlendStateCreateInfo::attachmentCount of the currently bound graphics pipeline

- VUID-vkCmdDrawIndirectByteCountEXT-None-08647
  If the colorWriteEnable feature is enabled on the device, and a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then the attachmentCount parameter of most recent call to vkCmdSetColorWriteEnableEXT in the current command buffer **must** be greater than or equal to the number of color attachments in the current render pass instance

- VUID-vkCmdDrawIndirectByteCountEXT-None-07751
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_DISCARD_RECTANGLE_EXT dynamic state enabled then vkCmdSetDiscardRectangleEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command for each discard rectangle in VkPipelineDiscardRectangleStateCreateInfoEXT::discardRectangleCount

- VUID-vkCmdDrawIndirectByteCountEXT-None-07880
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_DISCARD_RECTANGLE_ENABLE_EXT dynamic state enabled then vkCmdSetDiscardRectangleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirectByteCountEXT-rasterizerDiscardEnable-09236
  If the VK_EXT_discard_rectangles extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetDiscardRectangleEnableEXT in the current command buffer set discardRectangleEnable to VK_TRUE, then vkCmdSetDiscardRectangleEXT **must** have been

called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirectByteCountEXT-None-08648
  If the VK_EXT_discard_rectangles extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetDiscardRectangleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirectByteCountEXT-None-07881
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_DISCARD_RECTANGLE_MODE_EXT dynamic state enabled then vkCmdSetDiscardRectangleModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirectByteCountEXT-None-08649
  If the VK_EXT_discard_rectangles extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetDiscardRectangleEnableEXT in the current command buffer set discardRectangleEnable to VK_TRUE, then vkCmdSetDiscardRectangleModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirectByteCountEXT-dynamicRenderingUnusedAttachments-08913
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is not enabled, and VkRenderingInfo::pDepthAttachment->imageView was VK_NULL_HANDLE, the value of VkPipelineRenderingCreateInfo::depthAttachmentFormat used to create the currently bound graphics pipeline **must** be equal to VK_FORMAT_UNDEFINED

- VUID-vkCmdDrawIndirectByteCountEXT-dynamicRenderingUnusedAttachments-08914
  If current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is not enabled, and VkRenderingInfo::pDepthAttachment->imageView was not VK_NULL_HANDLE, the value of VkPipelineRenderingCreateInfo::depthAttachmentFormat used to create the currently bound graphics pipeline **must** be equal to the VkFormat used to create VkRenderingInfo::pDepthAttachment->imageView

- VUID-vkCmdDrawIndirectByteCountEXT-dynamicRenderingUnusedAttachments-08915
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is enabled, VkRenderingInfo::pDepthAttachment->imageView was not VK_NULL_HANDLE, and the value of VkPipelineRenderingCreateInfo::depthAttachmentFormat used to create the currently bound graphics pipeline was not equal to the VkFormat used to create VkRenderingInfo::pDepthAttachment->imageView, the value of the format **must** be VK_FORMAT_UNDEFINED

- VUID-vkCmdDrawIndirectByteCountEXT-dynamicRenderingUnusedAttachments-08916
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is not enabled, and VkRenderingInfo::pStencilAttachment->imageView was VK_NULL_HANDLE, the value of

VkPipelineRenderingCreateInfo::stencilAttachmentFormat used to create the currently bound graphics pipeline **must** be equal to VK_FORMAT_UNDEFINED

- VUID-vkCmdDrawIndirectByteCountEXT-dynamicRenderingUnusedAttachments-08917
  If current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is not enabled, and VkRenderingInfo::pStencilAttachment->imageView was not VK_NULL_HANDLE, the value of VkPipelineRenderingCreateInfo::stencilAttachmentFormat used to create the currently bound graphics pipeline **must** be equal to the VkFormat used to create VkRenderingInfo::pStencilAttachment->imageView
- VUID-vkCmdDrawIndirectByteCountEXT-dynamicRenderingUnusedAttachments-08918
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is enabled, VkRenderingInfo::pStencilAttachment->imageView was not VK_NULL_HANDLE, and the value of VkPipelineRenderingCreateInfo::stencilAttachmentFormat used to create the currently bound graphics pipeline was not equal to the VkFormat used to create VkRenderingInfo::pStencilAttachment->imageView, the value of the format **must** be VK_FORMAT_UNDEFINED
- VUID-vkCmdDrawIndirectByteCountEXT-imageView-06183
  If the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingFragmentShadingRateAttachmentInfoKHR::imageView was not VK_NULL_HANDLE, the currently bound graphics pipeline **must** have been created with VK_PIPELINE_CREATE_RENDERING_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR
- VUID-vkCmdDrawIndirectByteCountEXT-imageView-06184
  If the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingFragmentDensityMapAttachmentInfoEXT::imageView was not VK_NULL_HANDLE, the currently bound graphics pipeline **must** have been created with VK_PIPELINE_CREATE_RENDERING_FRAGMENT_DENSITY_MAP_ATTACHMENT_BIT_EXT
- VUID-vkCmdDrawIndirectByteCountEXT-colorAttachmentCount-06185
  If the currently bound pipeline was created with a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and the current render pass instance was begun with vkCmdBeginRenderingKHR with a VkRenderingInfo::colorAttachmentCount parameter greater than 0, then each element of the VkRenderingInfo::pColorAttachments array with a imageView not equal to VK_NULL_HANDLE **must** have been created with a sample count equal to the corresponding element of the pColorAttachmentSamples member of VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV used to create the currently bound graphics pipeline
- VUID-vkCmdDrawIndirectByteCountEXT-pDepthAttachment-06186
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the currently bound pipeline was created with a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and VkRenderingInfo::pDepthAttachment->imageView was not VK_NULL_HANDLE, the value of the depthStencilAttachmentSamples member of VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV used to create the currently bound graphics pipeline **must** be equal to the sample count used to create VkRenderingInfo::pDepthAttachment->imageView
- VUID-vkCmdDrawIndirectByteCountEXT-pStencilAttachment-06187
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the

currently bound pipeline was created with a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and VkRenderingInfo::pStencilAttachment->imageView was not VK_NULL_HANDLE, the value of the depthStencilAttachmentSamples member of VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV used to create the currently bound graphics pipeline **must** be equal to the sample count used to create VkRenderingInfo::pStencilAttachment->imageView

- VUID-vkCmdDrawIndirectByteCountEXT-multisampledRenderToSingleSampled-07285
  If the currently bound pipeline was created without a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and the multisampledRenderToSingleSampled feature is not enabled, and the current render pass instance was begun with vkCmdBeginRenderingKHR with a VkRenderingInfo::colorAttachmentCount parameter greater than 0, then each element of the VkRenderingInfo::pColorAttachments array with a imageView not equal to VK_NULL_HANDLE **must** have been created with a sample count equal to the value of rasterizationSamples for the currently bound graphics pipeline
- VUID-vkCmdDrawIndirectByteCountEXT-multisampledRenderToSingleSampled-07286
  If the currently bound pipeline was created without a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and the multisampledRenderToSingleSampled feature is not enabled, and VkRenderingInfo::pDepthAttachment->imageView was not VK_NULL_HANDLE, the value of rasterizationSamples for the currently bound graphics pipeline **must** be equal to the sample count used to create VkRenderingInfo::pDepthAttachment->imageView
- VUID-vkCmdDrawIndirectByteCountEXT-multisampledRenderToSingleSampled-07287
  If the currently bound pipeline was created without a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and the multisampledRenderToSingleSampled feature is not enabled, and VkRenderingInfo::pStencilAttachment->imageView was not VK_NULL_HANDLE, the value of rasterizationSamples for the currently bound graphics pipeline **must** be equal to the sample count used to create VkRenderingInfo::pStencilAttachment->imageView
- VUID-vkCmdDrawIndirectByteCountEXT-pNext-07935
  If this command has been called inside a render pass instance started with vkCmdBeginRenderingKHR, and the pNext chain of VkRenderingInfo includes a VkMultisampledRenderToSingleSampledInfoEXT structure with multisampledRenderToSingleSampledEnable equal to VK_TRUE, then the value of rasterizationSamples for the currently bound graphics pipeline **must** be equal to VkMultisampledRenderToSingleSampledInfoEXT::rasterizationSamples
- VUID-vkCmdDrawIndirectByteCountEXT-renderPass-06198
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the currently bound pipeline **must** have been created with a VkGraphicsPipelineCreateInfo::renderPass equal to VK_NULL_HANDLE
- VUID-vkCmdDrawIndirectByteCountEXT-pColorAttachments-08963
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is a graphics pipeline bound with a fragment shader that statically writes to a color attachment, the color write mask is not zero, color writes are enabled, and the corresponding element of the VkRenderingInfo::pColorAttachments->imageView was not

VK_NULL_HANDLE, then the corresponding element of VkPipelineRenderingCreateInfo::pColorAttachmentFormats used to create the pipeline **must** not be VK_FORMAT_UNDEFINED

- VUID-vkCmdDrawIndirectByteCountEXT-pDepthAttachment-08964
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is a graphics pipeline bound, depth test is enabled, depth write is enabled, and the VkRenderingInfo::pDepthAttachment->imageView was not VK_NULL_HANDLE, then the VkPipelineRenderingCreateInfo::depthAttachmentFormat used to create the pipeline **must** not be VK_FORMAT_UNDEFINED

- VUID-vkCmdDrawIndirectByteCountEXT-pStencilAttachment-08965
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is a graphics pipeline bound, stencil test is enabled and the VkRenderingInfo::pStencilAttachment->imageView was not VK_NULL_HANDLE, then the VkPipelineRenderingCreateInfo::stencilAttachmentFormat used to create the pipeline **must** not be VK_FORMAT_UNDEFINED

- VUID-vkCmdDrawIndirectByteCountEXT-primitivesGeneratedQueryWithRasterizerDiscard-06708
  If the primitivesGeneratedQueryWithRasterizerDiscard feature is not enabled and the VK_QUERY_TYPE_PRIMITIVES_GENERATED_EXT query is active, rasterization discard **must** not be enabled

- VUID-vkCmdDrawIndirectByteCountEXT-primitivesGeneratedQueryWithNonZeroStreams-06709
  If the primitivesGeneratedQueryWithNonZeroStreams feature is not enabled and the VK_QUERY_TYPE_PRIMITIVES_GENERATED_EXT query is active, the bound graphics pipeline **must** not have been created with a non-zero value in VkPipelineRasterizationStateStreamCreateInfoEXT::rasterizationStream

- VUID-vkCmdDrawIndirectByteCountEXT-None-07619
  If a shader object is bound to the VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_TESSELLATION_DOMAIN_ORIGIN_EXT dynamic state enabled, then vkCmdSetTessellationDomainOriginEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirectByteCountEXT-None-07620
  If the depthClamp feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_CLAMP_ENABLE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetDepthClampEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirectByteCountEXT-None-07621
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_POLYGON_MODE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetPolygonModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirectByteCountEXT-None-07622

If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetRasterizationSamplesEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirectByteCountEXT-None-07623
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_SAMPLE_MASK_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetSampleMaskEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirectByteCountEXT-alphaToCoverageEnable-08919
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_ALPHA_TO_COVERAGE_ENABLE_EXT dynamic state enabled, and alphaToCoverageEnable was VK_TRUE in the last call to vkCmdSetAlphaToCoverageEnableEXT, then the Fragment Output Interface **must** contain a variable for the alpha Component word in Location 0 at Index 0

- VUID-vkCmdDrawIndirectByteCountEXT-alphaToCoverageEnable-08920
  If a shader object is bound to any graphics stage, and the most recent call to vkCmdSetAlphaToCoverageEnableEXT in the current command buffer set alphaToCoverageEnable to VK_TRUE, then the Fragment Output Interface **must** contain a variable for the alpha Component word in Location 0 at Index 0

- VUID-vkCmdDrawIndirectByteCountEXT-None-07624
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_ALPHA_TO_COVERAGE_ENABLE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetAlphaToCoverageEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirectByteCountEXT-None-07625
  If the alphaToOne feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_ALPHA_TO_ONE_ENABLE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetAlphaToOneEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirectByteCountEXT-None-07626
  If the logicOp feature is enabled, a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_LOGIC_OP_ENABLE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetLogicOpEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirectByteCountEXT-None-07627
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT dynamic state enabled then

vkCmdSetColorBlendEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirectByteCountEXT-None-08657
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and both the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE and there are color attachments bound, then vkCmdSetColorBlendEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectByteCountEXT-None-07628
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_EQUATION_EXT dynamic state enabled then vkCmdSetColorBlendEquationEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectByteCountEXT-None-08658
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetColorBlendEnableEXT for any attachment set that attachment's value in pColorBlendEnables to VK_TRUE, then vkCmdSetColorBlendEquationEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectByteCountEXT-None-07629
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_WRITE_MASK_EXT dynamic state enabled then vkCmdSetColorWriteMaskEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectByteCountEXT-None-08659
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and both the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE and there are color attachments bound, then vkCmdSetColorWriteMaskEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectByteCountEXT-None-07630
  If the geometryStreams feature is enabled, and a shader object is bound to the VK_SHADER_STAGE_GEOMETRY_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_RASTERIZATION_STREAM_EXT dynamic state enabled, then vkCmdSetRasterizationStreamEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectByteCountEXT-None-07631
  If the VK_EXT_conservative_rasterization extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_CONSERVATIVE_RASTERIZATION_MODE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetConservativeRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirectByteCountEXT-None-07632
  If the VK_EXT_conservative_rasterization extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_EXTRA_PRIMITIVE_OVERESTIMATION_SIZE_EXT dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of conservativeRasterizationMode is VK_CONSERVATIVE_RASTERIZATION_MODE_OVERESTIMATE_EXT, then vkCmdSetExtraPrimitiveOverestimationSizeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectByteCountEXT-None-07633
  If the depthClipEnable feature is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_CLIP_ENABLE_EXT dynamic state, then vkCmdSetDepthClipEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectByteCountEXT-None-07634
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT dynamic state enabled then vkCmdSetSampleLocationsEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectByteCountEXT-None-08664
  If the VK_EXT_sample_locations extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetSampleLocationsEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectByteCountEXT-None-07635
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_ADVANCED_EXT dynamic state enabled then vkCmdSetColorBlendAdvancedEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectByteCountEXT-rasterizerDiscardEnable-09416
  If the VK_EXT_blend_operation_advanced extension is enabled, and a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then at least one of vkCmdSetColorBlendEquationEXT and vkCmdSetColorBlendAdvancedEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectByteCountEXT-None-07636
  If the VK_EXT_provoking_vertex extension is enabled, a shader object is bound to the VK_SHADER_STAGE_VERTEX_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_PROVOKING_VERTEX_MODE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetProvokingVertexModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirectByteCountEXT-None-07637
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT` dynamic state enabled then vkCmdSetLineRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectByteCountEXT-None-08666
  If the `VK_KHR_line_rasterization` or `VK_EXT_line_rasterization` extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE`, and the most recent call to vkCmdSetPolygonModeEXT in the current command buffer set `polygonMode` to `VK_POLYGON_MODE_LINE`, then vkCmdSetLineRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectByteCountEXT-None-08667
  If the `VK_KHR_line_rasterization` or `VK_EXT_line_rasterization` extension is enabled, and a shader object is bound to the `VK_SHADER_STAGE_VERTEX_BIT` stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE`, and the most recent call to vkCmdSetPrimitiveTopologyEXT in the current command buffer set `primitiveTopology` to any line topology, then vkCmdSetLineRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectByteCountEXT-None-08668
  If the `VK_KHR_line_rasterization` or `VK_EXT_line_rasterization` extension is enabled, and a shader object that outputs line primitives is bound to the `VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT` or `VK_SHADER_STAGE_GEOMETRY_BIT` stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE`, then vkCmdSetLineRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectByteCountEXT-None-07638
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT` dynamic state enabled then vkCmdSetLineStippleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectByteCountEXT-None-08669
  If the `VK_KHR_line_rasterization` or `VK_EXT_line_rasterization` extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE`, and the most recent call to vkCmdSetPolygonModeEXT in the current command buffer set `polygonMode` to `VK_POLYGON_MODE_LINE`, then vkCmdSetLineStippleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirectByteCountEXT-None-08670
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object is bound to the VK_SHADER_STAGE_VERTEX_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetPrimitiveTopologyEXT in the current command buffer set primitiveTopology to any line topology, then vkCmdSetLineStippleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectByteCountEXT-None-08671
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object that outputs line primitives is bound to the VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT or VK_SHADER_STAGE_GEOMETRY_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetLineStippleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectByteCountEXT-None-07849
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_KHR dynamic state enabled then vkCmdSetLineStippleKHR **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectByteCountEXT-None-08672
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetLineStippleEnableEXT in the current command buffer set stippledLineEnable to VK_TRUE, then vkCmdSetLineStippleEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectByteCountEXT-None-07639
  If the depthClipControl feature is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_CLIP_NEGATIVE_ONE_TO_ONE_EXT dynamic state enabled, then vkCmdSetDepthClipNegativeOneToOneEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectByteCountEXT-None-07640
  If the VK_NV_clip_space_w_scaling extension is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_VIEWPORT_W_SCALING_ENABLE_NV dynamic state enabled, then vkCmdSetViewportWScalingEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectByteCountEXT-None-07641
  If the VK_NV_viewport_swizzle extension is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_VIEWPORT_SWIZZLE_NV dynamic state enabled, then

vkCmdSetViewportSwizzleNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirectByteCountEXT-None-07642
  If the VK_NV_fragment_coverage_to_color extension is enabled, a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_TO_COLOR_ENABLE_NV dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetCoverageToColorEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectByteCountEXT-None-07643
  If the VK_NV_fragment_coverage_to_color extension is enabled, a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_TO_COLOR_LOCATION_NV dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of coverageToColorEnable is VK_TRUE, then vkCmdSetCoverageToColorLocationNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectByteCountEXT-None-07644
  If the VK_NV_framebuffer_mixed_samples extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_MODULATION_MODE_NV dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetCoverageModulationModeNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectByteCountEXT-None-07645
  If the VK_NV_framebuffer_mixed_samples extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_MODULATION_TABLE_ENABLE_NV dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of coverageModulationMode is any value other than VK_COVERAGE_MODULATION_MODE_NONE_NV, then vkCmdSetCoverageModulationTableEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectByteCountEXT-None-07646
  If the VK_NV_framebuffer_mixed_samples extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_MODULATION_TABLE_NV dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of coverageModulationTableEnable is VK_TRUE, then vkCmdSetCoverageModulationTableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectByteCountEXT-None-07647
  If the shadingRateImage feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_SHADING_RATE_IMAGE_ENABLE_NV dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetShadingRateImageEnableNV

**must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirectByteCountEXT-pipelineFragmentShadingRate-09238
  If the `pipelineFragmentShadingRate` feature is enabled, a shader object is bound to the `VK_SHADER_STAGE_FRAGMENT_BIT` stage or a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR` dynamic state enabled, and the current value of `rasterizerDiscardEnable` is `VK_FALSE`, then vkCmdSetFragmentShadingRateKHR **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirectByteCountEXT-None-07648
  If the `representativeFragmentTest` feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_REPRESENTATIVE_FRAGMENT_TEST_ENABLE_NV` dynamic state enabled, and the current value of `rasterizerDiscardEnable` is `VK_FALSE`, then vkCmdSetRepresentativeFragmentTestEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirectByteCountEXT-None-07649
  If the `coverageReductionMode` feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_COVERAGE_REDUCTION_MODE_NV` dynamic state enabled, and the current value of `rasterizerDiscardEnable` is `VK_FALSE`, then vkCmdSetCoverageReductionModeNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirectByteCountEXT-pColorBlendEnables-07470
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT` state enabled and the last call to vkCmdSetColorBlendEnableEXT set `pColorBlendEnables` for any attachment to `VK_TRUE`, then for those attachments in the subpass the corresponding image view's format features **must** contain `VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT`

- VUID-vkCmdDrawIndirectByteCountEXT-rasterizationSamples-07471
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT` state enabled, and the current subpass does not use any color and/or depth/stencil attachments, then the `rasterizationSamples` in the last call to vkCmdSetRasterizationSamplesEXT **must** follow the rules for a zero-attachment subpass

- VUID-vkCmdDrawIndirectByteCountEXT-samples-07472
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_SAMPLE_MASK_EXT` state enabled and the `VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT` state disabled, then the `samples` parameter in the last call to vkCmdSetSampleMaskEXT **must** be greater or equal to the VkPipelineMultisampleStateCreateInfo::`rasterizationSamples` parameter used to create the bound graphics pipeline

- VUID-vkCmdDrawIndirectByteCountEXT-samples-07473
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_SAMPLE_MASK_EXT` state and `VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT`

states enabled, then the samples parameter in the last call to vkCmdSetSampleMaskEXT **must** be greater or equal to the rasterizationSamples parameter in the last call to vkCmdSetRasterizationSamplesEXT

- VUID-vkCmdDrawIndirectByteCountEXT-rasterizationSamples-07474
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, and neither the VK_AMD_mixed_attachment_samples nor the VK_NV_framebuffer_mixed_samples extensions are enabled, then the rasterizationSamples in the last call to vkCmdSetRasterizationSamplesEXT **must** be the same as the current subpass color and/or depth/stencil attachments

- VUID-vkCmdDrawIndirectByteCountEXT-None-09211
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, or a shader object is bound to any graphics stage, and the current render pass instance includes a VkMultisampledRenderToSingleSampledInfoEXT structure with multisampledRenderToSingleSampledEnable equal to VK_TRUE, then the rasterizationSamples in the last call to vkCmdSetRasterizationSamplesEXT **must** be the same as the rasterizationSamples member of that structure

- VUID-vkCmdDrawIndirectByteCountEXT-firstAttachment-07476
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT dynamic state enabled then vkCmdSetColorBlendEnableEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the firstAttachment and attachmentCount parameters of vkCmdSetColorBlendEnableEXT calls **must** specify an enable for all active color attachments in the current subpass

- VUID-vkCmdDrawIndirectByteCountEXT-rasterizerDiscardEnable-09417
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetColorBlendEnableEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the firstAttachment and attachmentCount parameters of vkCmdSetColorBlendEnableEXT calls **must** specify an enable for all active color attachments in the current subpass

- VUID-vkCmdDrawIndirectByteCountEXT-firstAttachment-07477
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_EQUATION_EXT dynamic state enabled then vkCmdSetColorBlendEquationEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the firstAttachment and attachmentCount parameters of vkCmdSetColorBlendEquationEXT calls **must** specify the blend equations for all active color attachments in the current subpass where blending is enabled

- VUID-vkCmdDrawIndirectByteCountEXT-rasterizerDiscardEnable-09418
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and both the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE and there are color attachments bound, then

vkCmdSetColorBlendEquationEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the `firstAttachment` and `attachmentCount` parameters of `vkCmdSetColorBlendEquationEXT` calls **must** specify the blend equations for all active color attachments in the current subpass where blending is enabled

- VUID-vkCmdDrawIndirectByteCountEXT-firstAttachment-07478
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_COLOR_WRITE_MASK_EXT` dynamic state enabled then vkCmdSetColorWriteMaskEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the `firstAttachment` and `attachmentCount` parameters of `vkCmdSetColorWriteMaskEXT` calls **must** specify the color write mask for all active color attachments in the current subpass

- VUID-vkCmdDrawIndirectByteCountEXT-rasterizerDiscardEnable-09419
  If a shader object is bound to the `VK_SHADER_STAGE_FRAGMENT_BIT` stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE`, then vkCmdSetColorWriteMaskEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the `firstAttachment` and `attachmentCount` parameters of `vkCmdSetColorWriteMaskEXT` calls **must** specify the color write mask for all active color attachments in the current subpass

- VUID-vkCmdDrawIndirectByteCountEXT-firstAttachment-07479
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_COLOR_BLEND_ADVANCED_EXT` dynamic state enabled then vkCmdSetColorBlendAdvancedEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the `firstAttachment` and `attachmentCount` parameters of `vkCmdSetColorBlendAdvancedEXT` calls **must** specify the advanced blend equations for all active color attachments in the current subpass where blending is enabled

- VUID-vkCmdDrawIndirectByteCountEXT-advancedBlendMaxColorAttachments-07480
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_COLOR_BLEND_ADVANCED_EXT` and `VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT` dynamic states enabled and the last calls to vkCmdSetColorBlendEnableEXT and vkCmdSetColorBlendAdvancedEXT have enabled advanced blending, then the number of active color attachments in the current subpass **must** not exceed `advancedBlendMaxColorAttachments`

- VUID-vkCmdDrawIndirectByteCountEXT-primitivesGeneratedQueryWithNonZeroStreams-07481
  If the `primitivesGeneratedQueryWithNonZeroStreams` feature is not enabled and the `VK_QUERY_TYPE_PRIMITIVES_GENERATED_EXT` query is active, and the bound graphics pipeline was created with `VK_DYNAMIC_STATE_RASTERIZATION_STREAM_EXT` state enabled, the last call to vkCmdSetRasterizationStreamEXT **must** have set the `rasterizationStream` to zero

- VUID-vkCmdDrawIndirectByteCountEXT-sampleLocationsPerPixel-07482
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT` state enabled and the `VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT` state disabled, then the

sampleLocationsPerPixel member of pSampleLocationsInfo in the last call to vkCmdSetSampleLocationsEXT **must** equal the rasterizationSamples member of the VkPipelineMultisampleStateCreateInfo structure the bound graphics pipeline has been created with

- VUID-vkCmdDrawIndirectByteCountEXT-sampleLocationsPerPixel-07483
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state enabled and the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, then the sampleLocationsPerPixel member of pSampleLocationsInfo in the last call to vkCmdSetSampleLocationsEXT **must** equal the rasterizationSamples parameter of the last call to vkCmdSetRasterizationSamplesEXT

- VUID-vkCmdDrawIndirectByteCountEXT-sampleLocationsEnable-07484
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, or the bound graphics pipeline was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, and sampleLocationsEnable was VK_TRUE in the last call to vkCmdSetSampleLocationsEnableEXT, and the current subpass has a depth/stencil attachment, then that attachment **must** have been created with the VK_IMAGE_CREATE_SAMPLE_LOCATIONS_COMPATIBLE_DEPTH_BIT_EXT bit set

- VUID-vkCmdDrawIndirectByteCountEXT-sampleLocationsEnable-07485
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state enabled and the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, and if sampleLocationsEnable was VK_TRUE in the last call to vkCmdSetSampleLocationsEnableEXT, then the sampleLocationsInfo.sampleLocationGridSize.width in the last call to vkCmdSetSampleLocationsEXT **must** evenly divide VkMultisamplePropertiesEXT::sampleLocationGridSize.width as returned by vkGetPhysicalDeviceMultisamplePropertiesEXT with a samples parameter equaling rasterizationSamples

- VUID-vkCmdDrawIndirectByteCountEXT-sampleLocationsEnable-07486
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state enabled and the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, and if sampleLocationsEnable was VK_TRUE in the last call to vkCmdSetSampleLocationsEnableEXT, then the sampleLocationsInfo.sampleLocationGridSize.height in the last call to vkCmdSetSampleLocationsEXT **must** evenly divide VkMultisamplePropertiesEXT::sampleLocationGridSize.height as returned by vkGetPhysicalDeviceMultisamplePropertiesEXT with a samples parameter equaling rasterizationSamples

- VUID-vkCmdDrawIndirectByteCountEXT-sampleLocationsEnable-07487
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, and if sampleLocationsEnable was VK_TRUE in the last call to vkCmdSetSampleLocationsEnableEXT, the fragment shader code **must** not statically use the extended instruction InterpolateAtSample

- VUID-vkCmdDrawIndirectByteCountEXT-sampleLocationsEnable-07936
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT` state disabled and the `VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT` state enabled, the `sampleLocationsEnable` member of a VkPipelineSampleLocationsStateCreateInfoEXT::`sampleLocationsEnable` in the bound graphics pipeline is `VK_TRUE` or `VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT` state enabled, then, `sampleLocationsInfo.sampleLocationGridSize.width` **must** evenly divide VkMultisamplePropertiesEXT::`sampleLocationGridSize.width` as returned by vkGetPhysicalDeviceMultisamplePropertiesEXT with a `samples` parameter equaling the value of `rasterizationSamples` in the last call to vkCmdSetRasterizationSamplesEXT
- VUID-vkCmdDrawIndirectByteCountEXT-sampleLocationsEnable-07937
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT` state disabled and the `VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT` state enabled, the `sampleLocationsEnable` member of a VkPipelineSampleLocationsStateCreateInfoEXT::`sampleLocationsEnable` in the bound graphics pipeline is `VK_TRUE` or `VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT` state enabled, then, `sampleLocationsInfo.sampleLocationGridSize.height` **must** evenly divide VkMultisamplePropertiesEXT::`sampleLocationGridSize.height` as returned by vkGetPhysicalDeviceMultisamplePropertiesEXT with a `samples` parameter equaling the value of `rasterizationSamples` in the last call to vkCmdSetRasterizationSamplesEXT
- VUID-vkCmdDrawIndirectByteCountEXT-sampleLocationsEnable-07938
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT` state disabled and the `VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT` state enabled, the `sampleLocationsEnable` member of a VkPipelineSampleLocationsStateCreateInfoEXT::`sampleLocationsEnable` in the bound graphics pipeline is `VK_TRUE` or `VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT` state enabled, then, `sampleLocationsInfo.sampleLocationsPerPixel` **must** equal `rasterizationSamples` in the last call to vkCmdSetRasterizationSamplesEXT
- VUID-vkCmdDrawIndirectByteCountEXT-coverageModulationTableEnable-07488
  If a shader object is bound to any graphics stage or the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_COVERAGE_MODULATION_TABLE_ENABLE_NV` state enabled, and the last call to vkCmdSetCoverageModulationTableEnableNV set `coverageModulationTableEnable` to `VK_TRUE`, then the `coverageModulationTableCount` parameter in the last call to vkCmdSetCoverageModulationTableNV **must** equal the current `rasterizationSamples` divided by the number of color samples in the current subpass
- VUID-vkCmdDrawIndirectByteCountEXT-rasterizationSamples-07489
  If the `VK_NV_framebuffer_mixed_samples` extension is enabled, and if current subpass has a depth/stencil attachment and depth test, stencil test, or depth bounds test are enabled in the currently bound pipeline state, then the current `rasterizationSamples` **must** be the same as the sample count of the depth/stencil attachment
- VUID-vkCmdDrawIndirectByteCountEXT-coverageToColorEnable-07490
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_COVERAGE_TO_COLOR_ENABLE_NV` state enabled and the last call to vkCmdSetCoverageToColorEnableNV set the `coverageToColorEnable` to `VK_TRUE`, then the current subpass **must** have a color attachment at the location selected by the last call to

vkCmdSetCoverageToColorLocationNV coverageToColorLocation, with a VkFormat of VK_FORMAT_R8_UINT, VK_FORMAT_R8_SINT, VK_FORMAT_R16_UINT, VK_FORMAT_R16_SINT, VK_FORMAT_R32_UINT, or VK_FORMAT_R32_SINT

- VUID-vkCmdDrawIndirectByteCountEXT-rasterizerDiscardEnable-09420
  If the VK_NV_fragment_coverage_to_color extension is enabled, and a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the last call to vkCmdSetCoverageToColorEnableNV set the coverageToColorEnable to VK_TRUE, then the current subpass **must** have a color attachment at the location selected by the last call to vkCmdSetCoverageToColorLocationNV coverageToColorLocation, with a VkFormat of VK_FORMAT_R8_UINT, VK_FORMAT_R8_SINT, VK_FORMAT_R16_UINT, VK_FORMAT_R16_SINT, VK_FORMAT_R32_UINT, or VK_FORMAT_R32_SINT
- VUID-vkCmdDrawIndirectByteCountEXT-coverageReductionMode-07491
  If this VK_NV_coverage_reduction_mode extension is enabled, the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COVERAGE_TO_COLOR_ENABLE_NV and VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT states enabled, the current coverage reduction mode coverageReductionMode, then the current rasterizationSamples, and the sample counts for the color and depth/stencil attachments (if the subpass has them) **must** be a valid combination returned by vkGetPhysicalDeviceSupportedFramebufferMixedSamplesCombinationsNV
- VUID-vkCmdDrawIndirectByteCountEXT-viewportCount-07492
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, but not the VK_DYNAMIC_STATE_VIEWPORT_SWIZZLE_NV dynamic state enabled, then the bound graphics pipeline **must** have been created with VkPipelineViewportSwizzleStateCreateInfoNV::viewportCount greater or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawIndirectByteCountEXT-viewportCount-07493
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT and VK_DYNAMIC_STATE_VIEWPORT_SWIZZLE_NV dynamic states enabled then the viewportCount parameter in the last call to vkCmdSetViewportSwizzleNV **must** be greater than or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawIndirectByteCountEXT-viewportCount-09421
  If the VK_NV_viewport_swizzle extension is enabled, and a shader object is bound to any graphics stage, then the viewportCount parameter in the last call to vkCmdSetViewportSwizzleNV **must** be greater than or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawIndirectByteCountEXT-rasterizationSamples-07494
  If the VK_NV_framebuffer_mixed_samples extension is enabled, and if the current subpass has any color attachments and rasterizationSamples of the last call to vkCmdSetRasterizationSamplesEXT is greater than the number of color samples, then the pipeline sampleShadingEnable **must** be VK_FALSE
- VUID-vkCmdDrawIndirectByteCountEXT-stippledLineEnable-07495

If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT or VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT dynamic states enabled, and if the current stippledLineEnable state is VK_TRUE and the current lineRasterizationMode state is VK_LINE_RASTERIZATION_MODE_RECTANGULAR_KHR, then the stippledRectangularLines feature **must** be enabled

- VUID-vkCmdDrawIndirectByteCountEXT-stippledLineEnable-07496
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT or VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT dynamic states enabled, and if the current stippledLineEnable state is VK_TRUE and the current lineRasterizationMode state is VK_LINE_RASTERIZATION_MODE_BRESENHAM_KHR, then the stippledBresenhamLines feature **must** be enabled

- VUID-vkCmdDrawIndirectByteCountEXT-stippledLineEnable-07497
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT or VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT dynamic states enabled, and if the current stippledLineEnable state is VK_TRUE and the current lineRasterizationMode state is VK_LINE_RASTERIZATION_MODE_RECTANGULAR_SMOOTH_KHR, then the stippledSmoothLines feature **must** be enabled

- VUID-vkCmdDrawIndirectByteCountEXT-stippledLineEnable-07498
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT or VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT dynamic states enabled, and if the current stippledLineEnable state is VK_TRUE and the current lineRasterizationMode state is VK_LINE_RASTERIZATION_MODE_DEFAULT_KHR, then the stippledRectangularLines feature **must** be enabled and VkPhysicalDeviceLimits::strictLines **must** be VK_TRUE

- VUID-vkCmdDrawIndirectByteCountEXT-conservativePointAndLineRasterization-07499
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_CONSERVATIVE_RASTERIZATION_MODE_EXT dynamic state enabled, conservativePointAndLineRasterization is not supported, and the effective primitive topology output by the last pre-rasterization shader stage is a line or point, then the conservativeRasterizationMode set by the last call to vkCmdSetConservativeRasterizationModeEXT **must** be VK_CONSERVATIVE_RASTERIZATION_MODE_DISABLED_EXT

- VUID-vkCmdDrawIndirectByteCountEXT-stage-07073
  If the currently bound pipeline was created with the VkPipelineShaderStageCreateInfo::stage member of an element of VkGraphicsPipelineCreateInfo::pStages set to VK_SHADER_STAGE_VERTEX_BIT, VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT, VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT or VK_SHADER_STAGE_GEOMETRY_BIT, then Mesh Shader Queries **must** not be active

- VUID-vkCmdDrawIndirectByteCountEXT-None-08877
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_ATTACHMENT_FEEDBACK_LOOP_ENABLE_EXT dynamic state enabled, and the

current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetAttachmentFeedbackLoopEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirectByteCountEXT-None-07850
  If dynamic state was inherited from VkCommandBufferInheritanceViewportScissorInfoNV, it **must** be set in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectByteCountEXT-None-08684
  If there is no bound graphics pipeline, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_VERTEX_BIT
- VUID-vkCmdDrawIndirectByteCountEXT-None-08685
  If there is no bound graphics pipeline, and the tessellationShader feature is enabled, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT
- VUID-vkCmdDrawIndirectByteCountEXT-None-08686
  If there is no bound graphics pipeline, and the tessellationShader feature is enabled, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT
- VUID-vkCmdDrawIndirectByteCountEXT-None-08687
  If there is no bound graphics pipeline, and the geometryShader feature is enabled, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_GEOMETRY_BIT
- VUID-vkCmdDrawIndirectByteCountEXT-None-08688
  If there is no bound graphics pipeline, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_FRAGMENT_BIT
- VUID-vkCmdDrawIndirectByteCountEXT-None-08689
  If there is no bound graphics pipeline, and the taskShader feature is enabled, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_TASK_BIT_EXT
- VUID-vkCmdDrawIndirectByteCountEXT-None-08690
  If there is no bound graphics pipeline, and the meshShader feature is enabled, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_MESH_BIT_EXT
- VUID-vkCmdDrawIndirectByteCountEXT-None-08693
  If there is no bound graphics pipeline, and at least one of the taskShader and meshShader features is enabled, one of the VK_SHADER_STAGE_VERTEX_BIT or VK_SHADER_STAGE_MESH_BIT_EXT stages **must** have a valid VkShaderEXT bound, and the other **must** have no VkShaderEXT bound
- VUID-vkCmdDrawIndirectByteCountEXT-None-08694
  If there is no bound graphics pipeline, and both the taskShader and meshShader features are enabled, and a valid VkShaderEXT is bound the to the VK_SHADER_STAGE_MESH_BIT_EXT stage, and that VkShaderEXT was created without the VK_SHADER_CREATE_NO_TASK_SHADER_BIT_EXT flag, a valid VkShaderEXT **must** be bound to the VK_SHADER_STAGE_TASK_BIT_EXT stage

- VUID-vkCmdDrawIndirectByteCountEXT-None-08695
  If there is no bound graphics pipeline, and both the `taskShader` and `meshShader` features are enabled, and a valid `VkShaderEXT` is bound the to the `VK_SHADER_STAGE_MESH_BIT_EXT` stage, and that `VkShaderEXT` was created with the `VK_SHADER_CREATE_NO_TASK_SHADER_BIT_EXT` flag, there **must** be no `VkShaderEXT` bound to the `VK_SHADER_STAGE_TASK_BIT_EXT` stage
- VUID-vkCmdDrawIndirectByteCountEXT-None-08696
  If there is no bound graphics pipeline, and a valid `VkShaderEXT` is bound to the `VK_SHADER_STAGE_VERTEX_BIT` stage, there **must** be no `VkShaderEXT` bound to either the `VK_SHADER_STAGE_TASK_BIT_EXT` stage or the `VK_SHADER_STAGE_MESH_BIT_EXT` stage
- VUID-vkCmdDrawIndirectByteCountEXT-None-08698
  If any graphics shader is bound which was created with the `VK_SHADER_CREATE_LINK_STAGE_BIT_EXT` flag, then all shaders created with the `VK_SHADER_CREATE_LINK_STAGE_BIT_EXT` flag in the same vkCreateShadersEXT call **must** also be bound
- VUID-vkCmdDrawIndirectByteCountEXT-None-08699
  If any graphics shader is bound which was created with the `VK_SHADER_CREATE_LINK_STAGE_BIT_EXT` flag, any stages in between stages whose shaders which did not create a shader with the `VK_SHADER_CREATE_LINK_STAGE_BIT_EXT` flag as part of the same vkCreateShadersEXT call **must** not have any `VkShaderEXT` bound
- VUID-vkCmdDrawIndirectByteCountEXT-None-08878
  All bound graphics shader objects **must** have been created with identical or identically defined push constant ranges
- VUID-vkCmdDrawIndirectByteCountEXT-None-08879
  All bound graphics shader objects **must** have been created with identical or identically defined arrays of descriptor set layouts
- VUID-vkCmdDrawIndirectByteCountEXT-colorAttachmentCount-09372
  If the current render pass instance was begun with vkCmdBeginRenderingKHR and a VkRenderingInfo::`colorAttachmentCount` equal to `1`, a color attachment with a resolve mode of `VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID`, and a fragment shader is bound, it **must** not declare the `DepthReplacing` or `StencilRefReplacingEXT` execution modes
- VUID-vkCmdDrawIndirectByteCountEXT-pDynamicStates-08715
  If the bound graphics pipeline state includes a fragment shader stage, was created with `VK_DYNAMIC_STATE_DEPTH_WRITE_ENABLE` set in VkPipelineDynamicStateCreateInfo::`pDynamicStates`, and the fragment shader declares the `EarlyFragmentTests` execution mode and uses `OpDepthAttachmentReadEXT`, the `depthWriteEnable` parameter in the last call to vkCmdSetDepthWriteEnableEXT **must** be `VK_FALSE`
- VUID-vkCmdDrawIndirectByteCountEXT-pDynamicStates-08716
  If the bound graphics pipeline state includes a fragment shader stage, was created with `VK_DYNAMIC_STATE_STENCIL_WRITE_MASK` set in VkPipelineDynamicStateCreateInfo::`pDynamicStates`, and the fragment shader declares the `EarlyFragmentTests` execution mode and uses `OpStencilAttachmentReadEXT`, the `writeMask` parameter in the last call to vkCmdSetStencilWriteMask **must** be `0`
- VUID-vkCmdDrawIndirectByteCountEXT-None-09116
  If a shader object is bound to any graphics stage or the currently bound graphics pipeline

was created with VK_DYNAMIC_STATE_COLOR_WRITE_MASK_EXT, and the format of any color attachment is VK_FORMAT_E5B9G9R9_UFLOAT_PACK32, the corresponding element of the pColorWriteMasks parameter of vkCmdSetColorWriteMaskEXT **must** either include all of VK_COLOR_COMPONENT_R_BIT, VK_COLOR_COMPONENT_G_BIT, and VK_COLOR_COMPONENT_B_BIT, or none of them

- VUID-vkCmdDrawIndirectByteCountEXT-maxFragmentDualSrcAttachments-09239
  If blending is enabled for any attachment where either the source or destination blend factors for that attachment use the secondary color input, the maximum value of Location for any output attachment statically used in the Fragment Execution Model executed by this command **must** be less than maxFragmentDualSrcAttachments
- VUID-vkCmdDrawIndirectByteCountEXT-None-09548
  If the current render pass was begun with vkCmdBeginRenderingKHR, and there is no shader object bound to any graphics stage, the value of each element of VkRenderingAttachmentLocationInfoKHR::pColorAttachmentLocations set by vkCmdSetRenderingAttachmentLocationsKHR **must** match the value set for the corresponding element in the currently bound pipeline
- VUID-vkCmdDrawIndirectByteCountEXT-None-09549
  If the current render pass was begun with vkCmdBeginRenderingKHR, and there is no shader object bound to any graphics stage, input attachment index mappings in the currently bound pipeline **must** match those set for the current render pass instance via VkRenderingInputAttachmentIndexInfoKHR
- VUID-vkCmdDrawIndirectByteCountEXT-None-09642
  If the current render pass was begun with vkCmdBeginRenderingKHR with the VK_RENDERING_ENABLE_LEGACY_DITHERING_BIT_EXT flag, the bound graphics pipeline **must** have been created with VK_PIPELINE_CREATE_2_ENABLE_LEGACY_DITHERING_BIT_EXT
- VUID-vkCmdDrawIndirectByteCountEXT-None-09643
  If the bound graphics pipeline was created with VK_PIPELINE_CREATE_2_ENABLE_LEGACY_DITHERING_BIT_EXT, the current render pass **must** have begun with vkCmdBeginRenderingKHR with the VK_RENDERING_ENABLE_LEGACY_DITHERING_BIT_EXT flag
- VUID-vkCmdDrawIndirectByteCountEXT-None-04007
  All vertex input bindings accessed via vertex input variables declared in the vertex shader entry point's interface **must** have either valid or VK_NULL_HANDLE buffers bound
- VUID-vkCmdDrawIndirectByteCountEXT-None-04008
  If the nullDescriptor feature is not enabled, all vertex input bindings accessed via vertex input variables declared in the vertex shader entry point's interface **must** not be VK_NULL_HANDLE
- VUID-vkCmdDrawIndirectByteCountEXT-None-02721
  If robustBufferAccess is not enabled, and that pipeline was created without enabling VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_EXT for vertexInputs, then for a given vertex buffer binding, any attribute data fetched **must** be entirely contained within the corresponding vertex buffer binding, as described in Vertex Input Description
- VUID-vkCmdDrawIndirectByteCountEXT-None-07842

If there is a shader object bound to the VK_SHADER_STAGE_VERTEX_BIT stage or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_PRIMITIVE_TOPOLOGY dynamic state enabled then vkCmdSetPrimitiveTopologyEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirectByteCountEXT-dynamicPrimitiveTopologyUnrestricted-07500
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_PRIMITIVE_TOPOLOGY dynamic state enabled and the dynamicPrimitiveTopologyUnrestricted is VK_FALSE, then the primitiveTopology parameter of vkCmdSetPrimitiveTopologyEXT **must** be of the same topology class as the pipeline VkPipelineInputAssemblyStateCreateInfo::topology state
- VUID-vkCmdDrawIndirectByteCountEXT-pStrides-04913
  If the bound graphics pipeline was created with the VK_DYNAMIC_STATE_VERTEX_INPUT_BINDING_STRIDE_EXT dynamic state enabled, but without the VK_DYNAMIC_STATE_VERTEX_INPUT_EXT dynamic state enabled, then vkCmdBindVertexBuffers2EXT **must** have been called and not subsequently invalidated in the current command buffer prior to this draw command, and the pStrides parameter of vkCmdBindVertexBuffers2EXT **must** not be NULL
- VUID-vkCmdDrawIndirectByteCountEXT-None-04914
  If there is a shader object bound to the VK_SHADER_STAGE_VERTEX_BIT stage or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VERTEX_INPUT_EXT dynamic state enabled then vkCmdSetVertexInputEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this draw command
- VUID-vkCmdDrawIndirectByteCountEXT-Input-07939
  If there is a shader object bound to the VK_SHADER_STAGE_VERTEX_BIT stage or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VERTEX_INPUT_EXT dynamic state enabled then all variables with the Input storage class decorated with Location in the Vertex Execution Model OpEntryPoint **must** contain a location in VkVertexInputAttributeDescription2EXT::location
- VUID-vkCmdDrawIndirectByteCountEXT-Input-08734
  If there is a shader object bound to the VK_SHADER_STAGE_VERTEX_BIT stage or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VERTEX_INPUT_EXT dynamic state enabled and either legacyVertexAttributes is not enabled or the SPIR-V Type associated with a given Input variable of the corresponding Location in the Vertex Execution Model OpEntryPoint is 64-bit, then the numeric type associated with all Input variables of the corresponding Location in the Vertex Execution Model OpEntryPoint **must** be the same as VkVertexInputAttributeDescription2EXT::format
- VUID-vkCmdDrawIndirectByteCountEXT-format-08936
  If there is a shader object bound to the VK_SHADER_STAGE_VERTEX_BIT stage or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VERTEX_INPUT_EXT dynamic state enabled and VkVertexInputAttributeDescription2EXT::format has a 64-bit component, then the scalar width associated with all Input variables of the corresponding Location in the Vertex Execution Model OpEntryPoint **must** be 64-bit
- VUID-vkCmdDrawIndirectByteCountEXT-format-08937
  If there is a shader object bound to the VK_SHADER_STAGE_VERTEX_BIT stage or the bound

graphics pipeline state was created with the VK_DYNAMIC_STATE_VERTEX_INPUT_EXT dynamic state enabled and the scalar width associated with a Location decorated Input variable in the Vertex Execution Model OpEntryPoint is 64-bit, then the corresponding VkVertexInputAttributeDescription2EXT::format **must** have a 64-bit component

- VUID-vkCmdDrawIndirectByteCountEXT-None-09203
  If there is a shader object bound to the VK_SHADER_STAGE_VERTEX_BIT stage or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VERTEX_INPUT_EXT dynamic state enabled and VkVertexInputAttributeDescription2EXT::format has a 64-bit component, then all Input variables at the corresponding Location in the Vertex Execution Model OpEntryPoint **must** not use components that are not present in the format

- VUID-vkCmdDrawIndirectByteCountEXT-None-04875
  If there is a shader object bound to the VK_SHADER_STAGE_VERTEX_BIT stage and the most recent call to vkCmdSetPrimitiveTopologyEXT in the current command buffer set primitiveTopology to VK_PRIMITIVE_TOPOLOGY_PATCH_LIST, or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_PATCH_CONTROL_POINTS_EXT dynamic state enabled then vkCmdSetPatchControlPointsEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirectByteCountEXT-None-04879
  If there is a shader object bound to the VK_SHADER_STAGE_VERTEX_BIT stage or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_PRIMITIVE_RESTART_ENABLE dynamic state enabled then vkCmdSetPrimitiveRestartEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirectByteCountEXT-None-09637
  If the primitiveTopologyListRestart feature is not enabled, the topology is VK_PRIMITIVE_TOPOLOGY_POINT_LIST, VK_PRIMITIVE_TOPOLOGY_LINE_LIST, VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST, VK_PRIMITIVE_TOPOLOGY_LINE_LIST_WITH_ADJACENCY, or VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST_WITH_ADJACENCY, there is a shader object bound to the VK_SHADER_STAGE_VERTEX_BIT stage or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_PRIMITIVE_RESTART_ENABLE dynamic state enabled then vkCmdSetPrimitiveRestartEnableEXT **must** be VK_FALSE

- VUID-vkCmdDrawIndirectByteCountEXT-stage-06481
  The bound graphics pipeline **must** not have been created with the VkPipelineShaderStageCreateInfo::stage member of an element of VkGraphicsPipelineCreateInfo::pStages set to VK_SHADER_STAGE_TASK_BIT_EXT or VK_SHADER_STAGE_MESH_BIT_EXT

- VUID-vkCmdDrawIndirectByteCountEXT-None-08885
  There **must** be no shader object bound to either of the VK_SHADER_STAGE_TASK_BIT_EXT or VK_SHADER_STAGE_MESH_BIT_EXT stages

- VUID-vkCmdDrawIndirectByteCountEXT-pNext-09461
  If the bound graphics pipeline state was created with VkPipelineVertexInputDivisorStateCreateInfoKHR in the pNext chain of VkGraphicsPipelineCreateInfo::pVertexInputState, any member of VkPipelineVertexInputDivisorStateCreateInfoKHR::pVertexBindingDivisors has a value other than 1 in divisor, and VkPhysicalDeviceVertexAttributeDivisorPropertiesKHR

::supportsNonZeroFirstInstance is VK_FALSE, then firstInstance **must** be 0

- VUID-vkCmdDrawIndirectByteCountEXT-None-09462
  If shader objects are used for drawing or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VERTEX_INPUT_EXT dynamic state enabled, any member of the pVertexBindingDescriptions parameter to the vkCmdSetVertexInputEXT call that sets this dynamic state has a value other than 1 in divisor, and VkPhysicalDeviceVertexAttributeDivisorPropertiesKHR::supportsNonZeroFirstInstance is VK_FALSE, then firstInstance **must** be 0

- VUID-vkCmdDrawIndirectByteCountEXT-transformFeedback-02287
  VkPhysicalDeviceTransformFeedbackFeaturesEXT::transformFeedback **must** be enabled
- VUID-vkCmdDrawIndirectByteCountEXT-transformFeedbackDraw-02288
  The implementation **must** support VkPhysicalDeviceTransformFeedbackPropertiesEXT::transformFeedbackDraw
- VUID-vkCmdDrawIndirectByteCountEXT-vertexStride-02289
  vertexStride **must** be greater than 0 and less than or equal to VkPhysicalDeviceTransformFeedbackPropertiesEXT::maxTransformFeedbackBufferDataStride
- VUID-vkCmdDrawIndirectByteCountEXT-counterBuffer-04567
  If counterBuffer is non-sparse then it **must** be bound completely and contiguously to a single VkDeviceMemory object
- VUID-vkCmdDrawIndirectByteCountEXT-counterBuffer-02290
  counterBuffer **must** have been created with the VK_BUFFER_USAGE_INDIRECT_BUFFER_BIT bit set
- VUID-vkCmdDrawIndirectByteCountEXT-counterBufferOffset-04568
  counterBufferOffset **must** be a multiple of 4
- VUID-vkCmdDrawIndirectByteCountEXT-counterOffset-09474
  counterOffset **must** be a multiple of 4
- VUID-vkCmdDrawIndirectByteCountEXT-vertexStride-09475
  vertexStride **must** be a multiple of 4
- VUID-vkCmdDrawIndirectByteCountEXT-commandBuffer-02646
  commandBuffer **must** not be a protected command buffer

## Valid Usage (Implicit)

- VUID-vkCmdDrawIndirectByteCountEXT-commandBuffer-parameter
  commandBuffer **must** be a valid VkCommandBuffer handle
- VUID-vkCmdDrawIndirectByteCountEXT-counterBuffer-parameter
  counterBuffer **must** be a valid VkBuffer handle
- VUID-vkCmdDrawIndirectByteCountEXT-commandBuffer-recording
  commandBuffer **must** be in the recording state
- VUID-vkCmdDrawIndirectByteCountEXT-commandBuffer-cmdpool
  The VkCommandPool that commandBuffer was allocated from **must** support graphics

operations

- VUID-vkCmdDrawIndirectByteCountEXT-renderpass
  This command **must** only be called inside of a render pass instance
- VUID-vkCmdDrawIndirectByteCountEXT-videocoding
  This command **must** only be called outside of a video coding scope
- VUID-vkCmdDrawIndirectByteCountEXT-commonparent
  Both of `commandBuffer`, and `counterBuffer` **must** have been created, allocated, or retrieved from the same VkDevice

### Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

### Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Inside | Outside | Graphics | Action |

## 21.4. Conditional Rendering

Certain rendering commands **can** be executed conditionally based on a value in buffer memory. These rendering commands are limited to drawing commands, dispatching commands, and clearing attachments with vkCmdClearAttachments within a conditional rendering block which is defined by commands vkCmdBeginConditionalRenderingEXT and vkCmdEndConditionalRenderingEXT. Other rendering commands remain unaffected by conditional rendering.

After beginning conditional rendering, it is considered *active* within the command buffer it was called until it is ended with vkCmdEndConditionalRenderingEXT.

Conditional rendering **must** begin and end in the same command buffer. When conditional rendering is active, a primary command buffer **can** execute secondary command buffers if the `inheritedConditionalRendering` feature is enabled. For a secondary command buffer to be executed while conditional rendering is active in the primary command buffer, it **must** set the `conditionalRenderingEnable` flag of VkCommandBufferInheritanceConditionalRenderingInfoEXT, as described in the Command Buffer Recording section.

Conditional rendering **must** also either begin and end inside the same subpass of a render pass instance, or **must** both begin and end outside of a render pass instance (i.e. contain entire render

pass instances).

To begin conditional rendering, call:

```
// Provided by VK_EXT_conditional_rendering
void vkCmdBeginConditionalRenderingEXT(
    VkCommandBuffer                             commandBuffer,
    const VkConditionalRenderingBeginInfoEXT*   pConditionalRenderingBegin);
```

- `commandBuffer` is the command buffer into which this command will be recorded.
- `pConditionalRenderingBegin` is a pointer to a VkConditionalRenderingBeginInfoEXT structure specifying parameters of conditional rendering.

## Valid Usage

- VUID-vkCmdBeginConditionalRenderingEXT-None-01980
  Conditional rendering **must** not already be active

## Valid Usage (Implicit)

- VUID-vkCmdBeginConditionalRenderingEXT-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdBeginConditionalRenderingEXT-pConditionalRenderingBegin-parameter
  `pConditionalRenderingBegin` **must** be a valid pointer to a valid VkConditionalRenderingBeginInfoEXT structure
- VUID-vkCmdBeginConditionalRenderingEXT-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdBeginConditionalRenderingEXT-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics, or compute operations
- VUID-vkCmdBeginConditionalRenderingEXT-videocoding
  This command **must** only be called outside of a video coding scope

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics<br>Compute | Action<br>State |

The VkConditionalRenderingBeginInfoEXT structure is defined as:

```
// Provided by VK_EXT_conditional_rendering
typedef struct VkConditionalRenderingBeginInfoEXT {
    VkStructureType                    sType;
    const void*                        pNext;
    VkBuffer                           buffer;
    VkDeviceSize                       offset;
    VkConditionalRenderingFlagsEXT     flags;
} VkConditionalRenderingBeginInfoEXT;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- buffer is a buffer containing the predicate for conditional rendering.
- offset is the byte offset into buffer where the predicate is located.
- flags is a bitmask of VkConditionalRenderingFlagsEXT specifying the behavior of conditional rendering.

If the 32-bit value at offset in buffer memory is zero, then the rendering commands are discarded, otherwise they are executed as normal. If the value of the predicate in buffer memory changes while conditional rendering is active, the rendering commands **may** be discarded in an implementation-dependent way. Some implementations may latch the value of the predicate upon beginning conditional rendering while others may read it before every rendering command.

## Valid Usage

- VUID-VkConditionalRenderingBeginInfoEXT-buffer-01981
  If buffer is non-sparse then it **must** be bound completely and contiguously to a single VkDeviceMemory object
- VUID-VkConditionalRenderingBeginInfoEXT-buffer-01982
  buffer **must** have been created with the VK_BUFFER_USAGE_CONDITIONAL_RENDERING_BIT_EXT bit set
- VUID-VkConditionalRenderingBeginInfoEXT-offset-01983
  offset **must** be less than the size of buffer by at least 32 bits
- VUID-VkConditionalRenderingBeginInfoEXT-offset-01984
  offset **must** be a multiple of 4

### Valid Usage (Implicit)

- VUID-VkConditionalRenderingBeginInfoEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_CONDITIONAL_RENDERING_BEGIN_INFO_EXT`
- VUID-VkConditionalRenderingBeginInfoEXT-pNext-pNext
  `pNext` **must** be `NULL`
- VUID-VkConditionalRenderingBeginInfoEXT-buffer-parameter
  `buffer` **must** be a valid VkBuffer handle
- VUID-VkConditionalRenderingBeginInfoEXT-flags-parameter
  `flags` **must** be a valid combination of VkConditionalRenderingFlagBitsEXT values

Bits which **can** be set in vkCmdBeginConditionalRenderingEXT::`flags`, specifying the behavior of conditional rendering, are:

```
// Provided by VK_EXT_conditional_rendering
typedef enum VkConditionalRenderingFlagBitsEXT {
    VK_CONDITIONAL_RENDERING_INVERTED_BIT_EXT = 0x00000001,
} VkConditionalRenderingFlagBitsEXT;
```

- `VK_CONDITIONAL_RENDERING_INVERTED_BIT_EXT` specifies the condition used to determine whether to discard rendering commands or not. That is, if the 32-bit predicate read from `buffer` memory at `offset` is zero, the rendering commands are not discarded, and if non zero, then they are discarded.

```
// Provided by VK_EXT_conditional_rendering
typedef VkFlags VkConditionalRenderingFlagsEXT;
```

`VkConditionalRenderingFlagsEXT` is a bitmask type for setting a mask of zero or more VkConditionalRenderingFlagBitsEXT.

To end conditional rendering, call:

```
// Provided by VK_EXT_conditional_rendering
void vkCmdEndConditionalRenderingEXT(
    VkCommandBuffer                             commandBuffer);
```

- `commandBuffer` is the command buffer into which this command will be recorded.

Once ended, conditional rendering becomes inactive.

### Valid Usage

- VUID-vkCmdEndConditionalRenderingEXT-None-01985

Conditional rendering **must** be active

- VUID-vkCmdEndConditionalRenderingEXT-None-01986
  If conditional rendering was made active outside of a render pass instance, it **must** not be ended inside a render pass instance
- VUID-vkCmdEndConditionalRenderingEXT-None-01987
  If conditional rendering was made active within a subpass it **must** be ended in the same subpass

### Valid Usage (Implicit)

- VUID-vkCmdEndConditionalRenderingEXT-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdEndConditionalRenderingEXT-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdEndConditionalRenderingEXT-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics, or compute operations
- VUID-vkCmdEndConditionalRenderingEXT-videocoding
  This command **must** only be called outside of a video coding scope

### Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

### Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics<br>Compute | Action<br>State |

## 21.5. Programmable Mesh Shading

In this drawing approach, primitives are assembled by the mesh shader stage. Mesh shading operates similarly to dispatching compute as the shaders make use of workgroups.

To record a mesh tasks drawing command, call:

```
// Provided by VK_NV_mesh_shader
void vkCmdDrawMeshTasksNV(
    VkCommandBuffer                             commandBuffer,
    uint32_t                                    taskCount,
    uint32_t                                    firstTask);
```

- commandBuffer is the command buffer into which the command will be recorded.
- taskCount is the number of local workgroups to dispatch in the X dimension. Y and Z dimension are implicitly set to one.
- firstTask is the X component of the first workgroup ID.

When the command is executed, a global workgroup consisting of taskCount local workgroups is assembled.

## Valid Usage

- VUID-vkCmdDrawMeshTasksNV-magFilter-04553
  If a VkSampler created with magFilter or minFilter equal to VK_FILTER_LINEAR, reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE, and compareEnable equal to VK_FALSE is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT
- VUID-vkCmdDrawMeshTasksNV-magFilter-09598
  If a VkSampler created with magFilter or minFilter equal to VK_FILTER_LINEAR and reductionMode equal to either VK_SAMPLER_REDUCTION_MODE_MIN or VK_SAMPLER_REDUCTION_MODE_MAX is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT
- VUID-vkCmdDrawMeshTasksNV-mipmapMode-04770
  If a VkSampler created with mipmapMode equal to VK_SAMPLER_MIPMAP_MODE_LINEAR, reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE, and compareEnable equal to VK_FALSE is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT
- VUID-vkCmdDrawMeshTasksNV-mipmapMode-09599
  If a VkSampler created with mipmapMode equal to VK_SAMPLER_MIPMAP_MODE_LINEAR and reductionMode equal to either VK_SAMPLER_REDUCTION_MODE_MIN or VK_SAMPLER_REDUCTION_MODE_MAX is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT
- VUID-vkCmdDrawMeshTasksNV-unnormalizedCoordinates-09635
  If a VkSampler created with unnormalizedCoordinates equal to VK_TRUE is used to sample a VkImageView as a result of this command, then the image view's levelCount and layerCount **must** be 1

- VUID-vkCmdDrawMeshTasksNV-unnormalizedCoordinates-09636
  If a VkSampler created with unnormalizedCoordinates equal to VK_TRUE is used to sample a VkImageView as a result of this command, then the image view's viewType **must** be VK_IMAGE_VIEW_TYPE_1D or VK_IMAGE_VIEW_TYPE_2D
- VUID-vkCmdDrawMeshTasksNV-None-06479
  If a VkImageView is sampled with depth comparison, the image view's format features **must** contain VK_FORMAT_FEATURE_2_SAMPLED_IMAGE_DEPTH_COMPARISON_BIT
- VUID-vkCmdDrawMeshTasksNV-None-02691
  If a VkImageView is accessed using atomic operations as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_STORAGE_IMAGE_ATOMIC_BIT
- VUID-vkCmdDrawMeshTasksNV-None-07888
  If a VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER descriptor is accessed using atomic operations as a result of this command, then the storage texel buffer's format features **must** contain VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_ATOMIC_BIT
- VUID-vkCmdDrawMeshTasksNV-None-02692
  If a VkImageView is sampled with VK_FILTER_CUBIC_EXT as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_CUBIC_BIT_EXT
- VUID-vkCmdDrawMeshTasksNV-None-02693
  If the VK_EXT_filter_cubic extension is not enabled and any VkImageView is sampled with VK_FILTER_CUBIC_EXT as a result of this command, it **must** not have a VkImageViewType of VK_IMAGE_VIEW_TYPE_3D, VK_IMAGE_VIEW_TYPE_CUBE, or VK_IMAGE_VIEW_TYPE_CUBE_ARRAY
- VUID-vkCmdDrawMeshTasksNV-filterCubic-02694
  Any VkImageView being sampled with VK_FILTER_CUBIC_EXT as a result of this command **must** have a VkImageViewType and format that supports cubic filtering, as specified by VkFilterCubicImageViewImageFormatPropertiesEXT::filterCubic returned by vkGetPhysicalDeviceImageFormatProperties2
- VUID-vkCmdDrawMeshTasksNV-filterCubicMinmax-02695
  Any VkImageView being sampled with VK_FILTER_CUBIC_EXT with a reduction mode of either VK_SAMPLER_REDUCTION_MODE_MIN or VK_SAMPLER_REDUCTION_MODE_MAX as a result of this command **must** have a VkImageViewType and format that supports cubic filtering together with minmax filtering, as specified by VkFilterCubicImageViewImageFormatPropertiesEXT::filterCubicMinmax returned by vkGetPhysicalDeviceImageFormatProperties2
- VUID-vkCmdDrawMeshTasksNV-cubicRangeClamp-09212
  If the cubicRangeClamp feature is not enabled, then any VkImageView being sampled with VK_FILTER_CUBIC_EXT as a result of this command **must** not have a VkSamplerReductionModeCreateInfo::reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE_RANGECLAMP_QCOM
- VUID-vkCmdDrawMeshTasksNV-reductionMode-09213
  Any VkImageView being sampled with a VkSamplerReductionModeCreateInfo::reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE_RANGECLAMP_QCOM as a result of this command **must** sample with VK_FILTER_CUBIC_EXT

- VUID-vkCmdDrawMeshTasksNV-selectableCubicWeights-09214
  If the `selectableCubicWeights` feature is not enabled, then any VkImageView being sampled with `VK_FILTER_CUBIC_EXT` as a result of this command **must** have VkSamplerCubicWeightsCreateInfoQCOM::`cubicWeights` equal to `VK_CUBIC_FILTER_WEIGHTS_CATMULL_ROM_QCOM`
- VUID-vkCmdDrawMeshTasksNV-flags-02696
  Any VkImage created with a VkImageCreateInfo::`flags` containing `VK_IMAGE_CREATE_CORNER_SAMPLED_BIT_NV` sampled as a result of this command **must** only be sampled using a VkSamplerAddressMode of `VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_EDGE`
- VUID-vkCmdDrawMeshTasksNV-OpTypeImage-07027
  For any VkImageView being written as a storage image where the image format field of the `OpTypeImage` is `Unknown`, the view's format features **must** contain `VK_FORMAT_FEATURE_2_STORAGE_WRITE_WITHOUT_FORMAT_BIT`
- VUID-vkCmdDrawMeshTasksNV-OpTypeImage-07028
  For any VkImageView being read as a storage image where the image format field of the `OpTypeImage` is `Unknown`, the view's format features **must** contain `VK_FORMAT_FEATURE_2_STORAGE_READ_WITHOUT_FORMAT_BIT`
- VUID-vkCmdDrawMeshTasksNV-OpTypeImage-07029
  For any VkBufferView being written as a storage texel buffer where the image format field of the `OpTypeImage` is `Unknown`, the view's buffer features **must** contain `VK_FORMAT_FEATURE_2_STORAGE_WRITE_WITHOUT_FORMAT_BIT`
- VUID-vkCmdDrawMeshTasksNV-OpTypeImage-07030
  Any VkBufferView being read as a storage texel buffer where the image format field of the `OpTypeImage` is `Unknown` then the view's buffer features **must** contain `VK_FORMAT_FEATURE_2_STORAGE_READ_WITHOUT_FORMAT_BIT`
- VUID-vkCmdDrawMeshTasksNV-None-08600
  For each set *n* that is statically used by a bound shader, a descriptor set **must** have been bound to *n* at the same pipeline bind point, with a VkPipelineLayout that is compatible for set *n*, with the VkPipelineLayout used to create the current VkPipeline or the VkDescriptorSetLayout array used to create the current VkShaderEXT , as described in Pipeline Layout Compatibility
- VUID-vkCmdDrawMeshTasksNV-None-08601
  For each push constant that is statically used by a bound shader, a push constant value **must** have been set for the same pipeline bind point, with a VkPipelineLayout that is compatible for push constants, with the VkPipelineLayout used to create the current VkPipeline or the VkDescriptorSetLayout array used to create the current VkShaderEXT , as described in Pipeline Layout Compatibility
- VUID-vkCmdDrawMeshTasksNV-None-10068
  For each array of resources that is used by a bound shader, the indices used to access members of the array **must** be less than the descriptor count for the identified binding in the descriptor sets used by this command
- VUID-vkCmdDrawMeshTasksNV-maintenance4-08602
  If the `maintenance4` feature is not enabled, then for each push constant that is statically used by a bound shader, a push constant value **must** have been set for the same pipeline

bind point, with a VkPipelineLayout that is compatible for push constants, with the VkPipelineLayout used to create the current VkPipeline or the VkDescriptorSetLayout and VkPushConstantRange arrays used to create the current VkShaderEXT , as described in Pipeline Layout Compatibility

- VUID-vkCmdDrawMeshTasksNV-None-08114
  Descriptors in each bound descriptor set, specified via vkCmdBindDescriptorSets, **must** be valid as described by descriptor validity if they are statically used by the VkPipeline bound to the pipeline bind point used by this command and the bound VkPipeline was not created with `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`
- VUID-vkCmdDrawMeshTasksNV-None-08115
  If the descriptors used by the VkPipeline bound to the pipeline bind point were specified via vkCmdBindDescriptorSets, the bound VkPipeline **must** have been created without `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`
- VUID-vkCmdDrawMeshTasksNV-None-08116
  Descriptors in bound descriptor buffers, specified via vkCmdSetDescriptorBufferOffsetsEXT, **must** be valid if they are dynamically used by the VkPipeline bound to the pipeline bind point used by this command and the bound VkPipeline was created with `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`
- VUID-vkCmdDrawMeshTasksNV-None-08604
  Descriptors in bound descriptor buffers, specified via vkCmdSetDescriptorBufferOffsetsEXT, **must** be valid if they are dynamically used by any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command
- VUID-vkCmdDrawMeshTasksNV-None-08117
  If the descriptors used by the VkPipeline bound to the pipeline bind point were specified via vkCmdSetDescriptorBufferOffsetsEXT, the bound VkPipeline **must** have been created with `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`
- VUID-vkCmdDrawMeshTasksNV-None-08119
  If a descriptor is dynamically used with a VkPipeline created with `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`, the descriptor memory **must** be resident
- VUID-vkCmdDrawMeshTasksNV-None-08605
  If a descriptor is dynamically used with a VkShaderEXT created with a `VkDescriptorSetLayout` that was created with `VK_DESCRIPTOR_SET_LAYOUT_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`, the descriptor memory **must** be resident
- VUID-vkCmdDrawMeshTasksNV-None-08606
  If the `shaderObject` feature is not enabled, a valid pipeline **must** be bound to the pipeline bind point used by this command
- VUID-vkCmdDrawMeshTasksNV-None-08608
  If a pipeline is bound to the pipeline bind point used by this command, there **must** not have been any calls to dynamic state setting commands for any state specified statically in the VkPipeline object bound to the pipeline bind point used by this command, since that pipeline was bound
- VUID-vkCmdDrawMeshTasksNV-None-08609

If the VkPipeline object bound to the pipeline bind point used by this command or any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a VkSampler object that uses unnormalized coordinates, that sampler **must** not be used to sample from any VkImage with a VkImageView of the type VK_IMAGE_VIEW_TYPE_3D, VK_IMAGE_VIEW_TYPE_CUBE, VK_IMAGE_VIEW_TYPE_1D_ARRAY, VK_IMAGE_VIEW_TYPE_2D_ARRAY or VK_IMAGE_VIEW_TYPE_CUBE_ARRAY, in any shader stage

- VUID-vkCmdDrawMeshTasksNV-None-08610
  If the VkPipeline object bound to the pipeline bind point used by this command or any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a VkSampler object that uses unnormalized coordinates, that sampler **must** not be used with any of the SPIR-V OpImageSample* or OpImageSparseSample* instructions with ImplicitLod, Dref or Proj in their name, in any shader stage
- VUID-vkCmdDrawMeshTasksNV-None-08611
  If the VkPipeline object bound to the pipeline bind point used by this command or any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a VkSampler object that uses unnormalized coordinates, that sampler **must** not be used with any of the SPIR-V OpImageSample* or OpImageSparseSample* instructions that includes a LOD bias or any offset values, in any shader stage
- VUID-vkCmdDrawMeshTasksNV-None-08607
  If the shaderObject is enabled, either a valid pipeline **must** be bound to the pipeline bind point used by this command, or a valid combination of valid and VK_NULL_HANDLE shader objects **must** be bound to every supported shader stage corresponding to the pipeline bind point used by this command
- VUID-vkCmdDrawMeshTasksNV-uniformBuffers-06935
  If any stage of the VkPipeline object bound to the pipeline bind point used by this command accesses a uniform buffer, and that stage was created without enabling either VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_EXT or VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_2_EXT for uniformBuffers, and the robustBufferAccess feature is not enabled, that stage **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point
- VUID-vkCmdDrawMeshTasksNV-None-08612
  If the robustBufferAccess feature is not enabled, and any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a uniform buffer, it **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point
- VUID-vkCmdDrawMeshTasksNV-storageBuffers-06936
  If any stage of the VkPipeline object bound to the pipeline bind point used by this command accesses a storage buffer, and that stage was created without enabling either VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_EXT or VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_2_EXT for storageBuffers, and the robustBufferAccess feature is not enabled, that stage **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point
- VUID-vkCmdDrawMeshTasksNV-None-08613

If the robustBufferAccess feature is not enabled, and any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a storage buffer, it **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point

- VUID-vkCmdDrawMeshTasksNV-commandBuffer-02707
  If commandBuffer is an unprotected command buffer and protectedNoFault is not supported, any resource accessed by bound shaders **must** not be a protected resource
- VUID-vkCmdDrawMeshTasksNV-None-06550
  If a bound shader accesses a VkSampler or VkImageView object that enables sampler Y'C$_B$C$_R$ conversion, that object **must** only be used with OpImageSample* or OpImageSparseSample* instructions
- VUID-vkCmdDrawMeshTasksNV-ConstOffset-06551
  If a bound shader accesses a VkSampler or VkImageView object that enables sampler Y'C$_B$C$_R$ conversion, that object **must** not use the ConstOffset and Offset operands
- VUID-vkCmdDrawMeshTasksNV-viewType-07752
  If a VkImageView is accessed as a result of this command, then the image view's viewType **must** match the Dim operand of the OpTypeImage as described in Compatibility Between SPIR-V Image Dimensions and Vulkan ImageView Types
- VUID-vkCmdDrawMeshTasksNV-format-07753
  If a VkImageView is accessed as a result of this command, then the numeric type of the image view's format and the Sampled Type operand of the OpTypeImage **must** match
- VUID-vkCmdDrawMeshTasksNV-OpImageWrite-08795
  If a VkImageView created with a format other than VK_FORMAT_A8_UNORM_KHR is accessed using OpImageWrite as a result of this command, then the Type of the Texel operand of that instruction **must** have at least as many components as the image view's format
- VUID-vkCmdDrawMeshTasksNV-OpImageWrite-08796
  If a VkImageView created with the format VK_FORMAT_A8_UNORM_KHR is accessed using OpImageWrite as a result of this command, then the Type of the Texel operand of that instruction **must** have four components
- VUID-vkCmdDrawMeshTasksNV-OpImageWrite-04469
  If a VkBufferView is accessed using OpImageWrite as a result of this command, then the Type of the Texel operand of that instruction **must** have at least as many components as the buffer view's format
- VUID-vkCmdDrawMeshTasksNV-SampledType-04470
  If a VkImageView with a VkFormat that has a 64-bit component width is accessed as a result of this command, the SampledType of the OpTypeImage operand of that instruction **must** have a Width of 64
- VUID-vkCmdDrawMeshTasksNV-SampledType-04471
  If a VkImageView with a VkFormat that has a component width less than 64-bit is accessed as a result of this command, the SampledType of the OpTypeImage operand of that instruction **must** have a Width of 32
- VUID-vkCmdDrawMeshTasksNV-SampledType-04472
  If a VkBufferView with a VkFormat that has a 64-bit component width is accessed as a

result of this command, the SampledType of the OpTypeImage operand of that instruction **must** have a Width of 64

- VUID-vkCmdDrawMeshTasksNV-SampledType-04473
  If a VkBufferView with a VkFormat that has a component width less than 64-bit is accessed as a result of this command, the SampledType of the OpTypeImage operand of that instruction **must** have a Width of 32
- VUID-vkCmdDrawMeshTasksNV-sparseImageInt64Atomics-04474
  If the sparseImageInt64Atomics feature is not enabled, VkImage objects created with the VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT flag **must** not be accessed by atomic instructions through an OpTypeImage with a SampledType with a Width of 64 by this command
- VUID-vkCmdDrawMeshTasksNV-sparseImageInt64Atomics-04475
  If the sparseImageInt64Atomics feature is not enabled, VkBuffer objects created with the VK_BUFFER_CREATE_SPARSE_RESIDENCY_BIT flag **must** not be accessed by atomic instructions through an OpTypeImage with a SampledType with a Width of 64 by this command
- VUID-vkCmdDrawMeshTasksNV-OpImageWeightedSampleQCOM-06971
  If OpImageWeightedSampleQCOM is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_WEIGHT_SAMPLED_IMAGE_BIT_QCOM
- VUID-vkCmdDrawMeshTasksNV-OpImageWeightedSampleQCOM-06972
  If OpImageWeightedSampleQCOM uses a VkImageView as a sample weight image as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_WEIGHT_IMAGE_BIT_QCOM
- VUID-vkCmdDrawMeshTasksNV-OpImageBoxFilterQCOM-06973
  If OpImageBoxFilterQCOM is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_BOX_FILTER_SAMPLED_BIT_QCOM
- VUID-vkCmdDrawMeshTasksNV-OpImageBlockMatchSSDQCOM-06974
  If OpImageBlockMatchSSDQCOM is used to read from an VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM
- VUID-vkCmdDrawMeshTasksNV-OpImageBlockMatchSADQCOM-06975
  If OpImageBlockMatchSADQCOM is used to read from an VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM
- VUID-vkCmdDrawMeshTasksNV-OpImageBlockMatchSADQCOM-06976
  If OpImageBlockMatchSADQCOM or OpImageBlockMatchSSDQCOM is used to read from a reference image as result of this command, then the specified reference coordinates **must** not fail integer texel coordinate validation
- VUID-vkCmdDrawMeshTasksNV-OpImageWeightedSampleQCOM-06977
  If OpImageWeightedSampleQCOM, OpImageBoxFilterQCOM, OpImageBlockMatchWindowSSDQCOM, OpImageBlockMatchWindowSADQCOM, OpImageBlockMatchGatherSSDQCOM, OpImageBlockMatchGatherSADQCOM, OpImageBlockMatchSSDQCOM, or OpImageBlockMatchSADQCOM uses a VkSampler as a result of this command, then the sampler **must** have been created with VK_SAMPLER_CREATE_IMAGE_PROCESSING_BIT_QCOM

- VUID-vkCmdDrawMeshTasksNV-OpImageWeightedSampleQCOM-06978
  If any command other than OpImageWeightedSampleQCOM, OpImageBoxFilterQCOM, OpImageBlockMatchWindowSSDQCOM, OpImageBlockMatchWindowSADQCOM, OpImageBlockMatchGatherSSDQCOM, OpImageBlockMatchGatherSADQCOM, OpImageBlockMatchSSDQCOM, or OpImageBlockMatchSADQCOM uses a VkSampler as a result of this command, then the sampler **must** not have been created with VK_SAMPLER_CREATE_IMAGE_PROCESSING_BIT_QCOM
- VUID-vkCmdDrawMeshTasksNV-OpImageBlockMatchWindow-09215
  If a OpImageBlockMatchWindow*QCOM or OpImageBlockMatchGather*QCOM instruction is used to read from an VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM
- VUID-vkCmdDrawMeshTasksNV-OpImageBlockMatchWindow-09216
  If a OpImageBlockMatchWindow*QCOM or OpImageBlockMatchGather*QCOM instruction is used to read from an VkImageView as a result of this command, then the image view's format **must** be a single-component format
- VUID-vkCmdDrawMeshTasksNV-OpImageBlockMatchWindow-09217
  If a OpImageBlockMatchWindow*QCOM or OpImageBlockMatchGather*QCOM read from a reference image as result of this command, then the specified reference coordinates **must** not fail integer texel coordinate validation
- VUID-vkCmdDrawMeshTasksNV-None-07288
  Any shader invocation executed by this command **must** terminate
- VUID-vkCmdDrawMeshTasksNV-None-09600
  If a descriptor with type equal to any of VK_DESCRIPTOR_TYPE_SAMPLE_WEIGHT_IMAGE_QCOM, VK_DESCRIPTOR_TYPE_BLOCK_MATCH_IMAGE_QCOM, VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE, VK_DESCRIPTOR_TYPE_STORAGE_IMAGE, or VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT is accessed as a result of this command, the image subresource identified by that descriptor **must** be in the image layout identified when the descriptor was written
- VUID-vkCmdDrawMeshTasksNV-renderPass-02684
  The current render pass **must** be compatible with the renderPass member of the VkGraphicsPipelineCreateInfo structure specified when creating the VkPipeline bound to VK_PIPELINE_BIND_POINT_GRAPHICS
- VUID-vkCmdDrawMeshTasksNV-subpass-02685
  The subpass index of the current render pass **must** be equal to the subpass member of the VkGraphicsPipelineCreateInfo structure specified when creating the VkPipeline bound to VK_PIPELINE_BIND_POINT_GRAPHICS
- VUID-vkCmdDrawMeshTasksNV-None-07748
  If any shader statically accesses an input attachment, a valid descriptor **must** be bound to the pipeline via a descriptor set
- VUID-vkCmdDrawMeshTasksNV-OpTypeImage-07468
  If any shader executed by this pipeline accesses an OpTypeImage variable with a Dim operand of SubpassData, it **must** be decorated with an InputAttachmentIndex that corresponds to a valid input attachment in the current subpass
- VUID-vkCmdDrawMeshTasksNV-None-07469
  Input attachment views accessed in a subpass **must** be created with the same VkFormat

as the corresponding subpass definition, and be created with a VkImageView that is compatible with the attachment referenced by the subpass' pInputAttachments [InputAttachmentIndex] in the currently bound VkFramebuffer as specified by Fragment Input Attachment Compatibility

- VUID-vkCmdDrawMeshTasksNV-pDepthInputAttachmentIndex-09595
  Input attachment views accessed in a dynamic render pass with a InputAttachmentIndex referenced by VkRenderingInputAttachmentIndexInfoKHR, or no InputAttachmentIndex if VkRenderingInputAttachmentIndexInfoKHR:pDepthInputAttachmentIndex or VkRenderingInputAttachmentIndexInfoKHR:pStencilInputAttachmentIndex are NULL, **must** be created with a VkImageView that is compatible with the corresponding color, depth, or stencil attachment in VkRenderingInfo
- VUID-vkCmdDrawMeshTasksNV-pDepthInputAttachmentIndex-09596
  Input attachment views accessed in a dynamic render pass via a shader object **must** have an InputAttachmentIndex if both VkRenderingInputAttachmentIndexInfoKHR:pDepthInputAttachmentIndex and VkRenderingInputAttachmentIndexInfoKHR:pStencilInputAttachmentIndex are non-NULL
- VUID-vkCmdDrawMeshTasksNV-InputAttachmentIndex-09597
  If an input attachment view accessed in a dynamic render pass via a shader object has an InputAttachmentIndex, the InputAttachmentIndex **must** match an index in VkRenderingInputAttachmentIndexInfoKHR
- VUID-vkCmdDrawMeshTasksNV-None-06537
  Memory backing image subresources used as attachments in the current render pass **must** not be written in any way other than as an attachment by this command
- VUID-vkCmdDrawMeshTasksNV-None-09000
  If a color attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass, it is not in the VK_IMAGE_LAYOUT_ATTACHMENT_FEEDBACK_LOOP_OPTIMAL_EXT image layout, and either:
  - the VK_PIPELINE_CREATE_COLOR_ATTACHMENT_FEEDBACK_LOOP_BIT_EXT is set on the currently bound pipeline or
  - the last call to vkCmdSetAttachmentFeedbackLoopEnableEXT included VK_IMAGE_ASPECT_COLOR_BIT and
    - there is no currently bound graphics pipeline or
    - the currently bound graphics pipeline was created with VK_DYNAMIC_STATE_ATTACHMENT_FEEDBACK_LOOP_ENABLE_EXT

  it **must** not be accessed in any way other than as an attachment by this command
- VUID-vkCmdDrawMeshTasksNV-None-09001
  If a depth attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass, it is not in the VK_IMAGE_LAYOUT_ATTACHMENT_FEEDBACK_LOOP_OPTIMAL_EXT image layout, and either:
  - the VK_PIPELINE_CREATE_DEPTH_STENCIL_ATTACHMENT_FEEDBACK_LOOP_BIT_EXT is set on the currently bound pipeline or

- the last call to vkCmdSetAttachmentFeedbackLoopEnableEXT included VK_IMAGE_ASPECT_DEPTH_BIT and
  - there is no currently bound graphics pipeline or
  - the currently bound graphics pipeline was created with VK_DYNAMIC_STATE_ATTACHMENT_FEEDBACK_LOOP_ENABLE_EXT

it **must** not be accessed in any way other than as an attachment by this command

- VUID-vkCmdDrawMeshTasksNV-None-09002
  If a stencil attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass, it is not in the VK_IMAGE_LAYOUT_ATTACHMENT_FEEDBACK_LOOP_OPTIMAL_EXT image layout, and either:
  - the VK_PIPELINE_CREATE_DEPTH_STENCIL_ATTACHMENT_FEEDBACK_LOOP_BIT_EXT is set on the currently bound pipeline or
  - the last call to vkCmdSetAttachmentFeedbackLoopEnableEXT included VK_IMAGE_ASPECT_STENCIL_BIT and
    - there is no currently bound graphics pipeline or
    - the currently bound graphics pipeline was created with VK_DYNAMIC_STATE_ATTACHMENT_FEEDBACK_LOOP_ENABLE_EXT

  it **must** not be accessed in any way other than as an attachment by this command

- VUID-vkCmdDrawMeshTasksNV-None-09003
  If an attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass, it **must** not be accessed in any way other than as an attachment, storage image, or sampled image by this command
- VUID-vkCmdDrawMeshTasksNV-None-06539
  If any previously recorded command in the current subpass accessed an image subresource used as an attachment in this subpass in any way other than as an attachment, this command **must** not write to that image subresource as an attachment
- VUID-vkCmdDrawMeshTasksNV-None-06886
  If the current render pass instance uses a depth/stencil attachment with a read-only layout for the depth aspect, depth writes **must** be disabled
- VUID-vkCmdDrawMeshTasksNV-None-06887
  If the current render pass instance uses a depth/stencil attachment with a read-only layout for the stencil aspect, both front and back writeMask are not zero, and stencil test is enabled, all stencil ops **must** be VK_STENCIL_OP_KEEP
- VUID-vkCmdDrawMeshTasksNV-None-07831
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT dynamic state enabled then vkCmdSetViewport **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksNV-None-07832
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SCISSOR dynamic state enabled then vkCmdSetScissor **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksNV-None-07833
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_WIDTH dynamic state enabled then vkCmdSetLineWidth **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksNV-None-08617
  If a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetPolygonModeEXT in the current command buffer set polygonMode to VK_POLYGON_MODE_LINE, vkCmdSetLineWidth **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksNV-None-08618
  If a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetPrimitiveTopologyEXT in the current command buffer set primitiveTopology to any line topology, vkCmdSetLineWidth **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksNV-None-08619
  If a shader object that outputs line primitives is bound to the VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT or VK_SHADER_STAGE_GEOMETRY_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, vkCmdSetLineWidth **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksNV-None-07834
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_BIAS dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of depthBiasEnable is VK_TRUE, then vkCmdSetDepthBounds or vkCmdSetDepthBias2EXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksNV-None-07835
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_BLEND_CONSTANTS dynamic state enabled then vkCmdSetBlendConstants **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksNV-None-08621
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetColorBlendEnableEXT in the current command buffer set any element of pColorBlendEnables to VK_TRUE, and the most recent call to vkCmdSetColorBlendEquationEXT in the current command buffer set the same element of pColorBlendEquations to a VkColorBlendEquationEXT structure with any VkBlendFactor

member with a value of VK_BLEND_FACTOR_CONSTANT_COLOR, VK_BLEND_FACTOR_ONE_MINUS_CONSTANT_COLOR, VK_BLEND_FACTOR_CONSTANT_ALPHA, or VK_BLEND_FACTOR_ONE_MINUS_CONSTANT_ALPHA, vkCmdSetBlendConstants **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksNV-None-07836
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_BOUNDS dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of depthBoundsTestEnable is VK_TRUE, then vkCmdSetDepthBounds **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksNV-None-07837
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_STENCIL_COMPARE_MASK dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of stencilTestEnable is VK_TRUE, then vkCmdSetStencilCompareMask **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksNV-None-07838
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_STENCIL_WRITE_MASK dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of stencilTestEnable is VK_TRUE, then vkCmdSetStencilWriteMask **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksNV-None-07839
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_STENCIL_REFERENCE dynamic state enabled, the current value of and rasterizerDiscardEnable is VK_FALSE, the current value of stencilTestEnable is VK_TRUE, then vkCmdSetStencilReference **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksNV-maxMultiviewInstanceIndex-02688
  If the draw is recorded in a render pass instance with multiview enabled, the maximum instance index **must** be less than or equal to VkPhysicalDeviceMultiviewProperties::maxMultiviewInstanceIndex
- VUID-vkCmdDrawMeshTasksNV-sampleLocationsEnable-02689
  If the bound graphics pipeline was created with VkPipelineSampleLocationsStateCreateInfoEXT::sampleLocationsEnable set to VK_TRUE and the current subpass has a depth/stencil attachment, then that attachment **must** have been created with the VK_IMAGE_CREATE_SAMPLE_LOCATIONS_COMPATIBLE_DEPTH_BIT_EXT bit set
- VUID-vkCmdDrawMeshTasksNV-None-06666
  If the VK_EXT_sample_locations extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the

VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of sampleLocationsEnable is VK_TRUE, then vkCmdSetSampleLocationsEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksNV-None-07840
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_CULL_MODE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetCullModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksNV-None-07841
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_FRONT_FACE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetFrontFaceEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksNV-None-07843
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_TEST_ENABLE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, vkCmdSetDepthTestEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksNV-None-07844
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_WRITE_ENABLE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetDepthWriteEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksNV-None-07845
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_COMPARE_OP dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of depthTestEnable is VK_TRUE, then vkCmdSetDepthCompareOpEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksNV-None-07846
  If the depthBounds feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_BOUNDS_TEST_ENABLE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetDepthBoundsTestEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksNV-None-07847
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_STENCIL_TEST_ENABLE dynamic state enabled, and

the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetStencilTestEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksNV-None-07848
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_STENCIL_OP dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, the current value of stencilTestEnable is VK_TRUE, then vkCmdSetStencilOpEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksNV-viewportCount-03417
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, but not the VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT dynamic state enabled, then vkCmdSetViewportWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the viewportCount parameter of vkCmdSetViewportWithCountEXT **must** match the VkPipelineViewportStateCreateInfo::scissorCount of the pipeline

- VUID-vkCmdDrawMeshTasksNV-scissorCount-03418
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT dynamic state enabled, but not the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, then vkCmdSetScissorWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the scissorCount parameter of vkCmdSetScissorWithCountEXT **must** match the VkPipelineViewportStateCreateInfo::viewportCount of the pipeline

- VUID-vkCmdDrawMeshTasksNV-viewportCount-03419
  If the bound graphics pipeline state was created with both the VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT and VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic states enabled then both vkCmdSetViewportWithCountEXT and vkCmdSetScissorWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the viewportCount parameter of vkCmdSetViewportWithCountEXT **must** match the scissorCount parameter of vkCmdSetScissorWithCountEXT

- VUID-vkCmdDrawMeshTasksNV-None-08635
  If a shader object is bound to any graphics stage, then both vkCmdSetViewportWithCountEXT and vkCmdSetScissorWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the viewportCount parameter of vkCmdSetViewportWithCountEXT **must** match the scissorCount parameter of vkCmdSetScissorWithCountEXT

- VUID-vkCmdDrawMeshTasksNV-viewportCount-04137
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, but not the VK_DYNAMIC_STATE_VIEWPORT_W_SCALING_NV dynamic state enabled, then the bound graphics pipeline **must** have been created with VkPipelineViewportWScalingStateCreateInfoNV::viewportCount greater or equal to the viewportCount parameter in the last call to

vkCmdSetViewportWithCountEXT

- VUID-vkCmdDrawMeshTasksNV-viewportCount-04138
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT and VK_DYNAMIC_STATE_VIEWPORT_W_SCALING_NV dynamic states enabled then the viewportCount parameter in the last call to vkCmdSetViewportWScalingNV **must** be greater than or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT

- VUID-vkCmdDrawMeshTasksNV-None-09232
  If the VK_NV_clip_space_w_scaling extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetViewportWScalingEnableNV in the current command buffer set viewportWScalingEnable to VK_TRUE, then vkCmdSetViewportWScalingNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksNV-None-08636
  If the VK_NV_clip_space_w_scaling extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetViewportWScalingEnableNV in the current command buffer set viewportWScalingEnable to VK_TRUE, then the viewportCount parameter in the last call to vkCmdSetViewportWScalingNV **must** be greater than or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT

- VUID-vkCmdDrawMeshTasksNV-viewportCount-04139
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, but not the VK_DYNAMIC_STATE_VIEWPORT_SHADING_RATE_PALETTE_NV dynamic state enabled, then the bound graphics pipeline **must** have been created with VkPipelineViewportShadingRateImageStateCreateInfoNV::viewportCount greater or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT

- VUID-vkCmdDrawMeshTasksNV-viewportCount-04140
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT and VK_DYNAMIC_STATE_VIEWPORT_SHADING_RATE_PALETTE_NV dynamic states enabled then the viewportCount parameter in the last call to vkCmdSetViewportShadingRatePaletteNV **must** be greater than or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT

- VUID-vkCmdDrawMeshTasksNV-shadingRateImage-09233
  If the shadingRateImage feature is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetCoarseSampleOrderNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksNV-shadingRateImage-09234
  If the shadingRateImage feature is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetShadingRateImageEnableNV in the current command buffer set shadingRateImageEnable to VK_TRUE, then vkCmdSetViewportShadingRatePaletteNV **must**

have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksNV-None-08637
  If the `shadingRateImage` feature is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE`, and the most recent call to vkCmdSetShadingRateImageEnableNV in the current command buffer set `shadingRateImageEnable` to `VK_TRUE`, then the `viewportCount` parameter in the last call to vkCmdSetViewportShadingRatePaletteNV **must** be greater than or equal to the `viewportCount` parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawMeshTasksNV-VkPipelineVieportCreateInfo-04141
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT` dynamic state enabled and a VkPipelineViewportSwizzleStateCreateInfoNV structure chained from VkPipelineViewportStateCreateInfo, then the bound graphics pipeline **must** have been created with VkPipelineViewportSwizzleStateCreateInfoNV::`viewportCount` greater or equal to the `viewportCount` parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawMeshTasksNV-VkPipelineVieportCreateInfo-04142
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT` dynamic state enabled and a VkPipelineViewportExclusiveScissorStateCreateInfoNV structure chained from VkPipelineViewportStateCreateInfo, then the bound graphics pipeline **must** have been created with VkPipelineViewportExclusiveScissorStateCreateInfoNV::`exclusiveScissorCount` greater or equal to the `viewportCount` parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawMeshTasksNV-None-07878
  If the `exclusiveScissor` feature is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_EXCLUSIVE_SCISSOR_ENABLE_NV` dynamic state enabled, then vkCmdSetExclusiveScissorEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksNV-None-07879
  If the `exclusiveScissor` feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_EXCLUSIVE_SCISSOR_NV` dynamic state enabled, and the most recent call to vkCmdSetExclusiveScissorEnableNV in the current command buffer set any element of `pExclusiveScissorEnables` to `VK_TRUE`, then vkCmdSetExclusiveScissorNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksNV-None-04876
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_RASTERIZER_DISCARD_ENABLE` dynamic state enabled, then vkCmdSetRasterizerDiscardEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksNV-None-04877

If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_BIAS_ENABLE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetDepthBiasEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksNV-logicOp-04878
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_LOGIC_OP_EXT dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of logicOpEnable is VK_TRUE, then vkCmdSetLogicOpEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksNV-primitiveFragmentShadingRateWithMultipleViewports-04552
  If the primitiveFragmentShadingRateWithMultipleViewports limit is not supported, the bound graphics pipeline was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, and any of the shader stages of the bound graphics pipeline write to the PrimitiveShadingRateKHR built-in, then vkCmdSetViewportWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the viewportCount parameter of vkCmdSetViewportWithCountEXT **must** be 1
- VUID-vkCmdDrawMeshTasksNV-primitiveFragmentShadingRateWithMultipleViewports-08642
  If the primitiveFragmentShadingRateWithMultipleViewports limit is not supported, and any shader object bound to a graphics stage writes to the PrimitiveShadingRateKHR built-in, then vkCmdSetViewportWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the viewportCount parameter of vkCmdSetViewportWithCountEXT **must** be 1
- VUID-vkCmdDrawMeshTasksNV-blendEnable-04727
  If rasterization is not disabled in the bound graphics pipeline, then for each color attachment in the subpass, if the corresponding image view's format features do not contain VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT, then the blendEnable member of the corresponding element of the pAttachments member of pColorBlendState **must** be VK_FALSE
- VUID-vkCmdDrawMeshTasksNV-None-08643
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then for each color attachment in the render pass, if the corresponding image view's format features do not contain VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT, then the corresponding member of pColorBlendEnables in the most recent call to vkCmdSetColorBlendEnableEXT in the current command buffer that affected that attachment index **must** have been VK_FALSE
- VUID-vkCmdDrawMeshTasksNV-multisampledRenderToSingleSampled-07284
  If rasterization is not disabled in the bound graphics pipeline, and none of the following is enabled:
  - the VK_AMD_mixed_attachment_samples extension

- the VK_NV_framebuffer_mixed_samples extension
- the multisampledRenderToSingleSampled feature

then rasterizationSamples for the currently bound graphics pipeline **must** be the same as the current subpass color and/or depth/stencil attachments

- VUID-vkCmdDrawMeshTasksNV-None-08644
  If a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and none of the following is enabled:

  - the VK_AMD_mixed_attachment_samples extension
  - the VK_NV_framebuffer_mixed_samples extension
  - the multisampledRenderToSingleSampled feature

  then the most recent call to vkCmdSetRasterizationSamplesEXT in the current command buffer **must** have set rasterizationSamples to be the same as the number of samples for the current render pass color and/or depth/stencil attachments

- VUID-vkCmdDrawMeshTasksNV-None-08876
  If a shader object is bound to any graphics stage, the current render pass instance **must** have been begun with vkCmdBeginRenderingKHR
- VUID-vkCmdDrawMeshTasksNV-imageView-06172
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the imageView member of pDepthAttachment is not VK_NULL_HANDLE, and the layout member of pDepthAttachment is VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL, this command **must** not write any values to the depth attachment
- VUID-vkCmdDrawMeshTasksNV-imageView-06173
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the imageView member of pStencilAttachment is not VK_NULL_HANDLE, and the layout member of pStencilAttachment is VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL, this command **must** not write any values to the stencil attachment
- VUID-vkCmdDrawMeshTasksNV-imageView-06174
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the imageView member of pDepthAttachment is not VK_NULL_HANDLE, and the layout member of pDepthAttachment is VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_STENCIL_ATTACHMENT_OPTIMAL, this command **must** not write any values to the depth attachment
- VUID-vkCmdDrawMeshTasksNV-imageView-06175
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the imageView member of pStencilAttachment is not VK_NULL_HANDLE, and the layout member of pStencilAttachment is VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_STENCIL_READ_ONLY_OPTIMAL, this command **must** not write any values to the stencil attachment
- VUID-vkCmdDrawMeshTasksNV-imageView-06176
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the imageView member of pDepthAttachment is not VK_NULL_HANDLE, and the layout member

of pDepthAttachment is VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL, this command **must** not write any values to the depth attachment

- VUID-vkCmdDrawMeshTasksNV-imageView-06177
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the imageView member of pStencilAttachment is not VK_NULL_HANDLE, and the layout member of pStencilAttachment is VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL, this command **must** not write any values to the stencil attachment

- VUID-vkCmdDrawMeshTasksNV-viewMask-06178
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the currently bound graphics pipeline **must** have been created with a VkPipelineRenderingCreateInfo::viewMask equal to VkRenderingInfo::viewMask

- VUID-vkCmdDrawMeshTasksNV-colorAttachmentCount-06179
  If the dynamicRenderingUnusedAttachments feature is not enabled and the current render pass instance was begun with vkCmdBeginRenderingKHR, the currently bound graphics pipeline **must** have been created with a VkPipelineRenderingCreateInfo::colorAttachmentCount equal to VkRenderingInfo::colorAttachmentCount

- VUID-vkCmdDrawMeshTasksNV-dynamicRenderingUnusedAttachments-08910
  If the dynamicRenderingUnusedAttachments feature is not enabled, and the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingInfo::colorAttachmentCount greater than 0, then each element of the VkRenderingInfo::pColorAttachments array with an imageView not equal to VK_NULL_HANDLE **must** have been created with a VkFormat equal to the corresponding element of VkPipelineRenderingCreateInfo::pColorAttachmentFormats used to create the currently bound graphics pipeline

- VUID-vkCmdDrawMeshTasksNV-dynamicRenderingUnusedAttachments-08912
  If the dynamicRenderingUnusedAttachments feature is not enabled, and the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingInfo::colorAttachmentCount greater than 0, then each element of the VkRenderingInfo::pColorAttachments array with an imageView equal to VK_NULL_HANDLE **must** have the corresponding element of VkPipelineRenderingCreateInfo::pColorAttachmentFormats used to create the currently bound pipeline equal to VK_FORMAT_UNDEFINED

- VUID-vkCmdDrawMeshTasksNV-dynamicRenderingUnusedAttachments-08911
  If the dynamicRenderingUnusedAttachments feature is enabled, and the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingInfo::colorAttachmentCount greater than 0, then each element of the VkRenderingInfo::pColorAttachments array with an imageView not equal to VK_NULL_HANDLE **must** have been created with a VkFormat equal to the corresponding element of VkPipelineRenderingCreateInfo::pColorAttachmentFormats used to create the currently bound graphics pipeline, or the corresponding element of VkPipelineRenderingCreateInfo::pColorAttachmentFormats, if it exists, **must** be VK_FORMAT_UNDEFINED

- VUID-vkCmdDrawMeshTasksNV-colorAttachmentCount-09362
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, with a VkRenderingInfo::colorAttachmentCount equal to 1, there is no shader object bound to any graphics stage, and a color attachment with a resolve mode of

VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID, each element of the VkRenderingInfo::pColorAttachments array with a resolveImageView not equal to VK_NULL_HANDLE **must** have been created with an image created with a VkExternalFormatANDROID::externalFormat value equal to the VkExternalFormatANDROID::externalFormat value used to create the currently bound graphics pipeline

- VUID-vkCmdDrawMeshTasksNV-None-09363
  If there is no shader object bound to any graphics stage, the current render pass instance was begun with vkCmdBeginRenderingKHR and a VkRenderingInfo::colorAttachmentCount equal to 1, and a color attachment with a resolve mode of VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID, each element of the VkRenderingInfo::pColorAttachments array with a imageView not equal to VK_NULL_HANDLE **must** have been created with an image created with a VkExternalFormatANDROID::externalFormat value equal to the VkExternalFormatANDROID::externalFormat value used to create the currently bound graphics pipeline

- VUID-vkCmdDrawMeshTasksNV-None-09364
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is no shader object bound to any graphics stage, and the currently bound graphics pipeline was created with a non-zero VkExternalFormatANDROID::externalFormat value and with the VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT dynamic state enabled, then vkCmdSetColorBlendEnableEXT **must** have set the blend enable to VK_FALSE prior to this drawing command

- VUID-vkCmdDrawMeshTasksNV-None-09365
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is no shader object bound to any graphics stage, and the currently bound graphics pipeline was created with a non-zero VkExternalFormatANDROID::externalFormat value and with the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT dynamic state enabled, then vkCmdSetRasterizationSamplesEXT **must** have set rasterizationSamples to VK_SAMPLE_COUNT_1_BIT prior to this drawing command

- VUID-vkCmdDrawMeshTasksNV-None-09366
  If there is a shader object bound to any graphics stage, and the current render pass includes a color attachment that uses the VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID resolve mode, then vkCmdSetColorBlendEnableEXT **must** have set blend enable to VK_FALSE prior to this drawing command

- VUID-vkCmdDrawMeshTasksNV-rasterizationSamples-09367
  If there is a shader object bound to any graphics stage, and the current render pass includes a color attachment that uses the VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID resolve mode, then vkCmdSetRasterizationSamplesEXT **must** have set rasterizationSamples to VK_SAMPLE_COUNT_1_BIT prior to this drawing command

- VUID-vkCmdDrawMeshTasksNV-None-09368
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is no shader object bound to any graphics stage, and the currently bound graphics pipeline

was created with a non-zero VkExternalFormatANDROID::externalFormat value and with the VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR dynamic state enabled, then vkCmdSetFragmentShadingRateKHR **must** have set pFragmentSize->width to 1 prior to this drawing command

- VUID-vkCmdDrawMeshTasksNV-None-09369
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is no shader object bound to any graphics stage, and the currently bound graphics pipeline was created with a non-zero VkExternalFormatANDROID::externalFormat value and with the VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR dynamic state enabled, then vkCmdSetFragmentShadingRateKHR **must** have set pFragmentSize->height to 1 prior to this drawing command

- VUID-vkCmdDrawMeshTasksNV-pFragmentSize-09370
  If there is a shader object bound to any graphics stage, and the current render pass includes a color attachment that uses the VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID resolve mode, then vkCmdSetFragmentShadingRateKHR **must** have set pFragmentSize->width to 1 prior to this drawing command

- VUID-vkCmdDrawMeshTasksNV-pFragmentSize-09371
  If there is a shader object bound to any graphics stage, and the current render pass includes a color attachment that uses the VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID resolve mode, then vkCmdSetFragmentShadingRateKHR **must** have set pFragmentSize->height to 1 prior to this drawing command

- VUID-vkCmdDrawMeshTasksNV-None-07749
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_WRITE_ENABLE_EXT dynamic state enabled then vkCmdSetColorWriteEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksNV-None-08646
  If the colorWriteEnable feature is enabled on the device, and a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetColorWriteEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksNV-attachmentCount-07750
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_WRITE_ENABLE_EXT dynamic state enabled then the attachmentCount parameter of vkCmdSetColorWriteEnableEXT **must** be greater than or equal to the VkPipelineColorBlendStateCreateInfo::attachmentCount of the currently bound graphics pipeline

- VUID-vkCmdDrawMeshTasksNV-None-08647
  If the colorWriteEnable feature is enabled on the device, and a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set

rasterizerDiscardEnable to VK_FALSE, then the attachmentCount parameter of most recent call to vkCmdSetColorWriteEnableEXT in the current command buffer **must** be greater than or equal to the number of color attachments in the current render pass instance

- VUID-vkCmdDrawMeshTasksNV-None-07751
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_DISCARD_RECTANGLE_EXT dynamic state enabled then vkCmdSetDiscardRectangleEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command for each discard rectangle in VkPipelineDiscardRectangleStateCreateInfoEXT::discardRectangleCount

- VUID-vkCmdDrawMeshTasksNV-None-07880
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_DISCARD_RECTANGLE_ENABLE_EXT dynamic state enabled then vkCmdSetDiscardRectangleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksNV-rasterizerDiscardEnable-09236
  If the VK_EXT_discard_rectangles extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetDiscardRectangleEnableEXT in the current command buffer set discardRectangleEnable to VK_TRUE, then vkCmdSetDiscardRectangleEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksNV-None-08648
  If the VK_EXT_discard_rectangles extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetDiscardRectangleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksNV-None-07881
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_DISCARD_RECTANGLE_MODE_EXT dynamic state enabled then vkCmdSetDiscardRectangleModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksNV-None-08649
  If the VK_EXT_discard_rectangles extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetDiscardRectangleEnableEXT in the current command buffer set discardRectangleEnable to VK_TRUE, then vkCmdSetDiscardRectangleModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksNV-dynamicRenderingUnusedAttachments-08913
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is not enabled, and VkRenderingInfo::pDepthAttachment->imageView was VK_NULL_HANDLE, the value of

VkPipelineRenderingCreateInfo::depthAttachmentFormat used to create the currently bound graphics pipeline **must** be equal to VK_FORMAT_UNDEFINED

- VUID-vkCmdDrawMeshTasksNV-dynamicRenderingUnusedAttachments-08914
  If current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is not enabled, and VkRenderingInfo::pDepthAttachment->imageView was not VK_NULL_HANDLE, the value of VkPipelineRenderingCreateInfo::depthAttachmentFormat used to create the currently bound graphics pipeline **must** be equal to the VkFormat used to create VkRenderingInfo::pDepthAttachment->imageView
- VUID-vkCmdDrawMeshTasksNV-dynamicRenderingUnusedAttachments-08915
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is enabled, VkRenderingInfo::pDepthAttachment->imageView was not VK_NULL_HANDLE, and the value of VkPipelineRenderingCreateInfo::depthAttachmentFormat used to create the currently bound graphics pipeline was not equal to the VkFormat used to create VkRenderingInfo::pDepthAttachment->imageView, the value of the format **must** be VK_FORMAT_UNDEFINED
- VUID-vkCmdDrawMeshTasksNV-dynamicRenderingUnusedAttachments-08916
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is not enabled, and VkRenderingInfo::pStencilAttachment->imageView was VK_NULL_HANDLE, the value of VkPipelineRenderingCreateInfo::stencilAttachmentFormat used to create the currently bound graphics pipeline **must** be equal to VK_FORMAT_UNDEFINED
- VUID-vkCmdDrawMeshTasksNV-dynamicRenderingUnusedAttachments-08917
  If current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is not enabled, and VkRenderingInfo::pStencilAttachment->imageView was not VK_NULL_HANDLE, the value of VkPipelineRenderingCreateInfo::stencilAttachmentFormat used to create the currently bound graphics pipeline **must** be equal to the VkFormat used to create VkRenderingInfo::pStencilAttachment->imageView
- VUID-vkCmdDrawMeshTasksNV-dynamicRenderingUnusedAttachments-08918
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is enabled, VkRenderingInfo::pStencilAttachment->imageView was not VK_NULL_HANDLE, and the value of VkPipelineRenderingCreateInfo::stencilAttachmentFormat used to create the currently bound graphics pipeline was not equal to the VkFormat used to create VkRenderingInfo::pStencilAttachment->imageView, the value of the format **must** be VK_FORMAT_UNDEFINED
- VUID-vkCmdDrawMeshTasksNV-imageView-06183
  If the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingFragmentShadingRateAttachmentInfoKHR::imageView was not VK_NULL_HANDLE, the currently bound graphics pipeline **must** have been created with VK_PIPELINE_CREATE_RENDERING_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR
- VUID-vkCmdDrawMeshTasksNV-imageView-06184
  If the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingFragmentDensityMapAttachmentInfoEXT::imageView was not VK_NULL_HANDLE, the currently bound graphics pipeline **must** have been created with

VK_PIPELINE_CREATE_RENDERING_FRAGMENT_DENSITY_MAP_ATTACHMENT_BIT_EXT

- VUID-vkCmdDrawMeshTasksNV-colorAttachmentCount-06185
  If the currently bound pipeline was created with a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and the current render pass instance was begun with vkCmdBeginRenderingKHR with a VkRenderingInfo::colorAttachmentCount parameter greater than 0, then each element of the VkRenderingInfo::pColorAttachments array with a imageView not equal to VK_NULL_HANDLE **must** have been created with a sample count equal to the corresponding element of the pColorAttachmentSamples member of VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV used to create the currently bound graphics pipeline
- VUID-vkCmdDrawMeshTasksNV-pDepthAttachment-06186
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the currently bound pipeline was created with a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and VkRenderingInfo::pDepthAttachment->imageView was not VK_NULL_HANDLE, the value of the depthStencilAttachmentSamples member of VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV used to create the currently bound graphics pipeline **must** be equal to the sample count used to create VkRenderingInfo::pDepthAttachment->imageView
- VUID-vkCmdDrawMeshTasksNV-pStencilAttachment-06187
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the currently bound pipeline was created with a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and VkRenderingInfo::pStencilAttachment->imageView was not VK_NULL_HANDLE, the value of the depthStencilAttachmentSamples member of VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV used to create the currently bound graphics pipeline **must** be equal to the sample count used to create VkRenderingInfo::pStencilAttachment->imageView
- VUID-vkCmdDrawMeshTasksNV-multisampledRenderToSingleSampled-07285
  If the currently bound pipeline was created without a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and the multisampledRenderToSingleSampled feature is not enabled, and the current render pass instance was begun with vkCmdBeginRenderingKHR with a VkRenderingInfo::colorAttachmentCount parameter greater than 0, then each element of the VkRenderingInfo::pColorAttachments array with a imageView not equal to VK_NULL_HANDLE **must** have been created with a sample count equal to the value of rasterizationSamples for the currently bound graphics pipeline
- VUID-vkCmdDrawMeshTasksNV-multisampledRenderToSingleSampled-07286
  If the currently bound pipeline was created without a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and the multisampledRenderToSingleSampled feature is not enabled, and VkRenderingInfo::pDepthAttachment->imageView was not VK_NULL_HANDLE, the value of rasterizationSamples for the currently bound graphics pipeline **must** be equal to the sample count used to create VkRenderingInfo::pDepthAttachment->imageView
- VUID-vkCmdDrawMeshTasksNV-multisampledRenderToSingleSampled-07287
  If the currently bound pipeline was created without a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and

the multisampledRenderToSingleSampled feature is not enabled, and VkRenderingInfo::pStencilAttachment->imageView was not VK_NULL_HANDLE, the value of rasterizationSamples for the currently bound graphics pipeline **must** be equal to the sample count used to create VkRenderingInfo::pStencilAttachment->imageView

- VUID-vkCmdDrawMeshTasksNV-pNext-07935
  If this command has been called inside a render pass instance started with vkCmdBeginRenderingKHR, and the pNext chain of VkRenderingInfo includes a VkMultisampledRenderToSingleSampledInfoEXT structure with multisampledRenderToSingleSampledEnable equal to VK_TRUE, then the value of rasterizationSamples for the currently bound graphics pipeline **must** be equal to VkMultisampledRenderToSingleSampledInfoEXT::rasterizationSamples
- VUID-vkCmdDrawMeshTasksNV-renderPass-06198
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the currently bound pipeline **must** have been created with a VkGraphicsPipelineCreateInfo::renderPass equal to VK_NULL_HANDLE
- VUID-vkCmdDrawMeshTasksNV-pColorAttachments-08963
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is a graphics pipeline bound with a fragment shader that statically writes to a color attachment, the color write mask is not zero, color writes are enabled, and the corresponding element of the VkRenderingInfo::pColorAttachments->imageView was not VK_NULL_HANDLE, then the corresponding element of VkPipelineRenderingCreateInfo::pColorAttachmentFormats used to create the pipeline **must** not be VK_FORMAT_UNDEFINED
- VUID-vkCmdDrawMeshTasksNV-pDepthAttachment-08964
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is a graphics pipeline bound, depth test is enabled, depth write is enabled, and the VkRenderingInfo::pDepthAttachment->imageView was not VK_NULL_HANDLE, then the VkPipelineRenderingCreateInfo::depthAttachmentFormat used to create the pipeline **must** not be VK_FORMAT_UNDEFINED
- VUID-vkCmdDrawMeshTasksNV-pStencilAttachment-08965
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is a graphics pipeline bound, stencil test is enabled and the VkRenderingInfo::pStencilAttachment->imageView was not VK_NULL_HANDLE, then the VkPipelineRenderingCreateInfo::stencilAttachmentFormat used to create the pipeline **must** not be VK_FORMAT_UNDEFINED
- VUID-vkCmdDrawMeshTasksNV-primitivesGeneratedQueryWithRasterizerDiscard-06708
  If the primitivesGeneratedQueryWithRasterizerDiscard feature is not enabled and the VK_QUERY_TYPE_PRIMITIVES_GENERATED_EXT query is active, rasterization discard **must** not be enabled
- VUID-vkCmdDrawMeshTasksNV-primitivesGeneratedQueryWithNonZeroStreams-06709
  If the primitivesGeneratedQueryWithNonZeroStreams feature is not enabled and the VK_QUERY_TYPE_PRIMITIVES_GENERATED_EXT query is active, the bound graphics pipeline **must** not have been created with a non-zero value in VkPipelineRasterizationStateStreamCreateInfoEXT::rasterizationStream
- VUID-vkCmdDrawMeshTasksNV-None-07619
  If a shader object is bound to the VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT stage or a

graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_TESSELLATION_DOMAIN_ORIGIN_EXT dynamic state enabled, then vkCmdSetTessellationDomainOriginEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksNV-None-07620
  If the depthClamp feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_CLAMP_ENABLE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetDepthClampEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksNV-None-07621
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_POLYGON_MODE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetPolygonModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksNV-None-07622
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetRasterizationSamplesEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksNV-None-07623
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_SAMPLE_MASK_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetSampleMaskEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksNV-alphaToCoverageEnable-08919
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_ALPHA_TO_COVERAGE_ENABLE_EXT dynamic state enabled, and alphaToCoverageEnable was VK_TRUE in the last call to vkCmdSetAlphaToCoverageEnableEXT, then the Fragment Output Interface **must** contain a variable for the alpha Component word in Location 0 at Index 0
- VUID-vkCmdDrawMeshTasksNV-alphaToCoverageEnable-08920
  If a shader object is bound to any graphics stage, and the most recent call to vkCmdSetAlphaToCoverageEnableEXT in the current command buffer set alphaToCoverageEnable to VK_TRUE, then the Fragment Output Interface **must** contain a variable for the alpha Component word in Location 0 at Index 0
- VUID-vkCmdDrawMeshTasksNV-None-07624
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_ALPHA_TO_COVERAGE_ENABLE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetAlphaToCoverageEnableEXT **must** have been called and not subsequently

invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksNV-None-07625
  If the alphaToOne feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_ALPHA_TO_ONE_ENABLE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetAlphaToOneEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksNV-None-07626
  If the logicOp feature is enabled, a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_LOGIC_OP_ENABLE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetLogicOpEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksNV-None-07627
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT dynamic state enabled then vkCmdSetColorBlendEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksNV-None-08657
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and both the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE and there are color attachments bound, then vkCmdSetColorBlendEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksNV-None-07628
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_EQUATION_EXT dynamic state enabled then vkCmdSetColorBlendEquationEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksNV-None-08658
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetColorBlendEnableEXT for any attachment set that attachment's value in pColorBlendEnables to VK_TRUE, then vkCmdSetColorBlendEquationEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksNV-None-07629
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_WRITE_MASK_EXT dynamic state enabled then vkCmdSetColorWriteMaskEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksNV-None-08659

If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and both the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE and there are color attachments bound, then vkCmdSetColorWriteMaskEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksNV-None-07630
  If the geometryStreams feature is enabled, and a shader object is bound to the VK_SHADER_STAGE_GEOMETRY_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_RASTERIZATION_STREAM_EXT dynamic state enabled, then vkCmdSetRasterizationStreamEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksNV-None-07631
  If the VK_EXT_conservative_rasterization extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_CONSERVATIVE_RASTERIZATION_MODE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetConservativeRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksNV-None-07632
  If the VK_EXT_conservative_rasterization extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_EXTRA_PRIMITIVE_OVERESTIMATION_SIZE_EXT dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of conservativeRasterizationMode is VK_CONSERVATIVE_RASTERIZATION_MODE_OVERESTIMATE_EXT, then vkCmdSetExtraPrimitiveOverestimationSizeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksNV-None-07633
  If the depthClipEnable feature is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_CLIP_ENABLE_EXT dynamic state, then vkCmdSetDepthClipEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksNV-None-07634
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT dynamic state enabled then vkCmdSetSampleLocationsEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksNV-None-08664
  If the VK_EXT_sample_locations extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetSampleLocationsEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksNV-None-07635
  If the bound graphics pipeline state was created with the

VK_DYNAMIC_STATE_COLOR_BLEND_ADVANCED_EXT dynamic state enabled then vkCmdSetColorBlendAdvancedEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksNV-rasterizerDiscardEnable-09416
  If the VK_EXT_blend_operation_advanced extension is enabled, and a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then at least one of vkCmdSetColorBlendEquationEXT and vkCmdSetColorBlendAdvancedEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksNV-None-07636
  If the VK_EXT_provoking_vertex extension is enabled, a shader object is bound to the VK_SHADER_STAGE_VERTEX_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_PROVOKING_VERTEX_MODE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetProvokingVertexModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksNV-None-07637
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT dynamic state enabled then vkCmdSetLineRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksNV-None-08666
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetPolygonModeEXT in the current command buffer set polygonMode to VK_POLYGON_MODE_LINE, then vkCmdSetLineRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksNV-None-08667
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object is bound to the VK_SHADER_STAGE_VERTEX_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetPrimitiveTopologyEXT in the current command buffer set primitiveTopology to any line topology, then vkCmdSetLineRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksNV-None-08668
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object that outputs line primitives is bound to the VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT or VK_SHADER_STAGE_GEOMETRY_BIT stage, and

the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetLineRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksNV-None-07638
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT dynamic state enabled then vkCmdSetLineStippleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksNV-None-08669
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetPolygonModeEXT in the current command buffer set polygonMode to VK_POLYGON_MODE_LINE, then vkCmdSetLineStippleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksNV-None-08670
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object is bound to the VK_SHADER_STAGE_VERTEX_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetPrimitiveTopologyEXT in the current command buffer set primitiveTopology to any line topology, then vkCmdSetLineStippleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksNV-None-08671
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object that outputs line primitives is bound to the VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT or VK_SHADER_STAGE_GEOMETRY_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetLineStippleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksNV-None-07849
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_KHR dynamic state enabled then vkCmdSetLineStippleKHR **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksNV-None-08672
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetLineStippleEnableEXT in the current command buffer set stippledLineEnable to

VK_TRUE, then vkCmdSetLineStippleEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksNV-None-07639
  If the depthClipControl feature is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_CLIP_NEGATIVE_ONE_TO_ONE_EXT dynamic state enabled, then vkCmdSetDepthClipNegativeOneToOneEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksNV-None-07640
  If the VK_NV_clip_space_w_scaling extension is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_VIEWPORT_W_SCALING_ENABLE_NV dynamic state enabled, then vkCmdSetViewportWScalingEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksNV-None-07641
  If the VK_NV_viewport_swizzle extension is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_VIEWPORT_SWIZZLE_NV dynamic state enabled, then vkCmdSetViewportSwizzleNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksNV-None-07642
  If the VK_NV_fragment_coverage_to_color extension is enabled, a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_TO_COLOR_ENABLE_NV dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetCoverageToColorEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksNV-None-07643
  If the VK_NV_fragment_coverage_to_color extension is enabled, a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_TO_COLOR_LOCATION_NV dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of coverageToColorEnable is VK_TRUE, then vkCmdSetCoverageToColorLocationNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksNV-None-07644
  If the VK_NV_framebuffer_mixed_samples extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_MODULATION_MODE_NV dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetCoverageModulationModeNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksNV-None-07645
  If the VK_NV_framebuffer_mixed_samples extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the

VK_DYNAMIC_STATE_COVERAGE_MODULATION_TABLE_ENABLE_NV dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of coverageModulationMode is any value other than VK_COVERAGE_MODULATION_MODE_NONE_NV, then vkCmdSetCoverageModulationTableEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksNV-None-07646
  If the VK_NV_framebuffer_mixed_samples extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_MODULATION_TABLE_NV dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of coverageModulationTableEnable is VK_TRUE, then vkCmdSetCoverageModulationTableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksNV-None-07647
  If the shadingRateImage feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_SHADING_RATE_IMAGE_ENABLE_NV dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetShadingRateImageEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksNV-pipelineFragmentShadingRate-09238
  If the pipelineFragmentShadingRate feature is enabled, a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetFragmentShadingRateKHR **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksNV-None-07648
  If the representativeFragmentTest feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_REPRESENTATIVE_FRAGMENT_TEST_ENABLE_NV dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetRepresentativeFragmentTestEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksNV-None-07649
  If the coverageReductionMode feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_REDUCTION_MODE_NV dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetCoverageReductionModeNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksNV-pColorBlendEnables-07470
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT state enabled and the last call to vkCmdSetColorBlendEnableEXT set pColorBlendEnables for any attachment to VK_TRUE,

then for those attachments in the subpass the corresponding image view's format features **must** contain VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT

- VUID-vkCmdDrawMeshTasksNV-rasterizationSamples-07471
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, and the current subpass does not use any color and/or depth/stencil attachments, then the rasterizationSamples in the last call to vkCmdSetRasterizationSamplesEXT **must** follow the rules for a zero-attachment subpass
- VUID-vkCmdDrawMeshTasksNV-samples-07472
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_MASK_EXT state enabled and the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state disabled, then the samples parameter in the last call to vkCmdSetSampleMaskEXT **must** be greater or equal to the VkPipelineMultisampleStateCreateInfo::rasterizationSamples parameter used to create the bound graphics pipeline
- VUID-vkCmdDrawMeshTasksNV-samples-07473
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_MASK_EXT state and VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT states enabled, then the samples parameter in the last call to vkCmdSetSampleMaskEXT **must** be greater or equal to the rasterizationSamples parameter in the last call to vkCmdSetRasterizationSamplesEXT
- VUID-vkCmdDrawMeshTasksNV-rasterizationSamples-07474
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, and neither the VK_AMD_mixed_attachment_samples nor the VK_NV_framebuffer_mixed_samples extensions are enabled, then the rasterizationSamples in the last call to vkCmdSetRasterizationSamplesEXT **must** be the same as the current subpass color and/or depth/stencil attachments
- VUID-vkCmdDrawMeshTasksNV-None-09211
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, or a shader object is bound to any graphics stage, and the current render pass instance includes a VkMultisampledRenderToSingleSampledInfoEXT structure with multisampledRenderToSingleSampledEnable equal to VK_TRUE, then the rasterizationSamples in the last call to vkCmdSetRasterizationSamplesEXT **must** be the same as the rasterizationSamples member of that structure
- VUID-vkCmdDrawMeshTasksNV-firstAttachment-07476
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT dynamic state enabled then vkCmdSetColorBlendEnableEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the firstAttachment and attachmentCount parameters of vkCmdSetColorBlendEnableEXT calls **must** specify an enable for all active color attachments in the current subpass
- VUID-vkCmdDrawMeshTasksNV-rasterizerDiscardEnable-09417
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent

call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetColorBlendEnableEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the firstAttachment and attachmentCount parameters of vkCmdSetColorBlendEnableEXT calls **must** specify an enable for all active color attachments in the current subpass

- VUID-vkCmdDrawMeshTasksNV-firstAttachment-07477
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_EQUATION_EXT dynamic state enabled then vkCmdSetColorBlendEquationEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the firstAttachment and attachmentCount parameters of vkCmdSetColorBlendEquationEXT calls **must** specify the blend equations for all active color attachments in the current subpass where blending is enabled

- VUID-vkCmdDrawMeshTasksNV-rasterizerDiscardEnable-09418
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and both the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE and there are color attachments bound, then vkCmdSetColorBlendEquationEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the firstAttachment and attachmentCount parameters of vkCmdSetColorBlendEquationEXT calls **must** specify the blend equations for all active color attachments in the current subpass where blending is enabled

- VUID-vkCmdDrawMeshTasksNV-firstAttachment-07478
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_WRITE_MASK_EXT dynamic state enabled then vkCmdSetColorWriteMaskEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the firstAttachment and attachmentCount parameters of vkCmdSetColorWriteMaskEXT calls **must** specify the color write mask for all active color attachments in the current subpass

- VUID-vkCmdDrawMeshTasksNV-rasterizerDiscardEnable-09419
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetColorWriteMaskEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the firstAttachment and attachmentCount parameters of vkCmdSetColorWriteMaskEXT calls **must** specify the color write mask for all active color attachments in the current subpass

- VUID-vkCmdDrawMeshTasksNV-firstAttachment-07479
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_ADVANCED_EXT dynamic state enabled then vkCmdSetColorBlendAdvancedEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the firstAttachment and attachmentCount parameters of vkCmdSetColorBlendAdvancedEXT calls **must** specify the advanced blend equations for all active color attachments in the current subpass where blending is enabled

- VUID-vkCmdDrawMeshTasksNV-advancedBlendMaxColorAttachments-07480
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_COLOR_BLEND_ADVANCED_EXT` and `VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT` dynamic states enabled and the last calls to vkCmdSetColorBlendEnableEXT and vkCmdSetColorBlendAdvancedEXT have enabled advanced blending, then the number of active color attachments in the current subpass **must** not exceed `advancedBlendMaxColorAttachments`
- VUID-vkCmdDrawMeshTasksNV-primitivesGeneratedQueryWithNonZeroStreams-07481
  If the `primitivesGeneratedQueryWithNonZeroStreams` feature is not enabled and the `VK_QUERY_TYPE_PRIMITIVES_GENERATED_EXT` query is active, and the bound graphics pipeline was created with `VK_DYNAMIC_STATE_RASTERIZATION_STREAM_EXT` state enabled, the last call to vkCmdSetRasterizationStreamEXT **must** have set the `rasterizationStream` to zero
- VUID-vkCmdDrawMeshTasksNV-sampleLocationsPerPixel-07482
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT` state enabled and the `VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT` state disabled, then the `sampleLocationsPerPixel` member of `pSampleLocationsInfo` in the last call to vkCmdSetSampleLocationsEXT **must** equal the `rasterizationSamples` member of the VkPipelineMultisampleStateCreateInfo structure the bound graphics pipeline has been created with
- VUID-vkCmdDrawMeshTasksNV-sampleLocationsPerPixel-07483
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT` state enabled and the `VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT` state enabled, then the `sampleLocationsPerPixel` member of `pSampleLocationsInfo` in the last call to vkCmdSetSampleLocationsEXT **must** equal the `rasterizationSamples` parameter of the last call to vkCmdSetRasterizationSamplesEXT
- VUID-vkCmdDrawMeshTasksNV-sampleLocationsEnable-07484
  If a shader object is bound to the `VK_SHADER_STAGE_FRAGMENT_BIT` stage, or the bound graphics pipeline was created with the `VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT` state enabled, and `sampleLocationsEnable` was `VK_TRUE` in the last call to vkCmdSetSampleLocationsEnableEXT, and the current subpass has a depth/stencil attachment, then that attachment **must** have been created with the `VK_IMAGE_CREATE_SAMPLE_LOCATIONS_COMPATIBLE_DEPTH_BIT_EXT` bit set
- VUID-vkCmdDrawMeshTasksNV-sampleLocationsEnable-07485
  If a shader object is bound to the `VK_SHADER_STAGE_FRAGMENT_BIT` stage, or the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT` state enabled and the `VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT` state enabled, and if `sampleLocationsEnable` was `VK_TRUE` in the last call to vkCmdSetSampleLocationsEnableEXT, then the `sampleLocationsInfo.sampleLocationGridSize.width` in the last call to vkCmdSetSampleLocationsEXT **must** evenly divide VkMultisamplePropertiesEXT::`sampleLocationGridSize.width` as returned by vkGetPhysicalDeviceMultisamplePropertiesEXT with a `samples` parameter equaling `rasterizationSamples`

- VUID-vkCmdDrawMeshTasksNV-sampleLocationsEnable-07486
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state enabled and the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, and if sampleLocationsEnable was VK_TRUE in the last call to vkCmdSetSampleLocationsEnableEXT, then the sampleLocationsInfo.sampleLocationGridSize.height in the last call to vkCmdSetSampleLocationsEXT **must** evenly divide VkMultisamplePropertiesEXT::sampleLocationGridSize.height as returned by vkGetPhysicalDeviceMultisamplePropertiesEXT with a samples parameter equaling rasterizationSamples
- VUID-vkCmdDrawMeshTasksNV-sampleLocationsEnable-07487
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, and if sampleLocationsEnable was VK_TRUE in the last call to vkCmdSetSampleLocationsEnableEXT, the fragment shader code **must** not statically use the extended instruction InterpolateAtSample
- VUID-vkCmdDrawMeshTasksNV-sampleLocationsEnable-07936
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state disabled and the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, the sampleLocationsEnable member of a VkPipelineSampleLocationsStateCreateInfoEXT::sampleLocationsEnable in the bound graphics pipeline is VK_TRUE or VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, then, sampleLocationsInfo.sampleLocationGridSize.width **must** evenly divide VkMultisamplePropertiesEXT::sampleLocationGridSize.width as returned by vkGetPhysicalDeviceMultisamplePropertiesEXT with a samples parameter equaling the value of rasterizationSamples in the last call to vkCmdSetRasterizationSamplesEXT
- VUID-vkCmdDrawMeshTasksNV-sampleLocationsEnable-07937
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state disabled and the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, the sampleLocationsEnable member of a VkPipelineSampleLocationsStateCreateInfoEXT::sampleLocationsEnable in the bound graphics pipeline is VK_TRUE or VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, then, sampleLocationsInfo.sampleLocationGridSize.height **must** evenly divide VkMultisamplePropertiesEXT::sampleLocationGridSize.height as returned by vkGetPhysicalDeviceMultisamplePropertiesEXT with a samples parameter equaling the value of rasterizationSamples in the last call to vkCmdSetRasterizationSamplesEXT
- VUID-vkCmdDrawMeshTasksNV-sampleLocationsEnable-07938
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state disabled and the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, the sampleLocationsEnable member of a VkPipelineSampleLocationsStateCreateInfoEXT::sampleLocationsEnable in the bound graphics pipeline is VK_TRUE or VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, then, sampleLocationsInfo.sampleLocationsPerPixel **must** equal rasterizationSamples in the last call to vkCmdSetRasterizationSamplesEXT
- VUID-vkCmdDrawMeshTasksNV-coverageModulationTableEnable-07488

If a shader object is bound to any graphics stage or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COVERAGE_MODULATION_TABLE_ENABLE_NV state enabled, and the last call to vkCmdSetCoverageModulationTableEnableNV set coverageModulationTableEnable to VK_TRUE, then the coverageModulationTableCount parameter in the last call to vkCmdSetCoverageModulationTableNV **must** equal the current rasterizationSamples divided by the number of color samples in the current subpass

- VUID-vkCmdDrawMeshTasksNV-rasterizationSamples-07489
  If the VK_NV_framebuffer_mixed_samples extension is enabled, and if current subpass has a depth/stencil attachment and depth test, stencil test, or depth bounds test are enabled in the currently bound pipeline state, then the current rasterizationSamples **must** be the same as the sample count of the depth/stencil attachment

- VUID-vkCmdDrawMeshTasksNV-coverageToColorEnable-07490
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COVERAGE_TO_COLOR_ENABLE_NV state enabled and the last call to vkCmdSetCoverageToColorEnableNV set the coverageToColorEnable to VK_TRUE, then the current subpass **must** have a color attachment at the location selected by the last call to vkCmdSetCoverageToColorLocationNV coverageToColorLocation, with a VkFormat of VK_FORMAT_R8_UINT, VK_FORMAT_R8_SINT, VK_FORMAT_R16_UINT, VK_FORMAT_R16_SINT, VK_FORMAT_R32_UINT, or VK_FORMAT_R32_SINT

- VUID-vkCmdDrawMeshTasksNV-rasterizerDiscardEnable-09420
  If the VK_NV_fragment_coverage_to_color extension is enabled, and a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the last call to vkCmdSetCoverageToColorEnableNV set the coverageToColorEnable to VK_TRUE, then the current subpass **must** have a color attachment at the location selected by the last call to vkCmdSetCoverageToColorLocationNV coverageToColorLocation, with a VkFormat of VK_FORMAT_R8_UINT, VK_FORMAT_R8_SINT, VK_FORMAT_R16_UINT, VK_FORMAT_R16_SINT, VK_FORMAT_R32_UINT, or VK_FORMAT_R32_SINT

- VUID-vkCmdDrawMeshTasksNV-coverageReductionMode-07491
  If this VK_NV_coverage_reduction_mode extension is enabled, the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COVERAGE_TO_COLOR_ENABLE_NV and VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT states enabled, the current coverage reduction mode coverageReductionMode, then the current rasterizationSamples, and the sample counts for the color and depth/stencil attachments (if the subpass has them) **must** be a valid combination returned by vkGetPhysicalDeviceSupportedFramebufferMixedSamplesCombinationsNV

- VUID-vkCmdDrawMeshTasksNV-viewportCount-07492
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, but not the VK_DYNAMIC_STATE_VIEWPORT_SWIZZLE_NV dynamic state enabled, then the bound graphics pipeline **must** have been created with VkPipelineViewportSwizzleStateCreateInfoNV::viewportCount greater or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT

- VUID-vkCmdDrawMeshTasksNV-viewportCount-07493
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT` and `VK_DYNAMIC_STATE_VIEWPORT_SWIZZLE_NV` dynamic states enabled then the `viewportCount` parameter in the last call to vkCmdSetViewportSwizzleNV **must** be greater than or equal to the `viewportCount` parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawMeshTasksNV-viewportCount-09421
  If the `VK_NV_viewport_swizzle` extension is enabled, and a shader object is bound to any graphics stage, then the `viewportCount` parameter in the last call to vkCmdSetViewportSwizzleNV **must** be greater than or equal to the `viewportCount` parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawMeshTasksNV-rasterizationSamples-07494
  If the `VK_NV_framebuffer_mixed_samples` extension is enabled, and if the current subpass has any color attachments and `rasterizationSamples` of the last call to vkCmdSetRasterizationSamplesEXT is greater than the number of color samples, then the pipeline `sampleShadingEnable` **must** be `VK_FALSE`
- VUID-vkCmdDrawMeshTasksNV-stippledLineEnable-07495
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT` or `VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT` dynamic states enabled, and if the current `stippledLineEnable` state is `VK_TRUE` and the current `lineRasterizationMode` state is `VK_LINE_RASTERIZATION_MODE_RECTANGULAR_KHR`, then the `stippledRectangularLines` feature **must** be enabled
- VUID-vkCmdDrawMeshTasksNV-stippledLineEnable-07496
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT` or `VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT` dynamic states enabled, and if the current `stippledLineEnable` state is `VK_TRUE` and the current `lineRasterizationMode` state is `VK_LINE_RASTERIZATION_MODE_BRESENHAM_KHR`, then the `stippledBresenhamLines` feature **must** be enabled
- VUID-vkCmdDrawMeshTasksNV-stippledLineEnable-07497
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT` or `VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT` dynamic states enabled, and if the current `stippledLineEnable` state is `VK_TRUE` and the current `lineRasterizationMode` state is `VK_LINE_RASTERIZATION_MODE_RECTANGULAR_SMOOTH_KHR`, then the `stippledSmoothLines` feature **must** be enabled
- VUID-vkCmdDrawMeshTasksNV-stippledLineEnable-07498
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT` or `VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT` dynamic states enabled, and if the current `stippledLineEnable` state is `VK_TRUE` and the current `lineRasterizationMode` state is `VK_LINE_RASTERIZATION_MODE_DEFAULT_KHR`, then the `stippledRectangularLines` feature **must** be enabled and VkPhysicalDeviceLimits::`strictLines` **must** be `VK_TRUE`
- VUID-vkCmdDrawMeshTasksNV-conservativePointAndLineRasterization-07499

If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_CONSERVATIVE_RASTERIZATION_MODE_EXT dynamic state enabled, conservativePointAndLineRasterization is not supported, and the effective primitive topology output by the last pre-rasterization shader stage is a line or point, then the conservativeRasterizationMode set by the last call to vkCmdSetConservativeRasterizationModeEXT **must** be VK_CONSERVATIVE_RASTERIZATION_MODE_DISABLED_EXT

- VUID-vkCmdDrawMeshTasksNV-stage-07073
  If the currently bound pipeline was created with the VkPipelineShaderStageCreateInfo::stage member of an element of VkGraphicsPipelineCreateInfo::pStages set to VK_SHADER_STAGE_VERTEX_BIT, VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT, VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT or VK_SHADER_STAGE_GEOMETRY_BIT, then Mesh Shader Queries **must** not be active
- VUID-vkCmdDrawMeshTasksNV-None-08877
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_ATTACHMENT_FEEDBACK_LOOP_ENABLE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetAttachmentFeedbackLoopEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksNV-None-07850
  If dynamic state was inherited from VkCommandBufferInheritanceViewportScissorInfoNV, it **must** be set in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksNV-None-08684
  If there is no bound graphics pipeline, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_VERTEX_BIT
- VUID-vkCmdDrawMeshTasksNV-None-08685
  If there is no bound graphics pipeline, and the tessellationShader feature is enabled, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT
- VUID-vkCmdDrawMeshTasksNV-None-08686
  If there is no bound graphics pipeline, and the tessellationShader feature is enabled, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT
- VUID-vkCmdDrawMeshTasksNV-None-08687
  If there is no bound graphics pipeline, and the geometryShader feature is enabled, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_GEOMETRY_BIT
- VUID-vkCmdDrawMeshTasksNV-None-08688
  If there is no bound graphics pipeline, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_FRAGMENT_BIT
- VUID-vkCmdDrawMeshTasksNV-None-08689
  If there is no bound graphics pipeline, and the taskShader feature is enabled,

vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_TASK_BIT_EXT

- VUID-vkCmdDrawMeshTasksNV-None-08690
  If there is no bound graphics pipeline, and the meshShader feature is enabled, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_MESH_BIT_EXT
- VUID-vkCmdDrawMeshTasksNV-None-08693
  If there is no bound graphics pipeline, and at least one of the taskShader and meshShader features is enabled, one of the VK_SHADER_STAGE_VERTEX_BIT or VK_SHADER_STAGE_MESH_BIT_EXT stages **must** have a valid VkShaderEXT bound, and the other **must** have no VkShaderEXT bound
- VUID-vkCmdDrawMeshTasksNV-None-08694
  If there is no bound graphics pipeline, and both the taskShader and meshShader features are enabled, and a valid VkShaderEXT is bound the to the VK_SHADER_STAGE_MESH_BIT_EXT stage, and that VkShaderEXT was created without the VK_SHADER_CREATE_NO_TASK_SHADER_BIT_EXT flag, a valid VkShaderEXT **must** be bound to the VK_SHADER_STAGE_TASK_BIT_EXT stage
- VUID-vkCmdDrawMeshTasksNV-None-08695
  If there is no bound graphics pipeline, and both the taskShader and meshShader features are enabled, and a valid VkShaderEXT is bound the to the VK_SHADER_STAGE_MESH_BIT_EXT stage, and that VkShaderEXT was created with the VK_SHADER_CREATE_NO_TASK_SHADER_BIT_EXT flag, there **must** be no VkShaderEXT bound to the VK_SHADER_STAGE_TASK_BIT_EXT stage
- VUID-vkCmdDrawMeshTasksNV-None-08696
  If there is no bound graphics pipeline, and a valid VkShaderEXT is bound to the VK_SHADER_STAGE_VERTEX_BIT stage, there **must** be no VkShaderEXT bound to either the VK_SHADER_STAGE_TASK_BIT_EXT stage or the VK_SHADER_STAGE_MESH_BIT_EXT stage
- VUID-vkCmdDrawMeshTasksNV-None-08698
  If any graphics shader is bound which was created with the VK_SHADER_CREATE_LINK_STAGE_BIT_EXT flag, then all shaders created with the VK_SHADER_CREATE_LINK_STAGE_BIT_EXT flag in the same vkCreateShadersEXT call **must** also be bound
- VUID-vkCmdDrawMeshTasksNV-None-08699
  If any graphics shader is bound which was created with the VK_SHADER_CREATE_LINK_STAGE_BIT_EXT flag, any stages in between stages whose shaders which did not create a shader with the VK_SHADER_CREATE_LINK_STAGE_BIT_EXT flag as part of the same vkCreateShadersEXT call **must** not have any VkShaderEXT bound
- VUID-vkCmdDrawMeshTasksNV-None-08878
  All bound graphics shader objects **must** have been created with identical or identically defined push constant ranges
- VUID-vkCmdDrawMeshTasksNV-None-08879
  All bound graphics shader objects **must** have been created with identical or identically defined arrays of descriptor set layouts
- VUID-vkCmdDrawMeshTasksNV-colorAttachmentCount-09372
  If the current render pass instance was begun with vkCmdBeginRenderingKHR and a

VkRenderingInfo::colorAttachmentCount equal to 1, a color attachment with a resolve mode of VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID, and a fragment shader is bound, it **must** not declare the DepthReplacing or StencilRefReplacingEXT execution modes

- VUID-vkCmdDrawMeshTasksNV-pDynamicStates-08715
  If the bound graphics pipeline state includes a fragment shader stage, was created with VK_DYNAMIC_STATE_DEPTH_WRITE_ENABLE set in VkPipelineDynamicStateCreateInfo::pDynamicStates, and the fragment shader declares the EarlyFragmentTests execution mode and uses OpDepthAttachmentReadEXT, the depthWriteEnable parameter in the last call to vkCmdSetDepthWriteEnableEXT **must** be VK_FALSE

- VUID-vkCmdDrawMeshTasksNV-pDynamicStates-08716
  If the bound graphics pipeline state includes a fragment shader stage, was created with VK_DYNAMIC_STATE_STENCIL_WRITE_MASK set in VkPipelineDynamicStateCreateInfo::pDynamicStates, and the fragment shader declares the EarlyFragmentTests execution mode and uses OpStencilAttachmentReadEXT, the writeMask parameter in the last call to vkCmdSetStencilWriteMask **must** be 0

- VUID-vkCmdDrawMeshTasksNV-None-09116
  If a shader object is bound to any graphics stage or the currently bound graphics pipeline was created with VK_DYNAMIC_STATE_COLOR_WRITE_MASK_EXT, and the format of any color attachment is VK_FORMAT_E5B9G9R9_UFLOAT_PACK32, the corresponding element of the pColorWriteMasks parameter of vkCmdSetColorWriteMaskEXT **must** either include all of VK_COLOR_COMPONENT_R_BIT, VK_COLOR_COMPONENT_G_BIT, and VK_COLOR_COMPONENT_B_BIT, or none of them

- VUID-vkCmdDrawMeshTasksNV-maxFragmentDualSrcAttachments-09239
  If blending is enabled for any attachment where either the source or destination blend factors for that attachment use the secondary color input, the maximum value of Location for any output attachment statically used in the Fragment Execution Model executed by this command **must** be less than maxFragmentDualSrcAttachments

- VUID-vkCmdDrawMeshTasksNV-None-09548
  If the current render pass was begun with vkCmdBeginRenderingKHR, and there is no shader object bound to any graphics stage, the value of each element of VkRenderingAttachmentLocationInfoKHR::pColorAttachmentLocations set by vkCmdSetRenderingAttachmentLocationsKHR **must** match the value set for the corresponding element in the currently bound pipeline

- VUID-vkCmdDrawMeshTasksNV-None-09549
  If the current render pass was begun with vkCmdBeginRenderingKHR, and there is no shader object bound to any graphics stage, input attachment index mappings in the currently bound pipeline **must** match those set for the current render pass instance via VkRenderingInputAttachmentIndexInfoKHR

- VUID-vkCmdDrawMeshTasksNV-None-09642
  If the current render pass was begun with vkCmdBeginRenderingKHR with the VK_RENDERING_ENABLE_LEGACY_DITHERING_BIT_EXT flag, the bound graphics pipeline **must** have been created with VK_PIPELINE_CREATE_2_ENABLE_LEGACY_DITHERING_BIT_EXT

- VUID-vkCmdDrawMeshTasksNV-None-09643
  If the bound graphics pipeline was created with VK_PIPELINE_CREATE_2_ENABLE_LEGACY_DITHERING_BIT_EXT, the current render pass **must**

have begun with vkCmdBeginRenderingKHR with the VK_RENDERING_ENABLE_LEGACY_DITHERING_BIT_EXT flag

- VUID-vkCmdDrawMeshTasksNV-stage-06480
  The bound graphics pipeline **must** not have been created with the VkPipelineShaderStageCreateInfo::stage member of an element of VkGraphicsPipelineCreateInfo::pStages set to VK_SHADER_STAGE_VERTEX_BIT, VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT, VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT or VK_SHADER_STAGE_GEOMETRY_BIT
- VUID-vkCmdDrawMeshTasksNV-None-07074
  Transform Feedback Queries **must** not be active
- VUID-vkCmdDrawMeshTasksNV-None-07075
  Primitives Generated Queries **must** not be active
- VUID-vkCmdDrawMeshTasksNV-pipelineStatistics-07076
  The pipelineStatistics member used to create any active Pipeline Statistics Query **must** not contain VK_QUERY_PIPELINE_STATISTIC_INPUT_ASSEMBLY_VERTICES_BIT, VK_QUERY_PIPELINE_STATISTIC_INPUT_ASSEMBLY_PRIMITIVES_BIT, VK_QUERY_PIPELINE_STATISTIC_VERTEX_SHADER_INVOCATIONS_BIT, VK_QUERY_PIPELINE_STATISTIC_GEOMETRY_SHADER_INVOCATIONS_BIT, VK_QUERY_PIPELINE_STATISTIC_GEOMETRY_SHADER_PRIMITIVES_BIT, VK_QUERY_PIPELINE_STATISTIC_CLIPPING_INVOCATIONS_BIT, VK_QUERY_PIPELINE_STATISTIC_CLIPPING_PRIMITIVES_BIT, VK_QUERY_PIPELINE_STATISTIC_TESSELLATION_CONTROL_SHADER_PATCHES_BIT, or VK_QUERY_PIPELINE_STATISTIC_TESSELLATION_EVALUATION_SHADER_INVOCATIONS_BIT
- VUID-vkCmdDrawMeshTasksNV-taskCount-02119
  taskCount **must** be less than or equal to VkPhysicalDeviceMeshShaderPropertiesNV::maxDrawMeshTasksCount
- VUID-vkCmdDrawMeshTasksNV-MeshNV-07080
  The current pipeline bound to VK_PIPELINE_BIND_POINT_GRAPHICS **must** contain a shader stage using the MeshNV Execution Model

## Valid Usage (Implicit)

- VUID-vkCmdDrawMeshTasksNV-commandBuffer-parameter
  commandBuffer **must** be a valid VkCommandBuffer handle
- VUID-vkCmdDrawMeshTasksNV-commandBuffer-recording
  commandBuffer **must** be in the recording state
- VUID-vkCmdDrawMeshTasksNV-commandBuffer-cmdpool
  The VkCommandPool that commandBuffer was allocated from **must** support graphics operations
- VUID-vkCmdDrawMeshTasksNV-renderpass
  This command **must** only be called inside of a render pass instance
- VUID-vkCmdDrawMeshTasksNV-videocoding

This command **must** only be called outside of a video coding scope

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Inside | Outside | Graphics | Action |

To record an indirect mesh tasks drawing command, call:

```
// Provided by VK_NV_mesh_shader
void vkCmdDrawMeshTasksIndirectNV(
    VkCommandBuffer                             commandBuffer,
    VkBuffer                                    buffer,
    VkDeviceSize                                offset,
    uint32_t                                    drawCount,
    uint32_t                                    stride);
```

- `commandBuffer` is the command buffer into which the command is recorded.
- `buffer` is the buffer containing draw parameters.
- `offset` is the byte offset into `buffer` where parameters begin.
- `drawCount` is the number of draws to execute, and **can** be zero.
- `stride` is the byte stride between successive sets of draw parameters.

`vkCmdDrawMeshTasksIndirectNV` behaves similarly to vkCmdDrawMeshTasksNV except that the parameters are read by the device from a buffer during execution. `drawCount` draws are executed by the command, with parameters taken from `buffer` starting at `offset` and increasing by `stride` bytes for each successive draw. The parameters of each draw are encoded in an array of VkDrawMeshTasksIndirectCommandNV structures. If `drawCount` is less than or equal to one, `stride` is ignored.

## Valid Usage

- VUID-vkCmdDrawMeshTasksIndirectNV-magFilter-04553
  If a VkSampler created with `magFilter` or `minFilter` equal to `VK_FILTER_LINEAR`,

reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE, and compareEnable equal to VK_FALSE is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT

- VUID-vkCmdDrawMeshTasksIndirectNV-magFilter-09598
  If a VkSampler created with magFilter or minFilter equal to VK_FILTER_LINEAR and reductionMode equal to either VK_SAMPLER_REDUCTION_MODE_MIN or VK_SAMPLER_REDUCTION_MODE_MAX is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT
- VUID-vkCmdDrawMeshTasksIndirectNV-mipmapMode-04770
  If a VkSampler created with mipmapMode equal to VK_SAMPLER_MIPMAP_MODE_LINEAR, reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE, and compareEnable equal to VK_FALSE is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT
- VUID-vkCmdDrawMeshTasksIndirectNV-mipmapMode-09599
  If a VkSampler created with mipmapMode equal to VK_SAMPLER_MIPMAP_MODE_LINEAR and reductionMode equal to either VK_SAMPLER_REDUCTION_MODE_MIN or VK_SAMPLER_REDUCTION_MODE_MAX is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT
- VUID-vkCmdDrawMeshTasksIndirectNV-unnormalizedCoordinates-09635
  If a VkSampler created with unnormalizedCoordinates equal to VK_TRUE is used to sample a VkImageView as a result of this command, then the image view's levelCount and layerCount **must** be 1
- VUID-vkCmdDrawMeshTasksIndirectNV-unnormalizedCoordinates-09636
  If a VkSampler created with unnormalizedCoordinates equal to VK_TRUE is used to sample a VkImageView as a result of this command, then the image view's viewType **must** be VK_IMAGE_VIEW_TYPE_1D or VK_IMAGE_VIEW_TYPE_2D
- VUID-vkCmdDrawMeshTasksIndirectNV-None-06479
  If a VkImageView is sampled with depth comparison, the image view's format features **must** contain VK_FORMAT_FEATURE_2_SAMPLED_IMAGE_DEPTH_COMPARISON_BIT
- VUID-vkCmdDrawMeshTasksIndirectNV-None-02691
  If a VkImageView is accessed using atomic operations as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_STORAGE_IMAGE_ATOMIC_BIT
- VUID-vkCmdDrawMeshTasksIndirectNV-None-07888
  If a VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER descriptor is accessed using atomic operations as a result of this command, then the storage texel buffer's format features **must** contain VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_ATOMIC_BIT
- VUID-vkCmdDrawMeshTasksIndirectNV-None-02692
  If a VkImageView is sampled with VK_FILTER_CUBIC_EXT as a result of this command, then the image view's format features **must** contain

VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_CUBIC_BIT_EXT

- VUID-vkCmdDrawMeshTasksIndirectNV-None-02693
  If the VK_EXT_filter_cubic extension is not enabled and any VkImageView is sampled with VK_FILTER_CUBIC_EXT as a result of this command, it **must** not have a VkImageViewType of VK_IMAGE_VIEW_TYPE_3D, VK_IMAGE_VIEW_TYPE_CUBE, or VK_IMAGE_VIEW_TYPE_CUBE_ARRAY
- VUID-vkCmdDrawMeshTasksIndirectNV-filterCubic-02694
  Any VkImageView being sampled with VK_FILTER_CUBIC_EXT as a result of this command **must** have a VkImageViewType and format that supports cubic filtering, as specified by VkFilterCubicImageViewImageFormatPropertiesEXT::filterCubic returned by vkGetPhysicalDeviceImageFormatProperties2
- VUID-vkCmdDrawMeshTasksIndirectNV-filterCubicMinmax-02695
  Any VkImageView being sampled with VK_FILTER_CUBIC_EXT with a reduction mode of either VK_SAMPLER_REDUCTION_MODE_MIN or VK_SAMPLER_REDUCTION_MODE_MAX as a result of this command **must** have a VkImageViewType and format that supports cubic filtering together with minmax filtering, as specified by VkFilterCubicImageViewImageFormatPropertiesEXT::filterCubicMinmax returned by vkGetPhysicalDeviceImageFormatProperties2
- VUID-vkCmdDrawMeshTasksIndirectNV-cubicRangeClamp-09212
  If the cubicRangeClamp feature is not enabled, then any VkImageView being sampled with VK_FILTER_CUBIC_EXT as a result of this command **must** not have a VkSamplerReductionModeCreateInfo::reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE_RANGECLAMP_QCOM
- VUID-vkCmdDrawMeshTasksIndirectNV-reductionMode-09213
  Any VkImageView being sampled with a VkSamplerReductionModeCreateInfo::reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE_RANGECLAMP_QCOM as a result of this command **must** sample with VK_FILTER_CUBIC_EXT
- VUID-vkCmdDrawMeshTasksIndirectNV-selectableCubicWeights-09214
  If the selectableCubicWeights feature is not enabled, then any VkImageView being sampled with VK_FILTER_CUBIC_EXT as a result of this command **must** have VkSamplerCubicWeightsCreateInfoQCOM::cubicWeights equal to VK_CUBIC_FILTER_WEIGHTS_CATMULL_ROM_QCOM
- VUID-vkCmdDrawMeshTasksIndirectNV-flags-02696
  Any VkImage created with a VkImageCreateInfo::flags containing VK_IMAGE_CREATE_CORNER_SAMPLED_BIT_NV sampled as a result of this command **must** only be sampled using a VkSamplerAddressMode of VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_EDGE
- VUID-vkCmdDrawMeshTasksIndirectNV-OpTypeImage-07027
  For any VkImageView being written as a storage image where the image format field of the OpTypeImage is Unknown, the view's format features **must** contain VK_FORMAT_FEATURE_2_STORAGE_WRITE_WITHOUT_FORMAT_BIT
- VUID-vkCmdDrawMeshTasksIndirectNV-OpTypeImage-07028
  For any VkImageView being read as a storage image where the image format field of the OpTypeImage is Unknown, the view's format features **must** contain VK_FORMAT_FEATURE_2_STORAGE_READ_WITHOUT_FORMAT_BIT
- VUID-vkCmdDrawMeshTasksIndirectNV-OpTypeImage-07029

For any VkBufferView being written as a storage texel buffer where the image format field of the OpTypeImage is Unknown, the view's buffer features **must** contain VK_FORMAT_FEATURE_2_STORAGE_WRITE_WITHOUT_FORMAT_BIT

- VUID-vkCmdDrawMeshTasksIndirectNV-OpTypeImage-07030
  Any VkBufferView being read as a storage texel buffer where the image format field of the OpTypeImage is Unknown then the view's buffer features **must** contain VK_FORMAT_FEATURE_2_STORAGE_READ_WITHOUT_FORMAT_BIT

- VUID-vkCmdDrawMeshTasksIndirectNV-None-08600
  For each set *n* that is statically used by a bound shader, a descriptor set **must** have been bound to *n* at the same pipeline bind point, with a VkPipelineLayout that is compatible for set *n*, with the VkPipelineLayout used to create the current VkPipeline or the VkDescriptorSetLayout array used to create the current VkShaderEXT , as described in Pipeline Layout Compatibility

- VUID-vkCmdDrawMeshTasksIndirectNV-None-08601
  For each push constant that is statically used by a bound shader, a push constant value **must** have been set for the same pipeline bind point, with a VkPipelineLayout that is compatible for push constants, with the VkPipelineLayout used to create the current VkPipeline or the VkDescriptorSetLayout array used to create the current VkShaderEXT , as described in Pipeline Layout Compatibility

- VUID-vkCmdDrawMeshTasksIndirectNV-None-10068
  For each array of resources that is used by a bound shader, the indices used to access members of the array **must** be less than the descriptor count for the identified binding in the descriptor sets used by this command

- VUID-vkCmdDrawMeshTasksIndirectNV-maintenance4-08602
  If the maintenance4 feature is not enabled, then for each push constant that is statically used by a bound shader, a push constant value **must** have been set for the same pipeline bind point, with a VkPipelineLayout that is compatible for push constants, with the VkPipelineLayout used to create the current VkPipeline or the VkDescriptorSetLayout and VkPushConstantRange arrays used to create the current VkShaderEXT , as described in Pipeline Layout Compatibility

- VUID-vkCmdDrawMeshTasksIndirectNV-None-08114
  Descriptors in each bound descriptor set, specified via vkCmdBindDescriptorSets, **must** be valid as described by descriptor validity if they are statically used by the VkPipeline bound to the pipeline bind point used by this command and the bound VkPipeline was not created with VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT

- VUID-vkCmdDrawMeshTasksIndirectNV-None-08115
  If the descriptors used by the VkPipeline bound to the pipeline bind point were specified via vkCmdBindDescriptorSets, the bound VkPipeline **must** have been created without VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT

- VUID-vkCmdDrawMeshTasksIndirectNV-None-08116
  Descriptors in bound descriptor buffers, specified via vkCmdSetDescriptorBufferOffsetsEXT, **must** be valid if they are dynamically used by the VkPipeline bound to the pipeline bind point used by this command and the bound VkPipeline was created with VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT

- VUID-vkCmdDrawMeshTasksIndirectNV-None-08604
  Descriptors in bound descriptor buffers, specified via vkCmdSetDescriptorBufferOffsetsEXT, **must** be valid if they are dynamically used by any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command
- VUID-vkCmdDrawMeshTasksIndirectNV-None-08117
  If the descriptors used by the VkPipeline bound to the pipeline bind point were specified via vkCmdSetDescriptorBufferOffsetsEXT, the bound VkPipeline **must** have been created with `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`
- VUID-vkCmdDrawMeshTasksIndirectNV-None-08119
  If a descriptor is dynamically used with a VkPipeline created with `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`, the descriptor memory **must** be resident
- VUID-vkCmdDrawMeshTasksIndirectNV-None-08605
  If a descriptor is dynamically used with a VkShaderEXT created with a `VkDescriptorSetLayout` that was created with `VK_DESCRIPTOR_SET_LAYOUT_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`, the descriptor memory **must** be resident
- VUID-vkCmdDrawMeshTasksIndirectNV-None-08606
  If the `shaderObject` feature is not enabled, a valid pipeline **must** be bound to the pipeline bind point used by this command
- VUID-vkCmdDrawMeshTasksIndirectNV-None-08608
  If a pipeline is bound to the pipeline bind point used by this command, there **must** not have been any calls to dynamic state setting commands for any state specified statically in the VkPipeline object bound to the pipeline bind point used by this command, since that pipeline was bound
- VUID-vkCmdDrawMeshTasksIndirectNV-None-08609
  If the VkPipeline object bound to the pipeline bind point used by this command or any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a VkSampler object that uses unnormalized coordinates, that sampler **must** not be used to sample from any VkImage with a VkImageView of the type `VK_IMAGE_VIEW_TYPE_3D`, `VK_IMAGE_VIEW_TYPE_CUBE`, `VK_IMAGE_VIEW_TYPE_1D_ARRAY`, `VK_IMAGE_VIEW_TYPE_2D_ARRAY` or `VK_IMAGE_VIEW_TYPE_CUBE_ARRAY`, in any shader stage
- VUID-vkCmdDrawMeshTasksIndirectNV-None-08610
  If the VkPipeline object bound to the pipeline bind point used by this command or any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a VkSampler object that uses unnormalized coordinates, that sampler **must** not be used with any of the SPIR-V `OpImageSample*` or `OpImageSparseSample*` instructions with `ImplicitLod`, `Dref` or `Proj` in their name, in any shader stage
- VUID-vkCmdDrawMeshTasksIndirectNV-None-08611
  If the VkPipeline object bound to the pipeline bind point used by this command or any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a VkSampler object that uses unnormalized coordinates, that sampler **must** not be used with any of the SPIR-V `OpImageSample*` or `OpImageSparseSample*` instructions that includes a LOD bias or any offset values, in any shader stage

- VUID-vkCmdDrawMeshTasksIndirectNV-None-08607
  If the `shaderObject` is enabled, either a valid pipeline **must** be bound to the pipeline bind point used by this command, or a valid combination of valid and VK_NULL_HANDLE shader objects **must** be bound to every supported shader stage corresponding to the pipeline bind point used by this command
- VUID-vkCmdDrawMeshTasksIndirectNV-uniformBuffers-06935
  If any stage of the VkPipeline object bound to the pipeline bind point used by this command accesses a uniform buffer, and that stage was created without enabling either `VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_EXT` or `VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_2_EXT` for `uniformBuffers`, and the `robustBufferAccess` feature is not enabled, that stage **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point
- VUID-vkCmdDrawMeshTasksIndirectNV-None-08612
  If the `robustBufferAccess` feature is not enabled, and any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a uniform buffer, it **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point
- VUID-vkCmdDrawMeshTasksIndirectNV-storageBuffers-06936
  If any stage of the VkPipeline object bound to the pipeline bind point used by this command accesses a storage buffer, and that stage was created without enabling either `VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_EXT` or `VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_2_EXT` for `storageBuffers`, and the `robustBufferAccess` feature is not enabled, that stage **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point
- VUID-vkCmdDrawMeshTasksIndirectNV-None-08613
  If the `robustBufferAccess` feature is not enabled, and any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a storage buffer, it **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point
- VUID-vkCmdDrawMeshTasksIndirectNV-commandBuffer-02707
  If `commandBuffer` is an unprotected command buffer and `protectedNoFault` is not supported, any resource accessed by bound shaders **must** not be a protected resource
- VUID-vkCmdDrawMeshTasksIndirectNV-None-06550
  If a bound shader accesses a VkSampler or VkImageView object that enables sampler $Y'C_BC_R$ conversion, that object **must** only be used with `OpImageSample*` or `OpImageSparseSample*` instructions
- VUID-vkCmdDrawMeshTasksIndirectNV-ConstOffset-06551
  If a bound shader accesses a VkSampler or VkImageView object that enables sampler $Y'C_BC_R$ conversion, that object **must** not use the `ConstOffset` and `Offset` operands
- VUID-vkCmdDrawMeshTasksIndirectNV-viewType-07752
  If a VkImageView is accessed as a result of this command, then the image view's `viewType` **must** match the `Dim` operand of the `OpTypeImage` as described in Compatibility Between SPIR-V Image Dimensions and Vulkan ImageView Types

- VUID-vkCmdDrawMeshTasksIndirectNV-format-07753
  If a VkImageView is accessed as a result of this command, then the numeric type of the image view's `format` and the `Sampled Type` operand of the `OpTypeImage` **must** match
- VUID-vkCmdDrawMeshTasksIndirectNV-OpImageWrite-08795
  If a VkImageView created with a format other than `VK_FORMAT_A8_UNORM_KHR` is accessed using `OpImageWrite` as a result of this command, then the `Type` of the `Texel` operand of that instruction **must** have at least as many components as the image view's format
- VUID-vkCmdDrawMeshTasksIndirectNV-OpImageWrite-08796
  If a VkImageView created with the format `VK_FORMAT_A8_UNORM_KHR` is accessed using `OpImageWrite` as a result of this command, then the `Type` of the `Texel` operand of that instruction **must** have four components
- VUID-vkCmdDrawMeshTasksIndirectNV-OpImageWrite-04469
  If a VkBufferView is accessed using `OpImageWrite` as a result of this command, then the `Type` of the `Texel` operand of that instruction **must** have at least as many components as the buffer view's format
- VUID-vkCmdDrawMeshTasksIndirectNV-SampledType-04470
  If a VkImageView with a VkFormat that has a 64-bit component width is accessed as a result of this command, the `SampledType` of the `OpTypeImage` operand of that instruction **must** have a `Width` of 64
- VUID-vkCmdDrawMeshTasksIndirectNV-SampledType-04471
  If a VkImageView with a VkFormat that has a component width less than 64-bit is accessed as a result of this command, the `SampledType` of the `OpTypeImage` operand of that instruction **must** have a `Width` of 32
- VUID-vkCmdDrawMeshTasksIndirectNV-SampledType-04472
  If a VkBufferView with a VkFormat that has a 64-bit component width is accessed as a result of this command, the `SampledType` of the `OpTypeImage` operand of that instruction **must** have a `Width` of 64
- VUID-vkCmdDrawMeshTasksIndirectNV-SampledType-04473
  If a VkBufferView with a VkFormat that has a component width less than 64-bit is accessed as a result of this command, the `SampledType` of the `OpTypeImage` operand of that instruction **must** have a `Width` of 32
- VUID-vkCmdDrawMeshTasksIndirectNV-sparseImageInt64Atomics-04474
  If the `sparseImageInt64Atomics` feature is not enabled, VkImage objects created with the `VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT` flag **must** not be accessed by atomic instructions through an `OpTypeImage` with a `SampledType` with a `Width` of 64 by this command
- VUID-vkCmdDrawMeshTasksIndirectNV-sparseImageInt64Atomics-04475
  If the `sparseImageInt64Atomics` feature is not enabled, VkBuffer objects created with the `VK_BUFFER_CREATE_SPARSE_RESIDENCY_BIT` flag **must** not be accessed by atomic instructions through an `OpTypeImage` with a `SampledType` with a `Width` of 64 by this command
- VUID-vkCmdDrawMeshTasksIndirectNV-OpImageWeightedSampleQCOM-06971
  If `OpImageWeightedSampleQCOM` is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_2_WEIGHT_SAMPLED_IMAGE_BIT_QCOM`

- VUID-vkCmdDrawMeshTasksIndirectNV-OpImageWeightedSampleQCOM-06972
  If `OpImageWeightedSampleQCOM` uses a VkImageView as a sample weight image as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_2_WEIGHT_IMAGE_BIT_QCOM`
- VUID-vkCmdDrawMeshTasksIndirectNV-OpImageBoxFilterQCOM-06973
  If `OpImageBoxFilterQCOM` is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_2_BOX_FILTER_SAMPLED_BIT_QCOM`
- VUID-vkCmdDrawMeshTasksIndirectNV-OpImageBlockMatchSSDQCOM-06974
  If `OpImageBlockMatchSSDQCOM` is used to read from an VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM`
- VUID-vkCmdDrawMeshTasksIndirectNV-OpImageBlockMatchSADQCOM-06975
  If `OpImageBlockMatchSADQCOM` is used to read from an VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM`
- VUID-vkCmdDrawMeshTasksIndirectNV-OpImageBlockMatchSADQCOM-06976
  If `OpImageBlockMatchSADQCOM` or OpImageBlockMatchSSDQCOM is used to read from a reference image as result of this command, then the specified reference coordinates **must** not fail integer texel coordinate validation
- VUID-vkCmdDrawMeshTasksIndirectNV-OpImageWeightedSampleQCOM-06977
  If `OpImageWeightedSampleQCOM`, `OpImageBoxFilterQCOM`, `OpImageBlockMatchWindowSSDQCOM`, `OpImageBlockMatchWindowSADQCOM`, `OpImageBlockMatchGatherSSDQCOM`, `OpImageBlockMatchGatherSADQCOM`, `OpImageBlockMatchSSDQCOM`, or `OpImageBlockMatchSADQCOM` uses a VkSampler as a result of this command, then the sampler **must** have been created with `VK_SAMPLER_CREATE_IMAGE_PROCESSING_BIT_QCOM`
- VUID-vkCmdDrawMeshTasksIndirectNV-OpImageWeightedSampleQCOM-06978
  If any command other than `OpImageWeightedSampleQCOM`, `OpImageBoxFilterQCOM`, `OpImageBlockMatchWindowSSDQCOM`, `OpImageBlockMatchWindowSADQCOM`, `OpImageBlockMatchGatherSSDQCOM`, `OpImageBlockMatchGatherSADQCOM`, `OpImageBlockMatchSSDQCOM`, or `OpImageBlockMatchSADQCOM` uses a VkSampler as a result of this command, then the sampler **must** not have been created with `VK_SAMPLER_CREATE_IMAGE_PROCESSING_BIT_QCOM`
- VUID-vkCmdDrawMeshTasksIndirectNV-OpImageBlockMatchWindow-09215
  If a `OpImageBlockMatchWindow*QCOM` or `OpImageBlockMatchGather*QCOM` instruction is used to read from an VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM`
- VUID-vkCmdDrawMeshTasksIndirectNV-OpImageBlockMatchWindow-09216
  If a `OpImageBlockMatchWindow*QCOM` or `OpImageBlockMatchGather*QCOM` instruction is used to read from an VkImageView as a result of this command, then the image view's format **must** be a single-component format
- VUID-vkCmdDrawMeshTasksIndirectNV-OpImageBlockMatchWindow-09217
  If a `OpImageBlockMatchWindow*QCOM` or `OpImageBlockMatchGather*QCOM` read from a reference image as result of this command, then the specified reference coordinates **must** not fail

integer texel coordinate validation

- VUID-vkCmdDrawMeshTasksIndirectNV-None-07288
  Any shader invocation executed by this command **must** terminate
- VUID-vkCmdDrawMeshTasksIndirectNV-None-09600
  If a descriptor with type equal to any of `VK_DESCRIPTOR_TYPE_SAMPLE_WEIGHT_IMAGE_QCOM`, `VK_DESCRIPTOR_TYPE_BLOCK_MATCH_IMAGE_QCOM`, `VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE`, `VK_DESCRIPTOR_TYPE_STORAGE_IMAGE`, or `VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT` is accessed as a result of this command, the image subresource identified by that descriptor **must** be in the image layout identified when the descriptor was written
- VUID-vkCmdDrawMeshTasksIndirectNV-renderPass-02684
  The current render pass **must** be compatible with the `renderPass` member of the `VkGraphicsPipelineCreateInfo` structure specified when creating the `VkPipeline` bound to `VK_PIPELINE_BIND_POINT_GRAPHICS`
- VUID-vkCmdDrawMeshTasksIndirectNV-subpass-02685
  The subpass index of the current render pass **must** be equal to the `subpass` member of the `VkGraphicsPipelineCreateInfo` structure specified when creating the `VkPipeline` bound to `VK_PIPELINE_BIND_POINT_GRAPHICS`
- VUID-vkCmdDrawMeshTasksIndirectNV-None-07748
  If any shader statically accesses an input attachment, a valid descriptor **must** be bound to the pipeline via a descriptor set
- VUID-vkCmdDrawMeshTasksIndirectNV-OpTypeImage-07468
  If any shader executed by this pipeline accesses an `OpTypeImage` variable with a `Dim` operand of `SubpassData`, it **must** be decorated with an `InputAttachmentIndex` that corresponds to a valid input attachment in the current subpass
- VUID-vkCmdDrawMeshTasksIndirectNV-None-07469
  Input attachment views accessed in a subpass **must** be created with the same VkFormat as the corresponding subpass definition, and be created with a VkImageView that is compatible with the attachment referenced by the subpass' `pInputAttachments`[`InputAttachmentIndex`] in the currently bound VkFramebuffer as specified by Fragment Input Attachment Compatibility
- VUID-vkCmdDrawMeshTasksIndirectNV-pDepthInputAttachmentIndex-09595
  Input attachment views accessed in a dynamic render pass with a `InputAttachmentIndex` referenced by VkRenderingInputAttachmentIndexInfoKHR, or no `InputAttachmentIndex` if VkRenderingInputAttachmentIndexInfoKHR:`pDepthInputAttachmentIndex` or VkRenderingInputAttachmentIndexInfoKHR:`pStencilInputAttachmentIndex` are `NULL`, **must** be created with a VkImageView that is compatible with the corresponding color, depth, or stencil attachment in VkRenderingInfo
- VUID-vkCmdDrawMeshTasksIndirectNV-pDepthInputAttachmentIndex-09596
  Input attachment views accessed in a dynamic render pass via a shader object **must** have an `InputAttachmentIndex` if both VkRenderingInputAttachmentIndexInfoKHR:`pDepthInputAttachmentIndex` and VkRenderingInputAttachmentIndexInfoKHR:`pStencilInputAttachmentIndex` are non-`NULL`
- VUID-vkCmdDrawMeshTasksIndirectNV-InputAttachmentIndex-09597
  If an input attachment view accessed in a dynamic render pass via a shader object has an

InputAttachmentIndex, the InputAttachmentIndex **must** match an index in VkRenderingInputAttachmentIndexInfoKHR

- VUID-vkCmdDrawMeshTasksIndirectNV-None-06537
  Memory backing image subresources used as attachments in the current render pass **must** not be written in any way other than as an attachment by this command
- VUID-vkCmdDrawMeshTasksIndirectNV-None-09000
  If a color attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass, it is not in the VK_IMAGE_LAYOUT_ATTACHMENT_FEEDBACK_LOOP_OPTIMAL_EXT image layout, and either:
  - the VK_PIPELINE_CREATE_COLOR_ATTACHMENT_FEEDBACK_LOOP_BIT_EXT is set on the currently bound pipeline or
  - the last call to vkCmdSetAttachmentFeedbackLoopEnableEXT included VK_IMAGE_ASPECT_COLOR_BIT and
    - there is no currently bound graphics pipeline or
    - the currently bound graphics pipeline was created with VK_DYNAMIC_STATE_ATTACHMENT_FEEDBACK_LOOP_ENABLE_EXT

  it **must** not be accessed in any way other than as an attachment by this command
- VUID-vkCmdDrawMeshTasksIndirectNV-None-09001
  If a depth attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass, it is not in the VK_IMAGE_LAYOUT_ATTACHMENT_FEEDBACK_LOOP_OPTIMAL_EXT image layout, and either:
  - the VK_PIPELINE_CREATE_DEPTH_STENCIL_ATTACHMENT_FEEDBACK_LOOP_BIT_EXT is set on the currently bound pipeline or
  - the last call to vkCmdSetAttachmentFeedbackLoopEnableEXT included VK_IMAGE_ASPECT_DEPTH_BIT and
    - there is no currently bound graphics pipeline or
    - the currently bound graphics pipeline was created with VK_DYNAMIC_STATE_ATTACHMENT_FEEDBACK_LOOP_ENABLE_EXT

  it **must** not be accessed in any way other than as an attachment by this command
- VUID-vkCmdDrawMeshTasksIndirectNV-None-09002
  If a stencil attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass, it is not in the VK_IMAGE_LAYOUT_ATTACHMENT_FEEDBACK_LOOP_OPTIMAL_EXT image layout, and either:
  - the VK_PIPELINE_CREATE_DEPTH_STENCIL_ATTACHMENT_FEEDBACK_LOOP_BIT_EXT is set on the currently bound pipeline or
  - the last call to vkCmdSetAttachmentFeedbackLoopEnableEXT included VK_IMAGE_ASPECT_STENCIL_BIT and
    - there is no currently bound graphics pipeline or
    - the currently bound graphics pipeline was created with

VK_DYNAMIC_STATE_ATTACHMENT_FEEDBACK_LOOP_ENABLE_EXT

it **must** not be accessed in any way other than as an attachment by this command

- VUID-vkCmdDrawMeshTasksIndirectNV-None-09003
  If an attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass, it **must** not be accessed in any way other than as an attachment, storage image, or sampled image by this command
- VUID-vkCmdDrawMeshTasksIndirectNV-None-06539
  If any previously recorded command in the current subpass accessed an image subresource used as an attachment in this subpass in any way other than as an attachment, this command **must** not write to that image subresource as an attachment
- VUID-vkCmdDrawMeshTasksIndirectNV-None-06886
  If the current render pass instance uses a depth/stencil attachment with a read-only layout for the depth aspect, depth writes **must** be disabled
- VUID-vkCmdDrawMeshTasksIndirectNV-None-06887
  If the current render pass instance uses a depth/stencil attachment with a read-only layout for the stencil aspect, both front and back `writeMask` are not zero, and stencil test is enabled, all stencil ops **must** be `VK_STENCIL_OP_KEEP`
- VUID-vkCmdDrawMeshTasksIndirectNV-None-07831
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VIEWPORT` dynamic state enabled then vkCmdSetViewport **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectNV-None-07832
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_SCISSOR` dynamic state enabled then vkCmdSetScissor **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectNV-None-07833
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_LINE_WIDTH` dynamic state enabled then vkCmdSetLineWidth **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectNV-None-08617
  If a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE`, and the most recent call to vkCmdSetPolygonModeEXT in the current command buffer set `polygonMode` to `VK_POLYGON_MODE_LINE`, vkCmdSetLineWidth **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectNV-None-08618
  If a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE`, and the most recent call to vkCmdSetPrimitiveTopologyEXT in the current command buffer set `primitiveTopology` to any line topology, vkCmdSetLineWidth **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectNV-None-08619
  If a shader object that outputs line primitives is bound to the VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT or VK_SHADER_STAGE_GEOMETRY_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, vkCmdSetLineWidth **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectNV-None-07834
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_BIAS dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of depthBiasEnable is VK_TRUE, then vkCmdSetDepthBounds or vkCmdSetDepthBias2EXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectNV-None-07835
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_BLEND_CONSTANTS dynamic state enabled then vkCmdSetBlendConstants **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectNV-None-08621
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetColorBlendEnableEXT in the current command buffer set any element of pColorBlendEnables to VK_TRUE, and the most recent call to vkCmdSetColorBlendEquationEXT in the current command buffer set the same element of pColorBlendEquations to a VkColorBlendEquationEXT structure with any VkBlendFactor member with a value of VK_BLEND_FACTOR_CONSTANT_COLOR, VK_BLEND_FACTOR_ONE_MINUS_CONSTANT_COLOR, VK_BLEND_FACTOR_CONSTANT_ALPHA, or VK_BLEND_FACTOR_ONE_MINUS_CONSTANT_ALPHA, vkCmdSetBlendConstants **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectNV-None-07836
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_BOUNDS dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of depthBoundsTestEnable is VK_TRUE, then vkCmdSetDepthBounds **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectNV-None-07837
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_STENCIL_COMPARE_MASK dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of stencilTestEnable is VK_TRUE, then vkCmdSetStencilCompareMask **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectNV-None-07838
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_STENCIL_WRITE_MASK dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of stencilTestEnable is VK_TRUE, then vkCmdSetStencilWriteMask **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectNV-None-07839
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_STENCIL_REFERENCE dynamic state enabled, the current value of and rasterizerDiscardEnable is VK_FALSE, the current value of stencilTestEnable is VK_TRUE, then vkCmdSetStencilReference **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectNV-maxMultiviewInstanceIndex-02688
  If the draw is recorded in a render pass instance with multiview enabled, the maximum instance index **must** be less than or equal to VkPhysicalDeviceMultiviewProperties::maxMultiviewInstanceIndex
- VUID-vkCmdDrawMeshTasksIndirectNV-sampleLocationsEnable-02689
  If the bound graphics pipeline was created with VkPipelineSampleLocationsStateCreateInfoEXT::sampleLocationsEnable set to VK_TRUE and the current subpass has a depth/stencil attachment, then that attachment **must** have been created with the VK_IMAGE_CREATE_SAMPLE_LOCATIONS_COMPATIBLE_DEPTH_BIT_EXT bit set
- VUID-vkCmdDrawMeshTasksIndirectNV-None-06666
  If the VK_EXT_sample_locations extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of sampleLocationsEnable is VK_TRUE, then vkCmdSetSampleLocationsEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectNV-None-07840
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_CULL_MODE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetCullModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectNV-None-07841
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_FRONT_FACE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetFrontFaceEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectNV-None-07843
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_TEST_ENABLE dynamic state enabled, and the

current value of rasterizerDiscardEnable is VK_FALSE, vkCmdSetDepthTestEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectNV-None-07844
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_WRITE_ENABLE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetDepthWriteEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectNV-None-07845
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_COMPARE_OP dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of depthTestEnable is VK_TRUE, then vkCmdSetDepthCompareOpEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectNV-None-07846
  If the depthBounds feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_BOUNDS_TEST_ENABLE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetDepthBoundsTestEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectNV-None-07847
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_STENCIL_TEST_ENABLE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetStencilTestEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectNV-None-07848
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_STENCIL_OP dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, the current value of stencilTestEnable is VK_TRUE, then vkCmdSetStencilOpEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectNV-viewportCount-03417
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, but not the VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT dynamic state enabled, then vkCmdSetViewportWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the viewportCount parameter of vkCmdSetViewportWithCountEXT **must** match the VkPipelineViewportStateCreateInfo::scissorCount of the pipeline
- VUID-vkCmdDrawMeshTasksIndirectNV-scissorCount-03418
  If the bound graphics pipeline state was created with the

VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT dynamic state enabled, but not the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, then vkCmdSetScissorWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the scissorCount parameter of vkCmdSetScissorWithCountEXT **must** match the VkPipelineViewportStateCreateInfo::viewportCount of the pipeline

- VUID-vkCmdDrawMeshTasksIndirectNV-viewportCount-03419
  If the bound graphics pipeline state was created with both the VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT and VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic states enabled then both vkCmdSetViewportWithCountEXT and vkCmdSetScissorWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the viewportCount parameter of vkCmdSetViewportWithCountEXT **must** match the scissorCount parameter of vkCmdSetScissorWithCountEXT

- VUID-vkCmdDrawMeshTasksIndirectNV-None-08635
  If a shader object is bound to any graphics stage, then both vkCmdSetViewportWithCountEXT and vkCmdSetScissorWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the viewportCount parameter of vkCmdSetViewportWithCountEXT **must** match the scissorCount parameter of vkCmdSetScissorWithCountEXT

- VUID-vkCmdDrawMeshTasksIndirectNV-viewportCount-04137
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, but not the VK_DYNAMIC_STATE_VIEWPORT_W_SCALING_NV dynamic state enabled, then the bound graphics pipeline **must** have been created with VkPipelineViewportWScalingStateCreateInfoNV::viewportCount greater or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT

- VUID-vkCmdDrawMeshTasksIndirectNV-viewportCount-04138
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT and VK_DYNAMIC_STATE_VIEWPORT_W_SCALING_NV dynamic states enabled then the viewportCount parameter in the last call to vkCmdSetViewportWScalingNV **must** be greater than or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT

- VUID-vkCmdDrawMeshTasksIndirectNV-None-09232
  If the VK_NV_clip_space_w_scaling extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetViewportWScalingEnableNV in the current command buffer set viewportWScalingEnable to VK_TRUE, then vkCmdSetViewportWScalingNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectNV-None-08636
  If the VK_NV_clip_space_w_scaling extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetViewportWScalingEnableNV in the current command buffer set viewportWScalingEnable to VK_TRUE, then the viewportCount parameter in the last call to vkCmdSetViewportWScalingNV **must** be greater than or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT

- VUID-vkCmdDrawMeshTasksIndirectNV-viewportCount-04139
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT` dynamic state enabled, but not the `VK_DYNAMIC_STATE_VIEWPORT_SHADING_RATE_PALETTE_NV` dynamic state enabled, then the bound graphics pipeline **must** have been created with VkPipelineViewportShadingRateImageStateCreateInfoNV::`viewportCount` greater or equal to the `viewportCount` parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawMeshTasksIndirectNV-viewportCount-04140
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT` and `VK_DYNAMIC_STATE_VIEWPORT_SHADING_RATE_PALETTE_NV` dynamic states enabled then the `viewportCount` parameter in the last call to vkCmdSetViewportShadingRatePaletteNV **must** be greater than or equal to the `viewportCount` parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawMeshTasksIndirectNV-shadingRateImage-09233
  If the `shadingRateImage` feature is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE`, then vkCmdSetCoarseSampleOrderNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectNV-shadingRateImage-09234
  If the `shadingRateImage` feature is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE`, and the most recent call to vkCmdSetShadingRateImageEnableNV in the current command buffer set `shadingRateImageEnable` to `VK_TRUE`, then vkCmdSetViewportShadingRatePaletteNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectNV-None-08637
  If the `shadingRateImage` feature is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE`, and the most recent call to vkCmdSetShadingRateImageEnableNV in the current command buffer set `shadingRateImageEnable` to `VK_TRUE`, then the `viewportCount` parameter in the last call to vkCmdSetViewportShadingRatePaletteNV **must** be greater than or equal to the `viewportCount` parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawMeshTasksIndirectNV-VkPipelineVieportCreateInfo-04141
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT` dynamic state enabled and a VkPipelineViewportSwizzleStateCreateInfoNV structure chained from VkPipelineViewportStateCreateInfo, then the bound graphics pipeline **must** have been created with VkPipelineViewportSwizzleStateCreateInfoNV::`viewportCount` greater or equal to the `viewportCount` parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawMeshTasksIndirectNV-VkPipelineVieportCreateInfo-04142
  If the bound graphics pipeline state was created with the

VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled and a VkPipelineViewportExclusiveScissorStateCreateInfoNV structure chained from VkPipelineViewportStateCreateInfo, then the bound graphics pipeline **must** have been created with VkPipelineViewportExclusiveScissorStateCreateInfoNV::exclusiveScissorCount greater or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT

- VUID-vkCmdDrawMeshTasksIndirectNV-None-07878
  If the exclusiveScissor feature is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_EXCLUSIVE_SCISSOR_ENABLE_NV dynamic state enabled, then vkCmdSetExclusiveScissorEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectNV-None-07879
  If the exclusiveScissor feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_EXCLUSIVE_SCISSOR_NV dynamic state enabled, and the most recent call to vkCmdSetExclusiveScissorEnableNV in the current command buffer set any element of pExclusiveScissorEnables to VK_TRUE, then vkCmdSetExclusiveScissorNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectNV-None-04876
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_RASTERIZER_DISCARD_ENABLE dynamic state enabled, then vkCmdSetRasterizerDiscardEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectNV-None-04877
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_BIAS_ENABLE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetDepthBiasEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectNV-logicOp-04878
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_LOGIC_OP_EXT dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of logicOpEnable is VK_TRUE, then vkCmdSetLogicOpEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectNV-primitiveFragmentShadingRateWithMultipleViewports-04552
  If the primitiveFragmentShadingRateWithMultipleViewports limit is not supported, the bound graphics pipeline was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, and any of the shader stages of the bound graphics pipeline write to the PrimitiveShadingRateKHR built-in, then vkCmdSetViewportWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the viewportCount parameter of vkCmdSetViewportWithCountEXT **must** be 1

- VUID-vkCmdDrawMeshTasksIndirectNV-primitiveFragmentShadingRateWithMultipleViewports-08642
  If the `primitiveFragmentShadingRateWithMultipleViewports` limit is not supported, and any shader object bound to a graphics stage writes to the `PrimitiveShadingRateKHR` built-in, then vkCmdSetViewportWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the `viewportCount` parameter of `vkCmdSetViewportWithCountEXT` **must** be `1`
- VUID-vkCmdDrawMeshTasksIndirectNV-blendEnable-04727
  If rasterization is not disabled in the bound graphics pipeline, then for each color attachment in the subpass, if the corresponding image view's format features do not contain `VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT`, then the `blendEnable` member of the corresponding element of the `pAttachments` member of `pColorBlendState` **must** be `VK_FALSE`
- VUID-vkCmdDrawMeshTasksIndirectNV-None-08643
  If a shader object is bound to the `VK_SHADER_STAGE_FRAGMENT_BIT` stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE`, then for each color attachment in the render pass, if the corresponding image view's format features do not contain `VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT`, then the corresponding member of `pColorBlendEnables` in the most recent call to `vkCmdSetColorBlendEnableEXT` in the current command buffer that affected that attachment index **must** have been `VK_FALSE`
- VUID-vkCmdDrawMeshTasksIndirectNV-multisampledRenderToSingleSampled-07284
  If rasterization is not disabled in the bound graphics pipeline, and none of the following is enabled:
  - the `VK_AMD_mixed_attachment_samples` extension
  - the `VK_NV_framebuffer_mixed_samples` extension
  - the `multisampledRenderToSingleSampled` feature

  then `rasterizationSamples` for the currently bound graphics pipeline **must** be the same as the current subpass color and/or depth/stencil attachments
- VUID-vkCmdDrawMeshTasksIndirectNV-None-08644
  If a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE`, and none of the following is enabled:
  - the `VK_AMD_mixed_attachment_samples` extension
  - the `VK_NV_framebuffer_mixed_samples` extension
  - the `multisampledRenderToSingleSampled` feature

  then the most recent call to vkCmdSetRasterizationSamplesEXT in the current command buffer **must** have set `rasterizationSamples` to be the same as the number of samples for the current render pass color and/or depth/stencil attachments
- VUID-vkCmdDrawMeshTasksIndirectNV-None-08876
  If a shader object is bound to any graphics stage, the current render pass instance **must**

have been begun with vkCmdBeginRenderingKHR

- VUID-vkCmdDrawMeshTasksIndirectNV-imageView-06172
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the `imageView` member of `pDepthAttachment` is not VK_NULL_HANDLE, and the `layout` member of `pDepthAttachment` is `VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL`, this command **must** not write any values to the depth attachment
- VUID-vkCmdDrawMeshTasksIndirectNV-imageView-06173
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the `imageView` member of `pStencilAttachment` is not VK_NULL_HANDLE, and the `layout` member of `pStencilAttachment` is `VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL`, this command **must** not write any values to the stencil attachment
- VUID-vkCmdDrawMeshTasksIndirectNV-imageView-06174
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the `imageView` member of `pDepthAttachment` is not VK_NULL_HANDLE, and the `layout` member of `pDepthAttachment` is `VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_STENCIL_ATTACHMENT_OPTIMAL`, this command **must** not write any values to the depth attachment
- VUID-vkCmdDrawMeshTasksIndirectNV-imageView-06175
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the `imageView` member of `pStencilAttachment` is not VK_NULL_HANDLE, and the `layout` member of `pStencilAttachment` is `VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_STENCIL_READ_ONLY_OPTIMAL`, this command **must** not write any values to the stencil attachment
- VUID-vkCmdDrawMeshTasksIndirectNV-imageView-06176
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the `imageView` member of `pDepthAttachment` is not VK_NULL_HANDLE, and the `layout` member of `pDepthAttachment` is `VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL`, this command **must** not write any values to the depth attachment
- VUID-vkCmdDrawMeshTasksIndirectNV-imageView-06177
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the `imageView` member of `pStencilAttachment` is not VK_NULL_HANDLE, and the `layout` member of `pStencilAttachment` is `VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL`, this command **must** not write any values to the stencil attachment
- VUID-vkCmdDrawMeshTasksIndirectNV-viewMask-06178
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the currently bound graphics pipeline **must** have been created with a VkPipelineRenderingCreateInfo::`viewMask` equal to VkRenderingInfo::`viewMask`
- VUID-vkCmdDrawMeshTasksIndirectNV-colorAttachmentCount-06179
  If the `dynamicRenderingUnusedAttachments` feature is not enabled and the current render pass instance was begun with vkCmdBeginRenderingKHR, the currently bound graphics pipeline **must** have been created with a VkPipelineRenderingCreateInfo::`colorAttachmentCount` equal to VkRenderingInfo::`colorAttachmentCount`
- VUID-vkCmdDrawMeshTasksIndirectNV-dynamicRenderingUnusedAttachments-08910
  If the `dynamicRenderingUnusedAttachments` feature is not enabled, and the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingInfo

::colorAttachmentCount greater than 0, then each element of the VkRenderingInfo::pColorAttachments array with an imageView not equal to VK_NULL_HANDLE **must** have been created with a VkFormat equal to the corresponding element of VkPipelineRenderingCreateInfo::pColorAttachmentFormats used to create the currently bound graphics pipeline

- VUID-vkCmdDrawMeshTasksIndirectNV-dynamicRenderingUnusedAttachments-08912
  If the dynamicRenderingUnusedAttachments feature is not enabled, and the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingInfo::colorAttachmentCount greater than 0, then each element of the VkRenderingInfo::pColorAttachments array with an imageView equal to VK_NULL_HANDLE **must** have the corresponding element of VkPipelineRenderingCreateInfo::pColorAttachmentFormats used to create the currently bound pipeline equal to VK_FORMAT_UNDEFINED

- VUID-vkCmdDrawMeshTasksIndirectNV-dynamicRenderingUnusedAttachments-08911
  If the dynamicRenderingUnusedAttachments feature is enabled, and the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingInfo::colorAttachmentCount greater than 0, then each element of the VkRenderingInfo::pColorAttachments array with an imageView not equal to VK_NULL_HANDLE **must** have been created with a VkFormat equal to the corresponding element of VkPipelineRenderingCreateInfo::pColorAttachmentFormats used to create the currently bound graphics pipeline, or the corresponding element of VkPipelineRenderingCreateInfo::pColorAttachmentFormats, if it exists, **must** be VK_FORMAT_UNDEFINED

- VUID-vkCmdDrawMeshTasksIndirectNV-colorAttachmentCount-09362
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, with a VkRenderingInfo::colorAttachmentCount equal to 1, there is no shader object bound to any graphics stage, and a color attachment with a resolve mode of VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID, each element of the VkRenderingInfo::pColorAttachments array with a resolveImageView not equal to VK_NULL_HANDLE **must** have been created with an image created with a VkExternalFormatANDROID::externalFormat value equal to the VkExternalFormatANDROID::externalFormat value used to create the currently bound graphics pipeline

- VUID-vkCmdDrawMeshTasksIndirectNV-None-09363
  If there is no shader object bound to any graphics stage, the current render pass instance was begun with vkCmdBeginRenderingKHR and a VkRenderingInfo::colorAttachmentCount equal to 1, and a color attachment with a resolve mode of VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID, each element of the VkRenderingInfo::pColorAttachments array with a imageView not equal to VK_NULL_HANDLE **must** have been created with an image created with a VkExternalFormatANDROID::externalFormat value equal to the VkExternalFormatANDROID::externalFormat value used to create the currently bound graphics pipeline

- VUID-vkCmdDrawMeshTasksIndirectNV-None-09364
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is no shader object bound to any graphics stage, and the currently bound graphics pipeline was created with a non-zero VkExternalFormatANDROID::externalFormat value and with

the VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT dynamic state enabled, then vkCmdSetColorBlendEnableEXT **must** have set the blend enable to VK_FALSE prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectNV-None-09365
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is no shader object bound to any graphics stage, and the currently bound graphics pipeline was created with a non-zero VkExternalFormatANDROID::externalFormat value and with the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT dynamic state enabled, then vkCmdSetRasterizationSamplesEXT **must** have set rasterizationSamples to VK_SAMPLE_COUNT_1_BIT prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectNV-None-09366
  If there is a shader object bound to any graphics stage, and the current render pass includes a color attachment that uses the VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID resolve mode, then vkCmdSetColorBlendEnableEXT **must** have set blend enable to VK_FALSE prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectNV-rasterizationSamples-09367
  If there is a shader object bound to any graphics stage, and the current render pass includes a color attachment that uses the VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID resolve mode, then vkCmdSetRasterizationSamplesEXT **must** have set rasterizationSamples to VK_SAMPLE_COUNT_1_BIT prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectNV-None-09368
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is no shader object bound to any graphics stage, and the currently bound graphics pipeline was created with a non-zero VkExternalFormatANDROID::externalFormat value and with the VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR dynamic state enabled, then vkCmdSetFragmentShadingRateKHR **must** have set pFragmentSize->width to 1 prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectNV-None-09369
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is no shader object bound to any graphics stage, and the currently bound graphics pipeline was created with a non-zero VkExternalFormatANDROID::externalFormat value and with the VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR dynamic state enabled, then vkCmdSetFragmentShadingRateKHR **must** have set pFragmentSize->height to 1 prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectNV-pFragmentSize-09370
  If there is a shader object bound to any graphics stage, and the current render pass includes a color attachment that uses the VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID resolve mode, then vkCmdSetFragmentShadingRateKHR **must** have set pFragmentSize->width to 1 prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectNV-pFragmentSize-09371
  If there is a shader object bound to any graphics stage, and the current render pass includes a color attachment that uses the

VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID resolve mode, then vkCmdSetFragmentShadingRateKHR **must** have set pFragmentSize->height to 1 prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectNV-None-07749
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_WRITE_ENABLE_EXT dynamic state enabled then vkCmdSetColorWriteEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectNV-None-08646
  If the colorWriteEnable feature is enabled on the device, and a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetColorWriteEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectNV-attachmentCount-07750
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_WRITE_ENABLE_EXT dynamic state enabled then the attachmentCount parameter of vkCmdSetColorWriteEnableEXT **must** be greater than or equal to the VkPipelineColorBlendStateCreateInfo::attachmentCount of the currently bound graphics pipeline

- VUID-vkCmdDrawMeshTasksIndirectNV-None-08647
  If the colorWriteEnable feature is enabled on the device, and a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then the attachmentCount parameter of most recent call to vkCmdSetColorWriteEnableEXT in the current command buffer **must** be greater than or equal to the number of color attachments in the current render pass instance

- VUID-vkCmdDrawMeshTasksIndirectNV-None-07751
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_DISCARD_RECTANGLE_EXT dynamic state enabled then vkCmdSetDiscardRectangleEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command for each discard rectangle in VkPipelineDiscardRectangleStateCreateInfoEXT::discardRectangleCount

- VUID-vkCmdDrawMeshTasksIndirectNV-None-07880
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_DISCARD_RECTANGLE_ENABLE_EXT dynamic state enabled then vkCmdSetDiscardRectangleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectNV-rasterizerDiscardEnable-09236
  If the VK_EXT_discard_rectangles extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetDiscardRectangleEnableEXT in the current command buffer set discardRectangleEnable to VK_TRUE, then vkCmdSetDiscardRectangleEXT **must** have been

called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectNV-None-08648
  If the VK_EXT_discard_rectangles extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetDiscardRectangleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectNV-None-07881
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_DISCARD_RECTANGLE_MODE_EXT dynamic state enabled then vkCmdSetDiscardRectangleModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectNV-None-08649
  If the VK_EXT_discard_rectangles extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetDiscardRectangleEnableEXT in the current command buffer set discardRectangleEnable to VK_TRUE, then vkCmdSetDiscardRectangleModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectNV-dynamicRenderingUnusedAttachments-08913
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is not enabled, and VkRenderingInfo::pDepthAttachment->imageView was VK_NULL_HANDLE, the value of VkPipelineRenderingCreateInfo::depthAttachmentFormat used to create the currently bound graphics pipeline **must** be equal to VK_FORMAT_UNDEFINED

- VUID-vkCmdDrawMeshTasksIndirectNV-dynamicRenderingUnusedAttachments-08914
  If current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is not enabled, and VkRenderingInfo::pDepthAttachment->imageView was not VK_NULL_HANDLE, the value of VkPipelineRenderingCreateInfo::depthAttachmentFormat used to create the currently bound graphics pipeline **must** be equal to the VkFormat used to create VkRenderingInfo::pDepthAttachment->imageView

- VUID-vkCmdDrawMeshTasksIndirectNV-dynamicRenderingUnusedAttachments-08915
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is enabled, VkRenderingInfo::pDepthAttachment->imageView was not VK_NULL_HANDLE, and the value of VkPipelineRenderingCreateInfo::depthAttachmentFormat used to create the currently bound graphics pipeline was not equal to the VkFormat used to create VkRenderingInfo::pDepthAttachment->imageView, the value of the format **must** be VK_FORMAT_UNDEFINED

- VUID-vkCmdDrawMeshTasksIndirectNV-dynamicRenderingUnusedAttachments-08916
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is not enabled, and VkRenderingInfo::pStencilAttachment->imageView was VK_NULL_HANDLE, the value of

VkPipelineRenderingCreateInfo::stencilAttachmentFormat used to create the currently bound graphics pipeline **must** be equal to VK_FORMAT_UNDEFINED

- VUID-vkCmdDrawMeshTasksIndirectNV-dynamicRenderingUnusedAttachments-08917
  If current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is not enabled, and VkRenderingInfo::pStencilAttachment->imageView was not VK_NULL_HANDLE, the value of VkPipelineRenderingCreateInfo::stencilAttachmentFormat used to create the currently bound graphics pipeline **must** be equal to the VkFormat used to create VkRenderingInfo::pStencilAttachment->imageView
- VUID-vkCmdDrawMeshTasksIndirectNV-dynamicRenderingUnusedAttachments-08918
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is enabled, VkRenderingInfo::pStencilAttachment->imageView was not VK_NULL_HANDLE, and the value of VkPipelineRenderingCreateInfo::stencilAttachmentFormat used to create the currently bound graphics pipeline was not equal to the VkFormat used to create VkRenderingInfo::pStencilAttachment->imageView, the value of the format **must** be VK_FORMAT_UNDEFINED
- VUID-vkCmdDrawMeshTasksIndirectNV-imageView-06183
  If the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingFragmentShadingRateAttachmentInfoKHR::imageView was not VK_NULL_HANDLE, the currently bound graphics pipeline **must** have been created with VK_PIPELINE_CREATE_RENDERING_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR
- VUID-vkCmdDrawMeshTasksIndirectNV-imageView-06184
  If the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingFragmentDensityMapAttachmentInfoEXT::imageView was not VK_NULL_HANDLE, the currently bound graphics pipeline **must** have been created with VK_PIPELINE_CREATE_RENDERING_FRAGMENT_DENSITY_MAP_ATTACHMENT_BIT_EXT
- VUID-vkCmdDrawMeshTasksIndirectNV-colorAttachmentCount-06185
  If the currently bound pipeline was created with a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and the current render pass instance was begun with vkCmdBeginRenderingKHR with a VkRenderingInfo::colorAttachmentCount parameter greater than 0, then each element of the VkRenderingInfo::pColorAttachments array with a imageView not equal to VK_NULL_HANDLE **must** have been created with a sample count equal to the corresponding element of the pColorAttachmentSamples member of VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV used to create the currently bound graphics pipeline
- VUID-vkCmdDrawMeshTasksIndirectNV-pDepthAttachment-06186
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the currently bound pipeline was created with a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and VkRenderingInfo::pDepthAttachment->imageView was not VK_NULL_HANDLE, the value of the depthStencilAttachmentSamples member of VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV used to create the currently bound graphics pipeline **must** be equal to the sample count used to create VkRenderingInfo::pDepthAttachment->imageView
- VUID-vkCmdDrawMeshTasksIndirectNV-pStencilAttachment-06187
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the

currently bound pipeline was created with a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and VkRenderingInfo::pStencilAttachment->imageView was not VK_NULL_HANDLE, the value of the depthStencilAttachmentSamples member of VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV used to create the currently bound graphics pipeline **must** be equal to the sample count used to create VkRenderingInfo::pStencilAttachment->imageView

- VUID-vkCmdDrawMeshTasksIndirectNV-multisampledRenderToSingleSampled-07285
  If the currently bound pipeline was created without a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and the multisampledRenderToSingleSampled feature is not enabled, and the current render pass instance was begun with vkCmdBeginRenderingKHR with a VkRenderingInfo::colorAttachmentCount parameter greater than 0, then each element of the VkRenderingInfo::pColorAttachments array with a imageView not equal to VK_NULL_HANDLE **must** have been created with a sample count equal to the value of rasterizationSamples for the currently bound graphics pipeline
- VUID-vkCmdDrawMeshTasksIndirectNV-multisampledRenderToSingleSampled-07286
  If the currently bound pipeline was created without a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and the multisampledRenderToSingleSampled feature is not enabled, and VkRenderingInfo::pDepthAttachment->imageView was not VK_NULL_HANDLE, the value of rasterizationSamples for the currently bound graphics pipeline **must** be equal to the sample count used to create VkRenderingInfo::pDepthAttachment->imageView
- VUID-vkCmdDrawMeshTasksIndirectNV-multisampledRenderToSingleSampled-07287
  If the currently bound pipeline was created without a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and the multisampledRenderToSingleSampled feature is not enabled, and VkRenderingInfo::pStencilAttachment->imageView was not VK_NULL_HANDLE, the value of rasterizationSamples for the currently bound graphics pipeline **must** be equal to the sample count used to create VkRenderingInfo::pStencilAttachment->imageView
- VUID-vkCmdDrawMeshTasksIndirectNV-pNext-07935
  If this command has been called inside a render pass instance started with vkCmdBeginRenderingKHR, and the pNext chain of VkRenderingInfo includes a VkMultisampledRenderToSingleSampledInfoEXT structure with multisampledRenderToSingleSampledEnable equal to VK_TRUE, then the value of rasterizationSamples for the currently bound graphics pipeline **must** be equal to VkMultisampledRenderToSingleSampledInfoEXT::rasterizationSamples
- VUID-vkCmdDrawMeshTasksIndirectNV-renderPass-06198
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the currently bound pipeline **must** have been created with a VkGraphicsPipelineCreateInfo::renderPass equal to VK_NULL_HANDLE
- VUID-vkCmdDrawMeshTasksIndirectNV-pColorAttachments-08963
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is a graphics pipeline bound with a fragment shader that statically writes to a color attachment, the color write mask is not zero, color writes are enabled, and the corresponding element of the VkRenderingInfo::pColorAttachments->imageView was not

VK_NULL_HANDLE, then the corresponding element of VkPipelineRenderingCreateInfo::pColorAttachmentFormats used to create the pipeline **must** not be VK_FORMAT_UNDEFINED

- VUID-vkCmdDrawMeshTasksIndirectNV-pDepthAttachment-08964
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is a graphics pipeline bound, depth test is enabled, depth write is enabled, and the VkRenderingInfo::pDepthAttachment->imageView was not VK_NULL_HANDLE, then the VkPipelineRenderingCreateInfo::depthAttachmentFormat used to create the pipeline **must** not be VK_FORMAT_UNDEFINED

- VUID-vkCmdDrawMeshTasksIndirectNV-pStencilAttachment-08965
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is a graphics pipeline bound, stencil test is enabled and the VkRenderingInfo::pStencilAttachment->imageView was not VK_NULL_HANDLE, then the VkPipelineRenderingCreateInfo::stencilAttachmentFormat used to create the pipeline **must** not be VK_FORMAT_UNDEFINED

- VUID-vkCmdDrawMeshTasksIndirectNV-primitivesGeneratedQueryWithRasterizerDiscard-06708
  If the primitivesGeneratedQueryWithRasterizerDiscard feature is not enabled and the VK_QUERY_TYPE_PRIMITIVES_GENERATED_EXT query is active, rasterization discard **must** not be enabled

- VUID-vkCmdDrawMeshTasksIndirectNV-primitivesGeneratedQueryWithNonZeroStreams-06709
  If the primitivesGeneratedQueryWithNonZeroStreams feature is not enabled and the VK_QUERY_TYPE_PRIMITIVES_GENERATED_EXT query is active, the bound graphics pipeline **must** not have been created with a non-zero value in VkPipelineRasterizationStateStreamCreateInfoEXT::rasterizationStream

- VUID-vkCmdDrawMeshTasksIndirectNV-None-07619
  If a shader object is bound to the VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_TESSELLATION_DOMAIN_ORIGIN_EXT dynamic state enabled, then vkCmdSetTessellationDomainOriginEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectNV-None-07620
  If the depthClamp feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_CLAMP_ENABLE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetDepthClampEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectNV-None-07621
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_POLYGON_MODE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetPolygonModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectNV-None-07622

If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetRasterizationSamplesEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectNV-None-07623
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_SAMPLE_MASK_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetSampleMaskEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectNV-alphaToCoverageEnable-08919
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_ALPHA_TO_COVERAGE_ENABLE_EXT dynamic state enabled, and alphaToCoverageEnable was VK_TRUE in the last call to vkCmdSetAlphaToCoverageEnableEXT, then the Fragment Output Interface **must** contain a variable for the alpha Component word in Location 0 at Index 0

- VUID-vkCmdDrawMeshTasksIndirectNV-alphaToCoverageEnable-08920
  If a shader object is bound to any graphics stage, and the most recent call to vkCmdSetAlphaToCoverageEnableEXT in the current command buffer set alphaToCoverageEnable to VK_TRUE, then the Fragment Output Interface **must** contain a variable for the alpha Component word in Location 0 at Index 0

- VUID-vkCmdDrawMeshTasksIndirectNV-None-07624
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_ALPHA_TO_COVERAGE_ENABLE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetAlphaToCoverageEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectNV-None-07625
  If the alphaToOne feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_ALPHA_TO_ONE_ENABLE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetAlphaToOneEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectNV-None-07626
  If the logicOp feature is enabled, a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_LOGIC_OP_ENABLE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetLogicOpEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectNV-None-07627
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT dynamic state enabled then

vkCmdSetColorBlendEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectNV-None-08657
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and both the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE and there are color attachments bound, then vkCmdSetColorBlendEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectNV-None-07628
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_EQUATION_EXT dynamic state enabled then vkCmdSetColorBlendEquationEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectNV-None-08658
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetColorBlendEnableEXT for any attachment set that attachment's value in pColorBlendEnables to VK_TRUE, then vkCmdSetColorBlendEquationEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectNV-None-07629
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_WRITE_MASK_EXT dynamic state enabled then vkCmdSetColorWriteMaskEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectNV-None-08659
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and both the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE and there are color attachments bound, then vkCmdSetColorWriteMaskEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectNV-None-07630
  If the geometryStreams feature is enabled, and a shader object is bound to the VK_SHADER_STAGE_GEOMETRY_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_RASTERIZATION_STREAM_EXT dynamic state enabled, then vkCmdSetRasterizationStreamEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectNV-None-07631
  If the VK_EXT_conservative_rasterization extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_CONSERVATIVE_RASTERIZATION_MODE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetConservativeRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectNV-None-07632
  If the VK_EXT_conservative_rasterization extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_EXTRA_PRIMITIVE_OVERESTIMATION_SIZE_EXT dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of conservativeRasterizationMode is VK_CONSERVATIVE_RASTERIZATION_MODE_OVERESTIMATE_EXT, then vkCmdSetExtraPrimitiveOverestimationSizeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectNV-None-07633
  If the depthClipEnable feature is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_CLIP_ENABLE_EXT dynamic state, then vkCmdSetDepthClipEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectNV-None-07634
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT dynamic state enabled then vkCmdSetSampleLocationsEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectNV-None-08664
  If the VK_EXT_sample_locations extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetSampleLocationsEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectNV-None-07635
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_ADVANCED_EXT dynamic state enabled then vkCmdSetColorBlendAdvancedEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectNV-rasterizerDiscardEnable-09416
  If the VK_EXT_blend_operation_advanced extension is enabled, and a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then at least one of vkCmdSetColorBlendEquationEXT and vkCmdSetColorBlendAdvancedEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectNV-None-07636
  If the VK_EXT_provoking_vertex extension is enabled, a shader object is bound to the VK_SHADER_STAGE_VERTEX_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_PROVOKING_VERTEX_MODE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetProvokingVertexModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectNV-None-07637
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT dynamic state enabled then vkCmdSetLineRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectNV-None-08666
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetPolygonModeEXT in the current command buffer set polygonMode to VK_POLYGON_MODE_LINE, then vkCmdSetLineRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectNV-None-08667
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object is bound to the VK_SHADER_STAGE_VERTEX_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetPrimitiveTopologyEXT in the current command buffer set primitiveTopology to any line topology, then vkCmdSetLineRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectNV-None-08668
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object that outputs line primitives is bound to the VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT or VK_SHADER_STAGE_GEOMETRY_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetLineRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectNV-None-07638
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT dynamic state enabled then vkCmdSetLineStippleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectNV-None-08669
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetPolygonModeEXT in the current command buffer set polygonMode to VK_POLYGON_MODE_LINE, then vkCmdSetLineStippleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectNV-None-08670
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object is bound to the VK_SHADER_STAGE_VERTEX_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetPrimitiveTopologyEXT in the current command buffer set primitiveTopology to any line topology, then vkCmdSetLineStippleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectNV-None-08671
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object that outputs line primitives is bound to the VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT or VK_SHADER_STAGE_GEOMETRY_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetLineStippleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectNV-None-07849
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_KHR dynamic state enabled then vkCmdSetLineStippleKHR **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectNV-None-08672
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetLineStippleEnableEXT in the current command buffer set stippledLineEnable to VK_TRUE, then vkCmdSetLineStippleEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectNV-None-07639
  If the depthClipControl feature is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_CLIP_NEGATIVE_ONE_TO_ONE_EXT dynamic state enabled, then vkCmdSetDepthClipNegativeOneToOneEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectNV-None-07640
  If the VK_NV_clip_space_w_scaling extension is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_VIEWPORT_W_SCALING_ENABLE_NV dynamic state enabled, then vkCmdSetViewportWScalingEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectNV-None-07641
  If the VK_NV_viewport_swizzle extension is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_VIEWPORT_SWIZZLE_NV dynamic state enabled, then

vkCmdSetViewportSwizzleNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectNV-None-07642
  If the VK_NV_fragment_coverage_to_color extension is enabled, a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_TO_COLOR_ENABLE_NV dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetCoverageToColorEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectNV-None-07643
  If the VK_NV_fragment_coverage_to_color extension is enabled, a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_TO_COLOR_LOCATION_NV dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of coverageToColorEnable is VK_TRUE, then vkCmdSetCoverageToColorLocationNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectNV-None-07644
  If the VK_NV_framebuffer_mixed_samples extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_MODULATION_MODE_NV dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetCoverageModulationModeNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectNV-None-07645
  If the VK_NV_framebuffer_mixed_samples extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_MODULATION_TABLE_ENABLE_NV dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of coverageModulationMode is any value other than VK_COVERAGE_MODULATION_MODE_NONE_NV, then vkCmdSetCoverageModulationTableEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectNV-None-07646
  If the VK_NV_framebuffer_mixed_samples extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_MODULATION_TABLE_NV dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of coverageModulationTableEnable is VK_TRUE, then vkCmdSetCoverageModulationTableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectNV-None-07647
  If the shadingRateImage feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_SHADING_RATE_IMAGE_ENABLE_NV dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetShadingRateImageEnableNV

**must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectNV-pipelineFragmentShadingRate-09238
  If the `pipelineFragmentShadingRate` feature is enabled, a shader object is bound to the `VK_SHADER_STAGE_FRAGMENT_BIT` stage or a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR` dynamic state enabled, and the current value of `rasterizerDiscardEnable` is `VK_FALSE`, then vkCmdSetFragmentShadingRateKHR **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectNV-None-07648
  If the `representativeFragmentTest` feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_REPRESENTATIVE_FRAGMENT_TEST_ENABLE_NV` dynamic state enabled, and the current value of `rasterizerDiscardEnable` is `VK_FALSE`, then vkCmdSetRepresentativeFragmentTestEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectNV-None-07649
  If the `coverageReductionMode` feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_COVERAGE_REDUCTION_MODE_NV` dynamic state enabled, and the current value of `rasterizerDiscardEnable` is `VK_FALSE`, then vkCmdSetCoverageReductionModeNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectNV-pColorBlendEnables-07470
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT` state enabled and the last call to vkCmdSetColorBlendEnableEXT set `pColorBlendEnables` for any attachment to `VK_TRUE`, then for those attachments in the subpass the corresponding image view's format features **must** contain `VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT`

- VUID-vkCmdDrawMeshTasksIndirectNV-rasterizationSamples-07471
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT` state enabled, and the current subpass does not use any color and/or depth/stencil attachments, then the `rasterizationSamples` in the last call to vkCmdSetRasterizationSamplesEXT **must** follow the rules for a zero-attachment subpass

- VUID-vkCmdDrawMeshTasksIndirectNV-samples-07472
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_SAMPLE_MASK_EXT` state enabled and the `VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT` state disabled, then the `samples` parameter in the last call to vkCmdSetSampleMaskEXT **must** be greater or equal to the VkPipelineMultisampleStateCreateInfo::`rasterizationSamples` parameter used to create the bound graphics pipeline

- VUID-vkCmdDrawMeshTasksIndirectNV-samples-07473
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_SAMPLE_MASK_EXT` state and `VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT`

states enabled, then the samples parameter in the last call to vkCmdSetSampleMaskEXT **must** be greater or equal to the rasterizationSamples parameter in the last call to vkCmdSetRasterizationSamplesEXT

- VUID-vkCmdDrawMeshTasksIndirectNV-rasterizationSamples-07474
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, and neither the VK_AMD_mixed_attachment_samples nor the VK_NV_framebuffer_mixed_samples extensions are enabled, then the rasterizationSamples in the last call to vkCmdSetRasterizationSamplesEXT **must** be the same as the current subpass color and/or depth/stencil attachments

- VUID-vkCmdDrawMeshTasksIndirectNV-None-09211
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, or a shader object is bound to any graphics stage, and the current render pass instance includes a VkMultisampledRenderToSingleSampledInfoEXT structure with multisampledRenderToSingleSampledEnable equal to VK_TRUE, then the rasterizationSamples in the last call to vkCmdSetRasterizationSamplesEXT **must** be the same as the rasterizationSamples member of that structure

- VUID-vkCmdDrawMeshTasksIndirectNV-firstAttachment-07476
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT dynamic state enabled then vkCmdSetColorBlendEnableEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the firstAttachment and attachmentCount parameters of vkCmdSetColorBlendEnableEXT calls **must** specify an enable for all active color attachments in the current subpass

- VUID-vkCmdDrawMeshTasksIndirectNV-rasterizerDiscardEnable-09417
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetColorBlendEnableEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the firstAttachment and attachmentCount parameters of vkCmdSetColorBlendEnableEXT calls **must** specify an enable for all active color attachments in the current subpass

- VUID-vkCmdDrawMeshTasksIndirectNV-firstAttachment-07477
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_EQUATION_EXT dynamic state enabled then vkCmdSetColorBlendEquationEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the firstAttachment and attachmentCount parameters of vkCmdSetColorBlendEquationEXT calls **must** specify the blend equations for all active color attachments in the current subpass where blending is enabled

- VUID-vkCmdDrawMeshTasksIndirectNV-rasterizerDiscardEnable-09418
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and both the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE and there are color attachments bound, then

vkCmdSetColorBlendEquationEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the `firstAttachment` and `attachmentCount` parameters of `vkCmdSetColorBlendEquationEXT` calls **must** specify the blend equations for all active color attachments in the current subpass where blending is enabled

- VUID-vkCmdDrawMeshTasksIndirectNV-firstAttachment-07478
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_COLOR_WRITE_MASK_EXT` dynamic state enabled then vkCmdSetColorWriteMaskEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the `firstAttachment` and `attachmentCount` parameters of `vkCmdSetColorWriteMaskEXT` calls **must** specify the color write mask for all active color attachments in the current subpass

- VUID-vkCmdDrawMeshTasksIndirectNV-rasterizerDiscardEnable-09419
  If a shader object is bound to the `VK_SHADER_STAGE_FRAGMENT_BIT` stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE`, then vkCmdSetColorWriteMaskEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the `firstAttachment` and `attachmentCount` parameters of `vkCmdSetColorWriteMaskEXT` calls **must** specify the color write mask for all active color attachments in the current subpass

- VUID-vkCmdDrawMeshTasksIndirectNV-firstAttachment-07479
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_COLOR_BLEND_ADVANCED_EXT` dynamic state enabled then vkCmdSetColorBlendAdvancedEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the `firstAttachment` and `attachmentCount` parameters of `vkCmdSetColorBlendAdvancedEXT` calls **must** specify the advanced blend equations for all active color attachments in the current subpass where blending is enabled

- VUID-vkCmdDrawMeshTasksIndirectNV-advancedBlendMaxColorAttachments-07480
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_COLOR_BLEND_ADVANCED_EXT` and `VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT` dynamic states enabled and the last calls to vkCmdSetColorBlendEnableEXT and vkCmdSetColorBlendAdvancedEXT have enabled advanced blending, then the number of active color attachments in the current subpass **must** not exceed `advancedBlendMaxColorAttachments`

- VUID-vkCmdDrawMeshTasksIndirectNV-primitivesGeneratedQueryWithNonZeroStreams-07481
  If the `primitivesGeneratedQueryWithNonZeroStreams` feature is not enabled and the `VK_QUERY_TYPE_PRIMITIVES_GENERATED_EXT` query is active, and the bound graphics pipeline was created with `VK_DYNAMIC_STATE_RASTERIZATION_STREAM_EXT` state enabled, the last call to vkCmdSetRasterizationStreamEXT **must** have set the `rasterizationStream` to zero

- VUID-vkCmdDrawMeshTasksIndirectNV-sampleLocationsPerPixel-07482
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT` state enabled and the `VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT` state disabled, then the

sampleLocationsPerPixel member of pSampleLocationsInfo in the last call to vkCmdSetSampleLocationsEXT **must** equal the rasterizationSamples member of the VkPipelineMultisampleStateCreateInfo structure the bound graphics pipeline has been created with

- VUID-vkCmdDrawMeshTasksIndirectNV-sampleLocationsPerPixel-07483
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state enabled and the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, then the sampleLocationsPerPixel member of pSampleLocationsInfo in the last call to vkCmdSetSampleLocationsEXT **must** equal the rasterizationSamples parameter of the last call to vkCmdSetRasterizationSamplesEXT

- VUID-vkCmdDrawMeshTasksIndirectNV-sampleLocationsEnable-07484
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, or the bound graphics pipeline was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, and sampleLocationsEnable was VK_TRUE in the last call to vkCmdSetSampleLocationsEnableEXT, and the current subpass has a depth/stencil attachment, then that attachment **must** have been created with the VK_IMAGE_CREATE_SAMPLE_LOCATIONS_COMPATIBLE_DEPTH_BIT_EXT bit set

- VUID-vkCmdDrawMeshTasksIndirectNV-sampleLocationsEnable-07485
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state enabled and the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, and if sampleLocationsEnable was VK_TRUE in the last call to vkCmdSetSampleLocationsEnableEXT, then the sampleLocationsInfo.sampleLocationGridSize.width in the last call to vkCmdSetSampleLocationsEXT **must** evenly divide VkMultisamplePropertiesEXT::sampleLocationGridSize.width as returned by vkGetPhysicalDeviceMultisamplePropertiesEXT with a samples parameter equaling rasterizationSamples

- VUID-vkCmdDrawMeshTasksIndirectNV-sampleLocationsEnable-07486
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state enabled and the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, and if sampleLocationsEnable was VK_TRUE in the last call to vkCmdSetSampleLocationsEnableEXT, then the sampleLocationsInfo.sampleLocationGridSize.height in the last call to vkCmdSetSampleLocationsEXT **must** evenly divide VkMultisamplePropertiesEXT::sampleLocationGridSize.height as returned by vkGetPhysicalDeviceMultisamplePropertiesEXT with a samples parameter equaling rasterizationSamples

- VUID-vkCmdDrawMeshTasksIndirectNV-sampleLocationsEnable-07487
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, and if sampleLocationsEnable was VK_TRUE in the last call to vkCmdSetSampleLocationsEnableEXT, the fragment shader code **must** not statically use the extended instruction InterpolateAtSample

- VUID-vkCmdDrawMeshTasksIndirectNV-sampleLocationsEnable-07936
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT` state disabled and the `VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT` state enabled, the `sampleLocationsEnable` member of a VkPipelineSampleLocationsStateCreateInfoEXT::`sampleLocationsEnable` in the bound graphics pipeline is `VK_TRUE` or `VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT` state enabled, then, `sampleLocationsInfo.sampleLocationGridSize.width` **must** evenly divide VkMultisamplePropertiesEXT::`sampleLocationGridSize.width` as returned by vkGetPhysicalDeviceMultisamplePropertiesEXT with a `samples` parameter equaling the value of `rasterizationSamples` in the last call to vkCmdSetRasterizationSamplesEXT
- VUID-vkCmdDrawMeshTasksIndirectNV-sampleLocationsEnable-07937
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT` state disabled and the `VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT` state enabled, the `sampleLocationsEnable` member of a VkPipelineSampleLocationsStateCreateInfoEXT::`sampleLocationsEnable` in the bound graphics pipeline is `VK_TRUE` or `VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT` state enabled, then, `sampleLocationsInfo.sampleLocationGridSize.height` **must** evenly divide VkMultisamplePropertiesEXT::`sampleLocationGridSize.height` as returned by vkGetPhysicalDeviceMultisamplePropertiesEXT with a `samples` parameter equaling the value of `rasterizationSamples` in the last call to vkCmdSetRasterizationSamplesEXT
- VUID-vkCmdDrawMeshTasksIndirectNV-sampleLocationsEnable-07938
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT` state disabled and the `VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT` state enabled, the `sampleLocationsEnable` member of a VkPipelineSampleLocationsStateCreateInfoEXT::`sampleLocationsEnable` in the bound graphics pipeline is `VK_TRUE` or `VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT` state enabled, then, `sampleLocationsInfo.sampleLocationsPerPixel` **must** equal `rasterizationSamples` in the last call to vkCmdSetRasterizationSamplesEXT
- VUID-vkCmdDrawMeshTasksIndirectNV-coverageModulationTableEnable-07488
  If a shader object is bound to any graphics stage or the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_COVERAGE_MODULATION_TABLE_ENABLE_NV` state enabled, and the last call to vkCmdSetCoverageModulationTableEnableNV set `coverageModulationTableEnable` to `VK_TRUE`, then the `coverageModulationTableCount` parameter in the last call to vkCmdSetCoverageModulationTableNV **must** equal the current `rasterizationSamples` divided by the number of color samples in the current subpass
- VUID-vkCmdDrawMeshTasksIndirectNV-rasterizationSamples-07489
  If the `VK_NV_framebuffer_mixed_samples` extension is enabled, and if current subpass has a depth/stencil attachment and depth test, stencil test, or depth bounds test are enabled in the currently bound pipeline state, then the current `rasterizationSamples` **must** be the same as the sample count of the depth/stencil attachment
- VUID-vkCmdDrawMeshTasksIndirectNV-coverageToColorEnable-07490
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_COVERAGE_TO_COLOR_ENABLE_NV` state enabled and the last call to vkCmdSetCoverageToColorEnableNV set the `coverageToColorEnable` to `VK_TRUE`, then the current subpass **must** have a color attachment at the location selected by the last call to

vkCmdSetCoverageToColorLocationNV coverageToColorLocation, with a VkFormat of VK_FORMAT_R8_UINT, VK_FORMAT_R8_SINT, VK_FORMAT_R16_UINT, VK_FORMAT_R16_SINT, VK_FORMAT_R32_UINT, or VK_FORMAT_R32_SINT

- VUID-vkCmdDrawMeshTasksIndirectNV-rasterizerDiscardEnable-09420
  If the VK_NV_fragment_coverage_to_color extension is enabled, and a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the last call to vkCmdSetCoverageToColorEnableNV set the coverageToColorEnable to VK_TRUE, then the current subpass **must** have a color attachment at the location selected by the last call to vkCmdSetCoverageToColorLocationNV coverageToColorLocation, with a VkFormat of VK_FORMAT_R8_UINT, VK_FORMAT_R8_SINT, VK_FORMAT_R16_UINT, VK_FORMAT_R16_SINT, VK_FORMAT_R32_UINT, or VK_FORMAT_R32_SINT
- VUID-vkCmdDrawMeshTasksIndirectNV-coverageReductionMode-07491
  If this VK_NV_coverage_reduction_mode extension is enabled, the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COVERAGE_TO_COLOR_ENABLE_NV and VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT states enabled, the current coverage reduction mode coverageReductionMode, then the current rasterizationSamples, and the sample counts for the color and depth/stencil attachments (if the subpass has them) **must** be a valid combination returned by vkGetPhysicalDeviceSupportedFramebufferMixedSamplesCombinationsNV
- VUID-vkCmdDrawMeshTasksIndirectNV-viewportCount-07492
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, but not the VK_DYNAMIC_STATE_VIEWPORT_SWIZZLE_NV dynamic state enabled, then the bound graphics pipeline **must** have been created with VkPipelineViewportSwizzleStateCreateInfoNV ::viewportCount greater or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawMeshTasksIndirectNV-viewportCount-07493
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT and VK_DYNAMIC_STATE_VIEWPORT_SWIZZLE_NV dynamic states enabled then the viewportCount parameter in the last call to vkCmdSetViewportSwizzleNV **must** be greater than or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawMeshTasksIndirectNV-viewportCount-09421
  If the VK_NV_viewport_swizzle extension is enabled, and a shader object is bound to any graphics stage, then the viewportCount parameter in the last call to vkCmdSetViewportSwizzleNV **must** be greater than or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawMeshTasksIndirectNV-rasterizationSamples-07494
  If the VK_NV_framebuffer_mixed_samples extension is enabled, and if the current subpass has any color attachments and rasterizationSamples of the last call to vkCmdSetRasterizationSamplesEXT is greater than the number of color samples, then the pipeline sampleShadingEnable **must** be VK_FALSE
- VUID-vkCmdDrawMeshTasksIndirectNV-stippledLineEnable-07495

If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT or VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT dynamic states enabled, and if the current stippledLineEnable state is VK_TRUE and the current lineRasterizationMode state is VK_LINE_RASTERIZATION_MODE_RECTANGULAR_KHR, then the stippledRectangularLines feature **must** be enabled

- VUID-vkCmdDrawMeshTasksIndirectNV-stippledLineEnable-07496
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT or VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT dynamic states enabled, and if the current stippledLineEnable state is VK_TRUE and the current lineRasterizationMode state is VK_LINE_RASTERIZATION_MODE_BRESENHAM_KHR, then the stippledBresenhamLines feature **must** be enabled

- VUID-vkCmdDrawMeshTasksIndirectNV-stippledLineEnable-07497
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT or VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT dynamic states enabled, and if the current stippledLineEnable state is VK_TRUE and the current lineRasterizationMode state is VK_LINE_RASTERIZATION_MODE_RECTANGULAR_SMOOTH_KHR, then the stippledSmoothLines feature **must** be enabled

- VUID-vkCmdDrawMeshTasksIndirectNV-stippledLineEnable-07498
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT or VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT dynamic states enabled, and if the current stippledLineEnable state is VK_TRUE and the current lineRasterizationMode state is VK_LINE_RASTERIZATION_MODE_DEFAULT_KHR, then the stippledRectangularLines feature **must** be enabled and VkPhysicalDeviceLimits::strictLines **must** be VK_TRUE

- VUID-vkCmdDrawMeshTasksIndirectNV-conservativePointAndLineRasterization-07499
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_CONSERVATIVE_RASTERIZATION_MODE_EXT dynamic state enabled, conservativePointAndLineRasterization is not supported, and the effective primitive topology output by the last pre-rasterization shader stage is a line or point, then the conservativeRasterizationMode set by the last call to vkCmdSetConservativeRasterizationModeEXT **must** be VK_CONSERVATIVE_RASTERIZATION_MODE_DISABLED_EXT

- VUID-vkCmdDrawMeshTasksIndirectNV-stage-07073
  If the currently bound pipeline was created with the VkPipelineShaderStageCreateInfo::stage member of an element of VkGraphicsPipelineCreateInfo::pStages set to VK_SHADER_STAGE_VERTEX_BIT, VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT, VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT or VK_SHADER_STAGE_GEOMETRY_BIT, then Mesh Shader Queries **must** not be active

- VUID-vkCmdDrawMeshTasksIndirectNV-None-08877
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_ATTACHMENT_FEEDBACK_LOOP_ENABLE_EXT dynamic state enabled, and the

current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetAttachmentFeedbackLoopEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectNV-None-07850
  If dynamic state was inherited from VkCommandBufferInheritanceViewportScissorInfoNV, it **must** be set in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectNV-None-08684
  If there is no bound graphics pipeline, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_VERTEX_BIT
- VUID-vkCmdDrawMeshTasksIndirectNV-None-08685
  If there is no bound graphics pipeline, and the tessellationShader feature is enabled, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT
- VUID-vkCmdDrawMeshTasksIndirectNV-None-08686
  If there is no bound graphics pipeline, and the tessellationShader feature is enabled, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT
- VUID-vkCmdDrawMeshTasksIndirectNV-None-08687
  If there is no bound graphics pipeline, and the geometryShader feature is enabled, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_GEOMETRY_BIT
- VUID-vkCmdDrawMeshTasksIndirectNV-None-08688
  If there is no bound graphics pipeline, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_FRAGMENT_BIT
- VUID-vkCmdDrawMeshTasksIndirectNV-None-08689
  If there is no bound graphics pipeline, and the taskShader feature is enabled, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_TASK_BIT_EXT
- VUID-vkCmdDrawMeshTasksIndirectNV-None-08690
  If there is no bound graphics pipeline, and the meshShader feature is enabled, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_MESH_BIT_EXT
- VUID-vkCmdDrawMeshTasksIndirectNV-None-08693
  If there is no bound graphics pipeline, and at least one of the taskShader and meshShader features is enabled, one of the VK_SHADER_STAGE_VERTEX_BIT or VK_SHADER_STAGE_MESH_BIT_EXT stages **must** have a valid VkShaderEXT bound, and the other **must** have no VkShaderEXT bound
- VUID-vkCmdDrawMeshTasksIndirectNV-None-08694
  If there is no bound graphics pipeline, and both the taskShader and meshShader features are enabled, and a valid VkShaderEXT is bound the to the VK_SHADER_STAGE_MESH_BIT_EXT stage, and that VkShaderEXT was created without the VK_SHADER_CREATE_NO_TASK_SHADER_BIT_EXT flag, a valid VkShaderEXT **must** be bound to the VK_SHADER_STAGE_TASK_BIT_EXT stage

- VUID-vkCmdDrawMeshTasksIndirectNV-None-08695
  If there is no bound graphics pipeline, and both the taskShader and meshShader features are enabled, and a valid VkShaderEXT is bound the to the VK_SHADER_STAGE_MESH_BIT_EXT stage, and that VkShaderEXT was created with the VK_SHADER_CREATE_NO_TASK_SHADER_BIT_EXT flag, there **must** be no VkShaderEXT bound to the VK_SHADER_STAGE_TASK_BIT_EXT stage
- VUID-vkCmdDrawMeshTasksIndirectNV-None-08696
  If there is no bound graphics pipeline, and a valid VkShaderEXT is bound to the VK_SHADER_STAGE_VERTEX_BIT stage, there **must** be no VkShaderEXT bound to either the VK_SHADER_STAGE_TASK_BIT_EXT stage or the VK_SHADER_STAGE_MESH_BIT_EXT stage
- VUID-vkCmdDrawMeshTasksIndirectNV-None-08698
  If any graphics shader is bound which was created with the VK_SHADER_CREATE_LINK_STAGE_BIT_EXT flag, then all shaders created with the VK_SHADER_CREATE_LINK_STAGE_BIT_EXT flag in the same vkCreateShadersEXT call **must** also be bound
- VUID-vkCmdDrawMeshTasksIndirectNV-None-08699
  If any graphics shader is bound which was created with the VK_SHADER_CREATE_LINK_STAGE_BIT_EXT flag, any stages in between stages whose shaders which did not create a shader with the VK_SHADER_CREATE_LINK_STAGE_BIT_EXT flag as part of the same vkCreateShadersEXT call **must** not have any VkShaderEXT bound
- VUID-vkCmdDrawMeshTasksIndirectNV-None-08878
  All bound graphics shader objects **must** have been created with identical or identically defined push constant ranges
- VUID-vkCmdDrawMeshTasksIndirectNV-None-08879
  All bound graphics shader objects **must** have been created with identical or identically defined arrays of descriptor set layouts
- VUID-vkCmdDrawMeshTasksIndirectNV-colorAttachmentCount-09372
  If the current render pass instance was begun with vkCmdBeginRenderingKHR and a VkRenderingInfo::colorAttachmentCount equal to 1, a color attachment with a resolve mode of VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID, and a fragment shader is bound, it **must** not declare the DepthReplacing or StencilRefReplacingEXT execution modes
- VUID-vkCmdDrawMeshTasksIndirectNV-pDynamicStates-08715
  If the bound graphics pipeline state includes a fragment shader stage, was created with VK_DYNAMIC_STATE_DEPTH_WRITE_ENABLE set in VkPipelineDynamicStateCreateInfo::pDynamicStates, and the fragment shader declares the EarlyFragmentTests execution mode and uses OpDepthAttachmentReadEXT, the depthWriteEnable parameter in the last call to vkCmdSetDepthWriteEnableEXT **must** be VK_FALSE
- VUID-vkCmdDrawMeshTasksIndirectNV-pDynamicStates-08716
  If the bound graphics pipeline state includes a fragment shader stage, was created with VK_DYNAMIC_STATE_STENCIL_WRITE_MASK set in VkPipelineDynamicStateCreateInfo::pDynamicStates, and the fragment shader declares the EarlyFragmentTests execution mode and uses OpStencilAttachmentReadEXT, the writeMask parameter in the last call to vkCmdSetStencilWriteMask **must** be 0
- VUID-vkCmdDrawMeshTasksIndirectNV-None-09116
  If a shader object is bound to any graphics stage or the currently bound graphics pipeline

was created with VK_DYNAMIC_STATE_COLOR_WRITE_MASK_EXT, and the format of any color attachment is VK_FORMAT_E5B9G9R9_UFLOAT_PACK32, the corresponding element of the pColorWriteMasks parameter of vkCmdSetColorWriteMaskEXT **must** either include all of VK_COLOR_COMPONENT_R_BIT, VK_COLOR_COMPONENT_G_BIT, and VK_COLOR_COMPONENT_B_BIT, or none of them

- VUID-vkCmdDrawMeshTasksIndirectNV-maxFragmentDualSrcAttachments-09239
  If blending is enabled for any attachment where either the source or destination blend factors for that attachment use the secondary color input, the maximum value of Location for any output attachment statically used in the Fragment Execution Model executed by this command **must** be less than maxFragmentDualSrcAttachments
- VUID-vkCmdDrawMeshTasksIndirectNV-None-09548
  If the current render pass was begun with vkCmdBeginRenderingKHR, and there is no shader object bound to any graphics stage, the value of each element of VkRenderingAttachmentLocationInfoKHR::pColorAttachmentLocations set by vkCmdSetRenderingAttachmentLocationsKHR **must** match the value set for the corresponding element in the currently bound pipeline
- VUID-vkCmdDrawMeshTasksIndirectNV-None-09549
  If the current render pass was begun with vkCmdBeginRenderingKHR, and there is no shader object bound to any graphics stage, input attachment index mappings in the currently bound pipeline **must** match those set for the current render pass instance via VkRenderingInputAttachmentIndexInfoKHR
- VUID-vkCmdDrawMeshTasksIndirectNV-None-09642
  If the current render pass was begun with vkCmdBeginRenderingKHR with the VK_RENDERING_ENABLE_LEGACY_DITHERING_BIT_EXT flag, the bound graphics pipeline **must** have been created with VK_PIPELINE_CREATE_2_ENABLE_LEGACY_DITHERING_BIT_EXT
- VUID-vkCmdDrawMeshTasksIndirectNV-None-09643
  If the bound graphics pipeline was created with VK_PIPELINE_CREATE_2_ENABLE_LEGACY_DITHERING_BIT_EXT, the current render pass **must** have begun with vkCmdBeginRenderingKHR with the VK_RENDERING_ENABLE_LEGACY_DITHERING_BIT_EXT flag
- VUID-vkCmdDrawMeshTasksIndirectNV-stage-06480
  The bound graphics pipeline **must** not have been created with the VkPipelineShaderStageCreateInfo::stage member of an element of VkGraphicsPipelineCreateInfo::pStages set to VK_SHADER_STAGE_VERTEX_BIT, VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT, VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT or VK_SHADER_STAGE_GEOMETRY_BIT
- VUID-vkCmdDrawMeshTasksIndirectNV-None-07074
  Transform Feedback Queries **must** not be active
- VUID-vkCmdDrawMeshTasksIndirectNV-None-07075
  Primitives Generated Queries **must** not be active
- VUID-vkCmdDrawMeshTasksIndirectNV-pipelineStatistics-07076
  The pipelineStatistics member used to create any active Pipeline Statistics Query **must** not contain VK_QUERY_PIPELINE_STATISTIC_INPUT_ASSEMBLY_VERTICES_BIT, VK_QUERY_PIPELINE_STATISTIC_INPUT_ASSEMBLY_PRIMITIVES_BIT,

VK_QUERY_PIPELINE_STATISTIC_VERTEX_SHADER_INVOCATIONS_BIT, VK_QUERY_PIPELINE_STATISTIC_GEOMETRY_SHADER_INVOCATIONS_BIT, VK_QUERY_PIPELINE_STATISTIC_GEOMETRY_SHADER_PRIMITIVES_BIT, VK_QUERY_PIPELINE_STATISTIC_CLIPPING_INVOCATIONS_BIT, VK_QUERY_PIPELINE_STATISTIC_CLIPPING_PRIMITIVES_BIT, VK_QUERY_PIPELINE_STATISTIC_TESSELLATION_CONTROL_SHADER_PATCHES_BIT, or VK_QUERY_PIPELINE_STATISTIC_TESSELLATION_EVALUATION_SHADER_INVOCATIONS_BIT

- VUID-vkCmdDrawMeshTasksIndirectNV-buffer-02708
  If `buffer` is non-sparse then it **must** be bound completely and contiguously to a single `VkDeviceMemory` object
- VUID-vkCmdDrawMeshTasksIndirectNV-buffer-02709
  `buffer` **must** have been created with the `VK_BUFFER_USAGE_INDIRECT_BUFFER_BIT` bit set
- VUID-vkCmdDrawMeshTasksIndirectNV-offset-02710
  `offset` **must** be a multiple of `4`
- VUID-vkCmdDrawMeshTasksIndirectNV-commandBuffer-02711
  `commandBuffer` **must** not be a protected command buffer
- VUID-vkCmdDrawMeshTasksIndirectNV-drawCount-02718
  If the `multiDrawIndirect` feature is not enabled, `drawCount` **must** be `0` or `1`
- VUID-vkCmdDrawMeshTasksIndirectNV-drawCount-02719
  `drawCount` **must** be less than or equal to `VkPhysicalDeviceLimits::maxDrawIndirectCount`
- VUID-vkCmdDrawMeshTasksIndirectNV-drawCount-02146
  If `drawCount` is greater than `1`, `stride` **must** be a multiple of `4` and **must** be greater than or equal to `sizeof(VkDrawMeshTasksIndirectCommandNV)`
- VUID-vkCmdDrawMeshTasksIndirectNV-drawCount-02156
  If `drawCount` is equal to `1`, (`offset` + `sizeof`(VkDrawMeshTasksIndirectCommandNV)) **must** be less than or equal to the size of `buffer`
- VUID-vkCmdDrawMeshTasksIndirectNV-drawCount-02157
  If `drawCount` is greater than `1`, (`stride` × (`drawCount` - 1) + `offset` + `sizeof`(VkDrawMeshTasksIndirectCommandNV)) **must** be less than or equal to the size of `buffer`
- VUID-vkCmdDrawMeshTasksIndirectNV-MeshNV-07081
  The current pipeline bound to `VK_PIPELINE_BIND_POINT_GRAPHICS` **must** contain a shader stage using the `MeshNV Execution Model`

## Valid Usage (Implicit)

- VUID-vkCmdDrawMeshTasksIndirectNV-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdDrawMeshTasksIndirectNV-buffer-parameter
  `buffer` **must** be a valid VkBuffer handle
- VUID-vkCmdDrawMeshTasksIndirectNV-commandBuffer-recording
  `commandBuffer` **must** be in the recording state

- VUID-vkCmdDrawMeshTasksIndirectNV-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- VUID-vkCmdDrawMeshTasksIndirectNV-renderpass
  This command **must** only be called inside of a render pass instance
- VUID-vkCmdDrawMeshTasksIndirectNV-videocoding
  This command **must** only be called outside of a video coding scope
- VUID-vkCmdDrawMeshTasksIndirectNV-commonparent
  Both of `buffer`, and `commandBuffer` **must** have been created, allocated, or retrieved from the same VkDevice

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Inside | Outside | Graphics | Action |

The `VkDrawMeshTasksIndirectCommandNV` structure is defined as:

```
// Provided by VK_NV_mesh_shader
typedef struct VkDrawMeshTasksIndirectCommandNV {
    uint32_t    taskCount;
    uint32_t    firstTask;
} VkDrawMeshTasksIndirectCommandNV;
```

- `taskCount` is the number of local workgroups to dispatch in the X dimension. Y and Z dimension are implicitly set to one.
- `firstTask` is the X component of the first workgroup ID.

The members of `VkDrawMeshTasksIndirectCommandNV` have the same meaning as the similarly named parameters of vkCmdDrawMeshTasksNV.

## Valid Usage

- VUID-VkDrawMeshTasksIndirectCommandNV-taskCount-02175

taskCount **must** be less than or equal to VkPhysicalDeviceMeshShaderPropertiesNV::maxDrawMeshTasksCount

To record an indirect mesh tasks drawing command with the draw count sourced from a buffer, call:

```
// Provided by VK_NV_mesh_shader
void vkCmdDrawMeshTasksIndirectCountNV(
    VkCommandBuffer                             commandBuffer,
    VkBuffer                                    buffer,
    VkDeviceSize                                offset,
    VkBuffer                                    countBuffer,
    VkDeviceSize                                countBufferOffset,
    uint32_t                                    maxDrawCount,
    uint32_t                                    stride);
```

- commandBuffer is the command buffer into which the command is recorded.
- buffer is the buffer containing draw parameters.
- offset is the byte offset into buffer where parameters begin.
- countBuffer is the buffer containing the draw count.
- countBufferOffset is the byte offset into countBuffer where the draw count begins.
- maxDrawCount specifies the maximum number of draws that will be executed. The actual number of executed draw calls is the minimum of the count specified in countBuffer and maxDrawCount.
- stride is the byte stride between successive sets of draw parameters.

vkCmdDrawMeshTasksIndirectCountNV behaves similarly to vkCmdDrawMeshTasksIndirectNV except that the draw count is read by the device from a buffer during execution. The command will read an unsigned 32-bit integer from countBuffer located at countBufferOffset and use this as the draw count.

### Valid Usage

- VUID-vkCmdDrawMeshTasksIndirectCountNV-magFilter-04553
  If a VkSampler created with magFilter or minFilter equal to VK_FILTER_LINEAR, reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE, and compareEnable equal to VK_FALSE is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT
- VUID-vkCmdDrawMeshTasksIndirectCountNV-magFilter-09598
  If a VkSampler created with magFilter or minFilter equal to VK_FILTER_LINEAR and reductionMode equal to either VK_SAMPLER_REDUCTION_MODE_MIN or VK_SAMPLER_REDUCTION_MODE_MAX is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT

- VUID-vkCmdDrawMeshTasksIndirectCountNV-mipmapMode-04770
  If a VkSampler created with mipmapMode equal to VK_SAMPLER_MIPMAP_MODE_LINEAR, reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE, and compareEnable equal to VK_FALSE is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT
- VUID-vkCmdDrawMeshTasksIndirectCountNV-mipmapMode-09599
  If a VkSampler created with mipmapMode equal to VK_SAMPLER_MIPMAP_MODE_LINEAR and reductionMode equal to either VK_SAMPLER_REDUCTION_MODE_MIN or VK_SAMPLER_REDUCTION_MODE_MAX is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT
- VUID-vkCmdDrawMeshTasksIndirectCountNV-unnormalizedCoordinates-09635
  If a VkSampler created with unnormalizedCoordinates equal to VK_TRUE is used to sample a VkImageView as a result of this command, then the image view's levelCount and layerCount **must** be 1
- VUID-vkCmdDrawMeshTasksIndirectCountNV-unnormalizedCoordinates-09636
  If a VkSampler created with unnormalizedCoordinates equal to VK_TRUE is used to sample a VkImageView as a result of this command, then the image view's viewType **must** be VK_IMAGE_VIEW_TYPE_1D or VK_IMAGE_VIEW_TYPE_2D
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-06479
  If a VkImageView is sampled with depth comparison, the image view's format features **must** contain VK_FORMAT_FEATURE_2_SAMPLED_IMAGE_DEPTH_COMPARISON_BIT
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-02691
  If a VkImageView is accessed using atomic operations as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_STORAGE_IMAGE_ATOMIC_BIT
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-07888
  If a VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER descriptor is accessed using atomic operations as a result of this command, then the storage texel buffer's format features **must** contain VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_ATOMIC_BIT
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-02692
  If a VkImageView is sampled with VK_FILTER_CUBIC_EXT as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_CUBIC_BIT_EXT
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-02693
  If the VK_EXT_filter_cubic extension is not enabled and any VkImageView is sampled with VK_FILTER_CUBIC_EXT as a result of this command, it **must** not have a VkImageViewType of VK_IMAGE_VIEW_TYPE_3D, VK_IMAGE_VIEW_TYPE_CUBE, or VK_IMAGE_VIEW_TYPE_CUBE_ARRAY
- VUID-vkCmdDrawMeshTasksIndirectCountNV-filterCubic-02694
  Any VkImageView being sampled with VK_FILTER_CUBIC_EXT as a result of this command **must** have a VkImageViewType and format that supports cubic filtering, as specified by VkFilterCubicImageViewImageFormatPropertiesEXT::filterCubic returned by vkGetPhysicalDeviceImageFormatProperties2

- VUID-vkCmdDrawMeshTasksIndirectCountNV-filterCubicMinmax-02695
  Any VkImageView being sampled with VK_FILTER_CUBIC_EXT with a reduction mode of either VK_SAMPLER_REDUCTION_MODE_MIN or VK_SAMPLER_REDUCTION_MODE_MAX as a result of this command **must** have a VkImageViewType and format that supports cubic filtering together with minmax filtering, as specified by VkFilterCubicImageViewImageFormatPropertiesEXT::filterCubicMinmax returned by vkGetPhysicalDeviceImageFormatProperties2
- VUID-vkCmdDrawMeshTasksIndirectCountNV-cubicRangeClamp-09212
  If the cubicRangeClamp feature is not enabled, then any VkImageView being sampled with VK_FILTER_CUBIC_EXT as a result of this command **must** not have a VkSamplerReductionModeCreateInfo::reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE_RANGECLAMP_QCOM
- VUID-vkCmdDrawMeshTasksIndirectCountNV-reductionMode-09213
  Any VkImageView being sampled with a VkSamplerReductionModeCreateInfo::reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE_RANGECLAMP_QCOM as a result of this command **must** sample with VK_FILTER_CUBIC_EXT
- VUID-vkCmdDrawMeshTasksIndirectCountNV-selectableCubicWeights-09214
  If the selectableCubicWeights feature is not enabled, then any VkImageView being sampled with VK_FILTER_CUBIC_EXT as a result of this command **must** have VkSamplerCubicWeightsCreateInfoQCOM::cubicWeights equal to VK_CUBIC_FILTER_WEIGHTS_CATMULL_ROM_QCOM
- VUID-vkCmdDrawMeshTasksIndirectCountNV-flags-02696
  Any VkImage created with a VkImageCreateInfo::flags containing VK_IMAGE_CREATE_CORNER_SAMPLED_BIT_NV sampled as a result of this command **must** only be sampled using a VkSamplerAddressMode of VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_EDGE
- VUID-vkCmdDrawMeshTasksIndirectCountNV-OpTypeImage-07027
  For any VkImageView being written as a storage image where the image format field of the OpTypeImage is Unknown, the view's format features **must** contain VK_FORMAT_FEATURE_2_STORAGE_WRITE_WITHOUT_FORMAT_BIT
- VUID-vkCmdDrawMeshTasksIndirectCountNV-OpTypeImage-07028
  For any VkImageView being read as a storage image where the image format field of the OpTypeImage is Unknown, the view's format features **must** contain VK_FORMAT_FEATURE_2_STORAGE_READ_WITHOUT_FORMAT_BIT
- VUID-vkCmdDrawMeshTasksIndirectCountNV-OpTypeImage-07029
  For any VkBufferView being written as a storage texel buffer where the image format field of the OpTypeImage is Unknown, the view's buffer features **must** contain VK_FORMAT_FEATURE_2_STORAGE_WRITE_WITHOUT_FORMAT_BIT
- VUID-vkCmdDrawMeshTasksIndirectCountNV-OpTypeImage-07030
  Any VkBufferView being read as a storage texel buffer where the image format field of the OpTypeImage is Unknown then the view's buffer features **must** contain VK_FORMAT_FEATURE_2_STORAGE_READ_WITHOUT_FORMAT_BIT
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-08600
  For each set *n* that is statically used by a bound shader, a descriptor set **must** have been bound to *n* at the same pipeline bind point, with a VkPipelineLayout that is compatible for

set *n*, with the VkPipelineLayout used to create the current VkPipeline or the VkDescriptorSetLayout array used to create the current VkShaderEXT , as described in Pipeline Layout Compatibility

- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-08601
  For each push constant that is statically used by a bound shader, a push constant value **must** have been set for the same pipeline bind point, with a VkPipelineLayout that is compatible for push constants, with the VkPipelineLayout used to create the current VkPipeline or the VkDescriptorSetLayout array used to create the current VkShaderEXT , as described in Pipeline Layout Compatibility

- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-10068
  For each array of resources that is used by a bound shader, the indices used to access members of the array **must** be less than the descriptor count for the identified binding in the descriptor sets used by this command

- VUID-vkCmdDrawMeshTasksIndirectCountNV-maintenance4-08602
  If the `maintenance4` feature is not enabled, then for each push constant that is statically used by a bound shader, a push constant value **must** have been set for the same pipeline bind point, with a VkPipelineLayout that is compatible for push constants, with the VkPipelineLayout used to create the current VkPipeline or the VkDescriptorSetLayout and VkPushConstantRange arrays used to create the current VkShaderEXT , as described in Pipeline Layout Compatibility

- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-08114
  Descriptors in each bound descriptor set, specified via vkCmdBindDescriptorSets, **must** be valid as described by descriptor validity if they are statically used by the VkPipeline bound to the pipeline bind point used by this command and the bound VkPipeline was not created with `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`

- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-08115
  If the descriptors used by the VkPipeline bound to the pipeline bind point were specified via vkCmdBindDescriptorSets, the bound VkPipeline **must** have been created without `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`

- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-08116
  Descriptors in bound descriptor buffers, specified via vkCmdSetDescriptorBufferOffsetsEXT, **must** be valid if they are dynamically used by the VkPipeline bound to the pipeline bind point used by this command and the bound VkPipeline was created with `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`

- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-08604
  Descriptors in bound descriptor buffers, specified via vkCmdSetDescriptorBufferOffsetsEXT, **must** be valid if they are dynamically used by any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command

- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-08117
  If the descriptors used by the VkPipeline bound to the pipeline bind point were specified via vkCmdSetDescriptorBufferOffsetsEXT, the bound VkPipeline **must** have been created with `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`

- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-08119

If a descriptor is dynamically used with a VkPipeline created with VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT, the descriptor memory **must** be resident

- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-08605
  If a descriptor is dynamically used with a VkShaderEXT created with a VkDescriptorSetLayout that was created with VK_DESCRIPTOR_SET_LAYOUT_CREATE_DESCRIPTOR_BUFFER_BIT_EXT, the descriptor memory **must** be resident

- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-08606
  If the shaderObject feature is not enabled, a valid pipeline **must** be bound to the pipeline bind point used by this command

- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-08608
  If a pipeline is bound to the pipeline bind point used by this command, there **must** not have been any calls to dynamic state setting commands for any state specified statically in the VkPipeline object bound to the pipeline bind point used by this command, since that pipeline was bound

- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-08609
  If the VkPipeline object bound to the pipeline bind point used by this command or any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a VkSampler object that uses unnormalized coordinates, that sampler **must** not be used to sample from any VkImage with a VkImageView of the type VK_IMAGE_VIEW_TYPE_3D, VK_IMAGE_VIEW_TYPE_CUBE, VK_IMAGE_VIEW_TYPE_1D_ARRAY, VK_IMAGE_VIEW_TYPE_2D_ARRAY or VK_IMAGE_VIEW_TYPE_CUBE_ARRAY, in any shader stage

- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-08610
  If the VkPipeline object bound to the pipeline bind point used by this command or any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a VkSampler object that uses unnormalized coordinates, that sampler **must** not be used with any of the SPIR-V OpImageSample* or OpImageSparseSample* instructions with ImplicitLod, Dref or Proj in their name, in any shader stage

- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-08611
  If the VkPipeline object bound to the pipeline bind point used by this command or any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a VkSampler object that uses unnormalized coordinates, that sampler **must** not be used with any of the SPIR-V OpImageSample* or OpImageSparseSample* instructions that includes a LOD bias or any offset values, in any shader stage

- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-08607
  If the shaderObject is enabled, either a valid pipeline **must** be bound to the pipeline bind point used by this command, or a valid combination of valid and VK_NULL_HANDLE shader objects **must** be bound to every supported shader stage corresponding to the pipeline bind point used by this command

- VUID-vkCmdDrawMeshTasksIndirectCountNV-uniformBuffers-06935
  If any stage of the VkPipeline object bound to the pipeline bind point used by this command accesses a uniform buffer, and that stage was created without enabling either VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_EXT or VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_2_EXT for uniformBuffers, and the robustBufferAccess feature is not enabled, that stage **must** not access values

outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point

- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-08612
  If the `robustBufferAccess` feature is not enabled, and any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a uniform buffer, it **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point
- VUID-vkCmdDrawMeshTasksIndirectCountNV-storageBuffers-06936
  If any stage of the VkPipeline object bound to the pipeline bind point used by this command accesses a storage buffer, and that stage was created without enabling either `VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_EXT` or `VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_2_EXT` for `storageBuffers`, and the `robustBufferAccess` feature is not enabled, that stage **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-08613
  If the `robustBufferAccess` feature is not enabled, and any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a storage buffer, it **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point
- VUID-vkCmdDrawMeshTasksIndirectCountNV-commandBuffer-02707
  If `commandBuffer` is an unprotected command buffer and `protectedNoFault` is not supported, any resource accessed by bound shaders **must** not be a protected resource
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-06550
  If a bound shader accesses a VkSampler or VkImageView object that enables sampler $Y'C_BC_R$ conversion, that object **must** only be used with `OpImageSample*` or `OpImageSparseSample*` instructions
- VUID-vkCmdDrawMeshTasksIndirectCountNV-ConstOffset-06551
  If a bound shader accesses a VkSampler or VkImageView object that enables sampler $Y'C_BC_R$ conversion, that object **must** not use the `ConstOffset` and `Offset` operands
- VUID-vkCmdDrawMeshTasksIndirectCountNV-viewType-07752
  If a VkImageView is accessed as a result of this command, then the image view's `viewType` **must** match the `Dim` operand of the `OpTypeImage` as described in Compatibility Between SPIR-V Image Dimensions and Vulkan ImageView Types
- VUID-vkCmdDrawMeshTasksIndirectCountNV-format-07753
  If a VkImageView is accessed as a result of this command, then the numeric type of the image view's `format` and the `Sampled Type` operand of the `OpTypeImage` **must** match
- VUID-vkCmdDrawMeshTasksIndirectCountNV-OpImageWrite-08795
  If a VkImageView created with a format other than `VK_FORMAT_A8_UNORM_KHR` is accessed using `OpImageWrite` as a result of this command, then the `Type` of the `Texel` operand of that instruction **must** have at least as many components as the image view's format
- VUID-vkCmdDrawMeshTasksIndirectCountNV-OpImageWrite-08796
  If a VkImageView created with the format `VK_FORMAT_A8_UNORM_KHR` is accessed using `OpImageWrite` as a result of this command, then the `Type` of the `Texel` operand of that

instruction **must** have four components

- VUID-vkCmdDrawMeshTasksIndirectCountNV-OpImageWrite-04469
  If a VkBufferView is accessed using `OpImageWrite` as a result of this command, then the `Type` of the `Texel` operand of that instruction **must** have at least as many components as the buffer view's format
- VUID-vkCmdDrawMeshTasksIndirectCountNV-SampledType-04470
  If a VkImageView with a VkFormat that has a 64-bit component width is accessed as a result of this command, the `SampledType` of the `OpTypeImage` operand of that instruction **must** have a `Width` of 64
- VUID-vkCmdDrawMeshTasksIndirectCountNV-SampledType-04471
  If a VkImageView with a VkFormat that has a component width less than 64-bit is accessed as a result of this command, the `SampledType` of the `OpTypeImage` operand of that instruction **must** have a `Width` of 32
- VUID-vkCmdDrawMeshTasksIndirectCountNV-SampledType-04472
  If a VkBufferView with a VkFormat that has a 64-bit component width is accessed as a result of this command, the `SampledType` of the `OpTypeImage` operand of that instruction **must** have a `Width` of 64
- VUID-vkCmdDrawMeshTasksIndirectCountNV-SampledType-04473
  If a VkBufferView with a VkFormat that has a component width less than 64-bit is accessed as a result of this command, the `SampledType` of the `OpTypeImage` operand of that instruction **must** have a `Width` of 32
- VUID-vkCmdDrawMeshTasksIndirectCountNV-sparseImageInt64Atomics-04474
  If the `sparseImageInt64Atomics` feature is not enabled, VkImage objects created with the `VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT` flag **must** not be accessed by atomic instructions through an `OpTypeImage` with a `SampledType` with a `Width` of 64 by this command
- VUID-vkCmdDrawMeshTasksIndirectCountNV-sparseImageInt64Atomics-04475
  If the `sparseImageInt64Atomics` feature is not enabled, VkBuffer objects created with the `VK_BUFFER_CREATE_SPARSE_RESIDENCY_BIT` flag **must** not be accessed by atomic instructions through an `OpTypeImage` with a `SampledType` with a `Width` of 64 by this command
- VUID-vkCmdDrawMeshTasksIndirectCountNV-OpImageWeightedSampleQCOM-06971
  If `OpImageWeightedSampleQCOM` is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_2_WEIGHT_SAMPLED_IMAGE_BIT_QCOM`
- VUID-vkCmdDrawMeshTasksIndirectCountNV-OpImageWeightedSampleQCOM-06972
  If `OpImageWeightedSampleQCOM` uses a VkImageView as a sample weight image as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_2_WEIGHT_IMAGE_BIT_QCOM`
- VUID-vkCmdDrawMeshTasksIndirectCountNV-OpImageBoxFilterQCOM-06973
  If `OpImageBoxFilterQCOM` is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_2_BOX_FILTER_SAMPLED_BIT_QCOM`
- VUID-vkCmdDrawMeshTasksIndirectCountNV-OpImageBlockMatchSSDQCOM-06974
  If `OpImageBlockMatchSSDQCOM` is used to read from an VkImageView as a result of this

command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM

- VUID-vkCmdDrawMeshTasksIndirectCountNV-OpImageBlockMatchSADQCOM-06975
  If OpImageBlockMatchSADQCOM is used to read from an VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM
- VUID-vkCmdDrawMeshTasksIndirectCountNV-OpImageBlockMatchSADQCOM-06976
  If OpImageBlockMatchSADQCOM or OpImageBlockMatchSSDQCOM is used to read from a reference image as result of this command, then the specified reference coordinates **must** not fail integer texel coordinate validation
- VUID-vkCmdDrawMeshTasksIndirectCountNV-OpImageWeightedSampleQCOM-06977
  If OpImageWeightedSampleQCOM, OpImageBoxFilterQCOM, OpImageBlockMatchWindowSSDQCOM, OpImageBlockMatchWindowSADQCOM, OpImageBlockMatchGatherSSDQCOM, OpImageBlockMatchGatherSADQCOM, OpImageBlockMatchSSDQCOM, or OpImageBlockMatchSADQCOM uses a VkSampler as a result of this command, then the sampler **must** have been created with VK_SAMPLER_CREATE_IMAGE_PROCESSING_BIT_QCOM
- VUID-vkCmdDrawMeshTasksIndirectCountNV-OpImageWeightedSampleQCOM-06978
  If any command other than OpImageWeightedSampleQCOM, OpImageBoxFilterQCOM, OpImageBlockMatchWindowSSDQCOM, OpImageBlockMatchWindowSADQCOM, OpImageBlockMatchGatherSSDQCOM, OpImageBlockMatchGatherSADQCOM, OpImageBlockMatchSSDQCOM, or OpImageBlockMatchSADQCOM uses a VkSampler as a result of this command, then the sampler **must** not have been created with VK_SAMPLER_CREATE_IMAGE_PROCESSING_BIT_QCOM
- VUID-vkCmdDrawMeshTasksIndirectCountNV-OpImageBlockMatchWindow-09215
  If a OpImageBlockMatchWindow*QCOM or OpImageBlockMatchGather*QCOM instruction is used to read from an VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM
- VUID-vkCmdDrawMeshTasksIndirectCountNV-OpImageBlockMatchWindow-09216
  If a OpImageBlockMatchWindow*QCOM or OpImageBlockMatchGather*QCOM instruction is used to read from an VkImageView as a result of this command, then the image view's format **must** be a single-component format
- VUID-vkCmdDrawMeshTasksIndirectCountNV-OpImageBlockMatchWindow-09217
  If a OpImageBlockMatchWindow*QCOM or OpImageBlockMatchGather*QCOM read from a reference image as result of this command, then the specified reference coordinates **must** not fail integer texel coordinate validation
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-07288
  Any shader invocation executed by this command **must** terminate
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-09600
  If a descriptor with type equal to any of VK_DESCRIPTOR_TYPE_SAMPLE_WEIGHT_IMAGE_QCOM, VK_DESCRIPTOR_TYPE_BLOCK_MATCH_IMAGE_QCOM, VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE, VK_DESCRIPTOR_TYPE_STORAGE_IMAGE, or VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT is accessed as a result of this command, the image subresource identified by that descriptor **must** be in the image layout identified when the descriptor was written
- VUID-vkCmdDrawMeshTasksIndirectCountNV-renderPass-02684

The current render pass **must** be compatible with the `renderPass` member of the `VkGraphicsPipelineCreateInfo` structure specified when creating the `VkPipeline` bound to `VK_PIPELINE_BIND_POINT_GRAPHICS`

- VUID-vkCmdDrawMeshTasksIndirectCountNV-subpass-02685
  The subpass index of the current render pass **must** be equal to the `subpass` member of the `VkGraphicsPipelineCreateInfo` structure specified when creating the `VkPipeline` bound to `VK_PIPELINE_BIND_POINT_GRAPHICS`
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-07748
  If any shader statically accesses an input attachment, a valid descriptor **must** be bound to the pipeline via a descriptor set
- VUID-vkCmdDrawMeshTasksIndirectCountNV-OpTypeImage-07468
  If any shader executed by this pipeline accesses an `OpTypeImage` variable with a `Dim` operand of `SubpassData`, it **must** be decorated with an `InputAttachmentIndex` that corresponds to a valid input attachment in the current subpass
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-07469
  Input attachment views accessed in a subpass **must** be created with the same VkFormat as the corresponding subpass definition, and be created with a VkImageView that is compatible with the attachment referenced by the subpass' `pInputAttachments` [`InputAttachmentIndex`] in the currently bound VkFramebuffer as specified by Fragment Input Attachment Compatibility
- VUID-vkCmdDrawMeshTasksIndirectCountNV-pDepthInputAttachmentIndex-09595
  Input attachment views accessed in a dynamic render pass with a `InputAttachmentIndex` referenced by VkRenderingInputAttachmentIndexInfoKHR, or no `InputAttachmentIndex` if VkRenderingInputAttachmentIndexInfoKHR:`pDepthInputAttachmentIndex` or VkRenderingInputAttachmentIndexInfoKHR:`pStencilInputAttachmentIndex` are `NULL`, **must** be created with a VkImageView that is compatible with the corresponding color, depth, or stencil attachment in VkRenderingInfo
- VUID-vkCmdDrawMeshTasksIndirectCountNV-pDepthInputAttachmentIndex-09596
  Input attachment views accessed in a dynamic render pass via a shader object **must** have an `InputAttachmentIndex` if both VkRenderingInputAttachmentIndexInfoKHR:`pDepthInputAttachmentIndex` and VkRenderingInputAttachmentIndexInfoKHR:`pStencilInputAttachmentIndex` are non-`NULL`
- VUID-vkCmdDrawMeshTasksIndirectCountNV-InputAttachmentIndex-09597
  If an input attachment view accessed in a dynamic render pass via a shader object has an `InputAttachmentIndex`, the `InputAttachmentIndex` **must** match an index in VkRenderingInputAttachmentIndexInfoKHR
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-06537
  Memory backing image subresources used as attachments in the current render pass **must** not be written in any way other than as an attachment by this command
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-09000
  If a color attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass, it is not in the `VK_IMAGE_LAYOUT_ATTACHMENT_FEEDBACK_LOOP_OPTIMAL_EXT` image layout, and either:

- the VK_PIPELINE_CREATE_COLOR_ATTACHMENT_FEEDBACK_LOOP_BIT_EXT is set on the currently bound pipeline or
- the last call to vkCmdSetAttachmentFeedbackLoopEnableEXT included VK_IMAGE_ASPECT_COLOR_BIT and
  - there is no currently bound graphics pipeline or
  - the currently bound graphics pipeline was created with VK_DYNAMIC_STATE_ATTACHMENT_FEEDBACK_LOOP_ENABLE_EXT

it **must** not be accessed in any way other than as an attachment by this command

- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-09001
  If a depth attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass, it is not in the VK_IMAGE_LAYOUT_ATTACHMENT_FEEDBACK_LOOP_OPTIMAL_EXT image layout, and either:
  - the VK_PIPELINE_CREATE_DEPTH_STENCIL_ATTACHMENT_FEEDBACK_LOOP_BIT_EXT is set on the currently bound pipeline or
  - the last call to vkCmdSetAttachmentFeedbackLoopEnableEXT included VK_IMAGE_ASPECT_DEPTH_BIT and
    - there is no currently bound graphics pipeline or
    - the currently bound graphics pipeline was created with VK_DYNAMIC_STATE_ATTACHMENT_FEEDBACK_LOOP_ENABLE_EXT

  it **must** not be accessed in any way other than as an attachment by this command

- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-09002
  If a stencil attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass, it is not in the VK_IMAGE_LAYOUT_ATTACHMENT_FEEDBACK_LOOP_OPTIMAL_EXT image layout, and either:
  - the VK_PIPELINE_CREATE_DEPTH_STENCIL_ATTACHMENT_FEEDBACK_LOOP_BIT_EXT is set on the currently bound pipeline or
  - the last call to vkCmdSetAttachmentFeedbackLoopEnableEXT included VK_IMAGE_ASPECT_STENCIL_BIT and
    - there is no currently bound graphics pipeline or
    - the currently bound graphics pipeline was created with VK_DYNAMIC_STATE_ATTACHMENT_FEEDBACK_LOOP_ENABLE_EXT

  it **must** not be accessed in any way other than as an attachment by this command

- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-09003
  If an attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass, it **must** not be accessed in any way other than as an attachment, storage image, or sampled image by this command
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-06539
  If any previously recorded command in the current subpass accessed an image subresource used as an attachment in this subpass in any way other than as an

attachment, this command **must** not write to that image subresource as an attachment

- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-06886
  If the current render pass instance uses a depth/stencil attachment with a read-only layout for the depth aspect, depth writes **must** be disabled
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-06887
  If the current render pass instance uses a depth/stencil attachment with a read-only layout for the stencil aspect, both front and back `writeMask` are not zero, and stencil test is enabled, all stencil ops **must** be `VK_STENCIL_OP_KEEP`
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-07831
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VIEWPORT` dynamic state enabled then vkCmdSetViewport **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-07832
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_SCISSOR` dynamic state enabled then vkCmdSetScissor **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-07833
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_LINE_WIDTH` dynamic state enabled then vkCmdSetLineWidth **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-08617
  If a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE`, and the most recent call to vkCmdSetPolygonModeEXT in the current command buffer set `polygonMode` to `VK_POLYGON_MODE_LINE`, vkCmdSetLineWidth **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-08618
  If a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE`, and the most recent call to vkCmdSetPrimitiveTopologyEXT in the current command buffer set `primitiveTopology` to any line topology, vkCmdSetLineWidth **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-08619
  If a shader object that outputs line primitives is bound to the `VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT` or `VK_SHADER_STAGE_GEOMETRY_BIT` stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE`, vkCmdSetLineWidth **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-07834
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_DEPTH_BIAS` dynamic state enabled, the current

value of rasterizerDiscardEnable is VK_FALSE, and the current value of depthBiasEnable is VK_TRUE, then vkCmdSetDepthBounds or vkCmdSetDepthBias2EXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-07835
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_BLEND_CONSTANTS dynamic state enabled then vkCmdSetBlendConstants **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-08621
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetColorBlendEnableEXT in the current command buffer set any element of pColorBlendEnables to VK_TRUE, and the most recent call to vkCmdSetColorBlendEquationEXT in the current command buffer set the same element of pColorBlendEquations to a VkColorBlendEquationEXT structure with any VkBlendFactor member with a value of VK_BLEND_FACTOR_CONSTANT_COLOR, VK_BLEND_FACTOR_ONE_MINUS_CONSTANT_COLOR, VK_BLEND_FACTOR_CONSTANT_ALPHA, or VK_BLEND_FACTOR_ONE_MINUS_CONSTANT_ALPHA, vkCmdSetBlendConstants **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-07836
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_BOUNDS dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of depthBoundsTestEnable is VK_TRUE, then vkCmdSetDepthBounds **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-07837
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_STENCIL_COMPARE_MASK dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of stencilTestEnable is VK_TRUE, then vkCmdSetStencilCompareMask **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-07838
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_STENCIL_WRITE_MASK dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of stencilTestEnable is VK_TRUE, then vkCmdSetStencilWriteMask **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-07839
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which

was created with the VK_DYNAMIC_STATE_STENCIL_REFERENCE dynamic state enabled, the current value of and rasterizerDiscardEnable is VK_FALSE, the current value of stencilTestEnable is VK_TRUE, then vkCmdSetStencilReference **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectCountNV-maxMultiviewInstanceIndex-02688
  If the draw is recorded in a render pass instance with multiview enabled, the maximum instance index **must** be less than or equal to VkPhysicalDeviceMultiviewProperties::maxMultiviewInstanceIndex
- VUID-vkCmdDrawMeshTasksIndirectCountNV-sampleLocationsEnable-02689
  If the bound graphics pipeline was created with VkPipelineSampleLocationsStateCreateInfoEXT::sampleLocationsEnable set to VK_TRUE and the current subpass has a depth/stencil attachment, then that attachment **must** have been created with the VK_IMAGE_CREATE_SAMPLE_LOCATIONS_COMPATIBLE_DEPTH_BIT_EXT bit set
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-06666
  If the VK_EXT_sample_locations extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of sampleLocationsEnable is VK_TRUE, then vkCmdSetSampleLocationsEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-07840
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_CULL_MODE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetCullModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-07841
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_FRONT_FACE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetFrontFaceEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-07843
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_TEST_ENABLE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, vkCmdSetDepthTestEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-07844
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_WRITE_ENABLE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetDepthWriteEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-07845
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_COMPARE_OP dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of depthTestEnable is VK_TRUE, then vkCmdSetDepthCompareOpEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-07846
  If the depthBounds feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_BOUNDS_TEST_ENABLE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetDepthBoundsTestEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-07847
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_STENCIL_TEST_ENABLE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetStencilTestEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-07848
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_STENCIL_OP dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, the current value of stencilTestEnable is VK_TRUE, then vkCmdSetStencilOpEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountNV-viewportCount-03417
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, but not the VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT dynamic state enabled, then vkCmdSetViewportWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the viewportCount parameter of vkCmdSetViewportWithCountEXT **must** match the VkPipelineViewportStateCreateInfo::scissorCount of the pipeline
- VUID-vkCmdDrawMeshTasksIndirectCountNV-scissorCount-03418
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT dynamic state enabled, but not the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, then vkCmdSetScissorWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the scissorCount parameter of vkCmdSetScissorWithCountEXT **must** match the VkPipelineViewportStateCreateInfo::viewportCount of the pipeline
- VUID-vkCmdDrawMeshTasksIndirectCountNV-viewportCount-03419
  If the bound graphics pipeline state was created with both the VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT and VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic

states enabled then both vkCmdSetViewportWithCountEXT and vkCmdSetScissorWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the viewportCount parameter of vkCmdSetViewportWithCountEXT **must** match the scissorCount parameter of vkCmdSetScissorWithCountEXT

- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-08635
  If a shader object is bound to any graphics stage, then both vkCmdSetViewportWithCountEXT and vkCmdSetScissorWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the viewportCount parameter of vkCmdSetViewportWithCountEXT **must** match the scissorCount parameter of vkCmdSetScissorWithCountEXT

- VUID-vkCmdDrawMeshTasksIndirectCountNV-viewportCount-04137
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, but not the VK_DYNAMIC_STATE_VIEWPORT_W_SCALING_NV dynamic state enabled, then the bound graphics pipeline **must** have been created with VkPipelineViewportWScalingStateCreateInfoNV::viewportCount greater or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT

- VUID-vkCmdDrawMeshTasksIndirectCountNV-viewportCount-04138
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT and VK_DYNAMIC_STATE_VIEWPORT_W_SCALING_NV dynamic states enabled then the viewportCount parameter in the last call to vkCmdSetViewportWScalingNV **must** be greater than or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT

- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-09232
  If the VK_NV_clip_space_w_scaling extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetViewportWScalingEnableNV in the current command buffer set viewportWScalingEnable to VK_TRUE, then vkCmdSetViewportWScalingNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-08636
  If the VK_NV_clip_space_w_scaling extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetViewportWScalingEnableNV in the current command buffer set viewportWScalingEnable to VK_TRUE, then the viewportCount parameter in the last call to vkCmdSetViewportWScalingNV **must** be greater than or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT

- VUID-vkCmdDrawMeshTasksIndirectCountNV-viewportCount-04139
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, but not the VK_DYNAMIC_STATE_VIEWPORT_SHADING_RATE_PALETTE_NV dynamic state enabled, then the bound graphics pipeline **must** have been created with VkPipelineViewportShadingRateImageStateCreateInfoNV::viewportCount greater or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT

- VUID-vkCmdDrawMeshTasksIndirectCountNV-viewportCount-04140
  If the bound graphics pipeline state was created with the

VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT and VK_DYNAMIC_STATE_VIEWPORT_SHADING_RATE_PALETTE_NV dynamic states enabled then the viewportCount parameter in the last call to vkCmdSetViewportShadingRatePaletteNV **must** be greater than or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT

- VUID-vkCmdDrawMeshTasksIndirectCountNV-shadingRateImage-09233
  If the shadingRateImage feature is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetCoarseSampleOrderNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountNV-shadingRateImage-09234
  If the shadingRateImage feature is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetShadingRateImageEnableNV in the current command buffer set shadingRateImageEnable to VK_TRUE, then vkCmdSetViewportShadingRatePaletteNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-08637
  If the shadingRateImage feature is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetShadingRateImageEnableNV in the current command buffer set shadingRateImageEnable to VK_TRUE, then the viewportCount parameter in the last call to vkCmdSetViewportShadingRatePaletteNV **must** be greater than or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawMeshTasksIndirectCountNV-VkPipelineVieportCreateInfo-04141
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled and a VkPipelineViewportSwizzleStateCreateInfoNV structure chained from VkPipelineViewportStateCreateInfo, then the bound graphics pipeline **must** have been created with VkPipelineViewportSwizzleStateCreateInfoNV::viewportCount greater or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawMeshTasksIndirectCountNV-VkPipelineVieportCreateInfo-04142
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled and a VkPipelineViewportExclusiveScissorStateCreateInfoNV structure chained from VkPipelineViewportStateCreateInfo, then the bound graphics pipeline **must** have been created with VkPipelineViewportExclusiveScissorStateCreateInfoNV::exclusiveScissorCount greater or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-07878
  If the exclusiveScissor feature is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the

VK_DYNAMIC_STATE_EXCLUSIVE_SCISSOR_ENABLE_NV dynamic state enabled, then vkCmdSetExclusiveScissorEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-07879
  If the exclusiveScissor feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_EXCLUSIVE_SCISSOR_NV dynamic state enabled, and the most recent call to vkCmdSetExclusiveScissorEnableNV in the current command buffer set any element of pExclusiveScissorEnables to VK_TRUE, then vkCmdSetExclusiveScissorNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-04876
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_RASTERIZER_DISCARD_ENABLE dynamic state enabled, then vkCmdSetRasterizerDiscardEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-04877
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_BIAS_ENABLE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetDepthBiasEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountNV-logicOp-04878
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_LOGIC_OP_EXT dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of logicOpEnable is VK_TRUE, then vkCmdSetLogicOpEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountNV-primitiveFragmentShadingRateWithMultipleViewports-04552
  If the primitiveFragmentShadingRateWithMultipleViewports limit is not supported, the bound graphics pipeline was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, and any of the shader stages of the bound graphics pipeline write to the PrimitiveShadingRateKHR built-in, then vkCmdSetViewportWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the viewportCount parameter of vkCmdSetViewportWithCountEXT **must** be 1
- VUID-vkCmdDrawMeshTasksIndirectCountNV-primitiveFragmentShadingRateWithMultipleViewports-08642
  If the primitiveFragmentShadingRateWithMultipleViewports limit is not supported, and any shader object bound to a graphics stage writes to the PrimitiveShadingRateKHR built-in, then vkCmdSetViewportWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the viewportCount parameter of vkCmdSetViewportWithCountEXT **must** be 1
- VUID-vkCmdDrawMeshTasksIndirectCountNV-blendEnable-04727
  If rasterization is not disabled in the bound graphics pipeline, then for each color

attachment in the subpass, if the corresponding image view's format features do not contain VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT, then the blendEnable member of the corresponding element of the pAttachments member of pColorBlendState **must** be VK_FALSE

- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-08643
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then for each color attachment in the render pass, if the corresponding image view's format features do not contain VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT, then the corresponding member of pColorBlendEnables in the most recent call to vkCmdSetColorBlendEnableEXT in the current command buffer that affected that attachment index **must** have been VK_FALSE
- VUID-vkCmdDrawMeshTasksIndirectCountNV-multisampledRenderToSingleSampled-07284
  If rasterization is not disabled in the bound graphics pipeline, and none of the following is enabled:
  - the VK_AMD_mixed_attachment_samples extension
  - the VK_NV_framebuffer_mixed_samples extension
  - the multisampledRenderToSingleSampled feature

  then rasterizationSamples for the currently bound graphics pipeline **must** be the same as the current subpass color and/or depth/stencil attachments
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-08644
  If a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and none of the following is enabled:
  - the VK_AMD_mixed_attachment_samples extension
  - the VK_NV_framebuffer_mixed_samples extension
  - the multisampledRenderToSingleSampled feature

  then the most recent call to vkCmdSetRasterizationSamplesEXT in the current command buffer **must** have set rasterizationSamples to be the same as the number of samples for the current render pass color and/or depth/stencil attachments
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-08876
  If a shader object is bound to any graphics stage, the current render pass instance **must** have been begun with vkCmdBeginRenderingKHR
- VUID-vkCmdDrawMeshTasksIndirectCountNV-imageView-06172
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the imageView member of pDepthAttachment is not VK_NULL_HANDLE, and the layout member of pDepthAttachment is VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL, this command **must** not write any values to the depth attachment
- VUID-vkCmdDrawMeshTasksIndirectCountNV-imageView-06173
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the

imageView member of pStencilAttachment is not VK_NULL_HANDLE, and the layout member of pStencilAttachment is VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL, this command **must** not write any values to the stencil attachment

- VUID-vkCmdDrawMeshTasksIndirectCountNV-imageView-06174
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the imageView member of pDepthAttachment is not VK_NULL_HANDLE, and the layout member of pDepthAttachment is VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_STENCIL_ATTACHMENT_OPTIMAL, this command **must** not write any values to the depth attachment
- VUID-vkCmdDrawMeshTasksIndirectCountNV-imageView-06175
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the imageView member of pStencilAttachment is not VK_NULL_HANDLE, and the layout member of pStencilAttachment is VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_STENCIL_READ_ONLY_OPTIMAL, this command **must** not write any values to the stencil attachment
- VUID-vkCmdDrawMeshTasksIndirectCountNV-imageView-06176
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the imageView member of pDepthAttachment is not VK_NULL_HANDLE, and the layout member of pDepthAttachment is VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL, this command **must** not write any values to the depth attachment
- VUID-vkCmdDrawMeshTasksIndirectCountNV-imageView-06177
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the imageView member of pStencilAttachment is not VK_NULL_HANDLE, and the layout member of pStencilAttachment is VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL, this command **must** not write any values to the stencil attachment
- VUID-vkCmdDrawMeshTasksIndirectCountNV-viewMask-06178
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the currently bound graphics pipeline **must** have been created with a VkPipelineRenderingCreateInfo::viewMask equal to VkRenderingInfo::viewMask
- VUID-vkCmdDrawMeshTasksIndirectCountNV-colorAttachmentCount-06179
  If the dynamicRenderingUnusedAttachments feature is not enabled and the current render pass instance was begun with vkCmdBeginRenderingKHR, the currently bound graphics pipeline **must** have been created with a VkPipelineRenderingCreateInfo::colorAttachmentCount equal to VkRenderingInfo::colorAttachmentCount
- VUID-vkCmdDrawMeshTasksIndirectCountNV-dynamicRenderingUnusedAttachments-08910
  If the dynamicRenderingUnusedAttachments feature is not enabled, and the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingInfo::colorAttachmentCount greater than 0, then each element of the VkRenderingInfo::pColorAttachments array with an imageView not equal to VK_NULL_HANDLE **must** have been created with a VkFormat equal to the corresponding element of VkPipelineRenderingCreateInfo::pColorAttachmentFormats used to create the currently bound graphics pipeline
- VUID-vkCmdDrawMeshTasksIndirectCountNV-dynamicRenderingUnusedAttachments-08912
  If the dynamicRenderingUnusedAttachments feature is not enabled, and the current render

pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingInfo::colorAttachmentCount greater than 0, then each element of the VkRenderingInfo::pColorAttachments array with an imageView equal to VK_NULL_HANDLE **must** have the corresponding element of VkPipelineRenderingCreateInfo::pColorAttachmentFormats used to create the currently bound pipeline equal to VK_FORMAT_UNDEFINED

- VUID-vkCmdDrawMeshTasksIndirectCountNV-dynamicRenderingUnusedAttachments-08911
  If the dynamicRenderingUnusedAttachments feature is enabled, and the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingInfo::colorAttachmentCount greater than 0, then each element of the VkRenderingInfo::pColorAttachments array with an imageView not equal to VK_NULL_HANDLE **must** have been created with a VkFormat equal to the corresponding element of VkPipelineRenderingCreateInfo::pColorAttachmentFormats used to create the currently bound graphics pipeline, or the corresponding element of VkPipelineRenderingCreateInfo::pColorAttachmentFormats, if it exists, **must** be VK_FORMAT_UNDEFINED

- VUID-vkCmdDrawMeshTasksIndirectCountNV-colorAttachmentCount-09362
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, with a VkRenderingInfo::colorAttachmentCount equal to 1, there is no shader object bound to any graphics stage, and a color attachment with a resolve mode of VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID, each element of the VkRenderingInfo::pColorAttachments array with a resolveImageView not equal to VK_NULL_HANDLE **must** have been created with an image created with a VkExternalFormatANDROID::externalFormat value equal to the VkExternalFormatANDROID::externalFormat value used to create the currently bound graphics pipeline

- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-09363
  If there is no shader object bound to any graphics stage, the current render pass instance was begun with vkCmdBeginRenderingKHR and a VkRenderingInfo::colorAttachmentCount equal to 1, and a color attachment with a resolve mode of VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID, each element of the VkRenderingInfo::pColorAttachments array with a imageView not equal to VK_NULL_HANDLE **must** have been created with an image created with a VkExternalFormatANDROID::externalFormat value equal to the VkExternalFormatANDROID::externalFormat value used to create the currently bound graphics pipeline

- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-09364
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is no shader object bound to any graphics stage, and the currently bound graphics pipeline was created with a non-zero VkExternalFormatANDROID::externalFormat value and with the VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT dynamic state enabled, then vkCmdSetColorBlendEnableEXT **must** have set the blend enable to VK_FALSE prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-09365
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is no shader object bound to any graphics stage, and the currently bound graphics pipeline

was created with a non-zero VkExternalFormatANDROID::externalFormat value and with the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT dynamic state enabled, then vkCmdSetRasterizationSamplesEXT **must** have set rasterizationSamples to VK_SAMPLE_COUNT_1_BIT prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-09366
  If there is a shader object bound to any graphics stage, and the current render pass includes a color attachment that uses the VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID resolve mode, then vkCmdSetColorBlendEnableEXT **must** have set blend enable to VK_FALSE prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectCountNV-rasterizationSamples-09367
  If there is a shader object bound to any graphics stage, and the current render pass includes a color attachment that uses the VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID resolve mode, then vkCmdSetRasterizationSamplesEXT **must** have set rasterizationSamples to VK_SAMPLE_COUNT_1_BIT prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-09368
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is no shader object bound to any graphics stage, and the currently bound graphics pipeline was created with a non-zero VkExternalFormatANDROID::externalFormat value and with the VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR dynamic state enabled, then vkCmdSetFragmentShadingRateKHR **must** have set pFragmentSize->width to 1 prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-09369
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is no shader object bound to any graphics stage, and the currently bound graphics pipeline was created with a non-zero VkExternalFormatANDROID::externalFormat value and with the VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR dynamic state enabled, then vkCmdSetFragmentShadingRateKHR **must** have set pFragmentSize->height to 1 prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectCountNV-pFragmentSize-09370
  If there is a shader object bound to any graphics stage, and the current render pass includes a color attachment that uses the VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID resolve mode, then vkCmdSetFragmentShadingRateKHR **must** have set pFragmentSize->width to 1 prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectCountNV-pFragmentSize-09371
  If there is a shader object bound to any graphics stage, and the current render pass includes a color attachment that uses the VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID resolve mode, then vkCmdSetFragmentShadingRateKHR **must** have set pFragmentSize->height to 1 prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-07749
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_WRITE_ENABLE_EXT dynamic state enabled then

vkCmdSetColorWriteEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-08646
  If the `colorWriteEnable` feature is enabled on the device, and a shader object is bound to the `VK_SHADER_STAGE_FRAGMENT_BIT` stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE`, then vkCmdSetColorWriteEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountNV-attachmentCount-07750
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_COLOR_WRITE_ENABLE_EXT` dynamic state enabled then the `attachmentCount` parameter of `vkCmdSetColorWriteEnableEXT` **must** be greater than or equal to the `VkPipelineColorBlendStateCreateInfo`::`attachmentCount` of the currently bound graphics pipeline
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-08647
  If the `colorWriteEnable` feature is enabled on the device, and a shader object is bound to the `VK_SHADER_STAGE_FRAGMENT_BIT` stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE`, then the `attachmentCount` parameter of most recent call to `vkCmdSetColorWriteEnableEXT` in the current command buffer **must** be greater than or equal to the number of color attachments in the current render pass instance
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-07751
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_DISCARD_RECTANGLE_EXT` dynamic state enabled then vkCmdSetDiscardRectangleEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command for each discard rectangle in VkPipelineDiscardRectangleStateCreateInfoEXT::`discardRectangleCount`
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-07880
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_DISCARD_RECTANGLE_ENABLE_EXT` dynamic state enabled then vkCmdSetDiscardRectangleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountNV-rasterizerDiscardEnable-09236
  If the `VK_EXT_discard_rectangles` extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE`, and the most recent call to vkCmdSetDiscardRectangleEnableEXT in the current command buffer set `discardRectangleEnable` to `VK_TRUE`, then vkCmdSetDiscardRectangleEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-08648
  If the `VK_EXT_discard_rectangles` extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE`, then

vkCmdSetDiscardRectangleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-07881
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_DISCARD_RECTANGLE_MODE_EXT dynamic state enabled then vkCmdSetDiscardRectangleModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-08649
  If the VK_EXT_discard_rectangles extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetDiscardRectangleEnableEXT in the current command buffer set discardRectangleEnable to VK_TRUE, then vkCmdSetDiscardRectangleModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectCountNV-dynamicRenderingUnusedAttachments-08913
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is not enabled, and VkRenderingInfo::pDepthAttachment->imageView was VK_NULL_HANDLE, the value of VkPipelineRenderingCreateInfo::depthAttachmentFormat used to create the currently bound graphics pipeline **must** be equal to VK_FORMAT_UNDEFINED

- VUID-vkCmdDrawMeshTasksIndirectCountNV-dynamicRenderingUnusedAttachments-08914
  If current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is not enabled, and VkRenderingInfo::pDepthAttachment->imageView was not VK_NULL_HANDLE, the value of VkPipelineRenderingCreateInfo::depthAttachmentFormat used to create the currently bound graphics pipeline **must** be equal to the VkFormat used to create VkRenderingInfo::pDepthAttachment->imageView

- VUID-vkCmdDrawMeshTasksIndirectCountNV-dynamicRenderingUnusedAttachments-08915
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is enabled, VkRenderingInfo::pDepthAttachment->imageView was not VK_NULL_HANDLE, and the value of VkPipelineRenderingCreateInfo::depthAttachmentFormat used to create the currently bound graphics pipeline was not equal to the VkFormat used to create VkRenderingInfo::pDepthAttachment->imageView, the value of the format **must** be VK_FORMAT_UNDEFINED

- VUID-vkCmdDrawMeshTasksIndirectCountNV-dynamicRenderingUnusedAttachments-08916
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is not enabled, and VkRenderingInfo::pStencilAttachment->imageView was VK_NULL_HANDLE, the value of VkPipelineRenderingCreateInfo::stencilAttachmentFormat used to create the currently bound graphics pipeline **must** be equal to VK_FORMAT_UNDEFINED

- VUID-vkCmdDrawMeshTasksIndirectCountNV-dynamicRenderingUnusedAttachments-08917
  If current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is not enabled, and VkRenderingInfo::pStencilAttachment->imageView was not VK_NULL_HANDLE, the value of VkPipelineRenderingCreateInfo::stencilAttachmentFormat used to create the currently bound graphics pipeline **must** be equal to the VkFormat used to create VkRenderingInfo::pStencilAttachment->imageView
- VUID-vkCmdDrawMeshTasksIndirectCountNV-dynamicRenderingUnusedAttachments-08918
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is enabled, VkRenderingInfo::pStencilAttachment->imageView was not VK_NULL_HANDLE, and the value of VkPipelineRenderingCreateInfo::stencilAttachmentFormat used to create the currently bound graphics pipeline was not equal to the VkFormat used to create VkRenderingInfo::pStencilAttachment->imageView, the value of the format **must** be VK_FORMAT_UNDEFINED
- VUID-vkCmdDrawMeshTasksIndirectCountNV-imageView-06183
  If the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingFragmentShadingRateAttachmentInfoKHR::imageView was not VK_NULL_HANDLE, the currently bound graphics pipeline **must** have been created with VK_PIPELINE_CREATE_RENDERING_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR
- VUID-vkCmdDrawMeshTasksIndirectCountNV-imageView-06184
  If the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingFragmentDensityMapAttachmentInfoEXT::imageView was not VK_NULL_HANDLE, the currently bound graphics pipeline **must** have been created with VK_PIPELINE_CREATE_RENDERING_FRAGMENT_DENSITY_MAP_ATTACHMENT_BIT_EXT
- VUID-vkCmdDrawMeshTasksIndirectCountNV-colorAttachmentCount-06185
  If the currently bound pipeline was created with a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and the current render pass instance was begun with vkCmdBeginRenderingKHR with a VkRenderingInfo::colorAttachmentCount parameter greater than 0, then each element of the VkRenderingInfo::pColorAttachments array with a imageView not equal to VK_NULL_HANDLE **must** have been created with a sample count equal to the corresponding element of the pColorAttachmentSamples member of VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV used to create the currently bound graphics pipeline
- VUID-vkCmdDrawMeshTasksIndirectCountNV-pDepthAttachment-06186
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the currently bound pipeline was created with a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and VkRenderingInfo::pDepthAttachment->imageView was not VK_NULL_HANDLE, the value of the depthStencilAttachmentSamples member of VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV used to create the currently bound graphics pipeline **must** be equal to the sample count used to create VkRenderingInfo::pDepthAttachment->imageView
- VUID-vkCmdDrawMeshTasksIndirectCountNV-pStencilAttachment-06187
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the

currently bound pipeline was created with a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and VkRenderingInfo::pStencilAttachment->imageView was not VK_NULL_HANDLE, the value of the depthStencilAttachmentSamples member of VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV used to create the currently bound graphics pipeline **must** be equal to the sample count used to create VkRenderingInfo::pStencilAttachment->imageView

- VUID-vkCmdDrawMeshTasksIndirectCountNV-multisampledRenderToSingleSampled-07285
  If the currently bound pipeline was created without a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and the multisampledRenderToSingleSampled feature is not enabled, and the current render pass instance was begun with vkCmdBeginRenderingKHR with a VkRenderingInfo::colorAttachmentCount parameter greater than 0, then each element of the VkRenderingInfo::pColorAttachments array with a imageView not equal to VK_NULL_HANDLE **must** have been created with a sample count equal to the value of rasterizationSamples for the currently bound graphics pipeline
- VUID-vkCmdDrawMeshTasksIndirectCountNV-multisampledRenderToSingleSampled-07286
  If the currently bound pipeline was created without a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and the multisampledRenderToSingleSampled feature is not enabled, and VkRenderingInfo::pDepthAttachment->imageView was not VK_NULL_HANDLE, the value of rasterizationSamples for the currently bound graphics pipeline **must** be equal to the sample count used to create VkRenderingInfo::pDepthAttachment->imageView
- VUID-vkCmdDrawMeshTasksIndirectCountNV-multisampledRenderToSingleSampled-07287
  If the currently bound pipeline was created without a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and the multisampledRenderToSingleSampled feature is not enabled, and VkRenderingInfo::pStencilAttachment->imageView was not VK_NULL_HANDLE, the value of rasterizationSamples for the currently bound graphics pipeline **must** be equal to the sample count used to create VkRenderingInfo::pStencilAttachment->imageView
- VUID-vkCmdDrawMeshTasksIndirectCountNV-pNext-07935
  If this command has been called inside a render pass instance started with vkCmdBeginRenderingKHR, and the pNext chain of VkRenderingInfo includes a VkMultisampledRenderToSingleSampledInfoEXT structure with multisampledRenderToSingleSampledEnable equal to VK_TRUE, then the value of rasterizationSamples for the currently bound graphics pipeline **must** be equal to VkMultisampledRenderToSingleSampledInfoEXT::rasterizationSamples
- VUID-vkCmdDrawMeshTasksIndirectCountNV-renderPass-06198
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the currently bound pipeline **must** have been created with a VkGraphicsPipelineCreateInfo::renderPass equal to VK_NULL_HANDLE
- VUID-vkCmdDrawMeshTasksIndirectCountNV-pColorAttachments-08963
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is a

graphics pipeline bound with a fragment shader that statically writes to a color attachment, the color write mask is not zero, color writes are enabled, and the corresponding element of the VkRenderingInfo::pColorAttachments->imageView was not VK_NULL_HANDLE, then the corresponding element of VkPipelineRenderingCreateInfo::pColorAttachmentFormats used to create the pipeline **must** not be VK_FORMAT_UNDEFINED

- VUID-vkCmdDrawMeshTasksIndirectCountNV-pDepthAttachment-08964
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is a graphics pipeline bound, depth test is enabled, depth write is enabled, and the VkRenderingInfo::pDepthAttachment->imageView was not VK_NULL_HANDLE, then the VkPipelineRenderingCreateInfo::depthAttachmentFormat used to create the pipeline **must** not be VK_FORMAT_UNDEFINED

- VUID-vkCmdDrawMeshTasksIndirectCountNV-pStencilAttachment-08965
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is a graphics pipeline bound, stencil test is enabled and the VkRenderingInfo::pStencilAttachment->imageView was not VK_NULL_HANDLE, then the VkPipelineRenderingCreateInfo::stencilAttachmentFormat used to create the pipeline **must** not be VK_FORMAT_UNDEFINED

- VUID-vkCmdDrawMeshTasksIndirectCountNV-primitivesGeneratedQueryWithRasterizerDiscard-06708
  If the primitivesGeneratedQueryWithRasterizerDiscard feature is not enabled and the VK_QUERY_TYPE_PRIMITIVES_GENERATED_EXT query is active, rasterization discard **must** not be enabled

- VUID-vkCmdDrawMeshTasksIndirectCountNV-primitivesGeneratedQueryWithNonZeroStreams-06709
  If the primitivesGeneratedQueryWithNonZeroStreams feature is not enabled and the VK_QUERY_TYPE_PRIMITIVES_GENERATED_EXT query is active, the bound graphics pipeline **must** not have been created with a non-zero value in VkPipelineRasterizationStateStreamCreateInfoEXT::rasterizationStream

- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-07619
  If a shader object is bound to the VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_TESSELLATION_DOMAIN_ORIGIN_EXT dynamic state enabled, then vkCmdSetTessellationDomainOriginEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-07620
  If the depthClamp feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_CLAMP_ENABLE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetDepthClampEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-07621
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_POLYGON_MODE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetPolygonModeEXT

**must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-07622
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT` dynamic state enabled, and the current value of `rasterizerDiscardEnable` is `VK_FALSE`, then vkCmdSetRasterizationSamplesEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-07623
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_SAMPLE_MASK_EXT` dynamic state enabled, and the current value of `rasterizerDiscardEnable` is `VK_FALSE`, then vkCmdSetSampleMaskEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectCountNV-alphaToCoverageEnable-08919
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_ALPHA_TO_COVERAGE_ENABLE_EXT` dynamic state enabled, and `alphaToCoverageEnable` was `VK_TRUE` in the last call to vkCmdSetAlphaToCoverageEnableEXT, then the Fragment Output Interface **must** contain a variable for the alpha `Component` word in `Location` 0 at `Index` 0

- VUID-vkCmdDrawMeshTasksIndirectCountNV-alphaToCoverageEnable-08920
  If a shader object is bound to any graphics stage, and the most recent call to vkCmdSetAlphaToCoverageEnableEXT in the current command buffer set `alphaToCoverageEnable` to `VK_TRUE`, then the Fragment Output Interface **must** contain a variable for the alpha `Component` word in `Location` 0 at `Index` 0

- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-07624
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_ALPHA_TO_COVERAGE_ENABLE_EXT` dynamic state enabled, and the current value of `rasterizerDiscardEnable` is `VK_FALSE`, then vkCmdSetAlphaToCoverageEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-07625
  If the `alphaToOne` feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_ALPHA_TO_ONE_ENABLE_EXT` dynamic state enabled, and the current value of `rasterizerDiscardEnable` is `VK_FALSE`, then vkCmdSetAlphaToOneEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-07626
  If the `logicOp` feature is enabled, a shader object is bound to the `VK_SHADER_STAGE_FRAGMENT_BIT` stage or a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_LOGIC_OP_ENABLE_EXT` dynamic state enabled, and the current value of `rasterizerDiscardEnable` is `VK_FALSE`, then vkCmdSetLogicOpEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-07627
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT dynamic state enabled then vkCmdSetColorBlendEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-08657
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and both the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE and there are color attachments bound, then vkCmdSetColorBlendEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-07628
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_EQUATION_EXT dynamic state enabled then vkCmdSetColorBlendEquationEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-08658
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetColorBlendEnableEXT for any attachment set that attachment's value in pColorBlendEnables to VK_TRUE, then vkCmdSetColorBlendEquationEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-07629
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_WRITE_MASK_EXT dynamic state enabled then vkCmdSetColorWriteMaskEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-08659
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and both the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE and there are color attachments bound, then vkCmdSetColorWriteMaskEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-07630
  If the geometryStreams feature is enabled, and a shader object is bound to the VK_SHADER_STAGE_GEOMETRY_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_RASTERIZATION_STREAM_EXT dynamic state enabled, then vkCmdSetRasterizationStreamEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-07631
  If the VK_EXT_conservative_rasterization extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_CONSERVATIVE_RASTERIZATION_MODE_EXT dynamic state enabled, and the

current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetConservativeRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-07632
  If the VK_EXT_conservative_rasterization extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_EXTRA_PRIMITIVE_OVERESTIMATION_SIZE_EXT dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of conservativeRasterizationMode is VK_CONSERVATIVE_RASTERIZATION_MODE_OVERESTIMATE_EXT, then vkCmdSetExtraPrimitiveOverestimationSizeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-07633
  If the depthClipEnable feature is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_CLIP_ENABLE_EXT dynamic state, then vkCmdSetDepthClipEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-07634
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT dynamic state enabled then vkCmdSetSampleLocationsEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-08664
  If the VK_EXT_sample_locations extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetSampleLocationsEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-07635
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_ADVANCED_EXT dynamic state enabled then vkCmdSetColorBlendAdvancedEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountNV-rasterizerDiscardEnable-09416
  If the VK_EXT_blend_operation_advanced extension is enabled, and a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then at least one of vkCmdSetColorBlendEquationEXT and vkCmdSetColorBlendAdvancedEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-07636
  If the VK_EXT_provoking_vertex extension is enabled, a shader object is bound to the VK_SHADER_STAGE_VERTEX_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_PROVOKING_VERTEX_MODE_EXT dynamic state enabled, and the current

value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetProvokingVertexModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-07637
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT dynamic state enabled then vkCmdSetLineRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-08666
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetPolygonModeEXT in the current command buffer set polygonMode to VK_POLYGON_MODE_LINE, then vkCmdSetLineRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-08667
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object is bound to the VK_SHADER_STAGE_VERTEX_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetPrimitiveTopologyEXT in the current command buffer set primitiveTopology to any line topology, then vkCmdSetLineRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-08668
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object that outputs line primitives is bound to the VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT or VK_SHADER_STAGE_GEOMETRY_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetLineRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-07638
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT dynamic state enabled then vkCmdSetLineStippleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-08669
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetPolygonModeEXT in the current command buffer set polygonMode to

VK_POLYGON_MODE_LINE, then vkCmdSetLineStippleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-08670
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object is bound to the VK_SHADER_STAGE_VERTEX_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetPrimitiveTopologyEXT in the current command buffer set primitiveTopology to any line topology, then vkCmdSetLineStippleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-08671
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object that outputs line primitives is bound to the VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT or VK_SHADER_STAGE_GEOMETRY_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetLineStippleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-07849
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_KHR dynamic state enabled then vkCmdSetLineStippleKHR **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-08672
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetLineStippleEnableEXT in the current command buffer set stippledLineEnable to VK_TRUE, then vkCmdSetLineStippleEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-07639
  If the depthClipControl feature is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_CLIP_NEGATIVE_ONE_TO_ONE_EXT dynamic state enabled, then vkCmdSetDepthClipNegativeOneToOneEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-07640
  If the VK_NV_clip_space_w_scaling extension is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_VIEWPORT_W_SCALING_ENABLE_NV dynamic state enabled, then vkCmdSetViewportWScalingEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-07641

If the VK_NV_viewport_swizzle extension is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_VIEWPORT_SWIZZLE_NV dynamic state enabled, then vkCmdSetViewportSwizzleNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-07642
  If the VK_NV_fragment_coverage_to_color extension is enabled, a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_TO_COLOR_ENABLE_NV dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetCoverageToColorEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-07643
  If the VK_NV_fragment_coverage_to_color extension is enabled, a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_TO_COLOR_LOCATION_NV dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of coverageToColorEnable is VK_TRUE, then vkCmdSetCoverageToColorLocationNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-07644
  If the VK_NV_framebuffer_mixed_samples extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_MODULATION_MODE_NV dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetCoverageModulationModeNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-07645
  If the VK_NV_framebuffer_mixed_samples extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_MODULATION_TABLE_ENABLE_NV dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of coverageModulationMode is any value other than VK_COVERAGE_MODULATION_MODE_NONE_NV, then vkCmdSetCoverageModulationTableEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-07646
  If the VK_NV_framebuffer_mixed_samples extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_MODULATION_TABLE_NV dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of coverageModulationTableEnable is VK_TRUE, then vkCmdSetCoverageModulationTableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-07647
  If the shadingRateImage feature is enabled, a shader object is bound to any graphics stage

or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_SHADING_RATE_IMAGE_ENABLE_NV dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetShadingRateImageEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectCountNV-pipelineFragmentShadingRate-09238
  If the pipelineFragmentShadingRate feature is enabled, a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetFragmentShadingRateKHR **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-07648
  If the representativeFragmentTest feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_REPRESENTATIVE_FRAGMENT_TEST_ENABLE_NV dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetRepresentativeFragmentTestEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-07649
  If the coverageReductionMode feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_REDUCTION_MODE_NV dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetCoverageReductionModeNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountNV-pColorBlendEnables-07470
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT state enabled and the last call to vkCmdSetColorBlendEnableEXT set pColorBlendEnables for any attachment to VK_TRUE, then for those attachments in the subpass the corresponding image view's format features **must** contain VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT
- VUID-vkCmdDrawMeshTasksIndirectCountNV-rasterizationSamples-07471
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, and the current subpass does not use any color and/or depth/stencil attachments, then the rasterizationSamples in the last call to vkCmdSetRasterizationSamplesEXT **must** follow the rules for a zero-attachment subpass
- VUID-vkCmdDrawMeshTasksIndirectCountNV-samples-07472
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_MASK_EXT state enabled and the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state disabled, then the samples parameter in the last call to vkCmdSetSampleMaskEXT **must** be greater or equal to the VkPipelineMultisampleStateCreateInfo::rasterizationSamples parameter used to create the bound graphics pipeline

- VUID-vkCmdDrawMeshTasksIndirectCountNV-samples-07473
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_SAMPLE_MASK_EXT` state and `VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT` states enabled, then the `samples` parameter in the last call to vkCmdSetSampleMaskEXT **must** be greater or equal to the `rasterizationSamples` parameter in the last call to vkCmdSetRasterizationSamplesEXT
- VUID-vkCmdDrawMeshTasksIndirectCountNV-rasterizationSamples-07474
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT` state enabled, and neither the `VK_AMD_mixed_attachment_samples` nor the `VK_NV_framebuffer_mixed_samples` extensions are enabled, then the `rasterizationSamples` in the last call to vkCmdSetRasterizationSamplesEXT **must** be the same as the current subpass color and/or depth/stencil attachments
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-09211
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT` state enabled, or a shader object is bound to any graphics stage, and the current render pass instance includes a VkMultisampledRenderToSingleSampledInfoEXT structure with `multisampledRenderToSingleSampledEnable` equal to `VK_TRUE`, then the `rasterizationSamples` in the last call to vkCmdSetRasterizationSamplesEXT **must** be the same as the `rasterizationSamples` member of that structure
- VUID-vkCmdDrawMeshTasksIndirectCountNV-firstAttachment-07476
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT` dynamic state enabled then vkCmdSetColorBlendEnableEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the `firstAttachment` and `attachmentCount` parameters of `vkCmdSetColorBlendEnableEXT` calls **must** specify an enable for all active color attachments in the current subpass
- VUID-vkCmdDrawMeshTasksIndirectCountNV-rasterizerDiscardEnable-09417
  If a shader object is bound to the `VK_SHADER_STAGE_FRAGMENT_BIT` stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE`, then vkCmdSetColorBlendEnableEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the `firstAttachment` and `attachmentCount` parameters of `vkCmdSetColorBlendEnableEXT` calls **must** specify an enable for all active color attachments in the current subpass
- VUID-vkCmdDrawMeshTasksIndirectCountNV-firstAttachment-07477
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_COLOR_BLEND_EQUATION_EXT` dynamic state enabled then vkCmdSetColorBlendEquationEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the `firstAttachment` and `attachmentCount` parameters of `vkCmdSetColorBlendEquationEXT` calls **must** specify the blend equations for all active color attachments in the current subpass where blending is enabled
- VUID-vkCmdDrawMeshTasksIndirectCountNV-rasterizerDiscardEnable-09418

If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and both the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE and there are color attachments bound, then vkCmdSetColorBlendEquationEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the firstAttachment and attachmentCount parameters of vkCmdSetColorBlendEquationEXT calls **must** specify the blend equations for all active color attachments in the current subpass where blending is enabled

- VUID-vkCmdDrawMeshTasksIndirectCountNV-firstAttachment-07478
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_WRITE_MASK_EXT dynamic state enabled then vkCmdSetColorWriteMaskEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the firstAttachment and attachmentCount parameters of vkCmdSetColorWriteMaskEXT calls **must** specify the color write mask for all active color attachments in the current subpass
- VUID-vkCmdDrawMeshTasksIndirectCountNV-rasterizerDiscardEnable-09419
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetColorWriteMaskEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the firstAttachment and attachmentCount parameters of vkCmdSetColorWriteMaskEXT calls **must** specify the color write mask for all active color attachments in the current subpass
- VUID-vkCmdDrawMeshTasksIndirectCountNV-firstAttachment-07479
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_ADVANCED_EXT dynamic state enabled then vkCmdSetColorBlendAdvancedEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the firstAttachment and attachmentCount parameters of vkCmdSetColorBlendAdvancedEXT calls **must** specify the advanced blend equations for all active color attachments in the current subpass where blending is enabled
- VUID-vkCmdDrawMeshTasksIndirectCountNV-advancedBlendMaxColorAttachments-07480
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_ADVANCED_EXT and VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT dynamic states enabled and the last calls to vkCmdSetColorBlendEnableEXT and vkCmdSetColorBlendAdvancedEXT have enabled advanced blending, then the number of active color attachments in the current subpass **must** not exceed advancedBlendMaxColorAttachments
- VUID-vkCmdDrawMeshTasksIndirectCountNV-primitivesGeneratedQueryWithNonZeroStreams-07481
  If the primitivesGeneratedQueryWithNonZeroStreams feature is not enabled and the VK_QUERY_TYPE_PRIMITIVES_GENERATED_EXT query is active, and the bound graphics pipeline was created with VK_DYNAMIC_STATE_RASTERIZATION_STREAM_EXT state enabled, the last call to vkCmdSetRasterizationStreamEXT **must** have set the rasterizationStream to zero

- VUID-vkCmdDrawMeshTasksIndirectCountNV-sampleLocationsPerPixel-07482
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT` state enabled and the `VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT` state disabled, then the `sampleLocationsPerPixel` member of `pSampleLocationsInfo` in the last call to vkCmdSetSampleLocationsEXT **must** equal the `rasterizationSamples` member of the VkPipelineMultisampleStateCreateInfo structure the bound graphics pipeline has been created with
- VUID-vkCmdDrawMeshTasksIndirectCountNV-sampleLocationsPerPixel-07483
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT` state enabled and the `VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT` state enabled, then the `sampleLocationsPerPixel` member of `pSampleLocationsInfo` in the last call to vkCmdSetSampleLocationsEXT **must** equal the `rasterizationSamples` parameter of the last call to vkCmdSetRasterizationSamplesEXT
- VUID-vkCmdDrawMeshTasksIndirectCountNV-sampleLocationsEnable-07484
  If a shader object is bound to the `VK_SHADER_STAGE_FRAGMENT_BIT` stage, or the bound graphics pipeline was created with the `VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT` state enabled, and `sampleLocationsEnable` was `VK_TRUE` in the last call to vkCmdSetSampleLocationsEnableEXT, and the current subpass has a depth/stencil attachment, then that attachment **must** have been created with the `VK_IMAGE_CREATE_SAMPLE_LOCATIONS_COMPATIBLE_DEPTH_BIT_EXT` bit set
- VUID-vkCmdDrawMeshTasksIndirectCountNV-sampleLocationsEnable-07485
  If a shader object is bound to the `VK_SHADER_STAGE_FRAGMENT_BIT` stage, or the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT` state enabled and the `VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT` state enabled, and if `sampleLocationsEnable` was `VK_TRUE` in the last call to vkCmdSetSampleLocationsEnableEXT, then the `sampleLocationsInfo.sampleLocationGridSize.width` in the last call to vkCmdSetSampleLocationsEXT **must** evenly divide VkMultisamplePropertiesEXT::`sampleLocationGridSize.width` as returned by vkGetPhysicalDeviceMultisamplePropertiesEXT with a `samples` parameter equaling `rasterizationSamples`
- VUID-vkCmdDrawMeshTasksIndirectCountNV-sampleLocationsEnable-07486
  If a shader object is bound to the `VK_SHADER_STAGE_FRAGMENT_BIT` stage, or the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT` state enabled and the `VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT` state enabled, and if `sampleLocationsEnable` was `VK_TRUE` in the last call to vkCmdSetSampleLocationsEnableEXT, then the `sampleLocationsInfo.sampleLocationGridSize.height` in the last call to vkCmdSetSampleLocationsEXT **must** evenly divide VkMultisamplePropertiesEXT::`sampleLocationGridSize.height` as returned by vkGetPhysicalDeviceMultisamplePropertiesEXT with a `samples` parameter equaling `rasterizationSamples`
- VUID-vkCmdDrawMeshTasksIndirectCountNV-sampleLocationsEnable-07487
  If a shader object is bound to the `VK_SHADER_STAGE_FRAGMENT_BIT` stage, or the bound

graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, and if sampleLocationsEnable was VK_TRUE in the last call to vkCmdSetSampleLocationsEnableEXT, the fragment shader code **must** not statically use the extended instruction InterpolateAtSample

- VUID-vkCmdDrawMeshTasksIndirectCountNV-sampleLocationsEnable-07936
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state disabled and the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, the sampleLocationsEnable member of a VkPipelineSampleLocationsStateCreateInfoEXT::sampleLocationsEnable in the bound graphics pipeline is VK_TRUE or VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, then, sampleLocationsInfo.sampleLocationGridSize.width **must** evenly divide VkMultisamplePropertiesEXT::sampleLocationGridSize.width as returned by vkGetPhysicalDeviceMultisamplePropertiesEXT with a samples parameter equaling the value of rasterizationSamples in the last call to vkCmdSetRasterizationSamplesEXT

- VUID-vkCmdDrawMeshTasksIndirectCountNV-sampleLocationsEnable-07937
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state disabled and the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, the sampleLocationsEnable member of a VkPipelineSampleLocationsStateCreateInfoEXT::sampleLocationsEnable in the bound graphics pipeline is VK_TRUE or VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, then, sampleLocationsInfo.sampleLocationGridSize.height **must** evenly divide VkMultisamplePropertiesEXT::sampleLocationGridSize.height as returned by vkGetPhysicalDeviceMultisamplePropertiesEXT with a samples parameter equaling the value of rasterizationSamples in the last call to vkCmdSetRasterizationSamplesEXT

- VUID-vkCmdDrawMeshTasksIndirectCountNV-sampleLocationsEnable-07938
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state disabled and the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, the sampleLocationsEnable member of a VkPipelineSampleLocationsStateCreateInfoEXT::sampleLocationsEnable in the bound graphics pipeline is VK_TRUE or VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, then, sampleLocationsInfo.sampleLocationsPerPixel **must** equal rasterizationSamples in the last call to vkCmdSetRasterizationSamplesEXT

- VUID-vkCmdDrawMeshTasksIndirectCountNV-coverageModulationTableEnable-07488
  If a shader object is bound to any graphics stage or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COVERAGE_MODULATION_TABLE_ENABLE_NV state enabled, and the last call to vkCmdSetCoverageModulationTableEnableNV set coverageModulationTableEnable to VK_TRUE, then the coverageModulationTableCount parameter in the last call to vkCmdSetCoverageModulationTableNV **must** equal the current rasterizationSamples divided by the number of color samples in the current subpass

- VUID-vkCmdDrawMeshTasksIndirectCountNV-rasterizationSamples-07489
  If the VK_NV_framebuffer_mixed_samples extension is enabled, and if current subpass has a depth/stencil attachment and depth test, stencil test, or depth bounds test are enabled in the currently bound pipeline state, then the current rasterizationSamples **must** be the same as the sample count of the depth/stencil attachment

- VUID-vkCmdDrawMeshTasksIndirectCountNV-coverageToColorEnable-07490
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COVERAGE_TO_COLOR_ENABLE_NV state enabled and the last call to vkCmdSetCoverageToColorEnableNV set the coverageToColorEnable to VK_TRUE, then the current subpass **must** have a color attachment at the location selected by the last call to vkCmdSetCoverageToColorLocationNV coverageToColorLocation, with a VkFormat of VK_FORMAT_R8_UINT, VK_FORMAT_R8_SINT, VK_FORMAT_R16_UINT, VK_FORMAT_R16_SINT, VK_FORMAT_R32_UINT, or VK_FORMAT_R32_SINT
- VUID-vkCmdDrawMeshTasksIndirectCountNV-rasterizerDiscardEnable-09420
  If the VK_NV_fragment_coverage_to_color extension is enabled, and a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the last call to vkCmdSetCoverageToColorEnableNV set the coverageToColorEnable to VK_TRUE, then the current subpass **must** have a color attachment at the location selected by the last call to vkCmdSetCoverageToColorLocationNV coverageToColorLocation, with a VkFormat of VK_FORMAT_R8_UINT, VK_FORMAT_R8_SINT, VK_FORMAT_R16_UINT, VK_FORMAT_R16_SINT, VK_FORMAT_R32_UINT, or VK_FORMAT_R32_SINT
- VUID-vkCmdDrawMeshTasksIndirectCountNV-coverageReductionMode-07491
  If this VK_NV_coverage_reduction_mode extension is enabled, the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COVERAGE_TO_COLOR_ENABLE_NV and VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT states enabled, the current coverage reduction mode coverageReductionMode, then the current rasterizationSamples, and the sample counts for the color and depth/stencil attachments (if the subpass has them) **must** be a valid combination returned by vkGetPhysicalDeviceSupportedFramebufferMixedSamplesCombinationsNV
- VUID-vkCmdDrawMeshTasksIndirectCountNV-viewportCount-07492
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, but not the VK_DYNAMIC_STATE_VIEWPORT_SWIZZLE_NV dynamic state enabled, then the bound graphics pipeline **must** have been created with VkPipelineViewportSwizzleStateCreateInfoNV ::viewportCount greater or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawMeshTasksIndirectCountNV-viewportCount-07493
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT and VK_DYNAMIC_STATE_VIEWPORT_SWIZZLE_NV dynamic states enabled then the viewportCount parameter in the last call to vkCmdSetViewportSwizzleNV **must** be greater than or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawMeshTasksIndirectCountNV-viewportCount-09421
  If the VK_NV_viewport_swizzle extension is enabled, and a shader object is bound to any graphics stage, then the viewportCount parameter in the last call to vkCmdSetViewportSwizzleNV **must** be greater than or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawMeshTasksIndirectCountNV-rasterizationSamples-07494

If the VK_NV_framebuffer_mixed_samples extension is enabled, and if the current subpass has any color attachments and rasterizationSamples of the last call to vkCmdSetRasterizationSamplesEXT is greater than the number of color samples, then the pipeline sampleShadingEnable **must** be VK_FALSE

- VUID-vkCmdDrawMeshTasksIndirectCountNV-stippledLineEnable-07495
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT or VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT dynamic states enabled, and if the current stippledLineEnable state is VK_TRUE and the current lineRasterizationMode state is VK_LINE_RASTERIZATION_MODE_RECTANGULAR_KHR, then the stippledRectangularLines feature **must** be enabled

- VUID-vkCmdDrawMeshTasksIndirectCountNV-stippledLineEnable-07496
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT or VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT dynamic states enabled, and if the current stippledLineEnable state is VK_TRUE and the current lineRasterizationMode state is VK_LINE_RASTERIZATION_MODE_BRESENHAM_KHR, then the stippledBresenhamLines feature **must** be enabled

- VUID-vkCmdDrawMeshTasksIndirectCountNV-stippledLineEnable-07497
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT or VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT dynamic states enabled, and if the current stippledLineEnable state is VK_TRUE and the current lineRasterizationMode state is VK_LINE_RASTERIZATION_MODE_RECTANGULAR_SMOOTH_KHR, then the stippledSmoothLines feature **must** be enabled

- VUID-vkCmdDrawMeshTasksIndirectCountNV-stippledLineEnable-07498
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT or VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT dynamic states enabled, and if the current stippledLineEnable state is VK_TRUE and the current lineRasterizationMode state is VK_LINE_RASTERIZATION_MODE_DEFAULT_KHR, then the stippledRectangularLines feature **must** be enabled and VkPhysicalDeviceLimits::strictLines **must** be VK_TRUE

- VUID-vkCmdDrawMeshTasksIndirectCountNV-conservativePointAndLineRasterization-07499
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_CONSERVATIVE_RASTERIZATION_MODE_EXT dynamic state enabled, conservativePointAndLineRasterization is not supported, and the effective primitive topology output by the last pre-rasterization shader stage is a line or point, then the conservativeRasterizationMode set by the last call to vkCmdSetConservativeRasterizationModeEXT **must** be VK_CONSERVATIVE_RASTERIZATION_MODE_DISABLED_EXT

- VUID-vkCmdDrawMeshTasksIndirectCountNV-stage-07073
  If the currently bound pipeline was created with the VkPipelineShaderStageCreateInfo::stage member of an element of VkGraphicsPipelineCreateInfo::pStages set to VK_SHADER_STAGE_VERTEX_BIT, VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT,

VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT or VK_SHADER_STAGE_GEOMETRY_BIT, then Mesh Shader Queries **must** not be active

- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-08877
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_ATTACHMENT_FEEDBACK_LOOP_ENABLE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetAttachmentFeedbackLoopEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-07850
  If dynamic state was inherited from VkCommandBufferInheritanceViewportScissorInfoNV, it **must** be set in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-08684
  If there is no bound graphics pipeline, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_VERTEX_BIT
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-08685
  If there is no bound graphics pipeline, and the tessellationShader feature is enabled, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-08686
  If there is no bound graphics pipeline, and the tessellationShader feature is enabled, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-08687
  If there is no bound graphics pipeline, and the geometryShader feature is enabled, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_GEOMETRY_BIT
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-08688
  If there is no bound graphics pipeline, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_FRAGMENT_BIT
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-08689
  If there is no bound graphics pipeline, and the taskShader feature is enabled, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_TASK_BIT_EXT
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-08690
  If there is no bound graphics pipeline, and the meshShader feature is enabled, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_MESH_BIT_EXT
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-08693
  If there is no bound graphics pipeline, and at least one of the taskShader and meshShader features is enabled, one of the VK_SHADER_STAGE_VERTEX_BIT or VK_SHADER_STAGE_MESH_BIT_EXT stages **must** have a valid VkShaderEXT bound, and the other **must** have no VkShaderEXT bound

- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-08694
  If there is no bound graphics pipeline, and both the `taskShader` and `meshShader` features are enabled, and a valid `VkShaderEXT` is bound the to the `VK_SHADER_STAGE_MESH_BIT_EXT` stage, and that `VkShaderEXT` was created without the `VK_SHADER_CREATE_NO_TASK_SHADER_BIT_EXT` flag, a valid `VkShaderEXT` **must** be bound to the `VK_SHADER_STAGE_TASK_BIT_EXT` stage
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-08695
  If there is no bound graphics pipeline, and both the `taskShader` and `meshShader` features are enabled, and a valid `VkShaderEXT` is bound the to the `VK_SHADER_STAGE_MESH_BIT_EXT` stage, and that `VkShaderEXT` was created with the `VK_SHADER_CREATE_NO_TASK_SHADER_BIT_EXT` flag, there **must** be no `VkShaderEXT` bound to the `VK_SHADER_STAGE_TASK_BIT_EXT` stage
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-08696
  If there is no bound graphics pipeline, and a valid `VkShaderEXT` is bound to the `VK_SHADER_STAGE_VERTEX_BIT` stage, there **must** be no `VkShaderEXT` bound to either the `VK_SHADER_STAGE_TASK_BIT_EXT` stage or the `VK_SHADER_STAGE_MESH_BIT_EXT` stage
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-08698
  If any graphics shader is bound which was created with the `VK_SHADER_CREATE_LINK_STAGE_BIT_EXT` flag, then all shaders created with the `VK_SHADER_CREATE_LINK_STAGE_BIT_EXT` flag in the same vkCreateShadersEXT call **must** also be bound
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-08699
  If any graphics shader is bound which was created with the `VK_SHADER_CREATE_LINK_STAGE_BIT_EXT` flag, any stages in between stages whose shaders which did not create a shader with the `VK_SHADER_CREATE_LINK_STAGE_BIT_EXT` flag as part of the same vkCreateShadersEXT call **must** not have any `VkShaderEXT` bound
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-08878
  All bound graphics shader objects **must** have been created with identical or identically defined push constant ranges
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-08879
  All bound graphics shader objects **must** have been created with identical or identically defined arrays of descriptor set layouts
- VUID-vkCmdDrawMeshTasksIndirectCountNV-colorAttachmentCount-09372
  If the current render pass instance was begun with vkCmdBeginRenderingKHR and a VkRenderingInfo::`colorAttachmentCount` equal to `1`, a color attachment with a resolve mode of `VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID`, and a fragment shader is bound, it **must** not declare the `DepthReplacing` or `StencilRefReplacingEXT` execution modes
- VUID-vkCmdDrawMeshTasksIndirectCountNV-pDynamicStates-08715
  If the bound graphics pipeline state includes a fragment shader stage, was created with `VK_DYNAMIC_STATE_DEPTH_WRITE_ENABLE` set in VkPipelineDynamicStateCreateInfo::`pDynamicStates`, and the fragment shader declares the `EarlyFragmentTests` execution mode and uses `OpDepthAttachmentReadEXT`, the `depthWriteEnable` parameter in the last call to vkCmdSetDepthWriteEnableEXT **must** be `VK_FALSE`
- VUID-vkCmdDrawMeshTasksIndirectCountNV-pDynamicStates-08716
  If the bound graphics pipeline state includes a fragment shader stage, was created with

VK_DYNAMIC_STATE_STENCIL_WRITE_MASK set in VkPipelineDynamicStateCreateInfo::pDynamicStates, and the fragment shader declares the EarlyFragmentTests execution mode and uses OpStencilAttachmentReadEXT, the writeMask parameter in the last call to vkCmdSetStencilWriteMask **must** be 0

- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-09116
  If a shader object is bound to any graphics stage or the currently bound graphics pipeline was created with VK_DYNAMIC_STATE_COLOR_WRITE_MASK_EXT, and the format of any color attachment is VK_FORMAT_E5B9G9R9_UFLOAT_PACK32, the corresponding element of the pColorWriteMasks parameter of vkCmdSetColorWriteMaskEXT **must** either include all of VK_COLOR_COMPONENT_R_BIT, VK_COLOR_COMPONENT_G_BIT, and VK_COLOR_COMPONENT_B_BIT, or none of them
- VUID-vkCmdDrawMeshTasksIndirectCountNV-maxFragmentDualSrcAttachments-09239
  If blending is enabled for any attachment where either the source or destination blend factors for that attachment use the secondary color input, the maximum value of Location for any output attachment statically used in the Fragment Execution Model executed by this command **must** be less than maxFragmentDualSrcAttachments
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-09548
  If the current render pass was begun with vkCmdBeginRenderingKHR, and there is no shader object bound to any graphics stage, the value of each element of VkRenderingAttachmentLocationInfoKHR::pColorAttachmentLocations set by vkCmdSetRenderingAttachmentLocationsKHR **must** match the value set for the corresponding element in the currently bound pipeline
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-09549
  If the current render pass was begun with vkCmdBeginRenderingKHR, and there is no shader object bound to any graphics stage, input attachment index mappings in the currently bound pipeline **must** match those set for the current render pass instance via VkRenderingInputAttachmentIndexInfoKHR
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-09642
  If the current render pass was begun with vkCmdBeginRenderingKHR with the VK_RENDERING_ENABLE_LEGACY_DITHERING_BIT_EXT flag, the bound graphics pipeline **must** have been created with VK_PIPELINE_CREATE_2_ENABLE_LEGACY_DITHERING_BIT_EXT
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-09643
  If the bound graphics pipeline was created with VK_PIPELINE_CREATE_2_ENABLE_LEGACY_DITHERING_BIT_EXT, the current render pass **must** have begun with vkCmdBeginRenderingKHR with the VK_RENDERING_ENABLE_LEGACY_DITHERING_BIT_EXT flag
- VUID-vkCmdDrawMeshTasksIndirectCountNV-stage-06480
  The bound graphics pipeline **must** not have been created with the VkPipelineShaderStageCreateInfo::stage member of an element of VkGraphicsPipelineCreateInfo::pStages set to VK_SHADER_STAGE_VERTEX_BIT, VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT, VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT or VK_SHADER_STAGE_GEOMETRY_BIT
- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-07074
  Transform Feedback Queries **must** not be active

- VUID-vkCmdDrawMeshTasksIndirectCountNV-None-07075
  Primitives Generated Queries **must** not be active
- VUID-vkCmdDrawMeshTasksIndirectCountNV-pipelineStatistics-07076
  The `pipelineStatistics` member used to create any active Pipeline Statistics Query **must** not contain `VK_QUERY_PIPELINE_STATISTIC_INPUT_ASSEMBLY_VERTICES_BIT`, `VK_QUERY_PIPELINE_STATISTIC_INPUT_ASSEMBLY_PRIMITIVES_BIT`, `VK_QUERY_PIPELINE_STATISTIC_VERTEX_SHADER_INVOCATIONS_BIT`, `VK_QUERY_PIPELINE_STATISTIC_GEOMETRY_SHADER_INVOCATIONS_BIT`, `VK_QUERY_PIPELINE_STATISTIC_GEOMETRY_SHADER_PRIMITIVES_BIT`, `VK_QUERY_PIPELINE_STATISTIC_CLIPPING_INVOCATIONS_BIT`, `VK_QUERY_PIPELINE_STATISTIC_CLIPPING_PRIMITIVES_BIT`, `VK_QUERY_PIPELINE_STATISTIC_TESSELLATION_CONTROL_SHADER_PATCHES_BIT`, or `VK_QUERY_PIPELINE_STATISTIC_TESSELLATION_EVALUATION_SHADER_INVOCATIONS_BIT`
- VUID-vkCmdDrawMeshTasksIndirectCountNV-buffer-02708
  If `buffer` is non-sparse then it **must** be bound completely and contiguously to a single `VkDeviceMemory` object
- VUID-vkCmdDrawMeshTasksIndirectCountNV-buffer-02709
  `buffer` **must** have been created with the `VK_BUFFER_USAGE_INDIRECT_BUFFER_BIT` bit set
- VUID-vkCmdDrawMeshTasksIndirectCountNV-offset-02710
  `offset` **must** be a multiple of `4`
- VUID-vkCmdDrawMeshTasksIndirectCountNV-commandBuffer-02711
  `commandBuffer` **must** not be a protected command buffer
- VUID-vkCmdDrawMeshTasksIndirectCountNV-countBuffer-02714
  If `countBuffer` is non-sparse then it **must** be bound completely and contiguously to a single `VkDeviceMemory` object
- VUID-vkCmdDrawMeshTasksIndirectCountNV-countBuffer-02715
  `countBuffer` **must** have been created with the `VK_BUFFER_USAGE_INDIRECT_BUFFER_BIT` bit set
- VUID-vkCmdDrawMeshTasksIndirectCountNV-countBufferOffset-02716
  `countBufferOffset` **must** be a multiple of `4`
- VUID-vkCmdDrawMeshTasksIndirectCountNV-countBuffer-02717
  The count stored in `countBuffer` **must** be less than or equal to `VkPhysicalDeviceLimits::maxDrawIndirectCount`
- VUID-vkCmdDrawMeshTasksIndirectCountNV-countBufferOffset-04129
  (`countBufferOffset` + `sizeof`(uint32_t)) **must** be less than or equal to the size of `countBuffer`
- VUID-vkCmdDrawMeshTasksIndirectCountNV-stride-02182
  `stride` **must** be a multiple of `4` and **must** be greater than or equal to `sizeof`(`VkDrawMeshTasksIndirectCommandNV`)
- VUID-vkCmdDrawMeshTasksIndirectCountNV-maxDrawCount-02183
  If `maxDrawCount` is greater than or equal to `1`, (`stride` × (`maxDrawCount` - 1) + `offset` + `sizeof`(`VkDrawMeshTasksIndirectCommandNV`)) **must** be less than or equal to the size of `buffer`
- VUID-vkCmdDrawMeshTasksIndirectCountNV-countBuffer-02191
  If the count stored in `countBuffer` is equal to `1`, (`offset` + `sizeof`

(VkDrawMeshTasksIndirectCommandNV)) **must** be less than or equal to the size of buffer

- VUID-vkCmdDrawMeshTasksIndirectCountNV-countBuffer-02192
  If the count stored in countBuffer is greater than 1, (stride × (drawCount - 1) + offset + sizeof(VkDrawMeshTasksIndirectCommandNV)) **must** be less than or equal to the size of buffer
- VUID-vkCmdDrawMeshTasksIndirectCountNV-MeshNV-07082
  The current pipeline bound to VK_PIPELINE_BIND_POINT_GRAPHICS **must** contain a shader stage using the MeshNV Execution Model

## Valid Usage (Implicit)

- VUID-vkCmdDrawMeshTasksIndirectCountNV-commandBuffer-parameter
  commandBuffer **must** be a valid VkCommandBuffer handle
- VUID-vkCmdDrawMeshTasksIndirectCountNV-buffer-parameter
  buffer **must** be a valid VkBuffer handle
- VUID-vkCmdDrawMeshTasksIndirectCountNV-countBuffer-parameter
  countBuffer **must** be a valid VkBuffer handle
- VUID-vkCmdDrawMeshTasksIndirectCountNV-commandBuffer-recording
  commandBuffer **must** be in the recording state
- VUID-vkCmdDrawMeshTasksIndirectCountNV-commandBuffer-cmdpool
  The VkCommandPool that commandBuffer was allocated from **must** support graphics operations
- VUID-vkCmdDrawMeshTasksIndirectCountNV-renderpass
  This command **must** only be called inside of a render pass instance
- VUID-vkCmdDrawMeshTasksIndirectCountNV-videocoding
  This command **must** only be called outside of a video coding scope
- VUID-vkCmdDrawMeshTasksIndirectCountNV-commonparent
  Each of buffer, commandBuffer, and countBuffer **must** have been created, allocated, or retrieved from the same VkDevice

## Host Synchronization

- Host access to commandBuffer **must** be externally synchronized
- Host access to the VkCommandPool that commandBuffer was allocated from **must** be externally synchronized

### Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Inside | Outside | Graphics | Action |

To record a mesh tasks drawing command, call:

```
// Provided by VK_EXT_mesh_shader
void vkCmdDrawMeshTasksEXT(
    VkCommandBuffer                                 commandBuffer,
    uint32_t                                        groupCountX,
    uint32_t                                        groupCountY,
    uint32_t                                        groupCountZ);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `groupCountX` is the number of local workgroups to dispatch in the X dimension.
- `groupCountY` is the number of local workgroups to dispatch in the Y dimension.
- `groupCountZ` is the number of local workgroups to dispatch in the Z dimension.

When the command is executed, a global workgroup consisting of `groupCountX` × `groupCountY` × `groupCountZ` local workgroups is assembled.

### Valid Usage

- VUID-vkCmdDrawMeshTasksEXT-magFilter-04553
  If a VkSampler created with `magFilter` or `minFilter` equal to `VK_FILTER_LINEAR`, `reductionMode` equal to `VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE`, and `compareEnable` equal to `VK_FALSE` is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT`
- VUID-vkCmdDrawMeshTasksEXT-magFilter-09598
  If a VkSampler created with `magFilter` or `minFilter` equal to `VK_FILTER_LINEAR` and `reductionMode` equal to either `VK_SAMPLER_REDUCTION_MODE_MIN` or `VK_SAMPLER_REDUCTION_MODE_MAX` is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT`
- VUID-vkCmdDrawMeshTasksEXT-mipmapMode-04770
  If a VkSampler created with `mipmapMode` equal to `VK_SAMPLER_MIPMAP_MODE_LINEAR`, `reductionMode` equal to `VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE`, and `compareEnable` equal to `VK_FALSE` is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain

VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT

- VUID-vkCmdDrawMeshTasksEXT-mipmapMode-09599
  If a VkSampler created with mipmapMode equal to VK_SAMPLER_MIPMAP_MODE_LINEAR and reductionMode equal to either VK_SAMPLER_REDUCTION_MODE_MIN or VK_SAMPLER_REDUCTION_MODE_MAX is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT
- VUID-vkCmdDrawMeshTasksEXT-unnormalizedCoordinates-09635
  If a VkSampler created with unnormalizedCoordinates equal to VK_TRUE is used to sample a VkImageView as a result of this command, then the image view's levelCount and layerCount **must** be 1
- VUID-vkCmdDrawMeshTasksEXT-unnormalizedCoordinates-09636
  If a VkSampler created with unnormalizedCoordinates equal to VK_TRUE is used to sample a VkImageView as a result of this command, then the image view's viewType **must** be VK_IMAGE_VIEW_TYPE_1D or VK_IMAGE_VIEW_TYPE_2D
- VUID-vkCmdDrawMeshTasksEXT-None-06479
  If a VkImageView is sampled with depth comparison, the image view's format features **must** contain VK_FORMAT_FEATURE_2_SAMPLED_IMAGE_DEPTH_COMPARISON_BIT
- VUID-vkCmdDrawMeshTasksEXT-None-02691
  If a VkImageView is accessed using atomic operations as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_STORAGE_IMAGE_ATOMIC_BIT
- VUID-vkCmdDrawMeshTasksEXT-None-07888
  If a VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER descriptor is accessed using atomic operations as a result of this command, then the storage texel buffer's format features **must** contain VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_ATOMIC_BIT
- VUID-vkCmdDrawMeshTasksEXT-None-02692
  If a VkImageView is sampled with VK_FILTER_CUBIC_EXT as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_CUBIC_BIT_EXT
- VUID-vkCmdDrawMeshTasksEXT-None-02693
  If the VK_EXT_filter_cubic extension is not enabled and any VkImageView is sampled with VK_FILTER_CUBIC_EXT as a result of this command, it **must** not have a VkImageViewType of VK_IMAGE_VIEW_TYPE_3D, VK_IMAGE_VIEW_TYPE_CUBE, or VK_IMAGE_VIEW_TYPE_CUBE_ARRAY
- VUID-vkCmdDrawMeshTasksEXT-filterCubic-02694
  Any VkImageView being sampled with VK_FILTER_CUBIC_EXT as a result of this command **must** have a VkImageViewType and format that supports cubic filtering, as specified by VkFilterCubicImageViewImageFormatPropertiesEXT::filterCubic returned by vkGetPhysicalDeviceImageFormatProperties2
- VUID-vkCmdDrawMeshTasksEXT-filterCubicMinmax-02695
  Any VkImageView being sampled with VK_FILTER_CUBIC_EXT with a reduction mode of either VK_SAMPLER_REDUCTION_MODE_MIN or VK_SAMPLER_REDUCTION_MODE_MAX as a result of this command **must** have a VkImageViewType and format that supports cubic filtering together with minmax filtering, as specified by

VkFilterCubicImageViewImageFormatPropertiesEXT::filterCubicMinmax returned by vkGetPhysicalDeviceImageFormatProperties2

- VUID-vkCmdDrawMeshTasksEXT-cubicRangeClamp-09212
  If the cubicRangeClamp feature is not enabled, then any VkImageView being sampled with VK_FILTER_CUBIC_EXT as a result of this command **must** not have a VkSamplerReductionModeCreateInfo::reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE_RANGECLAMP_QCOM
- VUID-vkCmdDrawMeshTasksEXT-reductionMode-09213
  Any VkImageView being sampled with a VkSamplerReductionModeCreateInfo::reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE_RANGECLAMP_QCOM as a result of this command **must** sample with VK_FILTER_CUBIC_EXT
- VUID-vkCmdDrawMeshTasksEXT-selectableCubicWeights-09214
  If the selectableCubicWeights feature is not enabled, then any VkImageView being sampled with VK_FILTER_CUBIC_EXT as a result of this command **must** have VkSamplerCubicWeightsCreateInfoQCOM::cubicWeights equal to VK_CUBIC_FILTER_WEIGHTS_CATMULL_ROM_QCOM
- VUID-vkCmdDrawMeshTasksEXT-flags-02696
  Any VkImage created with a VkImageCreateInfo::flags containing VK_IMAGE_CREATE_CORNER_SAMPLED_BIT_NV sampled as a result of this command **must** only be sampled using a VkSamplerAddressMode of VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_EDGE
- VUID-vkCmdDrawMeshTasksEXT-OpTypeImage-07027
  For any VkImageView being written as a storage image where the image format field of the OpTypeImage is Unknown, the view's format features **must** contain VK_FORMAT_FEATURE_2_STORAGE_WRITE_WITHOUT_FORMAT_BIT
- VUID-vkCmdDrawMeshTasksEXT-OpTypeImage-07028
  For any VkImageView being read as a storage image where the image format field of the OpTypeImage is Unknown, the view's format features **must** contain VK_FORMAT_FEATURE_2_STORAGE_READ_WITHOUT_FORMAT_BIT
- VUID-vkCmdDrawMeshTasksEXT-OpTypeImage-07029
  For any VkBufferView being written as a storage texel buffer where the image format field of the OpTypeImage is Unknown, the view's buffer features **must** contain VK_FORMAT_FEATURE_2_STORAGE_WRITE_WITHOUT_FORMAT_BIT
- VUID-vkCmdDrawMeshTasksEXT-OpTypeImage-07030
  Any VkBufferView being read as a storage texel buffer where the image format field of the OpTypeImage is Unknown then the view's buffer features **must** contain VK_FORMAT_FEATURE_2_STORAGE_READ_WITHOUT_FORMAT_BIT
- VUID-vkCmdDrawMeshTasksEXT-None-08600
  For each set *n* that is statically used by a bound shader, a descriptor set **must** have been bound to *n* at the same pipeline bind point, with a VkPipelineLayout that is compatible for set *n*, with the VkPipelineLayout used to create the current VkPipeline or the VkDescriptorSetLayout array used to create the current VkShaderEXT , as described in Pipeline Layout Compatibility
- VUID-vkCmdDrawMeshTasksEXT-None-08601
  For each push constant that is statically used by a bound shader, a push constant value

**must** have been set for the same pipeline bind point, with a VkPipelineLayout that is compatible for push constants, with the VkPipelineLayout used to create the current VkPipeline or the VkDescriptorSetLayout array used to create the current VkShaderEXT , as described in Pipeline Layout Compatibility

- VUID-vkCmdDrawMeshTasksEXT-None-10068
  For each array of resources that is used by a bound shader, the indices used to access members of the array **must** be less than the descriptor count for the identified binding in the descriptor sets used by this command
- VUID-vkCmdDrawMeshTasksEXT-maintenance4-08602
  If the `maintenance4` feature is not enabled, then for each push constant that is statically used by a bound shader, a push constant value **must** have been set for the same pipeline bind point, with a VkPipelineLayout that is compatible for push constants, with the VkPipelineLayout used to create the current VkPipeline or the VkDescriptorSetLayout and VkPushConstantRange arrays used to create the current VkShaderEXT , as described in Pipeline Layout Compatibility
- VUID-vkCmdDrawMeshTasksEXT-None-08114
  Descriptors in each bound descriptor set, specified via vkCmdBindDescriptorSets, **must** be valid as described by descriptor validity if they are statically used by the VkPipeline bound to the pipeline bind point used by this command and the bound VkPipeline was not created with `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`
- VUID-vkCmdDrawMeshTasksEXT-None-08115
  If the descriptors used by the VkPipeline bound to the pipeline bind point were specified via vkCmdBindDescriptorSets, the bound VkPipeline **must** have been created without `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`
- VUID-vkCmdDrawMeshTasksEXT-None-08116
  Descriptors in bound descriptor buffers, specified via vkCmdSetDescriptorBufferOffsetsEXT, **must** be valid if they are dynamically used by the VkPipeline bound to the pipeline bind point used by this command and the bound VkPipeline was created with `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`
- VUID-vkCmdDrawMeshTasksEXT-None-08604
  Descriptors in bound descriptor buffers, specified via vkCmdSetDescriptorBufferOffsetsEXT, **must** be valid if they are dynamically used by any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command
- VUID-vkCmdDrawMeshTasksEXT-None-08117
  If the descriptors used by the VkPipeline bound to the pipeline bind point were specified via vkCmdSetDescriptorBufferOffsetsEXT, the bound VkPipeline **must** have been created with `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`
- VUID-vkCmdDrawMeshTasksEXT-None-08119
  If a descriptor is dynamically used with a VkPipeline created with `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`, the descriptor memory **must** be resident
- VUID-vkCmdDrawMeshTasksEXT-None-08605
  If a descriptor is dynamically used with a VkShaderEXT created with a `VkDescriptorSetLayout` that was created with

VK_DESCRIPTOR_SET_LAYOUT_CREATE_DESCRIPTOR_BUFFER_BIT_EXT, the descriptor memory **must** be resident

- VUID-vkCmdDrawMeshTasksEXT-None-08606
  If the shaderObject feature is not enabled, a valid pipeline **must** be bound to the pipeline bind point used by this command
- VUID-vkCmdDrawMeshTasksEXT-None-08608
  If a pipeline is bound to the pipeline bind point used by this command, there **must** not have been any calls to dynamic state setting commands for any state specified statically in the VkPipeline object bound to the pipeline bind point used by this command, since that pipeline was bound
- VUID-vkCmdDrawMeshTasksEXT-None-08609
  If the VkPipeline object bound to the pipeline bind point used by this command or any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a VkSampler object that uses unnormalized coordinates, that sampler **must** not be used to sample from any VkImage with a VkImageView of the type VK_IMAGE_VIEW_TYPE_3D, VK_IMAGE_VIEW_TYPE_CUBE, VK_IMAGE_VIEW_TYPE_1D_ARRAY, VK_IMAGE_VIEW_TYPE_2D_ARRAY or VK_IMAGE_VIEW_TYPE_CUBE_ARRAY, in any shader stage
- VUID-vkCmdDrawMeshTasksEXT-None-08610
  If the VkPipeline object bound to the pipeline bind point used by this command or any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a VkSampler object that uses unnormalized coordinates, that sampler **must** not be used with any of the SPIR-V OpImageSample* or OpImageSparseSample* instructions with ImplicitLod, Dref or Proj in their name, in any shader stage
- VUID-vkCmdDrawMeshTasksEXT-None-08611
  If the VkPipeline object bound to the pipeline bind point used by this command or any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a VkSampler object that uses unnormalized coordinates, that sampler **must** not be used with any of the SPIR-V OpImageSample* or OpImageSparseSample* instructions that includes a LOD bias or any offset values, in any shader stage
- VUID-vkCmdDrawMeshTasksEXT-None-08607
  If the shaderObject is enabled, either a valid pipeline **must** be bound to the pipeline bind point used by this command, or a valid combination of valid and VK_NULL_HANDLE shader objects **must** be bound to every supported shader stage corresponding to the pipeline bind point used by this command
- VUID-vkCmdDrawMeshTasksEXT-uniformBuffers-06935
  If any stage of the VkPipeline object bound to the pipeline bind point used by this command accesses a uniform buffer, and that stage was created without enabling either VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_EXT or VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_2_EXT for uniformBuffers, and the robustBufferAccess feature is not enabled, that stage **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point
- VUID-vkCmdDrawMeshTasksEXT-None-08612
  If the robustBufferAccess feature is not enabled, and any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a uniform

buffer, it **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point

- VUID-vkCmdDrawMeshTasksEXT-storageBuffers-06936
  If any stage of the VkPipeline object bound to the pipeline bind point used by this command accesses a storage buffer, and that stage was created without enabling either `VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_EXT` or `VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_2_EXT` for `storageBuffers`, and the `robustBufferAccess` feature is not enabled, that stage **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point
- VUID-vkCmdDrawMeshTasksEXT-None-08613
  If the `robustBufferAccess` feature is not enabled, and any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a storage buffer, it **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point
- VUID-vkCmdDrawMeshTasksEXT-commandBuffer-02707
  If `commandBuffer` is an unprotected command buffer and `protectedNoFault` is not supported, any resource accessed by bound shaders **must** not be a protected resource
- VUID-vkCmdDrawMeshTasksEXT-None-06550
  If a bound shader accesses a VkSampler or VkImageView object that enables sampler $Y'C_BC_R$ conversion, that object **must** only be used with `OpImageSample*` or `OpImageSparseSample*` instructions
- VUID-vkCmdDrawMeshTasksEXT-ConstOffset-06551
  If a bound shader accesses a VkSampler or VkImageView object that enables sampler $Y'C_BC_R$ conversion, that object **must** not use the `ConstOffset` and `Offset` operands
- VUID-vkCmdDrawMeshTasksEXT-viewType-07752
  If a VkImageView is accessed as a result of this command, then the image view's `viewType` **must** match the `Dim` operand of the `OpTypeImage` as described in Compatibility Between SPIR-V Image Dimensions and Vulkan ImageView Types
- VUID-vkCmdDrawMeshTasksEXT-format-07753
  If a VkImageView is accessed as a result of this command, then the numeric type of the image view's `format` and the `Sampled Type` operand of the `OpTypeImage` **must** match
- VUID-vkCmdDrawMeshTasksEXT-OpImageWrite-08795
  If a VkImageView created with a format other than `VK_FORMAT_A8_UNORM_KHR` is accessed using `OpImageWrite` as a result of this command, then the `Type` of the `Texel` operand of that instruction **must** have at least as many components as the image view's format
- VUID-vkCmdDrawMeshTasksEXT-OpImageWrite-08796
  If a VkImageView created with the format `VK_FORMAT_A8_UNORM_KHR` is accessed using `OpImageWrite` as a result of this command, then the `Type` of the `Texel` operand of that instruction **must** have four components
- VUID-vkCmdDrawMeshTasksEXT-OpImageWrite-04469
  If a VkBufferView is accessed using `OpImageWrite` as a result of this command, then the `Type` of the `Texel` operand of that instruction **must** have at least as many components as the buffer view's format

- VUID-vkCmdDrawMeshTasksEXT-SampledType-04470
  If a VkImageView with a VkFormat that has a 64-bit component width is accessed as a result of this command, the SampledType of the OpTypeImage operand of that instruction **must** have a Width of 64
- VUID-vkCmdDrawMeshTasksEXT-SampledType-04471
  If a VkImageView with a VkFormat that has a component width less than 64-bit is accessed as a result of this command, the SampledType of the OpTypeImage operand of that instruction **must** have a Width of 32
- VUID-vkCmdDrawMeshTasksEXT-SampledType-04472
  If a VkBufferView with a VkFormat that has a 64-bit component width is accessed as a result of this command, the SampledType of the OpTypeImage operand of that instruction **must** have a Width of 64
- VUID-vkCmdDrawMeshTasksEXT-SampledType-04473
  If a VkBufferView with a VkFormat that has a component width less than 64-bit is accessed as a result of this command, the SampledType of the OpTypeImage operand of that instruction **must** have a Width of 32
- VUID-vkCmdDrawMeshTasksEXT-sparseImageInt64Atomics-04474
  If the sparseImageInt64Atomics feature is not enabled, VkImage objects created with the VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT flag **must** not be accessed by atomic instructions through an OpTypeImage with a SampledType with a Width of 64 by this command
- VUID-vkCmdDrawMeshTasksEXT-sparseImageInt64Atomics-04475
  If the sparseImageInt64Atomics feature is not enabled, VkBuffer objects created with the VK_BUFFER_CREATE_SPARSE_RESIDENCY_BIT flag **must** not be accessed by atomic instructions through an OpTypeImage with a SampledType with a Width of 64 by this command
- VUID-vkCmdDrawMeshTasksEXT-OpImageWeightedSampleQCOM-06971
  If OpImageWeightedSampleQCOM is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_WEIGHT_SAMPLED_IMAGE_BIT_QCOM
- VUID-vkCmdDrawMeshTasksEXT-OpImageWeightedSampleQCOM-06972
  If OpImageWeightedSampleQCOM uses a VkImageView as a sample weight image as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_WEIGHT_IMAGE_BIT_QCOM
- VUID-vkCmdDrawMeshTasksEXT-OpImageBoxFilterQCOM-06973
  If OpImageBoxFilterQCOM is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_BOX_FILTER_SAMPLED_BIT_QCOM
- VUID-vkCmdDrawMeshTasksEXT-OpImageBlockMatchSSDQCOM-06974
  If OpImageBlockMatchSSDQCOM is used to read from an VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM
- VUID-vkCmdDrawMeshTasksEXT-OpImageBlockMatchSADQCOM-06975
  If OpImageBlockMatchSADQCOM is used to read from an VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM

- VUID-vkCmdDrawMeshTasksEXT-OpImageBlockMatchSADQCOM-06976
  If `OpImageBlockMatchSADQCOM` or OpImageBlockMatchSSDQCOM is used to read from a reference image as result of this command, then the specified reference coordinates **must** not fail integer texel coordinate validation
- VUID-vkCmdDrawMeshTasksEXT-OpImageWeightedSampleQCOM-06977
  If `OpImageWeightedSampleQCOM`, `OpImageBoxFilterQCOM`, `OpImageBlockMatchWindowSSDQCOM`, `OpImageBlockMatchWindowSADQCOM`, `OpImageBlockMatchGatherSSDQCOM`, `OpImageBlockMatchGatherSADQCOM`, `OpImageBlockMatchSSDQCOM`, or `OpImageBlockMatchSADQCOM` uses a VkSampler as a result of this command, then the sampler **must** have been created with `VK_SAMPLER_CREATE_IMAGE_PROCESSING_BIT_QCOM`
- VUID-vkCmdDrawMeshTasksEXT-OpImageWeightedSampleQCOM-06978
  If any command other than `OpImageWeightedSampleQCOM`, `OpImageBoxFilterQCOM`, `OpImageBlockMatchWindowSSDQCOM`, `OpImageBlockMatchWindowSADQCOM`, `OpImageBlockMatchGatherSSDQCOM`, `OpImageBlockMatchGatherSADQCOM`, `OpImageBlockMatchSSDQCOM`, or `OpImageBlockMatchSADQCOM` uses a VkSampler as a result of this command, then the sampler **must** not have been created with `VK_SAMPLER_CREATE_IMAGE_PROCESSING_BIT_QCOM`
- VUID-vkCmdDrawMeshTasksEXT-OpImageBlockMatchWindow-09215
  If a `OpImageBlockMatchWindow*QCOM` or `OpImageBlockMatchGather*QCOM` instruction is used to read from an VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM`
- VUID-vkCmdDrawMeshTasksEXT-OpImageBlockMatchWindow-09216
  If a `OpImageBlockMatchWindow*QCOM` or `OpImageBlockMatchGather*QCOM` instruction is used to read from an VkImageView as a result of this command, then the image view's format **must** be a single-component format
- VUID-vkCmdDrawMeshTasksEXT-OpImageBlockMatchWindow-09217
  If a `OpImageBlockMatchWindow*QCOM` or `OpImageBlockMatchGather*QCOM` read from a reference image as result of this command, then the specified reference coordinates **must** not fail integer texel coordinate validation
- VUID-vkCmdDrawMeshTasksEXT-None-07288
  Any shader invocation executed by this command **must** terminate
- VUID-vkCmdDrawMeshTasksEXT-None-09600
  If a descriptor with type equal to any of `VK_DESCRIPTOR_TYPE_SAMPLE_WEIGHT_IMAGE_QCOM`, `VK_DESCRIPTOR_TYPE_BLOCK_MATCH_IMAGE_QCOM`, `VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE`, `VK_DESCRIPTOR_TYPE_STORAGE_IMAGE`, or `VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT` is accessed as a result of this command, the image subresource identified by that descriptor **must** be in the image layout identified when the descriptor was written
- VUID-vkCmdDrawMeshTasksEXT-renderPass-02684
  The current render pass **must** be compatible with the `renderPass` member of the `VkGraphicsPipelineCreateInfo` structure specified when creating the `VkPipeline` bound to `VK_PIPELINE_BIND_POINT_GRAPHICS`
- VUID-vkCmdDrawMeshTasksEXT-subpass-02685
  The subpass index of the current render pass **must** be equal to the `subpass` member of the `VkGraphicsPipelineCreateInfo` structure specified when creating the `VkPipeline` bound to

VK_PIPELINE_BIND_POINT_GRAPHICS

- VUID-vkCmdDrawMeshTasksEXT-None-07748
  If any shader statically accesses an input attachment, a valid descriptor **must** be bound to the pipeline via a descriptor set
- VUID-vkCmdDrawMeshTasksEXT-OpTypeImage-07468
  If any shader executed by this pipeline accesses an OpTypeImage variable with a Dim operand of SubpassData, it **must** be decorated with an InputAttachmentIndex that corresponds to a valid input attachment in the current subpass
- VUID-vkCmdDrawMeshTasksEXT-None-07469
  Input attachment views accessed in a subpass **must** be created with the same VkFormat as the corresponding subpass definition, and be created with a VkImageView that is compatible with the attachment referenced by the subpass' pInputAttachments [InputAttachmentIndex] in the currently bound VkFramebuffer as specified by Fragment Input Attachment Compatibility
- VUID-vkCmdDrawMeshTasksEXT-pDepthInputAttachmentIndex-09595
  Input attachment views accessed in a dynamic render pass with a InputAttachmentIndex referenced by VkRenderingInputAttachmentIndexInfoKHR, or no InputAttachmentIndex if VkRenderingInputAttachmentIndexInfoKHR:pDepthInputAttachmentIndex or VkRenderingInputAttachmentIndexInfoKHR:pStencilInputAttachmentIndex are NULL, **must** be created with a VkImageView that is compatible with the corresponding color, depth, or stencil attachment in VkRenderingInfo
- VUID-vkCmdDrawMeshTasksEXT-pDepthInputAttachmentIndex-09596
  Input attachment views accessed in a dynamic render pass via a shader object **must** have an InputAttachmentIndex if both VkRenderingInputAttachmentIndexInfoKHR:pDepthInputAttachmentIndex and VkRenderingInputAttachmentIndexInfoKHR:pStencilInputAttachmentIndex are non-NULL
- VUID-vkCmdDrawMeshTasksEXT-InputAttachmentIndex-09597
  If an input attachment view accessed in a dynamic render pass via a shader object has an InputAttachmentIndex, the InputAttachmentIndex **must** match an index in VkRenderingInputAttachmentIndexInfoKHR
- VUID-vkCmdDrawMeshTasksEXT-None-06537
  Memory backing image subresources used as attachments in the current render pass **must** not be written in any way other than as an attachment by this command
- VUID-vkCmdDrawMeshTasksEXT-None-09000
  If a color attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass, it is not in the VK_IMAGE_LAYOUT_ATTACHMENT_FEEDBACK_LOOP_OPTIMAL_EXT image layout, and either:
  - the VK_PIPELINE_CREATE_COLOR_ATTACHMENT_FEEDBACK_LOOP_BIT_EXT is set on the currently bound pipeline or
  - the last call to vkCmdSetAttachmentFeedbackLoopEnableEXT included VK_IMAGE_ASPECT_COLOR_BIT and
    - there is no currently bound graphics pipeline or
    - the currently bound graphics pipeline was created with

VK_DYNAMIC_STATE_ATTACHMENT_FEEDBACK_LOOP_ENABLE_EXT

it **must** not be accessed in any way other than as an attachment by this command

- VUID-vkCmdDrawMeshTasksEXT-None-09001
  If a depth attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass, it is not in the VK_IMAGE_LAYOUT_ATTACHMENT_FEEDBACK_LOOP_OPTIMAL_EXT image layout, and either:
  - the VK_PIPELINE_CREATE_DEPTH_STENCIL_ATTACHMENT_FEEDBACK_LOOP_BIT_EXT is set on the currently bound pipeline or
  - the last call to vkCmdSetAttachmentFeedbackLoopEnableEXT included VK_IMAGE_ASPECT_DEPTH_BIT and
    - there is no currently bound graphics pipeline or
    - the currently bound graphics pipeline was created with VK_DYNAMIC_STATE_ATTACHMENT_FEEDBACK_LOOP_ENABLE_EXT

  it **must** not be accessed in any way other than as an attachment by this command

- VUID-vkCmdDrawMeshTasksEXT-None-09002
  If a stencil attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass, it is not in the VK_IMAGE_LAYOUT_ATTACHMENT_FEEDBACK_LOOP_OPTIMAL_EXT image layout, and either:
  - the VK_PIPELINE_CREATE_DEPTH_STENCIL_ATTACHMENT_FEEDBACK_LOOP_BIT_EXT is set on the currently bound pipeline or
  - the last call to vkCmdSetAttachmentFeedbackLoopEnableEXT included VK_IMAGE_ASPECT_STENCIL_BIT and
    - there is no currently bound graphics pipeline or
    - the currently bound graphics pipeline was created with VK_DYNAMIC_STATE_ATTACHMENT_FEEDBACK_LOOP_ENABLE_EXT

  it **must** not be accessed in any way other than as an attachment by this command

- VUID-vkCmdDrawMeshTasksEXT-None-09003
  If an attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass, it **must** not be accessed in any way other than as an attachment, storage image, or sampled image by this command
- VUID-vkCmdDrawMeshTasksEXT-None-06539
  If any previously recorded command in the current subpass accessed an image subresource used as an attachment in this subpass in any way other than as an attachment, this command **must** not write to that image subresource as an attachment
- VUID-vkCmdDrawMeshTasksEXT-None-06886
  If the current render pass instance uses a depth/stencil attachment with a read-only layout for the depth aspect, depth writes **must** be disabled
- VUID-vkCmdDrawMeshTasksEXT-None-06887
  If the current render pass instance uses a depth/stencil attachment with a read-only

layout for the stencil aspect, both front and back `writeMask` are not zero, and stencil test is enabled, all stencil ops **must** be `VK_STENCIL_OP_KEEP`

- VUID-vkCmdDrawMeshTasksEXT-None-07831
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VIEWPORT` dynamic state enabled then vkCmdSetViewport **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksEXT-None-07832
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_SCISSOR` dynamic state enabled then vkCmdSetScissor **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksEXT-None-07833
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_LINE_WIDTH` dynamic state enabled then vkCmdSetLineWidth **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksEXT-None-08617
  If a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE`, and the most recent call to vkCmdSetPolygonModeEXT in the current command buffer set `polygonMode` to `VK_POLYGON_MODE_LINE`, vkCmdSetLineWidth **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksEXT-None-08618
  If a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE`, and the most recent call to vkCmdSetPrimitiveTopologyEXT in the current command buffer set `primitiveTopology` to any line topology, vkCmdSetLineWidth **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksEXT-None-08619
  If a shader object that outputs line primitives is bound to the `VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT` or `VK_SHADER_STAGE_GEOMETRY_BIT` stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE`, vkCmdSetLineWidth **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksEXT-None-07834
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_DEPTH_BIAS` dynamic state enabled, the current value of `rasterizerDiscardEnable` is `VK_FALSE`, and the current value of `depthBiasEnable` is `VK_TRUE`, then vkCmdSetDepthBounds or vkCmdSetDepthBias2EXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksEXT-None-07835
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_BLEND_CONSTANTS` dynamic state enabled then vkCmdSetBlendConstants

**must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksEXT-None-08621
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetColorBlendEnableEXT in the current command buffer set any element of pColorBlendEnables to VK_TRUE, and the most recent call to vkCmdSetColorBlendEquationEXT in the current command buffer set the same element of pColorBlendEquations to a VkColorBlendEquationEXT structure with any VkBlendFactor member with a value of VK_BLEND_FACTOR_CONSTANT_COLOR, VK_BLEND_FACTOR_ONE_MINUS_CONSTANT_COLOR, VK_BLEND_FACTOR_CONSTANT_ALPHA, or VK_BLEND_FACTOR_ONE_MINUS_CONSTANT_ALPHA, vkCmdSetBlendConstants **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksEXT-None-07836
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_BOUNDS dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of depthBoundsTestEnable is VK_TRUE, then vkCmdSetDepthBounds **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksEXT-None-07837
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_STENCIL_COMPARE_MASK dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of stencilTestEnable is VK_TRUE, then vkCmdSetStencilCompareMask **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksEXT-None-07838
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_STENCIL_WRITE_MASK dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of stencilTestEnable is VK_TRUE, then vkCmdSetStencilWriteMask **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksEXT-None-07839
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_STENCIL_REFERENCE dynamic state enabled, the current value of and rasterizerDiscardEnable is VK_FALSE, the current value of stencilTestEnable is VK_TRUE, then vkCmdSetStencilReference **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksEXT-maxMultiviewInstanceIndex-02688
  If the draw is recorded in a render pass instance with multiview enabled, the maximum

instance index **must** be less than or equal to VkPhysicalDeviceMultiviewProperties::maxMultiviewInstanceIndex

- VUID-vkCmdDrawMeshTasksEXT-sampleLocationsEnable-02689
  If the bound graphics pipeline was created with VkPipelineSampleLocationsStateCreateInfoEXT::sampleLocationsEnable set to VK_TRUE and the current subpass has a depth/stencil attachment, then that attachment **must** have been created with the VK_IMAGE_CREATE_SAMPLE_LOCATIONS_COMPATIBLE_DEPTH_BIT_EXT bit set
- VUID-vkCmdDrawMeshTasksEXT-None-06666
  If the VK_EXT_sample_locations extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of sampleLocationsEnable is VK_TRUE, then vkCmdSetSampleLocationsEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksEXT-None-07840
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_CULL_MODE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetCullModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksEXT-None-07841
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_FRONT_FACE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetFrontFaceEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksEXT-None-07843
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_TEST_ENABLE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, vkCmdSetDepthTestEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksEXT-None-07844
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_WRITE_ENABLE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetDepthWriteEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksEXT-None-07845
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_COMPARE_OP dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of depthTestEnable is VK_TRUE, then vkCmdSetDepthCompareOpEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksEXT-None-07846
  If the depthBounds feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_BOUNDS_TEST_ENABLE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetDepthBoundsTestEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksEXT-None-07847
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_STENCIL_TEST_ENABLE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetStencilTestEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksEXT-None-07848
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_STENCIL_OP dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, the current value of stencilTestEnable is VK_TRUE, then vkCmdSetStencilOpEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksEXT-viewportCount-03417
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, but not the VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT dynamic state enabled, then vkCmdSetViewportWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the viewportCount parameter of vkCmdSetViewportWithCountEXT **must** match the VkPipelineViewportStateCreateInfo::scissorCount of the pipeline
- VUID-vkCmdDrawMeshTasksEXT-scissorCount-03418
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT dynamic state enabled, but not the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, then vkCmdSetScissorWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the scissorCount parameter of vkCmdSetScissorWithCountEXT **must** match the VkPipelineViewportStateCreateInfo::viewportCount of the pipeline
- VUID-vkCmdDrawMeshTasksEXT-viewportCount-03419
  If the bound graphics pipeline state was created with both the VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT and VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic states enabled then both vkCmdSetViewportWithCountEXT and vkCmdSetScissorWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the viewportCount parameter of vkCmdSetViewportWithCountEXT **must** match the scissorCount parameter of vkCmdSetScissorWithCountEXT
- VUID-vkCmdDrawMeshTasksEXT-None-08635
  If a shader object is bound to any graphics stage, then both

vkCmdSetViewportWithCountEXT and vkCmdSetScissorWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the viewportCount parameter of vkCmdSetViewportWithCountEXT **must** match the scissorCount parameter of vkCmdSetScissorWithCountEXT

- VUID-vkCmdDrawMeshTasksEXT-viewportCount-04137
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, but not the VK_DYNAMIC_STATE_VIEWPORT_W_SCALING_NV dynamic state enabled, then the bound graphics pipeline **must** have been created with VkPipelineViewportWScalingStateCreateInfoNV::viewportCount greater or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawMeshTasksEXT-viewportCount-04138
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT and VK_DYNAMIC_STATE_VIEWPORT_W_SCALING_NV dynamic states enabled then the viewportCount parameter in the last call to vkCmdSetViewportWScalingNV **must** be greater than or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawMeshTasksEXT-None-09232
  If the VK_NV_clip_space_w_scaling extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetViewportWScalingEnableNV in the current command buffer set viewportWScalingEnable to VK_TRUE, then vkCmdSetViewportWScalingNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksEXT-None-08636
  If the VK_NV_clip_space_w_scaling extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetViewportWScalingEnableNV in the current command buffer set viewportWScalingEnable to VK_TRUE, then the viewportCount parameter in the last call to vkCmdSetViewportWScalingNV **must** be greater than or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawMeshTasksEXT-viewportCount-04139
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, but not the VK_DYNAMIC_STATE_VIEWPORT_SHADING_RATE_PALETTE_NV dynamic state enabled, then the bound graphics pipeline **must** have been created with VkPipelineViewportShadingRateImageStateCreateInfoNV::viewportCount greater or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawMeshTasksEXT-viewportCount-04140
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT and VK_DYNAMIC_STATE_VIEWPORT_SHADING_RATE_PALETTE_NV dynamic states enabled then the viewportCount parameter in the last call to vkCmdSetViewportShadingRatePaletteNV **must** be greater than or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawMeshTasksEXT-shadingRateImage-09233
  If the shadingRateImage feature is enabled, and a shader object is bound to any graphics

stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetCoarseSampleOrderNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksEXT-shadingRateImage-09234
  If the shadingRateImage feature is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetShadingRateImageEnableNV in the current command buffer set shadingRateImageEnable to VK_TRUE, then vkCmdSetViewportShadingRatePaletteNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksEXT-None-08637
  If the shadingRateImage feature is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetShadingRateImageEnableNV in the current command buffer set shadingRateImageEnable to VK_TRUE, then the viewportCount parameter in the last call to vkCmdSetViewportShadingRatePaletteNV **must** be greater than or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT

- VUID-vkCmdDrawMeshTasksEXT-VkPipelineVieportCreateInfo-04141
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled and a VkPipelineViewportSwizzleStateCreateInfoNV structure chained from VkPipelineViewportStateCreateInfo, then the bound graphics pipeline **must** have been created with VkPipelineViewportSwizzleStateCreateInfoNV::viewportCount greater or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT

- VUID-vkCmdDrawMeshTasksEXT-VkPipelineVieportCreateInfo-04142
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled and a VkPipelineViewportExclusiveScissorStateCreateInfoNV structure chained from VkPipelineViewportStateCreateInfo, then the bound graphics pipeline **must** have been created with VkPipelineViewportExclusiveScissorStateCreateInfoNV::exclusiveScissorCount greater or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT

- VUID-vkCmdDrawMeshTasksEXT-None-07878
  If the exclusiveScissor feature is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_EXCLUSIVE_SCISSOR_ENABLE_NV dynamic state enabled, then vkCmdSetExclusiveScissorEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksEXT-None-07879
  If the exclusiveScissor feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_EXCLUSIVE_SCISSOR_NV dynamic state enabled, and the most recent call to

vkCmdSetExclusiveScissorEnableNV in the current command buffer set any element of pExclusiveScissorEnables to VK_TRUE, then vkCmdSetExclusiveScissorNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksEXT-None-04876
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_RASTERIZER_DISCARD_ENABLE dynamic state enabled, then vkCmdSetRasterizerDiscardEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksEXT-None-04877
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_BIAS_ENABLE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetDepthBiasEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksEXT-logicOp-04878
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_LOGIC_OP_EXT dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of logicOpEnable is VK_TRUE, then vkCmdSetLogicOpEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksEXT-primitiveFragmentShadingRateWithMultipleViewports-04552
  If the primitiveFragmentShadingRateWithMultipleViewports limit is not supported, the bound graphics pipeline was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, and any of the shader stages of the bound graphics pipeline write to the PrimitiveShadingRateKHR built-in, then vkCmdSetViewportWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the viewportCount parameter of vkCmdSetViewportWithCountEXT **must** be 1
- VUID-vkCmdDrawMeshTasksEXT-primitiveFragmentShadingRateWithMultipleViewports-08642
  If the primitiveFragmentShadingRateWithMultipleViewports limit is not supported, and any shader object bound to a graphics stage writes to the PrimitiveShadingRateKHR built-in, then vkCmdSetViewportWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the viewportCount parameter of vkCmdSetViewportWithCountEXT **must** be 1
- VUID-vkCmdDrawMeshTasksEXT-blendEnable-04727
  If rasterization is not disabled in the bound graphics pipeline, then for each color attachment in the subpass, if the corresponding image view's format features do not contain VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT, then the blendEnable member of the corresponding element of the pAttachments member of pColorBlendState **must** be VK_FALSE
- VUID-vkCmdDrawMeshTasksEXT-None-08643
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set

rasterizerDiscardEnable to VK_FALSE, then for each color attachment in the render pass, if the corresponding image view's format features do not contain VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT, then the corresponding member of pColorBlendEnables in the most recent call to vkCmdSetColorBlendEnableEXT in the current command buffer that affected that attachment index **must** have been VK_FALSE

- VUID-vkCmdDrawMeshTasksEXT-multisampledRenderToSingleSampled-07284
  If rasterization is not disabled in the bound graphics pipeline, and none of the following is enabled:
  - the VK_AMD_mixed_attachment_samples extension
  - the VK_NV_framebuffer_mixed_samples extension
  - the multisampledRenderToSingleSampled feature

  then rasterizationSamples for the currently bound graphics pipeline **must** be the same as the current subpass color and/or depth/stencil attachments

- VUID-vkCmdDrawMeshTasksEXT-None-08644
  If a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and none of the following is enabled:
  - the VK_AMD_mixed_attachment_samples extension
  - the VK_NV_framebuffer_mixed_samples extension
  - the multisampledRenderToSingleSampled feature

  then the most recent call to vkCmdSetRasterizationSamplesEXT in the current command buffer **must** have set rasterizationSamples to be the same as the number of samples for the current render pass color and/or depth/stencil attachments

- VUID-vkCmdDrawMeshTasksEXT-None-08876
  If a shader object is bound to any graphics stage, the current render pass instance **must** have been begun with vkCmdBeginRenderingKHR
- VUID-vkCmdDrawMeshTasksEXT-imageView-06172
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the imageView member of pDepthAttachment is not VK_NULL_HANDLE, and the layout member of pDepthAttachment is VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL, this command **must** not write any values to the depth attachment
- VUID-vkCmdDrawMeshTasksEXT-imageView-06173
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the imageView member of pStencilAttachment is not VK_NULL_HANDLE, and the layout member of pStencilAttachment is VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL, this command **must** not write any values to the stencil attachment
- VUID-vkCmdDrawMeshTasksEXT-imageView-06174
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the imageView member of pDepthAttachment is not VK_NULL_HANDLE, and the layout member of pDepthAttachment is VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_STENCIL_ATTACHMENT_OPTIMAL, this command **must** not write any values to the depth attachment

- VUID-vkCmdDrawMeshTasksEXT-imageView-06175
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the imageView member of pStencilAttachment is not VK_NULL_HANDLE, and the layout member of pStencilAttachment is VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_STENCIL_READ_ONLY_OPTIMAL, this command **must** not write any values to the stencil attachment
- VUID-vkCmdDrawMeshTasksEXT-imageView-06176
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the imageView member of pDepthAttachment is not VK_NULL_HANDLE, and the layout member of pDepthAttachment is VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL, this command **must** not write any values to the depth attachment
- VUID-vkCmdDrawMeshTasksEXT-imageView-06177
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the imageView member of pStencilAttachment is not VK_NULL_HANDLE, and the layout member of pStencilAttachment is VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL, this command **must** not write any values to the stencil attachment
- VUID-vkCmdDrawMeshTasksEXT-viewMask-06178
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the currently bound graphics pipeline **must** have been created with a VkPipelineRenderingCreateInfo::viewMask equal to VkRenderingInfo::viewMask
- VUID-vkCmdDrawMeshTasksEXT-colorAttachmentCount-06179
  If the dynamicRenderingUnusedAttachments feature is not enabled and the current render pass instance was begun with vkCmdBeginRenderingKHR, the currently bound graphics pipeline **must** have been created with a VkPipelineRenderingCreateInfo::colorAttachmentCount equal to VkRenderingInfo::colorAttachmentCount
- VUID-vkCmdDrawMeshTasksEXT-dynamicRenderingUnusedAttachments-08910
  If the dynamicRenderingUnusedAttachments feature is not enabled, and the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingInfo::colorAttachmentCount greater than 0, then each element of the VkRenderingInfo::pColorAttachments array with an imageView not equal to VK_NULL_HANDLE **must** have been created with a VkFormat equal to the corresponding element of VkPipelineRenderingCreateInfo::pColorAttachmentFormats used to create the currently bound graphics pipeline
- VUID-vkCmdDrawMeshTasksEXT-dynamicRenderingUnusedAttachments-08912
  If the dynamicRenderingUnusedAttachments feature is not enabled, and the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingInfo::colorAttachmentCount greater than 0, then each element of the VkRenderingInfo::pColorAttachments array with an imageView equal to VK_NULL_HANDLE **must** have the corresponding element of VkPipelineRenderingCreateInfo::pColorAttachmentFormats used to create the currently bound pipeline equal to VK_FORMAT_UNDEFINED
- VUID-vkCmdDrawMeshTasksEXT-dynamicRenderingUnusedAttachments-08911
  If the dynamicRenderingUnusedAttachments feature is enabled, and the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingInfo::colorAttachmentCount greater than 0, then each element of the VkRenderingInfo::pColorAttachments array with an imageView not equal to VK_NULL_HANDLE **must** have

been created with a VkFormat equal to the corresponding element of VkPipelineRenderingCreateInfo::pColorAttachmentFormats used to create the currently bound graphics pipeline, or the corresponding element of VkPipelineRenderingCreateInfo::pColorAttachmentFormats, if it exists, **must** be VK_FORMAT_UNDEFINED

- VUID-vkCmdDrawMeshTasksEXT-colorAttachmentCount-09362
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, with a VkRenderingInfo::colorAttachmentCount equal to 1, there is no shader object bound to any graphics stage, and a color attachment with a resolve mode of VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID, each element of the VkRenderingInfo::pColorAttachments array with a resolveImageView not equal to VK_NULL_HANDLE **must** have been created with an image created with a VkExternalFormatANDROID::externalFormat value equal to the VkExternalFormatANDROID::externalFormat value used to create the currently bound graphics pipeline

- VUID-vkCmdDrawMeshTasksEXT-None-09363
  If there is no shader object bound to any graphics stage, the current render pass instance was begun with vkCmdBeginRenderingKHR and a VkRenderingInfo::colorAttachmentCount equal to 1, and a color attachment with a resolve mode of VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID, each element of the VkRenderingInfo::pColorAttachments array with a imageView not equal to VK_NULL_HANDLE **must** have been created with an image created with a VkExternalFormatANDROID::externalFormat value equal to the VkExternalFormatANDROID::externalFormat value used to create the currently bound graphics pipeline

- VUID-vkCmdDrawMeshTasksEXT-None-09364
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is no shader object bound to any graphics stage, and the currently bound graphics pipeline was created with a non-zero VkExternalFormatANDROID::externalFormat value and with the VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT dynamic state enabled, then vkCmdSetColorBlendEnableEXT **must** have set the blend enable to VK_FALSE prior to this drawing command

- VUID-vkCmdDrawMeshTasksEXT-None-09365
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is no shader object bound to any graphics stage, and the currently bound graphics pipeline was created with a non-zero VkExternalFormatANDROID::externalFormat value and with the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT dynamic state enabled, then vkCmdSetRasterizationSamplesEXT **must** have set rasterizationSamples to VK_SAMPLE_COUNT_1_BIT prior to this drawing command

- VUID-vkCmdDrawMeshTasksEXT-None-09366
  If there is a shader object bound to any graphics stage, and the current render pass includes a color attachment that uses the VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID resolve mode, then vkCmdSetColorBlendEnableEXT **must** have set blend enable to VK_FALSE prior to this drawing command

- VUID-vkCmdDrawMeshTasksEXT-rasterizationSamples-09367
  If there is a shader object bound to any graphics stage, and the current render pass includes a color attachment that uses the `VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID` resolve mode, then vkCmdSetRasterizationSamplesEXT **must** have set `rasterizationSamples` to `VK_SAMPLE_COUNT_1_BIT` prior to this drawing command
- VUID-vkCmdDrawMeshTasksEXT-None-09368
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is no shader object bound to any graphics stage, and the currently bound graphics pipeline was created with a non-zero VkExternalFormatANDROID::`externalFormat` value and with the `VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR` dynamic state enabled, then vkCmdSetFragmentShadingRateKHR **must** have set `pFragmentSize->width` to `1` prior to this drawing command
- VUID-vkCmdDrawMeshTasksEXT-None-09369
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is no shader object bound to any graphics stage, and the currently bound graphics pipeline was created with a non-zero VkExternalFormatANDROID::`externalFormat` value and with the `VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR` dynamic state enabled, then vkCmdSetFragmentShadingRateKHR **must** have set `pFragmentSize->height` to `1` prior to this drawing command
- VUID-vkCmdDrawMeshTasksEXT-pFragmentSize-09370
  If there is a shader object bound to any graphics stage, and the current render pass includes a color attachment that uses the `VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID` resolve mode, then vkCmdSetFragmentShadingRateKHR **must** have set `pFragmentSize->width` to `1` prior to this drawing command
- VUID-vkCmdDrawMeshTasksEXT-pFragmentSize-09371
  If there is a shader object bound to any graphics stage, and the current render pass includes a color attachment that uses the `VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID` resolve mode, then vkCmdSetFragmentShadingRateKHR **must** have set `pFragmentSize->height` to `1` prior to this drawing command
- VUID-vkCmdDrawMeshTasksEXT-None-07749
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_COLOR_WRITE_ENABLE_EXT` dynamic state enabled then vkCmdSetColorWriteEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksEXT-None-08646
  If the `colorWriteEnable` feature is enabled on the device, and a shader object is bound to the `VK_SHADER_STAGE_FRAGMENT_BIT` stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE`, then vkCmdSetColorWriteEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksEXT-attachmentCount-07750

If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_WRITE_ENABLE_EXT dynamic state enabled then the attachmentCount parameter of vkCmdSetColorWriteEnableEXT **must** be greater than or equal to the VkPipelineColorBlendStateCreateInfo::attachmentCount of the currently bound graphics pipeline

- VUID-vkCmdDrawMeshTasksEXT-None-08647
  If the colorWriteEnable feature is enabled on the device, and a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then the attachmentCount parameter of most recent call to vkCmdSetColorWriteEnableEXT in the current command buffer **must** be greater than or equal to the number of color attachments in the current render pass instance
- VUID-vkCmdDrawMeshTasksEXT-None-07751
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_DISCARD_RECTANGLE_EXT dynamic state enabled then vkCmdSetDiscardRectangleEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command for each discard rectangle in VkPipelineDiscardRectangleStateCreateInfoEXT::discardRectangleCount
- VUID-vkCmdDrawMeshTasksEXT-None-07880
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_DISCARD_RECTANGLE_ENABLE_EXT dynamic state enabled then vkCmdSetDiscardRectangleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksEXT-rasterizerDiscardEnable-09236
  If the VK_EXT_discard_rectangles extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetDiscardRectangleEnableEXT in the current command buffer set discardRectangleEnable to VK_TRUE, then vkCmdSetDiscardRectangleEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksEXT-None-08648
  If the VK_EXT_discard_rectangles extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetDiscardRectangleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksEXT-None-07881
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_DISCARD_RECTANGLE_MODE_EXT dynamic state enabled then vkCmdSetDiscardRectangleModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksEXT-None-08649
  If the VK_EXT_discard_rectangles extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the

current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetDiscardRectangleEnableEXT in the current command buffer set discardRectangleEnable to VK_TRUE, then vkCmdSetDiscardRectangleModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksEXT-dynamicRenderingUnusedAttachments-08913
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is not enabled, and VkRenderingInfo::pDepthAttachment->imageView was VK_NULL_HANDLE, the value of VkPipelineRenderingCreateInfo::depthAttachmentFormat used to create the currently bound graphics pipeline **must** be equal to VK_FORMAT_UNDEFINED
- VUID-vkCmdDrawMeshTasksEXT-dynamicRenderingUnusedAttachments-08914
  If current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is not enabled, and VkRenderingInfo::pDepthAttachment->imageView was not VK_NULL_HANDLE, the value of VkPipelineRenderingCreateInfo::depthAttachmentFormat used to create the currently bound graphics pipeline **must** be equal to the VkFormat used to create VkRenderingInfo::pDepthAttachment->imageView
- VUID-vkCmdDrawMeshTasksEXT-dynamicRenderingUnusedAttachments-08915
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is enabled, VkRenderingInfo::pDepthAttachment->imageView was not VK_NULL_HANDLE, and the value of VkPipelineRenderingCreateInfo::depthAttachmentFormat used to create the currently bound graphics pipeline was not equal to the VkFormat used to create VkRenderingInfo::pDepthAttachment->imageView, the value of the format **must** be VK_FORMAT_UNDEFINED
- VUID-vkCmdDrawMeshTasksEXT-dynamicRenderingUnusedAttachments-08916
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is not enabled, and VkRenderingInfo::pStencilAttachment->imageView was VK_NULL_HANDLE, the value of VkPipelineRenderingCreateInfo::stencilAttachmentFormat used to create the currently bound graphics pipeline **must** be equal to VK_FORMAT_UNDEFINED
- VUID-vkCmdDrawMeshTasksEXT-dynamicRenderingUnusedAttachments-08917
  If current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is not enabled, and VkRenderingInfo::pStencilAttachment->imageView was not VK_NULL_HANDLE, the value of VkPipelineRenderingCreateInfo::stencilAttachmentFormat used to create the currently bound graphics pipeline **must** be equal to the VkFormat used to create VkRenderingInfo::pStencilAttachment->imageView
- VUID-vkCmdDrawMeshTasksEXT-dynamicRenderingUnusedAttachments-08918
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is enabled, VkRenderingInfo::pStencilAttachment->imageView was not VK_NULL_HANDLE, and the value of VkPipelineRenderingCreateInfo::stencilAttachmentFormat used to create the currently bound graphics pipeline was not equal to the VkFormat used to create VkRenderingInfo::pStencilAttachment->imageView, the value of the format **must** be VK_FORMAT_UNDEFINED

- VUID-vkCmdDrawMeshTasksEXT-imageView-06183
  If the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingFragmentShadingRateAttachmentInfoKHR::imageView was not VK_NULL_HANDLE, the currently bound graphics pipeline **must** have been created with VK_PIPELINE_CREATE_RENDERING_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR
- VUID-vkCmdDrawMeshTasksEXT-imageView-06184
  If the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingFragmentDensityMapAttachmentInfoEXT::imageView was not VK_NULL_HANDLE, the currently bound graphics pipeline **must** have been created with VK_PIPELINE_CREATE_RENDERING_FRAGMENT_DENSITY_MAP_ATTACHMENT_BIT_EXT
- VUID-vkCmdDrawMeshTasksEXT-colorAttachmentCount-06185
  If the currently bound pipeline was created with a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and the current render pass instance was begun with vkCmdBeginRenderingKHR with a VkRenderingInfo::colorAttachmentCount parameter greater than 0, then each element of the VkRenderingInfo::pColorAttachments array with a imageView not equal to VK_NULL_HANDLE **must** have been created with a sample count equal to the corresponding element of the pColorAttachmentSamples member of VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV used to create the currently bound graphics pipeline
- VUID-vkCmdDrawMeshTasksEXT-pDepthAttachment-06186
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the currently bound pipeline was created with a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and VkRenderingInfo::pDepthAttachment->imageView was not VK_NULL_HANDLE, the value of the depthStencilAttachmentSamples member of VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV used to create the currently bound graphics pipeline **must** be equal to the sample count used to create VkRenderingInfo::pDepthAttachment->imageView
- VUID-vkCmdDrawMeshTasksEXT-pStencilAttachment-06187
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the currently bound pipeline was created with a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and VkRenderingInfo::pStencilAttachment->imageView was not VK_NULL_HANDLE, the value of the depthStencilAttachmentSamples member of VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV used to create the currently bound graphics pipeline **must** be equal to the sample count used to create VkRenderingInfo::pStencilAttachment->imageView
- VUID-vkCmdDrawMeshTasksEXT-multisampledRenderToSingleSampled-07285
  If the currently bound pipeline was created without a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and the multisampledRenderToSingleSampled feature is not enabled, and the current render pass instance was begun with vkCmdBeginRenderingKHR with a VkRenderingInfo::colorAttachmentCount parameter greater than 0, then each element of the VkRenderingInfo::pColorAttachments array with a imageView not equal to VK_NULL_HANDLE **must** have been created with a sample count equal to the value of rasterizationSamples for the currently bound graphics pipeline
- VUID-vkCmdDrawMeshTasksEXT-multisampledRenderToSingleSampled-07286

If the currently bound pipeline was created without a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and the multisampledRenderToSingleSampled feature is not enabled, and VkRenderingInfo::pDepthAttachment->imageView was not VK_NULL_HANDLE, the value of rasterizationSamples for the currently bound graphics pipeline **must** be equal to the sample count used to create VkRenderingInfo::pDepthAttachment->imageView

- VUID-vkCmdDrawMeshTasksEXT-multisampledRenderToSingleSampled-07287
  If the currently bound pipeline was created without a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and the multisampledRenderToSingleSampled feature is not enabled, and VkRenderingInfo::pStencilAttachment->imageView was not VK_NULL_HANDLE, the value of rasterizationSamples for the currently bound graphics pipeline **must** be equal to the sample count used to create VkRenderingInfo::pStencilAttachment->imageView
- VUID-vkCmdDrawMeshTasksEXT-pNext-07935
  If this command has been called inside a render pass instance started with vkCmdBeginRenderingKHR, and the pNext chain of VkRenderingInfo includes a VkMultisampledRenderToSingleSampledInfoEXT structure with multisampledRenderToSingleSampledEnable equal to VK_TRUE, then the value of rasterizationSamples for the currently bound graphics pipeline **must** be equal to VkMultisampledRenderToSingleSampledInfoEXT::rasterizationSamples
- VUID-vkCmdDrawMeshTasksEXT-renderPass-06198
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the currently bound pipeline **must** have been created with a VkGraphicsPipelineCreateInfo::renderPass equal to VK_NULL_HANDLE
- VUID-vkCmdDrawMeshTasksEXT-pColorAttachments-08963
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is a graphics pipeline bound with a fragment shader that statically writes to a color attachment, the color write mask is not zero, color writes are enabled, and the corresponding element of the VkRenderingInfo::pColorAttachments->imageView was not VK_NULL_HANDLE, then the corresponding element of VkPipelineRenderingCreateInfo::pColorAttachmentFormats used to create the pipeline **must** not be VK_FORMAT_UNDEFINED
- VUID-vkCmdDrawMeshTasksEXT-pDepthAttachment-08964
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is a graphics pipeline bound, depth test is enabled, depth write is enabled, and the VkRenderingInfo::pDepthAttachment->imageView was not VK_NULL_HANDLE, then the VkPipelineRenderingCreateInfo::depthAttachmentFormat used to create the pipeline **must** not be VK_FORMAT_UNDEFINED
- VUID-vkCmdDrawMeshTasksEXT-pStencilAttachment-08965
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is a graphics pipeline bound, stencil test is enabled and the VkRenderingInfo::pStencilAttachment->imageView was not VK_NULL_HANDLE, then the VkPipelineRenderingCreateInfo::stencilAttachmentFormat used to create the pipeline **must** not be VK_FORMAT_UNDEFINED
- VUID-vkCmdDrawMeshTasksEXT-primitivesGeneratedQueryWithRasterizerDiscard-06708
  If the primitivesGeneratedQueryWithRasterizerDiscard feature is not enabled and the

VK_QUERY_TYPE_PRIMITIVES_GENERATED_EXT query is active, rasterization discard **must** not be enabled

- VUID-vkCmdDrawMeshTasksEXT-primitivesGeneratedQueryWithNonZeroStreams-06709
  If the primitivesGeneratedQueryWithNonZeroStreams feature is not enabled and the VK_QUERY_TYPE_PRIMITIVES_GENERATED_EXT query is active, the bound graphics pipeline **must** not have been created with a non-zero value in VkPipelineRasterizationStateStreamCreateInfoEXT::rasterizationStream
- VUID-vkCmdDrawMeshTasksEXT-None-07619
  If a shader object is bound to the VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_TESSELLATION_DOMAIN_ORIGIN_EXT dynamic state enabled, then vkCmdSetTessellationDomainOriginEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksEXT-None-07620
  If the depthClamp feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_CLAMP_ENABLE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetDepthClampEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksEXT-None-07621
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_POLYGON_MODE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetPolygonModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksEXT-None-07622
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetRasterizationSamplesEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksEXT-None-07623
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_SAMPLE_MASK_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetSampleMaskEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksEXT-alphaToCoverageEnable-08919
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_ALPHA_TO_COVERAGE_ENABLE_EXT dynamic state enabled, and alphaToCoverageEnable was VK_TRUE in the last call to vkCmdSetAlphaToCoverageEnableEXT, then the Fragment Output Interface **must** contain a variable for the alpha Component word in Location 0 at Index 0
- VUID-vkCmdDrawMeshTasksEXT-alphaToCoverageEnable-08920

If a shader object is bound to any graphics stage, and the most recent call to vkCmdSetAlphaToCoverageEnableEXT in the current command buffer set alphaToCoverageEnable to VK_TRUE, then the Fragment Output Interface **must** contain a variable for the alpha Component word in Location 0 at Index 0

- VUID-vkCmdDrawMeshTasksEXT-None-07624
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_ALPHA_TO_COVERAGE_ENABLE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetAlphaToCoverageEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksEXT-None-07625
  If the alphaToOne feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_ALPHA_TO_ONE_ENABLE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetAlphaToOneEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksEXT-None-07626
  If the logicOp feature is enabled, a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_LOGIC_OP_ENABLE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetLogicOpEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksEXT-None-07627
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT dynamic state enabled then vkCmdSetColorBlendEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksEXT-None-08657
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and both the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE and there are color attachments bound, then vkCmdSetColorBlendEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksEXT-None-07628
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_EQUATION_EXT dynamic state enabled then vkCmdSetColorBlendEquationEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksEXT-None-08658
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetColorBlendEnableEXT for any attachment set that attachment's value in

pColorBlendEnables to VK_TRUE, then vkCmdSetColorBlendEquationEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksEXT-None-07629
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_WRITE_MASK_EXT dynamic state enabled then vkCmdSetColorWriteMaskEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksEXT-None-08659
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and both the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE and there are color attachments bound, then vkCmdSetColorWriteMaskEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksEXT-None-07630
  If the geometryStreams feature is enabled, and a shader object is bound to the VK_SHADER_STAGE_GEOMETRY_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_RASTERIZATION_STREAM_EXT dynamic state enabled, then vkCmdSetRasterizationStreamEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksEXT-None-07631
  If the VK_EXT_conservative_rasterization extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_CONSERVATIVE_RASTERIZATION_MODE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetConservativeRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksEXT-None-07632
  If the VK_EXT_conservative_rasterization extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_EXTRA_PRIMITIVE_OVERESTIMATION_SIZE_EXT dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of conservativeRasterizationMode is VK_CONSERVATIVE_RASTERIZATION_MODE_OVERESTIMATE_EXT, then vkCmdSetExtraPrimitiveOverestimationSizeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksEXT-None-07633
  If the depthClipEnable feature is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_CLIP_ENABLE_EXT dynamic state, then vkCmdSetDepthClipEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksEXT-None-07634
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT dynamic state enabled then vkCmdSetSampleLocationsEnableEXT **must** have been called and not subsequently

invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksEXT-None-08664
  If the VK_EXT_sample_locations extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetSampleLocationsEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksEXT-None-07635
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_ADVANCED_EXT dynamic state enabled then vkCmdSetColorBlendAdvancedEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksEXT-rasterizerDiscardEnable-09416
  If the VK_EXT_blend_operation_advanced extension is enabled, and a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then at least one of vkCmdSetColorBlendEquationEXT and vkCmdSetColorBlendAdvancedEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksEXT-None-07636
  If the VK_EXT_provoking_vertex extension is enabled, a shader object is bound to the VK_SHADER_STAGE_VERTEX_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_PROVOKING_VERTEX_MODE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetProvokingVertexModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksEXT-None-07637
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT dynamic state enabled then vkCmdSetLineRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksEXT-None-08666
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetPolygonModeEXT in the current command buffer set polygonMode to VK_POLYGON_MODE_LINE, then vkCmdSetLineRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksEXT-None-08667
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object is bound to the VK_SHADER_STAGE_VERTEX_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set

rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetPrimitiveTopologyEXT in the current command buffer set primitiveTopology to any line topology, then vkCmdSetLineRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksEXT-None-08668
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object that outputs line primitives is bound to the VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT or VK_SHADER_STAGE_GEOMETRY_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetLineRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksEXT-None-07638
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT dynamic state enabled then vkCmdSetLineStippleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksEXT-None-08669
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetPolygonModeEXT in the current command buffer set polygonMode to VK_POLYGON_MODE_LINE, then vkCmdSetLineStippleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksEXT-None-08670
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object is bound to the VK_SHADER_STAGE_VERTEX_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetPrimitiveTopologyEXT in the current command buffer set primitiveTopology to any line topology, then vkCmdSetLineStippleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksEXT-None-08671
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object that outputs line primitives is bound to the VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT or VK_SHADER_STAGE_GEOMETRY_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetLineStippleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksEXT-None-07849
  If the bound graphics pipeline state was created with the

VK_DYNAMIC_STATE_LINE_STIPPLE_KHR dynamic state enabled then vkCmdSetLineStippleKHR **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksEXT-None-08672
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetLineStippleEnableEXT in the current command buffer set stippledLineEnable to VK_TRUE, then vkCmdSetLineStippleEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksEXT-None-07639
  If the depthClipControl feature is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_CLIP_NEGATIVE_ONE_TO_ONE_EXT dynamic state enabled, then vkCmdSetDepthClipNegativeOneToOneEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksEXT-None-07640
  If the VK_NV_clip_space_w_scaling extension is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_VIEWPORT_W_SCALING_ENABLE_NV dynamic state enabled, then vkCmdSetViewportWScalingEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksEXT-None-07641
  If the VK_NV_viewport_swizzle extension is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_VIEWPORT_SWIZZLE_NV dynamic state enabled, then vkCmdSetViewportSwizzleNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksEXT-None-07642
  If the VK_NV_fragment_coverage_to_color extension is enabled, a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_TO_COLOR_ENABLE_NV dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetCoverageToColorEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksEXT-None-07643
  If the VK_NV_fragment_coverage_to_color extension is enabled, a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_TO_COLOR_LOCATION_NV dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of coverageToColorEnable is VK_TRUE, then vkCmdSetCoverageToColorLocationNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksEXT-None-07644

If the VK_NV_framebuffer_mixed_samples extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_MODULATION_MODE_NV dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetCoverageModulationModeNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksEXT-None-07645
  If the VK_NV_framebuffer_mixed_samples extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_MODULATION_TABLE_ENABLE_NV dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of coverageModulationMode is any value other than VK_COVERAGE_MODULATION_MODE_NONE_NV, then vkCmdSetCoverageModulationTableEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksEXT-None-07646
  If the VK_NV_framebuffer_mixed_samples extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_MODULATION_TABLE_NV dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of coverageModulationTableEnable is VK_TRUE, then vkCmdSetCoverageModulationTableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksEXT-None-07647
  If the shadingRateImage feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_SHADING_RATE_IMAGE_ENABLE_NV dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetShadingRateImageEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksEXT-pipelineFragmentShadingRate-09238
  If the pipelineFragmentShadingRate feature is enabled, a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetFragmentShadingRateKHR **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksEXT-None-07648
  If the representativeFragmentTest feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_REPRESENTATIVE_FRAGMENT_TEST_ENABLE_NV dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetRepresentativeFragmentTestEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksEXT-None-07649
  If the coverageReductionMode feature is enabled, a shader object is bound to any graphics

stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_REDUCTION_MODE_NV dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetCoverageReductionModeNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksEXT-pColorBlendEnables-07470
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT state enabled and the last call to vkCmdSetColorBlendEnableEXT set pColorBlendEnables for any attachment to VK_TRUE, then for those attachments in the subpass the corresponding image view's format features **must** contain VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT

- VUID-vkCmdDrawMeshTasksEXT-rasterizationSamples-07471
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, and the current subpass does not use any color and/or depth/stencil attachments, then the rasterizationSamples in the last call to vkCmdSetRasterizationSamplesEXT **must** follow the rules for a zero-attachment subpass

- VUID-vkCmdDrawMeshTasksEXT-samples-07472
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_MASK_EXT state enabled and the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state disabled, then the samples parameter in the last call to vkCmdSetSampleMaskEXT **must** be greater or equal to the VkPipelineMultisampleStateCreateInfo::rasterizationSamples parameter used to create the bound graphics pipeline

- VUID-vkCmdDrawMeshTasksEXT-samples-07473
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_MASK_EXT state and VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT states enabled, then the samples parameter in the last call to vkCmdSetSampleMaskEXT **must** be greater or equal to the rasterizationSamples parameter in the last call to vkCmdSetRasterizationSamplesEXT

- VUID-vkCmdDrawMeshTasksEXT-rasterizationSamples-07474
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, and neither the VK_AMD_mixed_attachment_samples nor the VK_NV_framebuffer_mixed_samples extensions are enabled, then the rasterizationSamples in the last call to vkCmdSetRasterizationSamplesEXT **must** be the same as the current subpass color and/or depth/stencil attachments

- VUID-vkCmdDrawMeshTasksEXT-None-09211
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, or a shader object is bound to any graphics stage, and the current render pass instance includes a VkMultisampledRenderToSingleSampledInfoEXT structure with multisampledRenderToSingleSampledEnable equal to VK_TRUE, then the rasterizationSamples in the last call to vkCmdSetRasterizationSamplesEXT **must** be the same as the rasterizationSamples member of that structure

- VUID-vkCmdDrawMeshTasksEXT-firstAttachment-07476
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT` dynamic state enabled then vkCmdSetColorBlendEnableEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the `firstAttachment` and `attachmentCount` parameters of `vkCmdSetColorBlendEnableEXT` calls **must** specify an enable for all active color attachments in the current subpass
- VUID-vkCmdDrawMeshTasksEXT-rasterizerDiscardEnable-09417
  If a shader object is bound to the `VK_SHADER_STAGE_FRAGMENT_BIT` stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE`, then vkCmdSetColorBlendEnableEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the `firstAttachment` and `attachmentCount` parameters of `vkCmdSetColorBlendEnableEXT` calls **must** specify an enable for all active color attachments in the current subpass
- VUID-vkCmdDrawMeshTasksEXT-firstAttachment-07477
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_COLOR_BLEND_EQUATION_EXT` dynamic state enabled then vkCmdSetColorBlendEquationEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the `firstAttachment` and `attachmentCount` parameters of `vkCmdSetColorBlendEquationEXT` calls **must** specify the blend equations for all active color attachments in the current subpass where blending is enabled
- VUID-vkCmdDrawMeshTasksEXT-rasterizerDiscardEnable-09418
  If a shader object is bound to the `VK_SHADER_STAGE_FRAGMENT_BIT` stage, and both the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE` and there are color attachments bound, then vkCmdSetColorBlendEquationEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the `firstAttachment` and `attachmentCount` parameters of `vkCmdSetColorBlendEquationEXT` calls **must** specify the blend equations for all active color attachments in the current subpass where blending is enabled
- VUID-vkCmdDrawMeshTasksEXT-firstAttachment-07478
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_COLOR_WRITE_MASK_EXT` dynamic state enabled then vkCmdSetColorWriteMaskEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the `firstAttachment` and `attachmentCount` parameters of `vkCmdSetColorWriteMaskEXT` calls **must** specify the color write mask for all active color attachments in the current subpass
- VUID-vkCmdDrawMeshTasksEXT-rasterizerDiscardEnable-09419
  If a shader object is bound to the `VK_SHADER_STAGE_FRAGMENT_BIT` stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE`, then vkCmdSetColorWriteMaskEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the `firstAttachment` and `attachmentCount` parameters of `vkCmdSetColorWriteMaskEXT` calls **must** specify the color write mask for all active color

attachments in the current subpass

- VUID-vkCmdDrawMeshTasksEXT-firstAttachment-07479
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_COLOR_BLEND_ADVANCED_EXT` dynamic state enabled then vkCmdSetColorBlendAdvancedEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the `firstAttachment` and `attachmentCount` parameters of `vkCmdSetColorBlendAdvancedEXT` calls **must** specify the advanced blend equations for all active color attachments in the current subpass where blending is enabled

- VUID-vkCmdDrawMeshTasksEXT-advancedBlendMaxColorAttachments-07480
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_COLOR_BLEND_ADVANCED_EXT` and `VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT` dynamic states enabled and the last calls to vkCmdSetColorBlendEnableEXT and vkCmdSetColorBlendAdvancedEXT have enabled advanced blending, then the number of active color attachments in the current subpass **must** not exceed `advancedBlendMaxColorAttachments`

- VUID-vkCmdDrawMeshTasksEXT-primitivesGeneratedQueryWithNonZeroStreams-07481
  If the `primitivesGeneratedQueryWithNonZeroStreams` feature is not enabled and the `VK_QUERY_TYPE_PRIMITIVES_GENERATED_EXT` query is active, and the bound graphics pipeline was created with `VK_DYNAMIC_STATE_RASTERIZATION_STREAM_EXT` state enabled, the last call to vkCmdSetRasterizationStreamEXT **must** have set the `rasterizationStream` to zero

- VUID-vkCmdDrawMeshTasksEXT-sampleLocationsPerPixel-07482
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT` state enabled and the `VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT` state disabled, then the `sampleLocationsPerPixel` member of `pSampleLocationsInfo` in the last call to vkCmdSetSampleLocationsEXT **must** equal the `rasterizationSamples` member of the VkPipelineMultisampleStateCreateInfo structure the bound graphics pipeline has been created with

- VUID-vkCmdDrawMeshTasksEXT-sampleLocationsPerPixel-07483
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT` state enabled and the `VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT` state enabled, then the `sampleLocationsPerPixel` member of `pSampleLocationsInfo` in the last call to vkCmdSetSampleLocationsEXT **must** equal the `rasterizationSamples` parameter of the last call to vkCmdSetRasterizationSamplesEXT

- VUID-vkCmdDrawMeshTasksEXT-sampleLocationsEnable-07484
  If a shader object is bound to the `VK_SHADER_STAGE_FRAGMENT_BIT` stage, or the bound graphics pipeline was created with the `VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT` state enabled, and `sampleLocationsEnable` was `VK_TRUE` in the last call to vkCmdSetSampleLocationsEnableEXT, and the current subpass has a depth/stencil attachment, then that attachment **must** have been created with the `VK_IMAGE_CREATE_SAMPLE_LOCATIONS_COMPATIBLE_DEPTH_BIT_EXT` bit set

- VUID-vkCmdDrawMeshTasksEXT-sampleLocationsEnable-07485
  If a shader object is bound to the `VK_SHADER_STAGE_FRAGMENT_BIT` stage, or the bound

graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state enabled and the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, and if sampleLocationsEnable was VK_TRUE in the last call to vkCmdSetSampleLocationsEnableEXT, then the sampleLocationsInfo.sampleLocationGridSize.width in the last call to vkCmdSetSampleLocationsEXT **must** evenly divide VkMultisamplePropertiesEXT::sampleLocationGridSize.width as returned by vkGetPhysicalDeviceMultisamplePropertiesEXT with a samples parameter equaling rasterizationSamples

- VUID-vkCmdDrawMeshTasksEXT-sampleLocationsEnable-07486
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state enabled and the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, and if sampleLocationsEnable was VK_TRUE in the last call to vkCmdSetSampleLocationsEnableEXT, then the sampleLocationsInfo.sampleLocationGridSize.height in the last call to vkCmdSetSampleLocationsEXT **must** evenly divide VkMultisamplePropertiesEXT::sampleLocationGridSize.height as returned by vkGetPhysicalDeviceMultisamplePropertiesEXT with a samples parameter equaling rasterizationSamples

- VUID-vkCmdDrawMeshTasksEXT-sampleLocationsEnable-07487
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, and if sampleLocationsEnable was VK_TRUE in the last call to vkCmdSetSampleLocationsEnableEXT, the fragment shader code **must** not statically use the extended instruction InterpolateAtSample

- VUID-vkCmdDrawMeshTasksEXT-sampleLocationsEnable-07936
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state disabled and the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, the sampleLocationsEnable member of a VkPipelineSampleLocationsStateCreateInfoEXT::sampleLocationsEnable in the bound graphics pipeline is VK_TRUE or VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, then, sampleLocationsInfo.sampleLocationGridSize.width **must** evenly divide VkMultisamplePropertiesEXT::sampleLocationGridSize.width as returned by vkGetPhysicalDeviceMultisamplePropertiesEXT with a samples parameter equaling the value of rasterizationSamples in the last call to vkCmdSetRasterizationSamplesEXT

- VUID-vkCmdDrawMeshTasksEXT-sampleLocationsEnable-07937
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state disabled and the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, the sampleLocationsEnable member of a VkPipelineSampleLocationsStateCreateInfoEXT::sampleLocationsEnable in the bound graphics pipeline is VK_TRUE or VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, then, sampleLocationsInfo.sampleLocationGridSize.height **must** evenly divide VkMultisamplePropertiesEXT::sampleLocationGridSize.height as returned by vkGetPhysicalDeviceMultisamplePropertiesEXT with a samples parameter equaling the value of rasterizationSamples in the last call to vkCmdSetRasterizationSamplesEXT

- VUID-vkCmdDrawMeshTasksEXT-sampleLocationsEnable-07938
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state disabled and the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, the sampleLocationsEnable member of a VkPipelineSampleLocationsStateCreateInfoEXT::sampleLocationsEnable in the bound graphics pipeline is VK_TRUE or VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, then, sampleLocationsInfo.sampleLocationsPerPixel **must** equal rasterizationSamples in the last call to vkCmdSetRasterizationSamplesEXT
- VUID-vkCmdDrawMeshTasksEXT-coverageModulationTableEnable-07488
  If a shader object is bound to any graphics stage or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COVERAGE_MODULATION_TABLE_ENABLE_NV state enabled, and the last call to vkCmdSetCoverageModulationTableEnableNV set coverageModulationTableEnable to VK_TRUE, then the coverageModulationTableCount parameter in the last call to vkCmdSetCoverageModulationTableNV **must** equal the current rasterizationSamples divided by the number of color samples in the current subpass
- VUID-vkCmdDrawMeshTasksEXT-rasterizationSamples-07489
  If the VK_NV_framebuffer_mixed_samples extension is enabled, and if current subpass has a depth/stencil attachment and depth test, stencil test, or depth bounds test are enabled in the currently bound pipeline state, then the current rasterizationSamples **must** be the same as the sample count of the depth/stencil attachment
- VUID-vkCmdDrawMeshTasksEXT-coverageToColorEnable-07490
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COVERAGE_TO_COLOR_ENABLE_NV state enabled and the last call to vkCmdSetCoverageToColorEnableNV set the coverageToColorEnable to VK_TRUE, then the current subpass **must** have a color attachment at the location selected by the last call to vkCmdSetCoverageToColorLocationNV coverageToColorLocation, with a VkFormat of VK_FORMAT_R8_UINT, VK_FORMAT_R8_SINT, VK_FORMAT_R16_UINT, VK_FORMAT_R16_SINT, VK_FORMAT_R32_UINT, or VK_FORMAT_R32_SINT
- VUID-vkCmdDrawMeshTasksEXT-rasterizerDiscardEnable-09420
  If the VK_NV_fragment_coverage_to_color extension is enabled, and a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the last call to vkCmdSetCoverageToColorEnableNV set the coverageToColorEnable to VK_TRUE, then the current subpass **must** have a color attachment at the location selected by the last call to vkCmdSetCoverageToColorLocationNV coverageToColorLocation, with a VkFormat of VK_FORMAT_R8_UINT, VK_FORMAT_R8_SINT, VK_FORMAT_R16_UINT, VK_FORMAT_R16_SINT, VK_FORMAT_R32_UINT, or VK_FORMAT_R32_SINT
- VUID-vkCmdDrawMeshTasksEXT-coverageReductionMode-07491
  If this VK_NV_coverage_reduction_mode extension is enabled, the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COVERAGE_TO_COLOR_ENABLE_NV and VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT states enabled, the current coverage reduction mode coverageReductionMode, then the current rasterizationSamples, and the sample counts for the color and depth/stencil attachments (if the subpass has them) **must** be a valid combination returned by

vkGetPhysicalDeviceSupportedFramebufferMixedSamplesCombinationsNV

- VUID-vkCmdDrawMeshTasksEXT-viewportCount-07492
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, but not the VK_DYNAMIC_STATE_VIEWPORT_SWIZZLE_NV dynamic state enabled, then the bound graphics pipeline **must** have been created with VkPipelineViewportSwizzleStateCreateInfoNV::viewportCount greater or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawMeshTasksEXT-viewportCount-07493
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT and VK_DYNAMIC_STATE_VIEWPORT_SWIZZLE_NV dynamic states enabled then the viewportCount parameter in the last call to vkCmdSetViewportSwizzleNV **must** be greater than or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawMeshTasksEXT-viewportCount-09421
  If the VK_NV_viewport_swizzle extension is enabled, and a shader object is bound to any graphics stage, then the viewportCount parameter in the last call to vkCmdSetViewportSwizzleNV **must** be greater than or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawMeshTasksEXT-rasterizationSamples-07494
  If the VK_NV_framebuffer_mixed_samples extension is enabled, and if the current subpass has any color attachments and rasterizationSamples of the last call to vkCmdSetRasterizationSamplesEXT is greater than the number of color samples, then the pipeline sampleShadingEnable **must** be VK_FALSE
- VUID-vkCmdDrawMeshTasksEXT-stippledLineEnable-07495
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT or VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT dynamic states enabled, and if the current stippledLineEnable state is VK_TRUE and the current lineRasterizationMode state is VK_LINE_RASTERIZATION_MODE_RECTANGULAR_KHR, then the stippledRectangularLines feature **must** be enabled
- VUID-vkCmdDrawMeshTasksEXT-stippledLineEnable-07496
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT or VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT dynamic states enabled, and if the current stippledLineEnable state is VK_TRUE and the current lineRasterizationMode state is VK_LINE_RASTERIZATION_MODE_BRESENHAM_KHR, then the stippledBresenhamLines feature **must** be enabled
- VUID-vkCmdDrawMeshTasksEXT-stippledLineEnable-07497
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT or VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT dynamic states enabled, and if the current stippledLineEnable state is VK_TRUE and the current lineRasterizationMode state is VK_LINE_RASTERIZATION_MODE_RECTANGULAR_SMOOTH_KHR, then the stippledSmoothLines feature **must** be enabled

- VUID-vkCmdDrawMeshTasksEXT-stippledLineEnable-07498
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT or VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT dynamic states enabled, and if the current stippledLineEnable state is VK_TRUE and the current lineRasterizationMode state is VK_LINE_RASTERIZATION_MODE_DEFAULT_KHR, then the stippledRectangularLines feature **must** be enabled and VkPhysicalDeviceLimits::strictLines **must** be VK_TRUE
- VUID-vkCmdDrawMeshTasksEXT-conservativePointAndLineRasterization-07499
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_CONSERVATIVE_RASTERIZATION_MODE_EXT dynamic state enabled, conservativePointAndLineRasterization is not supported, and the effective primitive topology output by the last pre-rasterization shader stage is a line or point, then the conservativeRasterizationMode set by the last call to vkCmdSetConservativeRasterizationModeEXT **must** be VK_CONSERVATIVE_RASTERIZATION_MODE_DISABLED_EXT
- VUID-vkCmdDrawMeshTasksEXT-stage-07073
  If the currently bound pipeline was created with the VkPipelineShaderStageCreateInfo::stage member of an element of VkGraphicsPipelineCreateInfo::pStages set to VK_SHADER_STAGE_VERTEX_BIT, VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT, VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT or VK_SHADER_STAGE_GEOMETRY_BIT, then Mesh Shader Queries **must** not be active
- VUID-vkCmdDrawMeshTasksEXT-None-08877
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_ATTACHMENT_FEEDBACK_LOOP_ENABLE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetAttachmentFeedbackLoopEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksEXT-None-07850
  If dynamic state was inherited from VkCommandBufferInheritanceViewportScissorInfoNV, it **must** be set in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksEXT-None-08684
  If there is no bound graphics pipeline, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_VERTEX_BIT
- VUID-vkCmdDrawMeshTasksEXT-None-08685
  If there is no bound graphics pipeline, and the tessellationShader feature is enabled, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT
- VUID-vkCmdDrawMeshTasksEXT-None-08686
  If there is no bound graphics pipeline, and the tessellationShader feature is enabled, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT
- VUID-vkCmdDrawMeshTasksEXT-None-08687
  If there is no bound graphics pipeline, and the geometryShader feature is enabled,

vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_GEOMETRY_BIT

- VUID-vkCmdDrawMeshTasksEXT-None-08688
  If there is no bound graphics pipeline, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_FRAGMENT_BIT
- VUID-vkCmdDrawMeshTasksEXT-None-08689
  If there is no bound graphics pipeline, and the taskShader feature is enabled, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_TASK_BIT_EXT
- VUID-vkCmdDrawMeshTasksEXT-None-08690
  If there is no bound graphics pipeline, and the meshShader feature is enabled, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_MESH_BIT_EXT
- VUID-vkCmdDrawMeshTasksEXT-None-08693
  If there is no bound graphics pipeline, and at least one of the taskShader and meshShader features is enabled, one of the VK_SHADER_STAGE_VERTEX_BIT or VK_SHADER_STAGE_MESH_BIT_EXT stages **must** have a valid VkShaderEXT bound, and the other **must** have no VkShaderEXT bound
- VUID-vkCmdDrawMeshTasksEXT-None-08694
  If there is no bound graphics pipeline, and both the taskShader and meshShader features are enabled, and a valid VkShaderEXT is bound the to the VK_SHADER_STAGE_MESH_BIT_EXT stage, and that VkShaderEXT was created without the VK_SHADER_CREATE_NO_TASK_SHADER_BIT_EXT flag, a valid VkShaderEXT **must** be bound to the VK_SHADER_STAGE_TASK_BIT_EXT stage
- VUID-vkCmdDrawMeshTasksEXT-None-08695
  If there is no bound graphics pipeline, and both the taskShader and meshShader features are enabled, and a valid VkShaderEXT is bound the to the VK_SHADER_STAGE_MESH_BIT_EXT stage, and that VkShaderEXT was created with the VK_SHADER_CREATE_NO_TASK_SHADER_BIT_EXT flag, there **must** be no VkShaderEXT bound to the VK_SHADER_STAGE_TASK_BIT_EXT stage
- VUID-vkCmdDrawMeshTasksEXT-None-08696
  If there is no bound graphics pipeline, and a valid VkShaderEXT is bound to the VK_SHADER_STAGE_VERTEX_BIT stage, there **must** be no VkShaderEXT bound to either the VK_SHADER_STAGE_TASK_BIT_EXT stage or the VK_SHADER_STAGE_MESH_BIT_EXT stage
- VUID-vkCmdDrawMeshTasksEXT-None-08698
  If any graphics shader is bound which was created with the VK_SHADER_CREATE_LINK_STAGE_BIT_EXT flag, then all shaders created with the VK_SHADER_CREATE_LINK_STAGE_BIT_EXT flag in the same vkCreateShadersEXT call **must** also be bound
- VUID-vkCmdDrawMeshTasksEXT-None-08699
  If any graphics shader is bound which was created with the VK_SHADER_CREATE_LINK_STAGE_BIT_EXT flag, any stages in between stages whose shaders which did not create a shader with the VK_SHADER_CREATE_LINK_STAGE_BIT_EXT flag as part of the same vkCreateShadersEXT call **must** not have any VkShaderEXT bound
- VUID-vkCmdDrawMeshTasksEXT-None-08878

All bound graphics shader objects **must** have been created with identical or identically defined push constant ranges

- VUID-vkCmdDrawMeshTasksEXT-None-08879
  All bound graphics shader objects **must** have been created with identical or identically defined arrays of descriptor set layouts

- VUID-vkCmdDrawMeshTasksEXT-colorAttachmentCount-09372
  If the current render pass instance was begun with vkCmdBeginRenderingKHR and a VkRenderingInfo::`colorAttachmentCount` equal to `1`, a color attachment with a resolve mode of `VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID`, and a fragment shader is bound, it **must** not declare the `DepthReplacing` or `StencilRefReplacingEXT` execution modes

- VUID-vkCmdDrawMeshTasksEXT-pDynamicStates-08715
  If the bound graphics pipeline state includes a fragment shader stage, was created with `VK_DYNAMIC_STATE_DEPTH_WRITE_ENABLE` set in VkPipelineDynamicStateCreateInfo::`pDynamicStates`, and the fragment shader declares the `EarlyFragmentTests` execution mode and uses `OpDepthAttachmentReadEXT`, the `depthWriteEnable` parameter in the last call to vkCmdSetDepthWriteEnableEXT **must** be `VK_FALSE`

- VUID-vkCmdDrawMeshTasksEXT-pDynamicStates-08716
  If the bound graphics pipeline state includes a fragment shader stage, was created with `VK_DYNAMIC_STATE_STENCIL_WRITE_MASK` set in VkPipelineDynamicStateCreateInfo::`pDynamicStates`, and the fragment shader declares the `EarlyFragmentTests` execution mode and uses `OpStencilAttachmentReadEXT`, the `writeMask` parameter in the last call to vkCmdSetStencilWriteMask **must** be `0`

- VUID-vkCmdDrawMeshTasksEXT-None-09116
  If a shader object is bound to any graphics stage or the currently bound graphics pipeline was created with `VK_DYNAMIC_STATE_COLOR_WRITE_MASK_EXT`, and the format of any color attachment is `VK_FORMAT_E5B9G9R9_UFLOAT_PACK32`, the corresponding element of the `pColorWriteMasks` parameter of vkCmdSetColorWriteMaskEXT **must** either include all of `VK_COLOR_COMPONENT_R_BIT`, `VK_COLOR_COMPONENT_G_BIT`, and `VK_COLOR_COMPONENT_B_BIT`, or none of them

- VUID-vkCmdDrawMeshTasksEXT-maxFragmentDualSrcAttachments-09239
  If blending is enabled for any attachment where either the source or destination blend factors for that attachment use the secondary color input, the maximum value of `Location` for any output attachment statically used in the `Fragment Execution Model` executed by this command **must** be less than `maxFragmentDualSrcAttachments`

- VUID-vkCmdDrawMeshTasksEXT-None-09548
  If the current render pass was begun with vkCmdBeginRenderingKHR, and there is no shader object bound to any graphics stage, the value of each element of VkRenderingAttachmentLocationInfoKHR::`pColorAttachmentLocations` set by vkCmdSetRenderingAttachmentLocationsKHR **must** match the value set for the corresponding element in the currently bound pipeline

- VUID-vkCmdDrawMeshTasksEXT-None-09549
  If the current render pass was begun with vkCmdBeginRenderingKHR, and there is no shader object bound to any graphics stage, input attachment index mappings in the currently bound pipeline **must** match those set for the current render pass instance via VkRenderingInputAttachmentIndexInfoKHR

- VUID-vkCmdDrawMeshTasksEXT-None-09642
  If the current render pass was begun with vkCmdBeginRenderingKHR with the VK_RENDERING_ENABLE_LEGACY_DITHERING_BIT_EXT flag, the bound graphics pipeline **must** have been created with VK_PIPELINE_CREATE_2_ENABLE_LEGACY_DITHERING_BIT_EXT
- VUID-vkCmdDrawMeshTasksEXT-None-09643
  If the bound graphics pipeline was created with VK_PIPELINE_CREATE_2_ENABLE_LEGACY_DITHERING_BIT_EXT, the current render pass **must** have begun with vkCmdBeginRenderingKHR with the VK_RENDERING_ENABLE_LEGACY_DITHERING_BIT_EXT flag
- VUID-vkCmdDrawMeshTasksEXT-stage-06480
  The bound graphics pipeline **must** not have been created with the VkPipelineShaderStageCreateInfo::stage member of an element of VkGraphicsPipelineCreateInfo::pStages set to VK_SHADER_STAGE_VERTEX_BIT, VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT, VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT or VK_SHADER_STAGE_GEOMETRY_BIT
- VUID-vkCmdDrawMeshTasksEXT-None-07074
  Transform Feedback Queries **must** not be active
- VUID-vkCmdDrawMeshTasksEXT-None-07075
  Primitives Generated Queries **must** not be active
- VUID-vkCmdDrawMeshTasksEXT-pipelineStatistics-07076
  The pipelineStatistics member used to create any active Pipeline Statistics Query **must** not contain VK_QUERY_PIPELINE_STATISTIC_INPUT_ASSEMBLY_VERTICES_BIT, VK_QUERY_PIPELINE_STATISTIC_INPUT_ASSEMBLY_PRIMITIVES_BIT, VK_QUERY_PIPELINE_STATISTIC_VERTEX_SHADER_INVOCATIONS_BIT, VK_QUERY_PIPELINE_STATISTIC_GEOMETRY_SHADER_INVOCATIONS_BIT, VK_QUERY_PIPELINE_STATISTIC_GEOMETRY_SHADER_PRIMITIVES_BIT, VK_QUERY_PIPELINE_STATISTIC_CLIPPING_INVOCATIONS_BIT, VK_QUERY_PIPELINE_STATISTIC_CLIPPING_PRIMITIVES_BIT, VK_QUERY_PIPELINE_STATISTIC_TESSELLATION_CONTROL_SHADER_PATCHES_BIT, or VK_QUERY_PIPELINE_STATISTIC_TESSELLATION_EVALUATION_SHADER_INVOCATIONS_BIT
- VUID-vkCmdDrawMeshTasksEXT-TaskEXT-07322
  If the current pipeline bound to VK_PIPELINE_BIND_POINT_GRAPHICS contains a shader using the TaskEXT Execution Model, groupCountX **must** be less than or equal to VkPhysicalDeviceMeshShaderPropertiesEXT::maxTaskWorkGroupCount[0]
- VUID-vkCmdDrawMeshTasksEXT-TaskEXT-07323
  If the current pipeline bound to VK_PIPELINE_BIND_POINT_GRAPHICS contains a shader using the TaskEXT Execution Model, groupCountY **must** be less than or equal to VkPhysicalDeviceMeshShaderPropertiesEXT::maxTaskWorkGroupCount[1]
- VUID-vkCmdDrawMeshTasksEXT-TaskEXT-07324
  If the current pipeline bound to VK_PIPELINE_BIND_POINT_GRAPHICS contains a shader using the TaskEXT Execution Model, groupCountZ **must** be less than or equal to VkPhysicalDeviceMeshShaderPropertiesEXT::maxTaskWorkGroupCount[2]
- VUID-vkCmdDrawMeshTasksEXT-TaskEXT-07325

If the current pipeline bound to VK_PIPELINE_BIND_POINT_GRAPHICS contains a shader using the TaskEXT Execution Model, The product of groupCountX, groupCountY and groupCountZ **must** be less than or equal to VkPhysicalDeviceMeshShaderPropertiesEXT::maxTaskWorkGroupTotalCount

- VUID-vkCmdDrawMeshTasksEXT-TaskEXT-07326
  If the current pipeline bound to VK_PIPELINE_BIND_POINT_GRAPHICS does not contain a shader using the TaskEXT Execution Model, groupCountX **must** be less than or equal to VkPhysicalDeviceMeshShaderPropertiesEXT::maxMeshWorkGroupCount[0]
- VUID-vkCmdDrawMeshTasksEXT-TaskEXT-07327
  If the current pipeline bound to VK_PIPELINE_BIND_POINT_GRAPHICS does not contain a shader using the TaskEXT Execution Model, groupCountY **must** be less than or equal to VkPhysicalDeviceMeshShaderPropertiesEXT::maxMeshWorkGroupCount[1]
- VUID-vkCmdDrawMeshTasksEXT-TaskEXT-07328
  If the current pipeline bound to VK_PIPELINE_BIND_POINT_GRAPHICS does not contain a shader using the TaskEXT Execution Model, groupCountZ **must** be less than or equal to VkPhysicalDeviceMeshShaderPropertiesEXT::maxMeshWorkGroupCount[2]
- VUID-vkCmdDrawMeshTasksEXT-TaskEXT-07329
  If the current pipeline bound to VK_PIPELINE_BIND_POINT_GRAPHICS does not contain a shader using the TaskEXT Execution Model, The product of groupCountX, groupCountY and groupCountZ **must** be less than or equal to VkPhysicalDeviceMeshShaderPropertiesEXT::maxMeshWorkGroupTotalCount
- VUID-vkCmdDrawMeshTasksEXT-MeshEXT-07087
  The current pipeline bound to VK_PIPELINE_BIND_POINT_GRAPHICS **must** contain a shader stage using the MeshEXT Execution Model

## Valid Usage (Implicit)

- VUID-vkCmdDrawMeshTasksEXT-commandBuffer-parameter
  commandBuffer **must** be a valid VkCommandBuffer handle
- VUID-vkCmdDrawMeshTasksEXT-commandBuffer-recording
  commandBuffer **must** be in the recording state
- VUID-vkCmdDrawMeshTasksEXT-commandBuffer-cmdpool
  The VkCommandPool that commandBuffer was allocated from **must** support graphics operations
- VUID-vkCmdDrawMeshTasksEXT-renderpass
  This command **must** only be called inside of a render pass instance
- VUID-vkCmdDrawMeshTasksEXT-videocoding
  This command **must** only be called outside of a video coding scope

## Host Synchronization

- Host access to commandBuffer **must** be externally synchronized

- Host access to the VkCommandPool that commandBuffer was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Inside | Outside | Graphics | Action |

To record an indirect mesh tasks drawing command, call:

```
// Provided by VK_EXT_mesh_shader
void vkCmdDrawMeshTasksIndirectEXT(
    VkCommandBuffer                             commandBuffer,
    VkBuffer                                    buffer,
    VkDeviceSize                                offset,
    uint32_t                                    drawCount,
    uint32_t                                    stride);
```

- commandBuffer is the command buffer into which the command is recorded.
- buffer is the buffer containing draw parameters.
- offset is the byte offset into buffer where parameters begin.
- drawCount is the number of draws to execute, and **can** be zero.
- stride is the byte stride between successive sets of draw parameters.

vkCmdDrawMeshTasksIndirectEXT behaves similarly to vkCmdDrawMeshTasksEXT except that the parameters are read by the device from a buffer during execution. drawCount draws are executed by the command, with parameters taken from buffer starting at offset and increasing by stride bytes for each successive draw. The parameters of each draw are encoded in an array of VkDrawMeshTasksIndirectCommandEXT structures. If drawCount is less than or equal to one, stride is ignored.

## Valid Usage

- VUID-vkCmdDrawMeshTasksIndirectEXT-magFilter-04553
  If a VkSampler created with magFilter or minFilter equal to VK_FILTER_LINEAR, reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE, and compareEnable equal to VK_FALSE is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT
- VUID-vkCmdDrawMeshTasksIndirectEXT-magFilter-09598
  If a VkSampler created with magFilter or minFilter equal to VK_FILTER_LINEAR and

reductionMode equal to either VK_SAMPLER_REDUCTION_MODE_MIN or VK_SAMPLER_REDUCTION_MODE_MAX is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT

- VUID-vkCmdDrawMeshTasksIndirectEXT-mipmapMode-04770
  If a VkSampler created with mipmapMode equal to VK_SAMPLER_MIPMAP_MODE_LINEAR, reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE, and compareEnable equal to VK_FALSE is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT
- VUID-vkCmdDrawMeshTasksIndirectEXT-mipmapMode-09599
  If a VkSampler created with mipmapMode equal to VK_SAMPLER_MIPMAP_MODE_LINEAR and reductionMode equal to either VK_SAMPLER_REDUCTION_MODE_MIN or VK_SAMPLER_REDUCTION_MODE_MAX is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT
- VUID-vkCmdDrawMeshTasksIndirectEXT-unnormalizedCoordinates-09635
  If a VkSampler created with unnormalizedCoordinates equal to VK_TRUE is used to sample a VkImageView as a result of this command, then the image view's levelCount and layerCount **must** be 1
- VUID-vkCmdDrawMeshTasksIndirectEXT-unnormalizedCoordinates-09636
  If a VkSampler created with unnormalizedCoordinates equal to VK_TRUE is used to sample a VkImageView as a result of this command, then the image view's viewType **must** be VK_IMAGE_VIEW_TYPE_1D or VK_IMAGE_VIEW_TYPE_2D
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-06479
  If a VkImageView is sampled with depth comparison, the image view's format features **must** contain VK_FORMAT_FEATURE_2_SAMPLED_IMAGE_DEPTH_COMPARISON_BIT
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-02691
  If a VkImageView is accessed using atomic operations as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_STORAGE_IMAGE_ATOMIC_BIT
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-07888
  If a VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER descriptor is accessed using atomic operations as a result of this command, then the storage texel buffer's format features **must** contain VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_ATOMIC_BIT
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-02692
  If a VkImageView is sampled with VK_FILTER_CUBIC_EXT as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_CUBIC_BIT_EXT
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-02693
  If the VK_EXT_filter_cubic extension is not enabled and any VkImageView is sampled with VK_FILTER_CUBIC_EXT as a result of this command, it **must** not have a VkImageViewType of VK_IMAGE_VIEW_TYPE_3D, VK_IMAGE_VIEW_TYPE_CUBE, or VK_IMAGE_VIEW_TYPE_CUBE_ARRAY
- VUID-vkCmdDrawMeshTasksIndirectEXT-filterCubic-02694

Any VkImageView being sampled with VK_FILTER_CUBIC_EXT as a result of this command **must** have a VkImageViewType and format that supports cubic filtering, as specified by VkFilterCubicImageViewImageFormatPropertiesEXT::filterCubic returned by vkGetPhysicalDeviceImageFormatProperties2

- VUID-vkCmdDrawMeshTasksIndirectEXT-filterCubicMinmax-02695
  Any VkImageView being sampled with VK_FILTER_CUBIC_EXT with a reduction mode of either VK_SAMPLER_REDUCTION_MODE_MIN or VK_SAMPLER_REDUCTION_MODE_MAX as a result of this command **must** have a VkImageViewType and format that supports cubic filtering together with minmax filtering, as specified by VkFilterCubicImageViewImageFormatPropertiesEXT::filterCubicMinmax returned by vkGetPhysicalDeviceImageFormatProperties2
- VUID-vkCmdDrawMeshTasksIndirectEXT-cubicRangeClamp-09212
  If the cubicRangeClamp feature is not enabled, then any VkImageView being sampled with VK_FILTER_CUBIC_EXT as a result of this command **must** not have a VkSamplerReductionModeCreateInfo::reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE_RANGECLAMP_QCOM
- VUID-vkCmdDrawMeshTasksIndirectEXT-reductionMode-09213
  Any VkImageView being sampled with a VkSamplerReductionModeCreateInfo::reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE_RANGECLAMP_QCOM as a result of this command **must** sample with VK_FILTER_CUBIC_EXT
- VUID-vkCmdDrawMeshTasksIndirectEXT-selectableCubicWeights-09214
  If the selectableCubicWeights feature is not enabled, then any VkImageView being sampled with VK_FILTER_CUBIC_EXT as a result of this command **must** have VkSamplerCubicWeightsCreateInfoQCOM::cubicWeights equal to VK_CUBIC_FILTER_WEIGHTS_CATMULL_ROM_QCOM
- VUID-vkCmdDrawMeshTasksIndirectEXT-flags-02696
  Any VkImage created with a VkImageCreateInfo::flags containing VK_IMAGE_CREATE_CORNER_SAMPLED_BIT_NV sampled as a result of this command **must** only be sampled using a VkSamplerAddressMode of VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_EDGE
- VUID-vkCmdDrawMeshTasksIndirectEXT-OpTypeImage-07027
  For any VkImageView being written as a storage image where the image format field of the OpTypeImage is Unknown, the view's format features **must** contain VK_FORMAT_FEATURE_2_STORAGE_WRITE_WITHOUT_FORMAT_BIT
- VUID-vkCmdDrawMeshTasksIndirectEXT-OpTypeImage-07028
  For any VkImageView being read as a storage image where the image format field of the OpTypeImage is Unknown, the view's format features **must** contain VK_FORMAT_FEATURE_2_STORAGE_READ_WITHOUT_FORMAT_BIT
- VUID-vkCmdDrawMeshTasksIndirectEXT-OpTypeImage-07029
  For any VkBufferView being written as a storage texel buffer where the image format field of the OpTypeImage is Unknown, the view's buffer features **must** contain VK_FORMAT_FEATURE_2_STORAGE_WRITE_WITHOUT_FORMAT_BIT
- VUID-vkCmdDrawMeshTasksIndirectEXT-OpTypeImage-07030
  Any VkBufferView being read as a storage texel buffer where the image format field of the OpTypeImage is Unknown then the view's buffer features **must** contain

VK_FORMAT_FEATURE_2_STORAGE_READ_WITHOUT_FORMAT_BIT

- VUID-vkCmdDrawMeshTasksIndirectEXT-None-08600
  For each set *n* that is statically used by a bound shader, a descriptor set **must** have been bound to *n* at the same pipeline bind point, with a VkPipelineLayout that is compatible for set *n*, with the VkPipelineLayout used to create the current VkPipeline or the VkDescriptorSetLayout array used to create the current VkShaderEXT , as described in Pipeline Layout Compatibility
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-08601
  For each push constant that is statically used by a bound shader, a push constant value **must** have been set for the same pipeline bind point, with a VkPipelineLayout that is compatible for push constants, with the VkPipelineLayout used to create the current VkPipeline or the VkDescriptorSetLayout array used to create the current VkShaderEXT , as described in Pipeline Layout Compatibility
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-10068
  For each array of resources that is used by a bound shader, the indices used to access members of the array **must** be less than the descriptor count for the identified binding in the descriptor sets used by this command
- VUID-vkCmdDrawMeshTasksIndirectEXT-maintenance4-08602
  If the maintenance4 feature is not enabled, then for each push constant that is statically used by a bound shader, a push constant value **must** have been set for the same pipeline bind point, with a VkPipelineLayout that is compatible for push constants, with the VkPipelineLayout used to create the current VkPipeline or the VkDescriptorSetLayout and VkPushConstantRange arrays used to create the current VkShaderEXT , as described in Pipeline Layout Compatibility
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-08114
  Descriptors in each bound descriptor set, specified via vkCmdBindDescriptorSets, **must** be valid as described by descriptor validity if they are statically used by the VkPipeline bound to the pipeline bind point used by this command and the bound VkPipeline was not created with VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-08115
  If the descriptors used by the VkPipeline bound to the pipeline bind point were specified via vkCmdBindDescriptorSets, the bound VkPipeline **must** have been created without VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-08116
  Descriptors in bound descriptor buffers, specified via vkCmdSetDescriptorBufferOffsetsEXT, **must** be valid if they are dynamically used by the VkPipeline bound to the pipeline bind point used by this command and the bound VkPipeline was created with VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-08604
  Descriptors in bound descriptor buffers, specified via vkCmdSetDescriptorBufferOffsetsEXT, **must** be valid if they are dynamically used by any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-08117

If the descriptors used by the VkPipeline bound to the pipeline bind point were specified via vkCmdSetDescriptorBufferOffsetsEXT, the bound VkPipeline **must** have been created with VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT

- VUID-vkCmdDrawMeshTasksIndirectEXT-None-08119
  If a descriptor is dynamically used with a VkPipeline created with VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT, the descriptor memory **must** be resident
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-08605
  If a descriptor is dynamically used with a VkShaderEXT created with a VkDescriptorSetLayout that was created with VK_DESCRIPTOR_SET_LAYOUT_CREATE_DESCRIPTOR_BUFFER_BIT_EXT, the descriptor memory **must** be resident
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-08606
  If the shaderObject feature is not enabled, a valid pipeline **must** be bound to the pipeline bind point used by this command
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-08608
  If a pipeline is bound to the pipeline bind point used by this command, there **must** not have been any calls to dynamic state setting commands for any state specified statically in the VkPipeline object bound to the pipeline bind point used by this command, since that pipeline was bound
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-08609
  If the VkPipeline object bound to the pipeline bind point used by this command or any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a VkSampler object that uses unnormalized coordinates, that sampler **must** not be used to sample from any VkImage with a VkImageView of the type VK_IMAGE_VIEW_TYPE_3D, VK_IMAGE_VIEW_TYPE_CUBE, VK_IMAGE_VIEW_TYPE_1D_ARRAY, VK_IMAGE_VIEW_TYPE_2D_ARRAY or VK_IMAGE_VIEW_TYPE_CUBE_ARRAY, in any shader stage
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-08610
  If the VkPipeline object bound to the pipeline bind point used by this command or any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a VkSampler object that uses unnormalized coordinates, that sampler **must** not be used with any of the SPIR-V OpImageSample* or OpImageSparseSample* instructions with ImplicitLod, Dref or Proj in their name, in any shader stage
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-08611
  If the VkPipeline object bound to the pipeline bind point used by this command or any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a VkSampler object that uses unnormalized coordinates, that sampler **must** not be used with any of the SPIR-V OpImageSample* or OpImageSparseSample* instructions that includes a LOD bias or any offset values, in any shader stage
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-08607
  If the shaderObject is enabled, either a valid pipeline **must** be bound to the pipeline bind point used by this command, or a valid combination of valid and VK_NULL_HANDLE shader objects **must** be bound to every supported shader stage corresponding to the pipeline bind point used by this command
- VUID-vkCmdDrawMeshTasksIndirectEXT-uniformBuffers-06935

If any stage of the VkPipeline object bound to the pipeline bind point used by this command accesses a uniform buffer, and that stage was created without enabling either VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_EXT or VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_2_EXT for uniformBuffers, and the robustBufferAccess feature is not enabled, that stage **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point

- VUID-vkCmdDrawMeshTasksIndirectEXT-None-08612
  If the robustBufferAccess feature is not enabled, and any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a uniform buffer, it **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point
- VUID-vkCmdDrawMeshTasksIndirectEXT-storageBuffers-06936
  If any stage of the VkPipeline object bound to the pipeline bind point used by this command accesses a storage buffer, and that stage was created without enabling either VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_EXT or VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_2_EXT for storageBuffers, and the robustBufferAccess feature is not enabled, that stage **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-08613
  If the robustBufferAccess feature is not enabled, and any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a storage buffer, it **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point
- VUID-vkCmdDrawMeshTasksIndirectEXT-commandBuffer-02707
  If commandBuffer is an unprotected command buffer and protectedNoFault is not supported, any resource accessed by bound shaders **must** not be a protected resource
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-06550
  If a bound shader accesses a VkSampler or VkImageView object that enables sampler $Y'C_BC_R$ conversion, that object **must** only be used with OpImageSample* or OpImageSparseSample* instructions
- VUID-vkCmdDrawMeshTasksIndirectEXT-ConstOffset-06551
  If a bound shader accesses a VkSampler or VkImageView object that enables sampler $Y'C_BC_R$ conversion, that object **must** not use the ConstOffset and Offset operands
- VUID-vkCmdDrawMeshTasksIndirectEXT-viewType-07752
  If a VkImageView is accessed as a result of this command, then the image view's viewType **must** match the Dim operand of the OpTypeImage as described in Compatibility Between SPIR-V Image Dimensions and Vulkan ImageView Types
- VUID-vkCmdDrawMeshTasksIndirectEXT-format-07753
  If a VkImageView is accessed as a result of this command, then the numeric type of the image view's format and the Sampled Type operand of the OpTypeImage **must** match
- VUID-vkCmdDrawMeshTasksIndirectEXT-OpImageWrite-08795
  If a VkImageView created with a format other than VK_FORMAT_A8_UNORM_KHR is accessed

using OpImageWrite as a result of this command, then the Type of the Texel operand of that instruction **must** have at least as many components as the image view's format

- VUID-vkCmdDrawMeshTasksIndirectEXT-OpImageWrite-08796
  If a VkImageView created with the format VK_FORMAT_A8_UNORM_KHR is accessed using OpImageWrite as a result of this command, then the Type of the Texel operand of that instruction **must** have four components
- VUID-vkCmdDrawMeshTasksIndirectEXT-OpImageWrite-04469
  If a VkBufferView is accessed using OpImageWrite as a result of this command, then the Type of the Texel operand of that instruction **must** have at least as many components as the buffer view's format
- VUID-vkCmdDrawMeshTasksIndirectEXT-SampledType-04470
  If a VkImageView with a VkFormat that has a 64-bit component width is accessed as a result of this command, the SampledType of the OpTypeImage operand of that instruction **must** have a Width of 64
- VUID-vkCmdDrawMeshTasksIndirectEXT-SampledType-04471
  If a VkImageView with a VkFormat that has a component width less than 64-bit is accessed as a result of this command, the SampledType of the OpTypeImage operand of that instruction **must** have a Width of 32
- VUID-vkCmdDrawMeshTasksIndirectEXT-SampledType-04472
  If a VkBufferView with a VkFormat that has a 64-bit component width is accessed as a result of this command, the SampledType of the OpTypeImage operand of that instruction **must** have a Width of 64
- VUID-vkCmdDrawMeshTasksIndirectEXT-SampledType-04473
  If a VkBufferView with a VkFormat that has a component width less than 64-bit is accessed as a result of this command, the SampledType of the OpTypeImage operand of that instruction **must** have a Width of 32
- VUID-vkCmdDrawMeshTasksIndirectEXT-sparseImageInt64Atomics-04474
  If the sparseImageInt64Atomics feature is not enabled, VkImage objects created with the VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT flag **must** not be accessed by atomic instructions through an OpTypeImage with a SampledType with a Width of 64 by this command
- VUID-vkCmdDrawMeshTasksIndirectEXT-sparseImageInt64Atomics-04475
  If the sparseImageInt64Atomics feature is not enabled, VkBuffer objects created with the VK_BUFFER_CREATE_SPARSE_RESIDENCY_BIT flag **must** not be accessed by atomic instructions through an OpTypeImage with a SampledType with a Width of 64 by this command
- VUID-vkCmdDrawMeshTasksIndirectEXT-OpImageWeightedSampleQCOM-06971
  If OpImageWeightedSampleQCOM is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_WEIGHT_SAMPLED_IMAGE_BIT_QCOM
- VUID-vkCmdDrawMeshTasksIndirectEXT-OpImageWeightedSampleQCOM-06972
  If OpImageWeightedSampleQCOM uses a VkImageView as a sample weight image as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_WEIGHT_IMAGE_BIT_QCOM
- VUID-vkCmdDrawMeshTasksIndirectEXT-OpImageBoxFilterQCOM-06973

If OpImageBoxFilterQCOM is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_BOX_FILTER_SAMPLED_BIT_QCOM

- VUID-vkCmdDrawMeshTasksIndirectEXT-OpImageBlockMatchSSDQCOM-06974
  If OpImageBlockMatchSSDQCOM is used to read from an VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM
- VUID-vkCmdDrawMeshTasksIndirectEXT-OpImageBlockMatchSADQCOM-06975
  If OpImageBlockMatchSADQCOM is used to read from an VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM
- VUID-vkCmdDrawMeshTasksIndirectEXT-OpImageBlockMatchSADQCOM-06976
  If OpImageBlockMatchSADQCOM or OpImageBlockMatchSSDQCOM is used to read from a reference image as result of this command, then the specified reference coordinates **must** not fail integer texel coordinate validation
- VUID-vkCmdDrawMeshTasksIndirectEXT-OpImageWeightedSampleQCOM-06977
  If OpImageWeightedSampleQCOM, OpImageBoxFilterQCOM, OpImageBlockMatchWindowSSDQCOM, OpImageBlockMatchWindowSADQCOM, OpImageBlockMatchGatherSSDQCOM, OpImageBlockMatchGatherSADQCOM, OpImageBlockMatchSSDQCOM, or OpImageBlockMatchSADQCOM uses a VkSampler as a result of this command, then the sampler **must** have been created with VK_SAMPLER_CREATE_IMAGE_PROCESSING_BIT_QCOM
- VUID-vkCmdDrawMeshTasksIndirectEXT-OpImageWeightedSampleQCOM-06978
  If any command other than OpImageWeightedSampleQCOM, OpImageBoxFilterQCOM, OpImageBlockMatchWindowSSDQCOM, OpImageBlockMatchWindowSADQCOM, OpImageBlockMatchGatherSSDQCOM, OpImageBlockMatchGatherSADQCOM, OpImageBlockMatchSSDQCOM, or OpImageBlockMatchSADQCOM uses a VkSampler as a result of this command, then the sampler **must** not have been created with VK_SAMPLER_CREATE_IMAGE_PROCESSING_BIT_QCOM
- VUID-vkCmdDrawMeshTasksIndirectEXT-OpImageBlockMatchWindow-09215
  If a OpImageBlockMatchWindow*QCOM or OpImageBlockMatchGather*QCOM instruction is used to read from an VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM
- VUID-vkCmdDrawMeshTasksIndirectEXT-OpImageBlockMatchWindow-09216
  If a OpImageBlockMatchWindow*QCOM or OpImageBlockMatchGather*QCOM instruction is used to read from an VkImageView as a result of this command, then the image view's format **must** be a single-component format
- VUID-vkCmdDrawMeshTasksIndirectEXT-OpImageBlockMatchWindow-09217
  If a OpImageBlockMatchWindow*QCOM or OpImageBlockMatchGather*QCOM read from a reference image as result of this command, then the specified reference coordinates **must** not fail integer texel coordinate validation
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-07288
  Any shader invocation executed by this command **must** terminate
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-09600
  If a descriptor with type equal to any of VK_DESCRIPTOR_TYPE_SAMPLE_WEIGHT_IMAGE_QCOM,

VK_DESCRIPTOR_TYPE_BLOCK_MATCH_IMAGE_QCOM, VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE, VK_DESCRIPTOR_TYPE_STORAGE_IMAGE, or VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT is accessed as a result of this command, the image subresource identified by that descriptor **must** be in the image layout identified when the descriptor was written

- VUID-vkCmdDrawMeshTasksIndirectEXT-renderPass-02684
  The current render pass **must** be compatible with the renderPass member of the VkGraphicsPipelineCreateInfo structure specified when creating the VkPipeline bound to VK_PIPELINE_BIND_POINT_GRAPHICS
- VUID-vkCmdDrawMeshTasksIndirectEXT-subpass-02685
  The subpass index of the current render pass **must** be equal to the subpass member of the VkGraphicsPipelineCreateInfo structure specified when creating the VkPipeline bound to VK_PIPELINE_BIND_POINT_GRAPHICS
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-07748
  If any shader statically accesses an input attachment, a valid descriptor **must** be bound to the pipeline via a descriptor set
- VUID-vkCmdDrawMeshTasksIndirectEXT-OpTypeImage-07468
  If any shader executed by this pipeline accesses an OpTypeImage variable with a Dim operand of SubpassData, it **must** be decorated with an InputAttachmentIndex that corresponds to a valid input attachment in the current subpass
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-07469
  Input attachment views accessed in a subpass **must** be created with the same VkFormat as the corresponding subpass definition, and be created with a VkImageView that is compatible with the attachment referenced by the subpass' pInputAttachments [InputAttachmentIndex] in the currently bound VkFramebuffer as specified by Fragment Input Attachment Compatibility
- VUID-vkCmdDrawMeshTasksIndirectEXT-pDepthInputAttachmentIndex-09595
  Input attachment views accessed in a dynamic render pass with a InputAttachmentIndex referenced by VkRenderingInputAttachmentIndexInfoKHR, or no InputAttachmentIndex if VkRenderingInputAttachmentIndexInfoKHR:pDepthInputAttachmentIndex or VkRenderingInputAttachmentIndexInfoKHR:pStencilInputAttachmentIndex are NULL, **must** be created with a VkImageView that is compatible with the corresponding color, depth, or stencil attachment in VkRenderingInfo
- VUID-vkCmdDrawMeshTasksIndirectEXT-pDepthInputAttachmentIndex-09596
  Input attachment views accessed in a dynamic render pass via a shader object **must** have an InputAttachmentIndex if both VkRenderingInputAttachmentIndexInfoKHR:pDepthInputAttachmentIndex and VkRenderingInputAttachmentIndexInfoKHR:pStencilInputAttachmentIndex are non-NULL
- VUID-vkCmdDrawMeshTasksIndirectEXT-InputAttachmentIndex-09597
  If an input attachment view accessed in a dynamic render pass via a shader object has an InputAttachmentIndex, the InputAttachmentIndex **must** match an index in VkRenderingInputAttachmentIndexInfoKHR
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-06537
  Memory backing image subresources used as attachments in the current render pass **must** not be written in any way other than as an attachment by this command

- VUID-vkCmdDrawMeshTasksIndirectEXT-None-09000
  If a color attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass, it is not in the `VK_IMAGE_LAYOUT_ATTACHMENT_FEEDBACK_LOOP_OPTIMAL_EXT` image layout, and either:
  - the `VK_PIPELINE_CREATE_COLOR_ATTACHMENT_FEEDBACK_LOOP_BIT_EXT` is set on the currently bound pipeline or
  - the last call to vkCmdSetAttachmentFeedbackLoopEnableEXT included `VK_IMAGE_ASPECT_COLOR_BIT` and
    - there is no currently bound graphics pipeline or
    - the currently bound graphics pipeline was created with `VK_DYNAMIC_STATE_ATTACHMENT_FEEDBACK_LOOP_ENABLE_EXT`

  it **must** not be accessed in any way other than as an attachment by this command
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-09001
  If a depth attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass, it is not in the `VK_IMAGE_LAYOUT_ATTACHMENT_FEEDBACK_LOOP_OPTIMAL_EXT` image layout, and either:
  - the `VK_PIPELINE_CREATE_DEPTH_STENCIL_ATTACHMENT_FEEDBACK_LOOP_BIT_EXT` is set on the currently bound pipeline or
  - the last call to vkCmdSetAttachmentFeedbackLoopEnableEXT included `VK_IMAGE_ASPECT_DEPTH_BIT` and
    - there is no currently bound graphics pipeline or
    - the currently bound graphics pipeline was created with `VK_DYNAMIC_STATE_ATTACHMENT_FEEDBACK_LOOP_ENABLE_EXT`

  it **must** not be accessed in any way other than as an attachment by this command
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-09002
  If a stencil attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass, it is not in the `VK_IMAGE_LAYOUT_ATTACHMENT_FEEDBACK_LOOP_OPTIMAL_EXT` image layout, and either:
  - the `VK_PIPELINE_CREATE_DEPTH_STENCIL_ATTACHMENT_FEEDBACK_LOOP_BIT_EXT` is set on the currently bound pipeline or
  - the last call to vkCmdSetAttachmentFeedbackLoopEnableEXT included `VK_IMAGE_ASPECT_STENCIL_BIT` and
    - there is no currently bound graphics pipeline or
    - the currently bound graphics pipeline was created with `VK_DYNAMIC_STATE_ATTACHMENT_FEEDBACK_LOOP_ENABLE_EXT`

  it **must** not be accessed in any way other than as an attachment by this command
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-09003
  If an attachment is written by any prior command in this subpass or by the load, store, or

resolve operations for this subpass, it **must** not be accessed in any way other than as an attachment, storage image, or sampled image by this command

- VUID-vkCmdDrawMeshTasksIndirectEXT-None-06539
  If any previously recorded command in the current subpass accessed an image subresource used as an attachment in this subpass in any way other than as an attachment, this command **must** not write to that image subresource as an attachment
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-06886
  If the current render pass instance uses a depth/stencil attachment with a read-only layout for the depth aspect, depth writes **must** be disabled
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-06887
  If the current render pass instance uses a depth/stencil attachment with a read-only layout for the stencil aspect, both front and back `writeMask` are not zero, and stencil test is enabled, all stencil ops **must** be `VK_STENCIL_OP_KEEP`
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-07831
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VIEWPORT` dynamic state enabled then vkCmdSetViewport **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-07832
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_SCISSOR` dynamic state enabled then vkCmdSetScissor **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-07833
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_LINE_WIDTH` dynamic state enabled then vkCmdSetLineWidth **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-08617
  If a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE`, and the most recent call to vkCmdSetPolygonModeEXT in the current command buffer set `polygonMode` to `VK_POLYGON_MODE_LINE`, vkCmdSetLineWidth **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-08618
  If a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE`, and the most recent call to vkCmdSetPrimitiveTopologyEXT in the current command buffer set `primitiveTopology` to any line topology, vkCmdSetLineWidth **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-08619
  If a shader object that outputs line primitives is bound to the `VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT` or `VK_SHADER_STAGE_GEOMETRY_BIT` stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE`, vkCmdSetLineWidth **must** have been

called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectEXT-None-07834
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_BIAS dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of depthBiasEnable is VK_TRUE, then vkCmdSetDepthBounds or vkCmdSetDepthBias2EXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-07835
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_BLEND_CONSTANTS dynamic state enabled then vkCmdSetBlendConstants **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-08621
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetColorBlendEnableEXT in the current command buffer set any element of pColorBlendEnables to VK_TRUE, and the most recent call to vkCmdSetColorBlendEquationEXT in the current command buffer set the same element of pColorBlendEquations to a VkColorBlendEquationEXT structure with any VkBlendFactor member with a value of VK_BLEND_FACTOR_CONSTANT_COLOR, VK_BLEND_FACTOR_ONE_MINUS_CONSTANT_COLOR, VK_BLEND_FACTOR_CONSTANT_ALPHA, or VK_BLEND_FACTOR_ONE_MINUS_CONSTANT_ALPHA, vkCmdSetBlendConstants **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-07836
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_BOUNDS dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of depthBoundsTestEnable is VK_TRUE, then vkCmdSetDepthBounds **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-07837
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_STENCIL_COMPARE_MASK dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of stencilTestEnable is VK_TRUE, then vkCmdSetStencilCompareMask **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-07838
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_STENCIL_WRITE_MASK dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of

stencilTestEnable is VK_TRUE, then vkCmdSetStencilWriteMask **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectEXT-None-07839
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_STENCIL_REFERENCE dynamic state enabled, the current value of and rasterizerDiscardEnable is VK_FALSE, the current value of stencilTestEnable is VK_TRUE, then vkCmdSetStencilReference **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectEXT-maxMultiviewInstanceIndex-02688
  If the draw is recorded in a render pass instance with multiview enabled, the maximum instance index **must** be less than or equal to VkPhysicalDeviceMultiviewProperties::maxMultiviewInstanceIndex
- VUID-vkCmdDrawMeshTasksIndirectEXT-sampleLocationsEnable-02689
  If the bound graphics pipeline was created with VkPipelineSampleLocationsStateCreateInfoEXT::sampleLocationsEnable set to VK_TRUE and the current subpass has a depth/stencil attachment, then that attachment **must** have been created with the VK_IMAGE_CREATE_SAMPLE_LOCATIONS_COMPATIBLE_DEPTH_BIT_EXT bit set
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-06666
  If the VK_EXT_sample_locations extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of sampleLocationsEnable is VK_TRUE, then vkCmdSetSampleLocationsEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-07840
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_CULL_MODE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetCullModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-07841
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_FRONT_FACE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetFrontFaceEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-07843
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_TEST_ENABLE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, vkCmdSetDepthTestEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-07844

If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_WRITE_ENABLE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetDepthWriteEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectEXT-None-07845
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_COMPARE_OP dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of depthTestEnable is VK_TRUE, then vkCmdSetDepthCompareOpEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-07846
  If the depthBounds feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_BOUNDS_TEST_ENABLE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetDepthBoundsTestEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-07847
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_STENCIL_TEST_ENABLE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetStencilTestEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-07848
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_STENCIL_OP dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, the current value of stencilTestEnable is VK_TRUE, then vkCmdSetStencilOpEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectEXT-viewportCount-03417
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, but not the VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT dynamic state enabled, then vkCmdSetViewportWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the viewportCount parameter of vkCmdSetViewportWithCountEXT **must** match the VkPipelineViewportStateCreateInfo::scissorCount of the pipeline
- VUID-vkCmdDrawMeshTasksIndirectEXT-scissorCount-03418
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT dynamic state enabled, but not the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, then vkCmdSetScissorWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the scissorCount parameter of

vkCmdSetScissorWithCountEXT **must** match the VkPipelineViewportStateCreateInfo::viewportCount of the pipeline

- VUID-vkCmdDrawMeshTasksIndirectEXT-viewportCount-03419
  If the bound graphics pipeline state was created with both the VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT and VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic states enabled then both vkCmdSetViewportWithCountEXT and vkCmdSetScissorWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the viewportCount parameter of vkCmdSetViewportWithCountEXT **must** match the scissorCount parameter of vkCmdSetScissorWithCountEXT
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-08635
  If a shader object is bound to any graphics stage, then both vkCmdSetViewportWithCountEXT and vkCmdSetScissorWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the viewportCount parameter of vkCmdSetViewportWithCountEXT **must** match the scissorCount parameter of vkCmdSetScissorWithCountEXT
- VUID-vkCmdDrawMeshTasksIndirectEXT-viewportCount-04137
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, but not the VK_DYNAMIC_STATE_VIEWPORT_W_SCALING_NV dynamic state enabled, then the bound graphics pipeline **must** have been created with VkPipelineViewportWScalingStateCreateInfoNV::viewportCount greater or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawMeshTasksIndirectEXT-viewportCount-04138
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT and VK_DYNAMIC_STATE_VIEWPORT_W_SCALING_NV dynamic states enabled then the viewportCount parameter in the last call to vkCmdSetViewportWScalingNV **must** be greater than or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-09232
  If the VK_NV_clip_space_w_scaling extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetViewportWScalingEnableNV in the current command buffer set viewportWScalingEnable to VK_TRUE, then vkCmdSetViewportWScalingNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-08636
  If the VK_NV_clip_space_w_scaling extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetViewportWScalingEnableNV in the current command buffer set viewportWScalingEnable to VK_TRUE, then the viewportCount parameter in the last call to vkCmdSetViewportWScalingNV **must** be greater than or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawMeshTasksIndirectEXT-viewportCount-04139
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, but not the VK_DYNAMIC_STATE_VIEWPORT_SHADING_RATE_PALETTE_NV dynamic state enabled, then the

bound graphics pipeline **must** have been created with VkPipelineViewportShadingRateImageStateCreateInfoNV::viewportCount greater or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT

- VUID-vkCmdDrawMeshTasksIndirectEXT-viewportCount-04140
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT and VK_DYNAMIC_STATE_VIEWPORT_SHADING_RATE_PALETTE_NV dynamic states enabled then the viewportCount parameter in the last call to vkCmdSetViewportShadingRatePaletteNV **must** be greater than or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT

- VUID-vkCmdDrawMeshTasksIndirectEXT-shadingRateImage-09233
  If the shadingRateImage feature is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetCoarseSampleOrderNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectEXT-shadingRateImage-09234
  If the shadingRateImage feature is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetShadingRateImageEnableNV in the current command buffer set shadingRateImageEnable to VK_TRUE, then vkCmdSetViewportShadingRatePaletteNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectEXT-None-08637
  If the shadingRateImage feature is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetShadingRateImageEnableNV in the current command buffer set shadingRateImageEnable to VK_TRUE, then the viewportCount parameter in the last call to vkCmdSetViewportShadingRatePaletteNV **must** be greater than or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT

- VUID-vkCmdDrawMeshTasksIndirectEXT-VkPipelineVieportCreateInfo-04141
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled and a VkPipelineViewportSwizzleStateCreateInfoNV structure chained from VkPipelineViewportStateCreateInfo, then the bound graphics pipeline **must** have been created with VkPipelineViewportSwizzleStateCreateInfoNV::viewportCount greater or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT

- VUID-vkCmdDrawMeshTasksIndirectEXT-VkPipelineVieportCreateInfo-04142
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled and a VkPipelineViewportExclusiveScissorStateCreateInfoNV structure chained from VkPipelineViewportStateCreateInfo, then the bound graphics pipeline **must** have been created with VkPipelineViewportExclusiveScissorStateCreateInfoNV

::exclusiveScissorCount greater or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT

- VUID-vkCmdDrawMeshTasksIndirectEXT-None-07878
  If the exclusiveScissor feature is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_EXCLUSIVE_SCISSOR_ENABLE_NV dynamic state enabled, then vkCmdSetExclusiveScissorEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-07879
  If the exclusiveScissor feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_EXCLUSIVE_SCISSOR_NV dynamic state enabled, and the most recent call to vkCmdSetExclusiveScissorEnableNV in the current command buffer set any element of pExclusiveScissorEnables to VK_TRUE, then vkCmdSetExclusiveScissorNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-04876
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_RASTERIZER_DISCARD_ENABLE dynamic state enabled, then vkCmdSetRasterizerDiscardEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-04877
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_BIAS_ENABLE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetDepthBiasEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectEXT-logicOp-04878
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_LOGIC_OP_EXT dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of logicOpEnable is VK_TRUE, then vkCmdSetLogicOpEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectEXT-primitiveFragmentShadingRateWithMultipleViewports-04552
  If the primitiveFragmentShadingRateWithMultipleViewports limit is not supported, the bound graphics pipeline was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, and any of the shader stages of the bound graphics pipeline write to the PrimitiveShadingRateKHR built-in, then vkCmdSetViewportWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the viewportCount parameter of vkCmdSetViewportWithCountEXT **must** be 1
- VUID-vkCmdDrawMeshTasksIndirectEXT-primitiveFragmentShadingRateWithMultipleViewports-08642
  If the primitiveFragmentShadingRateWithMultipleViewports limit is not supported, and any shader object bound to a graphics stage writes to the PrimitiveShadingRateKHR built-in,

then vkCmdSetViewportWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the viewportCount parameter of vkCmdSetViewportWithCountEXT **must** be 1

- VUID-vkCmdDrawMeshTasksIndirectEXT-blendEnable-04727
  If rasterization is not disabled in the bound graphics pipeline, then for each color attachment in the subpass, if the corresponding image view's format features do not contain VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT, then the blendEnable member of the corresponding element of the pAttachments member of pColorBlendState **must** be VK_FALSE

- VUID-vkCmdDrawMeshTasksIndirectEXT-None-08643
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then for each color attachment in the render pass, if the corresponding image view's format features do not contain VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT, then the corresponding member of pColorBlendEnables in the most recent call to vkCmdSetColorBlendEnableEXT in the current command buffer that affected that attachment index **must** have been VK_FALSE

- VUID-vkCmdDrawMeshTasksIndirectEXT-multisampledRenderToSingleSampled-07284
  If rasterization is not disabled in the bound graphics pipeline, and none of the following is enabled:

  - the VK_AMD_mixed_attachment_samples extension
  - the VK_NV_framebuffer_mixed_samples extension
  - the multisampledRenderToSingleSampled feature

  then rasterizationSamples for the currently bound graphics pipeline **must** be the same as the current subpass color and/or depth/stencil attachments

- VUID-vkCmdDrawMeshTasksIndirectEXT-None-08644
  If a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and none of the following is enabled:

  - the VK_AMD_mixed_attachment_samples extension
  - the VK_NV_framebuffer_mixed_samples extension
  - the multisampledRenderToSingleSampled feature

  then the most recent call to vkCmdSetRasterizationSamplesEXT in the current command buffer **must** have set rasterizationSamples to be the same as the number of samples for the current render pass color and/or depth/stencil attachments

- VUID-vkCmdDrawMeshTasksIndirectEXT-None-08876
  If a shader object is bound to any graphics stage, the current render pass instance **must** have been begun with vkCmdBeginRenderingKHR

- VUID-vkCmdDrawMeshTasksIndirectEXT-imageView-06172
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the imageView member of pDepthAttachment is not VK_NULL_HANDLE, and the layout member

of pDepthAttachment is VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL, this command **must** not write any values to the depth attachment

- VUID-vkCmdDrawMeshTasksIndirectEXT-imageView-06173
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the imageView member of pStencilAttachment is not VK_NULL_HANDLE, and the layout member of pStencilAttachment is VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL, this command **must** not write any values to the stencil attachment
- VUID-vkCmdDrawMeshTasksIndirectEXT-imageView-06174
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the imageView member of pDepthAttachment is not VK_NULL_HANDLE, and the layout member of pDepthAttachment is VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_STENCIL_ATTACHMENT_OPTIMAL, this command **must** not write any values to the depth attachment
- VUID-vkCmdDrawMeshTasksIndirectEXT-imageView-06175
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the imageView member of pStencilAttachment is not VK_NULL_HANDLE, and the layout member of pStencilAttachment is VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_STENCIL_READ_ONLY_OPTIMAL, this command **must** not write any values to the stencil attachment
- VUID-vkCmdDrawMeshTasksIndirectEXT-imageView-06176
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the imageView member of pDepthAttachment is not VK_NULL_HANDLE, and the layout member of pDepthAttachment is VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL, this command **must** not write any values to the depth attachment
- VUID-vkCmdDrawMeshTasksIndirectEXT-imageView-06177
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the imageView member of pStencilAttachment is not VK_NULL_HANDLE, and the layout member of pStencilAttachment is VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL, this command **must** not write any values to the stencil attachment
- VUID-vkCmdDrawMeshTasksIndirectEXT-viewMask-06178
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the currently bound graphics pipeline **must** have been created with a VkPipelineRenderingCreateInfo::viewMask equal to VkRenderingInfo::viewMask
- VUID-vkCmdDrawMeshTasksIndirectEXT-colorAttachmentCount-06179
  If the dynamicRenderingUnusedAttachments feature is not enabled and the current render pass instance was begun with vkCmdBeginRenderingKHR, the currently bound graphics pipeline **must** have been created with a VkPipelineRenderingCreateInfo::colorAttachmentCount equal to VkRenderingInfo::colorAttachmentCount
- VUID-vkCmdDrawMeshTasksIndirectEXT-dynamicRenderingUnusedAttachments-08910
  If the dynamicRenderingUnusedAttachments feature is not enabled, and the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingInfo::colorAttachmentCount greater than 0, then each element of the VkRenderingInfo::pColorAttachments array with an imageView not equal to VK_NULL_HANDLE **must** have been created with a VkFormat equal to the corresponding element of VkPipelineRenderingCreateInfo::pColorAttachmentFormats used to create the currently bound graphics pipeline

- VUID-vkCmdDrawMeshTasksIndirectEXT-dynamicRenderingUnusedAttachments-08912
  If the dynamicRenderingUnusedAttachments feature is not enabled, and the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingInfo::colorAttachmentCount greater than 0, then each element of the VkRenderingInfo::pColorAttachments array with an imageView equal to VK_NULL_HANDLE **must** have the corresponding element of VkPipelineRenderingCreateInfo::pColorAttachmentFormats used to create the currently bound pipeline equal to VK_FORMAT_UNDEFINED
- VUID-vkCmdDrawMeshTasksIndirectEXT-dynamicRenderingUnusedAttachments-08911
  If the dynamicRenderingUnusedAttachments feature is enabled, and the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingInfo::colorAttachmentCount greater than 0, then each element of the VkRenderingInfo::pColorAttachments array with an imageView not equal to VK_NULL_HANDLE **must** have been created with a VkFormat equal to the corresponding element of VkPipelineRenderingCreateInfo::pColorAttachmentFormats used to create the currently bound graphics pipeline, or the corresponding element of VkPipelineRenderingCreateInfo::pColorAttachmentFormats, if it exists, **must** be VK_FORMAT_UNDEFINED
- VUID-vkCmdDrawMeshTasksIndirectEXT-colorAttachmentCount-09362
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, with a VkRenderingInfo::colorAttachmentCount equal to 1, there is no shader object bound to any graphics stage, and a color attachment with a resolve mode of VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID, each element of the VkRenderingInfo::pColorAttachments array with a resolveImageView not equal to VK_NULL_HANDLE **must** have been created with an image created with a VkExternalFormatANDROID::externalFormat value equal to the VkExternalFormatANDROID::externalFormat value used to create the currently bound graphics pipeline
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-09363
  If there is no shader object bound to any graphics stage, the current render pass instance was begun with vkCmdBeginRenderingKHR and a VkRenderingInfo::colorAttachmentCount equal to 1, and a color attachment with a resolve mode of VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID, each element of the VkRenderingInfo::pColorAttachments array with a imageView not equal to VK_NULL_HANDLE **must** have been created with an image created with a VkExternalFormatANDROID::externalFormat value equal to the VkExternalFormatANDROID::externalFormat value used to create the currently bound graphics pipeline
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-09364
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is no shader object bound to any graphics stage, and the currently bound graphics pipeline was created with a non-zero VkExternalFormatANDROID::externalFormat value and with the VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT dynamic state enabled, then vkCmdSetColorBlendEnableEXT **must** have set the blend enable to VK_FALSE prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-09365
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is

no shader object bound to any graphics stage, and the currently bound graphics pipeline was created with a non-zero VkExternalFormatANDROID::externalFormat value and with the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT dynamic state enabled, then vkCmdSetRasterizationSamplesEXT **must** have set rasterizationSamples to VK_SAMPLE_COUNT_1_BIT prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectEXT-None-09366
  If there is a shader object bound to any graphics stage, and the current render pass includes a color attachment that uses the VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID resolve mode, then vkCmdSetColorBlendEnableEXT **must** have set blend enable to VK_FALSE prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectEXT-rasterizationSamples-09367
  If there is a shader object bound to any graphics stage, and the current render pass includes a color attachment that uses the VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID resolve mode, then vkCmdSetRasterizationSamplesEXT **must** have set rasterizationSamples to VK_SAMPLE_COUNT_1_BIT prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-09368
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is no shader object bound to any graphics stage, and the currently bound graphics pipeline was created with a non-zero VkExternalFormatANDROID::externalFormat value and with the VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR dynamic state enabled, then vkCmdSetFragmentShadingRateKHR **must** have set pFragmentSize->width to 1 prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-09369
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is no shader object bound to any graphics stage, and the currently bound graphics pipeline was created with a non-zero VkExternalFormatANDROID::externalFormat value and with the VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR dynamic state enabled, then vkCmdSetFragmentShadingRateKHR **must** have set pFragmentSize->height to 1 prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectEXT-pFragmentSize-09370
  If there is a shader object bound to any graphics stage, and the current render pass includes a color attachment that uses the VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID resolve mode, then vkCmdSetFragmentShadingRateKHR **must** have set pFragmentSize->width to 1 prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectEXT-pFragmentSize-09371
  If there is a shader object bound to any graphics stage, and the current render pass includes a color attachment that uses the VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID resolve mode, then vkCmdSetFragmentShadingRateKHR **must** have set pFragmentSize->height to 1 prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-07749
  If the bound graphics pipeline state was created with the

VK_DYNAMIC_STATE_COLOR_WRITE_ENABLE_EXT dynamic state enabled then vkCmdSetColorWriteEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectEXT-None-08646
  If the colorWriteEnable feature is enabled on the device, and a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetColorWriteEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectEXT-attachmentCount-07750
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_WRITE_ENABLE_EXT dynamic state enabled then the attachmentCount parameter of vkCmdSetColorWriteEnableEXT **must** be greater than or equal to the VkPipelineColorBlendStateCreateInfo::attachmentCount of the currently bound graphics pipeline

- VUID-vkCmdDrawMeshTasksIndirectEXT-None-08647
  If the colorWriteEnable feature is enabled on the device, and a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then the attachmentCount parameter of most recent call to vkCmdSetColorWriteEnableEXT in the current command buffer **must** be greater than or equal to the number of color attachments in the current render pass instance

- VUID-vkCmdDrawMeshTasksIndirectEXT-None-07751
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_DISCARD_RECTANGLE_EXT dynamic state enabled then vkCmdSetDiscardRectangleEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command for each discard rectangle in VkPipelineDiscardRectangleStateCreateInfoEXT::discardRectangleCount

- VUID-vkCmdDrawMeshTasksIndirectEXT-None-07880
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_DISCARD_RECTANGLE_ENABLE_EXT dynamic state enabled then vkCmdSetDiscardRectangleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectEXT-rasterizerDiscardEnable-09236
  If the VK_EXT_discard_rectangles extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetDiscardRectangleEnableEXT in the current command buffer set discardRectangleEnable to VK_TRUE, then vkCmdSetDiscardRectangleEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectEXT-None-08648
  If the VK_EXT_discard_rectangles extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the

current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetDiscardRectangleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectEXT-None-07881
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_DISCARD_RECTANGLE_MODE_EXT dynamic state enabled then vkCmdSetDiscardRectangleModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectEXT-None-08649
  If the VK_EXT_discard_rectangles extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetDiscardRectangleEnableEXT in the current command buffer set discardRectangleEnable to VK_TRUE, then vkCmdSetDiscardRectangleModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectEXT-dynamicRenderingUnusedAttachments-08913
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is not enabled, and VkRenderingInfo::pDepthAttachment->imageView was VK_NULL_HANDLE, the value of VkPipelineRenderingCreateInfo::depthAttachmentFormat used to create the currently bound graphics pipeline **must** be equal to VK_FORMAT_UNDEFINED

- VUID-vkCmdDrawMeshTasksIndirectEXT-dynamicRenderingUnusedAttachments-08914
  If current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is not enabled, and VkRenderingInfo::pDepthAttachment->imageView was not VK_NULL_HANDLE, the value of VkPipelineRenderingCreateInfo::depthAttachmentFormat used to create the currently bound graphics pipeline **must** be equal to the VkFormat used to create VkRenderingInfo::pDepthAttachment->imageView

- VUID-vkCmdDrawMeshTasksIndirectEXT-dynamicRenderingUnusedAttachments-08915
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is enabled, VkRenderingInfo::pDepthAttachment->imageView was not VK_NULL_HANDLE, and the value of VkPipelineRenderingCreateInfo::depthAttachmentFormat used to create the currently bound graphics pipeline was not equal to the VkFormat used to create VkRenderingInfo::pDepthAttachment->imageView, the value of the format **must** be VK_FORMAT_UNDEFINED

- VUID-vkCmdDrawMeshTasksIndirectEXT-dynamicRenderingUnusedAttachments-08916
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is not enabled, and VkRenderingInfo::pStencilAttachment->imageView was VK_NULL_HANDLE, the value of VkPipelineRenderingCreateInfo::stencilAttachmentFormat used to create the currently bound graphics pipeline **must** be equal to VK_FORMAT_UNDEFINED

- VUID-vkCmdDrawMeshTasksIndirectEXT-dynamicRenderingUnusedAttachments-08917
  If current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is not enabled, and VkRenderingInfo

::pStencilAttachment->imageView was not VK_NULL_HANDLE, the value of VkPipelineRenderingCreateInfo::stencilAttachmentFormat used to create the currently bound graphics pipeline **must** be equal to the VkFormat used to create VkRenderingInfo::pStencilAttachment->imageView

- VUID-vkCmdDrawMeshTasksIndirectEXT-dynamicRenderingUnusedAttachments-08918
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is enabled, VkRenderingInfo::pStencilAttachment->imageView was not VK_NULL_HANDLE, and the value of VkPipelineRenderingCreateInfo::stencilAttachmentFormat used to create the currently bound graphics pipeline was not equal to the VkFormat used to create VkRenderingInfo::pStencilAttachment->imageView, the value of the format **must** be VK_FORMAT_UNDEFINED
- VUID-vkCmdDrawMeshTasksIndirectEXT-imageView-06183
  If the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingFragmentShadingRateAttachmentInfoKHR::imageView was not VK_NULL_HANDLE, the currently bound graphics pipeline **must** have been created with VK_PIPELINE_CREATE_RENDERING_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR
- VUID-vkCmdDrawMeshTasksIndirectEXT-imageView-06184
  If the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingFragmentDensityMapAttachmentInfoEXT::imageView was not VK_NULL_HANDLE, the currently bound graphics pipeline **must** have been created with VK_PIPELINE_CREATE_RENDERING_FRAGMENT_DENSITY_MAP_ATTACHMENT_BIT_EXT
- VUID-vkCmdDrawMeshTasksIndirectEXT-colorAttachmentCount-06185
  If the currently bound pipeline was created with a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and the current render pass instance was begun with vkCmdBeginRenderingKHR with a VkRenderingInfo::colorAttachmentCount parameter greater than 0, then each element of the VkRenderingInfo::pColorAttachments array with a imageView not equal to VK_NULL_HANDLE **must** have been created with a sample count equal to the corresponding element of the pColorAttachmentSamples member of VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV used to create the currently bound graphics pipeline
- VUID-vkCmdDrawMeshTasksIndirectEXT-pDepthAttachment-06186
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the currently bound pipeline was created with a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and VkRenderingInfo::pDepthAttachment->imageView was not VK_NULL_HANDLE, the value of the depthStencilAttachmentSamples member of VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV used to create the currently bound graphics pipeline **must** be equal to the sample count used to create VkRenderingInfo::pDepthAttachment->imageView
- VUID-vkCmdDrawMeshTasksIndirectEXT-pStencilAttachment-06187
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the currently bound pipeline was created with a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and VkRenderingInfo::pStencilAttachment->imageView was not VK_NULL_HANDLE, the value of the depthStencilAttachmentSamples member of VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV used to create the currently bound graphics pipeline **must** be equal to the sample count

used to create VkRenderingInfo::pStencilAttachment->imageView

- VUID-vkCmdDrawMeshTasksIndirectEXT-multisampledRenderToSingleSampled-07285
  If the currently bound pipeline was created without a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and the multisampledRenderToSingleSampled feature is not enabled, and the current render pass instance was begun with vkCmdBeginRenderingKHR with a VkRenderingInfo::colorAttachmentCount parameter greater than 0, then each element of the VkRenderingInfo::pColorAttachments array with a imageView not equal to VK_NULL_HANDLE **must** have been created with a sample count equal to the value of rasterizationSamples for the currently bound graphics pipeline

- VUID-vkCmdDrawMeshTasksIndirectEXT-multisampledRenderToSingleSampled-07286
  If the currently bound pipeline was created without a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and the multisampledRenderToSingleSampled feature is not enabled, and VkRenderingInfo::pDepthAttachment->imageView was not VK_NULL_HANDLE, the value of rasterizationSamples for the currently bound graphics pipeline **must** be equal to the sample count used to create VkRenderingInfo::pDepthAttachment->imageView

- VUID-vkCmdDrawMeshTasksIndirectEXT-multisampledRenderToSingleSampled-07287
  If the currently bound pipeline was created without a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and the multisampledRenderToSingleSampled feature is not enabled, and VkRenderingInfo::pStencilAttachment->imageView was not VK_NULL_HANDLE, the value of rasterizationSamples for the currently bound graphics pipeline **must** be equal to the sample count used to create VkRenderingInfo::pStencilAttachment->imageView

- VUID-vkCmdDrawMeshTasksIndirectEXT-pNext-07935
  If this command has been called inside a render pass instance started with vkCmdBeginRenderingKHR, and the pNext chain of VkRenderingInfo includes a VkMultisampledRenderToSingleSampledInfoEXT structure with multisampledRenderToSingleSampledEnable equal to VK_TRUE, then the value of rasterizationSamples for the currently bound graphics pipeline **must** be equal to VkMultisampledRenderToSingleSampledInfoEXT::rasterizationSamples

- VUID-vkCmdDrawMeshTasksIndirectEXT-renderPass-06198
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the currently bound pipeline **must** have been created with a VkGraphicsPipelineCreateInfo::renderPass equal to VK_NULL_HANDLE

- VUID-vkCmdDrawMeshTasksIndirectEXT-pColorAttachments-08963
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is a graphics pipeline bound with a fragment shader that statically writes to a color attachment, the color write mask is not zero, color writes are enabled, and the corresponding element of the VkRenderingInfo::pColorAttachments->imageView was not VK_NULL_HANDLE, then the corresponding element of VkPipelineRenderingCreateInfo::pColorAttachmentFormats used to create the pipeline **must** not be VK_FORMAT_UNDEFINED

- VUID-vkCmdDrawMeshTasksIndirectEXT-pDepthAttachment-08964
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is a graphics pipeline bound, depth test is enabled, depth write is enabled, and the

VkRenderingInfo::pDepthAttachment->imageView was not VK_NULL_HANDLE, then the VkPipelineRenderingCreateInfo::depthAttachmentFormat used to create the pipeline **must** not be VK_FORMAT_UNDEFINED

- VUID-vkCmdDrawMeshTasksIndirectEXT-pStencilAttachment-08965
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is a graphics pipeline bound, stencil test is enabled and the VkRenderingInfo::pStencilAttachment->imageView was not VK_NULL_HANDLE, then the VkPipelineRenderingCreateInfo::stencilAttachmentFormat used to create the pipeline **must** not be VK_FORMAT_UNDEFINED

- VUID-vkCmdDrawMeshTasksIndirectEXT-primitivesGeneratedQueryWithRasterizerDiscard-06708
  If the primitivesGeneratedQueryWithRasterizerDiscard feature is not enabled and the VK_QUERY_TYPE_PRIMITIVES_GENERATED_EXT query is active, rasterization discard **must** not be enabled

- VUID-vkCmdDrawMeshTasksIndirectEXT-primitivesGeneratedQueryWithNonZeroStreams-06709
  If the primitivesGeneratedQueryWithNonZeroStreams feature is not enabled and the VK_QUERY_TYPE_PRIMITIVES_GENERATED_EXT query is active, the bound graphics pipeline **must** not have been created with a non-zero value in VkPipelineRasterizationStateStreamCreateInfoEXT::rasterizationStream

- VUID-vkCmdDrawMeshTasksIndirectEXT-None-07619
  If a shader object is bound to the VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_TESSELLATION_DOMAIN_ORIGIN_EXT dynamic state enabled, then vkCmdSetTessellationDomainOriginEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectEXT-None-07620
  If the depthClamp feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_CLAMP_ENABLE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetDepthClampEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectEXT-None-07621
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_POLYGON_MODE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetPolygonModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectEXT-None-07622
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetRasterizationSamplesEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectEXT-None-07623
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_SAMPLE_MASK_EXT` dynamic state enabled, and the current value of `rasterizerDiscardEnable` is `VK_FALSE`, then vkCmdSetSampleMaskEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectEXT-alphaToCoverageEnable-08919
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_ALPHA_TO_COVERAGE_ENABLE_EXT` dynamic state enabled, and `alphaToCoverageEnable` was `VK_TRUE` in the last call to vkCmdSetAlphaToCoverageEnableEXT, then the Fragment Output Interface **must** contain a variable for the alpha `Component` word in `Location` 0 at `Index` 0
- VUID-vkCmdDrawMeshTasksIndirectEXT-alphaToCoverageEnable-08920
  If a shader object is bound to any graphics stage, and the most recent call to vkCmdSetAlphaToCoverageEnableEXT in the current command buffer set `alphaToCoverageEnable` to `VK_TRUE`, then the Fragment Output Interface **must** contain a variable for the alpha `Component` word in `Location` 0 at `Index` 0
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-07624
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_ALPHA_TO_COVERAGE_ENABLE_EXT` dynamic state enabled, and the current value of `rasterizerDiscardEnable` is `VK_FALSE`, then vkCmdSetAlphaToCoverageEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-07625
  If the `alphaToOne` feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_ALPHA_TO_ONE_ENABLE_EXT` dynamic state enabled, and the current value of `rasterizerDiscardEnable` is `VK_FALSE`, then vkCmdSetAlphaToOneEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-07626
  If the `logicOp` feature is enabled, a shader object is bound to the `VK_SHADER_STAGE_FRAGMENT_BIT` stage or a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_LOGIC_OP_ENABLE_EXT` dynamic state enabled, and the current value of `rasterizerDiscardEnable` is `VK_FALSE`, then vkCmdSetLogicOpEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-07627
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT` dynamic state enabled then vkCmdSetColorBlendEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-08657
  If a shader object is bound to the `VK_SHADER_STAGE_FRAGMENT_BIT` stage, and both the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set

rasterizerDiscardEnable to VK_FALSE and there are color attachments bound, then vkCmdSetColorBlendEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectEXT-None-07628
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_EQUATION_EXT dynamic state enabled then vkCmdSetColorBlendEquationEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectEXT-None-08658
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetColorBlendEnableEXT for any attachment set that attachment's value in pColorBlendEnables to VK_TRUE, then vkCmdSetColorBlendEquationEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectEXT-None-07629
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_WRITE_MASK_EXT dynamic state enabled then vkCmdSetColorWriteMaskEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectEXT-None-08659
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and both the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE and there are color attachments bound, then vkCmdSetColorWriteMaskEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectEXT-None-07630
  If the geometryStreams feature is enabled, and a shader object is bound to the VK_SHADER_STAGE_GEOMETRY_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_RASTERIZATION_STREAM_EXT dynamic state enabled, then vkCmdSetRasterizationStreamEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectEXT-None-07631
  If the VK_EXT_conservative_rasterization extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_CONSERVATIVE_RASTERIZATION_MODE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetConservativeRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectEXT-None-07632
  If the VK_EXT_conservative_rasterization extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_EXTRA_PRIMITIVE_OVERESTIMATION_SIZE_EXT dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of

conservativeRasterizationMode is VK_CONSERVATIVE_RASTERIZATION_MODE_OVERESTIMATE_EXT, then vkCmdSetExtraPrimitiveOverestimationSizeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectEXT-None-07633
  If the depthClipEnable feature is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_CLIP_ENABLE_EXT dynamic state, then vkCmdSetDepthClipEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectEXT-None-07634
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT dynamic state enabled then vkCmdSetSampleLocationsEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectEXT-None-08664
  If the VK_EXT_sample_locations extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetSampleLocationsEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectEXT-None-07635
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_ADVANCED_EXT dynamic state enabled then vkCmdSetColorBlendAdvancedEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectEXT-rasterizerDiscardEnable-09416
  If the VK_EXT_blend_operation_advanced extension is enabled, and a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then at least one of vkCmdSetColorBlendEquationEXT and vkCmdSetColorBlendAdvancedEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectEXT-None-07636
  If the VK_EXT_provoking_vertex extension is enabled, a shader object is bound to the VK_SHADER_STAGE_VERTEX_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_PROVOKING_VERTEX_MODE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetProvokingVertexModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectEXT-None-07637
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT dynamic state enabled then vkCmdSetLineRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectEXT-None-08666
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetPolygonModeEXT in the current command buffer set polygonMode to VK_POLYGON_MODE_LINE, then vkCmdSetLineRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-08667
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object is bound to the VK_SHADER_STAGE_VERTEX_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetPrimitiveTopologyEXT in the current command buffer set primitiveTopology to any line topology, then vkCmdSetLineRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-08668
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object that outputs line primitives is bound to the VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT or VK_SHADER_STAGE_GEOMETRY_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetLineRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-07638
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT dynamic state enabled then vkCmdSetLineStippleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-08669
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetPolygonModeEXT in the current command buffer set polygonMode to VK_POLYGON_MODE_LINE, then vkCmdSetLineStippleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-08670
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object is bound to the VK_SHADER_STAGE_VERTEX_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetPrimitiveTopologyEXT in the current command buffer set primitiveTopology to

any line topology, then vkCmdSetLineStippleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectEXT-None-08671
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object that outputs line primitives is bound to the VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT or VK_SHADER_STAGE_GEOMETRY_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetLineStippleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectEXT-None-07849
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_KHR dynamic state enabled then vkCmdSetLineStippleKHR **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectEXT-None-08672
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetLineStippleEnableEXT in the current command buffer set stippledLineEnable to VK_TRUE, then vkCmdSetLineStippleEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectEXT-None-07639
  If the depthClipControl feature is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_CLIP_NEGATIVE_ONE_TO_ONE_EXT dynamic state enabled, then vkCmdSetDepthClipNegativeOneToOneEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectEXT-None-07640
  If the VK_NV_clip_space_w_scaling extension is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_VIEWPORT_W_SCALING_ENABLE_NV dynamic state enabled, then vkCmdSetViewportWScalingEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectEXT-None-07641
  If the VK_NV_viewport_swizzle extension is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_VIEWPORT_SWIZZLE_NV dynamic state enabled, then vkCmdSetViewportSwizzleNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectEXT-None-07642
  If the VK_NV_fragment_coverage_to_color extension is enabled, a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_TO_COLOR_ENABLE_NV dynamic state enabled, and the

current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetCoverageToColorEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectEXT-None-07643
  If the VK_NV_fragment_coverage_to_color extension is enabled, a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_TO_COLOR_LOCATION_NV dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of coverageToColorEnable is VK_TRUE, then vkCmdSetCoverageToColorLocationNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-07644
  If the VK_NV_framebuffer_mixed_samples extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_MODULATION_MODE_NV dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetCoverageModulationModeNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-07645
  If the VK_NV_framebuffer_mixed_samples extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_MODULATION_TABLE_ENABLE_NV dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of coverageModulationMode is any value other than VK_COVERAGE_MODULATION_MODE_NONE_NV, then vkCmdSetCoverageModulationTableEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-07646
  If the VK_NV_framebuffer_mixed_samples extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_MODULATION_TABLE_NV dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of coverageModulationTableEnable is VK_TRUE, then vkCmdSetCoverageModulationTableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-07647
  If the shadingRateImage feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_SHADING_RATE_IMAGE_ENABLE_NV dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetShadingRateImageEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectEXT-pipelineFragmentShadingRate-09238
  If the pipelineFragmentShadingRate feature is enabled, a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR dynamic state enabled, and the

current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetFragmentShadingRateKHR **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectEXT-None-07648
  If the representativeFragmentTest feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_REPRESENTATIVE_FRAGMENT_TEST_ENABLE_NV dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetRepresentativeFragmentTestEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectEXT-None-07649
  If the coverageReductionMode feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_REDUCTION_MODE_NV dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetCoverageReductionModeNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectEXT-pColorBlendEnables-07470
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT state enabled and the last call to vkCmdSetColorBlendEnableEXT set pColorBlendEnables for any attachment to VK_TRUE, then for those attachments in the subpass the corresponding image view's format features **must** contain VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT

- VUID-vkCmdDrawMeshTasksIndirectEXT-rasterizationSamples-07471
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, and the current subpass does not use any color and/or depth/stencil attachments, then the rasterizationSamples in the last call to vkCmdSetRasterizationSamplesEXT **must** follow the rules for a zero-attachment subpass

- VUID-vkCmdDrawMeshTasksIndirectEXT-samples-07472
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_MASK_EXT state enabled and the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state disabled, then the samples parameter in the last call to vkCmdSetSampleMaskEXT **must** be greater or equal to the VkPipelineMultisampleStateCreateInfo::rasterizationSamples parameter used to create the bound graphics pipeline

- VUID-vkCmdDrawMeshTasksIndirectEXT-samples-07473
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_MASK_EXT state and VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT states enabled, then the samples parameter in the last call to vkCmdSetSampleMaskEXT **must** be greater or equal to the rasterizationSamples parameter in the last call to vkCmdSetRasterizationSamplesEXT

- VUID-vkCmdDrawMeshTasksIndirectEXT-rasterizationSamples-07474
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, and neither the

VK_AMD_mixed_attachment_samples nor the VK_NV_framebuffer_mixed_samples extensions are enabled, then the rasterizationSamples in the last call to vkCmdSetRasterizationSamplesEXT **must** be the same as the current subpass color and/or depth/stencil attachments

- VUID-vkCmdDrawMeshTasksIndirectEXT-None-09211
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, or a shader object is bound to any graphics stage, and the current render pass instance includes a VkMultisampledRenderToSingleSampledInfoEXT structure with multisampledRenderToSingleSampledEnable equal to VK_TRUE, then the rasterizationSamples in the last call to vkCmdSetRasterizationSamplesEXT **must** be the same as the rasterizationSamples member of that structure

- VUID-vkCmdDrawMeshTasksIndirectEXT-firstAttachment-07476
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT dynamic state enabled then vkCmdSetColorBlendEnableEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the firstAttachment and attachmentCount parameters of vkCmdSetColorBlendEnableEXT calls **must** specify an enable for all active color attachments in the current subpass

- VUID-vkCmdDrawMeshTasksIndirectEXT-rasterizerDiscardEnable-09417
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetColorBlendEnableEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the firstAttachment and attachmentCount parameters of vkCmdSetColorBlendEnableEXT calls **must** specify an enable for all active color attachments in the current subpass

- VUID-vkCmdDrawMeshTasksIndirectEXT-firstAttachment-07477
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_EQUATION_EXT dynamic state enabled then vkCmdSetColorBlendEquationEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the firstAttachment and attachmentCount parameters of vkCmdSetColorBlendEquationEXT calls **must** specify the blend equations for all active color attachments in the current subpass where blending is enabled

- VUID-vkCmdDrawMeshTasksIndirectEXT-rasterizerDiscardEnable-09418
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and both the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE and there are color attachments bound, then vkCmdSetColorBlendEquationEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the firstAttachment and attachmentCount parameters of vkCmdSetColorBlendEquationEXT calls **must** specify the blend equations for all active color attachments in the current subpass where blending is enabled

- VUID-vkCmdDrawMeshTasksIndirectEXT-firstAttachment-07478

If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_WRITE_MASK_EXT dynamic state enabled then vkCmdSetColorWriteMaskEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the firstAttachment and attachmentCount parameters of vkCmdSetColorWriteMaskEXT calls **must** specify the color write mask for all active color attachments in the current subpass

- VUID-vkCmdDrawMeshTasksIndirectEXT-rasterizerDiscardEnable-09419
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetColorWriteMaskEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the firstAttachment and attachmentCount parameters of vkCmdSetColorWriteMaskEXT calls **must** specify the color write mask for all active color attachments in the current subpass

- VUID-vkCmdDrawMeshTasksIndirectEXT-firstAttachment-07479
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_ADVANCED_EXT dynamic state enabled then vkCmdSetColorBlendAdvancedEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the firstAttachment and attachmentCount parameters of vkCmdSetColorBlendAdvancedEXT calls **must** specify the advanced blend equations for all active color attachments in the current subpass where blending is enabled

- VUID-vkCmdDrawMeshTasksIndirectEXT-advancedBlendMaxColorAttachments-07480
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_ADVANCED_EXT and VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT dynamic states enabled and the last calls to vkCmdSetColorBlendEnableEXT and vkCmdSetColorBlendAdvancedEXT have enabled advanced blending, then the number of active color attachments in the current subpass **must** not exceed advancedBlendMaxColorAttachments

- VUID-vkCmdDrawMeshTasksIndirectEXT-primitivesGeneratedQueryWithNonZeroStreams-07481
  If the primitivesGeneratedQueryWithNonZeroStreams feature is not enabled and the VK_QUERY_TYPE_PRIMITIVES_GENERATED_EXT query is active, and the bound graphics pipeline was created with VK_DYNAMIC_STATE_RASTERIZATION_STREAM_EXT state enabled, the last call to vkCmdSetRasterizationStreamEXT **must** have set the rasterizationStream to zero

- VUID-vkCmdDrawMeshTasksIndirectEXT-sampleLocationsPerPixel-07482
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state enabled and the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state disabled, then the sampleLocationsPerPixel member of pSampleLocationsInfo in the last call to vkCmdSetSampleLocationsEXT **must** equal the rasterizationSamples member of the VkPipelineMultisampleStateCreateInfo structure the bound graphics pipeline has been created with

- VUID-vkCmdDrawMeshTasksIndirectEXT-sampleLocationsPerPixel-07483
  If the bound graphics pipeline state was created with the

VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state enabled and the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, then the sampleLocationsPerPixel member of pSampleLocationsInfo in the last call to vkCmdSetSampleLocationsEXT **must** equal the rasterizationSamples parameter of the last call to vkCmdSetRasterizationSamplesEXT

- VUID-vkCmdDrawMeshTasksIndirectEXT-sampleLocationsEnable-07484
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, or the bound graphics pipeline was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, and sampleLocationsEnable was VK_TRUE in the last call to vkCmdSetSampleLocationsEnableEXT, and the current subpass has a depth/stencil attachment, then that attachment **must** have been created with the VK_IMAGE_CREATE_SAMPLE_LOCATIONS_COMPATIBLE_DEPTH_BIT_EXT bit set
- VUID-vkCmdDrawMeshTasksIndirectEXT-sampleLocationsEnable-07485
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state enabled and the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, and if sampleLocationsEnable was VK_TRUE in the last call to vkCmdSetSampleLocationsEnableEXT, then the sampleLocationsInfo.sampleLocationGridSize.width in the last call to vkCmdSetSampleLocationsEXT **must** evenly divide VkMultisamplePropertiesEXT::sampleLocationGridSize.width as returned by vkGetPhysicalDeviceMultisamplePropertiesEXT with a samples parameter equaling rasterizationSamples
- VUID-vkCmdDrawMeshTasksIndirectEXT-sampleLocationsEnable-07486
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state enabled and the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, and if sampleLocationsEnable was VK_TRUE in the last call to vkCmdSetSampleLocationsEnableEXT, then the sampleLocationsInfo.sampleLocationGridSize.height in the last call to vkCmdSetSampleLocationsEXT **must** evenly divide VkMultisamplePropertiesEXT::sampleLocationGridSize.height as returned by vkGetPhysicalDeviceMultisamplePropertiesEXT with a samples parameter equaling rasterizationSamples
- VUID-vkCmdDrawMeshTasksIndirectEXT-sampleLocationsEnable-07487
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, and if sampleLocationsEnable was VK_TRUE in the last call to vkCmdSetSampleLocationsEnableEXT, the fragment shader code **must** not statically use the extended instruction InterpolateAtSample
- VUID-vkCmdDrawMeshTasksIndirectEXT-sampleLocationsEnable-07936
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state disabled and the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, the sampleLocationsEnable member of a VkPipelineSampleLocationsStateCreateInfoEXT::sampleLocationsEnable in the bound graphics pipeline is VK_TRUE or VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state

enabled, then, sampleLocationsInfo.sampleLocationGridSize.width **must** evenly divide VkMultisamplePropertiesEXT::sampleLocationGridSize.width as returned by vkGetPhysicalDeviceMultisamplePropertiesEXT with a samples parameter equaling the value of rasterizationSamples in the last call to vkCmdSetRasterizationSamplesEXT

- VUID-vkCmdDrawMeshTasksIndirectEXT-sampleLocationsEnable-07937
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state disabled and the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, the sampleLocationsEnable member of a VkPipelineSampleLocationsStateCreateInfoEXT::sampleLocationsEnable in the bound graphics pipeline is VK_TRUE or VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, then, sampleLocationsInfo.sampleLocationGridSize.height **must** evenly divide VkMultisamplePropertiesEXT::sampleLocationGridSize.height as returned by vkGetPhysicalDeviceMultisamplePropertiesEXT with a samples parameter equaling the value of rasterizationSamples in the last call to vkCmdSetRasterizationSamplesEXT

- VUID-vkCmdDrawMeshTasksIndirectEXT-sampleLocationsEnable-07938
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state disabled and the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, the sampleLocationsEnable member of a VkPipelineSampleLocationsStateCreateInfoEXT::sampleLocationsEnable in the bound graphics pipeline is VK_TRUE or VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, then, sampleLocationsInfo.sampleLocationsPerPixel **must** equal rasterizationSamples in the last call to vkCmdSetRasterizationSamplesEXT

- VUID-vkCmdDrawMeshTasksIndirectEXT-coverageModulationTableEnable-07488
  If a shader object is bound to any graphics stage or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COVERAGE_MODULATION_TABLE_ENABLE_NV state enabled, and the last call to vkCmdSetCoverageModulationTableEnableNV set coverageModulationTableEnable to VK_TRUE, then the coverageModulationTableCount parameter in the last call to vkCmdSetCoverageModulationTableNV **must** equal the current rasterizationSamples divided by the number of color samples in the current subpass

- VUID-vkCmdDrawMeshTasksIndirectEXT-rasterizationSamples-07489
  If the VK_NV_framebuffer_mixed_samples extension is enabled, and if current subpass has a depth/stencil attachment and depth test, stencil test, or depth bounds test are enabled in the currently bound pipeline state, then the current rasterizationSamples **must** be the same as the sample count of the depth/stencil attachment

- VUID-vkCmdDrawMeshTasksIndirectEXT-coverageToColorEnable-07490
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COVERAGE_TO_COLOR_ENABLE_NV state enabled and the last call to vkCmdSetCoverageToColorEnableNV set the coverageToColorEnable to VK_TRUE, then the current subpass **must** have a color attachment at the location selected by the last call to vkCmdSetCoverageToColorLocationNV coverageToColorLocation, with a VkFormat of VK_FORMAT_R8_UINT, VK_FORMAT_R8_SINT, VK_FORMAT_R16_UINT, VK_FORMAT_R16_SINT, VK_FORMAT_R32_UINT, or VK_FORMAT_R32_SINT

- VUID-vkCmdDrawMeshTasksIndirectEXT-rasterizerDiscardEnable-09420
  If the VK_NV_fragment_coverage_to_color extension is enabled, and a shader object is

bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the last call to vkCmdSetCoverageToColorEnableNV set the coverageToColorEnable to VK_TRUE, then the current subpass **must** have a color attachment at the location selected by the last call to vkCmdSetCoverageToColorLocationNV coverageToColorLocation, with a VkFormat of VK_FORMAT_R8_UINT, VK_FORMAT_R8_SINT, VK_FORMAT_R16_UINT, VK_FORMAT_R16_SINT, VK_FORMAT_R32_UINT, or VK_FORMAT_R32_SINT

- VUID-vkCmdDrawMeshTasksIndirectEXT-coverageReductionMode-07491
  If this VK_NV_coverage_reduction_mode extension is enabled, the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COVERAGE_TO_COLOR_ENABLE_NV and VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT states enabled, the current coverage reduction mode coverageReductionMode, then the current rasterizationSamples, and the sample counts for the color and depth/stencil attachments (if the subpass has them) **must** be a valid combination returned by vkGetPhysicalDeviceSupportedFramebufferMixedSamplesCombinationsNV

- VUID-vkCmdDrawMeshTasksIndirectEXT-viewportCount-07492
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, but not the VK_DYNAMIC_STATE_VIEWPORT_SWIZZLE_NV dynamic state enabled, then the bound graphics pipeline **must** have been created with VkPipelineViewportSwizzleStateCreateInfoNV ::viewportCount greater or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT

- VUID-vkCmdDrawMeshTasksIndirectEXT-viewportCount-07493
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT and VK_DYNAMIC_STATE_VIEWPORT_SWIZZLE_NV dynamic states enabled then the viewportCount parameter in the last call to vkCmdSetViewportSwizzleNV **must** be greater than or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT

- VUID-vkCmdDrawMeshTasksIndirectEXT-viewportCount-09421
  If the VK_NV_viewport_swizzle extension is enabled, and a shader object is bound to any graphics stage, then the viewportCount parameter in the last call to vkCmdSetViewportSwizzleNV **must** be greater than or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT

- VUID-vkCmdDrawMeshTasksIndirectEXT-rasterizationSamples-07494
  If the VK_NV_framebuffer_mixed_samples extension is enabled, and if the current subpass has any color attachments and rasterizationSamples of the last call to vkCmdSetRasterizationSamplesEXT is greater than the number of color samples, then the pipeline sampleShadingEnable **must** be VK_FALSE

- VUID-vkCmdDrawMeshTasksIndirectEXT-stippledLineEnable-07495
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT or VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT dynamic states enabled, and if the current stippledLineEnable state is VK_TRUE and the current lineRasterizationMode state is VK_LINE_RASTERIZATION_MODE_RECTANGULAR_KHR, then the stippledRectangularLines feature

**must** be enabled

- VUID-vkCmdDrawMeshTasksIndirectEXT-stippledLineEnable-07496
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT` or `VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT` dynamic states enabled, and if the current `stippledLineEnable` state is `VK_TRUE` and the current `lineRasterizationMode` state is `VK_LINE_RASTERIZATION_MODE_BRESENHAM_KHR`, then the `stippledBresenhamLines` feature **must** be enabled

- VUID-vkCmdDrawMeshTasksIndirectEXT-stippledLineEnable-07497
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT` or `VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT` dynamic states enabled, and if the current `stippledLineEnable` state is `VK_TRUE` and the current `lineRasterizationMode` state is `VK_LINE_RASTERIZATION_MODE_RECTANGULAR_SMOOTH_KHR`, then the `stippledSmoothLines` feature **must** be enabled

- VUID-vkCmdDrawMeshTasksIndirectEXT-stippledLineEnable-07498
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT` or `VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT` dynamic states enabled, and if the current `stippledLineEnable` state is `VK_TRUE` and the current `lineRasterizationMode` state is `VK_LINE_RASTERIZATION_MODE_DEFAULT_KHR`, then the `stippledRectangularLines` feature **must** be enabled and VkPhysicalDeviceLimits::`strictLines` **must** be `VK_TRUE`

- VUID-vkCmdDrawMeshTasksIndirectEXT-conservativePointAndLineRasterization-07499
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_CONSERVATIVE_RASTERIZATION_MODE_EXT` dynamic state enabled, `conservativePointAndLineRasterization` is not supported, and the effective primitive topology output by the last pre-rasterization shader stage is a line or point, then the `conservativeRasterizationMode` set by the last call to vkCmdSetConservativeRasterizationModeEXT **must** be `VK_CONSERVATIVE_RASTERIZATION_MODE_DISABLED_EXT`

- VUID-vkCmdDrawMeshTasksIndirectEXT-stage-07073
  If the currently bound pipeline was created with the VkPipelineShaderStageCreateInfo::`stage` member of an element of VkGraphicsPipelineCreateInfo::`pStages` set to `VK_SHADER_STAGE_VERTEX_BIT`, `VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT`, `VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT` or `VK_SHADER_STAGE_GEOMETRY_BIT`, then Mesh Shader Queries **must** not be active

- VUID-vkCmdDrawMeshTasksIndirectEXT-None-08877
  If a shader object is bound to the `VK_SHADER_STAGE_FRAGMENT_BIT` stage or a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_ATTACHMENT_FEEDBACK_LOOP_ENABLE_EXT` dynamic state enabled, and the current value of `rasterizerDiscardEnable` is `VK_FALSE`, then vkCmdSetAttachmentFeedbackLoopEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectEXT-None-07850
  If dynamic state was inherited from

VkCommandBufferInheritanceViewportScissorInfoNV, it **must** be set in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectEXT-None-08684
  If there is no bound graphics pipeline, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_VERTEX_BIT
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-08685
  If there is no bound graphics pipeline, and the tessellationShader feature is enabled, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-08686
  If there is no bound graphics pipeline, and the tessellationShader feature is enabled, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-08687
  If there is no bound graphics pipeline, and the geometryShader feature is enabled, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_GEOMETRY_BIT
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-08688
  If there is no bound graphics pipeline, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_FRAGMENT_BIT
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-08689
  If there is no bound graphics pipeline, and the taskShader feature is enabled, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_TASK_BIT_EXT
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-08690
  If there is no bound graphics pipeline, and the meshShader feature is enabled, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_MESH_BIT_EXT
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-08693
  If there is no bound graphics pipeline, and at least one of the taskShader and meshShader features is enabled, one of the VK_SHADER_STAGE_VERTEX_BIT or VK_SHADER_STAGE_MESH_BIT_EXT stages **must** have a valid VkShaderEXT bound, and the other **must** have no VkShaderEXT bound
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-08694
  If there is no bound graphics pipeline, and both the taskShader and meshShader features are enabled, and a valid VkShaderEXT is bound the to the VK_SHADER_STAGE_MESH_BIT_EXT stage, and that VkShaderEXT was created without the VK_SHADER_CREATE_NO_TASK_SHADER_BIT_EXT flag, a valid VkShaderEXT **must** be bound to the VK_SHADER_STAGE_TASK_BIT_EXT stage
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-08695
  If there is no bound graphics pipeline, and both the taskShader and meshShader features are enabled, and a valid VkShaderEXT is bound the to the VK_SHADER_STAGE_MESH_BIT_EXT stage, and that VkShaderEXT was created with the VK_SHADER_CREATE_NO_TASK_SHADER_BIT_EXT flag, there **must** be no VkShaderEXT bound to the VK_SHADER_STAGE_TASK_BIT_EXT stage

- VUID-vkCmdDrawMeshTasksIndirectEXT-None-08696
  If there is no bound graphics pipeline, and a valid VkShaderEXT is bound to the VK_SHADER_STAGE_VERTEX_BIT stage, there **must** be no VkShaderEXT bound to either the VK_SHADER_STAGE_TASK_BIT_EXT stage or the VK_SHADER_STAGE_MESH_BIT_EXT stage
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-08698
  If any graphics shader is bound which was created with the VK_SHADER_CREATE_LINK_STAGE_BIT_EXT flag, then all shaders created with the VK_SHADER_CREATE_LINK_STAGE_BIT_EXT flag in the same vkCreateShadersEXT call **must** also be bound
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-08699
  If any graphics shader is bound which was created with the VK_SHADER_CREATE_LINK_STAGE_BIT_EXT flag, any stages in between stages whose shaders which did not create a shader with the VK_SHADER_CREATE_LINK_STAGE_BIT_EXT flag as part of the same vkCreateShadersEXT call **must** not have any VkShaderEXT bound
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-08878
  All bound graphics shader objects **must** have been created with identical or identically defined push constant ranges
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-08879
  All bound graphics shader objects **must** have been created with identical or identically defined arrays of descriptor set layouts
- VUID-vkCmdDrawMeshTasksIndirectEXT-colorAttachmentCount-09372
  If the current render pass instance was begun with vkCmdBeginRenderingKHR and a VkRenderingInfo::colorAttachmentCount equal to 1, a color attachment with a resolve mode of VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID, and a fragment shader is bound, it **must** not declare the DepthReplacing or StencilRefReplacingEXT execution modes
- VUID-vkCmdDrawMeshTasksIndirectEXT-pDynamicStates-08715
  If the bound graphics pipeline state includes a fragment shader stage, was created with VK_DYNAMIC_STATE_DEPTH_WRITE_ENABLE set in VkPipelineDynamicStateCreateInfo::pDynamicStates, and the fragment shader declares the EarlyFragmentTests execution mode and uses OpDepthAttachmentReadEXT, the depthWriteEnable parameter in the last call to vkCmdSetDepthWriteEnableEXT **must** be VK_FALSE
- VUID-vkCmdDrawMeshTasksIndirectEXT-pDynamicStates-08716
  If the bound graphics pipeline state includes a fragment shader stage, was created with VK_DYNAMIC_STATE_STENCIL_WRITE_MASK set in VkPipelineDynamicStateCreateInfo::pDynamicStates, and the fragment shader declares the EarlyFragmentTests execution mode and uses OpStencilAttachmentReadEXT, the writeMask parameter in the last call to vkCmdSetStencilWriteMask **must** be 0
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-09116
  If a shader object is bound to any graphics stage or the currently bound graphics pipeline was created with VK_DYNAMIC_STATE_COLOR_WRITE_MASK_EXT, and the format of any color attachment is VK_FORMAT_E5B9G9R9_UFLOAT_PACK32, the corresponding element of the pColorWriteMasks parameter of vkCmdSetColorWriteMaskEXT **must** either include all of VK_COLOR_COMPONENT_R_BIT, VK_COLOR_COMPONENT_G_BIT, and VK_COLOR_COMPONENT_B_BIT, or none of them

- VUID-vkCmdDrawMeshTasksIndirectEXT-maxFragmentDualSrcAttachments-09239
  If blending is enabled for any attachment where either the source or destination blend factors for that attachment use the secondary color input, the maximum value of `Location` for any output attachment statically used in the `Fragment Execution Model` executed by this command **must** be less than `maxFragmentDualSrcAttachments`
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-09548
  If the current render pass was begun with vkCmdBeginRenderingKHR, and there is no shader object bound to any graphics stage, the value of each element of VkRenderingAttachmentLocationInfoKHR::`pColorAttachmentLocations` set by vkCmdSetRenderingAttachmentLocationsKHR **must** match the value set for the corresponding element in the currently bound pipeline
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-09549
  If the current render pass was begun with vkCmdBeginRenderingKHR, and there is no shader object bound to any graphics stage, input attachment index mappings in the currently bound pipeline **must** match those set for the current render pass instance via VkRenderingInputAttachmentIndexInfoKHR
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-09642
  If the current render pass was begun with vkCmdBeginRenderingKHR with the `VK_RENDERING_ENABLE_LEGACY_DITHERING_BIT_EXT` flag, the bound graphics pipeline **must** have been created with `VK_PIPELINE_CREATE_2_ENABLE_LEGACY_DITHERING_BIT_EXT`
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-09643
  If the bound graphics pipeline was created with `VK_PIPELINE_CREATE_2_ENABLE_LEGACY_DITHERING_BIT_EXT`, the current render pass **must** have begun with vkCmdBeginRenderingKHR with the `VK_RENDERING_ENABLE_LEGACY_DITHERING_BIT_EXT` flag
- VUID-vkCmdDrawMeshTasksIndirectEXT-stage-06480
  The bound graphics pipeline **must** not have been created with the VkPipelineShaderStageCreateInfo::`stage` member of an element of VkGraphicsPipelineCreateInfo::`pStages` set to `VK_SHADER_STAGE_VERTEX_BIT`, `VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT`, `VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT` or `VK_SHADER_STAGE_GEOMETRY_BIT`
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-07074
  Transform Feedback Queries **must** not be active
- VUID-vkCmdDrawMeshTasksIndirectEXT-None-07075
  Primitives Generated Queries **must** not be active
- VUID-vkCmdDrawMeshTasksIndirectEXT-pipelineStatistics-07076
  The `pipelineStatistics` member used to create any active Pipeline Statistics Query **must** not contain `VK_QUERY_PIPELINE_STATISTIC_INPUT_ASSEMBLY_VERTICES_BIT`, `VK_QUERY_PIPELINE_STATISTIC_INPUT_ASSEMBLY_PRIMITIVES_BIT`, `VK_QUERY_PIPELINE_STATISTIC_VERTEX_SHADER_INVOCATIONS_BIT`, `VK_QUERY_PIPELINE_STATISTIC_GEOMETRY_SHADER_INVOCATIONS_BIT`, `VK_QUERY_PIPELINE_STATISTIC_GEOMETRY_SHADER_PRIMITIVES_BIT`, `VK_QUERY_PIPELINE_STATISTIC_CLIPPING_INVOCATIONS_BIT`, `VK_QUERY_PIPELINE_STATISTIC_CLIPPING_PRIMITIVES_BIT`,

VK_QUERY_PIPELINE_STATISTIC_TESSELLATION_CONTROL_SHADER_PATCHES_BIT, or VK_QUERY_PIPELINE_STATISTIC_TESSELLATION_EVALUATION_SHADER_INVOCATIONS_BIT

- VUID-vkCmdDrawMeshTasksIndirectEXT-buffer-02708
  If buffer is non-sparse then it **must** be bound completely and contiguously to a single VkDeviceMemory object
- VUID-vkCmdDrawMeshTasksIndirectEXT-buffer-02709
  buffer **must** have been created with the VK_BUFFER_USAGE_INDIRECT_BUFFER_BIT bit set
- VUID-vkCmdDrawMeshTasksIndirectEXT-offset-02710
  offset **must** be a multiple of 4
- VUID-vkCmdDrawMeshTasksIndirectEXT-commandBuffer-02711
  commandBuffer **must** not be a protected command buffer
- VUID-vkCmdDrawMeshTasksIndirectEXT-drawCount-02718
  If the multiDrawIndirect feature is not enabled, drawCount **must** be 0 or 1
- VUID-vkCmdDrawMeshTasksIndirectEXT-drawCount-02719
  drawCount **must** be less than or equal to VkPhysicalDeviceLimits::maxDrawIndirectCount
- VUID-vkCmdDrawMeshTasksIndirectEXT-drawCount-07088
  If drawCount is greater than 1, stride **must** be a multiple of 4 and **must** be greater than or equal to sizeof(VkDrawMeshTasksIndirectCommandEXT)
- VUID-vkCmdDrawMeshTasksIndirectEXT-drawCount-07089
  If drawCount is equal to 1, (offset + sizeof(VkDrawMeshTasksIndirectCommandEXT)) **must** be less than or equal to the size of buffer
- VUID-vkCmdDrawMeshTasksIndirectEXT-drawCount-07090
  If drawCount is greater than 1, (stride × (drawCount - 1) + offset + sizeof(VkDrawMeshTasksIndirectCommandEXT)) **must** be less than or equal to the size of buffer
- VUID-vkCmdDrawMeshTasksIndirectEXT-MeshEXT-07091
  The current pipeline bound to VK_PIPELINE_BIND_POINT_GRAPHICS **must** contain a shader stage using the MeshEXT Execution Model

## Valid Usage (Implicit)

- VUID-vkCmdDrawMeshTasksIndirectEXT-commandBuffer-parameter
  commandBuffer **must** be a valid VkCommandBuffer handle
- VUID-vkCmdDrawMeshTasksIndirectEXT-buffer-parameter
  buffer **must** be a valid VkBuffer handle
- VUID-vkCmdDrawMeshTasksIndirectEXT-commandBuffer-recording
  commandBuffer **must** be in the recording state
- VUID-vkCmdDrawMeshTasksIndirectEXT-commandBuffer-cmdpool
  The VkCommandPool that commandBuffer was allocated from **must** support graphics operations

- VUID-vkCmdDrawMeshTasksIndirectEXT-renderpass
  This command **must** only be called inside of a render pass instance
- VUID-vkCmdDrawMeshTasksIndirectEXT-videocoding
  This command **must** only be called outside of a video coding scope
- VUID-vkCmdDrawMeshTasksIndirectEXT-commonparent
  Both of `buffer`, and `commandBuffer` **must** have been created, allocated, or retrieved from the same VkDevice

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Inside | Outside | Graphics | Action |

The `VkDrawMeshTasksIndirectCommandEXT` structure is defined as:

```
// Provided by VK_EXT_mesh_shader
typedef struct VkDrawMeshTasksIndirectCommandEXT {
    uint32_t    groupCountX;
    uint32_t    groupCountY;
    uint32_t    groupCountZ;
} VkDrawMeshTasksIndirectCommandEXT;
```

- `groupCountX` is the number of local workgroups to dispatch in the X dimension.
- `groupCountY` is the number of local workgroups to dispatch in the Y dimension.
- `groupCountZ` is the number of local workgroups to dispatch in the Z dimension.

The members of `VkDrawMeshTasksIndirectCommandEXT` have the same meaning as the similarly named parameters of vkCmdDrawMeshTasksEXT.

## Valid Usage

- VUID-VkDrawMeshTasksIndirectCommandEXT-TaskEXT-07322
  If the current pipeline bound to `VK_PIPELINE_BIND_POINT_GRAPHICS` contains a shader using the `TaskEXT Execution Model`, `groupCountX` **must** be less than or equal to

VkPhysicalDeviceMeshShaderPropertiesEXT::maxTaskWorkGroupCount[0]

- VUID-VkDrawMeshTasksIndirectCommandEXT-TaskEXT-07323
  If the current pipeline bound to VK_PIPELINE_BIND_POINT_GRAPHICS contains a shader using the TaskEXT Execution Model, groupCountY **must** be less than or equal to VkPhysicalDeviceMeshShaderPropertiesEXT::maxTaskWorkGroupCount[1]
- VUID-VkDrawMeshTasksIndirectCommandEXT-TaskEXT-07324
  If the current pipeline bound to VK_PIPELINE_BIND_POINT_GRAPHICS contains a shader using the TaskEXT Execution Model, groupCountZ **must** be less than or equal to VkPhysicalDeviceMeshShaderPropertiesEXT::maxTaskWorkGroupCount[2]
- VUID-VkDrawMeshTasksIndirectCommandEXT-TaskEXT-07325
  If the current pipeline bound to VK_PIPELINE_BIND_POINT_GRAPHICS contains a shader using the TaskEXT Execution Model, The product of groupCountX, groupCountY and groupCountZ **must** be less than or equal to VkPhysicalDeviceMeshShaderPropertiesEXT::maxTaskWorkGroupTotalCount
- VUID-VkDrawMeshTasksIndirectCommandEXT-TaskEXT-07326
  If the current pipeline bound to VK_PIPELINE_BIND_POINT_GRAPHICS does not contain a shader using the TaskEXT Execution Model, groupCountX **must** be less than or equal to VkPhysicalDeviceMeshShaderPropertiesEXT::maxMeshWorkGroupCount[0]
- VUID-VkDrawMeshTasksIndirectCommandEXT-TaskEXT-07327
  If the current pipeline bound to VK_PIPELINE_BIND_POINT_GRAPHICS does not contain a shader using the TaskEXT Execution Model, groupCountY **must** be less than or equal to VkPhysicalDeviceMeshShaderPropertiesEXT::maxMeshWorkGroupCount[1]
- VUID-VkDrawMeshTasksIndirectCommandEXT-TaskEXT-07328
  If the current pipeline bound to VK_PIPELINE_BIND_POINT_GRAPHICS does not contain a shader using the TaskEXT Execution Model, groupCountZ **must** be less than or equal to VkPhysicalDeviceMeshShaderPropertiesEXT::maxMeshWorkGroupCount[2]
- VUID-VkDrawMeshTasksIndirectCommandEXT-TaskEXT-07329
  If the current pipeline bound to VK_PIPELINE_BIND_POINT_GRAPHICS does not contain a shader using the TaskEXT Execution Model, The product of groupCountX, groupCountY and groupCountZ **must** be less than or equal to VkPhysicalDeviceMeshShaderPropertiesEXT::maxMeshWorkGroupTotalCount

To record an indirect mesh tasks drawing command with the draw count sourced from a buffer, call:

```
// Provided by VK_EXT_mesh_shader
void vkCmdDrawMeshTasksIndirectCountEXT(
    VkCommandBuffer                             commandBuffer,
    VkBuffer                                    buffer,
    VkDeviceSize                                offset,
    VkBuffer                                    countBuffer,
    VkDeviceSize                                countBufferOffset,
    uint32_t                                    maxDrawCount,
    uint32_t                                    stride);
```

- commandBuffer is the command buffer into which the command is recorded.
- buffer is the buffer containing draw parameters.
- offset is the byte offset into buffer where parameters begin.
- countBuffer is the buffer containing the draw count.
- countBufferOffset is the byte offset into countBuffer where the draw count begins.
- maxDrawCount specifies the maximum number of draws that will be executed. The actual number of executed draw calls is the minimum of the count specified in countBuffer and maxDrawCount.
- stride is the byte stride between successive sets of draw parameters.

vkCmdDrawMeshTasksIndirectCountEXT behaves similarly to vkCmdDrawMeshTasksIndirectEXT except that the draw count is read by the device from a buffer during execution. The command will read an unsigned 32-bit integer from countBuffer located at countBufferOffset and use this as the draw count.

### Valid Usage

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-magFilter-04553
  If a VkSampler created with magFilter or minFilter equal to VK_FILTER_LINEAR, reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE, and compareEnable equal to VK_FALSE is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-magFilter-09598
  If a VkSampler created with magFilter or minFilter equal to VK_FILTER_LINEAR and reductionMode equal to either VK_SAMPLER_REDUCTION_MODE_MIN or VK_SAMPLER_REDUCTION_MODE_MAX is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-mipmapMode-04770
  If a VkSampler created with mipmapMode equal to VK_SAMPLER_MIPMAP_MODE_LINEAR, reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE, and compareEnable equal to VK_FALSE is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-mipmapMode-09599
  If a VkSampler created with mipmapMode equal to VK_SAMPLER_MIPMAP_MODE_LINEAR and reductionMode equal to either VK_SAMPLER_REDUCTION_MODE_MIN or VK_SAMPLER_REDUCTION_MODE_MAX is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-unnormalizedCoordinates-09635
  If a VkSampler created with unnormalizedCoordinates equal to VK_TRUE is used to sample a VkImageView as a result of this command, then the image view's levelCount and layerCount **must** be 1

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-unnormalizedCoordinates-09636
  If a VkSampler created with unnormalizedCoordinates equal to VK_TRUE is used to sample a VkImageView as a result of this command, then the image view's viewType **must** be VK_IMAGE_VIEW_TYPE_1D or VK_IMAGE_VIEW_TYPE_2D
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-06479
  If a VkImageView is sampled with depth comparison, the image view's format features **must** contain VK_FORMAT_FEATURE_2_SAMPLED_IMAGE_DEPTH_COMPARISON_BIT
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-02691
  If a VkImageView is accessed using atomic operations as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_STORAGE_IMAGE_ATOMIC_BIT
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-07888
  If a VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER descriptor is accessed using atomic operations as a result of this command, then the storage texel buffer's format features **must** contain VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_ATOMIC_BIT
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-02692
  If a VkImageView is sampled with VK_FILTER_CUBIC_EXT as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_CUBIC_BIT_EXT
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-02693
  If the VK_EXT_filter_cubic extension is not enabled and any VkImageView is sampled with VK_FILTER_CUBIC_EXT as a result of this command, it **must** not have a VkImageViewType of VK_IMAGE_VIEW_TYPE_3D, VK_IMAGE_VIEW_TYPE_CUBE, or VK_IMAGE_VIEW_TYPE_CUBE_ARRAY
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-filterCubic-02694
  Any VkImageView being sampled with VK_FILTER_CUBIC_EXT as a result of this command **must** have a VkImageViewType and format that supports cubic filtering, as specified by VkFilterCubicImageViewImageFormatPropertiesEXT::filterCubic returned by vkGetPhysicalDeviceImageFormatProperties2
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-filterCubicMinmax-02695
  Any VkImageView being sampled with VK_FILTER_CUBIC_EXT with a reduction mode of either VK_SAMPLER_REDUCTION_MODE_MIN or VK_SAMPLER_REDUCTION_MODE_MAX as a result of this command **must** have a VkImageViewType and format that supports cubic filtering together with minmax filtering, as specified by VkFilterCubicImageViewImageFormatPropertiesEXT::filterCubicMinmax returned by vkGetPhysicalDeviceImageFormatProperties2
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-cubicRangeClamp-09212
  If the cubicRangeClamp feature is not enabled, then any VkImageView being sampled with VK_FILTER_CUBIC_EXT as a result of this command **must** not have a VkSamplerReductionModeCreateInfo::reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE_RANGECLAMP_QCOM
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-reductionMode-09213
  Any VkImageView being sampled with a VkSamplerReductionModeCreateInfo::reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE_RANGECLAMP_QCOM as a result of this command **must** sample with VK_FILTER_CUBIC_EXT

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-selectableCubicWeights-09214
  If the `selectableCubicWeights` feature is not enabled, then any VkImageView being sampled with `VK_FILTER_CUBIC_EXT` as a result of this command **must** have VkSamplerCubicWeightsCreateInfoQCOM::`cubicWeights` equal to `VK_CUBIC_FILTER_WEIGHTS_CATMULL_ROM_QCOM`
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-flags-02696
  Any VkImage created with a VkImageCreateInfo::`flags` containing `VK_IMAGE_CREATE_CORNER_SAMPLED_BIT_NV` sampled as a result of this command **must** only be sampled using a VkSamplerAddressMode of `VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_EDGE`
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-OpTypeImage-07027
  For any VkImageView being written as a storage image where the image format field of the `OpTypeImage` is `Unknown`, the view's format features **must** contain `VK_FORMAT_FEATURE_2_STORAGE_WRITE_WITHOUT_FORMAT_BIT`
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-OpTypeImage-07028
  For any VkImageView being read as a storage image where the image format field of the `OpTypeImage` is `Unknown`, the view's format features **must** contain `VK_FORMAT_FEATURE_2_STORAGE_READ_WITHOUT_FORMAT_BIT`
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-OpTypeImage-07029
  For any VkBufferView being written as a storage texel buffer where the image format field of the `OpTypeImage` is `Unknown`, the view's buffer features **must** contain `VK_FORMAT_FEATURE_2_STORAGE_WRITE_WITHOUT_FORMAT_BIT`
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-OpTypeImage-07030
  Any VkBufferView being read as a storage texel buffer where the image format field of the `OpTypeImage` is `Unknown` then the view's buffer features **must** contain `VK_FORMAT_FEATURE_2_STORAGE_READ_WITHOUT_FORMAT_BIT`
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-08600
  For each set *n* that is statically used by a bound shader, a descriptor set **must** have been bound to *n* at the same pipeline bind point, with a VkPipelineLayout that is compatible for set *n*, with the VkPipelineLayout used to create the current VkPipeline or the VkDescriptorSetLayout array used to create the current VkShaderEXT , as described in Pipeline Layout Compatibility
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-08601
  For each push constant that is statically used by a bound shader, a push constant value **must** have been set for the same pipeline bind point, with a VkPipelineLayout that is compatible for push constants, with the VkPipelineLayout used to create the current VkPipeline or the VkDescriptorSetLayout array used to create the current VkShaderEXT , as described in Pipeline Layout Compatibility
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-10068
  For each array of resources that is used by a bound shader, the indices used to access members of the array **must** be less than the descriptor count for the identified binding in the descriptor sets used by this command
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-maintenance4-08602
  If the `maintenance4` feature is not enabled, then for each push constant that is statically used by a bound shader, a push constant value **must** have been set for the same pipeline

bind point, with a VkPipelineLayout that is compatible for push constants, with the VkPipelineLayout used to create the current VkPipeline or the VkDescriptorSetLayout and VkPushConstantRange arrays used to create the current VkShaderEXT , as described in Pipeline Layout Compatibility

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-08114
  Descriptors in each bound descriptor set, specified via vkCmdBindDescriptorSets, **must** be valid as described by descriptor validity if they are statically used by the VkPipeline bound to the pipeline bind point used by this command and the bound VkPipeline was not created with `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-08115
  If the descriptors used by the VkPipeline bound to the pipeline bind point were specified via vkCmdBindDescriptorSets, the bound VkPipeline **must** have been created without `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-08116
  Descriptors in bound descriptor buffers, specified via vkCmdSetDescriptorBufferOffsetsEXT, **must** be valid if they are dynamically used by the VkPipeline bound to the pipeline bind point used by this command and the bound VkPipeline was created with `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-08604
  Descriptors in bound descriptor buffers, specified via vkCmdSetDescriptorBufferOffsetsEXT, **must** be valid if they are dynamically used by any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-08117
  If the descriptors used by the VkPipeline bound to the pipeline bind point were specified via vkCmdSetDescriptorBufferOffsetsEXT, the bound VkPipeline **must** have been created with `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-08119
  If a descriptor is dynamically used with a VkPipeline created with `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`, the descriptor memory **must** be resident
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-08605
  If a descriptor is dynamically used with a VkShaderEXT created with a `VkDescriptorSetLayout` that was created with `VK_DESCRIPTOR_SET_LAYOUT_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`, the descriptor memory **must** be resident
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-08606
  If the `shaderObject` feature is not enabled, a valid pipeline **must** be bound to the pipeline bind point used by this command
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-08608
  If a pipeline is bound to the pipeline bind point used by this command, there **must** not have been any calls to dynamic state setting commands for any state specified statically in the VkPipeline object bound to the pipeline bind point used by this command, since that pipeline was bound
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-08609

If the VkPipeline object bound to the pipeline bind point used by this command or any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a VkSampler object that uses unnormalized coordinates, that sampler **must** not be used to sample from any VkImage with a VkImageView of the type VK_IMAGE_VIEW_TYPE_3D, VK_IMAGE_VIEW_TYPE_CUBE, VK_IMAGE_VIEW_TYPE_1D_ARRAY, VK_IMAGE_VIEW_TYPE_2D_ARRAY or VK_IMAGE_VIEW_TYPE_CUBE_ARRAY, in any shader stage

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-08610
  If the VkPipeline object bound to the pipeline bind point used by this command or any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a VkSampler object that uses unnormalized coordinates, that sampler **must** not be used with any of the SPIR-V OpImageSample* or OpImageSparseSample* instructions with ImplicitLod, Dref or Proj in their name, in any shader stage
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-08611
  If the VkPipeline object bound to the pipeline bind point used by this command or any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a VkSampler object that uses unnormalized coordinates, that sampler **must** not be used with any of the SPIR-V OpImageSample* or OpImageSparseSample* instructions that includes a LOD bias or any offset values, in any shader stage
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-08607
  If the shaderObject is enabled, either a valid pipeline **must** be bound to the pipeline bind point used by this command, or a valid combination of valid and VK_NULL_HANDLE shader objects **must** be bound to every supported shader stage corresponding to the pipeline bind point used by this command
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-uniformBuffers-06935
  If any stage of the VkPipeline object bound to the pipeline bind point used by this command accesses a uniform buffer, and that stage was created without enabling either VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_EXT or VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_2_EXT for uniformBuffers, and the robustBufferAccess feature is not enabled, that stage **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-08612
  If the robustBufferAccess feature is not enabled, and any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a uniform buffer, it **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-storageBuffers-06936
  If any stage of the VkPipeline object bound to the pipeline bind point used by this command accesses a storage buffer, and that stage was created without enabling either VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_EXT or VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_2_EXT for storageBuffers, and the robustBufferAccess feature is not enabled, that stage **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-08613

If the robustBufferAccess feature is not enabled, and any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a storage buffer, it **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-commandBuffer-02707
  If commandBuffer is an unprotected command buffer and protectedNoFault is not supported, any resource accessed by bound shaders **must** not be a protected resource
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-06550
  If a bound shader accesses a VkSampler or VkImageView object that enables sampler $Y'C_BC_R$ conversion, that object **must** only be used with OpImageSample* or OpImageSparseSample* instructions
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-ConstOffset-06551
  If a bound shader accesses a VkSampler or VkImageView object that enables sampler $Y'C_BC_R$ conversion, that object **must** not use the ConstOffset and Offset operands
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-viewType-07752
  If a VkImageView is accessed as a result of this command, then the image view's viewType **must** match the Dim operand of the OpTypeImage as described in Compatibility Between SPIR-V Image Dimensions and Vulkan ImageView Types
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-format-07753
  If a VkImageView is accessed as a result of this command, then the numeric type of the image view's format and the Sampled Type operand of the OpTypeImage **must** match
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-OpImageWrite-08795
  If a VkImageView created with a format other than VK_FORMAT_A8_UNORM_KHR is accessed using OpImageWrite as a result of this command, then the Type of the Texel operand of that instruction **must** have at least as many components as the image view's format
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-OpImageWrite-08796
  If a VkImageView created with the format VK_FORMAT_A8_UNORM_KHR is accessed using OpImageWrite as a result of this command, then the Type of the Texel operand of that instruction **must** have four components
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-OpImageWrite-04469
  If a VkBufferView is accessed using OpImageWrite as a result of this command, then the Type of the Texel operand of that instruction **must** have at least as many components as the buffer view's format
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-SampledType-04470
  If a VkImageView with a VkFormat that has a 64-bit component width is accessed as a result of this command, the SampledType of the OpTypeImage operand of that instruction **must** have a Width of 64
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-SampledType-04471
  If a VkImageView with a VkFormat that has a component width less than 64-bit is accessed as a result of this command, the SampledType of the OpTypeImage operand of that instruction **must** have a Width of 32
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-SampledType-04472
  If a VkBufferView with a VkFormat that has a 64-bit component width is accessed as a

result of this command, the SampledType of the OpTypeImage operand of that instruction **must** have a Width of 64

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-SampledType-04473
  If a VkBufferView with a VkFormat that has a component width less than 64-bit is accessed as a result of this command, the SampledType of the OpTypeImage operand of that instruction **must** have a Width of 32
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-sparseImageInt64Atomics-04474
  If the sparseImageInt64Atomics feature is not enabled, VkImage objects created with the VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT flag **must** not be accessed by atomic instructions through an OpTypeImage with a SampledType with a Width of 64 by this command
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-sparseImageInt64Atomics-04475
  If the sparseImageInt64Atomics feature is not enabled, VkBuffer objects created with the VK_BUFFER_CREATE_SPARSE_RESIDENCY_BIT flag **must** not be accessed by atomic instructions through an OpTypeImage with a SampledType with a Width of 64 by this command
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-OpImageWeightedSampleQCOM-06971
  If OpImageWeightedSampleQCOM is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_WEIGHT_SAMPLED_IMAGE_BIT_QCOM
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-OpImageWeightedSampleQCOM-06972
  If OpImageWeightedSampleQCOM uses a VkImageView as a sample weight image as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_WEIGHT_IMAGE_BIT_QCOM
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-OpImageBoxFilterQCOM-06973
  If OpImageBoxFilterQCOM is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_BOX_FILTER_SAMPLED_BIT_QCOM
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-OpImageBlockMatchSSDQCOM-06974
  If OpImageBlockMatchSSDQCOM is used to read from an VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-OpImageBlockMatchSADQCOM-06975
  If OpImageBlockMatchSADQCOM is used to read from an VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-OpImageBlockMatchSADQCOM-06976
  If OpImageBlockMatchSADQCOM or OpImageBlockMatchSSDQCOM is used to read from a reference image as result of this command, then the specified reference coordinates **must** not fail integer texel coordinate validation
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-OpImageWeightedSampleQCOM-06977
  If OpImageWeightedSampleQCOM, OpImageBoxFilterQCOM, OpImageBlockMatchWindowSSDQCOM, OpImageBlockMatchWindowSADQCOM, OpImageBlockMatchGatherSSDQCOM, OpImageBlockMatchGatherSADQCOM, OpImageBlockMatchSSDQCOM, or OpImageBlockMatchSADQCOM uses a VkSampler as a result of this command, then the sampler **must** have been created with VK_SAMPLER_CREATE_IMAGE_PROCESSING_BIT_QCOM

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-OpImageWeightedSampleQCOM-06978
  If any command other than `OpImageWeightedSampleQCOM`, `OpImageBoxFilterQCOM`, `OpImageBlockMatchWindowSSDQCOM`, `OpImageBlockMatchWindowSADQCOM`, `OpImageBlockMatchGatherSSDQCOM`, `OpImageBlockMatchGatherSADQCOM`, `OpImageBlockMatchSSDQCOM`, or `OpImageBlockMatchSADQCOM` uses a VkSampler as a result of this command, then the sampler **must** not have been created with `VK_SAMPLER_CREATE_IMAGE_PROCESSING_BIT_QCOM`
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-OpImageBlockMatchWindow-09215
  If a `OpImageBlockMatchWindow*QCOM` or `OpImageBlockMatchGather*QCOM` instruction is used to read from an VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM`
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-OpImageBlockMatchWindow-09216
  If a `OpImageBlockMatchWindow*QCOM` or `OpImageBlockMatchGather*QCOM` instruction is used to read from an VkImageView as a result of this command, then the image view's format **must** be a single-component format
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-OpImageBlockMatchWindow-09217
  If a `OpImageBlockMatchWindow*QCOM` or `OpImageBlockMatchGather*QCOM` read from a reference image as result of this command, then the specified reference coordinates **must** not fail integer texel coordinate validation
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-07288
  Any shader invocation executed by this command **must** terminate
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-09600
  If a descriptor with type equal to any of `VK_DESCRIPTOR_TYPE_SAMPLE_WEIGHT_IMAGE_QCOM`, `VK_DESCRIPTOR_TYPE_BLOCK_MATCH_IMAGE_QCOM`, `VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE`, `VK_DESCRIPTOR_TYPE_STORAGE_IMAGE`, or `VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT` is accessed as a result of this command, the image subresource identified by that descriptor **must** be in the image layout identified when the descriptor was written
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-renderPass-02684
  The current render pass **must** be compatible with the `renderPass` member of the `VkGraphicsPipelineCreateInfo` structure specified when creating the `VkPipeline` bound to `VK_PIPELINE_BIND_POINT_GRAPHICS`
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-subpass-02685
  The subpass index of the current render pass **must** be equal to the `subpass` member of the `VkGraphicsPipelineCreateInfo` structure specified when creating the `VkPipeline` bound to `VK_PIPELINE_BIND_POINT_GRAPHICS`
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-07748
  If any shader statically accesses an input attachment, a valid descriptor **must** be bound to the pipeline via a descriptor set
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-OpTypeImage-07468
  If any shader executed by this pipeline accesses an `OpTypeImage` variable with a `Dim` operand of `SubpassData`, it **must** be decorated with an `InputAttachmentIndex` that corresponds to a valid input attachment in the current subpass
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-07469
  Input attachment views accessed in a subpass **must** be created with the same VkFormat

as the corresponding subpass definition, and be created with a VkImageView that is compatible with the attachment referenced by the subpass' pInputAttachments [InputAttachmentIndex] in the currently bound VkFramebuffer as specified by Fragment Input Attachment Compatibility

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-pDepthInputAttachmentIndex-09595
  Input attachment views accessed in a dynamic render pass with a InputAttachmentIndex referenced by VkRenderingInputAttachmentIndexInfoKHR, or no InputAttachmentIndex if VkRenderingInputAttachmentIndexInfoKHR:pDepthInputAttachmentIndex or VkRenderingInputAttachmentIndexInfoKHR:pStencilInputAttachmentIndex are NULL, **must** be created with a VkImageView that is compatible with the corresponding color, depth, or stencil attachment in VkRenderingInfo
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-pDepthInputAttachmentIndex-09596
  Input attachment views accessed in a dynamic render pass via a shader object **must** have an InputAttachmentIndex if both VkRenderingInputAttachmentIndexInfoKHR:pDepthInputAttachmentIndex and VkRenderingInputAttachmentIndexInfoKHR:pStencilInputAttachmentIndex are non-NULL
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-InputAttachmentIndex-09597
  If an input attachment view accessed in a dynamic render pass via a shader object has an InputAttachmentIndex, the InputAttachmentIndex **must** match an index in VkRenderingInputAttachmentIndexInfoKHR
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-06537
  Memory backing image subresources used as attachments in the current render pass **must** not be written in any way other than as an attachment by this command
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-09000
  If a color attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass, it is not in the VK_IMAGE_LAYOUT_ATTACHMENT_FEEDBACK_LOOP_OPTIMAL_EXT image layout, and either:
  - the VK_PIPELINE_CREATE_COLOR_ATTACHMENT_FEEDBACK_LOOP_BIT_EXT is set on the currently bound pipeline or
  - the last call to vkCmdSetAttachmentFeedbackLoopEnableEXT included VK_IMAGE_ASPECT_COLOR_BIT and
    - there is no currently bound graphics pipeline or
    - the currently bound graphics pipeline was created with VK_DYNAMIC_STATE_ATTACHMENT_FEEDBACK_LOOP_ENABLE_EXT

  it **must** not be accessed in any way other than as an attachment by this command
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-09001
  If a depth attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass, it is not in the VK_IMAGE_LAYOUT_ATTACHMENT_FEEDBACK_LOOP_OPTIMAL_EXT image layout, and either:
  - the VK_PIPELINE_CREATE_DEPTH_STENCIL_ATTACHMENT_FEEDBACK_LOOP_BIT_EXT is set on the currently bound pipeline or

- the last call to vkCmdSetAttachmentFeedbackLoopEnableEXT included VK_IMAGE_ASPECT_DEPTH_BIT and
  - there is no currently bound graphics pipeline or
  - the currently bound graphics pipeline was created with VK_DYNAMIC_STATE_ATTACHMENT_FEEDBACK_LOOP_ENABLE_EXT

it **must** not be accessed in any way other than as an attachment by this command

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-09002
  If a stencil attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass, it is not in the VK_IMAGE_LAYOUT_ATTACHMENT_FEEDBACK_LOOP_OPTIMAL_EXT image layout, and either:
  - the VK_PIPELINE_CREATE_DEPTH_STENCIL_ATTACHMENT_FEEDBACK_LOOP_BIT_EXT is set on the currently bound pipeline or
  - the last call to vkCmdSetAttachmentFeedbackLoopEnableEXT included VK_IMAGE_ASPECT_STENCIL_BIT and
    - there is no currently bound graphics pipeline or
    - the currently bound graphics pipeline was created with VK_DYNAMIC_STATE_ATTACHMENT_FEEDBACK_LOOP_ENABLE_EXT

  it **must** not be accessed in any way other than as an attachment by this command

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-09003
  If an attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass, it **must** not be accessed in any way other than as an attachment, storage image, or sampled image by this command
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-06539
  If any previously recorded command in the current subpass accessed an image subresource used as an attachment in this subpass in any way other than as an attachment, this command **must** not write to that image subresource as an attachment
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-06886
  If the current render pass instance uses a depth/stencil attachment with a read-only layout for the depth aspect, depth writes **must** be disabled
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-06887
  If the current render pass instance uses a depth/stencil attachment with a read-only layout for the stencil aspect, both front and back writeMask are not zero, and stencil test is enabled, all stencil ops **must** be VK_STENCIL_OP_KEEP
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-07831
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT dynamic state enabled then vkCmdSetViewport **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-07832
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SCISSOR dynamic state enabled then vkCmdSetScissor **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-07833
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_WIDTH dynamic state enabled then vkCmdSetLineWidth **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-08617
  If a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetPolygonModeEXT in the current command buffer set polygonMode to VK_POLYGON_MODE_LINE, vkCmdSetLineWidth **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-08618
  If a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetPrimitiveTopologyEXT in the current command buffer set primitiveTopology to any line topology, vkCmdSetLineWidth **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-08619
  If a shader object that outputs line primitives is bound to the VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT or VK_SHADER_STAGE_GEOMETRY_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, vkCmdSetLineWidth **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-07834
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_BIAS dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of depthBiasEnable is VK_TRUE, then vkCmdSetDepthBounds or vkCmdSetDepthBias2EXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-07835
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_BLEND_CONSTANTS dynamic state enabled then vkCmdSetBlendConstants **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-08621
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetColorBlendEnableEXT in the current command buffer set any element of pColorBlendEnables to VK_TRUE, and the most recent call to vkCmdSetColorBlendEquationEXT in the current command buffer set the same element of pColorBlendEquations to a VkColorBlendEquationEXT structure with any VkBlendFactor

member with a value of VK_BLEND_FACTOR_CONSTANT_COLOR, VK_BLEND_FACTOR_ONE_MINUS_CONSTANT_COLOR, VK_BLEND_FACTOR_CONSTANT_ALPHA, or VK_BLEND_FACTOR_ONE_MINUS_CONSTANT_ALPHA, vkCmdSetBlendConstants **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-07836
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_BOUNDS dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of depthBoundsTestEnable is VK_TRUE, then vkCmdSetDepthBounds **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-07837
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_STENCIL_COMPARE_MASK dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of stencilTestEnable is VK_TRUE, then vkCmdSetStencilCompareMask **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-07838
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_STENCIL_WRITE_MASK dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of stencilTestEnable is VK_TRUE, then vkCmdSetStencilWriteMask **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-07839
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_STENCIL_REFERENCE dynamic state enabled, the current value of and rasterizerDiscardEnable is VK_FALSE, the current value of stencilTestEnable is VK_TRUE, then vkCmdSetStencilReference **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-maxMultiviewInstanceIndex-02688
  If the draw is recorded in a render pass instance with multiview enabled, the maximum instance index **must** be less than or equal to VkPhysicalDeviceMultiviewProperties::maxMultiviewInstanceIndex
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-sampleLocationsEnable-02689
  If the bound graphics pipeline was created with VkPipelineSampleLocationsStateCreateInfoEXT::sampleLocationsEnable set to VK_TRUE and the current subpass has a depth/stencil attachment, then that attachment **must** have been created with the VK_IMAGE_CREATE_SAMPLE_LOCATIONS_COMPATIBLE_DEPTH_BIT_EXT bit set
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-06666
  If the VK_EXT_sample_locations extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the

VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of sampleLocationsEnable is VK_TRUE, then vkCmdSetSampleLocationsEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-07840
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_CULL_MODE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetCullModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-07841
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_FRONT_FACE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetFrontFaceEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-07843
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_TEST_ENABLE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, vkCmdSetDepthTestEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-07844
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_WRITE_ENABLE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetDepthWriteEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-07845
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_COMPARE_OP dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of depthTestEnable is VK_TRUE, then vkCmdSetDepthCompareOpEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-07846
  If the depthBounds feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_BOUNDS_TEST_ENABLE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetDepthBoundsTestEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-07847
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_STENCIL_TEST_ENABLE dynamic state enabled, and

the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetStencilTestEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-07848
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_STENCIL_OP dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, the current value of stencilTestEnable is VK_TRUE, then vkCmdSetStencilOpEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-viewportCount-03417
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, but not the VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT dynamic state enabled, then vkCmdSetViewportWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the viewportCount parameter of vkCmdSetViewportWithCountEXT **must** match the VkPipelineViewportStateCreateInfo::scissorCount of the pipeline

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-scissorCount-03418
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT dynamic state enabled, but not the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, then vkCmdSetScissorWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the scissorCount parameter of vkCmdSetScissorWithCountEXT **must** match the VkPipelineViewportStateCreateInfo::viewportCount of the pipeline

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-viewportCount-03419
  If the bound graphics pipeline state was created with both the VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT and VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic states enabled then both vkCmdSetViewportWithCountEXT and vkCmdSetScissorWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the viewportCount parameter of vkCmdSetViewportWithCountEXT **must** match the scissorCount parameter of vkCmdSetScissorWithCountEXT

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-08635
  If a shader object is bound to any graphics stage, then both vkCmdSetViewportWithCountEXT and vkCmdSetScissorWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the viewportCount parameter of vkCmdSetViewportWithCountEXT **must** match the scissorCount parameter of vkCmdSetScissorWithCountEXT

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-viewportCount-04137
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, but not the VK_DYNAMIC_STATE_VIEWPORT_W_SCALING_NV dynamic state enabled, then the bound graphics pipeline **must** have been created with VkPipelineViewportWScalingStateCreateInfoNV::viewportCount greater or equal to the viewportCount parameter in the last call to

vkCmdSetViewportWithCountEXT

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-viewportCount-04138
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT and VK_DYNAMIC_STATE_VIEWPORT_W_SCALING_NV dynamic states enabled then the viewportCount parameter in the last call to vkCmdSetViewportWScalingNV **must** be greater than or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-09232
  If the VK_NV_clip_space_w_scaling extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetViewportWScalingEnableNV in the current command buffer set viewportWScalingEnable to VK_TRUE, then vkCmdSetViewportWScalingNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-08636
  If the VK_NV_clip_space_w_scaling extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetViewportWScalingEnableNV in the current command buffer set viewportWScalingEnable to VK_TRUE, then the viewportCount parameter in the last call to vkCmdSetViewportWScalingNV **must** be greater than or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-viewportCount-04139
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, but not the VK_DYNAMIC_STATE_VIEWPORT_SHADING_RATE_PALETTE_NV dynamic state enabled, then the bound graphics pipeline **must** have been created with VkPipelineViewportShadingRateImageStateCreateInfoNV::viewportCount greater or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-viewportCount-04140
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT and VK_DYNAMIC_STATE_VIEWPORT_SHADING_RATE_PALETTE_NV dynamic states enabled then the viewportCount parameter in the last call to vkCmdSetViewportShadingRatePaletteNV **must** be greater than or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-shadingRateImage-09233
  If the shadingRateImage feature is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetCoarseSampleOrderNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-shadingRateImage-09234
  If the shadingRateImage feature is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetShadingRateImageEnableNV in the current command buffer set shadingRateImageEnable to VK_TRUE, then vkCmdSetViewportShadingRatePaletteNV **must**

have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-08637
  If the shadingRateImage feature is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetShadingRateImageEnableNV in the current command buffer set shadingRateImageEnable to VK_TRUE, then the viewportCount parameter in the last call to vkCmdSetViewportShadingRatePaletteNV **must** be greater than or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-VkPipelineVieportCreateInfo-04141
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled and a VkPipelineViewportSwizzleStateCreateInfoNV structure chained from VkPipelineViewportStateCreateInfo, then the bound graphics pipeline **must** have been created with VkPipelineViewportSwizzleStateCreateInfoNV::viewportCount greater or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-VkPipelineVieportCreateInfo-04142
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled and a VkPipelineViewportExclusiveScissorStateCreateInfoNV structure chained from VkPipelineViewportStateCreateInfo, then the bound graphics pipeline **must** have been created with VkPipelineViewportExclusiveScissorStateCreateInfoNV::exclusiveScissorCount greater or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-07878
  If the exclusiveScissor feature is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_EXCLUSIVE_SCISSOR_ENABLE_NV dynamic state enabled, then vkCmdSetExclusiveScissorEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-07879
  If the exclusiveScissor feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_EXCLUSIVE_SCISSOR_NV dynamic state enabled, and the most recent call to vkCmdSetExclusiveScissorEnableNV in the current command buffer set any element of pExclusiveScissorEnables to VK_TRUE, then vkCmdSetExclusiveScissorNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-04876
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_RASTERIZER_DISCARD_ENABLE dynamic state enabled, then vkCmdSetRasterizerDiscardEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-04877

If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_BIAS_ENABLE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetDepthBiasEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-logicOp-04878
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_LOGIC_OP_EXT dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of logicOpEnable is VK_TRUE, then vkCmdSetLogicOpEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-primitiveFragmentShadingRateWithMultipleViewports-04552
  If the primitiveFragmentShadingRateWithMultipleViewports limit is not supported, the bound graphics pipeline was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, and any of the shader stages of the bound graphics pipeline write to the PrimitiveShadingRateKHR built-in, then vkCmdSetViewportWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the viewportCount parameter of vkCmdSetViewportWithCountEXT **must** be 1
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-primitiveFragmentShadingRateWithMultipleViewports-08642
  If the primitiveFragmentShadingRateWithMultipleViewports limit is not supported, and any shader object bound to a graphics stage writes to the PrimitiveShadingRateKHR built-in, then vkCmdSetViewportWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the viewportCount parameter of vkCmdSetViewportWithCountEXT **must** be 1
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-blendEnable-04727
  If rasterization is not disabled in the bound graphics pipeline, then for each color attachment in the subpass, if the corresponding image view's format features do not contain VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT, then the blendEnable member of the corresponding element of the pAttachments member of pColorBlendState **must** be VK_FALSE
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-08643
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then for each color attachment in the render pass, if the corresponding image view's format features do not contain VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT, then the corresponding member of pColorBlendEnables in the most recent call to vkCmdSetColorBlendEnableEXT in the current command buffer that affected that attachment index **must** have been VK_FALSE
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-multisampledRenderToSingleSampled-07284
  If rasterization is not disabled in the bound graphics pipeline, and none of the following is enabled:
  - the VK_AMD_mixed_attachment_samples extension

- the VK_NV_framebuffer_mixed_samples extension
- the multisampledRenderToSingleSampled feature

then rasterizationSamples for the currently bound graphics pipeline **must** be the same as the current subpass color and/or depth/stencil attachments

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-08644
  If a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and none of the following is enabled:

  - the VK_AMD_mixed_attachment_samples extension
  - the VK_NV_framebuffer_mixed_samples extension
  - the multisampledRenderToSingleSampled feature

  then the most recent call to vkCmdSetRasterizationSamplesEXT in the current command buffer **must** have set rasterizationSamples to be the same as the number of samples for the current render pass color and/or depth/stencil attachments

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-08876
  If a shader object is bound to any graphics stage, the current render pass instance **must** have been begun with vkCmdBeginRenderingKHR
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-imageView-06172
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the imageView member of pDepthAttachment is not VK_NULL_HANDLE, and the layout member of pDepthAttachment is VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL, this command **must** not write any values to the depth attachment
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-imageView-06173
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the imageView member of pStencilAttachment is not VK_NULL_HANDLE, and the layout member of pStencilAttachment is VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL, this command **must** not write any values to the stencil attachment
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-imageView-06174
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the imageView member of pDepthAttachment is not VK_NULL_HANDLE, and the layout member of pDepthAttachment is VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_STENCIL_ATTACHMENT_OPTIMAL, this command **must** not write any values to the depth attachment
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-imageView-06175
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the imageView member of pStencilAttachment is not VK_NULL_HANDLE, and the layout member of pStencilAttachment is VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_STENCIL_READ_ONLY_OPTIMAL, this command **must** not write any values to the stencil attachment
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-imageView-06176
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the imageView member of pDepthAttachment is not VK_NULL_HANDLE, and the layout member

of pDepthAttachment is VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL, this command **must** not write any values to the depth attachment

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-imageView-06177
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the imageView member of pStencilAttachment is not VK_NULL_HANDLE, and the layout member of pStencilAttachment is VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL, this command **must** not write any values to the stencil attachment

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-viewMask-06178
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the currently bound graphics pipeline **must** have been created with a VkPipelineRenderingCreateInfo::viewMask equal to VkRenderingInfo::viewMask

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-colorAttachmentCount-06179
  If the dynamicRenderingUnusedAttachments feature is not enabled and the current render pass instance was begun with vkCmdBeginRenderingKHR, the currently bound graphics pipeline **must** have been created with a VkPipelineRenderingCreateInfo::colorAttachmentCount equal to VkRenderingInfo::colorAttachmentCount

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-dynamicRenderingUnusedAttachments-08910
  If the dynamicRenderingUnusedAttachments feature is not enabled, and the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingInfo::colorAttachmentCount greater than 0, then each element of the VkRenderingInfo::pColorAttachments array with an imageView not equal to VK_NULL_HANDLE **must** have been created with a VkFormat equal to the corresponding element of VkPipelineRenderingCreateInfo::pColorAttachmentFormats used to create the currently bound graphics pipeline

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-dynamicRenderingUnusedAttachments-08912
  If the dynamicRenderingUnusedAttachments feature is not enabled, and the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingInfo::colorAttachmentCount greater than 0, then each element of the VkRenderingInfo::pColorAttachments array with an imageView equal to VK_NULL_HANDLE **must** have the corresponding element of VkPipelineRenderingCreateInfo::pColorAttachmentFormats used to create the currently bound pipeline equal to VK_FORMAT_UNDEFINED

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-dynamicRenderingUnusedAttachments-08911
  If the dynamicRenderingUnusedAttachments feature is enabled, and the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingInfo::colorAttachmentCount greater than 0, then each element of the VkRenderingInfo::pColorAttachments array with an imageView not equal to VK_NULL_HANDLE **must** have been created with a VkFormat equal to the corresponding element of VkPipelineRenderingCreateInfo::pColorAttachmentFormats used to create the currently bound graphics pipeline, or the corresponding element of VkPipelineRenderingCreateInfo::pColorAttachmentFormats, if it exists, **must** be VK_FORMAT_UNDEFINED

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-colorAttachmentCount-09362

If the current render pass instance was begun with vkCmdBeginRenderingKHR, with a VkRenderingInfo::colorAttachmentCount equal to 1, there is no shader object bound to any graphics stage, and a color attachment with a resolve mode of VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID, each element of the VkRenderingInfo::pColorAttachments array with a resolveImageView not equal to VK_NULL_HANDLE **must** have been created with an image created with a VkExternalFormatANDROID::externalFormat value equal to the VkExternalFormatANDROID::externalFormat value used to create the currently bound graphics pipeline

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-09363
  If there is no shader object bound to any graphics stage, the current render pass instance was begun with vkCmdBeginRenderingKHR and a VkRenderingInfo::colorAttachmentCount equal to 1, and a color attachment with a resolve mode of VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID, each element of the VkRenderingInfo::pColorAttachments array with a imageView not equal to VK_NULL_HANDLE **must** have been created with an image created with a VkExternalFormatANDROID::externalFormat value equal to the VkExternalFormatANDROID::externalFormat value used to create the currently bound graphics pipeline
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-09364
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is no shader object bound to any graphics stage, and the currently bound graphics pipeline was created with a non-zero VkExternalFormatANDROID::externalFormat value and with the VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT dynamic state enabled, then vkCmdSetColorBlendEnableEXT **must** have set the blend enable to VK_FALSE prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-09365
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is no shader object bound to any graphics stage, and the currently bound graphics pipeline was created with a non-zero VkExternalFormatANDROID::externalFormat value and with the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT dynamic state enabled, then vkCmdSetRasterizationSamplesEXT **must** have set rasterizationSamples to VK_SAMPLE_COUNT_1_BIT prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-09366
  If there is a shader object bound to any graphics stage, and the current render pass includes a color attachment that uses the VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID resolve mode, then vkCmdSetColorBlendEnableEXT **must** have set blend enable to VK_FALSE prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-rasterizationSamples-09367
  If there is a shader object bound to any graphics stage, and the current render pass includes a color attachment that uses the VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID resolve mode, then vkCmdSetRasterizationSamplesEXT **must** have set rasterizationSamples to VK_SAMPLE_COUNT_1_BIT prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-09368
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is no shader object bound to any graphics stage, and the currently bound graphics pipeline was created with a non-zero VkExternalFormatANDROID::externalFormat value and with the VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR dynamic state enabled, then vkCmdSetFragmentShadingRateKHR **must** have set pFragmentSize->width to 1 prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-09369
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is no shader object bound to any graphics stage, and the currently bound graphics pipeline was created with a non-zero VkExternalFormatANDROID::externalFormat value and with the VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR dynamic state enabled, then vkCmdSetFragmentShadingRateKHR **must** have set pFragmentSize->height to 1 prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-pFragmentSize-09370
  If there is a shader object bound to any graphics stage, and the current render pass includes a color attachment that uses the VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID resolve mode, then vkCmdSetFragmentShadingRateKHR **must** have set pFragmentSize->width to 1 prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-pFragmentSize-09371
  If there is a shader object bound to any graphics stage, and the current render pass includes a color attachment that uses the VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID resolve mode, then vkCmdSetFragmentShadingRateKHR **must** have set pFragmentSize->height to 1 prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-07749
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_WRITE_ENABLE_EXT dynamic state enabled then vkCmdSetColorWriteEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-08646
  If the colorWriteEnable feature is enabled on the device, and a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetColorWriteEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-attachmentCount-07750
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_WRITE_ENABLE_EXT dynamic state enabled then the attachmentCount parameter of vkCmdSetColorWriteEnableEXT **must** be greater than or equal to the VkPipelineColorBlendStateCreateInfo::attachmentCount of the currently bound graphics pipeline
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-08647

If the colorWriteEnable feature is enabled on the device, and a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then the attachmentCount parameter of most recent call to vkCmdSetColorWriteEnableEXT in the current command buffer **must** be greater than or equal to the number of color attachments in the current render pass instance

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-07751
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_DISCARD_RECTANGLE_EXT dynamic state enabled then vkCmdSetDiscardRectangleEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command for each discard rectangle in VkPipelineDiscardRectangleStateCreateInfoEXT::discardRectangleCount
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-07880
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_DISCARD_RECTANGLE_ENABLE_EXT dynamic state enabled then vkCmdSetDiscardRectangleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-rasterizerDiscardEnable-09236
  If the VK_EXT_discard_rectangles extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetDiscardRectangleEnableEXT in the current command buffer set discardRectangleEnable to VK_TRUE, then vkCmdSetDiscardRectangleEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-08648
  If the VK_EXT_discard_rectangles extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetDiscardRectangleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-07881
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_DISCARD_RECTANGLE_MODE_EXT dynamic state enabled then vkCmdSetDiscardRectangleModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-08649
  If the VK_EXT_discard_rectangles extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetDiscardRectangleEnableEXT in the current command buffer set discardRectangleEnable to VK_TRUE, then vkCmdSetDiscardRectangleModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-dynamicRenderingUnusedAttachments-

08913
If the current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is not enabled, and VkRenderingInfo::pDepthAttachment->imageView was VK_NULL_HANDLE, the value of VkPipelineRenderingCreateInfo::depthAttachmentFormat used to create the currently bound graphics pipeline **must** be equal to VK_FORMAT_UNDEFINED

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-dynamicRenderingUnusedAttachments-08914
If current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is not enabled, and VkRenderingInfo::pDepthAttachment->imageView was not VK_NULL_HANDLE, the value of VkPipelineRenderingCreateInfo::depthAttachmentFormat used to create the currently bound graphics pipeline **must** be equal to the VkFormat used to create VkRenderingInfo::pDepthAttachment->imageView

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-dynamicRenderingUnusedAttachments-08915
If the current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is enabled, VkRenderingInfo::pDepthAttachment->imageView was not VK_NULL_HANDLE, and the value of VkPipelineRenderingCreateInfo::depthAttachmentFormat used to create the currently bound graphics pipeline was not equal to the VkFormat used to create VkRenderingInfo::pDepthAttachment->imageView, the value of the format **must** be VK_FORMAT_UNDEFINED

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-dynamicRenderingUnusedAttachments-08916
If the current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is not enabled, and VkRenderingInfo::pStencilAttachment->imageView was VK_NULL_HANDLE, the value of VkPipelineRenderingCreateInfo::stencilAttachmentFormat used to create the currently bound graphics pipeline **must** be equal to VK_FORMAT_UNDEFINED

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-dynamicRenderingUnusedAttachments-08917
If current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is not enabled, and VkRenderingInfo::pStencilAttachment->imageView was not VK_NULL_HANDLE, the value of VkPipelineRenderingCreateInfo::stencilAttachmentFormat used to create the currently bound graphics pipeline **must** be equal to the VkFormat used to create VkRenderingInfo::pStencilAttachment->imageView

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-dynamicRenderingUnusedAttachments-08918
If the current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is enabled, VkRenderingInfo::pStencilAttachment->imageView was not VK_NULL_HANDLE, and the value of VkPipelineRenderingCreateInfo::stencilAttachmentFormat used to create the currently bound graphics pipeline was not equal to the VkFormat used to create VkRenderingInfo::pStencilAttachment->imageView, the value of the format **must** be VK_FORMAT_UNDEFINED

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-imageView-06183
  If the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingFragmentShadingRateAttachmentInfoKHR::imageView was not VK_NULL_HANDLE, the currently bound graphics pipeline **must** have been created with VK_PIPELINE_CREATE_RENDERING_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-imageView-06184
  If the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingFragmentDensityMapAttachmentInfoEXT::imageView was not VK_NULL_HANDLE, the currently bound graphics pipeline **must** have been created with VK_PIPELINE_CREATE_RENDERING_FRAGMENT_DENSITY_MAP_ATTACHMENT_BIT_EXT
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-colorAttachmentCount-06185
  If the currently bound pipeline was created with a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and the current render pass instance was begun with vkCmdBeginRenderingKHR with a VkRenderingInfo::colorAttachmentCount parameter greater than 0, then each element of the VkRenderingInfo::pColorAttachments array with a imageView not equal to VK_NULL_HANDLE **must** have been created with a sample count equal to the corresponding element of the pColorAttachmentSamples member of VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV used to create the currently bound graphics pipeline
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-pDepthAttachment-06186
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the currently bound pipeline was created with a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and VkRenderingInfo::pDepthAttachment->imageView was not VK_NULL_HANDLE, the value of the depthStencilAttachmentSamples member of VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV used to create the currently bound graphics pipeline **must** be equal to the sample count used to create VkRenderingInfo::pDepthAttachment->imageView
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-pStencilAttachment-06187
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the currently bound pipeline was created with a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and VkRenderingInfo::pStencilAttachment->imageView was not VK_NULL_HANDLE, the value of the depthStencilAttachmentSamples member of VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV used to create the currently bound graphics pipeline **must** be equal to the sample count used to create VkRenderingInfo::pStencilAttachment->imageView
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-multisampledRenderToSingleSampled-07285
  If the currently bound pipeline was created without a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and the multisampledRenderToSingleSampled feature is not enabled, and the current render pass instance was begun with vkCmdBeginRenderingKHR with a VkRenderingInfo::colorAttachmentCount parameter greater than 0, then each element of the VkRenderingInfo::pColorAttachments array with a imageView not equal to VK_NULL_HANDLE **must** have been created with a sample count equal to the value of rasterizationSamples for the currently bound graphics pipeline

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-multisampledRenderToSingleSampled-07286
  If the currently bound pipeline was created without a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and the `multisampledRenderToSingleSampled` feature is not enabled, and VkRenderingInfo::`pDepthAttachment->imageView` was not VK_NULL_HANDLE, the value of `rasterizationSamples` for the currently bound graphics pipeline **must** be equal to the sample count used to create VkRenderingInfo::`pDepthAttachment->imageView`
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-multisampledRenderToSingleSampled-07287
  If the currently bound pipeline was created without a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and the `multisampledRenderToSingleSampled` feature is not enabled, and VkRenderingInfo::`pStencilAttachment->imageView` was not VK_NULL_HANDLE, the value of `rasterizationSamples` for the currently bound graphics pipeline **must** be equal to the sample count used to create VkRenderingInfo::`pStencilAttachment->imageView`
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-pNext-07935
  If this command has been called inside a render pass instance started with vkCmdBeginRenderingKHR, and the `pNext` chain of VkRenderingInfo includes a VkMultisampledRenderToSingleSampledInfoEXT structure with `multisampledRenderToSingleSampledEnable` equal to `VK_TRUE`, then the value of `rasterizationSamples` for the currently bound graphics pipeline **must** be equal to VkMultisampledRenderToSingleSampledInfoEXT::`rasterizationSamples`
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-renderPass-06198
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the currently bound pipeline **must** have been created with a VkGraphicsPipelineCreateInfo::`renderPass` equal to VK_NULL_HANDLE
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-pColorAttachments-08963
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is a graphics pipeline bound with a fragment shader that statically writes to a color attachment, the color write mask is not zero, color writes are enabled, and the corresponding element of the VkRenderingInfo::`pColorAttachments->imageView` was not VK_NULL_HANDLE, then the corresponding element of VkPipelineRenderingCreateInfo::`pColorAttachmentFormats` used to create the pipeline **must** not be `VK_FORMAT_UNDEFINED`
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-pDepthAttachment-08964
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is a graphics pipeline bound, depth test is enabled, depth write is enabled, and the VkRenderingInfo::`pDepthAttachment->imageView` was not VK_NULL_HANDLE, then the VkPipelineRenderingCreateInfo::`depthAttachmentFormat` used to create the pipeline **must** not be `VK_FORMAT_UNDEFINED`
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-pStencilAttachment-08965
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is a graphics pipeline bound, stencil test is enabled and the VkRenderingInfo::`pStencilAttachment->imageView` was not VK_NULL_HANDLE, then the VkPipelineRenderingCreateInfo::`stencilAttachmentFormat` used to create the pipeline **must**

not be VK_FORMAT_UNDEFINED

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-primitivesGeneratedQueryWithRasterizerDiscard-06708
  If the primitivesGeneratedQueryWithRasterizerDiscard feature is not enabled and the VK_QUERY_TYPE_PRIMITIVES_GENERATED_EXT query is active, rasterization discard **must** not be enabled
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-primitivesGeneratedQueryWithNonZeroStreams-06709
  If the primitivesGeneratedQueryWithNonZeroStreams feature is not enabled and the VK_QUERY_TYPE_PRIMITIVES_GENERATED_EXT query is active, the bound graphics pipeline **must** not have been created with a non-zero value in VkPipelineRasterizationStateStreamCreateInfoEXT::rasterizationStream
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-07619
  If a shader object is bound to the VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_TESSELLATION_DOMAIN_ORIGIN_EXT dynamic state enabled, then vkCmdSetTessellationDomainOriginEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-07620
  If the depthClamp feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_CLAMP_ENABLE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetDepthClampEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-07621
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_POLYGON_MODE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetPolygonModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-07622
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetRasterizationSamplesEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-07623
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_SAMPLE_MASK_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetSampleMaskEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-alphaToCoverageEnable-08919
  If the bound graphics pipeline state was created with the

VK_DYNAMIC_STATE_ALPHA_TO_COVERAGE_ENABLE_EXT dynamic state enabled, and alphaToCoverageEnable was VK_TRUE in the last call to vkCmdSetAlphaToCoverageEnableEXT, then the Fragment Output Interface **must** contain a variable for the alpha Component word in Location 0 at Index 0

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-alphaToCoverageEnable-08920
  If a shader object is bound to any graphics stage, and the most recent call to vkCmdSetAlphaToCoverageEnableEXT in the current command buffer set alphaToCoverageEnable to VK_TRUE, then the Fragment Output Interface **must** contain a variable for the alpha Component word in Location 0 at Index 0

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-07624
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_ALPHA_TO_COVERAGE_ENABLE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetAlphaToCoverageEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-07625
  If the alphaToOne feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_ALPHA_TO_ONE_ENABLE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetAlphaToOneEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-07626
  If the logicOp feature is enabled, a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_LOGIC_OP_ENABLE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetLogicOpEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-07627
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT dynamic state enabled then vkCmdSetColorBlendEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-08657
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and both the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE and there are color attachments bound, then vkCmdSetColorBlendEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-07628
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_EQUATION_EXT dynamic state enabled then vkCmdSetColorBlendEquationEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-08658
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetColorBlendEnableEXT for any attachment set that attachment's value in pColorBlendEnables to VK_TRUE, then vkCmdSetColorBlendEquationEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-07629
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_WRITE_MASK_EXT dynamic state enabled then vkCmdSetColorWriteMaskEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-08659
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and both the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE and there are color attachments bound, then vkCmdSetColorWriteMaskEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-07630
  If the geometryStreams feature is enabled, and a shader object is bound to the VK_SHADER_STAGE_GEOMETRY_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_RASTERIZATION_STREAM_EXT dynamic state enabled, then vkCmdSetRasterizationStreamEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-07631
  If the VK_EXT_conservative_rasterization extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_CONSERVATIVE_RASTERIZATION_MODE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetConservativeRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-07632
  If the VK_EXT_conservative_rasterization extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_EXTRA_PRIMITIVE_OVERESTIMATION_SIZE_EXT dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of conservativeRasterizationMode is VK_CONSERVATIVE_RASTERIZATION_MODE_OVERESTIMATE_EXT, then vkCmdSetExtraPrimitiveOverestimationSizeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-07633
  If the depthClipEnable feature is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_CLIP_ENABLE_EXT dynamic state, then vkCmdSetDepthClipEnableEXT **must** have been called and not subsequently invalidated

in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-07634
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT dynamic state enabled then vkCmdSetSampleLocationsEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-08664
  If the VK_EXT_sample_locations extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetSampleLocationsEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-07635
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_ADVANCED_EXT dynamic state enabled then vkCmdSetColorBlendAdvancedEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-rasterizerDiscardEnable-09416
  If the VK_EXT_blend_operation_advanced extension is enabled, and a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then at least one of vkCmdSetColorBlendEquationEXT and vkCmdSetColorBlendAdvancedEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-07636
  If the VK_EXT_provoking_vertex extension is enabled, a shader object is bound to the VK_SHADER_STAGE_VERTEX_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_PROVOKING_VERTEX_MODE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetProvokingVertexModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-07637
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT dynamic state enabled then vkCmdSetLineRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-08666
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetPolygonModeEXT in the current command buffer set polygonMode to VK_POLYGON_MODE_LINE, then vkCmdSetLineRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing

command

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-08667
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object is bound to the VK_SHADER_STAGE_VERTEX_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetPrimitiveTopologyEXT in the current command buffer set primitiveTopology to any line topology, then vkCmdSetLineRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-08668
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object that outputs line primitives is bound to the VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT or VK_SHADER_STAGE_GEOMETRY_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetLineRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-07638
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT dynamic state enabled then vkCmdSetLineStippleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-08669
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetPolygonModeEXT in the current command buffer set polygonMode to VK_POLYGON_MODE_LINE, then vkCmdSetLineStippleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-08670
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object is bound to the VK_SHADER_STAGE_VERTEX_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetPrimitiveTopologyEXT in the current command buffer set primitiveTopology to any line topology, then vkCmdSetLineStippleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-08671
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object that outputs line primitives is bound to the VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT or VK_SHADER_STAGE_GEOMETRY_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command

buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetLineStippleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-07849
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_KHR dynamic state enabled then vkCmdSetLineStippleKHR **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-08672
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetLineStippleEnableEXT in the current command buffer set stippledLineEnable to VK_TRUE, then vkCmdSetLineStippleEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-07639
  If the depthClipControl feature is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_CLIP_NEGATIVE_ONE_TO_ONE_EXT dynamic state enabled, then vkCmdSetDepthClipNegativeOneToOneEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-07640
  If the VK_NV_clip_space_w_scaling extension is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_VIEWPORT_W_SCALING_ENABLE_NV dynamic state enabled, then vkCmdSetViewportWScalingEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-07641
  If the VK_NV_viewport_swizzle extension is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_VIEWPORT_SWIZZLE_NV dynamic state enabled, then vkCmdSetViewportSwizzleNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-07642
  If the VK_NV_fragment_coverage_to_color extension is enabled, a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_TO_COLOR_ENABLE_NV dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetCoverageToColorEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-07643
  If the VK_NV_fragment_coverage_to_color extension is enabled, a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_TO_COLOR_LOCATION_NV dynamic state enabled, the

current value of rasterizerDiscardEnable is VK_FALSE, and the current value of coverageToColorEnable is VK_TRUE, then vkCmdSetCoverageToColorLocationNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-07644
  If the VK_NV_framebuffer_mixed_samples extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_MODULATION_MODE_NV dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetCoverageModulationModeNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-07645
  If the VK_NV_framebuffer_mixed_samples extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_MODULATION_TABLE_ENABLE_NV dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of coverageModulationMode is any value other than VK_COVERAGE_MODULATION_MODE_NONE_NV, then vkCmdSetCoverageModulationTableEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-07646
  If the VK_NV_framebuffer_mixed_samples extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_MODULATION_TABLE_NV dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of coverageModulationTableEnable is VK_TRUE, then vkCmdSetCoverageModulationTableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-07647
  If the shadingRateImage feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_SHADING_RATE_IMAGE_ENABLE_NV dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetShadingRateImageEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-pipelineFragmentShadingRate-09238
  If the pipelineFragmentShadingRate feature is enabled, a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetFragmentShadingRateKHR **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-07648
  If the representativeFragmentTest feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_REPRESENTATIVE_FRAGMENT_TEST_ENABLE_NV dynamic state enabled, and the

current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetRepresentativeFragmentTestEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-07649
  If the coverageReductionMode feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_REDUCTION_MODE_NV dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetCoverageReductionModeNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-pColorBlendEnables-07470
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT state enabled and the last call to vkCmdSetColorBlendEnableEXT set pColorBlendEnables for any attachment to VK_TRUE, then for those attachments in the subpass the corresponding image view's format features **must** contain VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-rasterizationSamples-07471
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, and the current subpass does not use any color and/or depth/stencil attachments, then the rasterizationSamples in the last call to vkCmdSetRasterizationSamplesEXT **must** follow the rules for a zero-attachment subpass

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-samples-07472
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_MASK_EXT state enabled and the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state disabled, then the samples parameter in the last call to vkCmdSetSampleMaskEXT **must** be greater or equal to the VkPipelineMultisampleStateCreateInfo::rasterizationSamples parameter used to create the bound graphics pipeline

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-samples-07473
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_MASK_EXT state and VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT states enabled, then the samples parameter in the last call to vkCmdSetSampleMaskEXT **must** be greater or equal to the rasterizationSamples parameter in the last call to vkCmdSetRasterizationSamplesEXT

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-rasterizationSamples-07474
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, and neither the VK_AMD_mixed_attachment_samples nor the VK_NV_framebuffer_mixed_samples extensions are enabled, then the rasterizationSamples in the last call to vkCmdSetRasterizationSamplesEXT **must** be the same as the current subpass color and/or depth/stencil attachments

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-09211
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, or a shader object is bound to

any graphics stage, and the current render pass instance includes a VkMultisampledRenderToSingleSampledInfoEXT structure with multisampledRenderToSingleSampledEnable equal to VK_TRUE, then the rasterizationSamples in the last call to vkCmdSetRasterizationSamplesEXT **must** be the same as the rasterizationSamples member of that structure

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-firstAttachment-07476
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT dynamic state enabled then vkCmdSetColorBlendEnableEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the firstAttachment and attachmentCount parameters of vkCmdSetColorBlendEnableEXT calls **must** specify an enable for all active color attachments in the current subpass

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-rasterizerDiscardEnable-09417
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetColorBlendEnableEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the firstAttachment and attachmentCount parameters of vkCmdSetColorBlendEnableEXT calls **must** specify an enable for all active color attachments in the current subpass

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-firstAttachment-07477
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_EQUATION_EXT dynamic state enabled then vkCmdSetColorBlendEquationEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the firstAttachment and attachmentCount parameters of vkCmdSetColorBlendEquationEXT calls **must** specify the blend equations for all active color attachments in the current subpass where blending is enabled

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-rasterizerDiscardEnable-09418
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and both the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE and there are color attachments bound, then vkCmdSetColorBlendEquationEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the firstAttachment and attachmentCount parameters of vkCmdSetColorBlendEquationEXT calls **must** specify the blend equations for all active color attachments in the current subpass where blending is enabled

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-firstAttachment-07478
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_WRITE_MASK_EXT dynamic state enabled then vkCmdSetColorWriteMaskEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the firstAttachment and attachmentCount parameters of vkCmdSetColorWriteMaskEXT calls **must** specify the color write mask for all active color attachments in the current subpass

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-rasterizerDiscardEnable-09419

If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetColorWriteMaskEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the firstAttachment and attachmentCount parameters of vkCmdSetColorWriteMaskEXT calls **must** specify the color write mask for all active color attachments in the current subpass

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-firstAttachment-07479
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_ADVANCED_EXT dynamic state enabled then vkCmdSetColorBlendAdvancedEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the firstAttachment and attachmentCount parameters of vkCmdSetColorBlendAdvancedEXT calls **must** specify the advanced blend equations for all active color attachments in the current subpass where blending is enabled

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-advancedBlendMaxColorAttachments-07480
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_ADVANCED_EXT and VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT dynamic states enabled and the last calls to vkCmdSetColorBlendEnableEXT and vkCmdSetColorBlendAdvancedEXT have enabled advanced blending, then the number of active color attachments in the current subpass **must** not exceed advancedBlendMaxColorAttachments

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-primitivesGeneratedQueryWithNonZeroStreams-07481
  If the primitivesGeneratedQueryWithNonZeroStreams feature is not enabled and the VK_QUERY_TYPE_PRIMITIVES_GENERATED_EXT query is active, and the bound graphics pipeline was created with VK_DYNAMIC_STATE_RASTERIZATION_STREAM_EXT state enabled, the last call to vkCmdSetRasterizationStreamEXT **must** have set the rasterizationStream to zero

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-sampleLocationsPerPixel-07482
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state enabled and the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state disabled, then the sampleLocationsPerPixel member of pSampleLocationsInfo in the last call to vkCmdSetSampleLocationsEXT **must** equal the rasterizationSamples member of the VkPipelineMultisampleStateCreateInfo structure the bound graphics pipeline has been created with

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-sampleLocationsPerPixel-07483
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state enabled and the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, then the sampleLocationsPerPixel member of pSampleLocationsInfo in the last call to vkCmdSetSampleLocationsEXT **must** equal the rasterizationSamples parameter of the last call to vkCmdSetRasterizationSamplesEXT

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-sampleLocationsEnable-07484

If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, or the bound graphics pipeline was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, and sampleLocationsEnable was VK_TRUE in the last call to vkCmdSetSampleLocationsEnableEXT, and the current subpass has a depth/stencil attachment, then that attachment **must** have been created with the VK_IMAGE_CREATE_SAMPLE_LOCATIONS_COMPATIBLE_DEPTH_BIT_EXT bit set

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-sampleLocationsEnable-07485
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state enabled and the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, and if sampleLocationsEnable was VK_TRUE in the last call to vkCmdSetSampleLocationsEnableEXT, then the sampleLocationsInfo.sampleLocationGridSize.width in the last call to vkCmdSetSampleLocationsEXT **must** evenly divide VkMultisamplePropertiesEXT::sampleLocationGridSize.width as returned by vkGetPhysicalDeviceMultisamplePropertiesEXT with a samples parameter equaling rasterizationSamples

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-sampleLocationsEnable-07486
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state enabled and the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, and if sampleLocationsEnable was VK_TRUE in the last call to vkCmdSetSampleLocationsEnableEXT, then the sampleLocationsInfo.sampleLocationGridSize.height in the last call to vkCmdSetSampleLocationsEXT **must** evenly divide VkMultisamplePropertiesEXT::sampleLocationGridSize.height as returned by vkGetPhysicalDeviceMultisamplePropertiesEXT with a samples parameter equaling rasterizationSamples

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-sampleLocationsEnable-07487
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, and if sampleLocationsEnable was VK_TRUE in the last call to vkCmdSetSampleLocationsEnableEXT, the fragment shader code **must** not statically use the extended instruction InterpolateAtSample

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-sampleLocationsEnable-07936
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state disabled and the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, the sampleLocationsEnable member of a VkPipelineSampleLocationsStateCreateInfoEXT::sampleLocationsEnable in the bound graphics pipeline is VK_TRUE or VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, then, sampleLocationsInfo.sampleLocationGridSize.width **must** evenly divide VkMultisamplePropertiesEXT::sampleLocationGridSize.width as returned by vkGetPhysicalDeviceMultisamplePropertiesEXT with a samples parameter equaling the value of rasterizationSamples in the last call to vkCmdSetRasterizationSamplesEXT

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-sampleLocationsEnable-07937
  If the bound graphics pipeline state was created with the

VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state disabled and the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, the sampleLocationsEnable member of a VkPipelineSampleLocationsStateCreateInfoEXT::sampleLocationsEnable in the bound graphics pipeline is VK_TRUE or VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, then, sampleLocationsInfo.sampleLocationGridSize.height **must** evenly divide VkMultisamplePropertiesEXT::sampleLocationGridSize.height as returned by vkGetPhysicalDeviceMultisamplePropertiesEXT with a samples parameter equaling the value of rasterizationSamples in the last call to vkCmdSetRasterizationSamplesEXT

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-sampleLocationsEnable-07938
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state disabled and the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, the sampleLocationsEnable member of a VkPipelineSampleLocationsStateCreateInfoEXT::sampleLocationsEnable in the bound graphics pipeline is VK_TRUE or VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, then, sampleLocationsInfo.sampleLocationsPerPixel **must** equal rasterizationSamples in the last call to vkCmdSetRasterizationSamplesEXT

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-coverageModulationTableEnable-07488
  If a shader object is bound to any graphics stage or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COVERAGE_MODULATION_TABLE_ENABLE_NV state enabled, and the last call to vkCmdSetCoverageModulationTableEnableNV set coverageModulationTableEnable to VK_TRUE, then the coverageModulationTableCount parameter in the last call to vkCmdSetCoverageModulationTableNV **must** equal the current rasterizationSamples divided by the number of color samples in the current subpass

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-rasterizationSamples-07489
  If the VK_NV_framebuffer_mixed_samples extension is enabled, and if current subpass has a depth/stencil attachment and depth test, stencil test, or depth bounds test are enabled in the currently bound pipeline state, then the current rasterizationSamples **must** be the same as the sample count of the depth/stencil attachment

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-coverageToColorEnable-07490
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COVERAGE_TO_COLOR_ENABLE_NV state enabled and the last call to vkCmdSetCoverageToColorEnableNV set the coverageToColorEnable to VK_TRUE, then the current subpass **must** have a color attachment at the location selected by the last call to vkCmdSetCoverageToColorLocationNV coverageToColorLocation, with a VkFormat of VK_FORMAT_R8_UINT, VK_FORMAT_R8_SINT, VK_FORMAT_R16_UINT, VK_FORMAT_R16_SINT, VK_FORMAT_R32_UINT, or VK_FORMAT_R32_SINT

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-rasterizerDiscardEnable-09420
  If the VK_NV_fragment_coverage_to_color extension is enabled, and a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the last call to vkCmdSetCoverageToColorEnableNV set the coverageToColorEnable to VK_TRUE, then the current subpass **must** have a color attachment at the location selected by the last call to vkCmdSetCoverageToColorLocationNV coverageToColorLocation, with a VkFormat of VK_FORMAT_R8_UINT, VK_FORMAT_R8_SINT, VK_FORMAT_R16_UINT, VK_FORMAT_R16_SINT,

VK_FORMAT_R32_UINT, or VK_FORMAT_R32_SINT

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-coverageReductionMode-07491
  If this VK_NV_coverage_reduction_mode extension is enabled, the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COVERAGE_TO_COLOR_ENABLE_NV and VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT states enabled, the current coverage reduction mode coverageReductionMode, then the current rasterizationSamples, and the sample counts for the color and depth/stencil attachments (if the subpass has them) **must** be a valid combination returned by vkGetPhysicalDeviceSupportedFramebufferMixedSamplesCombinationsNV
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-viewportCount-07492
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, but not the VK_DYNAMIC_STATE_VIEWPORT_SWIZZLE_NV dynamic state enabled, then the bound graphics pipeline **must** have been created with VkPipelineViewportSwizzleStateCreateInfoNV::viewportCount greater or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-viewportCount-07493
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT and VK_DYNAMIC_STATE_VIEWPORT_SWIZZLE_NV dynamic states enabled then the viewportCount parameter in the last call to vkCmdSetViewportSwizzleNV **must** be greater than or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-viewportCount-09421
  If the VK_NV_viewport_swizzle extension is enabled, and a shader object is bound to any graphics stage, then the viewportCount parameter in the last call to vkCmdSetViewportSwizzleNV **must** be greater than or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-rasterizationSamples-07494
  If the VK_NV_framebuffer_mixed_samples extension is enabled, and if the current subpass has any color attachments and rasterizationSamples of the last call to vkCmdSetRasterizationSamplesEXT is greater than the number of color samples, then the pipeline sampleShadingEnable **must** be VK_FALSE
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-stippledLineEnable-07495
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT or VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT dynamic states enabled, and if the current stippledLineEnable state is VK_TRUE and the current lineRasterizationMode state is VK_LINE_RASTERIZATION_MODE_RECTANGULAR_KHR, then the stippledRectangularLines feature **must** be enabled
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-stippledLineEnable-07496
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT or VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT dynamic states enabled, and if the current stippledLineEnable state is VK_TRUE and the current lineRasterizationMode state is VK_LINE_RASTERIZATION_MODE_BRESENHAM_KHR, then the stippledBresenhamLines feature **must**

be enabled

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-stippledLineEnable-07497
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT or VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT dynamic states enabled, and if the current stippledLineEnable state is VK_TRUE and the current lineRasterizationMode state is VK_LINE_RASTERIZATION_MODE_RECTANGULAR_SMOOTH_KHR, then the stippledSmoothLines feature **must** be enabled

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-stippledLineEnable-07498
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT or VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT dynamic states enabled, and if the current stippledLineEnable state is VK_TRUE and the current lineRasterizationMode state is VK_LINE_RASTERIZATION_MODE_DEFAULT_KHR, then the stippledRectangularLines feature **must** be enabled and VkPhysicalDeviceLimits::strictLines **must** be VK_TRUE

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-conservativePointAndLineRasterization-07499
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_CONSERVATIVE_RASTERIZATION_MODE_EXT dynamic state enabled, conservativePointAndLineRasterization is not supported, and the effective primitive topology output by the last pre-rasterization shader stage is a line or point, then the conservativeRasterizationMode set by the last call to vkCmdSetConservativeRasterizationModeEXT **must** be VK_CONSERVATIVE_RASTERIZATION_MODE_DISABLED_EXT

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-stage-07073
  If the currently bound pipeline was created with the VkPipelineShaderStageCreateInfo::stage member of an element of VkGraphicsPipelineCreateInfo::pStages set to VK_SHADER_STAGE_VERTEX_BIT, VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT, VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT or VK_SHADER_STAGE_GEOMETRY_BIT, then Mesh Shader Queries **must** not be active

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-08877
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_ATTACHMENT_FEEDBACK_LOOP_ENABLE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetAttachmentFeedbackLoopEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-07850
  If dynamic state was inherited from VkCommandBufferInheritanceViewportScissorInfoNV, it **must** be set in the current command buffer prior to this drawing command

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-08684
  If there is no bound graphics pipeline, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_VERTEX_BIT

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-08685

If there is no bound graphics pipeline, and the tessellationShader feature is enabled, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-08686
  If there is no bound graphics pipeline, and the tessellationShader feature is enabled, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-08687
  If there is no bound graphics pipeline, and the geometryShader feature is enabled, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_GEOMETRY_BIT
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-08688
  If there is no bound graphics pipeline, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_FRAGMENT_BIT
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-08689
  If there is no bound graphics pipeline, and the taskShader feature is enabled, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_TASK_BIT_EXT
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-08690
  If there is no bound graphics pipeline, and the meshShader feature is enabled, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_MESH_BIT_EXT
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-08693
  If there is no bound graphics pipeline, and at least one of the taskShader and meshShader features is enabled, one of the VK_SHADER_STAGE_VERTEX_BIT or VK_SHADER_STAGE_MESH_BIT_EXT stages **must** have a valid VkShaderEXT bound, and the other **must** have no VkShaderEXT bound
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-08694
  If there is no bound graphics pipeline, and both the taskShader and meshShader features are enabled, and a valid VkShaderEXT is bound the to the VK_SHADER_STAGE_MESH_BIT_EXT stage, and that VkShaderEXT was created without the VK_SHADER_CREATE_NO_TASK_SHADER_BIT_EXT flag, a valid VkShaderEXT **must** be bound to the VK_SHADER_STAGE_TASK_BIT_EXT stage
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-08695
  If there is no bound graphics pipeline, and both the taskShader and meshShader features are enabled, and a valid VkShaderEXT is bound the to the VK_SHADER_STAGE_MESH_BIT_EXT stage, and that VkShaderEXT was created with the VK_SHADER_CREATE_NO_TASK_SHADER_BIT_EXT flag, there **must** be no VkShaderEXT bound to the VK_SHADER_STAGE_TASK_BIT_EXT stage
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-08696
  If there is no bound graphics pipeline, and a valid VkShaderEXT is bound to the VK_SHADER_STAGE_VERTEX_BIT stage, there **must** be no VkShaderEXT bound to either the VK_SHADER_STAGE_TASK_BIT_EXT stage or the VK_SHADER_STAGE_MESH_BIT_EXT stage
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-08698
  If any graphics shader is bound which was created with the

VK_SHADER_CREATE_LINK_STAGE_BIT_EXT flag, then all shaders created with the VK_SHADER_CREATE_LINK_STAGE_BIT_EXT flag in the same vkCreateShadersEXT call **must** also be bound

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-08699
  If any graphics shader is bound which was created with the VK_SHADER_CREATE_LINK_STAGE_BIT_EXT flag, any stages in between stages whose shaders which did not create a shader with the VK_SHADER_CREATE_LINK_STAGE_BIT_EXT flag as part of the same vkCreateShadersEXT call **must** not have any VkShaderEXT bound
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-08878
  All bound graphics shader objects **must** have been created with identical or identically defined push constant ranges
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-08879
  All bound graphics shader objects **must** have been created with identical or identically defined arrays of descriptor set layouts
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-colorAttachmentCount-09372
  If the current render pass instance was begun with vkCmdBeginRenderingKHR and a VkRenderingInfo::colorAttachmentCount equal to 1, a color attachment with a resolve mode of VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID, and a fragment shader is bound, it **must** not declare the DepthReplacing or StencilRefReplacingEXT execution modes
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-pDynamicStates-08715
  If the bound graphics pipeline state includes a fragment shader stage, was created with VK_DYNAMIC_STATE_DEPTH_WRITE_ENABLE set in VkPipelineDynamicStateCreateInfo::pDynamicStates, and the fragment shader declares the EarlyFragmentTests execution mode and uses OpDepthAttachmentReadEXT, the depthWriteEnable parameter in the last call to vkCmdSetDepthWriteEnableEXT **must** be VK_FALSE
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-pDynamicStates-08716
  If the bound graphics pipeline state includes a fragment shader stage, was created with VK_DYNAMIC_STATE_STENCIL_WRITE_MASK set in VkPipelineDynamicStateCreateInfo::pDynamicStates, and the fragment shader declares the EarlyFragmentTests execution mode and uses OpStencilAttachmentReadEXT, the writeMask parameter in the last call to vkCmdSetStencilWriteMask **must** be 0
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-09116
  If a shader object is bound to any graphics stage or the currently bound graphics pipeline was created with VK_DYNAMIC_STATE_COLOR_WRITE_MASK_EXT, and the format of any color attachment is VK_FORMAT_E5B9G9R9_UFLOAT_PACK32, the corresponding element of the pColorWriteMasks parameter of vkCmdSetColorWriteMaskEXT **must** either include all of VK_COLOR_COMPONENT_R_BIT, VK_COLOR_COMPONENT_G_BIT, and VK_COLOR_COMPONENT_B_BIT, or none of them
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-maxFragmentDualSrcAttachments-09239
  If blending is enabled for any attachment where either the source or destination blend factors for that attachment use the secondary color input, the maximum value of Location for any output attachment statically used in the Fragment Execution Model executed by this command **must** be less than maxFragmentDualSrcAttachments
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-09548

If the current render pass was begun with vkCmdBeginRenderingKHR, and there is no shader object bound to any graphics stage, the value of each element of VkRenderingAttachmentLocationInfoKHR::pColorAttachmentLocations set by vkCmdSetRenderingAttachmentLocationsKHR **must** match the value set for the corresponding element in the currently bound pipeline

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-09549
  If the current render pass was begun with vkCmdBeginRenderingKHR, and there is no shader object bound to any graphics stage, input attachment index mappings in the currently bound pipeline **must** match those set for the current render pass instance via VkRenderingInputAttachmentIndexInfoKHR

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-09642
  If the current render pass was begun with vkCmdBeginRenderingKHR with the VK_RENDERING_ENABLE_LEGACY_DITHERING_BIT_EXT flag, the bound graphics pipeline **must** have been created with VK_PIPELINE_CREATE_2_ENABLE_LEGACY_DITHERING_BIT_EXT

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-09643
  If the bound graphics pipeline was created with VK_PIPELINE_CREATE_2_ENABLE_LEGACY_DITHERING_BIT_EXT, the current render pass **must** have begun with vkCmdBeginRenderingKHR with the VK_RENDERING_ENABLE_LEGACY_DITHERING_BIT_EXT flag

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-stage-06480
  The bound graphics pipeline **must** not have been created with the VkPipelineShaderStageCreateInfo::stage member of an element of VkGraphicsPipelineCreateInfo::pStages set to VK_SHADER_STAGE_VERTEX_BIT, VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT, VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT or VK_SHADER_STAGE_GEOMETRY_BIT

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-07074
  Transform Feedback Queries **must** not be active

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-None-07075
  Primitives Generated Queries **must** not be active

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-pipelineStatistics-07076
  The pipelineStatistics member used to create any active Pipeline Statistics Query **must** not contain VK_QUERY_PIPELINE_STATISTIC_INPUT_ASSEMBLY_VERTICES_BIT, VK_QUERY_PIPELINE_STATISTIC_INPUT_ASSEMBLY_PRIMITIVES_BIT, VK_QUERY_PIPELINE_STATISTIC_VERTEX_SHADER_INVOCATIONS_BIT, VK_QUERY_PIPELINE_STATISTIC_GEOMETRY_SHADER_INVOCATIONS_BIT, VK_QUERY_PIPELINE_STATISTIC_GEOMETRY_SHADER_PRIMITIVES_BIT, VK_QUERY_PIPELINE_STATISTIC_CLIPPING_INVOCATIONS_BIT, VK_QUERY_PIPELINE_STATISTIC_CLIPPING_PRIMITIVES_BIT, VK_QUERY_PIPELINE_STATISTIC_TESSELLATION_CONTROL_SHADER_PATCHES_BIT, or VK_QUERY_PIPELINE_STATISTIC_TESSELLATION_EVALUATION_SHADER_INVOCATIONS_BIT

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-buffer-02708
  If buffer is non-sparse then it **must** be bound completely and contiguously to a single VkDeviceMemory object

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-buffer-02709
  `buffer` **must** have been created with the `VK_BUFFER_USAGE_INDIRECT_BUFFER_BIT` bit set
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-offset-02710
  `offset` **must** be a multiple of `4`
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-commandBuffer-02711
  `commandBuffer` **must** not be a protected command buffer
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-countBuffer-02714
  If `countBuffer` is non-sparse then it **must** be bound completely and contiguously to a single `VkDeviceMemory` object
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-countBuffer-02715
  `countBuffer` **must** have been created with the `VK_BUFFER_USAGE_INDIRECT_BUFFER_BIT` bit set
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-countBufferOffset-02716
  `countBufferOffset` **must** be a multiple of `4`
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-countBuffer-02717
  The count stored in `countBuffer` **must** be less than or equal to `VkPhysicalDeviceLimits::maxDrawIndirectCount`
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-countBufferOffset-04129
  (`countBufferOffset` + `sizeof`(uint32_t)) **must** be less than or equal to the size of `countBuffer`
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-stride-07096
  `stride` **must** be a multiple of `4` and **must** be greater than or equal to `sizeof`(`VkDrawMeshTasksIndirectCommandEXT`)
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-maxDrawCount-07097
  If `maxDrawCount` is greater than or equal to `1`, (`stride` × (`maxDrawCount` - 1) + `offset` + `sizeof`(`VkDrawMeshTasksIndirectCommandEXT`)) **must** be less than or equal to the size of `buffer`
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-countBuffer-07098
  If the count stored in `countBuffer` is equal to `1`, (`offset` + `sizeof`(`VkDrawMeshTasksIndirectCommandEXT`)) **must** be less than or equal to the size of `buffer`
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-countBuffer-07099
  If the count stored in `countBuffer` is greater than `1`, (`stride` × (`drawCount` - 1) + `offset` + `sizeof`(`VkDrawMeshTasksIndirectCommandEXT`)) **must** be less than or equal to the size of `buffer`
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-MeshEXT-07100
  The current pipeline bound to `VK_PIPELINE_BIND_POINT_GRAPHICS` **must** contain a shader stage using the `MeshEXT Execution Model`

## Valid Usage (Implicit)

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-buffer-parameter
  `buffer` **must** be a valid VkBuffer handle
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-countBuffer-parameter

countBuffer **must** be a valid VkBuffer handle

- VUID-vkCmdDrawMeshTasksIndirectCountEXT-commandBuffer-recording
  commandBuffer **must** be in the recording state
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-commandBuffer-cmdpool
  The VkCommandPool that commandBuffer was allocated from **must** support graphics operations
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-renderpass
  This command **must** only be called inside of a render pass instance
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-videocoding
  This command **must** only be called outside of a video coding scope
- VUID-vkCmdDrawMeshTasksIndirectCountEXT-commonparent
  Each of buffer, commandBuffer, and countBuffer **must** have been created, allocated, or retrieved from the same VkDevice

### Host Synchronization

- Host access to commandBuffer **must** be externally synchronized
- Host access to the VkCommandPool that commandBuffer was allocated from **must** be externally synchronized

### Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Inside | Outside | Graphics | Action |

## 21.6. Programmable Cluster Culling Shading

In this drawing approach, cluster are generated by the cluster culling shader stage. It operates similarly to dispatching compute as the shaders make use of workgroups.

To record a cluster culling shader drawing command, call:

```
// Provided by VK_HUAWEI_cluster_culling_shader
void vkCmdDrawClusterHUAWEI(
    VkCommandBuffer                             commandBuffer,
    uint32_t                                    groupCountX,
    uint32_t                                    groupCountY,
    uint32_t                                    groupCountZ);
```

- commandBuffer is the command buffer into which the command will be recorded.
- groupCountX is the number of local workgroups to dispatch in the X dimension.
- groupCountY is the number of local workgroups to dispatch in the Y dimension.
- groupCountZ is the number of local workgroups to dispatch in the Z dimension.

When the command is executed,a global workgroup consisting of groupCountX*groupCountY*groupCountZ local workgroup is assembled. Note that the cluster culling shader pipeline only accepts vkCmdDrawClusterHUAWEI and vkCmdDrawClusterIndirectHUAWEI as drawing commands.

## Valid Usage

- VUID-vkCmdDrawClusterHUAWEI-magFilter-04553
  If a VkSampler created with magFilter or minFilter equal to VK_FILTER_LINEAR, reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE, and compareEnable equal to VK_FALSE is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT
- VUID-vkCmdDrawClusterHUAWEI-magFilter-09598
  If a VkSampler created with magFilter or minFilter equal to VK_FILTER_LINEAR and reductionMode equal to either VK_SAMPLER_REDUCTION_MODE_MIN or VK_SAMPLER_REDUCTION_MODE_MAX is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT
- VUID-vkCmdDrawClusterHUAWEI-mipmapMode-04770
  If a VkSampler created with mipmapMode equal to VK_SAMPLER_MIPMAP_MODE_LINEAR, reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE, and compareEnable equal to VK_FALSE is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT
- VUID-vkCmdDrawClusterHUAWEI-mipmapMode-09599
  If a VkSampler created with mipmapMode equal to VK_SAMPLER_MIPMAP_MODE_LINEAR and reductionMode equal to either VK_SAMPLER_REDUCTION_MODE_MIN or VK_SAMPLER_REDUCTION_MODE_MAX is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT
- VUID-vkCmdDrawClusterHUAWEI-unnormalizedCoordinates-09635
  If a VkSampler created with unnormalizedCoordinates equal to VK_TRUE is used to sample a VkImageView as a result of this command, then the image view's levelCount and layerCount **must** be 1
- VUID-vkCmdDrawClusterHUAWEI-unnormalizedCoordinates-09636
  If a VkSampler created with unnormalizedCoordinates equal to VK_TRUE is used to sample a VkImageView as a result of this command, then the image view's viewType **must** be VK_IMAGE_VIEW_TYPE_1D or VK_IMAGE_VIEW_TYPE_2D

- VUID-vkCmdDrawClusterHUAWEI-None-06479
  If a VkImageView is sampled with depth comparison, the image view's format features **must** contain VK_FORMAT_FEATURE_2_SAMPLED_IMAGE_DEPTH_COMPARISON_BIT
- VUID-vkCmdDrawClusterHUAWEI-None-02691
  If a VkImageView is accessed using atomic operations as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_STORAGE_IMAGE_ATOMIC_BIT
- VUID-vkCmdDrawClusterHUAWEI-None-07888
  If a VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER descriptor is accessed using atomic operations as a result of this command, then the storage texel buffer's format features **must** contain VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_ATOMIC_BIT
- VUID-vkCmdDrawClusterHUAWEI-None-02692
  If a VkImageView is sampled with VK_FILTER_CUBIC_EXT as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_CUBIC_BIT_EXT
- VUID-vkCmdDrawClusterHUAWEI-None-02693
  If the VK_EXT_filter_cubic extension is not enabled and any VkImageView is sampled with VK_FILTER_CUBIC_EXT as a result of this command, it **must** not have a VkImageViewType of VK_IMAGE_VIEW_TYPE_3D, VK_IMAGE_VIEW_TYPE_CUBE, or VK_IMAGE_VIEW_TYPE_CUBE_ARRAY
- VUID-vkCmdDrawClusterHUAWEI-filterCubic-02694
  Any VkImageView being sampled with VK_FILTER_CUBIC_EXT as a result of this command **must** have a VkImageViewType and format that supports cubic filtering, as specified by VkFilterCubicImageViewImageFormatPropertiesEXT::filterCubic returned by vkGetPhysicalDeviceImageFormatProperties2
- VUID-vkCmdDrawClusterHUAWEI-filterCubicMinmax-02695
  Any VkImageView being sampled with VK_FILTER_CUBIC_EXT with a reduction mode of either VK_SAMPLER_REDUCTION_MODE_MIN or VK_SAMPLER_REDUCTION_MODE_MAX as a result of this command **must** have a VkImageViewType and format that supports cubic filtering together with minmax filtering, as specified by VkFilterCubicImageViewImageFormatPropertiesEXT::filterCubicMinmax returned by vkGetPhysicalDeviceImageFormatProperties2
- VUID-vkCmdDrawClusterHUAWEI-cubicRangeClamp-09212
  If the cubicRangeClamp feature is not enabled, then any VkImageView being sampled with VK_FILTER_CUBIC_EXT as a result of this command **must** not have a VkSamplerReductionModeCreateInfo::reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE_RANGECLAMP_QCOM
- VUID-vkCmdDrawClusterHUAWEI-reductionMode-09213
  Any VkImageView being sampled with a VkSamplerReductionModeCreateInfo::reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE_RANGECLAMP_QCOM as a result of this command **must** sample with VK_FILTER_CUBIC_EXT
- VUID-vkCmdDrawClusterHUAWEI-selectableCubicWeights-09214
  If the selectableCubicWeights feature is not enabled, then any VkImageView being sampled with VK_FILTER_CUBIC_EXT as a result of this command **must** have VkSamplerCubicWeightsCreateInfoQCOM::cubicWeights equal to

VK_CUBIC_FILTER_WEIGHTS_CATMULL_ROM_QCOM

- VUID-vkCmdDrawClusterHUAWEI-flags-02696
  Any VkImage created with a VkImageCreateInfo::flags containing VK_IMAGE_CREATE_CORNER_SAMPLED_BIT_NV sampled as a result of this command **must** only be sampled using a VkSamplerAddressMode of VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_EDGE
- VUID-vkCmdDrawClusterHUAWEI-OpTypeImage-07027
  For any VkImageView being written as a storage image where the image format field of the OpTypeImage is Unknown, the view's format features **must** contain VK_FORMAT_FEATURE_2_STORAGE_WRITE_WITHOUT_FORMAT_BIT
- VUID-vkCmdDrawClusterHUAWEI-OpTypeImage-07028
  For any VkImageView being read as a storage image where the image format field of the OpTypeImage is Unknown, the view's format features **must** contain VK_FORMAT_FEATURE_2_STORAGE_READ_WITHOUT_FORMAT_BIT
- VUID-vkCmdDrawClusterHUAWEI-OpTypeImage-07029
  For any VkBufferView being written as a storage texel buffer where the image format field of the OpTypeImage is Unknown, the view's buffer features **must** contain VK_FORMAT_FEATURE_2_STORAGE_WRITE_WITHOUT_FORMAT_BIT
- VUID-vkCmdDrawClusterHUAWEI-OpTypeImage-07030
  Any VkBufferView being read as a storage texel buffer where the image format field of the OpTypeImage is Unknown then the view's buffer features **must** contain VK_FORMAT_FEATURE_2_STORAGE_READ_WITHOUT_FORMAT_BIT
- VUID-vkCmdDrawClusterHUAWEI-None-08600
  For each set *n* that is statically used by a bound shader, a descriptor set **must** have been bound to *n* at the same pipeline bind point, with a VkPipelineLayout that is compatible for set *n*, with the VkPipelineLayout used to create the current VkPipeline or the VkDescriptorSetLayout array used to create the current VkShaderEXT , as described in Pipeline Layout Compatibility
- VUID-vkCmdDrawClusterHUAWEI-None-08601
  For each push constant that is statically used by a bound shader, a push constant value **must** have been set for the same pipeline bind point, with a VkPipelineLayout that is compatible for push constants, with the VkPipelineLayout used to create the current VkPipeline or the VkDescriptorSetLayout array used to create the current VkShaderEXT , as described in Pipeline Layout Compatibility
- VUID-vkCmdDrawClusterHUAWEI-None-10068
  For each array of resources that is used by a bound shader, the indices used to access members of the array **must** be less than the descriptor count for the identified binding in the descriptor sets used by this command
- VUID-vkCmdDrawClusterHUAWEI-maintenance4-08602
  If the maintenance4 feature is not enabled, then for each push constant that is statically used by a bound shader, a push constant value **must** have been set for the same pipeline bind point, with a VkPipelineLayout that is compatible for push constants, with the VkPipelineLayout used to create the current VkPipeline or the VkDescriptorSetLayout and VkPushConstantRange arrays used to create the current VkShaderEXT , as described in Pipeline Layout Compatibility

- VUID-vkCmdDrawClusterHUAWEI-None-08114
  Descriptors in each bound descriptor set, specified via vkCmdBindDescriptorSets, **must** be valid as described by descriptor validity if they are statically used by the VkPipeline bound to the pipeline bind point used by this command and the bound VkPipeline was not created with `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`
- VUID-vkCmdDrawClusterHUAWEI-None-08115
  If the descriptors used by the VkPipeline bound to the pipeline bind point were specified via vkCmdBindDescriptorSets, the bound VkPipeline **must** have been created without `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`
- VUID-vkCmdDrawClusterHUAWEI-None-08116
  Descriptors in bound descriptor buffers, specified via vkCmdSetDescriptorBufferOffsetsEXT, **must** be valid if they are dynamically used by the VkPipeline bound to the pipeline bind point used by this command and the bound VkPipeline was created with `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`
- VUID-vkCmdDrawClusterHUAWEI-None-08604
  Descriptors in bound descriptor buffers, specified via vkCmdSetDescriptorBufferOffsetsEXT, **must** be valid if they are dynamically used by any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command
- VUID-vkCmdDrawClusterHUAWEI-None-08117
  If the descriptors used by the VkPipeline bound to the pipeline bind point were specified via vkCmdSetDescriptorBufferOffsetsEXT, the bound VkPipeline **must** have been created with `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`
- VUID-vkCmdDrawClusterHUAWEI-None-08119
  If a descriptor is dynamically used with a VkPipeline created with `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`, the descriptor memory **must** be resident
- VUID-vkCmdDrawClusterHUAWEI-None-08605
  If a descriptor is dynamically used with a VkShaderEXT created with a `VkDescriptorSetLayout` that was created with `VK_DESCRIPTOR_SET_LAYOUT_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`, the descriptor memory **must** be resident
- VUID-vkCmdDrawClusterHUAWEI-None-08606
  If the `shaderObject` feature is not enabled, a valid pipeline **must** be bound to the pipeline bind point used by this command
- VUID-vkCmdDrawClusterHUAWEI-None-08608
  If a pipeline is bound to the pipeline bind point used by this command, there **must** not have been any calls to dynamic state setting commands for any state specified statically in the VkPipeline object bound to the pipeline bind point used by this command, since that pipeline was bound
- VUID-vkCmdDrawClusterHUAWEI-None-08609
  If the VkPipeline object bound to the pipeline bind point used by this command or any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a VkSampler object that uses unnormalized coordinates, that sampler **must** not be used to sample from any VkImage with a VkImageView of the type

VK_IMAGE_VIEW_TYPE_3D, VK_IMAGE_VIEW_TYPE_CUBE, VK_IMAGE_VIEW_TYPE_1D_ARRAY, VK_IMAGE_VIEW_TYPE_2D_ARRAY or VK_IMAGE_VIEW_TYPE_CUBE_ARRAY, in any shader stage

- VUID-vkCmdDrawClusterHUAWEI-None-08610
  If the VkPipeline object bound to the pipeline bind point used by this command or any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a VkSampler object that uses unnormalized coordinates, that sampler **must** not be used with any of the SPIR-V OpImageSample* or OpImageSparseSample* instructions with ImplicitLod, Dref or Proj in their name, in any shader stage
- VUID-vkCmdDrawClusterHUAWEI-None-08611
  If the VkPipeline object bound to the pipeline bind point used by this command or any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a VkSampler object that uses unnormalized coordinates, that sampler **must** not be used with any of the SPIR-V OpImageSample* or OpImageSparseSample* instructions that includes a LOD bias or any offset values, in any shader stage
- VUID-vkCmdDrawClusterHUAWEI-None-08607
  If the shaderObject is enabled, either a valid pipeline **must** be bound to the pipeline bind point used by this command, or a valid combination of valid and VK_NULL_HANDLE shader objects **must** be bound to every supported shader stage corresponding to the pipeline bind point used by this command
- VUID-vkCmdDrawClusterHUAWEI-uniformBuffers-06935
  If any stage of the VkPipeline object bound to the pipeline bind point used by this command accesses a uniform buffer, and that stage was created without enabling either VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_EXT or VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_2_EXT for uniformBuffers, and the robustBufferAccess feature is not enabled, that stage **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point
- VUID-vkCmdDrawClusterHUAWEI-None-08612
  If the robustBufferAccess feature is not enabled, and any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a uniform buffer, it **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point
- VUID-vkCmdDrawClusterHUAWEI-storageBuffers-06936
  If any stage of the VkPipeline object bound to the pipeline bind point used by this command accesses a storage buffer, and that stage was created without enabling either VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_EXT or VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_2_EXT for storageBuffers, and the robustBufferAccess feature is not enabled, that stage **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point
- VUID-vkCmdDrawClusterHUAWEI-None-08613
  If the robustBufferAccess feature is not enabled, and any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a storage buffer, it **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point

- VUID-vkCmdDrawClusterHUAWEI-commandBuffer-02707
  If `commandBuffer` is an unprotected command buffer and `protectedNoFault` is not supported, any resource accessed by bound shaders **must** not be a protected resource
- VUID-vkCmdDrawClusterHUAWEI-None-06550
  If a bound shader accesses a VkSampler or VkImageView object that enables sampler $Y'C_BC_R$ conversion, that object **must** only be used with `OpImageSample*` or `OpImageSparseSample*` instructions
- VUID-vkCmdDrawClusterHUAWEI-ConstOffset-06551
  If a bound shader accesses a VkSampler or VkImageView object that enables sampler $Y'C_BC_R$ conversion, that object **must** not use the `ConstOffset` and `Offset` operands
- VUID-vkCmdDrawClusterHUAWEI-viewType-07752
  If a VkImageView is accessed as a result of this command, then the image view's `viewType` **must** match the `Dim` operand of the `OpTypeImage` as described in Compatibility Between SPIR-V Image Dimensions and Vulkan ImageView Types
- VUID-vkCmdDrawClusterHUAWEI-format-07753
  If a VkImageView is accessed as a result of this command, then the numeric type of the image view's `format` and the `Sampled Type` operand of the `OpTypeImage` **must** match
- VUID-vkCmdDrawClusterHUAWEI-OpImageWrite-08795
  If a VkImageView created with a format other than `VK_FORMAT_A8_UNORM_KHR` is accessed using `OpImageWrite` as a result of this command, then the `Type` of the `Texel` operand of that instruction **must** have at least as many components as the image view's format
- VUID-vkCmdDrawClusterHUAWEI-OpImageWrite-08796
  If a VkImageView created with the format `VK_FORMAT_A8_UNORM_KHR` is accessed using `OpImageWrite` as a result of this command, then the `Type` of the `Texel` operand of that instruction **must** have four components
- VUID-vkCmdDrawClusterHUAWEI-OpImageWrite-04469
  If a VkBufferView is accessed using `OpImageWrite` as a result of this command, then the `Type` of the `Texel` operand of that instruction **must** have at least as many components as the buffer view's format
- VUID-vkCmdDrawClusterHUAWEI-SampledType-04470
  If a VkImageView with a VkFormat that has a 64-bit component width is accessed as a result of this command, the `SampledType` of the `OpTypeImage` operand of that instruction **must** have a `Width` of 64
- VUID-vkCmdDrawClusterHUAWEI-SampledType-04471
  If a VkImageView with a VkFormat that has a component width less than 64-bit is accessed as a result of this command, the `SampledType` of the `OpTypeImage` operand of that instruction **must** have a `Width` of 32
- VUID-vkCmdDrawClusterHUAWEI-SampledType-04472
  If a VkBufferView with a VkFormat that has a 64-bit component width is accessed as a result of this command, the `SampledType` of the `OpTypeImage` operand of that instruction **must** have a `Width` of 64
- VUID-vkCmdDrawClusterHUAWEI-SampledType-04473
  If a VkBufferView with a VkFormat that has a component width less than 64-bit is

accessed as a result of this command, the SampledType of the OpTypeImage operand of that instruction **must** have a Width of 32

- VUID-vkCmdDrawClusterHUAWEI-sparseImageInt64Atomics-04474
  If the sparseImageInt64Atomics feature is not enabled, VkImage objects created with the VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT flag **must** not be accessed by atomic instructions through an OpTypeImage with a SampledType with a Width of 64 by this command

- VUID-vkCmdDrawClusterHUAWEI-sparseImageInt64Atomics-04475
  If the sparseImageInt64Atomics feature is not enabled, VkBuffer objects created with the VK_BUFFER_CREATE_SPARSE_RESIDENCY_BIT flag **must** not be accessed by atomic instructions through an OpTypeImage with a SampledType with a Width of 64 by this command

- VUID-vkCmdDrawClusterHUAWEI-OpImageWeightedSampleQCOM-06971
  If OpImageWeightedSampleQCOM is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_WEIGHT_SAMPLED_IMAGE_BIT_QCOM

- VUID-vkCmdDrawClusterHUAWEI-OpImageWeightedSampleQCOM-06972
  If OpImageWeightedSampleQCOM uses a VkImageView as a sample weight image as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_WEIGHT_IMAGE_BIT_QCOM

- VUID-vkCmdDrawClusterHUAWEI-OpImageBoxFilterQCOM-06973
  If OpImageBoxFilterQCOM is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_BOX_FILTER_SAMPLED_BIT_QCOM

- VUID-vkCmdDrawClusterHUAWEI-OpImageBlockMatchSSDQCOM-06974
  If OpImageBlockMatchSSDQCOM is used to read from an VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM

- VUID-vkCmdDrawClusterHUAWEI-OpImageBlockMatchSADQCOM-06975
  If OpImageBlockMatchSADQCOM is used to read from an VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM

- VUID-vkCmdDrawClusterHUAWEI-OpImageBlockMatchSADQCOM-06976
  If OpImageBlockMatchSADQCOM or OpImageBlockMatchSSDQCOM is used to read from a reference image as result of this command, then the specified reference coordinates **must** not fail integer texel coordinate validation

- VUID-vkCmdDrawClusterHUAWEI-OpImageWeightedSampleQCOM-06977
  If OpImageWeightedSampleQCOM, OpImageBoxFilterQCOM, OpImageBlockMatchWindowSSDQCOM, OpImageBlockMatchWindowSADQCOM, OpImageBlockMatchGatherSSDQCOM, OpImageBlockMatchGatherSADQCOM, OpImageBlockMatchSSDQCOM, or OpImageBlockMatchSADQCOM uses a VkSampler as a result of this command, then the sampler **must** have been created with VK_SAMPLER_CREATE_IMAGE_PROCESSING_BIT_QCOM

- VUID-vkCmdDrawClusterHUAWEI-OpImageWeightedSampleQCOM-06978
  If any command other than OpImageWeightedSampleQCOM, OpImageBoxFilterQCOM, OpImageBlockMatchWindowSSDQCOM, OpImageBlockMatchWindowSADQCOM, OpImageBlockMatchGatherSSDQCOM, OpImageBlockMatchGatherSADQCOM,

OpImageBlockMatchSSDQCOM, or OpImageBlockMatchSADQCOM uses a VkSampler as a result of this command, then the sampler **must** not have been created with VK_SAMPLER_CREATE_IMAGE_PROCESSING_BIT_QCOM

- VUID-vkCmdDrawClusterHUAWEI-OpImageBlockMatchWindow-09215
  If a OpImageBlockMatchWindow*QCOM or OpImageBlockMatchGather*QCOM instruction is used to read from an VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM
- VUID-vkCmdDrawClusterHUAWEI-OpImageBlockMatchWindow-09216
  If a OpImageBlockMatchWindow*QCOM or OpImageBlockMatchGather*QCOM instruction is used to read from an VkImageView as a result of this command, then the image view's format **must** be a single-component format
- VUID-vkCmdDrawClusterHUAWEI-OpImageBlockMatchWindow-09217
  If a OpImageBlockMatchWindow*QCOM or OpImageBlockMatchGather*QCOM read from a reference image as result of this command, then the specified reference coordinates **must** not fail integer texel coordinate validation
- VUID-vkCmdDrawClusterHUAWEI-None-07288
  Any shader invocation executed by this command **must** terminate
- VUID-vkCmdDrawClusterHUAWEI-None-09600
  If a descriptor with type equal to any of VK_DESCRIPTOR_TYPE_SAMPLE_WEIGHT_IMAGE_QCOM, VK_DESCRIPTOR_TYPE_BLOCK_MATCH_IMAGE_QCOM, VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE, VK_DESCRIPTOR_TYPE_STORAGE_IMAGE, or VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT is accessed as a result of this command, the image subresource identified by that descriptor **must** be in the image layout identified when the descriptor was written
- VUID-vkCmdDrawClusterHUAWEI-renderPass-02684
  The current render pass **must** be compatible with the renderPass member of the VkGraphicsPipelineCreateInfo structure specified when creating the VkPipeline bound to VK_PIPELINE_BIND_POINT_GRAPHICS
- VUID-vkCmdDrawClusterHUAWEI-subpass-02685
  The subpass index of the current render pass **must** be equal to the subpass member of the VkGraphicsPipelineCreateInfo structure specified when creating the VkPipeline bound to VK_PIPELINE_BIND_POINT_GRAPHICS
- VUID-vkCmdDrawClusterHUAWEI-None-07748
  If any shader statically accesses an input attachment, a valid descriptor **must** be bound to the pipeline via a descriptor set
- VUID-vkCmdDrawClusterHUAWEI-OpTypeImage-07468
  If any shader executed by this pipeline accesses an OpTypeImage variable with a Dim operand of SubpassData, it **must** be decorated with an InputAttachmentIndex that corresponds to a valid input attachment in the current subpass
- VUID-vkCmdDrawClusterHUAWEI-None-07469
  Input attachment views accessed in a subpass **must** be created with the same VkFormat as the corresponding subpass definition, and be created with a VkImageView that is compatible with the attachment referenced by the subpass' pInputAttachments [InputAttachmentIndex] in the currently bound VkFramebuffer as specified by Fragment Input Attachment Compatibility

- VUID-vkCmdDrawClusterHUAWEI-pDepthInputAttachmentIndex-09595
  Input attachment views accessed in a dynamic render pass with a `InputAttachmentIndex` referenced by VkRenderingInputAttachmentIndexInfoKHR, or no `InputAttachmentIndex` if VkRenderingInputAttachmentIndexInfoKHR:`pDepthInputAttachmentIndex` or VkRenderingInputAttachmentIndexInfoKHR:`pStencilInputAttachmentIndex` are `NULL`, **must** be created with a VkImageView that is compatible with the corresponding color, depth, or stencil attachment in VkRenderingInfo
- VUID-vkCmdDrawClusterHUAWEI-pDepthInputAttachmentIndex-09596
  Input attachment views accessed in a dynamic render pass via a shader object **must** have an `InputAttachmentIndex` if both VkRenderingInputAttachmentIndexInfoKHR:`pDepthInputAttachmentIndex` and VkRenderingInputAttachmentIndexInfoKHR:`pStencilInputAttachmentIndex` are non-`NULL`
- VUID-vkCmdDrawClusterHUAWEI-InputAttachmentIndex-09597
  If an input attachment view accessed in a dynamic render pass via a shader object has an `InputAttachmentIndex`, the `InputAttachmentIndex` **must** match an index in VkRenderingInputAttachmentIndexInfoKHR
- VUID-vkCmdDrawClusterHUAWEI-None-06537
  Memory backing image subresources used as attachments in the current render pass **must** not be written in any way other than as an attachment by this command
- VUID-vkCmdDrawClusterHUAWEI-None-09000
  If a color attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass, it is not in the `VK_IMAGE_LAYOUT_ATTACHMENT_FEEDBACK_LOOP_OPTIMAL_EXT` image layout, and either:
  - the `VK_PIPELINE_CREATE_COLOR_ATTACHMENT_FEEDBACK_LOOP_BIT_EXT` is set on the currently bound pipeline or
  - the last call to vkCmdSetAttachmentFeedbackLoopEnableEXT included `VK_IMAGE_ASPECT_COLOR_BIT` and
    - there is no currently bound graphics pipeline or
    - the currently bound graphics pipeline was created with `VK_DYNAMIC_STATE_ATTACHMENT_FEEDBACK_LOOP_ENABLE_EXT`

  it **must** not be accessed in any way other than as an attachment by this command
- VUID-vkCmdDrawClusterHUAWEI-None-09001
  If a depth attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass, it is not in the `VK_IMAGE_LAYOUT_ATTACHMENT_FEEDBACK_LOOP_OPTIMAL_EXT` image layout, and either:
  - the `VK_PIPELINE_CREATE_DEPTH_STENCIL_ATTACHMENT_FEEDBACK_LOOP_BIT_EXT` is set on the currently bound pipeline or
  - the last call to vkCmdSetAttachmentFeedbackLoopEnableEXT included `VK_IMAGE_ASPECT_DEPTH_BIT` and
    - there is no currently bound graphics pipeline or
    - the currently bound graphics pipeline was created with

VK_DYNAMIC_STATE_ATTACHMENT_FEEDBACK_LOOP_ENABLE_EXT

it **must** not be accessed in any way other than as an attachment by this command

- VUID-vkCmdDrawClusterHUAWEI-None-09002
  If a stencil attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass, it is not in the VK_IMAGE_LAYOUT_ATTACHMENT_FEEDBACK_LOOP_OPTIMAL_EXT image layout, and either:
  - the VK_PIPELINE_CREATE_DEPTH_STENCIL_ATTACHMENT_FEEDBACK_LOOP_BIT_EXT is set on the currently bound pipeline or
  - the last call to vkCmdSetAttachmentFeedbackLoopEnableEXT included VK_IMAGE_ASPECT_STENCIL_BIT and
    - there is no currently bound graphics pipeline or
    - the currently bound graphics pipeline was created with VK_DYNAMIC_STATE_ATTACHMENT_FEEDBACK_LOOP_ENABLE_EXT

  it **must** not be accessed in any way other than as an attachment by this command

- VUID-vkCmdDrawClusterHUAWEI-None-09003
  If an attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass, it **must** not be accessed in any way other than as an attachment, storage image, or sampled image by this command
- VUID-vkCmdDrawClusterHUAWEI-None-06539
  If any previously recorded command in the current subpass accessed an image subresource used as an attachment in this subpass in any way other than as an attachment, this command **must** not write to that image subresource as an attachment
- VUID-vkCmdDrawClusterHUAWEI-None-06886
  If the current render pass instance uses a depth/stencil attachment with a read-only layout for the depth aspect, depth writes **must** be disabled
- VUID-vkCmdDrawClusterHUAWEI-None-06887
  If the current render pass instance uses a depth/stencil attachment with a read-only layout for the stencil aspect, both front and back writeMask are not zero, and stencil test is enabled, all stencil ops **must** be VK_STENCIL_OP_KEEP
- VUID-vkCmdDrawClusterHUAWEI-None-07831
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT dynamic state enabled then vkCmdSetViewport **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterHUAWEI-None-07832
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SCISSOR dynamic state enabled then vkCmdSetScissor **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterHUAWEI-None-07833
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_WIDTH dynamic state enabled then vkCmdSetLineWidth **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawClusterHUAWEI-None-08617
  If a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetPolygonModeEXT in the current command buffer set polygonMode to VK_POLYGON_MODE_LINE, vkCmdSetLineWidth **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterHUAWEI-None-08618
  If a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetPrimitiveTopologyEXT in the current command buffer set primitiveTopology to any line topology, vkCmdSetLineWidth **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterHUAWEI-None-08619
  If a shader object that outputs line primitives is bound to the VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT or VK_SHADER_STAGE_GEOMETRY_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, vkCmdSetLineWidth **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterHUAWEI-None-07834
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_BIAS dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of depthBiasEnable is VK_TRUE, then vkCmdSetDepthBounds or vkCmdSetDepthBias2EXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterHUAWEI-None-07835
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_BLEND_CONSTANTS dynamic state enabled then vkCmdSetBlendConstants **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterHUAWEI-None-08621
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetColorBlendEnableEXT in the current command buffer set any element of pColorBlendEnables to VK_TRUE, and the most recent call to vkCmdSetColorBlendEquationEXT in the current command buffer set the same element of pColorBlendEquations to a VkColorBlendEquationEXT structure with any VkBlendFactor member with a value of VK_BLEND_FACTOR_CONSTANT_COLOR, VK_BLEND_FACTOR_ONE_MINUS_CONSTANT_COLOR, VK_BLEND_FACTOR_CONSTANT_ALPHA, or VK_BLEND_FACTOR_ONE_MINUS_CONSTANT_ALPHA, vkCmdSetBlendConstants **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawClusterHUAWEI-None-07836
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_BOUNDS dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of depthBoundsTestEnable is VK_TRUE, then vkCmdSetDepthBounds **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterHUAWEI-None-07837
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_STENCIL_COMPARE_MASK dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of stencilTestEnable is VK_TRUE, then vkCmdSetStencilCompareMask **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterHUAWEI-None-07838
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_STENCIL_WRITE_MASK dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of stencilTestEnable is VK_TRUE, then vkCmdSetStencilWriteMask **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterHUAWEI-None-07839
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_STENCIL_REFERENCE dynamic state enabled, the current value of and rasterizerDiscardEnable is VK_FALSE, the current value of stencilTestEnable is VK_TRUE, then vkCmdSetStencilReference **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterHUAWEI-maxMultiviewInstanceIndex-02688
  If the draw is recorded in a render pass instance with multiview enabled, the maximum instance index **must** be less than or equal to VkPhysicalDeviceMultiviewProperties::maxMultiviewInstanceIndex
- VUID-vkCmdDrawClusterHUAWEI-sampleLocationsEnable-02689
  If the bound graphics pipeline was created with VkPipelineSampleLocationsStateCreateInfoEXT::sampleLocationsEnable set to VK_TRUE and the current subpass has a depth/stencil attachment, then that attachment **must** have been created with the VK_IMAGE_CREATE_SAMPLE_LOCATIONS_COMPATIBLE_DEPTH_BIT_EXT bit set
- VUID-vkCmdDrawClusterHUAWEI-None-06666
  If the VK_EXT_sample_locations extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of sampleLocationsEnable is VK_TRUE, then vkCmdSetSampleLocationsEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterHUAWEI-None-07840

If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_CULL_MODE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetCullModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawClusterHUAWEI-None-07841
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_FRONT_FACE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetFrontFaceEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterHUAWEI-None-07843
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_TEST_ENABLE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, vkCmdSetDepthTestEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterHUAWEI-None-07844
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_WRITE_ENABLE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetDepthWriteEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterHUAWEI-None-07845
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_COMPARE_OP dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of depthTestEnable is VK_TRUE, then vkCmdSetDepthCompareOpEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterHUAWEI-None-07846
  If the depthBounds feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_BOUNDS_TEST_ENABLE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetDepthBoundsTestEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterHUAWEI-None-07847
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_STENCIL_TEST_ENABLE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetStencilTestEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterHUAWEI-None-07848
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which

was created with the VK_DYNAMIC_STATE_STENCIL_OP dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, the current value of stencilTestEnable is VK_TRUE, then vkCmdSetStencilOpEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawClusterHUAWEI-viewportCount-03417
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, but not the VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT dynamic state enabled, then vkCmdSetViewportWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the viewportCount parameter of vkCmdSetViewportWithCountEXT **must** match the VkPipelineViewportStateCreateInfo::scissorCount of the pipeline
- VUID-vkCmdDrawClusterHUAWEI-scissorCount-03418
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT dynamic state enabled, but not the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, then vkCmdSetScissorWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the scissorCount parameter of vkCmdSetScissorWithCountEXT **must** match the VkPipelineViewportStateCreateInfo::viewportCount of the pipeline
- VUID-vkCmdDrawClusterHUAWEI-viewportCount-03419
  If the bound graphics pipeline state was created with both the VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT and VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic states enabled then both vkCmdSetViewportWithCountEXT and vkCmdSetScissorWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the viewportCount parameter of vkCmdSetViewportWithCountEXT **must** match the scissorCount parameter of vkCmdSetScissorWithCountEXT
- VUID-vkCmdDrawClusterHUAWEI-None-08635
  If a shader object is bound to any graphics stage, then both vkCmdSetViewportWithCountEXT and vkCmdSetScissorWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the viewportCount parameter of vkCmdSetViewportWithCountEXT **must** match the scissorCount parameter of vkCmdSetScissorWithCountEXT
- VUID-vkCmdDrawClusterHUAWEI-viewportCount-04137
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, but not the VK_DYNAMIC_STATE_VIEWPORT_W_SCALING_NV dynamic state enabled, then the bound graphics pipeline **must** have been created with VkPipelineViewportWScalingStateCreateInfoNV::viewportCount greater or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawClusterHUAWEI-viewportCount-04138
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT and VK_DYNAMIC_STATE_VIEWPORT_W_SCALING_NV dynamic states enabled then the viewportCount parameter in the last call to

vkCmdSetViewportWScalingNV **must** be greater than or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT

- VUID-vkCmdDrawClusterHUAWEI-None-09232
  If the VK_NV_clip_space_w_scaling extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetViewportWScalingEnableNV in the current command buffer set viewportWScalingEnable to VK_TRUE, then vkCmdSetViewportWScalingNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterHUAWEI-None-08636
  If the VK_NV_clip_space_w_scaling extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetViewportWScalingEnableNV in the current command buffer set viewportWScalingEnable to VK_TRUE, then the viewportCount parameter in the last call to vkCmdSetViewportWScalingNV **must** be greater than or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawClusterHUAWEI-viewportCount-04139
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, but not the VK_DYNAMIC_STATE_VIEWPORT_SHADING_RATE_PALETTE_NV dynamic state enabled, then the bound graphics pipeline **must** have been created with VkPipelineViewportShadingRateImageStateCreateInfoNV::viewportCount greater or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawClusterHUAWEI-viewportCount-04140
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT and VK_DYNAMIC_STATE_VIEWPORT_SHADING_RATE_PALETTE_NV dynamic states enabled then the viewportCount parameter in the last call to vkCmdSetViewportShadingRatePaletteNV **must** be greater than or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawClusterHUAWEI-shadingRateImage-09233
  If the shadingRateImage feature is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetCoarseSampleOrderNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterHUAWEI-shadingRateImage-09234
  If the shadingRateImage feature is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetShadingRateImageEnableNV in the current command buffer set shadingRateImageEnable to VK_TRUE, then vkCmdSetViewportShadingRatePaletteNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterHUAWEI-None-08637
  If the shadingRateImage feature is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current

command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetShadingRateImageEnableNV in the current command buffer set shadingRateImageEnable to VK_TRUE, then the viewportCount parameter in the last call to vkCmdSetViewportShadingRatePaletteNV **must** be greater than or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT

- VUID-vkCmdDrawClusterHUAWEI-VkPipelineVieportCreateInfo-04141
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled and a VkPipelineViewportSwizzleStateCreateInfoNV structure chained from VkPipelineViewportStateCreateInfo, then the bound graphics pipeline **must** have been created with VkPipelineViewportSwizzleStateCreateInfoNV::viewportCount greater or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawClusterHUAWEI-VkPipelineVieportCreateInfo-04142
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled and a VkPipelineViewportExclusiveScissorStateCreateInfoNV structure chained from VkPipelineViewportStateCreateInfo, then the bound graphics pipeline **must** have been created with VkPipelineViewportExclusiveScissorStateCreateInfoNV::exclusiveScissorCount greater or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawClusterHUAWEI-None-07878
  If the exclusiveScissor feature is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_EXCLUSIVE_SCISSOR_ENABLE_NV dynamic state enabled, then vkCmdSetExclusiveScissorEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterHUAWEI-None-07879
  If the exclusiveScissor feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_EXCLUSIVE_SCISSOR_NV dynamic state enabled, and the most recent call to vkCmdSetExclusiveScissorEnableNV in the current command buffer set any element of pExclusiveScissorEnables to VK_TRUE, then vkCmdSetExclusiveScissorNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterHUAWEI-None-04876
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_RASTERIZER_DISCARD_ENABLE dynamic state enabled, then vkCmdSetRasterizerDiscardEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterHUAWEI-None-04877
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_BIAS_ENABLE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetDepthBiasEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawClusterHUAWEI-logicOp-04878
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_LOGIC_OP_EXT dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of logicOpEnable is VK_TRUE, then vkCmdSetLogicOpEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterHUAWEI-primitiveFragmentShadingRateWithMultipleViewports-04552
  If the primitiveFragmentShadingRateWithMultipleViewports limit is not supported, the bound graphics pipeline was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, and any of the shader stages of the bound graphics pipeline write to the PrimitiveShadingRateKHR built-in, then vkCmdSetViewportWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the viewportCount parameter of vkCmdSetViewportWithCountEXT **must** be 1
- VUID-vkCmdDrawClusterHUAWEI-primitiveFragmentShadingRateWithMultipleViewports-08642
  If the primitiveFragmentShadingRateWithMultipleViewports limit is not supported, and any shader object bound to a graphics stage writes to the PrimitiveShadingRateKHR built-in, then vkCmdSetViewportWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the viewportCount parameter of vkCmdSetViewportWithCountEXT **must** be 1
- VUID-vkCmdDrawClusterHUAWEI-blendEnable-04727
  If rasterization is not disabled in the bound graphics pipeline, then for each color attachment in the subpass, if the corresponding image view's format features do not contain VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT, then the blendEnable member of the corresponding element of the pAttachments member of pColorBlendState **must** be VK_FALSE
- VUID-vkCmdDrawClusterHUAWEI-None-08643
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then for each color attachment in the render pass, if the corresponding image view's format features do not contain VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT, then the corresponding member of pColorBlendEnables in the most recent call to vkCmdSetColorBlendEnableEXT in the current command buffer that affected that attachment index **must** have been VK_FALSE
- VUID-vkCmdDrawClusterHUAWEI-multisampledRenderToSingleSampled-07284
  If rasterization is not disabled in the bound graphics pipeline, and none of the following is enabled:
  - the VK_AMD_mixed_attachment_samples extension
  - the VK_NV_framebuffer_mixed_samples extension
  - the multisampledRenderToSingleSampled feature

  then rasterizationSamples for the currently bound graphics pipeline **must** be the same as the current subpass color and/or depth/stencil attachments

- VUID-vkCmdDrawClusterHUAWEI-None-08644
  If a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and none of the following is enabled:

  - the VK_AMD_mixed_attachment_samples extension
  - the VK_NV_framebuffer_mixed_samples extension
  - the multisampledRenderToSingleSampled feature

  then the most recent call to vkCmdSetRasterizationSamplesEXT in the current command buffer **must** have set rasterizationSamples to be the same as the number of samples for the current render pass color and/or depth/stencil attachments

- VUID-vkCmdDrawClusterHUAWEI-None-08876
  If a shader object is bound to any graphics stage, the current render pass instance **must** have been begun with vkCmdBeginRenderingKHR
- VUID-vkCmdDrawClusterHUAWEI-imageView-06172
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the imageView member of pDepthAttachment is not VK_NULL_HANDLE, and the layout member of pDepthAttachment is VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL, this command **must** not write any values to the depth attachment
- VUID-vkCmdDrawClusterHUAWEI-imageView-06173
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the imageView member of pStencilAttachment is not VK_NULL_HANDLE, and the layout member of pStencilAttachment is VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL, this command **must** not write any values to the stencil attachment
- VUID-vkCmdDrawClusterHUAWEI-imageView-06174
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the imageView member of pDepthAttachment is not VK_NULL_HANDLE, and the layout member of pDepthAttachment is VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_STENCIL_ATTACHMENT_OPTIMAL, this command **must** not write any values to the depth attachment
- VUID-vkCmdDrawClusterHUAWEI-imageView-06175
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the imageView member of pStencilAttachment is not VK_NULL_HANDLE, and the layout member of pStencilAttachment is VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_STENCIL_READ_ONLY_OPTIMAL, this command **must** not write any values to the stencil attachment
- VUID-vkCmdDrawClusterHUAWEI-imageView-06176
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the imageView member of pDepthAttachment is not VK_NULL_HANDLE, and the layout member of pDepthAttachment is VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL, this command **must** not write any values to the depth attachment
- VUID-vkCmdDrawClusterHUAWEI-imageView-06177
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the imageView member of pStencilAttachment is not VK_NULL_HANDLE, and the layout member of pStencilAttachment is VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL, this

command **must** not write any values to the stencil attachment

- VUID-vkCmdDrawClusterHUAWEI-viewMask-06178
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the currently bound graphics pipeline **must** have been created with a VkPipelineRenderingCreateInfo::viewMask equal to VkRenderingInfo::viewMask

- VUID-vkCmdDrawClusterHUAWEI-colorAttachmentCount-06179
  If the dynamicRenderingUnusedAttachments feature is not enabled and the current render pass instance was begun with vkCmdBeginRenderingKHR, the currently bound graphics pipeline **must** have been created with a VkPipelineRenderingCreateInfo::colorAttachmentCount equal to VkRenderingInfo::colorAttachmentCount

- VUID-vkCmdDrawClusterHUAWEI-dynamicRenderingUnusedAttachments-08910
  If the dynamicRenderingUnusedAttachments feature is not enabled, and the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingInfo::colorAttachmentCount greater than 0, then each element of the VkRenderingInfo::pColorAttachments array with an imageView not equal to VK_NULL_HANDLE **must** have been created with a VkFormat equal to the corresponding element of VkPipelineRenderingCreateInfo::pColorAttachmentFormats used to create the currently bound graphics pipeline

- VUID-vkCmdDrawClusterHUAWEI-dynamicRenderingUnusedAttachments-08912
  If the dynamicRenderingUnusedAttachments feature is not enabled, and the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingInfo::colorAttachmentCount greater than 0, then each element of the VkRenderingInfo::pColorAttachments array with an imageView equal to VK_NULL_HANDLE **must** have the corresponding element of VkPipelineRenderingCreateInfo::pColorAttachmentFormats used to create the currently bound pipeline equal to VK_FORMAT_UNDEFINED

- VUID-vkCmdDrawClusterHUAWEI-dynamicRenderingUnusedAttachments-08911
  If the dynamicRenderingUnusedAttachments feature is enabled, and the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingInfo::colorAttachmentCount greater than 0, then each element of the VkRenderingInfo::pColorAttachments array with an imageView not equal to VK_NULL_HANDLE **must** have been created with a VkFormat equal to the corresponding element of VkPipelineRenderingCreateInfo::pColorAttachmentFormats used to create the currently bound graphics pipeline, or the corresponding element of VkPipelineRenderingCreateInfo::pColorAttachmentFormats, if it exists, **must** be VK_FORMAT_UNDEFINED

- VUID-vkCmdDrawClusterHUAWEI-colorAttachmentCount-09362
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, with a VkRenderingInfo::colorAttachmentCount equal to 1, there is no shader object bound to any graphics stage, and a color attachment with a resolve mode of VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID, each element of the VkRenderingInfo::pColorAttachments array with a resolveImageView not equal to VK_NULL_HANDLE **must** have been created with an image created with a VkExternalFormatANDROID::externalFormat value equal to the VkExternalFormatANDROID::externalFormat value used to create the currently bound graphics pipeline

- VUID-vkCmdDrawClusterHUAWEI-None-09363
  If there is no shader object bound to any graphics stage, the current render pass instance was begun with vkCmdBeginRenderingKHR and a VkRenderingInfo::colorAttachmentCount equal to 1, and a color attachment with a resolve mode of VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID, each element of the VkRenderingInfo::pColorAttachments array with a imageView not equal to VK_NULL_HANDLE **must** have been created with an image created with a VkExternalFormatANDROID::externalFormat value equal to the VkExternalFormatANDROID::externalFormat value used to create the currently bound graphics pipeline
- VUID-vkCmdDrawClusterHUAWEI-None-09364
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is no shader object bound to any graphics stage, and the currently bound graphics pipeline was created with a non-zero VkExternalFormatANDROID::externalFormat value and with the VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT dynamic state enabled, then vkCmdSetColorBlendEnableEXT **must** have set the blend enable to VK_FALSE prior to this drawing command
- VUID-vkCmdDrawClusterHUAWEI-None-09365
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is no shader object bound to any graphics stage, and the currently bound graphics pipeline was created with a non-zero VkExternalFormatANDROID::externalFormat value and with the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT dynamic state enabled, then vkCmdSetRasterizationSamplesEXT **must** have set rasterizationSamples to VK_SAMPLE_COUNT_1_BIT prior to this drawing command
- VUID-vkCmdDrawClusterHUAWEI-None-09366
  If there is a shader object bound to any graphics stage, and the current render pass includes a color attachment that uses the VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID resolve mode, then vkCmdSetColorBlendEnableEXT **must** have set blend enable to VK_FALSE prior to this drawing command
- VUID-vkCmdDrawClusterHUAWEI-rasterizationSamples-09367
  If there is a shader object bound to any graphics stage, and the current render pass includes a color attachment that uses the VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID resolve mode, then vkCmdSetRasterizationSamplesEXT **must** have set rasterizationSamples to VK_SAMPLE_COUNT_1_BIT prior to this drawing command
- VUID-vkCmdDrawClusterHUAWEI-None-09368
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is no shader object bound to any graphics stage, and the currently bound graphics pipeline was created with a non-zero VkExternalFormatANDROID::externalFormat value and with the VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR dynamic state enabled, then vkCmdSetFragmentShadingRateKHR **must** have set pFragmentSize->width to 1 prior to this drawing command
- VUID-vkCmdDrawClusterHUAWEI-None-09369
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is

no shader object bound to any graphics stage, and the currently bound graphics pipeline was created with a non-zero VkExternalFormatANDROID::externalFormat value and with the VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR dynamic state enabled, then vkCmdSetFragmentShadingRateKHR **must** have set pFragmentSize->height to 1 prior to this drawing command

- VUID-vkCmdDrawClusterHUAWEI-pFragmentSize-09370
  If there is a shader object bound to any graphics stage, and the current render pass includes a color attachment that uses the VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID resolve mode, then vkCmdSetFragmentShadingRateKHR **must** have set pFragmentSize->width to 1 prior to this drawing command
- VUID-vkCmdDrawClusterHUAWEI-pFragmentSize-09371
  If there is a shader object bound to any graphics stage, and the current render pass includes a color attachment that uses the VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID resolve mode, then vkCmdSetFragmentShadingRateKHR **must** have set pFragmentSize->height to 1 prior to this drawing command
- VUID-vkCmdDrawClusterHUAWEI-None-07749
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_WRITE_ENABLE_EXT dynamic state enabled then vkCmdSetColorWriteEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterHUAWEI-None-08646
  If the colorWriteEnable feature is enabled on the device, and a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetColorWriteEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterHUAWEI-attachmentCount-07750
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_WRITE_ENABLE_EXT dynamic state enabled then the attachmentCount parameter of vkCmdSetColorWriteEnableEXT **must** be greater than or equal to the VkPipelineColorBlendStateCreateInfo::attachmentCount of the currently bound graphics pipeline
- VUID-vkCmdDrawClusterHUAWEI-None-08647
  If the colorWriteEnable feature is enabled on the device, and a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then the attachmentCount parameter of most recent call to vkCmdSetColorWriteEnableEXT in the current command buffer **must** be greater than or equal to the number of color attachments in the current render pass instance
- VUID-vkCmdDrawClusterHUAWEI-None-07751
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_DISCARD_RECTANGLE_EXT dynamic state enabled then

vkCmdSetDiscardRectangleEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command for each discard rectangle in VkPipelineDiscardRectangleStateCreateInfoEXT::discardRectangleCount

- VUID-vkCmdDrawClusterHUAWEI-None-07880
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_DISCARD_RECTANGLE_ENABLE_EXT dynamic state enabled then vkCmdSetDiscardRectangleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterHUAWEI-rasterizerDiscardEnable-09236
  If the VK_EXT_discard_rectangles extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetDiscardRectangleEnableEXT in the current command buffer set discardRectangleEnable to VK_TRUE, then vkCmdSetDiscardRectangleEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterHUAWEI-None-08648
  If the VK_EXT_discard_rectangles extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetDiscardRectangleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterHUAWEI-None-07881
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_DISCARD_RECTANGLE_MODE_EXT dynamic state enabled then vkCmdSetDiscardRectangleModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterHUAWEI-None-08649
  If the VK_EXT_discard_rectangles extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetDiscardRectangleEnableEXT in the current command buffer set discardRectangleEnable to VK_TRUE, then vkCmdSetDiscardRectangleModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterHUAWEI-dynamicRenderingUnusedAttachments-08913
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is not enabled, and VkRenderingInfo::pDepthAttachment->imageView was VK_NULL_HANDLE, the value of VkPipelineRenderingCreateInfo::depthAttachmentFormat used to create the currently bound graphics pipeline **must** be equal to VK_FORMAT_UNDEFINED
- VUID-vkCmdDrawClusterHUAWEI-dynamicRenderingUnusedAttachments-08914
  If current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is not enabled, and VkRenderingInfo::pDepthAttachment->imageView was not VK_NULL_HANDLE, the value of

VkPipelineRenderingCreateInfo::depthAttachmentFormat used to create the currently bound graphics pipeline **must** be equal to the VkFormat used to create VkRenderingInfo::pDepthAttachment->imageView

- VUID-vkCmdDrawClusterHUAWEI-dynamicRenderingUnusedAttachments-08915
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is enabled, VkRenderingInfo::pDepthAttachment->imageView was not VK_NULL_HANDLE, and the value of VkPipelineRenderingCreateInfo::depthAttachmentFormat used to create the currently bound graphics pipeline was not equal to the VkFormat used to create VkRenderingInfo::pDepthAttachment->imageView, the value of the format **must** be VK_FORMAT_UNDEFINED
- VUID-vkCmdDrawClusterHUAWEI-dynamicRenderingUnusedAttachments-08916
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is not enabled, and VkRenderingInfo::pStencilAttachment->imageView was VK_NULL_HANDLE, the value of VkPipelineRenderingCreateInfo::stencilAttachmentFormat used to create the currently bound graphics pipeline **must** be equal to VK_FORMAT_UNDEFINED
- VUID-vkCmdDrawClusterHUAWEI-dynamicRenderingUnusedAttachments-08917
  If current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is not enabled, and VkRenderingInfo::pStencilAttachment->imageView was not VK_NULL_HANDLE, the value of VkPipelineRenderingCreateInfo::stencilAttachmentFormat used to create the currently bound graphics pipeline **must** be equal to the VkFormat used to create VkRenderingInfo::pStencilAttachment->imageView
- VUID-vkCmdDrawClusterHUAWEI-dynamicRenderingUnusedAttachments-08918
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is enabled, VkRenderingInfo::pStencilAttachment->imageView was not VK_NULL_HANDLE, and the value of VkPipelineRenderingCreateInfo::stencilAttachmentFormat used to create the currently bound graphics pipeline was not equal to the VkFormat used to create VkRenderingInfo::pStencilAttachment->imageView, the value of the format **must** be VK_FORMAT_UNDEFINED
- VUID-vkCmdDrawClusterHUAWEI-imageView-06183
  If the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingFragmentShadingRateAttachmentInfoKHR::imageView was not VK_NULL_HANDLE, the currently bound graphics pipeline **must** have been created with VK_PIPELINE_CREATE_RENDERING_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR
- VUID-vkCmdDrawClusterHUAWEI-imageView-06184
  If the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingFragmentDensityMapAttachmentInfoEXT::imageView was not VK_NULL_HANDLE, the currently bound graphics pipeline **must** have been created with VK_PIPELINE_CREATE_RENDERING_FRAGMENT_DENSITY_MAP_ATTACHMENT_BIT_EXT
- VUID-vkCmdDrawClusterHUAWEI-colorAttachmentCount-06185
  If the currently bound pipeline was created with a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and the current render pass instance was begun with vkCmdBeginRenderingKHR with a VkRenderingInfo::colorAttachmentCount parameter greater than 0, then each element of the VkRenderingInfo::pColorAttachments

array with a imageView not equal to VK_NULL_HANDLE **must** have been created with a sample count equal to the corresponding element of the pColorAttachmentSamples member of VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV used to create the currently bound graphics pipeline

- VUID-vkCmdDrawClusterHUAWEI-pDepthAttachment-06186
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the currently bound pipeline was created with a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and VkRenderingInfo::pDepthAttachment->imageView was not VK_NULL_HANDLE, the value of the depthStencilAttachmentSamples member of VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV used to create the currently bound graphics pipeline **must** be equal to the sample count used to create VkRenderingInfo::pDepthAttachment->imageView
- VUID-vkCmdDrawClusterHUAWEI-pStencilAttachment-06187
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the currently bound pipeline was created with a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and VkRenderingInfo::pStencilAttachment->imageView was not VK_NULL_HANDLE, the value of the depthStencilAttachmentSamples member of VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV used to create the currently bound graphics pipeline **must** be equal to the sample count used to create VkRenderingInfo::pStencilAttachment->imageView
- VUID-vkCmdDrawClusterHUAWEI-multisampledRenderToSingleSampled-07285
  If the currently bound pipeline was created without a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and the multisampledRenderToSingleSampled feature is not enabled, and the current render pass instance was begun with vkCmdBeginRenderingKHR with a VkRenderingInfo::colorAttachmentCount parameter greater than 0, then each element of the VkRenderingInfo::pColorAttachments array with a imageView not equal to VK_NULL_HANDLE **must** have been created with a sample count equal to the value of rasterizationSamples for the currently bound graphics pipeline
- VUID-vkCmdDrawClusterHUAWEI-multisampledRenderToSingleSampled-07286
  If the currently bound pipeline was created without a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and the multisampledRenderToSingleSampled feature is not enabled, and VkRenderingInfo::pDepthAttachment->imageView was not VK_NULL_HANDLE, the value of rasterizationSamples for the currently bound graphics pipeline **must** be equal to the sample count used to create VkRenderingInfo::pDepthAttachment->imageView
- VUID-vkCmdDrawClusterHUAWEI-multisampledRenderToSingleSampled-07287
  If the currently bound pipeline was created without a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and the multisampledRenderToSingleSampled feature is not enabled, and VkRenderingInfo::pStencilAttachment->imageView was not VK_NULL_HANDLE, the value of rasterizationSamples for the currently bound graphics pipeline **must** be equal to the sample count used to create VkRenderingInfo::pStencilAttachment->imageView
- VUID-vkCmdDrawClusterHUAWEI-pNext-07935
  If this command has been called inside a render pass instance started with

vkCmdBeginRenderingKHR, and the pNext chain of VkRenderingInfo includes a VkMultisampledRenderToSingleSampledInfoEXT structure with multisampledRenderToSingleSampledEnable equal to VK_TRUE, then the value of rasterizationSamples for the currently bound graphics pipeline **must** be equal to VkMultisampledRenderToSingleSampledInfoEXT::rasterizationSamples

- VUID-vkCmdDrawClusterHUAWEI-renderPass-06198
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the currently bound pipeline **must** have been created with a VkGraphicsPipelineCreateInfo::renderPass equal to VK_NULL_HANDLE
- VUID-vkCmdDrawClusterHUAWEI-pColorAttachments-08963
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is a graphics pipeline bound with a fragment shader that statically writes to a color attachment, the color write mask is not zero, color writes are enabled, and the corresponding element of the VkRenderingInfo::pColorAttachments->imageView was not VK_NULL_HANDLE, then the corresponding element of VkPipelineRenderingCreateInfo::pColorAttachmentFormats used to create the pipeline **must** not be VK_FORMAT_UNDEFINED
- VUID-vkCmdDrawClusterHUAWEI-pDepthAttachment-08964
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is a graphics pipeline bound, depth test is enabled, depth write is enabled, and the VkRenderingInfo::pDepthAttachment->imageView was not VK_NULL_HANDLE, then the VkPipelineRenderingCreateInfo::depthAttachmentFormat used to create the pipeline **must** not be VK_FORMAT_UNDEFINED
- VUID-vkCmdDrawClusterHUAWEI-pStencilAttachment-08965
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is a graphics pipeline bound, stencil test is enabled and the VkRenderingInfo::pStencilAttachment->imageView was not VK_NULL_HANDLE, then the VkPipelineRenderingCreateInfo::stencilAttachmentFormat used to create the pipeline **must** not be VK_FORMAT_UNDEFINED
- VUID-vkCmdDrawClusterHUAWEI-primitivesGeneratedQueryWithRasterizerDiscard-06708
  If the primitivesGeneratedQueryWithRasterizerDiscard feature is not enabled and the VK_QUERY_TYPE_PRIMITIVES_GENERATED_EXT query is active, rasterization discard **must** not be enabled
- VUID-vkCmdDrawClusterHUAWEI-primitivesGeneratedQueryWithNonZeroStreams-06709
  If the primitivesGeneratedQueryWithNonZeroStreams feature is not enabled and the VK_QUERY_TYPE_PRIMITIVES_GENERATED_EXT query is active, the bound graphics pipeline **must** not have been created with a non-zero value in VkPipelineRasterizationStateStreamCreateInfoEXT::rasterizationStream
- VUID-vkCmdDrawClusterHUAWEI-None-07619
  If a shader object is bound to the VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_TESSELLATION_DOMAIN_ORIGIN_EXT dynamic state enabled, then vkCmdSetTessellationDomainOriginEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterHUAWEI-None-07620

If the depthClamp feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_CLAMP_ENABLE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetDepthClampEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawClusterHUAWEI-None-07621
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_POLYGON_MODE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetPolygonModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterHUAWEI-None-07622
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetRasterizationSamplesEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterHUAWEI-None-07623
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_SAMPLE_MASK_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetSampleMaskEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterHUAWEI-alphaToCoverageEnable-08919
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_ALPHA_TO_COVERAGE_ENABLE_EXT dynamic state enabled, and alphaToCoverageEnable was VK_TRUE in the last call to vkCmdSetAlphaToCoverageEnableEXT, then the Fragment Output Interface **must** contain a variable for the alpha Component word in Location 0 at Index 0
- VUID-vkCmdDrawClusterHUAWEI-alphaToCoverageEnable-08920
  If a shader object is bound to any graphics stage, and the most recent call to vkCmdSetAlphaToCoverageEnableEXT in the current command buffer set alphaToCoverageEnable to VK_TRUE, then the Fragment Output Interface **must** contain a variable for the alpha Component word in Location 0 at Index 0
- VUID-vkCmdDrawClusterHUAWEI-None-07624
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_ALPHA_TO_COVERAGE_ENABLE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetAlphaToCoverageEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterHUAWEI-None-07625
  If the alphaToOne feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_ALPHA_TO_ONE_ENABLE_EXT dynamic state enabled, and the current value

of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetAlphaToOneEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawClusterHUAWEI-None-07626
  If the logicOp feature is enabled, a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_LOGIC_OP_ENABLE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetLogicOpEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawClusterHUAWEI-None-07627
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT dynamic state enabled then vkCmdSetColorBlendEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawClusterHUAWEI-None-08657
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and both the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE and there are color attachments bound, then vkCmdSetColorBlendEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawClusterHUAWEI-None-07628
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_EQUATION_EXT dynamic state enabled then vkCmdSetColorBlendEquationEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawClusterHUAWEI-None-08658
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetColorBlendEnableEXT for any attachment set that attachment's value in pColorBlendEnables to VK_TRUE, then vkCmdSetColorBlendEquationEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawClusterHUAWEI-None-07629
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_WRITE_MASK_EXT dynamic state enabled then vkCmdSetColorWriteMaskEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawClusterHUAWEI-None-08659
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and both the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE and there are color attachments bound, then vkCmdSetColorWriteMaskEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawClusterHUAWEI-None-07630
  If the geometryStreams feature is enabled, and a shader object is bound to the VK_SHADER_STAGE_GEOMETRY_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_RASTERIZATION_STREAM_EXT dynamic state enabled, then vkCmdSetRasterizationStreamEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterHUAWEI-None-07631
  If the VK_EXT_conservative_rasterization extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_CONSERVATIVE_RASTERIZATION_MODE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetConservativeRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterHUAWEI-None-07632
  If the VK_EXT_conservative_rasterization extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_EXTRA_PRIMITIVE_OVERESTIMATION_SIZE_EXT dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of conservativeRasterizationMode is VK_CONSERVATIVE_RASTERIZATION_MODE_OVERESTIMATE_EXT, then vkCmdSetExtraPrimitiveOverestimationSizeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterHUAWEI-None-07633
  If the depthClipEnable feature is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_CLIP_ENABLE_EXT dynamic state, then vkCmdSetDepthClipEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterHUAWEI-None-07634
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT dynamic state enabled then vkCmdSetSampleLocationsEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterHUAWEI-None-08664
  If the VK_EXT_sample_locations extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetSampleLocationsEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterHUAWEI-None-07635
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_ADVANCED_EXT dynamic state enabled then vkCmdSetColorBlendAdvancedEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterHUAWEI-rasterizerDiscardEnable-09416
  If the VK_EXT_blend_operation_advanced extension is enabled, and a shader object is bound

to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then at least one of vkCmdSetColorBlendEquationEXT and vkCmdSetColorBlendAdvancedEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawClusterHUAWEI-None-07636
  If the VK_EXT_provoking_vertex extension is enabled, a shader object is bound to the VK_SHADER_STAGE_VERTEX_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_PROVOKING_VERTEX_MODE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetProvokingVertexModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterHUAWEI-None-07637
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT dynamic state enabled then vkCmdSetLineRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterHUAWEI-None-08666
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetPolygonModeEXT in the current command buffer set polygonMode to VK_POLYGON_MODE_LINE, then vkCmdSetLineRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterHUAWEI-None-08667
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object is bound to the VK_SHADER_STAGE_VERTEX_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetPrimitiveTopologyEXT in the current command buffer set primitiveTopology to any line topology, then vkCmdSetLineRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterHUAWEI-None-08668
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object that outputs line primitives is bound to the VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT or VK_SHADER_STAGE_GEOMETRY_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetLineRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterHUAWEI-None-07638

If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT dynamic state enabled then vkCmdSetLineStippleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawClusterHUAWEI-None-08669
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetPolygonModeEXT in the current command buffer set polygonMode to VK_POLYGON_MODE_LINE, then vkCmdSetLineStippleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterHUAWEI-None-08670
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object is bound to the VK_SHADER_STAGE_VERTEX_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetPrimitiveTopologyEXT in the current command buffer set primitiveTopology to any line topology, then vkCmdSetLineStippleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterHUAWEI-None-08671
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object that outputs line primitives is bound to the VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT or VK_SHADER_STAGE_GEOMETRY_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetLineStippleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterHUAWEI-None-07849
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_KHR dynamic state enabled then vkCmdSetLineStippleKHR **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterHUAWEI-None-08672
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetLineStippleEnableEXT in the current command buffer set stippledLineEnable to VK_TRUE, then vkCmdSetLineStippleEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterHUAWEI-None-07639
  If the depthClipControl feature is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the

VK_DYNAMIC_STATE_DEPTH_CLIP_NEGATIVE_ONE_TO_ONE_EXT dynamic state enabled, then vkCmdSetDepthClipNegativeOneToOneEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawClusterHUAWEI-None-07640
  If the VK_NV_clip_space_w_scaling extension is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_VIEWPORT_W_SCALING_ENABLE_NV dynamic state enabled, then vkCmdSetViewportWScalingEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterHUAWEI-None-07641
  If the VK_NV_viewport_swizzle extension is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_VIEWPORT_SWIZZLE_NV dynamic state enabled, then vkCmdSetViewportSwizzleNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterHUAWEI-None-07642
  If the VK_NV_fragment_coverage_to_color extension is enabled, a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_TO_COLOR_ENABLE_NV dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetCoverageToColorEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterHUAWEI-None-07643
  If the VK_NV_fragment_coverage_to_color extension is enabled, a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_TO_COLOR_LOCATION_NV dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of coverageToColorEnable is VK_TRUE, then vkCmdSetCoverageToColorLocationNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterHUAWEI-None-07644
  If the VK_NV_framebuffer_mixed_samples extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_MODULATION_MODE_NV dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetCoverageModulationModeNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterHUAWEI-None-07645
  If the VK_NV_framebuffer_mixed_samples extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_MODULATION_TABLE_ENABLE_NV dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of coverageModulationMode is any value other than VK_COVERAGE_MODULATION_MODE_NONE_NV, then vkCmdSetCoverageModulationTableEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawClusterHUAWEI-None-07646
  If the VK_NV_framebuffer_mixed_samples extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_MODULATION_TABLE_NV dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of coverageModulationTableEnable is VK_TRUE, then vkCmdSetCoverageModulationTableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterHUAWEI-None-07647
  If the shadingRateImage feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_SHADING_RATE_IMAGE_ENABLE_NV dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetShadingRateImageEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterHUAWEI-pipelineFragmentShadingRate-09238
  If the pipelineFragmentShadingRate feature is enabled, a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetFragmentShadingRateKHR **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterHUAWEI-None-07648
  If the representativeFragmentTest feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_REPRESENTATIVE_FRAGMENT_TEST_ENABLE_NV dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetRepresentativeFragmentTestEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterHUAWEI-None-07649
  If the coverageReductionMode feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_REDUCTION_MODE_NV dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetCoverageReductionModeNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterHUAWEI-pColorBlendEnables-07470
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT state enabled and the last call to vkCmdSetColorBlendEnableEXT set pColorBlendEnables for any attachment to VK_TRUE, then for those attachments in the subpass the corresponding image view's format features **must** contain VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT
- VUID-vkCmdDrawClusterHUAWEI-rasterizationSamples-07471
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, and the current subpass does

not use any color and/or depth/stencil attachments, then the rasterizationSamples in the last call to vkCmdSetRasterizationSamplesEXT **must** follow the rules for a zero-attachment subpass

- VUID-vkCmdDrawClusterHUAWEI-samples-07472
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_MASK_EXT state enabled and the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state disabled, then the samples parameter in the last call to vkCmdSetSampleMaskEXT **must** be greater or equal to the VkPipelineMultisampleStateCreateInfo::rasterizationSamples parameter used to create the bound graphics pipeline

- VUID-vkCmdDrawClusterHUAWEI-samples-07473
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_MASK_EXT state and VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT states enabled, then the samples parameter in the last call to vkCmdSetSampleMaskEXT **must** be greater or equal to the rasterizationSamples parameter in the last call to vkCmdSetRasterizationSamplesEXT

- VUID-vkCmdDrawClusterHUAWEI-rasterizationSamples-07474
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, and neither the VK_AMD_mixed_attachment_samples nor the VK_NV_framebuffer_mixed_samples extensions are enabled, then the rasterizationSamples in the last call to vkCmdSetRasterizationSamplesEXT **must** be the same as the current subpass color and/or depth/stencil attachments

- VUID-vkCmdDrawClusterHUAWEI-None-09211
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, or a shader object is bound to any graphics stage, and the current render pass instance includes a VkMultisampledRenderToSingleSampledInfoEXT structure with multisampledRenderToSingleSampledEnable equal to VK_TRUE, then the rasterizationSamples in the last call to vkCmdSetRasterizationSamplesEXT **must** be the same as the rasterizationSamples member of that structure

- VUID-vkCmdDrawClusterHUAWEI-firstAttachment-07476
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT dynamic state enabled then vkCmdSetColorBlendEnableEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the firstAttachment and attachmentCount parameters of vkCmdSetColorBlendEnableEXT calls **must** specify an enable for all active color attachments in the current subpass

- VUID-vkCmdDrawClusterHUAWEI-rasterizerDiscardEnable-09417
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetColorBlendEnableEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the firstAttachment and attachmentCount parameters of vkCmdSetColorBlendEnableEXT calls **must** specify an enable for all active color attachments

in the current subpass

- VUID-vkCmdDrawClusterHUAWEI-firstAttachment-07477
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_COLOR_BLEND_EQUATION_EXT` dynamic state enabled then vkCmdSetColorBlendEquationEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the `firstAttachment` and `attachmentCount` parameters of `vkCmdSetColorBlendEquationEXT` calls **must** specify the blend equations for all active color attachments in the current subpass where blending is enabled

- VUID-vkCmdDrawClusterHUAWEI-rasterizerDiscardEnable-09418
  If a shader object is bound to the `VK_SHADER_STAGE_FRAGMENT_BIT` stage, and both the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE` and there are color attachments bound, then vkCmdSetColorBlendEquationEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the `firstAttachment` and `attachmentCount` parameters of `vkCmdSetColorBlendEquationEXT` calls **must** specify the blend equations for all active color attachments in the current subpass where blending is enabled

- VUID-vkCmdDrawClusterHUAWEI-firstAttachment-07478
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_COLOR_WRITE_MASK_EXT` dynamic state enabled then vkCmdSetColorWriteMaskEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the `firstAttachment` and `attachmentCount` parameters of `vkCmdSetColorWriteMaskEXT` calls **must** specify the color write mask for all active color attachments in the current subpass

- VUID-vkCmdDrawClusterHUAWEI-rasterizerDiscardEnable-09419
  If a shader object is bound to the `VK_SHADER_STAGE_FRAGMENT_BIT` stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE`, then vkCmdSetColorWriteMaskEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the `firstAttachment` and `attachmentCount` parameters of `vkCmdSetColorWriteMaskEXT` calls **must** specify the color write mask for all active color attachments in the current subpass

- VUID-vkCmdDrawClusterHUAWEI-firstAttachment-07479
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_COLOR_BLEND_ADVANCED_EXT` dynamic state enabled then vkCmdSetColorBlendAdvancedEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the `firstAttachment` and `attachmentCount` parameters of `vkCmdSetColorBlendAdvancedEXT` calls **must** specify the advanced blend equations for all active color attachments in the current subpass where blending is enabled

- VUID-vkCmdDrawClusterHUAWEI-advancedBlendMaxColorAttachments-07480
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_COLOR_BLEND_ADVANCED_EXT` and `VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT` dynamic states enabled and the last calls to vkCmdSetColorBlendEnableEXT and

vkCmdSetColorBlendAdvancedEXT have enabled advanced blending, then the number of active color attachments in the current subpass **must** not exceed advancedBlendMaxColorAttachments

- VUID-vkCmdDrawClusterHUAWEI-primitivesGeneratedQueryWithNonZeroStreams-07481
  If the primitivesGeneratedQueryWithNonZeroStreams feature is not enabled and the VK_QUERY_TYPE_PRIMITIVES_GENERATED_EXT query is active, and the bound graphics pipeline was created with VK_DYNAMIC_STATE_RASTERIZATION_STREAM_EXT state enabled, the last call to vkCmdSetRasterizationStreamEXT **must** have set the rasterizationStream to zero

- VUID-vkCmdDrawClusterHUAWEI-sampleLocationsPerPixel-07482
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state enabled and the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state disabled, then the sampleLocationsPerPixel member of pSampleLocationsInfo in the last call to vkCmdSetSampleLocationsEXT **must** equal the rasterizationSamples member of the VkPipelineMultisampleStateCreateInfo structure the bound graphics pipeline has been created with

- VUID-vkCmdDrawClusterHUAWEI-sampleLocationsPerPixel-07483
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state enabled and the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, then the sampleLocationsPerPixel member of pSampleLocationsInfo in the last call to vkCmdSetSampleLocationsEXT **must** equal the rasterizationSamples parameter of the last call to vkCmdSetRasterizationSamplesEXT

- VUID-vkCmdDrawClusterHUAWEI-sampleLocationsEnable-07484
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, or the bound graphics pipeline was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, and sampleLocationsEnable was VK_TRUE in the last call to vkCmdSetSampleLocationsEnableEXT, and the current subpass has a depth/stencil attachment, then that attachment **must** have been created with the VK_IMAGE_CREATE_SAMPLE_LOCATIONS_COMPATIBLE_DEPTH_BIT_EXT bit set

- VUID-vkCmdDrawClusterHUAWEI-sampleLocationsEnable-07485
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state enabled and the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, and if sampleLocationsEnable was VK_TRUE in the last call to vkCmdSetSampleLocationsEnableEXT, then the sampleLocationsInfo.sampleLocationGridSize.width in the last call to vkCmdSetSampleLocationsEXT **must** evenly divide VkMultisamplePropertiesEXT::sampleLocationGridSize.width as returned by vkGetPhysicalDeviceMultisamplePropertiesEXT with a samples parameter equaling rasterizationSamples

- VUID-vkCmdDrawClusterHUAWEI-sampleLocationsEnable-07486
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state enabled and the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, and if

sampleLocationsEnable was VK_TRUE in the last call to vkCmdSetSampleLocationsEnableEXT, then the sampleLocationsInfo.sampleLocationGridSize.height in the last call to vkCmdSetSampleLocationsEXT **must** evenly divide VkMultisamplePropertiesEXT::sampleLocationGridSize.height as returned by vkGetPhysicalDeviceMultisamplePropertiesEXT with a samples parameter equaling rasterizationSamples

- VUID-vkCmdDrawClusterHUAWEI-sampleLocationsEnable-07487
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, and if sampleLocationsEnable was VK_TRUE in the last call to vkCmdSetSampleLocationsEnableEXT, the fragment shader code **must** not statically use the extended instruction InterpolateAtSample

- VUID-vkCmdDrawClusterHUAWEI-sampleLocationsEnable-07936
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state disabled and the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, the sampleLocationsEnable member of a VkPipelineSampleLocationsStateCreateInfoEXT::sampleLocationsEnable in the bound graphics pipeline is VK_TRUE or VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, then, sampleLocationsInfo.sampleLocationGridSize.width **must** evenly divide VkMultisamplePropertiesEXT::sampleLocationGridSize.width as returned by vkGetPhysicalDeviceMultisamplePropertiesEXT with a samples parameter equaling the value of rasterizationSamples in the last call to vkCmdSetRasterizationSamplesEXT

- VUID-vkCmdDrawClusterHUAWEI-sampleLocationsEnable-07937
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state disabled and the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, the sampleLocationsEnable member of a VkPipelineSampleLocationsStateCreateInfoEXT::sampleLocationsEnable in the bound graphics pipeline is VK_TRUE or VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, then, sampleLocationsInfo.sampleLocationGridSize.height **must** evenly divide VkMultisamplePropertiesEXT::sampleLocationGridSize.height as returned by vkGetPhysicalDeviceMultisamplePropertiesEXT with a samples parameter equaling the value of rasterizationSamples in the last call to vkCmdSetRasterizationSamplesEXT

- VUID-vkCmdDrawClusterHUAWEI-sampleLocationsEnable-07938
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state disabled and the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, the sampleLocationsEnable member of a VkPipelineSampleLocationsStateCreateInfoEXT::sampleLocationsEnable in the bound graphics pipeline is VK_TRUE or VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, then, sampleLocationsInfo.sampleLocationsPerPixel **must** equal rasterizationSamples in the last call to vkCmdSetRasterizationSamplesEXT

- VUID-vkCmdDrawClusterHUAWEI-coverageModulationTableEnable-07488
  If a shader object is bound to any graphics stage or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COVERAGE_MODULATION_TABLE_ENABLE_NV state enabled, and the last call to vkCmdSetCoverageModulationTableEnableNV set coverageModulationTableEnable to VK_TRUE, then the coverageModulationTableCount

parameter in the last call to vkCmdSetCoverageModulationTableNV **must** equal the current rasterizationSamples divided by the number of color samples in the current subpass

- VUID-vkCmdDrawClusterHUAWEI-rasterizationSamples-07489
  If the VK_NV_framebuffer_mixed_samples extension is enabled, and if current subpass has a depth/stencil attachment and depth test, stencil test, or depth bounds test are enabled in the currently bound pipeline state, then the current rasterizationSamples **must** be the same as the sample count of the depth/stencil attachment

- VUID-vkCmdDrawClusterHUAWEI-coverageToColorEnable-07490
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COVERAGE_TO_COLOR_ENABLE_NV state enabled and the last call to vkCmdSetCoverageToColorEnableNV set the coverageToColorEnable to VK_TRUE, then the current subpass **must** have a color attachment at the location selected by the last call to vkCmdSetCoverageToColorLocationNV coverageToColorLocation, with a VkFormat of VK_FORMAT_R8_UINT, VK_FORMAT_R8_SINT, VK_FORMAT_R16_UINT, VK_FORMAT_R16_SINT, VK_FORMAT_R32_UINT, or VK_FORMAT_R32_SINT

- VUID-vkCmdDrawClusterHUAWEI-rasterizerDiscardEnable-09420
  If the VK_NV_fragment_coverage_to_color extension is enabled, and a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the last call to vkCmdSetCoverageToColorEnableNV set the coverageToColorEnable to VK_TRUE, then the current subpass **must** have a color attachment at the location selected by the last call to vkCmdSetCoverageToColorLocationNV coverageToColorLocation, with a VkFormat of VK_FORMAT_R8_UINT, VK_FORMAT_R8_SINT, VK_FORMAT_R16_UINT, VK_FORMAT_R16_SINT, VK_FORMAT_R32_UINT, or VK_FORMAT_R32_SINT

- VUID-vkCmdDrawClusterHUAWEI-coverageReductionMode-07491
  If this VK_NV_coverage_reduction_mode extension is enabled, the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COVERAGE_TO_COLOR_ENABLE_NV and VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT states enabled, the current coverage reduction mode coverageReductionMode, then the current rasterizationSamples, and the sample counts for the color and depth/stencil attachments (if the subpass has them) **must** be a valid combination returned by vkGetPhysicalDeviceSupportedFramebufferMixedSamplesCombinationsNV

- VUID-vkCmdDrawClusterHUAWEI-viewportCount-07492
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, but not the VK_DYNAMIC_STATE_VIEWPORT_SWIZZLE_NV dynamic state enabled, then the bound graphics pipeline **must** have been created with VkPipelineViewportSwizzleStateCreateInfoNV::viewportCount greater or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT

- VUID-vkCmdDrawClusterHUAWEI-viewportCount-07493
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT and VK_DYNAMIC_STATE_VIEWPORT_SWIZZLE_NV dynamic states enabled then the viewportCount parameter in the last call to

vkCmdSetViewportSwizzleNV **must** be greater than or equal to the `viewportCount` parameter in the last call to vkCmdSetViewportWithCountEXT

- VUID-vkCmdDrawClusterHUAWEI-viewportCount-09421
  If the `VK_NV_viewport_swizzle` extension is enabled, and a shader object is bound to any graphics stage, then the `viewportCount` parameter in the last call to vkCmdSetViewportSwizzleNV **must** be greater than or equal to the `viewportCount` parameter in the last call to vkCmdSetViewportWithCountEXT

- VUID-vkCmdDrawClusterHUAWEI-rasterizationSamples-07494
  If the `VK_NV_framebuffer_mixed_samples` extension is enabled, and if the current subpass has any color attachments and `rasterizationSamples` of the last call to vkCmdSetRasterizationSamplesEXT is greater than the number of color samples, then the pipeline `sampleShadingEnable` **must** be `VK_FALSE`

- VUID-vkCmdDrawClusterHUAWEI-stippledLineEnable-07495
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT` or `VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT` dynamic states enabled, and if the current `stippledLineEnable` state is `VK_TRUE` and the current `lineRasterizationMode` state is `VK_LINE_RASTERIZATION_MODE_RECTANGULAR_KHR`, then the `stippledRectangularLines` feature **must** be enabled

- VUID-vkCmdDrawClusterHUAWEI-stippledLineEnable-07496
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT` or `VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT` dynamic states enabled, and if the current `stippledLineEnable` state is `VK_TRUE` and the current `lineRasterizationMode` state is `VK_LINE_RASTERIZATION_MODE_BRESENHAM_KHR`, then the `stippledBresenhamLines` feature **must** be enabled

- VUID-vkCmdDrawClusterHUAWEI-stippledLineEnable-07497
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT` or `VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT` dynamic states enabled, and if the current `stippledLineEnable` state is `VK_TRUE` and the current `lineRasterizationMode` state is `VK_LINE_RASTERIZATION_MODE_RECTANGULAR_SMOOTH_KHR`, then the `stippledSmoothLines` feature **must** be enabled

- VUID-vkCmdDrawClusterHUAWEI-stippledLineEnable-07498
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT` or `VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT` dynamic states enabled, and if the current `stippledLineEnable` state is `VK_TRUE` and the current `lineRasterizationMode` state is `VK_LINE_RASTERIZATION_MODE_DEFAULT_KHR`, then the `stippledRectangularLines` feature **must** be enabled and VkPhysicalDeviceLimits::`strictLines` **must** be `VK_TRUE`

- VUID-vkCmdDrawClusterHUAWEI-conservativePointAndLineRasterization-07499
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_CONSERVATIVE_RASTERIZATION_MODE_EXT` dynamic state enabled, `conservativePointAndLineRasterization` is not supported, and the effective primitive topology output by the last pre-rasterization shader stage is a line or point, then the

conservativeRasterizationMode set by the last call to vkCmdSetConservativeRasterizationModeEXT **must** be VK_CONSERVATIVE_RASTERIZATION_MODE_DISABLED_EXT

- VUID-vkCmdDrawClusterHUAWEI-stage-07073
  If the currently bound pipeline was created with the VkPipelineShaderStageCreateInfo::stage member of an element of VkGraphicsPipelineCreateInfo::pStages set to VK_SHADER_STAGE_VERTEX_BIT, VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT, VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT or VK_SHADER_STAGE_GEOMETRY_BIT, then Mesh Shader Queries **must** not be active
- VUID-vkCmdDrawClusterHUAWEI-None-08877
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_ATTACHMENT_FEEDBACK_LOOP_ENABLE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetAttachmentFeedbackLoopEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterHUAWEI-None-07850
  If dynamic state was inherited from VkCommandBufferInheritanceViewportScissorInfoNV, it **must** be set in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterHUAWEI-None-08684
  If there is no bound graphics pipeline, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_VERTEX_BIT
- VUID-vkCmdDrawClusterHUAWEI-None-08685
  If there is no bound graphics pipeline, and the tessellationShader feature is enabled, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT
- VUID-vkCmdDrawClusterHUAWEI-None-08686
  If there is no bound graphics pipeline, and the tessellationShader feature is enabled, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT
- VUID-vkCmdDrawClusterHUAWEI-None-08687
  If there is no bound graphics pipeline, and the geometryShader feature is enabled, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_GEOMETRY_BIT
- VUID-vkCmdDrawClusterHUAWEI-None-08688
  If there is no bound graphics pipeline, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_FRAGMENT_BIT
- VUID-vkCmdDrawClusterHUAWEI-None-08689
  If there is no bound graphics pipeline, and the taskShader feature is enabled, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_TASK_BIT_EXT
- VUID-vkCmdDrawClusterHUAWEI-None-08690
  If there is no bound graphics pipeline, and the meshShader feature is enabled,

vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_MESH_BIT_EXT

- VUID-vkCmdDrawClusterHUAWEI-None-08693
  If there is no bound graphics pipeline, and at least one of the taskShader and meshShader features is enabled, one of the VK_SHADER_STAGE_VERTEX_BIT or VK_SHADER_STAGE_MESH_BIT_EXT stages **must** have a valid VkShaderEXT bound, and the other **must** have no VkShaderEXT bound
- VUID-vkCmdDrawClusterHUAWEI-None-08694
  If there is no bound graphics pipeline, and both the taskShader and meshShader features are enabled, and a valid VkShaderEXT is bound the to the VK_SHADER_STAGE_MESH_BIT_EXT stage, and that VkShaderEXT was created without the VK_SHADER_CREATE_NO_TASK_SHADER_BIT_EXT flag, a valid VkShaderEXT **must** be bound to the VK_SHADER_STAGE_TASK_BIT_EXT stage
- VUID-vkCmdDrawClusterHUAWEI-None-08695
  If there is no bound graphics pipeline, and both the taskShader and meshShader features are enabled, and a valid VkShaderEXT is bound the to the VK_SHADER_STAGE_MESH_BIT_EXT stage, and that VkShaderEXT was created with the VK_SHADER_CREATE_NO_TASK_SHADER_BIT_EXT flag, there **must** be no VkShaderEXT bound to the VK_SHADER_STAGE_TASK_BIT_EXT stage
- VUID-vkCmdDrawClusterHUAWEI-None-08696
  If there is no bound graphics pipeline, and a valid VkShaderEXT is bound to the VK_SHADER_STAGE_VERTEX_BIT stage, there **must** be no VkShaderEXT bound to either the VK_SHADER_STAGE_TASK_BIT_EXT stage or the VK_SHADER_STAGE_MESH_BIT_EXT stage
- VUID-vkCmdDrawClusterHUAWEI-None-08698
  If any graphics shader is bound which was created with the VK_SHADER_CREATE_LINK_STAGE_BIT_EXT flag, then all shaders created with the VK_SHADER_CREATE_LINK_STAGE_BIT_EXT flag in the same vkCreateShadersEXT call **must** also be bound
- VUID-vkCmdDrawClusterHUAWEI-None-08699
  If any graphics shader is bound which was created with the VK_SHADER_CREATE_LINK_STAGE_BIT_EXT flag, any stages in between stages whose shaders which did not create a shader with the VK_SHADER_CREATE_LINK_STAGE_BIT_EXT flag as part of the same vkCreateShadersEXT call **must** not have any VkShaderEXT bound
- VUID-vkCmdDrawClusterHUAWEI-None-08878
  All bound graphics shader objects **must** have been created with identical or identically defined push constant ranges
- VUID-vkCmdDrawClusterHUAWEI-None-08879
  All bound graphics shader objects **must** have been created with identical or identically defined arrays of descriptor set layouts
- VUID-vkCmdDrawClusterHUAWEI-colorAttachmentCount-09372
  If the current render pass instance was begun with vkCmdBeginRenderingKHR and a VkRenderingInfo::colorAttachmentCount equal to 1, a color attachment with a resolve mode of VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID, and a fragment shader is bound, it **must** not declare the DepthReplacing or StencilRefReplacingEXT execution modes
- VUID-vkCmdDrawClusterHUAWEI-pDynamicStates-08715

If the bound graphics pipeline state includes a fragment shader stage, was created with VK_DYNAMIC_STATE_DEPTH_WRITE_ENABLE set in VkPipelineDynamicStateCreateInfo::pDynamicStates, and the fragment shader declares the EarlyFragmentTests execution mode and uses OpDepthAttachmentReadEXT, the depthWriteEnable parameter in the last call to vkCmdSetDepthWriteEnableEXT **must** be VK_FALSE

- VUID-vkCmdDrawClusterHUAWEI-pDynamicStates-08716
  If the bound graphics pipeline state includes a fragment shader stage, was created with VK_DYNAMIC_STATE_STENCIL_WRITE_MASK set in VkPipelineDynamicStateCreateInfo::pDynamicStates, and the fragment shader declares the EarlyFragmentTests execution mode and uses OpStencilAttachmentReadEXT, the writeMask parameter in the last call to vkCmdSetStencilWriteMask **must** be 0
- VUID-vkCmdDrawClusterHUAWEI-None-09116
  If a shader object is bound to any graphics stage or the currently bound graphics pipeline was created with VK_DYNAMIC_STATE_COLOR_WRITE_MASK_EXT, and the format of any color attachment is VK_FORMAT_E5B9G9R9_UFLOAT_PACK32, the corresponding element of the pColorWriteMasks parameter of vkCmdSetColorWriteMaskEXT **must** either include all of VK_COLOR_COMPONENT_R_BIT, VK_COLOR_COMPONENT_G_BIT, and VK_COLOR_COMPONENT_B_BIT, or none of them
- VUID-vkCmdDrawClusterHUAWEI-maxFragmentDualSrcAttachments-09239
  If blending is enabled for any attachment where either the source or destination blend factors for that attachment use the secondary color input, the maximum value of Location for any output attachment statically used in the Fragment Execution Model executed by this command **must** be less than maxFragmentDualSrcAttachments
- VUID-vkCmdDrawClusterHUAWEI-None-09548
  If the current render pass was begun with vkCmdBeginRenderingKHR, and there is no shader object bound to any graphics stage, the value of each element of VkRenderingAttachmentLocationInfoKHR::pColorAttachmentLocations set by vkCmdSetRenderingAttachmentLocationsKHR **must** match the value set for the corresponding element in the currently bound pipeline
- VUID-vkCmdDrawClusterHUAWEI-None-09549
  If the current render pass was begun with vkCmdBeginRenderingKHR, and there is no shader object bound to any graphics stage, input attachment index mappings in the currently bound pipeline **must** match those set for the current render pass instance via VkRenderingInputAttachmentIndexInfoKHR
- VUID-vkCmdDrawClusterHUAWEI-None-09642
  If the current render pass was begun with vkCmdBeginRenderingKHR with the VK_RENDERING_ENABLE_LEGACY_DITHERING_BIT_EXT flag, the bound graphics pipeline **must** have been created with VK_PIPELINE_CREATE_2_ENABLE_LEGACY_DITHERING_BIT_EXT
- VUID-vkCmdDrawClusterHUAWEI-None-09643
  If the bound graphics pipeline was created with VK_PIPELINE_CREATE_2_ENABLE_LEGACY_DITHERING_BIT_EXT, the current render pass **must** have begun with vkCmdBeginRenderingKHR with the VK_RENDERING_ENABLE_LEGACY_DITHERING_BIT_EXT flag
- VUID-vkCmdDrawClusterHUAWEI-stage-06480

The bound graphics pipeline **must** not have been created with the VkPipelineShaderStageCreateInfo::stage member of an element of VkGraphicsPipelineCreateInfo::pStages set to VK_SHADER_STAGE_VERTEX_BIT, VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT, VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT or VK_SHADER_STAGE_GEOMETRY_BIT

- VUID-vkCmdDrawClusterHUAWEI-None-07074
  Transform Feedback Queries **must** not be active
- VUID-vkCmdDrawClusterHUAWEI-None-07075
  Primitives Generated Queries **must** not be active
- VUID-vkCmdDrawClusterHUAWEI-pipelineStatistics-07076
  The pipelineStatistics member used to create any active Pipeline Statistics Query **must** not contain VK_QUERY_PIPELINE_STATISTIC_INPUT_ASSEMBLY_VERTICES_BIT, VK_QUERY_PIPELINE_STATISTIC_INPUT_ASSEMBLY_PRIMITIVES_BIT, VK_QUERY_PIPELINE_STATISTIC_VERTEX_SHADER_INVOCATIONS_BIT, VK_QUERY_PIPELINE_STATISTIC_GEOMETRY_SHADER_INVOCATIONS_BIT, VK_QUERY_PIPELINE_STATISTIC_GEOMETRY_SHADER_PRIMITIVES_BIT, VK_QUERY_PIPELINE_STATISTIC_CLIPPING_INVOCATIONS_BIT, VK_QUERY_PIPELINE_STATISTIC_CLIPPING_PRIMITIVES_BIT, VK_QUERY_PIPELINE_STATISTIC_TESSELLATION_CONTROL_SHADER_PATCHES_BIT, or VK_QUERY_PIPELINE_STATISTIC_TESSELLATION_EVALUATION_SHADER_INVOCATIONS_BIT
- VUID-vkCmdDrawClusterHUAWEI-None-07819
  The pipelineStatistics member used to create any active Pipeline Statistics Query **must** not contain VK_QUERY_PIPELINE_STATISTIC_TASK_SHADER_INVOCATIONS_BIT_EXT, or VK_QUERY_PIPELINE_STATISTIC_MESH_SHADER_INVOCATIONS_BIT_EXT
- VUID-vkCmdDrawClusterHUAWEI-groupCountX-07820
  groupCountX **must** be less than or equal to VkPhysicalDeviceClusterCullingShaderPropertiesHUAWEI::maxWorkGroupCount[0]
- VUID-vkCmdDrawClusterHUAWEI-groupCountY-07821
  groupCountY **must** be less than or equal to VkPhysicalDeviceClusterCullingShaderPropertiesHUAWEI::maxWorkGroupCount[1]
- VUID-vkCmdDrawClusterHUAWEI-groupCountZ-07822
  groupCountZ **must** be less than or equal to VkPhysicalDeviceClusterCullingShaderPropertiesHUAWEI::maxWorkGroupCount[2]
- VUID-vkCmdDrawClusterHUAWEI-ClusterCullingHUAWEI-07823
  The current pipeline bound to VK_PIPELINE_BIND_POINT_GRAPHICS **must** contain a shader stage using the ClusterCullingHUAWEI Execution Model

## Valid Usage (Implicit)

- VUID-vkCmdDrawClusterHUAWEI-commandBuffer-parameter
  commandBuffer **must** be a valid VkCommandBuffer handle
- VUID-vkCmdDrawClusterHUAWEI-commandBuffer-recording
  commandBuffer **must** be in the recording state

- VUID-vkCmdDrawClusterHUAWEI-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- VUID-vkCmdDrawClusterHUAWEI-renderpass
  This command **must** only be called inside of a render pass instance
- VUID-vkCmdDrawClusterHUAWEI-videocoding
  This command **must** only be called outside of a video coding scope

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Inside | Outside | Graphics | Action |

To record an indirect cluster culling drawing command, call:

```
// Provided by VK_HUAWEI_cluster_culling_shader
void vkCmdDrawClusterIndirectHUAWEI(
    VkCommandBuffer                             commandBuffer,
    VkBuffer                                    buffer,
    VkDeviceSize                                offset);
```

- `commandBuffer` is the command buffer into which the command is recorded.
- `buffer` is the buffer containing draw parameters.
- `offset` is the byte offset into `buffer` where parameters begin.

`vkCmdDrawClusterIndirectHUAWEI` behaves similarly to vkCmdDrawClusterHUAWEI except that the parameters are read by the device from a buffer during execution. The parameters of the dispatch are encoded in a VkDispatchIndirectCommand structure taken from buffer starting at offset. Note the cluster culling shader pipeline only accepts vkCmdDrawClusterHUAWEI and `vkCmdDrawClusterIndirectHUAWEI` as drawing commands.

## Valid Usage

- VUID-vkCmdDrawClusterIndirectHUAWEI-magFilter-04553

If a VkSampler created with magFilter or minFilter equal to VK_FILTER_LINEAR, reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE, and compareEnable equal to VK_FALSE is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT

- VUID-vkCmdDrawClusterIndirectHUAWEI-magFilter-09598
  If a VkSampler created with magFilter or minFilter equal to VK_FILTER_LINEAR and reductionMode equal to either VK_SAMPLER_REDUCTION_MODE_MIN or VK_SAMPLER_REDUCTION_MODE_MAX is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT
- VUID-vkCmdDrawClusterIndirectHUAWEI-mipmapMode-04770
  If a VkSampler created with mipmapMode equal to VK_SAMPLER_MIPMAP_MODE_LINEAR, reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE, and compareEnable equal to VK_FALSE is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT
- VUID-vkCmdDrawClusterIndirectHUAWEI-mipmapMode-09599
  If a VkSampler created with mipmapMode equal to VK_SAMPLER_MIPMAP_MODE_LINEAR and reductionMode equal to either VK_SAMPLER_REDUCTION_MODE_MIN or VK_SAMPLER_REDUCTION_MODE_MAX is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT
- VUID-vkCmdDrawClusterIndirectHUAWEI-unnormalizedCoordinates-09635
  If a VkSampler created with unnormalizedCoordinates equal to VK_TRUE is used to sample a VkImageView as a result of this command, then the image view's levelCount and layerCount **must** be 1
- VUID-vkCmdDrawClusterIndirectHUAWEI-unnormalizedCoordinates-09636
  If a VkSampler created with unnormalizedCoordinates equal to VK_TRUE is used to sample a VkImageView as a result of this command, then the image view's viewType **must** be VK_IMAGE_VIEW_TYPE_1D or VK_IMAGE_VIEW_TYPE_2D
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-06479
  If a VkImageView is sampled with depth comparison, the image view's format features **must** contain VK_FORMAT_FEATURE_2_SAMPLED_IMAGE_DEPTH_COMPARISON_BIT
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-02691
  If a VkImageView is accessed using atomic operations as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_STORAGE_IMAGE_ATOMIC_BIT
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-07888
  If a VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER descriptor is accessed using atomic operations as a result of this command, then the storage texel buffer's format features **must** contain VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_ATOMIC_BIT
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-02692
  If a VkImageView is sampled with VK_FILTER_CUBIC_EXT as a result of this command, then

the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_CUBIC_BIT_EXT

- VUID-vkCmdDrawClusterIndirectHUAWEI-None-02693
  If the VK_EXT_filter_cubic extension is not enabled and any VkImageView is sampled with VK_FILTER_CUBIC_EXT as a result of this command, it **must** not have a VkImageViewType of VK_IMAGE_VIEW_TYPE_3D, VK_IMAGE_VIEW_TYPE_CUBE, or VK_IMAGE_VIEW_TYPE_CUBE_ARRAY

- VUID-vkCmdDrawClusterIndirectHUAWEI-filterCubic-02694
  Any VkImageView being sampled with VK_FILTER_CUBIC_EXT as a result of this command **must** have a VkImageViewType and format that supports cubic filtering, as specified by VkFilterCubicImageViewImageFormatPropertiesEXT::filterCubic returned by vkGetPhysicalDeviceImageFormatProperties2

- VUID-vkCmdDrawClusterIndirectHUAWEI-filterCubicMinmax-02695
  Any VkImageView being sampled with VK_FILTER_CUBIC_EXT with a reduction mode of either VK_SAMPLER_REDUCTION_MODE_MIN or VK_SAMPLER_REDUCTION_MODE_MAX as a result of this command **must** have a VkImageViewType and format that supports cubic filtering together with minmax filtering, as specified by VkFilterCubicImageViewImageFormatPropertiesEXT::filterCubicMinmax returned by vkGetPhysicalDeviceImageFormatProperties2

- VUID-vkCmdDrawClusterIndirectHUAWEI-cubicRangeClamp-09212
  If the cubicRangeClamp feature is not enabled, then any VkImageView being sampled with VK_FILTER_CUBIC_EXT as a result of this command **must** not have a VkSamplerReductionModeCreateInfo::reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE_RANGECLAMP_QCOM

- VUID-vkCmdDrawClusterIndirectHUAWEI-reductionMode-09213
  Any VkImageView being sampled with a VkSamplerReductionModeCreateInfo::reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE_RANGECLAMP_QCOM as a result of this command **must** sample with VK_FILTER_CUBIC_EXT

- VUID-vkCmdDrawClusterIndirectHUAWEI-selectableCubicWeights-09214
  If the selectableCubicWeights feature is not enabled, then any VkImageView being sampled with VK_FILTER_CUBIC_EXT as a result of this command **must** have VkSamplerCubicWeightsCreateInfoQCOM::cubicWeights equal to VK_CUBIC_FILTER_WEIGHTS_CATMULL_ROM_QCOM

- VUID-vkCmdDrawClusterIndirectHUAWEI-flags-02696
  Any VkImage created with a VkImageCreateInfo::flags containing VK_IMAGE_CREATE_CORNER_SAMPLED_BIT_NV sampled as a result of this command **must** only be sampled using a VkSamplerAddressMode of VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_EDGE

- VUID-vkCmdDrawClusterIndirectHUAWEI-OpTypeImage-07027
  For any VkImageView being written as a storage image where the image format field of the OpTypeImage is Unknown, the view's format features **must** contain VK_FORMAT_FEATURE_2_STORAGE_WRITE_WITHOUT_FORMAT_BIT

- VUID-vkCmdDrawClusterIndirectHUAWEI-OpTypeImage-07028
  For any VkImageView being read as a storage image where the image format field of the OpTypeImage is Unknown, the view's format features **must** contain VK_FORMAT_FEATURE_2_STORAGE_READ_WITHOUT_FORMAT_BIT

- VUID-vkCmdDrawClusterIndirectHUAWEI-OpTypeImage-07029
  For any VkBufferView being written as a storage texel buffer where the image format field of the `OpTypeImage` is `Unknown`, the view's buffer features **must** contain `VK_FORMAT_FEATURE_2_STORAGE_WRITE_WITHOUT_FORMAT_BIT`
- VUID-vkCmdDrawClusterIndirectHUAWEI-OpTypeImage-07030
  Any VkBufferView being read as a storage texel buffer where the image format field of the `OpTypeImage` is `Unknown` then the view's buffer features **must** contain `VK_FORMAT_FEATURE_2_STORAGE_READ_WITHOUT_FORMAT_BIT`
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-08600
  For each set *n* that is statically used by a bound shader, a descriptor set **must** have been bound to *n* at the same pipeline bind point, with a VkPipelineLayout that is compatible for set *n*, with the VkPipelineLayout used to create the current VkPipeline or the VkDescriptorSetLayout array used to create the current VkShaderEXT , as described in Pipeline Layout Compatibility
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-08601
  For each push constant that is statically used by a bound shader, a push constant value **must** have been set for the same pipeline bind point, with a VkPipelineLayout that is compatible for push constants, with the VkPipelineLayout used to create the current VkPipeline or the VkDescriptorSetLayout array used to create the current VkShaderEXT , as described in Pipeline Layout Compatibility
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-10068
  For each array of resources that is used by a bound shader, the indices used to access members of the array **must** be less than the descriptor count for the identified binding in the descriptor sets used by this command
- VUID-vkCmdDrawClusterIndirectHUAWEI-maintenance4-08602
  If the `maintenance4` feature is not enabled, then for each push constant that is statically used by a bound shader, a push constant value **must** have been set for the same pipeline bind point, with a VkPipelineLayout that is compatible for push constants, with the VkPipelineLayout used to create the current VkPipeline or the VkDescriptorSetLayout and VkPushConstantRange arrays used to create the current VkShaderEXT , as described in Pipeline Layout Compatibility
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-08114
  Descriptors in each bound descriptor set, specified via vkCmdBindDescriptorSets, **must** be valid as described by descriptor validity if they are statically used by the VkPipeline bound to the pipeline bind point used by this command and the bound VkPipeline was not created with `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-08115
  If the descriptors used by the VkPipeline bound to the pipeline bind point were specified via vkCmdBindDescriptorSets, the bound VkPipeline **must** have been created without `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-08116
  Descriptors in bound descriptor buffers, specified via vkCmdSetDescriptorBufferOffsetsEXT, **must** be valid if they are dynamically used by the VkPipeline bound to the pipeline bind point used by this command and the bound VkPipeline was created with `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`

- VUID-vkCmdDrawClusterIndirectHUAWEI-None-08604
  Descriptors in bound descriptor buffers, specified via vkCmdSetDescriptorBufferOffsetsEXT, **must** be valid if they are dynamically used by any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-08117
  If the descriptors used by the VkPipeline bound to the pipeline bind point were specified via vkCmdSetDescriptorBufferOffsetsEXT, the bound VkPipeline **must** have been created with `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-08119
  If a descriptor is dynamically used with a VkPipeline created with `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`, the descriptor memory **must** be resident
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-08605
  If a descriptor is dynamically used with a VkShaderEXT created with a `VkDescriptorSetLayout` that was created with `VK_DESCRIPTOR_SET_LAYOUT_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`, the descriptor memory **must** be resident
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-08606
  If the `shaderObject` feature is not enabled, a valid pipeline **must** be bound to the pipeline bind point used by this command
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-08608
  If a pipeline is bound to the pipeline bind point used by this command, there **must** not have been any calls to dynamic state setting commands for any state specified statically in the VkPipeline object bound to the pipeline bind point used by this command, since that pipeline was bound
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-08609
  If the VkPipeline object bound to the pipeline bind point used by this command or any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a VkSampler object that uses unnormalized coordinates, that sampler **must** not be used to sample from any VkImage with a VkImageView of the type `VK_IMAGE_VIEW_TYPE_3D`, `VK_IMAGE_VIEW_TYPE_CUBE`, `VK_IMAGE_VIEW_TYPE_1D_ARRAY`, `VK_IMAGE_VIEW_TYPE_2D_ARRAY` or `VK_IMAGE_VIEW_TYPE_CUBE_ARRAY`, in any shader stage
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-08610
  If the VkPipeline object bound to the pipeline bind point used by this command or any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a VkSampler object that uses unnormalized coordinates, that sampler **must** not be used with any of the SPIR-V `OpImageSample*` or `OpImageSparseSample*` instructions with `ImplicitLod`, `Dref` or `Proj` in their name, in any shader stage
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-08611
  If the VkPipeline object bound to the pipeline bind point used by this command or any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a VkSampler object that uses unnormalized coordinates, that sampler **must** not be used with any of the SPIR-V `OpImageSample*` or `OpImageSparseSample*` instructions that includes a LOD bias or any offset values, in any shader stage

- VUID-vkCmdDrawClusterIndirectHUAWEI-None-08607
  If the `shaderObject` is enabled, either a valid pipeline **must** be bound to the pipeline bind point used by this command, or a valid combination of valid and VK_NULL_HANDLE shader objects **must** be bound to every supported shader stage corresponding to the pipeline bind point used by this command
- VUID-vkCmdDrawClusterIndirectHUAWEI-uniformBuffers-06935
  If any stage of the VkPipeline object bound to the pipeline bind point used by this command accesses a uniform buffer, and that stage was created without enabling either `VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_EXT` or `VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_2_EXT` for `uniformBuffers`, and the `robustBufferAccess` feature is not enabled, that stage **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-08612
  If the `robustBufferAccess` feature is not enabled, and any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a uniform buffer, it **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point
- VUID-vkCmdDrawClusterIndirectHUAWEI-storageBuffers-06936
  If any stage of the VkPipeline object bound to the pipeline bind point used by this command accesses a storage buffer, and that stage was created without enabling either `VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_EXT` or `VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_2_EXT` for `storageBuffers`, and the `robustBufferAccess` feature is not enabled, that stage **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-08613
  If the `robustBufferAccess` feature is not enabled, and any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a storage buffer, it **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point
- VUID-vkCmdDrawClusterIndirectHUAWEI-commandBuffer-02707
  If `commandBuffer` is an unprotected command buffer and `protectedNoFault` is not supported, any resource accessed by bound shaders **must** not be a protected resource
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-06550
  If a bound shader accesses a VkSampler or VkImageView object that enables sampler $Y'C_BC_R$ conversion, that object **must** only be used with `OpImageSample*` or `OpImageSparseSample*` instructions
- VUID-vkCmdDrawClusterIndirectHUAWEI-ConstOffset-06551
  If a bound shader accesses a VkSampler or VkImageView object that enables sampler $Y'C_BC_R$ conversion, that object **must** not use the `ConstOffset` and `Offset` operands
- VUID-vkCmdDrawClusterIndirectHUAWEI-viewType-07752
  If a VkImageView is accessed as a result of this command, then the image view's `viewType` **must** match the `Dim` operand of the `OpTypeImage` as described in Compatibility Between SPIR-V Image Dimensions and Vulkan ImageView Types

- VUID-vkCmdDrawClusterIndirectHUAWEI-format-07753
  If a VkImageView is accessed as a result of this command, then the numeric type of the image view's `format` and the `Sampled Type` operand of the `OpTypeImage` **must** match
- VUID-vkCmdDrawClusterIndirectHUAWEI-OpImageWrite-08795
  If a VkImageView created with a format other than `VK_FORMAT_A8_UNORM_KHR` is accessed using `OpImageWrite` as a result of this command, then the `Type` of the `Texel` operand of that instruction **must** have at least as many components as the image view's format
- VUID-vkCmdDrawClusterIndirectHUAWEI-OpImageWrite-08796
  If a VkImageView created with the format `VK_FORMAT_A8_UNORM_KHR` is accessed using `OpImageWrite` as a result of this command, then the `Type` of the `Texel` operand of that instruction **must** have four components
- VUID-vkCmdDrawClusterIndirectHUAWEI-OpImageWrite-04469
  If a VkBufferView is accessed using `OpImageWrite` as a result of this command, then the `Type` of the `Texel` operand of that instruction **must** have at least as many components as the buffer view's format
- VUID-vkCmdDrawClusterIndirectHUAWEI-SampledType-04470
  If a VkImageView with a VkFormat that has a 64-bit component width is accessed as a result of this command, the `SampledType` of the `OpTypeImage` operand of that instruction **must** have a `Width` of 64
- VUID-vkCmdDrawClusterIndirectHUAWEI-SampledType-04471
  If a VkImageView with a VkFormat that has a component width less than 64-bit is accessed as a result of this command, the `SampledType` of the `OpTypeImage` operand of that instruction **must** have a `Width` of 32
- VUID-vkCmdDrawClusterIndirectHUAWEI-SampledType-04472
  If a VkBufferView with a VkFormat that has a 64-bit component width is accessed as a result of this command, the `SampledType` of the `OpTypeImage` operand of that instruction **must** have a `Width` of 64
- VUID-vkCmdDrawClusterIndirectHUAWEI-SampledType-04473
  If a VkBufferView with a VkFormat that has a component width less than 64-bit is accessed as a result of this command, the `SampledType` of the `OpTypeImage` operand of that instruction **must** have a `Width` of 32
- VUID-vkCmdDrawClusterIndirectHUAWEI-sparseImageInt64Atomics-04474
  If the `sparseImageInt64Atomics` feature is not enabled, VkImage objects created with the `VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT` flag **must** not be accessed by atomic instructions through an `OpTypeImage` with a `SampledType` with a `Width` of 64 by this command
- VUID-vkCmdDrawClusterIndirectHUAWEI-sparseImageInt64Atomics-04475
  If the `sparseImageInt64Atomics` feature is not enabled, VkBuffer objects created with the `VK_BUFFER_CREATE_SPARSE_RESIDENCY_BIT` flag **must** not be accessed by atomic instructions through an `OpTypeImage` with a `SampledType` with a `Width` of 64 by this command
- VUID-vkCmdDrawClusterIndirectHUAWEI-OpImageWeightedSampleQCOM-06971
  If `OpImageWeightedSampleQCOM` is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_2_WEIGHT_SAMPLED_IMAGE_BIT_QCOM`

- VUID-vkCmdDrawClusterIndirectHUAWEI-OpImageWeightedSampleQCOM-06972
  If `OpImageWeightedSampleQCOM` uses a VkImageView as a sample weight image as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_2_WEIGHT_IMAGE_BIT_QCOM`
- VUID-vkCmdDrawClusterIndirectHUAWEI-OpImageBoxFilterQCOM-06973
  If `OpImageBoxFilterQCOM` is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_2_BOX_FILTER_SAMPLED_BIT_QCOM`
- VUID-vkCmdDrawClusterIndirectHUAWEI-OpImageBlockMatchSSDQCOM-06974
  If `OpImageBlockMatchSSDQCOM` is used to read from an VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM`
- VUID-vkCmdDrawClusterIndirectHUAWEI-OpImageBlockMatchSADQCOM-06975
  If `OpImageBlockMatchSADQCOM` is used to read from an VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM`
- VUID-vkCmdDrawClusterIndirectHUAWEI-OpImageBlockMatchSADQCOM-06976
  If `OpImageBlockMatchSADQCOM` or OpImageBlockMatchSSDQCOM is used to read from a reference image as result of this command, then the specified reference coordinates **must** not fail integer texel coordinate validation
- VUID-vkCmdDrawClusterIndirectHUAWEI-OpImageWeightedSampleQCOM-06977
  If `OpImageWeightedSampleQCOM`, `OpImageBoxFilterQCOM`, `OpImageBlockMatchWindowSSDQCOM`, `OpImageBlockMatchWindowSADQCOM`, `OpImageBlockMatchGatherSSDQCOM`, `OpImageBlockMatchGatherSADQCOM`, `OpImageBlockMatchSSDQCOM`, or `OpImageBlockMatchSADQCOM` uses a VkSampler as a result of this command, then the sampler **must** have been created with `VK_SAMPLER_CREATE_IMAGE_PROCESSING_BIT_QCOM`
- VUID-vkCmdDrawClusterIndirectHUAWEI-OpImageWeightedSampleQCOM-06978
  If any command other than `OpImageWeightedSampleQCOM`, `OpImageBoxFilterQCOM`, `OpImageBlockMatchWindowSSDQCOM`, `OpImageBlockMatchWindowSADQCOM`, `OpImageBlockMatchGatherSSDQCOM`, `OpImageBlockMatchGatherSADQCOM`, `OpImageBlockMatchSSDQCOM`, or `OpImageBlockMatchSADQCOM` uses a VkSampler as a result of this command, then the sampler **must** not have been created with `VK_SAMPLER_CREATE_IMAGE_PROCESSING_BIT_QCOM`
- VUID-vkCmdDrawClusterIndirectHUAWEI-OpImageBlockMatchWindow-09215
  If a `OpImageBlockMatchWindow*QCOM` or `OpImageBlockMatchGather*QCOM` instruction is used to read from an VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM`
- VUID-vkCmdDrawClusterIndirectHUAWEI-OpImageBlockMatchWindow-09216
  If a `OpImageBlockMatchWindow*QCOM` or `OpImageBlockMatchGather*QCOM` instruction is used to read from an VkImageView as a result of this command, then the image view's format **must** be a single-component format
- VUID-vkCmdDrawClusterIndirectHUAWEI-OpImageBlockMatchWindow-09217
  If a `OpImageBlockMatchWindow*QCOM` or `OpImageBlockMatchGather*QCOM` read from a reference image as result of this command, then the specified reference coordinates **must** not fail

integer texel coordinate validation

- VUID-vkCmdDrawClusterIndirectHUAWEI-None-07288
  Any shader invocation executed by this command **must** terminate
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-09600
  If a descriptor with type equal to any of `VK_DESCRIPTOR_TYPE_SAMPLE_WEIGHT_IMAGE_QCOM`, `VK_DESCRIPTOR_TYPE_BLOCK_MATCH_IMAGE_QCOM`, `VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE`, `VK_DESCRIPTOR_TYPE_STORAGE_IMAGE`, or `VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT` is accessed as a result of this command, the image subresource identified by that descriptor **must** be in the image layout identified when the descriptor was written
- VUID-vkCmdDrawClusterIndirectHUAWEI-renderPass-02684
  The current render pass **must** be compatible with the `renderPass` member of the `VkGraphicsPipelineCreateInfo` structure specified when creating the `VkPipeline` bound to `VK_PIPELINE_BIND_POINT_GRAPHICS`
- VUID-vkCmdDrawClusterIndirectHUAWEI-subpass-02685
  The subpass index of the current render pass **must** be equal to the `subpass` member of the `VkGraphicsPipelineCreateInfo` structure specified when creating the `VkPipeline` bound to `VK_PIPELINE_BIND_POINT_GRAPHICS`
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-07748
  If any shader statically accesses an input attachment, a valid descriptor **must** be bound to the pipeline via a descriptor set
- VUID-vkCmdDrawClusterIndirectHUAWEI-OpTypeImage-07468
  If any shader executed by this pipeline accesses an `OpTypeImage` variable with a `Dim` operand of `SubpassData`, it **must** be decorated with an `InputAttachmentIndex` that corresponds to a valid input attachment in the current subpass
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-07469
  Input attachment views accessed in a subpass **must** be created with the same VkFormat as the corresponding subpass definition, and be created with a VkImageView that is compatible with the attachment referenced by the subpass' `pInputAttachments`[`InputAttachmentIndex`] in the currently bound VkFramebuffer as specified by Fragment Input Attachment Compatibility
- VUID-vkCmdDrawClusterIndirectHUAWEI-pDepthInputAttachmentIndex-09595
  Input attachment views accessed in a dynamic render pass with a `InputAttachmentIndex` referenced by VkRenderingInputAttachmentIndexInfoKHR, or no `InputAttachmentIndex` if VkRenderingInputAttachmentIndexInfoKHR:`pDepthInputAttachmentIndex` or VkRenderingInputAttachmentIndexInfoKHR:`pStencilInputAttachmentIndex` are `NULL`, **must** be created with a VkImageView that is compatible with the corresponding color, depth, or stencil attachment in VkRenderingInfo
- VUID-vkCmdDrawClusterIndirectHUAWEI-pDepthInputAttachmentIndex-09596
  Input attachment views accessed in a dynamic render pass via a shader object **must** have an `InputAttachmentIndex` if both VkRenderingInputAttachmentIndexInfoKHR:`pDepthInputAttachmentIndex` and VkRenderingInputAttachmentIndexInfoKHR:`pStencilInputAttachmentIndex` are non-`NULL`
- VUID-vkCmdDrawClusterIndirectHUAWEI-InputAttachmentIndex-09597
  If an input attachment view accessed in a dynamic render pass via a shader object has an

InputAttachmentIndex, the InputAttachmentIndex **must** match an index in VkRenderingInputAttachmentIndexInfoKHR

- VUID-vkCmdDrawClusterIndirectHUAWEI-None-06537
  Memory backing image subresources used as attachments in the current render pass **must** not be written in any way other than as an attachment by this command
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-09000
  If a color attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass, it is not in the VK_IMAGE_LAYOUT_ATTACHMENT_FEEDBACK_LOOP_OPTIMAL_EXT image layout, and either:
  - the VK_PIPELINE_CREATE_COLOR_ATTACHMENT_FEEDBACK_LOOP_BIT_EXT is set on the currently bound pipeline or
  - the last call to vkCmdSetAttachmentFeedbackLoopEnableEXT included VK_IMAGE_ASPECT_COLOR_BIT and
    - there is no currently bound graphics pipeline or
    - the currently bound graphics pipeline was created with VK_DYNAMIC_STATE_ATTACHMENT_FEEDBACK_LOOP_ENABLE_EXT

  it **must** not be accessed in any way other than as an attachment by this command
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-09001
  If a depth attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass, it is not in the VK_IMAGE_LAYOUT_ATTACHMENT_FEEDBACK_LOOP_OPTIMAL_EXT image layout, and either:
  - the VK_PIPELINE_CREATE_DEPTH_STENCIL_ATTACHMENT_FEEDBACK_LOOP_BIT_EXT is set on the currently bound pipeline or
  - the last call to vkCmdSetAttachmentFeedbackLoopEnableEXT included VK_IMAGE_ASPECT_DEPTH_BIT and
    - there is no currently bound graphics pipeline or
    - the currently bound graphics pipeline was created with VK_DYNAMIC_STATE_ATTACHMENT_FEEDBACK_LOOP_ENABLE_EXT

  it **must** not be accessed in any way other than as an attachment by this command
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-09002
  If a stencil attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass, it is not in the VK_IMAGE_LAYOUT_ATTACHMENT_FEEDBACK_LOOP_OPTIMAL_EXT image layout, and either:
  - the VK_PIPELINE_CREATE_DEPTH_STENCIL_ATTACHMENT_FEEDBACK_LOOP_BIT_EXT is set on the currently bound pipeline or
  - the last call to vkCmdSetAttachmentFeedbackLoopEnableEXT included VK_IMAGE_ASPECT_STENCIL_BIT and
    - there is no currently bound graphics pipeline or
    - the currently bound graphics pipeline was created with

VK_DYNAMIC_STATE_ATTACHMENT_FEEDBACK_LOOP_ENABLE_EXT

it **must** not be accessed in any way other than as an attachment by this command

- VUID-vkCmdDrawClusterIndirectHUAWEI-None-09003
  If an attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass, it **must** not be accessed in any way other than as an attachment, storage image, or sampled image by this command
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-06539
  If any previously recorded command in the current subpass accessed an image subresource used as an attachment in this subpass in any way other than as an attachment, this command **must** not write to that image subresource as an attachment
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-06886
  If the current render pass instance uses a depth/stencil attachment with a read-only layout for the depth aspect, depth writes **must** be disabled
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-06887
  If the current render pass instance uses a depth/stencil attachment with a read-only layout for the stencil aspect, both front and back `writeMask` are not zero, and stencil test is enabled, all stencil ops **must** be `VK_STENCIL_OP_KEEP`
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-07831
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VIEWPORT` dynamic state enabled then vkCmdSetViewport **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-07832
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_SCISSOR` dynamic state enabled then vkCmdSetScissor **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-07833
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_LINE_WIDTH` dynamic state enabled then vkCmdSetLineWidth **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-08617
  If a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE`, and the most recent call to vkCmdSetPolygonModeEXT in the current command buffer set `polygonMode` to `VK_POLYGON_MODE_LINE`, vkCmdSetLineWidth **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-08618
  If a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE`, and the most recent call to vkCmdSetPrimitiveTopologyEXT in the current command buffer set `primitiveTopology` to any line topology, vkCmdSetLineWidth **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawClusterIndirectHUAWEI-None-08619
  If a shader object that outputs line primitives is bound to the VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT or VK_SHADER_STAGE_GEOMETRY_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, vkCmdSetLineWidth **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-07834
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_BIAS dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of depthBiasEnable is VK_TRUE, then vkCmdSetDepthBounds or vkCmdSetDepthBias2EXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-07835
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_BLEND_CONSTANTS dynamic state enabled then vkCmdSetBlendConstants **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-08621
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetColorBlendEnableEXT in the current command buffer set any element of pColorBlendEnables to VK_TRUE, and the most recent call to vkCmdSetColorBlendEquationEXT in the current command buffer set the same element of pColorBlendEquations to a VkColorBlendEquationEXT structure with any VkBlendFactor member with a value of VK_BLEND_FACTOR_CONSTANT_COLOR, VK_BLEND_FACTOR_ONE_MINUS_CONSTANT_COLOR, VK_BLEND_FACTOR_CONSTANT_ALPHA, or VK_BLEND_FACTOR_ONE_MINUS_CONSTANT_ALPHA, vkCmdSetBlendConstants **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-07836
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_BOUNDS dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of depthBoundsTestEnable is VK_TRUE, then vkCmdSetDepthBounds **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-07837
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_STENCIL_COMPARE_MASK dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of stencilTestEnable is VK_TRUE, then vkCmdSetStencilCompareMask **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawClusterIndirectHUAWEI-None-07838
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_STENCIL_WRITE_MASK dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of stencilTestEnable is VK_TRUE, then vkCmdSetStencilWriteMask **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-07839
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_STENCIL_REFERENCE dynamic state enabled, the current value of and rasterizerDiscardEnable is VK_FALSE, the current value of stencilTestEnable is VK_TRUE, then vkCmdSetStencilReference **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterIndirectHUAWEI-maxMultiviewInstanceIndex-02688
  If the draw is recorded in a render pass instance with multiview enabled, the maximum instance index **must** be less than or equal to VkPhysicalDeviceMultiviewProperties::maxMultiviewInstanceIndex
- VUID-vkCmdDrawClusterIndirectHUAWEI-sampleLocationsEnable-02689
  If the bound graphics pipeline was created with VkPipelineSampleLocationsStateCreateInfoEXT::sampleLocationsEnable set to VK_TRUE and the current subpass has a depth/stencil attachment, then that attachment **must** have been created with the VK_IMAGE_CREATE_SAMPLE_LOCATIONS_COMPATIBLE_DEPTH_BIT_EXT bit set
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-06666
  If the VK_EXT_sample_locations extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of sampleLocationsEnable is VK_TRUE, then vkCmdSetSampleLocationsEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-07840
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_CULL_MODE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetCullModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-07841
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_FRONT_FACE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetFrontFaceEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-07843
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_TEST_ENABLE dynamic state enabled, and the

current value of rasterizerDiscardEnable is VK_FALSE, vkCmdSetDepthTestEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawClusterIndirectHUAWEI-None-07844
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_WRITE_ENABLE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetDepthWriteEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-07845
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_COMPARE_OP dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of depthTestEnable is VK_TRUE, then vkCmdSetDepthCompareOpEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-07846
  If the depthBounds feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_BOUNDS_TEST_ENABLE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetDepthBoundsTestEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-07847
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_STENCIL_TEST_ENABLE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetStencilTestEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-07848
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_STENCIL_OP dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, the current value of stencilTestEnable is VK_TRUE, then vkCmdSetStencilOpEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterIndirectHUAWEI-viewportCount-03417
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, but not the VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT dynamic state enabled, then vkCmdSetViewportWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the viewportCount parameter of vkCmdSetViewportWithCountEXT **must** match the VkPipelineViewportStateCreateInfo::scissorCount of the pipeline
- VUID-vkCmdDrawClusterIndirectHUAWEI-scissorCount-03418
  If the bound graphics pipeline state was created with the

VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT dynamic state enabled, but not the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, then vkCmdSetScissorWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the scissorCount parameter of vkCmdSetScissorWithCountEXT **must** match the VkPipelineViewportStateCreateInfo::viewportCount of the pipeline

- VUID-vkCmdDrawClusterIndirectHUAWEI-viewportCount-03419
  If the bound graphics pipeline state was created with both the VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT and VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic states enabled then both vkCmdSetViewportWithCountEXT and vkCmdSetScissorWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the viewportCount parameter of vkCmdSetViewportWithCountEXT **must** match the scissorCount parameter of vkCmdSetScissorWithCountEXT

- VUID-vkCmdDrawClusterIndirectHUAWEI-None-08635
  If a shader object is bound to any graphics stage, then both vkCmdSetViewportWithCountEXT and vkCmdSetScissorWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the viewportCount parameter of vkCmdSetViewportWithCountEXT **must** match the scissorCount parameter of vkCmdSetScissorWithCountEXT

- VUID-vkCmdDrawClusterIndirectHUAWEI-viewportCount-04137
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, but not the VK_DYNAMIC_STATE_VIEWPORT_W_SCALING_NV dynamic state enabled, then the bound graphics pipeline **must** have been created with VkPipelineViewportWScalingStateCreateInfoNV::viewportCount greater or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT

- VUID-vkCmdDrawClusterIndirectHUAWEI-viewportCount-04138
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT and VK_DYNAMIC_STATE_VIEWPORT_W_SCALING_NV dynamic states enabled then the viewportCount parameter in the last call to vkCmdSetViewportWScalingNV **must** be greater than or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT

- VUID-vkCmdDrawClusterIndirectHUAWEI-None-09232
  If the VK_NV_clip_space_w_scaling extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetViewportWScalingEnableNV in the current command buffer set viewportWScalingEnable to VK_TRUE, then vkCmdSetViewportWScalingNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawClusterIndirectHUAWEI-None-08636
  If the VK_NV_clip_space_w_scaling extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetViewportWScalingEnableNV in the current command buffer set viewportWScalingEnable to VK_TRUE, then the viewportCount parameter in the last call to vkCmdSetViewportWScalingNV **must** be greater than or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT

- VUID-vkCmdDrawClusterIndirectHUAWEI-viewportCount-04139
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT` dynamic state enabled, but not the `VK_DYNAMIC_STATE_VIEWPORT_SHADING_RATE_PALETTE_NV` dynamic state enabled, then the bound graphics pipeline **must** have been created with VkPipelineViewportShadingRateImageStateCreateInfoNV::`viewportCount` greater or equal to the `viewportCount` parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawClusterIndirectHUAWEI-viewportCount-04140
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT` and `VK_DYNAMIC_STATE_VIEWPORT_SHADING_RATE_PALETTE_NV` dynamic states enabled then the `viewportCount` parameter in the last call to vkCmdSetViewportShadingRatePaletteNV **must** be greater than or equal to the `viewportCount` parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawClusterIndirectHUAWEI-shadingRateImage-09233
  If the `shadingRateImage` feature is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE`, then vkCmdSetCoarseSampleOrderNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterIndirectHUAWEI-shadingRateImage-09234
  If the `shadingRateImage` feature is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE`, and the most recent call to vkCmdSetShadingRateImageEnableNV in the current command buffer set `shadingRateImageEnable` to `VK_TRUE`, then vkCmdSetViewportShadingRatePaletteNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-08637
  If the `shadingRateImage` feature is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE`, and the most recent call to vkCmdSetShadingRateImageEnableNV in the current command buffer set `shadingRateImageEnable` to `VK_TRUE`, then the `viewportCount` parameter in the last call to vkCmdSetViewportShadingRatePaletteNV **must** be greater than or equal to the `viewportCount` parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawClusterIndirectHUAWEI-VkPipelineVieportCreateInfo-04141
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT` dynamic state enabled and a VkPipelineViewportSwizzleStateCreateInfoNV structure chained from VkPipelineViewportStateCreateInfo, then the bound graphics pipeline **must** have been created with VkPipelineViewportSwizzleStateCreateInfoNV::`viewportCount` greater or equal to the `viewportCount` parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawClusterIndirectHUAWEI-VkPipelineVieportCreateInfo-04142
  If the bound graphics pipeline state was created with the

VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled and a VkPipelineViewportExclusiveScissorStateCreateInfoNV structure chained from VkPipelineViewportStateCreateInfo, then the bound graphics pipeline **must** have been created with VkPipelineViewportExclusiveScissorStateCreateInfoNV ::exclusiveScissorCount greater or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT

- VUID-vkCmdDrawClusterIndirectHUAWEI-None-07878
  If the exclusiveScissor feature is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_EXCLUSIVE_SCISSOR_ENABLE_NV dynamic state enabled, then vkCmdSetExclusiveScissorEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-07879
  If the exclusiveScissor feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_EXCLUSIVE_SCISSOR_NV dynamic state enabled, and the most recent call to vkCmdSetExclusiveScissorEnableNV in the current command buffer set any element of pExclusiveScissorEnables to VK_TRUE, then vkCmdSetExclusiveScissorNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-04876
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_RASTERIZER_DISCARD_ENABLE dynamic state enabled, then vkCmdSetRasterizerDiscardEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-04877
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_BIAS_ENABLE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetDepthBiasEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterIndirectHUAWEI-logicOp-04878
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_LOGIC_OP_EXT dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of logicOpEnable is VK_TRUE, then vkCmdSetLogicOpEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterIndirectHUAWEI-primitiveFragmentShadingRateWithMultipleViewports-04552
  If the primitiveFragmentShadingRateWithMultipleViewports limit is not supported, the bound graphics pipeline was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, and any of the shader stages of the bound graphics pipeline write to the PrimitiveShadingRateKHR built-in, then vkCmdSetViewportWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the viewportCount parameter of vkCmdSetViewportWithCountEXT **must** be 1

- VUID-vkCmdDrawClusterIndirectHUAWEI-primitiveFragmentShadingRateWithMultipleViewports-08642
  If the `primitiveFragmentShadingRateWithMultipleViewports` limit is not supported, and any shader object bound to a graphics stage writes to the `PrimitiveShadingRateKHR` built-in, then vkCmdSetViewportWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the `viewportCount` parameter of `vkCmdSetViewportWithCountEXT` **must** be `1`
- VUID-vkCmdDrawClusterIndirectHUAWEI-blendEnable-04727
  If rasterization is not disabled in the bound graphics pipeline, then for each color attachment in the subpass, if the corresponding image view's format features do not contain `VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT`, then the `blendEnable` member of the corresponding element of the `pAttachments` member of `pColorBlendState` **must** be `VK_FALSE`
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-08643
  If a shader object is bound to the `VK_SHADER_STAGE_FRAGMENT_BIT` stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE`, then for each color attachment in the render pass, if the corresponding image view's format features do not contain `VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT`, then the corresponding member of `pColorBlendEnables` in the most recent call to `vkCmdSetColorBlendEnableEXT` in the current command buffer that affected that attachment index **must** have been `VK_FALSE`
- VUID-vkCmdDrawClusterIndirectHUAWEI-multisampledRenderToSingleSampled-07284
  If rasterization is not disabled in the bound graphics pipeline, and none of the following is enabled:
  - the `VK_AMD_mixed_attachment_samples` extension
  - the `VK_NV_framebuffer_mixed_samples` extension
  - the `multisampledRenderToSingleSampled` feature

  then `rasterizationSamples` for the currently bound graphics pipeline **must** be the same as the current subpass color and/or depth/stencil attachments
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-08644
  If a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE`, and none of the following is enabled:
  - the `VK_AMD_mixed_attachment_samples` extension
  - the `VK_NV_framebuffer_mixed_samples` extension
  - the `multisampledRenderToSingleSampled` feature

  then the most recent call to vkCmdSetRasterizationSamplesEXT in the current command buffer **must** have set `rasterizationSamples` to be the same as the number of samples for the current render pass color and/or depth/stencil attachments
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-08876
  If a shader object is bound to any graphics stage, the current render pass instance **must**

have been begun with vkCmdBeginRenderingKHR

- VUID-vkCmdDrawClusterIndirectHUAWEI-imageView-06172
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the `imageView` member of `pDepthAttachment` is not VK_NULL_HANDLE, and the `layout` member of `pDepthAttachment` is `VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL`, this command **must** not write any values to the depth attachment
- VUID-vkCmdDrawClusterIndirectHUAWEI-imageView-06173
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the `imageView` member of `pStencilAttachment` is not VK_NULL_HANDLE, and the `layout` member of `pStencilAttachment` is `VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL`, this command **must** not write any values to the stencil attachment
- VUID-vkCmdDrawClusterIndirectHUAWEI-imageView-06174
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the `imageView` member of `pDepthAttachment` is not VK_NULL_HANDLE, and the `layout` member of `pDepthAttachment` is `VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_STENCIL_ATTACHMENT_OPTIMAL`, this command **must** not write any values to the depth attachment
- VUID-vkCmdDrawClusterIndirectHUAWEI-imageView-06175
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the `imageView` member of `pStencilAttachment` is not VK_NULL_HANDLE, and the `layout` member of `pStencilAttachment` is `VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_STENCIL_READ_ONLY_OPTIMAL`, this command **must** not write any values to the stencil attachment
- VUID-vkCmdDrawClusterIndirectHUAWEI-imageView-06176
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the `imageView` member of `pDepthAttachment` is not VK_NULL_HANDLE, and the `layout` member of `pDepthAttachment` is `VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL`, this command **must** not write any values to the depth attachment
- VUID-vkCmdDrawClusterIndirectHUAWEI-imageView-06177
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the `imageView` member of `pStencilAttachment` is not VK_NULL_HANDLE, and the `layout` member of `pStencilAttachment` is `VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL`, this command **must** not write any values to the stencil attachment
- VUID-vkCmdDrawClusterIndirectHUAWEI-viewMask-06178
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the currently bound graphics pipeline **must** have been created with a VkPipelineRenderingCreateInfo::`viewMask` equal to VkRenderingInfo::`viewMask`
- VUID-vkCmdDrawClusterIndirectHUAWEI-colorAttachmentCount-06179
  If the `dynamicRenderingUnusedAttachments` feature is not enabled and the current render pass instance was begun with vkCmdBeginRenderingKHR, the currently bound graphics pipeline **must** have been created with a VkPipelineRenderingCreateInfo::`colorAttachmentCount` equal to VkRenderingInfo::`colorAttachmentCount`
- VUID-vkCmdDrawClusterIndirectHUAWEI-dynamicRenderingUnusedAttachments-08910
  If the `dynamicRenderingUnusedAttachments` feature is not enabled, and the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingInfo

::colorAttachmentCount greater than 0, then each element of the VkRenderingInfo::pColorAttachments array with an imageView not equal to VK_NULL_HANDLE **must** have been created with a VkFormat equal to the corresponding element of VkPipelineRenderingCreateInfo::pColorAttachmentFormats used to create the currently bound graphics pipeline

- VUID-vkCmdDrawClusterIndirectHUAWEI-dynamicRenderingUnusedAttachments-08912
  If the dynamicRenderingUnusedAttachments feature is not enabled, and the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingInfo::colorAttachmentCount greater than 0, then each element of the VkRenderingInfo::pColorAttachments array with an imageView equal to VK_NULL_HANDLE **must** have the corresponding element of VkPipelineRenderingCreateInfo::pColorAttachmentFormats used to create the currently bound pipeline equal to VK_FORMAT_UNDEFINED

- VUID-vkCmdDrawClusterIndirectHUAWEI-dynamicRenderingUnusedAttachments-08911
  If the dynamicRenderingUnusedAttachments feature is enabled, and the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingInfo::colorAttachmentCount greater than 0, then each element of the VkRenderingInfo::pColorAttachments array with an imageView not equal to VK_NULL_HANDLE **must** have been created with a VkFormat equal to the corresponding element of VkPipelineRenderingCreateInfo::pColorAttachmentFormats used to create the currently bound graphics pipeline, or the corresponding element of VkPipelineRenderingCreateInfo::pColorAttachmentFormats, if it exists, **must** be VK_FORMAT_UNDEFINED

- VUID-vkCmdDrawClusterIndirectHUAWEI-colorAttachmentCount-09362
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, with a VkRenderingInfo::colorAttachmentCount equal to 1, there is no shader object bound to any graphics stage, and a color attachment with a resolve mode of VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID, each element of the VkRenderingInfo::pColorAttachments array with a resolveImageView not equal to VK_NULL_HANDLE **must** have been created with an image created with a VkExternalFormatANDROID::externalFormat value equal to the VkExternalFormatANDROID::externalFormat value used to create the currently bound graphics pipeline

- VUID-vkCmdDrawClusterIndirectHUAWEI-None-09363
  If there is no shader object bound to any graphics stage, the current render pass instance was begun with vkCmdBeginRenderingKHR and a VkRenderingInfo::colorAttachmentCount equal to 1, and a color attachment with a resolve mode of VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID, each element of the VkRenderingInfo::pColorAttachments array with a imageView not equal to VK_NULL_HANDLE **must** have been created with an image created with a VkExternalFormatANDROID::externalFormat value equal to the VkExternalFormatANDROID::externalFormat value used to create the currently bound graphics pipeline

- VUID-vkCmdDrawClusterIndirectHUAWEI-None-09364
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is no shader object bound to any graphics stage, and the currently bound graphics pipeline was created with a non-zero VkExternalFormatANDROID::externalFormat value and with

the VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT dynamic state enabled, then vkCmdSetColorBlendEnableEXT **must** have set the blend enable to VK_FALSE prior to this drawing command

- VUID-vkCmdDrawClusterIndirectHUAWEI-None-09365
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is no shader object bound to any graphics stage, and the currently bound graphics pipeline was created with a non-zero VkExternalFormatANDROID::externalFormat value and with the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT dynamic state enabled, then vkCmdSetRasterizationSamplesEXT **must** have set rasterizationSamples to VK_SAMPLE_COUNT_1_BIT prior to this drawing command
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-09366
  If there is a shader object bound to any graphics stage, and the current render pass includes a color attachment that uses the VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID resolve mode, then vkCmdSetColorBlendEnableEXT **must** have set blend enable to VK_FALSE prior to this drawing command
- VUID-vkCmdDrawClusterIndirectHUAWEI-rasterizationSamples-09367
  If there is a shader object bound to any graphics stage, and the current render pass includes a color attachment that uses the VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID resolve mode, then vkCmdSetRasterizationSamplesEXT **must** have set rasterizationSamples to VK_SAMPLE_COUNT_1_BIT prior to this drawing command
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-09368
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is no shader object bound to any graphics stage, and the currently bound graphics pipeline was created with a non-zero VkExternalFormatANDROID::externalFormat value and with the VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR dynamic state enabled, then vkCmdSetFragmentShadingRateKHR **must** have set pFragmentSize->width to 1 prior to this drawing command
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-09369
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is no shader object bound to any graphics stage, and the currently bound graphics pipeline was created with a non-zero VkExternalFormatANDROID::externalFormat value and with the VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR dynamic state enabled, then vkCmdSetFragmentShadingRateKHR **must** have set pFragmentSize->height to 1 prior to this drawing command
- VUID-vkCmdDrawClusterIndirectHUAWEI-pFragmentSize-09370
  If there is a shader object bound to any graphics stage, and the current render pass includes a color attachment that uses the VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID resolve mode, then vkCmdSetFragmentShadingRateKHR **must** have set pFragmentSize->width to 1 prior to this drawing command
- VUID-vkCmdDrawClusterIndirectHUAWEI-pFragmentSize-09371
  If there is a shader object bound to any graphics stage, and the current render pass includes a color attachment that uses the

VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID resolve mode, then vkCmdSetFragmentShadingRateKHR **must** have set pFragmentSize->height to 1 prior to this drawing command

- VUID-vkCmdDrawClusterIndirectHUAWEI-None-07749
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_WRITE_ENABLE_EXT dynamic state enabled then vkCmdSetColorWriteEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawClusterIndirectHUAWEI-None-08646
  If the colorWriteEnable feature is enabled on the device, and a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetColorWriteEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawClusterIndirectHUAWEI-attachmentCount-07750
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_WRITE_ENABLE_EXT dynamic state enabled then the attachmentCount parameter of vkCmdSetColorWriteEnableEXT **must** be greater than or equal to the VkPipelineColorBlendStateCreateInfo::attachmentCount of the currently bound graphics pipeline

- VUID-vkCmdDrawClusterIndirectHUAWEI-None-08647
  If the colorWriteEnable feature is enabled on the device, and a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then the attachmentCount parameter of most recent call to vkCmdSetColorWriteEnableEXT in the current command buffer **must** be greater than or equal to the number of color attachments in the current render pass instance

- VUID-vkCmdDrawClusterIndirectHUAWEI-None-07751
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_DISCARD_RECTANGLE_EXT dynamic state enabled then vkCmdSetDiscardRectangleEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command for each discard rectangle in VkPipelineDiscardRectangleStateCreateInfoEXT::discardRectangleCount

- VUID-vkCmdDrawClusterIndirectHUAWEI-None-07880
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_DISCARD_RECTANGLE_ENABLE_EXT dynamic state enabled then vkCmdSetDiscardRectangleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawClusterIndirectHUAWEI-rasterizerDiscardEnable-09236
  If the VK_EXT_discard_rectangles extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetDiscardRectangleEnableEXT in the current command buffer set discardRectangleEnable to VK_TRUE, then vkCmdSetDiscardRectangleEXT **must** have been

called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawClusterIndirectHUAWEI-None-08648
  If the VK_EXT_discard_rectangles extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetDiscardRectangleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawClusterIndirectHUAWEI-None-07881
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_DISCARD_RECTANGLE_MODE_EXT dynamic state enabled then vkCmdSetDiscardRectangleModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawClusterIndirectHUAWEI-None-08649
  If the VK_EXT_discard_rectangles extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetDiscardRectangleEnableEXT in the current command buffer set discardRectangleEnable to VK_TRUE, then vkCmdSetDiscardRectangleModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawClusterIndirectHUAWEI-dynamicRenderingUnusedAttachments-08913
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is not enabled, and VkRenderingInfo::pDepthAttachment->imageView was VK_NULL_HANDLE, the value of VkPipelineRenderingCreateInfo::depthAttachmentFormat used to create the currently bound graphics pipeline **must** be equal to VK_FORMAT_UNDEFINED

- VUID-vkCmdDrawClusterIndirectHUAWEI-dynamicRenderingUnusedAttachments-08914
  If current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is not enabled, and VkRenderingInfo::pDepthAttachment->imageView was not VK_NULL_HANDLE, the value of VkPipelineRenderingCreateInfo::depthAttachmentFormat used to create the currently bound graphics pipeline **must** be equal to the VkFormat used to create VkRenderingInfo::pDepthAttachment->imageView

- VUID-vkCmdDrawClusterIndirectHUAWEI-dynamicRenderingUnusedAttachments-08915
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is enabled, VkRenderingInfo::pDepthAttachment->imageView was not VK_NULL_HANDLE, and the value of VkPipelineRenderingCreateInfo::depthAttachmentFormat used to create the currently bound graphics pipeline was not equal to the VkFormat used to create VkRenderingInfo::pDepthAttachment->imageView, the value of the format **must** be VK_FORMAT_UNDEFINED

- VUID-vkCmdDrawClusterIndirectHUAWEI-dynamicRenderingUnusedAttachments-08916
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is not enabled, and VkRenderingInfo::pStencilAttachment->imageView was VK_NULL_HANDLE, the value of

VkPipelineRenderingCreateInfo::stencilAttachmentFormat used to create the currently bound graphics pipeline **must** be equal to VK_FORMAT_UNDEFINED

- VUID-vkCmdDrawClusterIndirectHUAWEI-dynamicRenderingUnusedAttachments-08917
  If current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is not enabled, and VkRenderingInfo::pStencilAttachment->imageView was not VK_NULL_HANDLE, the value of VkPipelineRenderingCreateInfo::stencilAttachmentFormat used to create the currently bound graphics pipeline **must** be equal to the VkFormat used to create VkRenderingInfo::pStencilAttachment->imageView
- VUID-vkCmdDrawClusterIndirectHUAWEI-dynamicRenderingUnusedAttachments-08918
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is enabled, VkRenderingInfo::pStencilAttachment->imageView was not VK_NULL_HANDLE, and the value of VkPipelineRenderingCreateInfo::stencilAttachmentFormat used to create the currently bound graphics pipeline was not equal to the VkFormat used to create VkRenderingInfo::pStencilAttachment->imageView, the value of the format **must** be VK_FORMAT_UNDEFINED
- VUID-vkCmdDrawClusterIndirectHUAWEI-imageView-06183
  If the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingFragmentShadingRateAttachmentInfoKHR::imageView was not VK_NULL_HANDLE, the currently bound graphics pipeline **must** have been created with VK_PIPELINE_CREATE_RENDERING_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR
- VUID-vkCmdDrawClusterIndirectHUAWEI-imageView-06184
  If the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingFragmentDensityMapAttachmentInfoEXT::imageView was not VK_NULL_HANDLE, the currently bound graphics pipeline **must** have been created with VK_PIPELINE_CREATE_RENDERING_FRAGMENT_DENSITY_MAP_ATTACHMENT_BIT_EXT
- VUID-vkCmdDrawClusterIndirectHUAWEI-colorAttachmentCount-06185
  If the currently bound pipeline was created with a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and the current render pass instance was begun with vkCmdBeginRenderingKHR with a VkRenderingInfo::colorAttachmentCount parameter greater than 0, then each element of the VkRenderingInfo::pColorAttachments array with a imageView not equal to VK_NULL_HANDLE **must** have been created with a sample count equal to the corresponding element of the pColorAttachmentSamples member of VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV used to create the currently bound graphics pipeline
- VUID-vkCmdDrawClusterIndirectHUAWEI-pDepthAttachment-06186
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the currently bound pipeline was created with a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and VkRenderingInfo::pDepthAttachment->imageView was not VK_NULL_HANDLE, the value of the depthStencilAttachmentSamples member of VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV used to create the currently bound graphics pipeline **must** be equal to the sample count used to create VkRenderingInfo::pDepthAttachment->imageView
- VUID-vkCmdDrawClusterIndirectHUAWEI-pStencilAttachment-06187
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the

currently bound pipeline was created with a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and VkRenderingInfo::pStencilAttachment->imageView was not VK_NULL_HANDLE, the value of the depthStencilAttachmentSamples member of VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV used to create the currently bound graphics pipeline **must** be equal to the sample count used to create VkRenderingInfo::pStencilAttachment->imageView

- VUID-vkCmdDrawClusterIndirectHUAWEI-multisampledRenderToSingleSampled-07285
  If the currently bound pipeline was created without a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and the multisampledRenderToSingleSampled feature is not enabled, and the current render pass instance was begun with vkCmdBeginRenderingKHR with a VkRenderingInfo::colorAttachmentCount parameter greater than 0, then each element of the VkRenderingInfo::pColorAttachments array with a imageView not equal to VK_NULL_HANDLE **must** have been created with a sample count equal to the value of rasterizationSamples for the currently bound graphics pipeline
- VUID-vkCmdDrawClusterIndirectHUAWEI-multisampledRenderToSingleSampled-07286
  If the currently bound pipeline was created without a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and the multisampledRenderToSingleSampled feature is not enabled, and VkRenderingInfo::pDepthAttachment->imageView was not VK_NULL_HANDLE, the value of rasterizationSamples for the currently bound graphics pipeline **must** be equal to the sample count used to create VkRenderingInfo::pDepthAttachment->imageView
- VUID-vkCmdDrawClusterIndirectHUAWEI-multisampledRenderToSingleSampled-07287
  If the currently bound pipeline was created without a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and the multisampledRenderToSingleSampled feature is not enabled, and VkRenderingInfo::pStencilAttachment->imageView was not VK_NULL_HANDLE, the value of rasterizationSamples for the currently bound graphics pipeline **must** be equal to the sample count used to create VkRenderingInfo::pStencilAttachment->imageView
- VUID-vkCmdDrawClusterIndirectHUAWEI-pNext-07935
  If this command has been called inside a render pass instance started with vkCmdBeginRenderingKHR, and the pNext chain of VkRenderingInfo includes a VkMultisampledRenderToSingleSampledInfoEXT structure with multisampledRenderToSingleSampledEnable equal to VK_TRUE, then the value of rasterizationSamples for the currently bound graphics pipeline **must** be equal to VkMultisampledRenderToSingleSampledInfoEXT::rasterizationSamples
- VUID-vkCmdDrawClusterIndirectHUAWEI-renderPass-06198
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the currently bound pipeline **must** have been created with a VkGraphicsPipelineCreateInfo::renderPass equal to VK_NULL_HANDLE
- VUID-vkCmdDrawClusterIndirectHUAWEI-pColorAttachments-08963
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is a graphics pipeline bound with a fragment shader that statically writes to a color attachment, the color write mask is not zero, color writes are enabled, and the corresponding element of the VkRenderingInfo::pColorAttachments->imageView was not

VK_NULL_HANDLE, then the corresponding element of VkPipelineRenderingCreateInfo::pColorAttachmentFormats used to create the pipeline **must** not be VK_FORMAT_UNDEFINED

- VUID-vkCmdDrawClusterIndirectHUAWEI-pDepthAttachment-08964
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is a graphics pipeline bound, depth test is enabled, depth write is enabled, and the VkRenderingInfo::pDepthAttachment->imageView was not VK_NULL_HANDLE, then the VkPipelineRenderingCreateInfo::depthAttachmentFormat used to create the pipeline **must** not be VK_FORMAT_UNDEFINED

- VUID-vkCmdDrawClusterIndirectHUAWEI-pStencilAttachment-08965
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is a graphics pipeline bound, stencil test is enabled and the VkRenderingInfo::pStencilAttachment->imageView was not VK_NULL_HANDLE, then the VkPipelineRenderingCreateInfo::stencilAttachmentFormat used to create the pipeline **must** not be VK_FORMAT_UNDEFINED

- VUID-vkCmdDrawClusterIndirectHUAWEI-primitivesGeneratedQueryWithRasterizerDiscard-06708
  If the primitivesGeneratedQueryWithRasterizerDiscard feature is not enabled and the VK_QUERY_TYPE_PRIMITIVES_GENERATED_EXT query is active, rasterization discard **must** not be enabled

- VUID-vkCmdDrawClusterIndirectHUAWEI-primitivesGeneratedQueryWithNonZeroStreams-06709
  If the primitivesGeneratedQueryWithNonZeroStreams feature is not enabled and the VK_QUERY_TYPE_PRIMITIVES_GENERATED_EXT query is active, the bound graphics pipeline **must** not have been created with a non-zero value in VkPipelineRasterizationStateStreamCreateInfoEXT::rasterizationStream

- VUID-vkCmdDrawClusterIndirectHUAWEI-None-07619
  If a shader object is bound to the VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_TESSELLATION_DOMAIN_ORIGIN_EXT dynamic state enabled, then vkCmdSetTessellationDomainOriginEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawClusterIndirectHUAWEI-None-07620
  If the depthClamp feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_CLAMP_ENABLE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetDepthClampEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawClusterIndirectHUAWEI-None-07621
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_POLYGON_MODE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetPolygonModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawClusterIndirectHUAWEI-None-07622

If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetRasterizationSamplesEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawClusterIndirectHUAWEI-None-07623
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_SAMPLE_MASK_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetSampleMaskEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawClusterIndirectHUAWEI-alphaToCoverageEnable-08919
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_ALPHA_TO_COVERAGE_ENABLE_EXT dynamic state enabled, and alphaToCoverageEnable was VK_TRUE in the last call to vkCmdSetAlphaToCoverageEnableEXT, then the Fragment Output Interface **must** contain a variable for the alpha Component word in Location 0 at Index 0

- VUID-vkCmdDrawClusterIndirectHUAWEI-alphaToCoverageEnable-08920
  If a shader object is bound to any graphics stage, and the most recent call to vkCmdSetAlphaToCoverageEnableEXT in the current command buffer set alphaToCoverageEnable to VK_TRUE, then the Fragment Output Interface **must** contain a variable for the alpha Component word in Location 0 at Index 0

- VUID-vkCmdDrawClusterIndirectHUAWEI-None-07624
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_ALPHA_TO_COVERAGE_ENABLE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetAlphaToCoverageEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawClusterIndirectHUAWEI-None-07625
  If the alphaToOne feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_ALPHA_TO_ONE_ENABLE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetAlphaToOneEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawClusterIndirectHUAWEI-None-07626
  If the logicOp feature is enabled, a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_LOGIC_OP_ENABLE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetLogicOpEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawClusterIndirectHUAWEI-None-07627
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT dynamic state enabled then

vkCmdSetColorBlendEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawClusterIndirectHUAWEI-None-08657
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and both the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE and there are color attachments bound, then vkCmdSetColorBlendEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-07628
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_EQUATION_EXT dynamic state enabled then vkCmdSetColorBlendEquationEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-08658
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetColorBlendEnableEXT for any attachment set that attachment's value in pColorBlendEnables to VK_TRUE, then vkCmdSetColorBlendEquationEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-07629
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_WRITE_MASK_EXT dynamic state enabled then vkCmdSetColorWriteMaskEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-08659
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and both the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE and there are color attachments bound, then vkCmdSetColorWriteMaskEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-07630
  If the geometryStreams feature is enabled, and a shader object is bound to the VK_SHADER_STAGE_GEOMETRY_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_RASTERIZATION_STREAM_EXT dynamic state enabled, then vkCmdSetRasterizationStreamEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-07631
  If the VK_EXT_conservative_rasterization extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_CONSERVATIVE_RASTERIZATION_MODE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetConservativeRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawClusterIndirectHUAWEI-None-07632
  If the VK_EXT_conservative_rasterization extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_EXTRA_PRIMITIVE_OVERESTIMATION_SIZE_EXT dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of conservativeRasterizationMode is VK_CONSERVATIVE_RASTERIZATION_MODE_OVERESTIMATE_EXT, then vkCmdSetExtraPrimitiveOverestimationSizeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-07633
  If the depthClipEnable feature is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_CLIP_ENABLE_EXT dynamic state, then vkCmdSetDepthClipEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-07634
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT dynamic state enabled then vkCmdSetSampleLocationsEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-08664
  If the VK_EXT_sample_locations extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetSampleLocationsEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-07635
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_ADVANCED_EXT dynamic state enabled then vkCmdSetColorBlendAdvancedEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterIndirectHUAWEI-rasterizerDiscardEnable-09416
  If the VK_EXT_blend_operation_advanced extension is enabled, and a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then at least one of vkCmdSetColorBlendEquationEXT and vkCmdSetColorBlendAdvancedEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-07636
  If the VK_EXT_provoking_vertex extension is enabled, a shader object is bound to the VK_SHADER_STAGE_VERTEX_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_PROVOKING_VERTEX_MODE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetProvokingVertexModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawClusterIndirectHUAWEI-None-07637
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT` dynamic state enabled then vkCmdSetLineRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-08666
  If the `VK_KHR_line_rasterization` or `VK_EXT_line_rasterization` extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE`, and the most recent call to vkCmdSetPolygonModeEXT in the current command buffer set `polygonMode` to `VK_POLYGON_MODE_LINE`, then vkCmdSetLineRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-08667
  If the `VK_KHR_line_rasterization` or `VK_EXT_line_rasterization` extension is enabled, and a shader object is bound to the `VK_SHADER_STAGE_VERTEX_BIT` stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE`, and the most recent call to vkCmdSetPrimitiveTopologyEXT in the current command buffer set `primitiveTopology` to any line topology, then vkCmdSetLineRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-08668
  If the `VK_KHR_line_rasterization` or `VK_EXT_line_rasterization` extension is enabled, and a shader object that outputs line primitives is bound to the `VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT` or `VK_SHADER_STAGE_GEOMETRY_BIT` stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE`, then vkCmdSetLineRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-07638
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT` dynamic state enabled then vkCmdSetLineStippleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-08669
  If the `VK_KHR_line_rasterization` or `VK_EXT_line_rasterization` extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE`, and the most recent call to vkCmdSetPolygonModeEXT in the current command buffer set `polygonMode` to `VK_POLYGON_MODE_LINE`, then vkCmdSetLineStippleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawClusterIndirectHUAWEI-None-08670
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object is bound to the VK_SHADER_STAGE_VERTEX_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetPrimitiveTopologyEXT in the current command buffer set primitiveTopology to any line topology, then vkCmdSetLineStippleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-08671
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object that outputs line primitives is bound to the VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT or VK_SHADER_STAGE_GEOMETRY_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetLineStippleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-07849
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_KHR dynamic state enabled then vkCmdSetLineStippleKHR **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-08672
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetLineStippleEnableEXT in the current command buffer set stippledLineEnable to VK_TRUE, then vkCmdSetLineStippleEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-07639
  If the depthClipControl feature is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_CLIP_NEGATIVE_ONE_TO_ONE_EXT dynamic state enabled, then vkCmdSetDepthClipNegativeOneToOneEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-07640
  If the VK_NV_clip_space_w_scaling extension is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_VIEWPORT_W_SCALING_ENABLE_NV dynamic state enabled, then vkCmdSetViewportWScalingEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-07641
  If the VK_NV_viewport_swizzle extension is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_VIEWPORT_SWIZZLE_NV dynamic state enabled, then

vkCmdSetViewportSwizzleNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawClusterIndirectHUAWEI-None-07642
  If the VK_NV_fragment_coverage_to_color extension is enabled, a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_TO_COLOR_ENABLE_NV dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetCoverageToColorEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-07643
  If the VK_NV_fragment_coverage_to_color extension is enabled, a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_TO_COLOR_LOCATION_NV dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of coverageToColorEnable is VK_TRUE, then vkCmdSetCoverageToColorLocationNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-07644
  If the VK_NV_framebuffer_mixed_samples extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_MODULATION_MODE_NV dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetCoverageModulationModeNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-07645
  If the VK_NV_framebuffer_mixed_samples extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_MODULATION_TABLE_ENABLE_NV dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of coverageModulationMode is any value other than VK_COVERAGE_MODULATION_MODE_NONE_NV, then vkCmdSetCoverageModulationTableEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-07646
  If the VK_NV_framebuffer_mixed_samples extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_MODULATION_TABLE_NV dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of coverageModulationTableEnable is VK_TRUE, then vkCmdSetCoverageModulationTableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-07647
  If the shadingRateImage feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_SHADING_RATE_IMAGE_ENABLE_NV dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetShadingRateImageEnableNV

**must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawClusterIndirectHUAWEI-pipelineFragmentShadingRate-09238
  If the `pipelineFragmentShadingRate` feature is enabled, a shader object is bound to the `VK_SHADER_STAGE_FRAGMENT_BIT` stage or a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR` dynamic state enabled, and the current value of `rasterizerDiscardEnable` is `VK_FALSE`, then vkCmdSetFragmentShadingRateKHR **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawClusterIndirectHUAWEI-None-07648
  If the `representativeFragmentTest` feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_REPRESENTATIVE_FRAGMENT_TEST_ENABLE_NV` dynamic state enabled, and the current value of `rasterizerDiscardEnable` is `VK_FALSE`, then vkCmdSetRepresentativeFragmentTestEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawClusterIndirectHUAWEI-None-07649
  If the `coverageReductionMode` feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_COVERAGE_REDUCTION_MODE_NV` dynamic state enabled, and the current value of `rasterizerDiscardEnable` is `VK_FALSE`, then vkCmdSetCoverageReductionModeNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawClusterIndirectHUAWEI-pColorBlendEnables-07470
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT` state enabled and the last call to vkCmdSetColorBlendEnableEXT set `pColorBlendEnables` for any attachment to `VK_TRUE`, then for those attachments in the subpass the corresponding image view's format features **must** contain `VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT`

- VUID-vkCmdDrawClusterIndirectHUAWEI-rasterizationSamples-07471
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT` state enabled, and the current subpass does not use any color and/or depth/stencil attachments, then the `rasterizationSamples` in the last call to vkCmdSetRasterizationSamplesEXT **must** follow the rules for a zero-attachment subpass

- VUID-vkCmdDrawClusterIndirectHUAWEI-samples-07472
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_SAMPLE_MASK_EXT` state enabled and the `VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT` state disabled, then the `samples` parameter in the last call to vkCmdSetSampleMaskEXT **must** be greater or equal to the VkPipelineMultisampleStateCreateInfo::`rasterizationSamples` parameter used to create the bound graphics pipeline

- VUID-vkCmdDrawClusterIndirectHUAWEI-samples-07473
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_SAMPLE_MASK_EXT` state and `VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT`

states enabled, then the samples parameter in the last call to vkCmdSetSampleMaskEXT **must** be greater or equal to the rasterizationSamples parameter in the last call to vkCmdSetRasterizationSamplesEXT

- VUID-vkCmdDrawClusterIndirectHUAWEI-rasterizationSamples-07474
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, and neither the VK_AMD_mixed_attachment_samples nor the VK_NV_framebuffer_mixed_samples extensions are enabled, then the rasterizationSamples in the last call to vkCmdSetRasterizationSamplesEXT **must** be the same as the current subpass color and/or depth/stencil attachments

- VUID-vkCmdDrawClusterIndirectHUAWEI-None-09211
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, or a shader object is bound to any graphics stage, and the current render pass instance includes a VkMultisampledRenderToSingleSampledInfoEXT structure with multisampledRenderToSingleSampledEnable equal to VK_TRUE, then the rasterizationSamples in the last call to vkCmdSetRasterizationSamplesEXT **must** be the same as the rasterizationSamples member of that structure

- VUID-vkCmdDrawClusterIndirectHUAWEI-firstAttachment-07476
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT dynamic state enabled then vkCmdSetColorBlendEnableEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the firstAttachment and attachmentCount parameters of vkCmdSetColorBlendEnableEXT calls **must** specify an enable for all active color attachments in the current subpass

- VUID-vkCmdDrawClusterIndirectHUAWEI-rasterizerDiscardEnable-09417
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetColorBlendEnableEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the firstAttachment and attachmentCount parameters of vkCmdSetColorBlendEnableEXT calls **must** specify an enable for all active color attachments in the current subpass

- VUID-vkCmdDrawClusterIndirectHUAWEI-firstAttachment-07477
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_EQUATION_EXT dynamic state enabled then vkCmdSetColorBlendEquationEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the firstAttachment and attachmentCount parameters of vkCmdSetColorBlendEquationEXT calls **must** specify the blend equations for all active color attachments in the current subpass where blending is enabled

- VUID-vkCmdDrawClusterIndirectHUAWEI-rasterizerDiscardEnable-09418
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and both the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE and there are color attachments bound, then

vkCmdSetColorBlendEquationEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the `firstAttachment` and `attachmentCount` parameters of `vkCmdSetColorBlendEquationEXT` calls **must** specify the blend equations for all active color attachments in the current subpass where blending is enabled

- VUID-vkCmdDrawClusterIndirectHUAWEI-firstAttachment-07478
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_COLOR_WRITE_MASK_EXT` dynamic state enabled then vkCmdSetColorWriteMaskEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the `firstAttachment` and `attachmentCount` parameters of `vkCmdSetColorWriteMaskEXT` calls **must** specify the color write mask for all active color attachments in the current subpass

- VUID-vkCmdDrawClusterIndirectHUAWEI-rasterizerDiscardEnable-09419
  If a shader object is bound to the `VK_SHADER_STAGE_FRAGMENT_BIT` stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE`, then vkCmdSetColorWriteMaskEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the `firstAttachment` and `attachmentCount` parameters of `vkCmdSetColorWriteMaskEXT` calls **must** specify the color write mask for all active color attachments in the current subpass

- VUID-vkCmdDrawClusterIndirectHUAWEI-firstAttachment-07479
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_COLOR_BLEND_ADVANCED_EXT` dynamic state enabled then vkCmdSetColorBlendAdvancedEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the `firstAttachment` and `attachmentCount` parameters of `vkCmdSetColorBlendAdvancedEXT` calls **must** specify the advanced blend equations for all active color attachments in the current subpass where blending is enabled

- VUID-vkCmdDrawClusterIndirectHUAWEI-advancedBlendMaxColorAttachments-07480
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_COLOR_BLEND_ADVANCED_EXT` and `VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT` dynamic states enabled and the last calls to vkCmdSetColorBlendEnableEXT and vkCmdSetColorBlendAdvancedEXT have enabled advanced blending, then the number of active color attachments in the current subpass **must** not exceed `advancedBlendMaxColorAttachments`

- VUID-vkCmdDrawClusterIndirectHUAWEI-primitivesGeneratedQueryWithNonZeroStreams-07481
  If the `primitivesGeneratedQueryWithNonZeroStreams` feature is not enabled and the `VK_QUERY_TYPE_PRIMITIVES_GENERATED_EXT` query is active, and the bound graphics pipeline was created with `VK_DYNAMIC_STATE_RASTERIZATION_STREAM_EXT` state enabled, the last call to vkCmdSetRasterizationStreamEXT **must** have set the `rasterizationStream` to zero

- VUID-vkCmdDrawClusterIndirectHUAWEI-sampleLocationsPerPixel-07482
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT` state enabled and the `VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT` state disabled, then the

sampleLocationsPerPixel member of pSampleLocationsInfo in the last call to vkCmdSetSampleLocationsEXT **must** equal the rasterizationSamples member of the VkPipelineMultisampleStateCreateInfo structure the bound graphics pipeline has been created with

- VUID-vkCmdDrawClusterIndirectHUAWEI-sampleLocationsPerPixel-07483
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state enabled and the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, then the sampleLocationsPerPixel member of pSampleLocationsInfo in the last call to vkCmdSetSampleLocationsEXT **must** equal the rasterizationSamples parameter of the last call to vkCmdSetRasterizationSamplesEXT

- VUID-vkCmdDrawClusterIndirectHUAWEI-sampleLocationsEnable-07484
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, or the bound graphics pipeline was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, and sampleLocationsEnable was VK_TRUE in the last call to vkCmdSetSampleLocationsEnableEXT, and the current subpass has a depth/stencil attachment, then that attachment **must** have been created with the VK_IMAGE_CREATE_SAMPLE_LOCATIONS_COMPATIBLE_DEPTH_BIT_EXT bit set

- VUID-vkCmdDrawClusterIndirectHUAWEI-sampleLocationsEnable-07485
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state enabled and the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, and if sampleLocationsEnable was VK_TRUE in the last call to vkCmdSetSampleLocationsEnableEXT, then the sampleLocationsInfo.sampleLocationGridSize.width in the last call to vkCmdSetSampleLocationsEXT **must** evenly divide VkMultisamplePropertiesEXT::sampleLocationGridSize.width as returned by vkGetPhysicalDeviceMultisamplePropertiesEXT with a samples parameter equaling rasterizationSamples

- VUID-vkCmdDrawClusterIndirectHUAWEI-sampleLocationsEnable-07486
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state enabled and the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, and if sampleLocationsEnable was VK_TRUE in the last call to vkCmdSetSampleLocationsEnableEXT, then the sampleLocationsInfo.sampleLocationGridSize.height in the last call to vkCmdSetSampleLocationsEXT **must** evenly divide VkMultisamplePropertiesEXT::sampleLocationGridSize.height as returned by vkGetPhysicalDeviceMultisamplePropertiesEXT with a samples parameter equaling rasterizationSamples

- VUID-vkCmdDrawClusterIndirectHUAWEI-sampleLocationsEnable-07487
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, and if sampleLocationsEnable was VK_TRUE in the last call to vkCmdSetSampleLocationsEnableEXT, the fragment shader code **must** not statically use the extended instruction InterpolateAtSample

- VUID-vkCmdDrawClusterIndirectHUAWEI-sampleLocationsEnable-07936
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT` state disabled and the `VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT` state enabled, the `sampleLocationsEnable` member of a VkPipelineSampleLocationsStateCreateInfoEXT::`sampleLocationsEnable` in the bound graphics pipeline is `VK_TRUE` or `VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT` state enabled, then, `sampleLocationsInfo.sampleLocationGridSize.width` **must** evenly divide VkMultisamplePropertiesEXT::`sampleLocationGridSize.width` as returned by vkGetPhysicalDeviceMultisamplePropertiesEXT with a `samples` parameter equaling the value of `rasterizationSamples` in the last call to vkCmdSetRasterizationSamplesEXT
- VUID-vkCmdDrawClusterIndirectHUAWEI-sampleLocationsEnable-07937
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT` state disabled and the `VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT` state enabled, the `sampleLocationsEnable` member of a VkPipelineSampleLocationsStateCreateInfoEXT::`sampleLocationsEnable` in the bound graphics pipeline is `VK_TRUE` or `VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT` state enabled, then, `sampleLocationsInfo.sampleLocationGridSize.height` **must** evenly divide VkMultisamplePropertiesEXT::`sampleLocationGridSize.height` as returned by vkGetPhysicalDeviceMultisamplePropertiesEXT with a `samples` parameter equaling the value of `rasterizationSamples` in the last call to vkCmdSetRasterizationSamplesEXT
- VUID-vkCmdDrawClusterIndirectHUAWEI-sampleLocationsEnable-07938
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT` state disabled and the `VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT` state enabled, the `sampleLocationsEnable` member of a VkPipelineSampleLocationsStateCreateInfoEXT::`sampleLocationsEnable` in the bound graphics pipeline is `VK_TRUE` or `VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT` state enabled, then, `sampleLocationsInfo.sampleLocationsPerPixel` **must** equal `rasterizationSamples` in the last call to vkCmdSetRasterizationSamplesEXT
- VUID-vkCmdDrawClusterIndirectHUAWEI-coverageModulationTableEnable-07488
  If a shader object is bound to any graphics stage or the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_COVERAGE_MODULATION_TABLE_ENABLE_NV` state enabled, and the last call to vkCmdSetCoverageModulationTableEnableNV set `coverageModulationTableEnable` to `VK_TRUE`, then the `coverageModulationTableCount` parameter in the last call to vkCmdSetCoverageModulationTableNV **must** equal the current `rasterizationSamples` divided by the number of color samples in the current subpass
- VUID-vkCmdDrawClusterIndirectHUAWEI-rasterizationSamples-07489
  If the `VK_NV_framebuffer_mixed_samples` extension is enabled, and if current subpass has a depth/stencil attachment and depth test, stencil test, or depth bounds test are enabled in the currently bound pipeline state, then the current `rasterizationSamples` **must** be the same as the sample count of the depth/stencil attachment
- VUID-vkCmdDrawClusterIndirectHUAWEI-coverageToColorEnable-07490
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_COVERAGE_TO_COLOR_ENABLE_NV` state enabled and the last call to vkCmdSetCoverageToColorEnableNV set the `coverageToColorEnable` to `VK_TRUE`, then the current subpass **must** have a color attachment at the location selected by the last call to

vkCmdSetCoverageToColorLocationNV coverageToColorLocation, with a VkFormat of VK_FORMAT_R8_UINT, VK_FORMAT_R8_SINT, VK_FORMAT_R16_UINT, VK_FORMAT_R16_SINT, VK_FORMAT_R32_UINT, or VK_FORMAT_R32_SINT

- VUID-vkCmdDrawClusterIndirectHUAWEI-rasterizerDiscardEnable-09420
  If the VK_NV_fragment_coverage_to_color extension is enabled, and a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the last call to vkCmdSetCoverageToColorEnableNV set the coverageToColorEnable to VK_TRUE, then the current subpass **must** have a color attachment at the location selected by the last call to vkCmdSetCoverageToColorLocationNV coverageToColorLocation, with a VkFormat of VK_FORMAT_R8_UINT, VK_FORMAT_R8_SINT, VK_FORMAT_R16_UINT, VK_FORMAT_R16_SINT, VK_FORMAT_R32_UINT, or VK_FORMAT_R32_SINT
- VUID-vkCmdDrawClusterIndirectHUAWEI-coverageReductionMode-07491
  If this VK_NV_coverage_reduction_mode extension is enabled, the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COVERAGE_TO_COLOR_ENABLE_NV and VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT states enabled, the current coverage reduction mode coverageReductionMode, then the current rasterizationSamples, and the sample counts for the color and depth/stencil attachments (if the subpass has them) **must** be a valid combination returned by vkGetPhysicalDeviceSupportedFramebufferMixedSamplesCombinationsNV
- VUID-vkCmdDrawClusterIndirectHUAWEI-viewportCount-07492
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, but not the VK_DYNAMIC_STATE_VIEWPORT_SWIZZLE_NV dynamic state enabled, then the bound graphics pipeline **must** have been created with VkPipelineViewportSwizzleStateCreateInfoNV ::viewportCount greater or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawClusterIndirectHUAWEI-viewportCount-07493
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT and VK_DYNAMIC_STATE_VIEWPORT_SWIZZLE_NV dynamic states enabled then the viewportCount parameter in the last call to vkCmdSetViewportSwizzleNV **must** be greater than or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawClusterIndirectHUAWEI-viewportCount-09421
  If the VK_NV_viewport_swizzle extension is enabled, and a shader object is bound to any graphics stage, then the viewportCount parameter in the last call to vkCmdSetViewportSwizzleNV **must** be greater than or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdDrawClusterIndirectHUAWEI-rasterizationSamples-07494
  If the VK_NV_framebuffer_mixed_samples extension is enabled, and if the current subpass has any color attachments and rasterizationSamples of the last call to vkCmdSetRasterizationSamplesEXT is greater than the number of color samples, then the pipeline sampleShadingEnable **must** be VK_FALSE
- VUID-vkCmdDrawClusterIndirectHUAWEI-stippledLineEnable-07495

If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT or VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT dynamic states enabled, and if the current stippledLineEnable state is VK_TRUE and the current lineRasterizationMode state is VK_LINE_RASTERIZATION_MODE_RECTANGULAR_KHR, then the stippledRectangularLines feature **must** be enabled

- VUID-vkCmdDrawClusterIndirectHUAWEI-stippledLineEnable-07496
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT or VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT dynamic states enabled, and if the current stippledLineEnable state is VK_TRUE and the current lineRasterizationMode state is VK_LINE_RASTERIZATION_MODE_BRESENHAM_KHR, then the stippledBresenhamLines feature **must** be enabled

- VUID-vkCmdDrawClusterIndirectHUAWEI-stippledLineEnable-07497
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT or VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT dynamic states enabled, and if the current stippledLineEnable state is VK_TRUE and the current lineRasterizationMode state is VK_LINE_RASTERIZATION_MODE_RECTANGULAR_SMOOTH_KHR, then the stippledSmoothLines feature **must** be enabled

- VUID-vkCmdDrawClusterIndirectHUAWEI-stippledLineEnable-07498
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT or VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT dynamic states enabled, and if the current stippledLineEnable state is VK_TRUE and the current lineRasterizationMode state is VK_LINE_RASTERIZATION_MODE_DEFAULT_KHR, then the stippledRectangularLines feature **must** be enabled and VkPhysicalDeviceLimits::strictLines **must** be VK_TRUE

- VUID-vkCmdDrawClusterIndirectHUAWEI-conservativePointAndLineRasterization-07499
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_CONSERVATIVE_RASTERIZATION_MODE_EXT dynamic state enabled, conservativePointAndLineRasterization is not supported, and the effective primitive topology output by the last pre-rasterization shader stage is a line or point, then the conservativeRasterizationMode set by the last call to vkCmdSetConservativeRasterizationModeEXT **must** be VK_CONSERVATIVE_RASTERIZATION_MODE_DISABLED_EXT

- VUID-vkCmdDrawClusterIndirectHUAWEI-stage-07073
  If the currently bound pipeline was created with the VkPipelineShaderStageCreateInfo::stage member of an element of VkGraphicsPipelineCreateInfo::pStages set to VK_SHADER_STAGE_VERTEX_BIT, VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT, VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT or VK_SHADER_STAGE_GEOMETRY_BIT, then Mesh Shader Queries **must** not be active

- VUID-vkCmdDrawClusterIndirectHUAWEI-None-08877
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_ATTACHMENT_FEEDBACK_LOOP_ENABLE_EXT dynamic state enabled, and the

current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetAttachmentFeedbackLoopEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawClusterIndirectHUAWEI-None-07850
  If dynamic state was inherited from VkCommandBufferInheritanceViewportScissorInfoNV, it **must** be set in the current command buffer prior to this drawing command
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-08684
  If there is no bound graphics pipeline, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_VERTEX_BIT
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-08685
  If there is no bound graphics pipeline, and the tessellationShader feature is enabled, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-08686
  If there is no bound graphics pipeline, and the tessellationShader feature is enabled, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-08687
  If there is no bound graphics pipeline, and the geometryShader feature is enabled, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_GEOMETRY_BIT
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-08688
  If there is no bound graphics pipeline, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_FRAGMENT_BIT
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-08689
  If there is no bound graphics pipeline, and the taskShader feature is enabled, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_TASK_BIT_EXT
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-08690
  If there is no bound graphics pipeline, and the meshShader feature is enabled, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_MESH_BIT_EXT
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-08693
  If there is no bound graphics pipeline, and at least one of the taskShader and meshShader features is enabled, one of the VK_SHADER_STAGE_VERTEX_BIT or VK_SHADER_STAGE_MESH_BIT_EXT stages **must** have a valid VkShaderEXT bound, and the other **must** have no VkShaderEXT bound
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-08694
  If there is no bound graphics pipeline, and both the taskShader and meshShader features are enabled, and a valid VkShaderEXT is bound the to the VK_SHADER_STAGE_MESH_BIT_EXT stage, and that VkShaderEXT was created without the VK_SHADER_CREATE_NO_TASK_SHADER_BIT_EXT flag, a valid VkShaderEXT **must** be bound to the VK_SHADER_STAGE_TASK_BIT_EXT stage

- VUID-vkCmdDrawClusterIndirectHUAWEI-None-08695
  If there is no bound graphics pipeline, and both the `taskShader` and `meshShader` features are enabled, and a valid `VkShaderEXT` is bound the to the `VK_SHADER_STAGE_MESH_BIT_EXT` stage, and that `VkShaderEXT` was created with the `VK_SHADER_CREATE_NO_TASK_SHADER_BIT_EXT` flag, there **must** be no `VkShaderEXT` bound to the `VK_SHADER_STAGE_TASK_BIT_EXT` stage
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-08696
  If there is no bound graphics pipeline, and a valid `VkShaderEXT` is bound to the `VK_SHADER_STAGE_VERTEX_BIT` stage, there **must** be no `VkShaderEXT` bound to either the `VK_SHADER_STAGE_TASK_BIT_EXT` stage or the `VK_SHADER_STAGE_MESH_BIT_EXT` stage
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-08698
  If any graphics shader is bound which was created with the `VK_SHADER_CREATE_LINK_STAGE_BIT_EXT` flag, then all shaders created with the `VK_SHADER_CREATE_LINK_STAGE_BIT_EXT` flag in the same vkCreateShadersEXT call **must** also be bound
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-08699
  If any graphics shader is bound which was created with the `VK_SHADER_CREATE_LINK_STAGE_BIT_EXT` flag, any stages in between stages whose shaders which did not create a shader with the `VK_SHADER_CREATE_LINK_STAGE_BIT_EXT` flag as part of the same vkCreateShadersEXT call **must** not have any `VkShaderEXT` bound
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-08878
  All bound graphics shader objects **must** have been created with identical or identically defined push constant ranges
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-08879
  All bound graphics shader objects **must** have been created with identical or identically defined arrays of descriptor set layouts
- VUID-vkCmdDrawClusterIndirectHUAWEI-colorAttachmentCount-09372
  If the current render pass instance was begun with vkCmdBeginRenderingKHR and a VkRenderingInfo::`colorAttachmentCount` equal to `1`, a color attachment with a resolve mode of `VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID`, and a fragment shader is bound, it **must** not declare the `DepthReplacing` or `StencilRefReplacingEXT` execution modes
- VUID-vkCmdDrawClusterIndirectHUAWEI-pDynamicStates-08715
  If the bound graphics pipeline state includes a fragment shader stage, was created with `VK_DYNAMIC_STATE_DEPTH_WRITE_ENABLE` set in VkPipelineDynamicStateCreateInfo::`pDynamicStates`, and the fragment shader declares the `EarlyFragmentTests` execution mode and uses `OpDepthAttachmentReadEXT`, the `depthWriteEnable` parameter in the last call to vkCmdSetDepthWriteEnableEXT **must** be `VK_FALSE`
- VUID-vkCmdDrawClusterIndirectHUAWEI-pDynamicStates-08716
  If the bound graphics pipeline state includes a fragment shader stage, was created with `VK_DYNAMIC_STATE_STENCIL_WRITE_MASK` set in VkPipelineDynamicStateCreateInfo::`pDynamicStates`, and the fragment shader declares the `EarlyFragmentTests` execution mode and uses `OpStencilAttachmentReadEXT`, the `writeMask` parameter in the last call to vkCmdSetStencilWriteMask **must** be `0`
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-09116
  If a shader object is bound to any graphics stage or the currently bound graphics pipeline

was created with VK_DYNAMIC_STATE_COLOR_WRITE_MASK_EXT, and the format of any color attachment is VK_FORMAT_E5B9G9R9_UFLOAT_PACK32, the corresponding element of the pColorWriteMasks parameter of vkCmdSetColorWriteMaskEXT **must** either include all of VK_COLOR_COMPONENT_R_BIT, VK_COLOR_COMPONENT_G_BIT, and VK_COLOR_COMPONENT_B_BIT, or none of them

- VUID-vkCmdDrawClusterIndirectHUAWEI-maxFragmentDualSrcAttachments-09239
  If blending is enabled for any attachment where either the source or destination blend factors for that attachment use the secondary color input, the maximum value of Location for any output attachment statically used in the Fragment Execution Model executed by this command **must** be less than maxFragmentDualSrcAttachments
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-09548
  If the current render pass was begun with vkCmdBeginRenderingKHR, and there is no shader object bound to any graphics stage, the value of each element of VkRenderingAttachmentLocationInfoKHR::pColorAttachmentLocations set by vkCmdSetRenderingAttachmentLocationsKHR **must** match the value set for the corresponding element in the currently bound pipeline
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-09549
  If the current render pass was begun with vkCmdBeginRenderingKHR, and there is no shader object bound to any graphics stage, input attachment index mappings in the currently bound pipeline **must** match those set for the current render pass instance via VkRenderingInputAttachmentIndexInfoKHR
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-09642
  If the current render pass was begun with vkCmdBeginRenderingKHR with the VK_RENDERING_ENABLE_LEGACY_DITHERING_BIT_EXT flag, the bound graphics pipeline **must** have been created with VK_PIPELINE_CREATE_2_ENABLE_LEGACY_DITHERING_BIT_EXT
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-09643
  If the bound graphics pipeline was created with VK_PIPELINE_CREATE_2_ENABLE_LEGACY_DITHERING_BIT_EXT, the current render pass **must** have begun with vkCmdBeginRenderingKHR with the VK_RENDERING_ENABLE_LEGACY_DITHERING_BIT_EXT flag
- VUID-vkCmdDrawClusterIndirectHUAWEI-stage-06480
  The bound graphics pipeline **must** not have been created with the VkPipelineShaderStageCreateInfo::stage member of an element of VkGraphicsPipelineCreateInfo::pStages set to VK_SHADER_STAGE_VERTEX_BIT, VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT, VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT or VK_SHADER_STAGE_GEOMETRY_BIT
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-07074
  Transform Feedback Queries **must** not be active
- VUID-vkCmdDrawClusterIndirectHUAWEI-None-07075
  Primitives Generated Queries **must** not be active
- VUID-vkCmdDrawClusterIndirectHUAWEI-pipelineStatistics-07076
  The pipelineStatistics member used to create any active Pipeline Statistics Query **must** not contain VK_QUERY_PIPELINE_STATISTIC_INPUT_ASSEMBLY_VERTICES_BIT, VK_QUERY_PIPELINE_STATISTIC_INPUT_ASSEMBLY_PRIMITIVES_BIT,

VK_QUERY_PIPELINE_STATISTIC_VERTEX_SHADER_INVOCATIONS_BIT, VK_QUERY_PIPELINE_STATISTIC_GEOMETRY_SHADER_INVOCATIONS_BIT, VK_QUERY_PIPELINE_STATISTIC_GEOMETRY_SHADER_PRIMITIVES_BIT, VK_QUERY_PIPELINE_STATISTIC_CLIPPING_INVOCATIONS_BIT, VK_QUERY_PIPELINE_STATISTIC_CLIPPING_PRIMITIVES_BIT, VK_QUERY_PIPELINE_STATISTIC_TESSELLATION_CONTROL_SHADER_PATCHES_BIT, or VK_QUERY_PIPELINE_STATISTIC_TESSELLATION_EVALUATION_SHADER_INVOCATIONS_BIT

- VUID-vkCmdDrawClusterIndirectHUAWEI-drawCount-02718
  If the multiDrawIndirect feature is not enabled, drawCount **must** be 0 or 1
- VUID-vkCmdDrawClusterIndirectHUAWEI-drawCount-02719
  drawCount **must** be less than or equal to VkPhysicalDeviceLimits::maxDrawIndirectCount
- VUID-vkCmdDrawClusterIndirectHUAWEI-ClusterCullingHUAWEI-07824
  The current pipeline bound to VK_PIPELINE_BIND_POINT_GRAPHICS **must** contain a shader stage using the ClusterCullingHUAWEI Execution Model.
- VUID-vkCmdDrawClusterIndirectHUAWEI-offset-07918
  offset **must** be a multiple of VkPhysicalDeviceClusterCullingShaderPropertiesHUAWEI::indirectBufferOffsetAlignment

## Valid Usage (Implicit)

- VUID-vkCmdDrawClusterIndirectHUAWEI-commandBuffer-parameter
  commandBuffer **must** be a valid VkCommandBuffer handle
- VUID-vkCmdDrawClusterIndirectHUAWEI-buffer-parameter
  buffer **must** be a valid VkBuffer handle
- VUID-vkCmdDrawClusterIndirectHUAWEI-commandBuffer-recording
  commandBuffer **must** be in the recording state
- VUID-vkCmdDrawClusterIndirectHUAWEI-commandBuffer-cmdpool
  The VkCommandPool that commandBuffer was allocated from **must** support graphics operations
- VUID-vkCmdDrawClusterIndirectHUAWEI-renderpass
  This command **must** only be called inside of a render pass instance
- VUID-vkCmdDrawClusterIndirectHUAWEI-videocoding
  This command **must** only be called outside of a video coding scope
- VUID-vkCmdDrawClusterIndirectHUAWEI-commonparent
  Both of buffer, and commandBuffer **must** have been created, allocated, or retrieved from the same VkDevice

## Host Synchronization

- Host access to commandBuffer **must** be externally synchronized
- Host access to the VkCommandPool that commandBuffer was allocated from **must** be externally

synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary Secondary | Inside | Outside | Graphics | Action |

# Chapter 22. Fixed-Function Vertex Processing

Vertex fetching is controlled via configurable state, as a logically distinct graphics pipeline stage.

## 22.1. Vertex Attributes

Vertex shaders **can** define input variables, which receive *vertex attribute* data transferred from one or more `VkBuffer`(s) by drawing commands. Vertex shader input variables are bound to buffers via an indirect binding where the vertex shader associates a *vertex input attribute* number with each variable, vertex input attributes are associated to *vertex input bindings* on a per-pipeline basis, and vertex input bindings are associated with specific buffers on a per-draw basis via the `vkCmdBindVertexBuffers` command. Vertex input attribute and vertex input binding descriptions also contain format information controlling how data is extracted from buffer memory and converted to the format expected by the vertex shader.

There are `VkPhysicalDeviceLimits::maxVertexInputAttributes` number of vertex input attributes and `VkPhysicalDeviceLimits::maxVertexInputBindings` number of vertex input bindings (each referred to by zero-based indices), where there are at least as many vertex input attributes as there are vertex input bindings. Applications **can** store multiple vertex input attributes interleaved in a single buffer, and use a single vertex input binding to access those attributes.

In GLSL, vertex shaders associate input variables with a vertex input attribute number using the `location` layout qualifier. The `Component` layout qualifier associates components of a vertex shader input variable with components of a vertex input attribute.

*GLSL example*

```
// Assign location M to variableName
layout (location=M, component=2) in vec2 variableName;

// Assign locations [N,N+L) to the array elements of variableNameArray
layout (location=N) in vec4 variableNameArray[L];
```

In SPIR-V, vertex shaders associate input variables with a vertex input attribute number using the `Location` decoration. The `Component` decoration associates components of a vertex shader input variable with components of a vertex input attribute. The `Location` and `Component` decorations are specified via the `OpDecorate` instruction.

*SPIR-V example*

```
        ...
%1 = OpExtInstImport "GLSL.std.450"
        ...
        OpName %9 "variableName"
        OpName %15 "variableNameArray"
        OpDecorate %18 BuiltIn VertexIndex
```

```
        OpDecorate %19 BuiltIn InstanceIndex
        OpDecorate %9 Location M
        OpDecorate %9 Component 2
        OpDecorate %15 Location N
        ...
  %2 = OpTypeVoid
  %3 = OpTypeFunction %2
  %6 = OpTypeFloat 32
  %7 = OpTypeVector %6 2
  %8 = OpTypePointer Input %7
  %9 = OpVariable %8 Input
 %10 = OpTypeVector %6 4
 %11 = OpTypeInt 32 0
 %12 = OpConstant %11 L
 %13 = OpTypeArray %10 %12
 %14 = OpTypePointer Input %13
 %15 = OpVariable %14 Input
        ...
```

### 22.1.1. Attribute Location and Component Assignment

The `Location` decoration specifies which vertex input attribute is used to read and interpret the data that a variable will consume.

When a vertex shader input variable declared using a 16- or 32-bit scalar or vector data type is assigned a `Location`, its value(s) are taken from the components of the input attribute specified with the corresponding `VkVertexInputAttributeDescription::location`. The components used depend on the type of variable and the `Component` decoration specified in the variable declaration, as identified in Input attribute components accessed by 16-bit and 32-bit input variables. Any 16-bit or 32-bit scalar or vector input will consume a single `Location`. For 16-bit and 32-bit data types, missing components are filled in with default values as described below.

If an implementation supports `storageInputOutput16`, vertex shader input variables **can** have a width of 16 bits.

*Table 32. Input attribute components accessed by 16-bit and 32-bit input variables*

| 16-bit or 32-bit data type | `Component` decoration | Components consumed |
|---|---|---|
| scalar | 0 or unspecified | (x, 0, 0, 0) |
| scalar | 1 | (0, y, 0, 0) |
| scalar | 2 | (0, 0, z, 0) |
| scalar | 3 | (0, 0, 0, w) |
| two-component vector | 0 or unspecified | (x, y, 0, 0) |
| two-component vector | 1 | (0, y, z, 0) |
| two-component vector | 2 | (0, 0, z, w) |

| 16-bit or 32-bit data type | `Component` decoration | Components consumed |
|---|---|---|
| three-component vector | 0 or unspecified | (x, y, z, o) |
| three-component vector | 1 | (o, y, z, w) |
| four-component vector | 0 or unspecified | (x, y, z, w) |

Components indicated by “o” are available for use by other input variables which are sourced from the same attribute, and if used, are either filled with the corresponding component from the input format (if present), or the default value.

When a vertex shader input variable declared using a 32-bit floating-point matrix type is assigned a `Location` *i*, its values are taken from consecutive input attributes starting with the corresponding `VkVertexInputAttributeDescription`::`location`. Such matrices are treated as an array of column vectors with values taken from the input attributes identified in Input attributes accessed by 32-bit input matrix variables. The `VkVertexInputAttributeDescription`::`format` **must** be specified with a VkFormat that corresponds to the appropriate type of column vector. The `Component` decoration **must** not be used with matrix types.

*Table 33. Input attributes accessed by 32-bit input matrix variables*

| Data type | Column vector type | Locations consumed | Components consumed |
|---|---|---|---|
| mat2 | two-component vector | i, i+1 | (x, y, o, o), (x, y, o, o) |
| mat2x3 | three-component vector | i, i+1 | (x, y, z, o), (x, y, z, o) |
| mat2x4 | four-component vector | i, i+1 | (x, y, z, w), (x, y, z, w) |
| mat3x2 | two-component vector | i, i+1, i+2 | (x, y, o, o), (x, y, o, o), (x, y, o, o) |
| mat3 | three-component vector | i, i+1, i+2 | (x, y, z, o), (x, y, z, o), (x, y, z, o) |
| mat3x4 | four-component vector | i, i+1, i+2 | (x, y, z, w), (x, y, z, w), (x, y, z, w) |
| mat4x2 | two-component vector | i, i+1, i+2, i+3 | (x, y, o, o), (x, y, o, o), (x, y, o, o), (x, y, o, o) |
| mat4x3 | three-component vector | i, i+1, i+2, i+3 | (x, y, z, o), (x, y, z, o), (x, y, z, o), (x, y, z, o) |
| mat4 | four-component vector | i, i+1, i+2, i+3 | (x, y, z, w), (x, y, z, w), (x, y, z, w), (x, y, z, w) |

Components indicated by “o” are available for use by other input variables which are sourced from the same attribute, and if used, are either filled with the corresponding component from the input (if present), or the default value.

When a vertex shader input variable declared using a scalar or vector 64-bit data type is assigned a `Location` *i*, its values are taken from consecutive input attributes starting with the corresponding

VkVertexInputAttributeDescription::location. The Location slots and Component words used depend on the type of variable and the Component decoration specified in the variable declaration, as identified in Input attribute locations and components accessed by 64-bit input variables. For 64-bit data types, no default attribute values are provided. Input variables **must** not use more components than provided by the attribute.

*Table 34. Input attribute locations and components accessed by 64-bit input variables*

| Input format | Locations consumed | 64-bit data type | Location decoration | Component decoration | 32-bit components consumed |
|---|---|---|---|---|---|
| R64 | i | scalar | i | 0 or unspecified | (x, y, -, -) |
| R64G64 | i | scalar | i | 0 or unspecified | (x, y, o, o) |
| | | scalar | i | 2 | (o, o, z, w) |
| | | two-component vector | i | 0 or unspecified | (x, y, z, w) |
| R64G64B64 | i, i+1 | scalar | i | 0 or unspecified | (x, y, o, o), (o, o, -, -) |
| | | scalar | i | 2 | (o, o, z, w), (o, o, -, -) |
| | | scalar | i+1 | 0 or unspecified | (o, o, o, o), (x, y, -, -) |
| | | two-component vector | i | 0 or unspecified | (x, y, z, w), (o, o, -, -) |
| | | three-component vector | i | unspecified | (x, y, z, w), (x, y, -, -) |
| R64G64B64A64 | i, i+1 | scalar | i | 0 or unspecified | (x, y, o, o), (o, o, o, o) |
| | | scalar | i | 2 | (o, o, z, w), (o, o, o, o) |
| | | scalar | i+1 | 0 or unspecified | (o, o, o, o), (x, y, o, o) |
| | | scalar | i+1 | 2 | (o, o, o, o), (o, o, z, w) |
| | | two-component vector | i | 0 or unspecified | (x, y, z, w), (o, o, o, o) |
| | | two-component vector | i+1 | 0 or unspecified | (o, o, o, o), (x, y, z, w) |
| | | three-component vector | i | unspecified | (x, y, z, w), (x, y, o, o) |
| | | four-component vector | i | unspecified | (x, y, z, w), (x, y, z, w) |

Components indicated by “o” are available for use by other input variables which are sourced from the same attribute. Components indicated by “-” are not available for input variables as there are no default values provided for 64-bit data types, and there is no data provided by the input format.

When a vertex shader input variable declared using a 64-bit floating-point matrix type is assigned a `Location` *i*, its values are taken from consecutive input attribute locations. Such matrices are treated as an array of column vectors with values taken from the input attributes as shown in Input attribute locations and components accessed by 64-bit input variables. Each column vector starts at the `Location` immediately following the last `Location` of the previous column vector. The number of attributes and components assigned to each matrix is determined by the matrix dimensions and ranges from two to eight locations.

When a vertex shader input variable declared using an array type is assigned a location, its values are taken from consecutive input attributes starting with the corresponding `VkVertexInputAttributeDescription::location`. The number of attributes and components assigned to each element are determined according to the data type of the array elements and `Component` decoration (if any) specified in the declaration of the array, as described above. Each element of the array, in order, is assigned to consecutive locations, but all at the same specified component within each location.

Only input variables declared with the data types and component decorations as specified above are supported. Two variables are allowed to share the same `Location` slot only if their `Component` words do not overlap. If multiple variables share the same `Location` slot, they **must** all have the same SPIR-V floating-point component type or all have the same width scalar type components.

## 22.2. Vertex Input Description

Applications specify vertex input attribute and vertex input binding descriptions as part of graphics pipeline creation by setting the VkGraphicsPipelineCreateInfo::`pVertexInputState` pointer to a VkPipelineVertexInputStateCreateInfo structure. Alternatively, if the graphics pipeline is created with the `VK_DYNAMIC_STATE_VERTEX_INPUT_EXT` dynamic state enabled, then the vertex input attribute and vertex input binding descriptions are specified dynamically with vkCmdSetVertexInputEXT, and the VkGraphicsPipelineCreateInfo::`pVertexInputState` pointer is ignored.

The `VkPipelineVertexInputStateCreateInfo` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkPipelineVertexInputStateCreateInfo {
    VkStructureType                             sType;
    const void*                                 pNext;
    VkPipelineVertexInputStateCreateFlags       flags;
    uint32_t                                    vertexBindingDescriptionCount;
    const VkVertexInputBindingDescription*      pVertexBindingDescriptions;
    uint32_t                                    vertexAttributeDescriptionCount;
    const VkVertexInputAttributeDescription*    pVertexAttributeDescriptions;
} VkPipelineVertexInputStateCreateInfo;
```

- `sType` is a VkStructureType value identifying this structure.

- pNext is NULL or a pointer to a structure extending this structure.
- flags is reserved for future use.
- vertexBindingDescriptionCount is the number of vertex binding descriptions provided in pVertexBindingDescriptions.
- pVertexBindingDescriptions is a pointer to an array of VkVertexInputBindingDescription structures.
- vertexAttributeDescriptionCount is the number of vertex attribute descriptions provided in pVertexAttributeDescriptions.
- pVertexAttributeDescriptions is a pointer to an array of VkVertexInputAttributeDescription structures.

## Valid Usage

- VUID-VkPipelineVertexInputStateCreateInfo-vertexBindingDescriptionCount-00613
  vertexBindingDescriptionCount **must** be less than or equal to VkPhysicalDeviceLimits::maxVertexInputBindings
- VUID-VkPipelineVertexInputStateCreateInfo-vertexAttributeDescriptionCount-00614
  vertexAttributeDescriptionCount **must** be less than or equal to VkPhysicalDeviceLimits::maxVertexInputAttributes
- VUID-VkPipelineVertexInputStateCreateInfo-binding-00615
  For every binding specified by each element of pVertexAttributeDescriptions, a VkVertexInputBindingDescription **must** exist in pVertexBindingDescriptions with the same value of binding
- VUID-VkPipelineVertexInputStateCreateInfo-pVertexBindingDescriptions-00616
  All elements of pVertexBindingDescriptions **must** describe distinct binding numbers
- VUID-VkPipelineVertexInputStateCreateInfo-pVertexAttributeDescriptions-00617
  All elements of pVertexAttributeDescriptions **must** describe distinct attribute locations

## Valid Usage (Implicit)

- VUID-VkPipelineVertexInputStateCreateInfo-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PIPELINE_VERTEX_INPUT_STATE_CREATE_INFO
- VUID-VkPipelineVertexInputStateCreateInfo-pNext-pNext
  pNext **must** be NULL or a pointer to a valid instance of VkPipelineVertexInputDivisorStateCreateInfoKHR
- VUID-VkPipelineVertexInputStateCreateInfo-sType-unique
  The sType value of each struct in the pNext chain **must** be unique
- VUID-VkPipelineVertexInputStateCreateInfo-flags-zerobitmask
  flags **must** be 0
- VUID-VkPipelineVertexInputStateCreateInfo-pVertexBindingDescriptions-parameter
  If vertexBindingDescriptionCount is not 0, pVertexBindingDescriptions **must** be a valid

pointer to an array of vertexBindingDescriptionCount valid VkVertexInputBindingDescription structures

- VUID-VkPipelineVertexInputStateCreateInfo-pVertexAttributeDescriptions-parameter
  If vertexAttributeDescriptionCount is not 0, pVertexAttributeDescriptions **must** be a valid pointer to an array of vertexAttributeDescriptionCount valid VkVertexInputAttributeDescription structures

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkPipelineVertexInputStateCreateFlags;
```

VkPipelineVertexInputStateCreateFlags is a bitmask type for setting a mask, but is currently reserved for future use.

Each vertex input binding is specified by the VkVertexInputBindingDescription structure, defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkVertexInputBindingDescription {
    uint32_t            binding;
    uint32_t            stride;
    VkVertexInputRate   inputRate;
} VkVertexInputBindingDescription;
```

- binding is the binding number that this structure describes.
- stride is the byte stride between consecutive elements within the buffer.
- inputRate is a VkVertexInputRate value specifying whether vertex attribute addressing is a function of the vertex index or of the instance index.

### Valid Usage

- VUID-VkVertexInputBindingDescription-binding-00618
  binding **must** be less than VkPhysicalDeviceLimits::maxVertexInputBindings
- VUID-VkVertexInputBindingDescription-stride-00619
  stride **must** be less than or equal to VkPhysicalDeviceLimits::maxVertexInputBindingStride
- VUID-VkVertexInputBindingDescription-stride-04456
  If the VK_KHR_portability_subset extension is enabled, stride **must** be a multiple of, and at least as large as, VkPhysicalDevicePortabilitySubsetPropertiesKHR::minVertexInputBindingStrideAlignment

### Valid Usage (Implicit)

- VUID-VkVertexInputBindingDescription-inputRate-parameter
  inputRate **must** be a valid VkVertexInputRate value

Possible values of VkVertexInputBindingDescription::inputRate, specifying the rate at which vertex attributes are pulled from buffers, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkVertexInputRate {
    VK_VERTEX_INPUT_RATE_VERTEX = 0,
    VK_VERTEX_INPUT_RATE_INSTANCE = 1,
} VkVertexInputRate;
```

- VK_VERTEX_INPUT_RATE_VERTEX specifies that vertex attribute addressing is a function of the vertex index.
- VK_VERTEX_INPUT_RATE_INSTANCE specifies that vertex attribute addressing is a function of the instance index.

Each vertex input attribute is specified by the VkVertexInputAttributeDescription structure, defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkVertexInputAttributeDescription {
    uint32_t    location;
    uint32_t    binding;
    VkFormat    format;
    uint32_t    offset;
} VkVertexInputAttributeDescription;
```

- location is the shader input location number for this attribute.
- binding is the binding number which this attribute takes its data from.
- format is the size and type of the vertex attribute data.
- offset is a byte offset of this attribute relative to the start of an element in the vertex input binding.

### Valid Usage

- VUID-VkVertexInputAttributeDescription-location-00620
  location **must** be less than VkPhysicalDeviceLimits::maxVertexInputAttributes
- VUID-VkVertexInputAttributeDescription-binding-00621
  binding **must** be less than VkPhysicalDeviceLimits::maxVertexInputBindings
- VUID-VkVertexInputAttributeDescription-offset-00622
  offset **must** be less than or equal to VkPhysicalDeviceLimits::maxVertexInputAttributeOffset
- VUID-VkVertexInputAttributeDescription-format-00623
  The format features of format **must** contain VK_FORMAT_FEATURE_VERTEX_BUFFER_BIT
- VUID-VkVertexInputAttributeDescription-vertexAttributeAccessBeyondStride-04457

If the VK_KHR_portability_subset extension is enabled, and VkPhysicalDevicePortabilitySubsetFeaturesKHR::vertexAttributeAccessBeyondStride is VK_FALSE, the sum of offset plus the size of the vertex attribute data described by format **must** not be greater than stride in the VkVertexInputBindingDescription referenced in binding

### Valid Usage (Implicit)

- VUID-VkVertexInputAttributeDescription-format-parameter
  format **must** be a valid VkFormat value

To dynamically set the vertex input attribute and vertex input binding descriptions, call:

```
// Provided by VK_EXT_shader_object, VK_EXT_vertex_input_dynamic_state
void vkCmdSetVertexInputEXT(
    VkCommandBuffer                             commandBuffer,
    uint32_t                                    vertexBindingDescriptionCount,
    const VkVertexInputBindingDescription2EXT*  pVertexBindingDescriptions,
    uint32_t                                    vertexAttributeDescriptionCount,
    const VkVertexInputAttributeDescription2EXT* pVertexAttributeDescriptions);
```

- commandBuffer is the command buffer into which the command will be recorded.
- vertexBindingDescriptionCount is the number of vertex binding descriptions provided in pVertexBindingDescriptions.
- pVertexBindingDescriptions is a pointer to an array of VkVertexInputBindingDescription2EXT structures.
- vertexAttributeDescriptionCount is the number of vertex attribute descriptions provided in pVertexAttributeDescriptions.
- pVertexAttributeDescriptions is a pointer to an array of VkVertexInputAttributeDescription2EXT structures.

This command sets the vertex input attribute and vertex input binding descriptions state for subsequent drawing commands when drawing using shader objects, or when the graphics pipeline is created with VK_DYNAMIC_STATE_VERTEX_INPUT_EXT set in VkPipelineDynamicStateCreateInfo::pDynamicStates. Otherwise, this state is specified by the VkGraphicsPipelineCreateInfo::pVertexInputState values used to create the currently active pipeline.

If drawing using shader objects, or if the bound pipeline state object was also created with the VK_DYNAMIC_STATE_VERTEX_INPUT_BINDING_STRIDE dynamic state enabled, then vkCmdBindVertexBuffers2EXT can be used instead of vkCmdSetVertexInputEXT to dynamically set the stride.

## Valid Usage

- VUID-vkCmdSetVertexInputEXT-None-08546
  Either the `vertexInputDynamicState` feature or the `shaderObject` feature or both **must** be enabled
- VUID-vkCmdSetVertexInputEXT-vertexBindingDescriptionCount-04791
  `vertexBindingDescriptionCount` **must** be less than or equal to `VkPhysicalDeviceLimits::maxVertexInputBindings`
- VUID-vkCmdSetVertexInputEXT-vertexAttributeDescriptionCount-04792
  `vertexAttributeDescriptionCount` **must** be less than or equal to `VkPhysicalDeviceLimits::maxVertexInputAttributes`
- VUID-vkCmdSetVertexInputEXT-binding-04793
  For every `binding` specified by each element of `pVertexAttributeDescriptions`, a VkVertexInputBindingDescription2EXT **must** exist in `pVertexBindingDescriptions` with the same value of `binding`
- VUID-vkCmdSetVertexInputEXT-pVertexBindingDescriptions-04794
  All elements of `pVertexBindingDescriptions` **must** describe distinct binding numbers
- VUID-vkCmdSetVertexInputEXT-pVertexAttributeDescriptions-04795
  All elements of `pVertexAttributeDescriptions` **must** describe distinct attribute locations

## Valid Usage (Implicit)

- VUID-vkCmdSetVertexInputEXT-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdSetVertexInputEXT-pVertexBindingDescriptions-parameter
  If `vertexBindingDescriptionCount` is not `0`, `pVertexBindingDescriptions` **must** be a valid pointer to an array of `vertexBindingDescriptionCount` valid VkVertexInputBindingDescription2EXT structures
- VUID-vkCmdSetVertexInputEXT-pVertexAttributeDescriptions-parameter
  If `vertexAttributeDescriptionCount` is not `0`, `pVertexAttributeDescriptions` **must** be a valid pointer to an array of `vertexAttributeDescriptionCount` valid VkVertexInputAttributeDescription2EXT structures
- VUID-vkCmdSetVertexInputEXT-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdSetVertexInputEXT-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- VUID-vkCmdSetVertexInputEXT-videocoding
  This command **must** only be called outside of a video coding scope

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics | State |

The `VkVertexInputBindingDescription2EXT` structure is defined as:

```
// Provided by VK_EXT_shader_object, VK_EXT_vertex_input_dynamic_state
typedef struct VkVertexInputBindingDescription2EXT {
    VkStructureType      sType;
    void*                pNext;
    uint32_t             binding;
    uint32_t             stride;
    VkVertexInputRate    inputRate;
    uint32_t             divisor;
} VkVertexInputBindingDescription2EXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `binding` is the binding number that this structure describes.
- `stride` is the byte stride between consecutive elements within the buffer.
- `inputRate` is a VkVertexInputRate value specifying whether vertex attribute addressing is a function of the vertex index or of the instance index.
- `divisor` is the number of successive instances that will use the same value of the vertex attribute when instanced rendering is enabled. This member **can** be set to a value other than `1` if the `vertexAttributeInstanceRateDivisor` feature is enabled. For example, if the divisor is N, the same vertex attribute will be applied to N successive instances before moving on to the next vertex attribute. The maximum value of `divisor` is implementation-dependent and can be queried using `VkPhysicalDeviceVertexAttributeDivisorPropertiesEXT::maxVertexAttribDivisor`. A value of `0` **can** be used for the divisor if the `vertexAttributeInstanceRateZeroDivisor` feature is enabled. In this case, the same vertex attribute will be applied to all instances.

## Valid Usage

- VUID-VkVertexInputBindingDescription2EXT-binding-04796
  `binding` **must** be less than `VkPhysicalDeviceLimits::maxVertexInputBindings`
- VUID-VkVertexInputBindingDescription2EXT-stride-04797
  `stride` **must** be less than or equal to `VkPhysicalDeviceLimits::maxVertexInputBindingStride`
- VUID-VkVertexInputBindingDescription2EXT-divisor-04798
  If the `vertexAttributeInstanceRateZeroDivisor` feature is not enabled, `divisor` **must** not be `0`
- VUID-VkVertexInputBindingDescription2EXT-divisor-04799
  If the `vertexAttributeInstanceRateDivisor` feature is not enabled, `divisor` **must** be `1`
- VUID-VkVertexInputBindingDescription2EXT-divisor-06226
  `divisor` **must** be a value between `0` and `VkPhysicalDeviceVertexAttributeDivisorPropertiesEXT::maxVertexAttribDivisor`, inclusive
- VUID-VkVertexInputBindingDescription2EXT-divisor-06227
  If `divisor` is not `1` then `inputRate` **must** be of type `VK_VERTEX_INPUT_RATE_INSTANCE`

## Valid Usage (Implicit)

- VUID-VkVertexInputBindingDescription2EXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_VERTEX_INPUT_BINDING_DESCRIPTION_2_EXT`
- VUID-VkVertexInputBindingDescription2EXT-inputRate-parameter
  `inputRate` **must** be a valid VkVertexInputRate value

The `VkVertexInputAttributeDescription2EXT` structure is defined as:

```
// Provided by VK_EXT_shader_object, VK_EXT_vertex_input_dynamic_state
typedef struct VkVertexInputAttributeDescription2EXT {
    VkStructureType    sType;
    void*              pNext;
    uint32_t           location;
    uint32_t           binding;
    VkFormat           format;
    uint32_t           offset;
} VkVertexInputAttributeDescription2EXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `location` is the shader input location number for this attribute.
- `binding` is the binding number which this attribute takes its data from.
- `format` is the size and type of the vertex attribute data.

- offset is a byte offset of this attribute relative to the start of an element in the vertex input binding.

### Valid Usage

- VUID-VkVertexInputAttributeDescription2EXT-location-06228
  location **must** be less than VkPhysicalDeviceLimits::maxVertexInputAttributes
- VUID-VkVertexInputAttributeDescription2EXT-binding-06229
  binding **must** be less than VkPhysicalDeviceLimits::maxVertexInputBindings
- VUID-VkVertexInputAttributeDescription2EXT-offset-06230
  offset **must** be less than or equal to VkPhysicalDeviceLimits::maxVertexInputAttributeOffset
- VUID-VkVertexInputAttributeDescription2EXT-format-04805
  The format features of format **must** contain VK_FORMAT_FEATURE_VERTEX_BUFFER_BIT
- VUID-VkVertexInputAttributeDescription2EXT-vertexAttributeAccessBeyondStride-04806
  If the VK_KHR_portability_subset extension is enabled, and VkPhysicalDevicePortabilitySubsetFeaturesKHR::vertexAttributeAccessBeyondStride is VK_FALSE, the sum of offset plus the size of the vertex attribute data described by format **must** not be greater than stride in the VkVertexInputBindingDescription2EXT referenced in binding

### Valid Usage (Implicit)

- VUID-VkVertexInputAttributeDescription2EXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_VERTEX_INPUT_ATTRIBUTE_DESCRIPTION_2_EXT
- VUID-VkVertexInputAttributeDescription2EXT-format-parameter
  format **must** be a valid VkFormat value

To bind vertex buffers to a command buffer for use in subsequent drawing commands, call:

```
// Provided by VK_VERSION_1_0
void vkCmdBindVertexBuffers(
    VkCommandBuffer                             commandBuffer,
    uint32_t                                    firstBinding,
    uint32_t                                    bindingCount,
    const VkBuffer*                             pBuffers,
    const VkDeviceSize*                         pOffsets);
```

- commandBuffer is the command buffer into which the command is recorded.
- firstBinding is the index of the first vertex input binding whose state is updated by the command.
- bindingCount is the number of vertex input bindings whose state is updated by the command.

- pBuffers is a pointer to an array of buffer handles.
- pOffsets is a pointer to an array of buffer offsets.

The values taken from elements i of pBuffers and pOffsets replace the current state for the vertex input binding firstBinding + i, for i in [0, bindingCount). The vertex input binding is updated to start at the offset indicated by pOffsets[i] from the start of the buffer pBuffers[i]. All vertex input attributes that use each of these bindings will use these updated addresses in their address calculations for subsequent drawing commands. If the nullDescriptor feature is enabled, elements of pBuffers **can** be VK_NULL_HANDLE, and **can** be used by the vertex shader. If a vertex input attribute is bound to a vertex input binding that is VK_NULL_HANDLE, the values taken from memory are considered to be zero, and missing G, B, or A components are filled with (0,0,1).

## Valid Usage

- VUID-vkCmdBindVertexBuffers-firstBinding-00624
  firstBinding **must** be less than VkPhysicalDeviceLimits::maxVertexInputBindings
- VUID-vkCmdBindVertexBuffers-firstBinding-00625
  The sum of firstBinding and bindingCount **must** be less than or equal to VkPhysicalDeviceLimits::maxVertexInputBindings
- VUID-vkCmdBindVertexBuffers-pOffsets-00626
  All elements of pOffsets **must** be less than the size of the corresponding element in pBuffers
- VUID-vkCmdBindVertexBuffers-pBuffers-00627
  All elements of pBuffers **must** have been created with the VK_BUFFER_USAGE_VERTEX_BUFFER_BIT flag
- VUID-vkCmdBindVertexBuffers-pBuffers-00628
  Each element of pBuffers that is non-sparse **must** be bound completely and contiguously to a single VkDeviceMemory object
- VUID-vkCmdBindVertexBuffers-pBuffers-04001
  If the nullDescriptor feature is not enabled, all elements of pBuffers **must** not be VK_NULL_HANDLE
- VUID-vkCmdBindVertexBuffers-pBuffers-04002
  If an element of pBuffers is VK_NULL_HANDLE, then the corresponding element of pOffsets **must** be zero

## Valid Usage (Implicit)

- VUID-vkCmdBindVertexBuffers-commandBuffer-parameter
  commandBuffer **must** be a valid VkCommandBuffer handle
- VUID-vkCmdBindVertexBuffers-pBuffers-parameter
  pBuffers **must** be a valid pointer to an array of bindingCount valid or VK_NULL_HANDLE VkBuffer handles
- VUID-vkCmdBindVertexBuffers-pOffsets-parameter

pOffsets **must** be a valid pointer to an array of bindingCount VkDeviceSize values

- VUID-vkCmdBindVertexBuffers-commandBuffer-recording
  commandBuffer **must** be in the recording state
- VUID-vkCmdBindVertexBuffers-commandBuffer-cmdpool
  The VkCommandPool that commandBuffer was allocated from **must** support graphics operations
- VUID-vkCmdBindVertexBuffers-videocoding
  This command **must** only be called outside of a video coding scope
- VUID-vkCmdBindVertexBuffers-bindingCount-arraylength
  bindingCount **must** be greater than 0
- VUID-vkCmdBindVertexBuffers-commonparent
  Both of commandBuffer, and the elements of pBuffers that are valid handles of non-ignored parameters **must** have been created, allocated, or retrieved from the same VkDevice

## Host Synchronization

- Host access to commandBuffer **must** be externally synchronized
- Host access to the VkCommandPool that commandBuffer was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics | State |

Alternatively, to bind vertex buffers, along with their sizes and strides, to a command buffer for use in subsequent drawing commands, call:

```
// Provided by VK_EXT_extended_dynamic_state, VK_EXT_shader_object
void vkCmdBindVertexBuffers2EXT(
    VkCommandBuffer                             commandBuffer,
    uint32_t                                    firstBinding,
    uint32_t                                    bindingCount,
    const VkBuffer*                             pBuffers,
    const VkDeviceSize*                         pOffsets,
    const VkDeviceSize*                         pSizes,
    const VkDeviceSize*                         pStrides);
```

- commandBuffer is the command buffer into which the command is recorded.

- firstBinding is the index of the first vertex input binding whose state is updated by the command.
- bindingCount is the number of vertex input bindings whose state is updated by the command.
- pBuffers is a pointer to an array of buffer handles.
- pOffsets is a pointer to an array of buffer offsets.
- pSizes is NULL or a pointer to an array of the size in bytes of vertex data bound from pBuffers.
- pStrides is NULL or a pointer to an array of buffer strides.

The values taken from elements i of pBuffers and pOffsets replace the current state for the vertex input binding firstBinding + i, for i in [0, bindingCount). The vertex input binding is updated to start at the offset indicated by pOffsets[i] from the start of the buffer pBuffers[i]. If pSizes is not NULL then pSizes[i] specifies the bound size of the vertex buffer starting from the corresponding elements of pBuffers[i] plus pOffsets[i]. If pSizes[i] is VK_WHOLE_SIZE then the bound size is from pBuffers[i] plus pOffsets[i] to the end of the buffer pBuffers[i]. All vertex input attributes that use each of these bindings will use these updated addresses in their address calculations for subsequent drawing commands. If the nullDescriptor feature is enabled, elements of pBuffers **can** be VK_NULL_HANDLE, and **can** be used by the vertex shader. If a vertex input attribute is bound to a vertex input binding that is VK_NULL_HANDLE, the values taken from memory are considered to be zero, and missing G, B, or A components are filled with (0,0,1).

This command also dynamically sets the byte strides between consecutive elements within buffer pBuffers[i] to the corresponding pStrides[i] value when drawing using shader objects, or when the graphics pipeline is created with VK_DYNAMIC_STATE_VERTEX_INPUT_BINDING_STRIDE set in VkPipelineDynamicStateCreateInfo::pDynamicStates. Otherwise, strides are specified by the VkVertexInputBindingDescription::stride values used to create the currently active pipeline.

If drawing using shader objects or if the bound pipeline state object was also created with the VK_DYNAMIC_STATE_VERTEX_INPUT_EXT dynamic state enabled then vkCmdSetVertexInputEXT **can** be used instead of vkCmdBindVertexBuffers2EXT to set the stride.

| | |
|---|---|
| **NOTE** | Unlike the static state to set the same, pStrides must be between 0 and the maximum extent of the attributes in the binding. vkCmdSetVertexInputEXT does not have this restriction so can be used if other stride values are desired. |

## Valid Usage

- VUID-vkCmdBindVertexBuffers2-firstBinding-03355
  firstBinding **must** be less than VkPhysicalDeviceLimits::maxVertexInputBindings
- VUID-vkCmdBindVertexBuffers2-firstBinding-03356
  The sum of firstBinding and bindingCount **must** be less than or equal to VkPhysicalDeviceLimits::maxVertexInputBindings
- VUID-vkCmdBindVertexBuffers2-pOffsets-03357
  If pSizes is not NULL, all elements of pOffsets **must** be less than the size of the corresponding element in pBuffers
- VUID-vkCmdBindVertexBuffers2-pSizes-03358

If pSizes is not NULL, all elements of pOffsets plus pSizes , where pSizes is not VK_WHOLE_SIZE, **must** be less than or equal to the size of the corresponding element in pBuffers

- VUID-vkCmdBindVertexBuffers2-pBuffers-03359
  All elements of pBuffers **must** have been created with the VK_BUFFER_USAGE_VERTEX_BUFFER_BIT flag
- VUID-vkCmdBindVertexBuffers2-pBuffers-03360
  Each element of pBuffers that is non-sparse **must** be bound completely and contiguously to a single VkDeviceMemory object
- VUID-vkCmdBindVertexBuffers2-pBuffers-04111
  If the nullDescriptor feature is not enabled, all elements of pBuffers **must** not be VK_NULL_HANDLE
- VUID-vkCmdBindVertexBuffers2-pBuffers-04112
  If an element of pBuffers is VK_NULL_HANDLE, then the corresponding element of pOffsets **must** be zero
- VUID-vkCmdBindVertexBuffers2-pStrides-03362
  If pStrides is not NULL each element of pStrides **must** be less than or equal to VkPhysicalDeviceLimits::maxVertexInputBindingStride
- VUID-vkCmdBindVertexBuffers2-pStrides-06209
  If pStrides is not NULL each element of pStrides **must** be either 0 or greater than or equal to the maximum extent of all vertex input attributes fetched from the corresponding binding, where the extent is calculated as the VkVertexInputAttributeDescription::offset plus VkVertexInputAttributeDescription::format size

## Valid Usage (Implicit)

- VUID-vkCmdBindVertexBuffers2-commandBuffer-parameter
  commandBuffer **must** be a valid VkCommandBuffer handle
- VUID-vkCmdBindVertexBuffers2-pBuffers-parameter
  pBuffers **must** be a valid pointer to an array of bindingCount valid or VK_NULL_HANDLE VkBuffer handles
- VUID-vkCmdBindVertexBuffers2-pOffsets-parameter
  pOffsets **must** be a valid pointer to an array of bindingCount VkDeviceSize values
- VUID-vkCmdBindVertexBuffers2-pSizes-parameter
  If pSizes is not NULL, pSizes **must** be a valid pointer to an array of bindingCount VkDeviceSize values
- VUID-vkCmdBindVertexBuffers2-pStrides-parameter
  If pStrides is not NULL, pStrides **must** be a valid pointer to an array of bindingCount VkDeviceSize values
- VUID-vkCmdBindVertexBuffers2-commandBuffer-recording
  commandBuffer **must** be in the recording state
- VUID-vkCmdBindVertexBuffers2-commandBuffer-cmdpool

The VkCommandPool that commandBuffer was allocated from **must** support graphics operations

- VUID-vkCmdBindVertexBuffers2-videocoding
  This command **must** only be called outside of a video coding scope
- VUID-vkCmdBindVertexBuffers2-bindingCount-arraylength
  If any of pSizes, or pStrides are not NULL, bindingCount **must** be greater than 0
- VUID-vkCmdBindVertexBuffers2-commonparent
  Both of commandBuffer, and the elements of pBuffers that are valid handles of non-ignored parameters **must** have been created, allocated, or retrieved from the same VkDevice

### Host Synchronization

- Host access to commandBuffer **must** be externally synchronized
- Host access to the VkCommandPool that commandBuffer was allocated from **must** be externally synchronized

### Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics | State |

## 22.3. Vertex Attribute Divisor in Instanced Rendering

If the vertexAttributeInstanceRateDivisor feature is enabled and the pNext chain of VkPipelineVertexInputStateCreateInfo includes a VkPipelineVertexInputDivisorStateCreateInfoKHR structure, then that structure controls how vertex attributes are assigned to an instance when instanced rendering is enabled.

The VkPipelineVertexInputDivisorStateCreateInfoKHR structure is defined as:

```
// Provided by VK_KHR_vertex_attribute_divisor
typedef struct VkPipelineVertexInputDivisorStateCreateInfoKHR {
    VkStructureType                                     sType;
    const void*                                         pNext;
    uint32_t                                            vertexBindingDivisorCount;
    const VkVertexInputBindingDivisorDescriptionKHR*    pVertexBindingDivisors;
} VkPipelineVertexInputDivisorStateCreateInfoKHR;
```

or the equivalent

```
// Provided by VK_EXT_vertex_attribute_divisor
typedef VkPipelineVertexInputDivisorStateCreateInfoKHR
VkPipelineVertexInputDivisorStateCreateInfoEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `vertexBindingDivisorCount` is the number of elements in the `pVertexBindingDivisors` array.
- `pVertexBindingDivisors` is a pointer to an array of VkVertexInputBindingDivisorDescriptionKHR structures specifying the divisor value for each binding.

### Valid Usage (Implicit)

- VUID-VkPipelineVertexInputDivisorStateCreateInfoKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PIPELINE_VERTEX_INPUT_DIVISOR_STATE_CREATE_INFO_KHR`
- VUID-VkPipelineVertexInputDivisorStateCreateInfoKHR-pVertexBindingDivisors-parameter
  `pVertexBindingDivisors` **must** be a valid pointer to an array of `vertexBindingDivisorCount` VkVertexInputBindingDivisorDescriptionKHR structures
- VUID-VkPipelineVertexInputDivisorStateCreateInfoKHR-vertexBindingDivisorCount-arraylength
  `vertexBindingDivisorCount` **must** be greater than `0`

The individual divisor values per binding are specified using the `VkVertexInputBindingDivisorDescriptionKHR` structure which is defined as:

```
// Provided by VK_KHR_vertex_attribute_divisor
typedef struct VkVertexInputBindingDivisorDescriptionKHR {
    uint32_t    binding;
    uint32_t    divisor;
} VkVertexInputBindingDivisorDescriptionKHR;
```

or the equivalent

```
// Provided by VK_EXT_vertex_attribute_divisor
typedef VkVertexInputBindingDivisorDescriptionKHR
VkVertexInputBindingDivisorDescriptionEXT;
```

- `binding` is the binding number for which the divisor is specified.
- `divisor` is the number of successive instances that will use the same value of the vertex attribute when instanced rendering is enabled. For example, if the divisor is N, the same vertex attribute will be applied to N successive instances before moving on to the next vertex attribute. The maximum value of `divisor` is implementation-dependent and can be queried using

VkPhysicalDeviceVertexAttributeDivisorPropertiesKHR::maxVertexAttribDivisor. A value of 0 **can** be used for the divisor if the vertexAttributeInstanceRateZeroDivisor feature is enabled. In this case, the same vertex attribute will be applied to all instances.

If this structure is not used to define a divisor value for an attribute, then the divisor has a logical default value of 1.

### Valid Usage

- VUID-VkVertexInputBindingDivisorDescriptionKHR-binding-01869
  binding **must** be less than VkPhysicalDeviceLimits::maxVertexInputBindings
- VUID-VkVertexInputBindingDivisorDescriptionKHR-vertexAttributeInstanceRateZeroDivisor-02228
  If the vertexAttributeInstanceRateZeroDivisor feature is not enabled, divisor **must** not be 0
- VUID-VkVertexInputBindingDivisorDescriptionKHR-vertexAttributeInstanceRateDivisor-02229
  If the vertexAttributeInstanceRateDivisor feature is not enabled, divisor **must** be 1
- VUID-VkVertexInputBindingDivisorDescriptionKHR-divisor-01870
  divisor **must** be a value between 0 and VkPhysicalDeviceVertexAttributeDivisorPropertiesKHR::maxVertexAttribDivisor, inclusive
- VUID-VkVertexInputBindingDivisorDescriptionKHR-inputRate-01871
  VkVertexInputBindingDescription::inputRate **must** be of type VK_VERTEX_INPUT_RATE_INSTANCE for this binding

## 22.4. Vertex Input Address Calculation

The address of each attribute for each vertexIndex and instanceIndex is calculated as follows:

- Let attribDesc be the member of VkPipelineVertexInputStateCreateInfo::pVertexAttributeDescriptions with VkVertexInputAttributeDescription::location equal to the vertex input attribute number.
- Let bindingDesc be the member of VkPipelineVertexInputStateCreateInfo::pVertexBindingDescriptions with VkVertexInputAttributeDescription::binding equal to attribDesc.binding.
- Let vertexIndex be the index of the vertex within the draw (a value between firstVertex and firstVertex+vertexCount for vkCmdDraw, or a value taken from the index buffer plus vertexOffset for vkCmdDrawIndexed), and let instanceIndex be the instance number of the draw (a value between firstInstance and firstInstance+instanceCount).
- Let offset be an array of offsets into the currently bound vertex buffers specified during vkCmdBindVertexBuffers or vkCmdBindVertexBuffers2EXT with pOffsets.
- Let divisor be the member of VkPipelineVertexInputDivisorStateCreateInfoKHR::pVertexBindingDivisors with VkVertexInputBindingDivisorDescriptionKHR::binding equal to attribDesc.binding. If the vertex binding state is dynamically set, instead let divisor be the

member of the pVertexBindingDescriptions parameter to the vkCmdSetVertexInputEXT call with VkVertexInputBindingDescription2EXT::binding equal to attribDesc.binding.

- Let stride be the member of VkPipelineVertexInputStateCreateInfo::pVertexBindingDescriptions->stride unless there is dynamic state causing the value to be ignored. In this case the value is set from the last value from one of the following
  - vkCmdSetVertexInputEXT::pVertexBindingDescriptions->stride
  - vkCmdBindVertexBuffers2EXT::pStride, if not NULL

```
bufferBindingAddress = buffer[binding].baseAddress + offset[binding];

if (bindingDesc.inputRate == VK_VERTEX_INPUT_RATE_VERTEX)
    effectiveVertexOffset = vertexIndex * stride;
else
    if (divisor == 0)
        effectiveVertexOffset = firstInstance * stride;
    else
        effectiveVertexOffset = (firstInstance + ((instanceIndex - firstInstance) /
divisor)) * stride;

attribAddress = bufferBindingAddress + effectiveVertexOffset + attribDesc.offset;
```

### 22.4.1. Vertex Input Extraction

For each attribute, raw data is extracted starting at attribAddress and is converted from the VkVertexInputAttributeDescription's format to either floating-point, unsigned integer, or signed integer based on the numeric type of format. The numeric type of format **must** match the numeric type of the input variable in the shader. The input variable in the shader **must** be declared as a 64-bit data type if and only if format is a 64-bit data type. If either format is a 64-bit format or legacyVertexAttributes is not enabled, and format is a packed format, attribAddress **must** be a multiple of the size in bytes of the whole attribute data type as described in Packed Formats. Otherwise, if either format is a 64-bit format or legacyVertexAttributes is not enabled, attribAddress **must** be a multiple of the size in bytes of the component type indicated by format (see Formats). For attributes that are not 64-bit data types, each component is converted to the format of the input variable based on its type and size (as defined in the Format Definition section for each VkFormat), using the appropriate equations in 16-Bit Floating-Point Numbers, Unsigned 11-Bit Floating-Point Numbers, Unsigned 10-Bit Floating-Point Numbers, Fixed-Point Data Conversion, and Shared Exponent to RGB. Signed integer components smaller than 32 bits are sign-extended. Attributes that are not 64-bit data types are expanded to four components in the same way as described in conversion to RGBA. The number of components in the vertex shader input variable need not exactly match the number of components in the format. If the vertex shader has fewer components, the extra components are discarded.

# Chapter 23. Tessellation

Tessellation involves three pipeline stages. First, a tessellation control shader transforms control points of a patch and **can** produce per-patch data. Second, a fixed-function tessellator generates multiple primitives corresponding to a tessellation of the patch in (u,v) or (u,v,w) parameter space. Third, a tessellation evaluation shader transforms the vertices of the tessellated patch, for example to compute their positions and attributes as part of the tessellated surface. The tessellator is enabled when the pipeline contains both a tessellation control shader and a tessellation evaluation shader.

## 23.1. Tessellator

If a pipeline includes both tessellation shaders (control and evaluation), the tessellator consumes each input patch (after vertex shading) and produces a new set of independent primitives (points, lines, or triangles). These primitives are logically produced by subdividing a geometric primitive (rectangle or triangle) according to the per-patch outer and inner tessellation levels written by the tessellation control shader. These levels are specified using the built-in variables `TessLevelOuter` and `TessLevelInner`, respectively. This subdivision is performed in an implementation-dependent manner. If no tessellation shaders are present in the pipeline, the tessellator is disabled and incoming primitives are passed through without modification.

The type of subdivision performed by the tessellator is specified by an `OpExecutionMode` instruction using one of the `Triangles`, `Quads`, or `IsoLines` execution modes. When using shader objects, this instruction **must** be specified in the tessellation evaluation shader, and **may** also be specified in the tessellation control shader. When using pipelines, this instruction **may** be specified in either the tessellation evaluation or tessellation control shader. When using shader objects, tessellation-related modes that are **required must** be specified in the tessellation evaluation shader, and **may** also be specified in the tessellation control shader. Other tessellation-related modes **may** be specified in the tessellation evaluation shader. When using pipelines, other tessellation-related execution modes **can** also be specified in either the tessellation control or tessellation evaluation shaders.

Any tessellation-related modes specified in both the tessellation control and tessellation evaluation shaders **must** be the same.

Tessellation execution modes include:

- `Triangles`, `Quads`, and `IsoLines`. These control the type of subdivision and topology of the output primitives. When using shader objects, one mode **must** be set in at least the tessellation evaluation stage. When using pipelines, one mode **must** be set in at least one of the tessellation shader stages. If the `VK_KHR_portability_subset` extension is enabled, and VkPhysicalDevicePortabilitySubsetFeaturesKHR::`tessellationIsolines` is `VK_FALSE`, then isoline tessellation is not supported by the implementation, and `IsoLines` **must** not be used in either tessellation shader stage.
- `VertexOrderCw` and `VertexOrderCcw`. These control the orientation of triangles generated by the tessellator. When using shader objects, one mode **must** be set in at least the tessellation evaluation stage. When using pipelines, one mode **must** be set in at least one of the tessellation

shader stages.

- `PointMode`. Controls generation of points rather than triangles or lines. This functionality defaults to disabled, and is enabled if either shader stage includes the execution mode. When using shader objects, if `PointMode` is set in the tessellation control stage, it **must** be identically set in the tessellation evaluation stage. If the `VK_KHR_portability_subset` extension is enabled, and VkPhysicalDevicePortabilitySubsetFeaturesKHR::`tessellationPointMode` is `VK_FALSE`, then point mode tessellation is not supported by the implementation, and `PointMode` **must** not be used in either tessellation shader stage.
- `SpacingEqual`, `SpacingFractionalEven`, and `SpacingFractionalOdd`. Controls the spacing of segments on the edges of tessellated primitives. When using shader objects, one mode **must** be set in at least the tessellation evaluation stage. When using pipelines, one mode **must** be set in at least one of the tessellation shader stages.
- `OutputVertices`. Controls the size of the output patch of the tessellation control shader. When using shader objects, one value **must** be set in at least the tessellation control stage. When using pipelines, one value **must** be set in at least one of the tessellation shader stages.

For triangles, the tessellator subdivides a triangle primitive into smaller triangles. For quads, the tessellator subdivides a rectangle primitive into smaller triangles. For isolines, the tessellator subdivides a rectangle primitive into a collection of line segments arranged in strips stretching across the rectangle in the u dimension (i.e. the coordinates in `TessCoord` are of the form (0,x) through (1,x) for all tessellation evaluation shader invocations that share a line).

Each vertex produced by the tessellator has an associated (u,v,w) or (u,v) position in a normalized parameter space, with parameter values in the range [0,1], as illustrated in figures Domain parameterization for tessellation primitive modes (upper-left origin) and Domain parameterization for tessellation primitive modes (lower-left origin). The domain space **can** have either an upper-left or lower-left origin, selected by the `domainOrigin` member of VkPipelineTessellationDomainOriginStateCreateInfo.

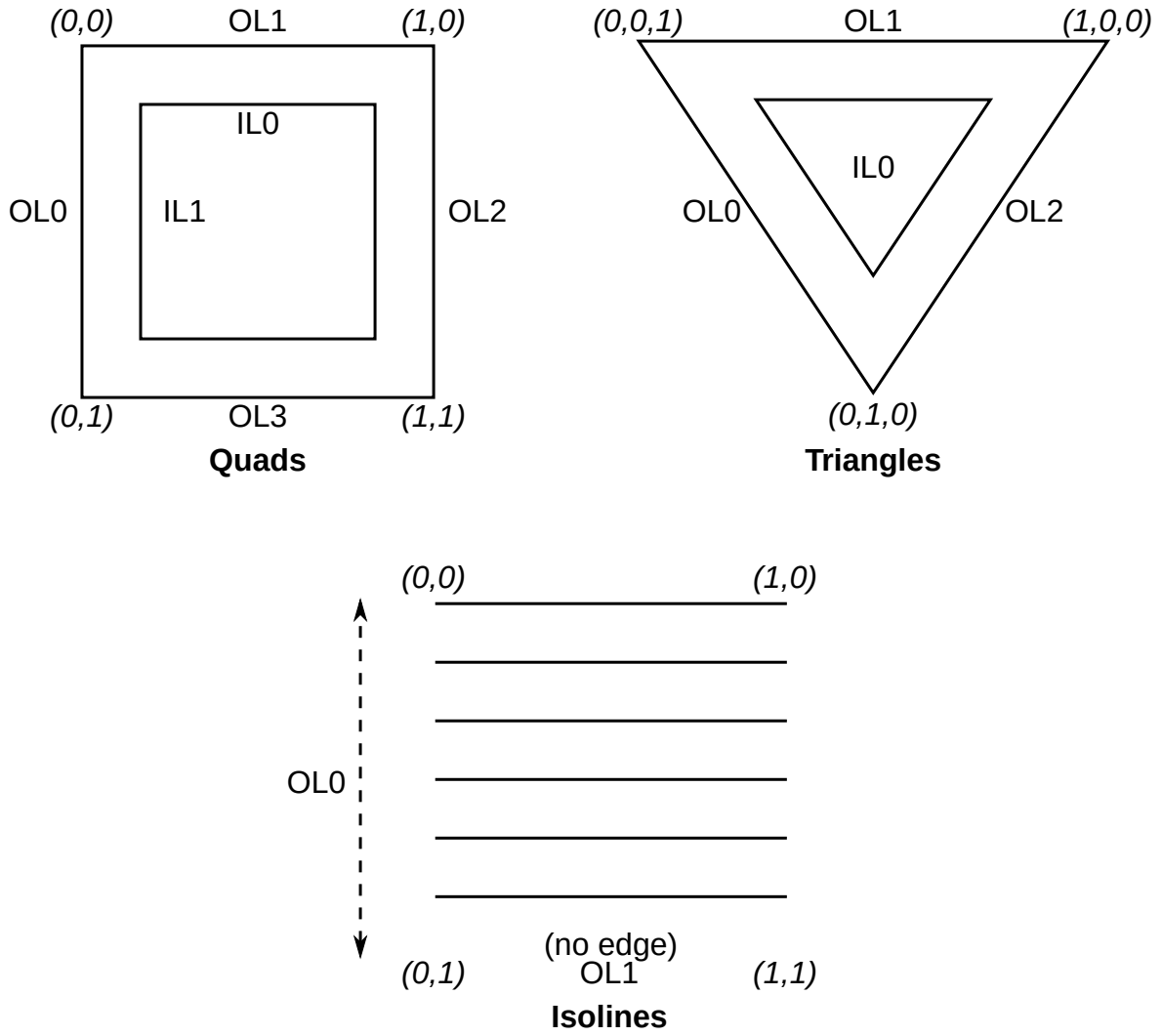



*Figure 15. Domain parameterization for tessellation primitive modes (upper-left origin)*

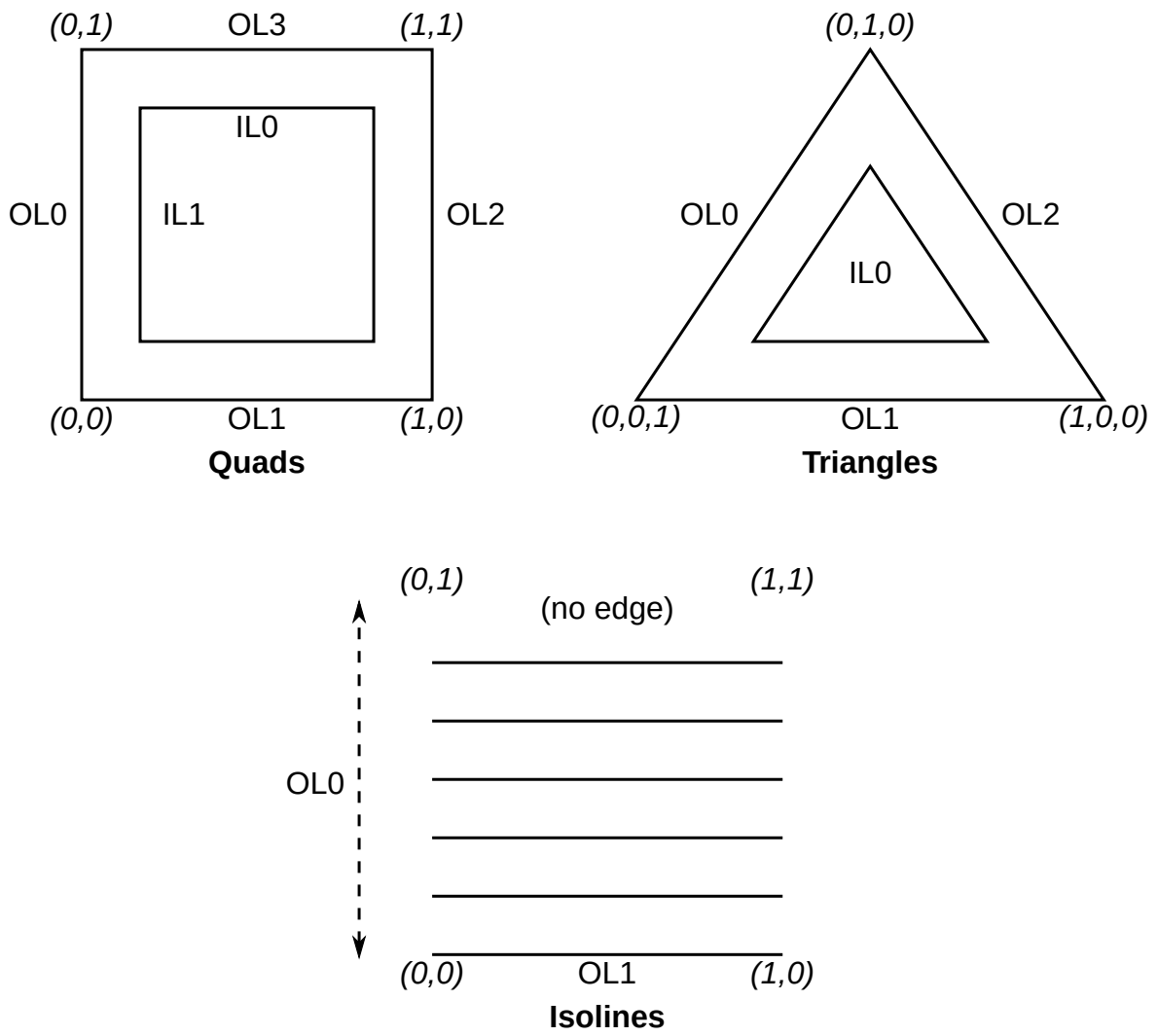



*Figure 16. Domain parameterization for tessellation primitive modes (lower-left origin)*

**Caption**

In the domain parameterization diagrams, the coordinates illustrate the value of `TessCoord` at the corners of the domain. The labels on the edges indicate the inner (IL0 and IL1) and outer (OL0 through OL3) tessellation level values used to control the number of subdivisions along each edge of the domain.

For triangles, the vertex's position is a barycentric coordinate (u,v,w), where u + v + w = 1.0, and indicates the relative influence of the three vertices of the triangle on the position of the vertex. For quads and isolines, the position is a (u,v) coordinate indicating the relative horizontal and vertical position of the vertex relative to the subdivided rectangle. The subdivision process is explained in more detail in subsequent sections.

## 23.2. Tessellator Patch Discard

A patch is discarded by the tessellator if any relevant outer tessellation level is less than or equal to zero.

Patches will also be discarded if any relevant outer tessellation level corresponds to a floating-point

NaN (not a number) in implementations supporting NaN.

No new primitives are generated and the tessellation evaluation shader is not executed for patches that are discarded. For `Quads`, all four outer levels are relevant. For `Triangles` and `IsoLines`, only the first three or two outer levels, respectively, are relevant. Negative inner levels will not cause a patch to be discarded; they will be clamped as described below.

## 23.3. Tessellator Spacing

Each of the tessellation levels is used to determine the number and spacing of segments used to subdivide a corresponding edge. The method used to derive the number and spacing of segments is specified by an `OpExecutionMode` in the tessellation control or tessellation evaluation shader using one of the identifiers `SpacingEqual`, `SpacingFractionalEven`, or `SpacingFractionalOdd`.

If `SpacingEqual` is used, the floating-point tessellation level is first clamped to [1, `maxLevel`], where `maxLevel` is the implementation-dependent maximum tessellation level (`VkPhysicalDeviceLimits::maxTessellationGenerationLevel`). The result is rounded up to the nearest integer n, and the corresponding edge is divided into n segments of equal length in (u,v) space.

If `SpacingFractionalEven` is used, the tessellation level is first clamped to [2, `maxLevel`] and then rounded up to the nearest even integer n. If `SpacingFractionalOdd` is used, the tessellation level is clamped to [1, `maxLevel` - 1] and then rounded up to the nearest odd integer n. If n is one, the edge will not be subdivided. Otherwise, the corresponding edge will be divided into n - 2 segments of equal length, and two additional segments of equal length that are typically shorter than the other segments. The length of the two additional segments relative to the others will decrease monotonically with n - f, where f is the clamped floating-point tessellation level. When n - f is zero, the additional segments will have equal length to the other segments. As n - f approaches 2.0, the relative length of the additional segments approaches zero. The two additional segments **must** be placed symmetrically on opposite sides of the subdivided edge. The relative location of these two segments is implementation-dependent, but **must** be identical for any pair of subdivided edges with identical values of f.

When tessellating triangles or quads using point mode with fractional odd spacing, the tessellator **may** produce *interior vertices* that are positioned on the edge of the patch if an inner tessellation level is less than or equal to one. Such vertices are considered distinct from vertices produced by subdividing the outer edge of the patch, even if there are pairs of vertices with identical coordinates.

## 23.4. Tessellation Primitive Ordering

Few guarantees are provided for the relative ordering of primitives produced by tessellation, as they pertain to primitive order.

- The output primitives generated from each input primitive are passed to subsequent pipeline stages in an implementation-dependent order.
- All output primitives generated from a given input primitive are passed to subsequent pipeline stages before any output primitives generated from subsequent input primitives.

## 23.5. Tessellator Vertex Winding Order

When the tessellator produces triangles (in the `Triangles` or `Quads` modes), the orientation of all triangles is specified with an `OpExecutionMode` of `VertexOrderCw` or `VertexOrderCcw` in the tessellation control or tessellation evaluation shaders. If the order is `VertexOrderCw`, the vertices of all generated triangles will have clockwise ordering in (u,v) or (u,v,w) space. If the order is `VertexOrderCcw`, the vertices will have counter-clockwise ordering in that space.

If the tessellation domain has an upper-left origin, the vertices of a triangle have counter-clockwise ordering if

$$a = u_0 v_1 - u_1 v_0 + u_1 v_2 - u_2 v_1 + u_2 v_0 - u_0 v_2$$

is negative, and clockwise ordering if a is positive. $u_i$ and $v_i$ are the u and v coordinates in normalized parameter space of the ith vertex of the triangle. If the tessellation domain has a lower-left origin, the vertices of a triangle have counter-clockwise ordering if a is positive, and clockwise ordering if a is negative.

**NOTE**

The value a is proportional (with a positive factor) to the signed area of the triangle.

In `Triangles` mode, even though the vertex coordinates have a w value, it does not participate directly in the computation of a, being an affine combination of u and v.

## 23.6. Triangle Tessellation

If the tessellation primitive mode is `Triangles`, an equilateral triangle is subdivided into a collection of triangles covering the area of the original triangle. First, the original triangle is subdivided into a collection of concentric equilateral triangles. The edges of each of these triangles are subdivided, and the area between each triangle pair is filled by triangles produced by joining the vertices on the subdivided edges. The number of concentric triangles and the number of subdivisions along each triangle except the outermost is derived from the first inner tessellation level. The edges of the outermost triangle are subdivided independently, using the first, second, and third outer tessellation levels to control the number of subdivisions of the u = 0 (left), v = 0 (bottom), and w = 0 (right) edges, respectively. The second inner tessellation level and the fourth outer tessellation level have no effect in this mode.

If the first inner tessellation level and all three outer tessellation levels are exactly one after clamping and rounding, only a single triangle with (u,v,w) coordinates of (0,0,1), (1,0,0), and (0,1,0) is generated. If the inner tessellation level is one and any of the outer tessellation levels is greater than one, the inner tessellation level is treated as though it were originally specified as 1 + ε and will result in a two- or three-segment subdivision depending on the tessellation spacing. When used with fractional odd spacing, the three-segment subdivision **may** produce *inner vertices* positioned on the edge of the triangle.

If any tessellation level is greater than one, tessellation begins by producing a set of concentric inner triangles and subdividing their edges. First, the three outer edges are temporarily subdivided using the clamped and rounded first inner tessellation level and the specified tessellation spacing,

generating n segments. For the outermost inner triangle, the inner triangle is degenerate — a single point at the center of the triangle — if n is two. Otherwise, for each corner of the outer triangle, an inner triangle corner is produced at the intersection of two lines extended perpendicular to the corner's two adjacent edges running through the vertex of the subdivided outer edge nearest that corner. If n is three, the edges of the inner triangle are not subdivided and it is the final triangle in the set of concentric triangles. Otherwise, each edge of the inner triangle is divided into n - 2 segments, with the n - 1 vertices of this subdivision produced by intersecting the inner edge with lines perpendicular to the edge running through the n - 1 innermost vertices of the subdivision of the outer edge. Once the outermost inner triangle is subdivided, the previous subdivision process repeats itself, using the generated triangle as an outer triangle. This subdivision process is illustrated in Inner Triangle Tessellation.

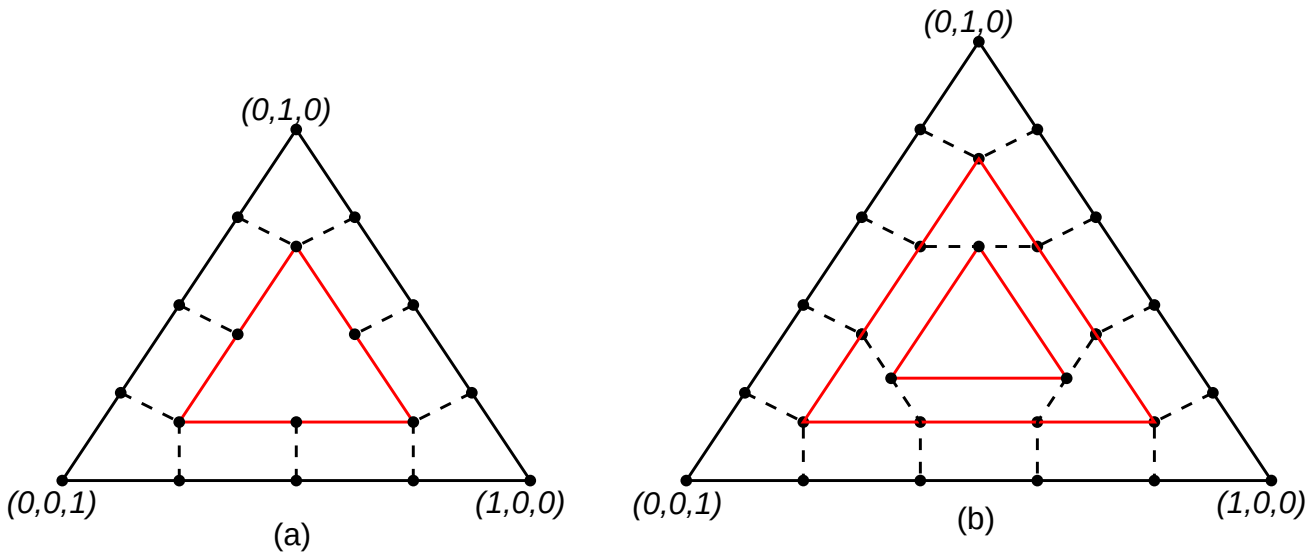



*Figure 17. Inner Triangle Tessellation*

## Caption

In the Inner Triangle Tessellation diagram, inner tessellation levels of (a) four and (b) five are shown (not to scale). Solid black circles depict vertices along the edges of the concentric triangles. The edges of inner triangles are subdivided by intersecting the edge with segments perpendicular to the edge passing through each inner vertex of the subdivided outer edge. Dotted lines depict edges connecting corresponding vertices on the inner and outer triangle edges.

Once all the concentric triangles are produced and their edges are subdivided, the area between each pair of adjacent inner triangles is filled completely with a set of non-overlapping triangles. In this subdivision, two of the three vertices of each triangle are taken from adjacent vertices on a subdivided edge of one triangle; the third is one of the vertices on the corresponding edge of the other triangle. If the innermost triangle is degenerate (i.e., a point), the triangle containing it is subdivided into six triangles by connecting each of the six vertices on that triangle with the center point. If the innermost triangle is not degenerate, that triangle is added to the set of generated triangles as-is.

After the area corresponding to any inner triangles is filled, the tessellator generates triangles to cover the area between the outermost triangle and the outermost inner triangle. To do this, the

temporary subdivision of the outer triangle edge above is discarded. Instead, the u = 0, v = 0, and w = 0 edges are subdivided according to the first, second, and third outer tessellation levels, respectively, and the tessellation spacing. The original subdivision of the first inner triangle is retained. The area between the outer and first inner triangles is completely filled by non-overlapping triangles as described above. If the first (and only) inner triangle is degenerate, a set of triangles is produced by connecting each vertex on the outer triangle edges with the center point.

After all triangles are generated, each vertex in the subdivided triangle is assigned a barycentric (u,v,w) coordinate based on its location relative to the three vertices of the outer triangle.

The algorithm used to subdivide the triangular domain in (u,v,w) space into individual triangles is implementation-dependent. However, the set of triangles produced will completely cover the domain, and no portion of the domain will be covered by multiple triangles.

Output triangles are generated with a topology similar to triangle lists, except that the order in which each triangle is generated, and the order in which the vertices are generated for each triangle, are implementation-dependent. However, the order of vertices in each triangle is consistent across the domain as described in Tessellator Vertex Winding Order.

## 23.7. Quad Tessellation

If the tessellation primitive mode is `Quads`, a rectangle is subdivided into a collection of triangles covering the area of the original rectangle. First, the original rectangle is subdivided into a regular mesh of rectangles, where the number of rectangles along the u = 0 and u = 1 (vertical) and v = 0 and v = 1 (horizontal) edges are derived from the first and second inner tessellation levels, respectively. All rectangles, except those adjacent to one of the outer rectangle edges, are decomposed into triangle pairs. The outermost rectangle edges are subdivided independently, using the first, second, third, and fourth outer tessellation levels to control the number of subdivisions of the u = 0 (left), v = 0 (bottom), u = 1 (right), and v = 1 (top) edges, respectively. The area between the inner rectangles of the mesh and the outer rectangle edges are filled by triangles produced by joining the vertices on the subdivided outer edges to the vertices on the edge of the inner rectangle mesh.

If both clamped inner tessellation levels and all four clamped outer tessellation levels are exactly one, only a single triangle pair covering the outer rectangle is generated. Otherwise, if either clamped inner tessellation level is one, that tessellation level is treated as though it was originally specified as 1 + ε and will result in a two- or three-segment subdivision depending on the tessellation spacing. When used with fractional odd spacing, the three-segment subdivision **may** produce *inner vertices* positioned on the edge of the rectangle.

If any tessellation level is greater than one, tessellation begins by subdividing the u = 0 and u = 1 edges of the outer rectangle into m segments using the clamped and rounded first inner tessellation level and the tessellation spacing. The v = 0 and v = 1 edges are subdivided into n segments using the second inner tessellation level. Each vertex on the u = 0 and v = 0 edges are joined with the corresponding vertex on the u = 1 and v = 1 edges to produce a set of vertical and horizontal lines that divide the rectangle into a grid of smaller rectangles. The primitive generator emits a pair of non-overlapping triangles covering each such rectangle not adjacent to an edge of the outer rectangle. The boundary of the region covered by these triangles forms an inner rectangle, the

edges of which are subdivided by the grid vertices that lie on the edge. If either m or n is two, the inner rectangle is degenerate, and one or both of the rectangle's *edges* consist of a single point. This subdivision is illustrated in Figure Inner Quad Tessellation.

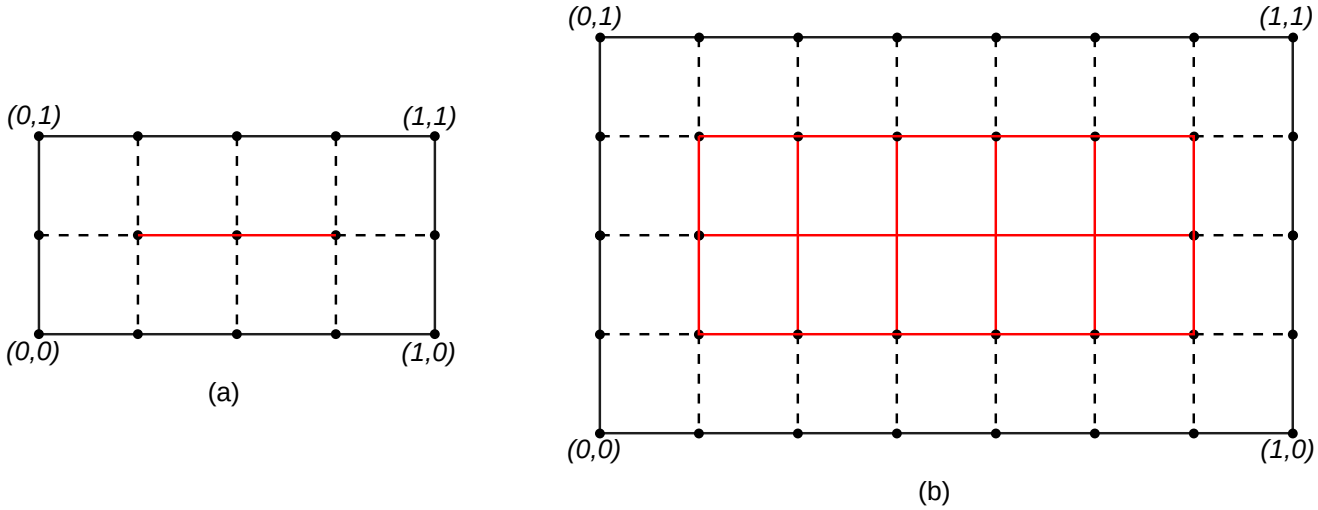



*Figure 18. Inner Quad Tessellation*

## Caption

In the Inner Quad Tessellation diagram, inner quad tessellation levels of (a) (4,2) and (b) (7,4) are shown. The regions highlighted in red in figure (b) depict the 10 inner rectangles, each of which will be subdivided into two triangles. Solid black circles depict vertices on the boundary of the outer and inner rectangles, where the inner rectangle of figure (a) is degenerate (a single line segment). Dotted lines depict the horizontal and vertical edges connecting corresponding vertices on the inner and outer rectangle edges.

After the area corresponding to the inner rectangle is filled, the tessellator **must** produce triangles to cover the area between the inner and outer rectangles. To do this, the subdivision of the outer rectangle edge above is discarded. Instead, the u = 0, v = 0, u = 1, and v = 1 edges are subdivided according to the first, second, third, and fourth outer tessellation levels, respectively, and the tessellation spacing. The original subdivision of the inner rectangle is retained. The area between the outer and inner rectangles is completely filled by non-overlapping triangles. Two of the three vertices of each triangle are adjacent vertices on a subdivided edge of one rectangle; the third is one of the vertices on the corresponding edge of the other rectangle. If either edge of the innermost rectangle is degenerate, the area near the corresponding outer edges is filled by connecting each vertex on the outer edge with the single vertex making up the *inner edge*.

The algorithm used to subdivide the rectangular domain in (u,v) space into individual triangles is implementation-dependent. However, the set of triangles produced will completely cover the domain, and no portion of the domain will be covered by multiple triangles.

Output triangles are generated with a topology similar to triangle lists, except that the order in which each triangle is generated, and the order in which the vertices are generated for each triangle, are implementation-dependent. However, the order of vertices in each triangle is consistent across the domain as described in Tessellator Vertex Winding Order.

## 23.8. Isoline Tessellation

If the tessellation primitive mode is `IsoLines`, a set of independent horizontal line segments is drawn. The segments are arranged into connected strips called *isolines*, where the vertices of each isoline have a constant v coordinate and u coordinates covering the full range [0,1]. The number of isolines generated is derived from the first outer tessellation level; the number of segments in each isoline is derived from the second outer tessellation level. Both inner tessellation levels and the third and fourth outer tessellation levels have no effect in this mode.

As with quad tessellation above, isoline tessellation begins with a rectangle. The u = 0 and u = 1 edges of the rectangle are subdivided according to the first outer tessellation level. For the purposes of this subdivision, the tessellation spacing mode is ignored and treated as equal_spacing. An isoline is drawn connecting each vertex on the u = 0 rectangle edge to the corresponding vertex on the u = 1 rectangle edge, except that no line is drawn between (0,1) and (1,1). If the number of isolines on the subdivided u = 0 and u = 1 edges is n, this process will result in n equally spaced lines with constant v coordinates of 0, $\frac{1}{n}$, $\frac{2}{n}$, …, $\frac{n-1}{n}$.

Each of the n isolines is then subdivided according to the second outer tessellation level and the tessellation spacing, resulting in m line segments. Each segment of each line is emitted by the tessellator. These line segments are generated with a topology similar to line lists, except that the order in which each line is generated, and the order in which the vertices are generated for each line segment, are implementation-dependent.

**NOTE** If the `VK_KHR_portability_subset` extension is enabled, and VkPhysicalDevicePortabilitySubsetFeaturesKHR::`tessellationIsolines` is `VK_FALSE`, then isoline tessellation is not supported by the implementation.

## 23.9. Tessellation Point Mode

For all primitive modes, the tessellator is capable of generating points instead of lines or triangles. If the tessellation control or tessellation evaluation shader specifies the `OpExecutionMode PointMode`, the primitive generator will generate one point for each distinct vertex produced by tessellation, rather than emitting triangles or lines. Otherwise, the tessellator will produce a collection of line segments or triangles according to the primitive mode. These points are generated with a topology similar to point lists, except the order in which the points are generated for each input primitive is undefined.

**NOTE** If the `VK_KHR_portability_subset` extension is enabled, and VkPhysicalDevicePortabilitySubsetFeaturesKHR::`tessellationPointMode` is `VK_FALSE`, then tessellation point mode is not supported by the implementation.

## 23.10. Tessellation Pipeline State

The `pTessellationState` member of VkGraphicsPipelineCreateInfo is a pointer to a `VkPipelineTessellationStateCreateInfo` structure.

The `VkPipelineTessellationStateCreateInfo` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkPipelineTessellationStateCreateInfo {
    VkStructureType                             sType;
    const void*                                 pNext;
    VkPipelineTessellationStateCreateFlags      flags;
    uint32_t                                    patchControlPoints;
} VkPipelineTessellationStateCreateInfo;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is reserved for future use.
- `patchControlPoints` is the number of control points per patch.

## Valid Usage

- VUID-VkPipelineTessellationStateCreateInfo-patchControlPoints-01214
  `patchControlPoints` **must** be greater than zero and less than or equal to `VkPhysicalDeviceLimits::maxTessellationPatchSize`

## Valid Usage (Implicit)

- VUID-VkPipelineTessellationStateCreateInfo-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PIPELINE_TESSELLATION_STATE_CREATE_INFO`
- VUID-VkPipelineTessellationStateCreateInfo-pNext-pNext
  `pNext` **must** be `NULL` or a pointer to a valid instance of VkPipelineTessellationDomainOriginStateCreateInfo
- VUID-VkPipelineTessellationStateCreateInfo-sType-unique
  The `sType` value of each struct in the `pNext` chain **must** be unique
- VUID-VkPipelineTessellationStateCreateInfo-flags-zerobitmask
  `flags` **must** be `0`

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkPipelineTessellationStateCreateFlags;
```

`VkPipelineTessellationStateCreateFlags` is a bitmask type for setting a mask, but is currently reserved for future use.

The `VkPipelineTessellationDomainOriginStateCreateInfo` structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkPipelineTessellationDomainOriginStateCreateInfo {
    VkStructureType                sType;
    const void*                    pNext;
    VkTessellationDomainOrigin     domainOrigin;
} VkPipelineTessellationDomainOriginStateCreateInfo;
```

or the equivalent

```
// Provided by VK_KHR_maintenance2
typedef VkPipelineTessellationDomainOriginStateCreateInfo
VkPipelineTessellationDomainOriginStateCreateInfoKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `domainOrigin` is a VkTessellationDomainOrigin value controlling the origin of the tessellation domain space.

If the `VkPipelineTessellationDomainOriginStateCreateInfo` structure is included in the `pNext` chain of VkPipelineTessellationStateCreateInfo, it controls the origin of the tessellation domain. If this structure is not present, it is as if `domainOrigin` was `VK_TESSELLATION_DOMAIN_ORIGIN_UPPER_LEFT`.

### Valid Usage (Implicit)

- VUID-VkPipelineTessellationDomainOriginStateCreateInfo-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PIPELINE_TESSELLATION_DOMAIN_ORIGIN_STATE_CREATE_INFO`
- VUID-VkPipelineTessellationDomainOriginStateCreateInfo-domainOrigin-parameter
  `domainOrigin` **must** be a valid VkTessellationDomainOrigin value

The possible tessellation domain origins are specified by the VkTessellationDomainOrigin enumeration:

```
// Provided by VK_VERSION_1_1
typedef enum VkTessellationDomainOrigin {
    VK_TESSELLATION_DOMAIN_ORIGIN_UPPER_LEFT = 0,
    VK_TESSELLATION_DOMAIN_ORIGIN_LOWER_LEFT = 1,
  // Provided by VK_KHR_maintenance2
    VK_TESSELLATION_DOMAIN_ORIGIN_UPPER_LEFT_KHR =
VK_TESSELLATION_DOMAIN_ORIGIN_UPPER_LEFT,
  // Provided by VK_KHR_maintenance2
    VK_TESSELLATION_DOMAIN_ORIGIN_LOWER_LEFT_KHR =
VK_TESSELLATION_DOMAIN_ORIGIN_LOWER_LEFT,
} VkTessellationDomainOrigin;
```

or the equivalent

```
// Provided by VK_KHR_maintenance2
typedef VkTessellationDomainOrigin VkTessellationDomainOriginKHR;
```

- VK_TESSELLATION_DOMAIN_ORIGIN_UPPER_LEFT specifies that the origin of the domain space is in the upper left corner, as shown in figure Domain parameterization for tessellation primitive modes (upper-left origin).
- VK_TESSELLATION_DOMAIN_ORIGIN_LOWER_LEFT specifies that the origin of the domain space is in the lower left corner, as shown in figure Domain parameterization for tessellation primitive modes (lower-left origin).

This enum affects how the VertexOrderCw and VertexOrderCcw tessellation execution modes are interpreted, since the winding is defined relative to the orientation of the domain.

To dynamically set the origin of the tessellation domain space, call:

```
// Provided by VK_EXT_extended_dynamic_state3 with VK_KHR_maintenance2 or
VK_VERSION_1_1, VK_EXT_shader_object
void vkCmdSetTessellationDomainOriginEXT(
    VkCommandBuffer                             commandBuffer,
    VkTessellationDomainOrigin                  domainOrigin);
```

- commandBuffer is the command buffer into which the command will be recorded.
- domainOrigin specifies the origin of the tessellation domain space.

This command sets the origin of the tessellation domain space for subsequent drawing commands when drawing using shader objects, or when the graphics pipeline is created with VK_DYNAMIC_STATE_TESSELLATION_DOMAIN_ORIGIN_EXT set in VkPipelineDynamicStateCreateInfo::pDynamicStates. Otherwise, this state is specified by the VkPipelineTessellationDomainOriginStateCreateInfo::domainOrigin value used to create the currently active pipeline.

### Valid Usage

- VUID-vkCmdSetTessellationDomainOriginEXT-None-09423
  At least one of the following **must** be true:
  - The extendedDynamicState3TessellationDomainOrigin feature is enabled
  - The shaderObject feature is enabled

### Valid Usage (Implicit)

- VUID-vkCmdSetTessellationDomainOriginEXT-commandBuffer-parameter
  commandBuffer **must** be a valid VkCommandBuffer handle

- VUID-vkCmdSetTessellationDomainOriginEXT-domainOrigin-parameter
  `domainOrigin` **must** be a valid VkTessellationDomainOrigin value
- VUID-vkCmdSetTessellationDomainOriginEXT-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdSetTessellationDomainOriginEXT-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- VUID-vkCmdSetTessellationDomainOriginEXT-videocoding
  This command **must** only be called outside of a video coding scope

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics | State |

# Chapter 24. Geometry Shading

The geometry shader operates on a group of vertices and their associated data assembled from a single input primitive, and emits zero or more output primitives and the group of vertices and their associated data required for each output primitive. Geometry shading is enabled when a geometry shader is included in the pipeline.

## 24.1. Geometry Shader Input Primitives

Each geometry shader invocation has access to all vertices in the primitive (and their associated data), which are presented to the shader as an array of inputs.

The input primitive type expected by the geometry shader is specified with an `OpExecutionMode` instruction in the geometry shader, and **must** match the incoming primitive type specified by either the pipeline's primitive topology if tessellation is inactive, or the tessellation mode if tessellation is active, as follows:

- An input primitive type of `InputPoints` **must** only be used with a pipeline topology of `VK_PRIMITIVE_TOPOLOGY_POINT_LIST`, or with a tessellation shader specifying `PointMode`. The input arrays always contain one element, as described by the point list topology or tessellation in point mode.
- An input primitive type of `InputLines` **must** only be used with a pipeline topology of `VK_PRIMITIVE_TOPOLOGY_LINE_LIST` or `VK_PRIMITIVE_TOPOLOGY_LINE_STRIP`, or with a tessellation shader specifying `IsoLines` that does not specify `PointMode`. The input arrays always contain two elements, as described by the line list topology or line strip topology, or by isoline tessellation.
- An input primitive type of `InputLinesAdjacency` **must** only be used when tessellation is inactive, with a pipeline topology of `VK_PRIMITIVE_TOPOLOGY_LINE_LIST_WITH_ADJACENCY` or `VK_PRIMITIVE_TOPOLOGY_LINE_STRIP_WITH_ADJACENCY`. The input arrays always contain four elements, as described by the line list with adjacency topology or line strip with adjacency topology.
- An input primitive type of `Triangles` **must** only be used with a pipeline topology of `VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST`, `VK_PRIMITIVE_TOPOLOGY_TRIANGLE_STRIP`, or `VK_PRIMITIVE_TOPOLOGY_TRIANGLE_FAN`; or with a tessellation shader specifying `Quads` or `Triangles` that does not specify `PointMode`. The input arrays always contain three elements, as described by the triangle list topology, triangle strip topology, or triangle fan topology, or by triangle or quad tessellation. Vertices **may** be in a different absolute order than specified by the topology, but **must** adhere to the specified winding order.
- An input primitive type of `InputTrianglesAdjacency` **must** only be used when tessellation is inactive, with a pipeline topology of `VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST_WITH_ADJACENCY` or `VK_PRIMITIVE_TOPOLOGY_TRIANGLE_STRIP_WITH_ADJACENCY`. The input arrays always contain six elements, as described by the triangle list with adjacency topology or triangle strip with adjacency topology. Vertices **may** be in a different absolute order than specified by the topology, but **must** adhere to the specified winding order, and the vertices making up the main primitive **must** still occur at the first, third, and fifth index.

## 24.2. Geometry Shader Output Primitives

A geometry shader generates primitives in one of three output modes: points, line strips, or triangle strips. The primitive mode is specified in the shader using an `OpExecutionMode` instruction with the `OutputPoints`, `OutputLineStrip` or `OutputTriangleStrip` modes, respectively. Each geometry shader **must** include exactly one output primitive mode.

The vertices output by the geometry shader are assembled into points, lines, or triangles based on the output primitive type and the resulting primitives are then further processed as described in Rasterization. If the number of vertices emitted by the geometry shader is not sufficient to produce a single primitive, vertices corresponding to incomplete primitives are not processed by subsequent pipeline stages. The number of vertices output by the geometry shader is limited to a maximum count specified in the shader.

The maximum output vertex count is specified in the shader using an `OpExecutionMode` instruction with the mode set to `OutputVertices` and the maximum number of vertices that will be produced by the geometry shader specified as a literal. Each geometry shader **must** specify a maximum output vertex count.

## 24.3. Multiple Invocations of Geometry Shaders

Geometry shaders **can** be invoked more than one time for each input primitive. This is known as *geometry shader instancing* and is requested by including an `OpExecutionMode` instruction with `mode` specified as `Invocations` and the number of invocations specified as an integer literal.

In this mode, the geometry shader will execute at least n times for each input primitive, where n is the number of invocations specified in the `OpExecutionMode` instruction. The instance number is available to each invocation as a built-in input using `InvocationId`.

## 24.4. Geometry Shader Primitive Ordering

Limited guarantees are provided for the relative ordering of primitives produced by a geometry shader, as they pertain to primitive order.

- For instanced geometry shaders, the output primitives generated from each input primitive are passed to subsequent pipeline stages using the invocation number to order the primitives, from least to greatest.
- All output primitives generated from a given input primitive are passed to subsequent pipeline stages before any output primitives generated from subsequent input primitives.

## 24.5. Geometry Shader Passthrough

A geometry shader that uses the `PassthroughNV` decoration on a variable in its input interface is considered a *passthrough geometry shader*. Output primitives in a passthrough geometry shader **must** have the same topology as the input primitive and are not produced by emitting vertices. The vertices of the output primitive have two different types of attributes, per-vertex and per-primitive. Geometry shader input variables with `PassthroughNV` decoration are considered to produce per-

vertex outputs, where values for each output vertex are copied from the corresponding input vertex. Any built-in or user-defined geometry shader outputs are considered per-primitive in a passthrough geometry shader, where a single output value is copied to all output vertices.

The remainder of this section details the usage of the PassthroughNV decoration and modifications to the interface matching rules when using passthrough geometry shaders.

### 24.5.1. PassthroughNV Decoration

Decorating a geometry shader input variable with the PassthroughNV decoration indicates that values of this input are copied through to the corresponding vertex of the output primitive. Input variables and block members which do not have the PassthroughNV decoration are consumed by the geometry shader without being passed through to subsequent stages.

The PassthroughNV decoration **must** only be used within a geometry shader.

Any variable decorated with PassthroughNV **must** be declared using the Input storage class.

The PassthroughNV decoration **must** not be used with any of:

- an input primitive type other than InputPoints, InputLines, or Triangles, as specified by the mode for OpExecutionMode.
- an invocation count other than one, as specified by the Invocations mode for OpExecutionMode.
- an OpEntryPoint which statically uses the OpEmitVertex or OpEndPrimitive instructions.
- a variable decorated with the InvocationId built-in decoration.
- a variable decorated with the PrimitiveId built-in decoration that is declared using the Input storage class.

### 24.5.2. Passthrough Interface Matching

When a passthrough geometry shader is in use, the Interface Matching rules involving the geometry shader input and output interfaces operate as described in this section.

For the purposes of matching passthrough geometry shader inputs with outputs of the previous pipeline stages, the PassthroughNV decoration is ignored.

For the purposes of matching the outputs of the geometry shader with subsequent pipeline stages, each input variable with the PassthroughNV decoration is considered to add an equivalent output variable with the same type, decoration (other than PassthroughNV), number, and declaration order on the output interface. The output variable declaration corresponding to an input variable decorated with PassthroughNV will be identical to the input declaration, except that the outermost array dimension of such variables is removed. The output block declaration corresponding to an input block decorated with PassthroughNV or having members decorated with PassthroughNV will be identical to the input declaration, except that the outermost array dimension of such declaration is removed.

If an input block is decorated with PassthroughNV, the equivalent output block contains all the members of the input block. Otherwise, the equivalent output block contains only those input block

members decorated with `PassthroughNV`. All members of the corresponding output block are assigned `Location` and `Component` decorations identical to those assigned to the corresponding input block members.

Output variables and blocks generated from inputs decorated with `PassthroughNV` will only exist for the purposes of interface matching; these declarations are not available to geometry shader code or listed in the module interface.

For the purposes of component counting, passthrough geometry shaders count all statically used input variable components declared with the `PassthroughNV` decoration as output components as well, since their values will be copied to the output primitive produced by the geometry shader.

# Chapter 25. Mesh Shading

Task and mesh shaders operate in workgroups to produce a collection of primitives that will be processed by subsequent stages of the graphics pipeline.

Work on the mesh pipeline is initiated by the application drawing a set of mesh tasks organized in global workgroups. If the optional task shader is active, each workgroup triggers the execution of task shader invocations that will create a new set of mesh workgroups upon completion. Each of these created workgroups, or each of the original workgroups if no task shader is present, triggers the execution of mesh shader invocations.

Each mesh shader workgroup emits zero or more output primitives along with the group of vertices and their associated data required for each output primitive.

## 25.1. Task Shader Input

For every workgroup issued via the drawing commands a group of task shader invocations is executed. There are no inputs other than the builtin workgroup identifiers.

## 25.2. Task Shader Output

The task shader can emit zero or more mesh workgroups to be generated. Shaders using the `TaskNV Execution Model` can do so using the built-in variable `TaskCountNV`. This value **must** be less than or equal to `VkPhysicalDeviceMeshShaderPropertiesNV::maxTaskOutputCount`. Shaders using the `TaskEXT Execution Model` can do so using the `OpEmitMeshTasksEXT` instruction. The `groupCountX`, `groupCountY` and `groupCountZ` arguments passed to this instruction **must** be less than or equal to the respective dimension within `VkPhysicalDeviceMeshShaderPropertiesEXT::maxMeshWorkGroupCount`. The product of these arguments **must** be less than or equal to `VkPhysicalDeviceMeshShaderPropertiesEXT::maxMeshWorkGroupTotalCount`.

The task shader can also pass user-defined data to all mesh shader invocations that it creates. Shaders using the `TaskNV Execution Model` can do so by writing to output variables that are decorated with `PerTaskNV`. They are available as inputs in mesh shaders. Shaders using the `TaskEXT Execution Model` can do so by writing to a payload variable with `TaskPayloadWorkgroupEXT` storage class that is passed to the `OpEmitMeshTasksEXT` instruction.

## 25.3. Mesh Generation

If a task shader exists, the mesh assembler creates a variable amount of mesh workgroups depending on each task's output. If there is no task shader, the drawing commands emit the mesh shader invocations directly.

## 25.4. Mesh Shader Input

The only inputs available to the mesh shader are variables identifying the specific workgroup and invocation and, if applicable, any outputs written as `PerTaskNV` or the payload variable passed to the

OpEmitMeshTasksEXT instruction by the task shader that spawned the mesh shader's workgroup. The mesh shader can operate without a task shader as well.

## 25.5. Mesh Shader Output

A mesh shader generates primitives in one of three output modes: points, lines, or triangles. For shaders using the MeshNV Execution Model the primitive mode is specified in the shader using an OpExecutionMode instruction with the OutputPoints, OutputLinesNV, or OutputTrianglesNV modes, respectively. For shaders using the MeshEXT Execution Model the primitive mode is specified in the shader using an OpExecutionMode instruction with the OutputPoints, OutputLinesEXT, or OutputTrianglesEXT modes, respectively. Each mesh shader **must** include exactly one output primitive mode.

For shaders using the MeshNV Execution Model the maximum output vertex count is specified as a literal in the shader using an OpExecutionMode instruction with the mode set to OutputVertices and **must** be less than or equal to VkPhysicalDeviceMeshShaderPropertiesNV::maxMeshOutputVertices. For shaders using the MeshEXT Execution Model the maximum output vertex count is specified as a literal in the shader using an OpExecutionMode instruction with the mode set to OutputVertices and **must** be less than or equal to VkPhysicalDeviceMeshShaderPropertiesEXT::maxMeshOutputVertices.

For shaders using the MeshNV Execution Model the maximum output primitive count is specified as a literal in the shader using an OpExecutionMode instruction with the mode set to OutputPrimitivesNV and **must** be less than or equal to VkPhysicalDeviceMeshShaderPropertiesNV::maxMeshOutputPrimitives. For shaders using the MeshEXT Execution Model the maximum output primitive count is specified as a literal in the shader using an OpExecutionMode instruction with the mode set to OutputPrimitivesEXT, and **must** be less than or equal to VkPhysicalDeviceMeshShaderPropertiesEXT::maxMeshOutputPrimitives.

For shaders using the MeshNV Execution Model the number of primitives output by the mesh shader is provided via writing to the built-in variable PrimitiveCountNV and **must** be less than or equal to the maximum output primitive count specified in the shader. A variable decorated with PrimitiveIndicesNV is an output array of local index values into the vertex output arrays from which primitives are assembled according to the output primitive type. For shaders using the MeshEXT Execution Model the number of vertices and primitives output by the mesh shader is provided via calling the OpSetMeshOutputsEXT instruction. The vertexCount argument **must** be less than or equal to the maximum output vertex count specified in the shader. The primitiveCount argument **must** be less than or equal to the maximum output primitive count specified in the shader.

Depending on the output primitive mode an appropriately-decorated variable is the output array of local index values into the vertex output arrays from which primitives are assembled according to the output primitive type:

- OutputPoints uses the PrimitivePointIndicesEXT decoration.
- OutputLinesEXT uses the PrimitiveLineIndicesEXT decoration.
- OutputTrianglesEXT uses the PrimitiveTriangleIndicesEXT decoration.

These resulting primitives are then further processed as described in Rasterization.

With the exception of primitive indices, all output built-ins and custom attributes count towards the total storage size occupied by output variables in mesh shaders. This size can be calculated as follows, taking into account the fact that the number of effective scalar attributes is 4 times the number of effective locations used according to the location assignment rules. Let $v$ be the number of views, $n_v$ be the number of effective scalar per-vertex attributes not dependent on `ViewIndex`, $n_{vpv}$ be the number of effective scalar per-vertex attributes dependent on `ViewIndex`, $m_v$ be the maximum number of vertices specified by the `OutputVertices` `Execution Mode`, $g_v$ be `meshOutputPerVertexGranularity`, $n_p$ be the number of effective scalar per-primitive attributes not dependent on `ViewIndex`, $n_{ppv}$ be the number of effective scalar per-primitive attributes dependent on `ViewIndex`, $m_p$ be the maximum number of primitives specified by the `OutputPrimitivesEXT` `Execution Mode` and $g_p$ be `meshOutputPerPrimitiveGranularity`:

$$(n_v + (n_{vpv} \times v)) \times 4 \times \mathrm{align}(m_v, g_v) + (n_p + (n_{ppv} \times v)) \times 4 \times \mathrm{align}(m_p, g_p)$$

## 25.6. Mesh Shader Per-View Outputs

The mesh shader outputs decorated with the `PositionPerViewNV`, `ClipDistancePerViewNV`, `CullDistancePerViewNV`, `LayerPerViewNV`, and `ViewportMaskPerViewNV` built-in decorations are the per-view versions of the single-view variables with equivalent names (that is `Position`, `ClipDistance`, `CullDistance`, `Layer`, and `ViewportMaskNV`, respectively). If a shader statically assigns a value to any element of a per-view array it **must** not statically assign a value to the equivalent single-view variable.

Each of these outputs is considered arrayed, with separate values for each view. The view number is used to index the first dimension of these arrays.

The second dimension of the `ClipDistancePerViewNV`, and `CullDistancePerViewNV` arrays have the same requirements as the `ClipDistance`, and `CullDistance` arrays.

If a mesh shader output is *per-view*, the corresponding fragment shader input is taken from the element of the per-view output array that corresponds to the view that is currently being processed by the fragment shader.

These *per-view* outputs are available only in shaders using the `MeshNV` `Execution Model`. They are not available in shaders using the `MeshEXT` `Execution Model`.

## 25.7. Mesh Shader Primitive Ordering

Following guarantees are provided for the relative ordering of primitives produced by a mesh shader, as they pertain to primitive order.

- When a task shader is used, mesh workgroups spawned from lower tasks will be ordered prior those workgroups from subsequent tasks.
- All output primitives generated from a given mesh workgroup are passed to subsequent pipeline stages before any output primitives generated from subsequent input workgroups.
- All output primitives within a mesh workgroup, will be generated in the ordering provided by the builtin primitive indexbuffer (from low address to high address).

# Chapter 26. Cluster Culling Shading

This shader type has an execution environment similar to that of a compute shader, where a collection of shader invocations form a workgroup and cooperate to perform coarse level geometry culling and LOD selection. A shader invocation can emit a set of built-in output variables via a new built-in function. The cluster culling shader organizes these emitted variables into a drawing command used by the subsequent rendering pipeline.

## 26.1. Cluster Culling Shader Input

The only inputs available to the cluster culling shader are variables identifying the specific workgroup and invocation.

## 26.2. Cluster Culling Shader Output

If a cluster survives after culling in a cluster culling shader invocation, a drawing command to draw this cluster should be emitted by this shader invocation for further rendering processing. There are two types of drawing command, indexed mode and non-indexed mode. Both type of drawing commands consist of a set of built-in output variables which have a similar definition to VkDrawIndexedIndirectCommand and VkDrawIndirectCommand members.

Cluster culling shaders have the following built-in output variables:

- built-in variable `IndexCountHUAWEI` is the number of vertices to draw.
- built-in variable `VertexCountHUAWEI` is the number of vertices to draw.
- built-in variable `InstanceCountHUAWEI` is the number of instances to draw.
- built-in variable `FirstIndexHUAWEI` is the base index within the index buffer.
- built-in variable `FirstVertexHUAWEI` is the index of the first vertex to draw
- built-in variable `VertexOffsetHUAWEI` is the value added to the vertex index before indexing into the vertex buffer.
- built-in variable `FirstInstanceHUAWEI` is the instance ID of the first instance to draw.
- built-in variable `ClusterIDHUAWEI` is the index of cluster being rendered by this drawing command. When cluster culling shader is enabled, `ClusterIDHUAWEI` will replace `gl_DrawID` pass to vertex shader.
- built-in variable `ClusterShadingRate` is the shading rate of cluster being rendered by this drawing command.

## 26.3. Cluster Culling Shader Cluster Ordering

- When a cluster culling shader is used, all output clusters generated by `DispatchClusterHUAWEI`() in a given workgroup are passed to subsequent pipeline stage before any cluster generated from subsequent workgroup.
- In a workgroup, the order of output clusters generated by `DispatchClusterHUAWEI`() is specified by

the local invocation id, from lower to higher values.

- If any cluster culling invocation in the workgroup does not call `DispatchClusterHUAWEI`(), no cluster will be sent to the subsequent rendering pipeline.
- Any cluster culling shader invocation may also call `DispatchClusterHUAWEI`() many times as shown below:

```
// Cluster Culling Shader sample code:
        ......
    DispatchClusterHUAWEI();  // dispatch 0
        ......
    DispatchClusterHUAWEI();  // dispatch 1
        ......
    DispatchClusterHUAWEI();  // dispatch 2
        ......
```

In this case, the output sequence of clusters in a workgroup are specified as shown below ( in case of 32 shader invocations in a workgroup):

```
1. shader invocation0.dispatch0
2. shader invocation1.dispatch0,
            ..........
32. shader invocation31.dispatch0
33. shader invocation0.dispatch1
34. shader invocation1.dispatch1
            ..........
64. shader invocation31.dispatch1
65. shader invocation0.dispatch2
66. shader invocation1.dispatch2
            ..........
96. shader Invocation31.dispatch2
```

## 26.4. Cluster Culling Shader Primitive Ordering

Following guarantees are provided for the relative ordering of primitives produced by a cluster culling shader, as they pertain to primitive order.

- Limited guarantees are provided for the relative ordering of primitives produced by a cluster culling shader, as they pertain to primitive order.
- The order of primitives in a given cluster is specified by the content of
  - `DispatchClusterHUAWEI`() with indexed output built-in variables, vertices sourced from a lower index buffer addresses to higher addresses.
  - `DispatchClusterHUAWEI`() with non-indexed output built-in variables, from vertices with a lower numbered vertexIndex to a higher numbered vertexIndex.

# Chapter 27. Fixed-Function Vertex Post-Processing

After pre-rasterization shader stages, the following fixed-function operations are applied to vertices of the resulting primitives:

- Transform feedback (see Transform Feedback)
- Viewport swizzle (see Viewport Swizzle)
- Flat shading (see Flat Shading).
- Primitive clipping, including application-defined half-spaces (see Primitive Clipping).
- Shader output attribute clipping (see Clipping Shader Outputs).
- Clip space W scaling (see Controlling Viewport W Scaling).
- Perspective division on clip coordinates (see Coordinate Transformations).
- Viewport mapping, including depth range scaling (see Controlling the Viewport).
- Front face determination for polygon primitives (see Basic Polygon Rasterization).

Next, rasterization is performed on primitives as described in chapter Rasterization.

## 27.1. Transform Feedback

Before any other fixed-function vertex post-processing, vertex outputs from the last shader in the pre-rasterization shader stage **can** be written out to one or more transform feedback buffers bound to the command buffer. To capture vertex outputs the last pre-rasterization shader stage shader **must** be declared with the `Xfb` execution mode. Outputs decorated with `XfbBuffer` will be written out to the corresponding transform feedback buffers bound to the command buffer when transform feedback is active. Transform feedback buffers are bound to the command buffer by using vkCmdBindTransformFeedbackBuffersEXT. Transform feedback is made active by calling vkCmdBeginTransformFeedbackEXT and made inactive by calling vkCmdEndTransformFeedbackEXT. After vertex data is written it is possible to use vkCmdDrawIndirectByteCountEXT to start a new draw where the `vertexCount` is derived from the number of bytes written by a previous transform feedback.

When an individual point, line, or triangle primitive reaches the transform feedback stage while transform feedback is active, the values of the specified output variables are assembled into primitives and appended to the bound transform feedback buffers. After activating transform feedback, the values of the first assembled primitive are written at the starting offsets of the bound transform feedback buffers, and subsequent primitives are appended to the buffer. If the optional `pCounterBuffers` and `pCounterBufferOffsets` parameters are specified, the starting points within the transform feedback buffers are adjusted so data is appended to the previously written values indicated by the value stored by the implementation in the counter buffer.

For multi-vertex primitives, all values for a given vertex are written before writing values for any other vertex. When `transformFeedbackPreservesProvokingVertex` is not enabled, implementations **may** write out any vertex within the primitive first, but all subsequent vertices for that primitive

**must** be written out in a consistent winding order defined as follows:

- If neither geometry or tessellation shading is active, vertices within a primitive are appended according to the winding order described by the primitive topology defined by the VkPipelineInputAssemblyStateCreateInfo:`topology` used to execute the drawing command.
- If geometry shading is active, vertices within a primitive are appended according to the winding order described by the primitive topology defined by the `OutputPoints`, `OutputLineStrip`, or `OutputTriangleStrip` execution mode.
- If tessellation shading is active but geometry shading is not, vertices within a primitive are appended according to the winding order defined by triangle tessellation, quad tessellation, and isoline tessellation.

When `transformFeedbackPreservesProvokingVertex` is enabled, then in addition to writing vertices with a consistent winding order, the vertex order **must** preserve the provoking vertex of each primitive:

- When the pipeline's provoking vertex mode is `VK_PROVOKING_VERTEX_MODE_FIRST_VERTEX_EXT`, the primitive's provoking vertex **must** be the first vertex written.
- When the pipeline's provoking vertex mode is `VK_PROVOKING_VERTEX_MODE_LAST_VERTEX_EXT`, the primitive's provoking vertex **must** be the last vertex written.

If `transformFeedbackPreservesTriangleFanProvokingVertex` is `VK_FALSE`, neither geometry nor tessellation shading is active, and the primitive topology is `VK_PRIMITIVE_TOPOLOGY_TRIANGLE_FAN`, then the first vertex written from each primitive is implementation-defined even when `transformFeedbackPreservesProvokingVertex` is enabled.

When capturing vertices, the stride associated with each transform feedback buffer, as indicated by the `XfbStride` decoration, indicates the number of bytes of storage reserved for each vertex in the transform feedback buffer. For every vertex captured, each output attribute with a `Offset` decoration will be written to the storage reserved for the vertex at the associated transform feedback buffer. When writing output variables that are arrays or structures, individual array elements or structure members are written tightly packed in order. For vector types, individual components are written in order. For matrix types, outputs are written as an array of column vectors.

If any component of an output with an assigned transform feedback offset was not written to by its shader, the value recorded for that component is undefined. All components of an output variable **must** be written at an offset aligned to the size of the component. The size of each component of an output variable **must** be at least 32-bits. When capturing a vertex, any portion of the reserved storage not associated with an output variable with an assigned transform feedback offset will be unmodified.

When transform feedback is inactive, no vertices are recorded. If there is a valid counter buffer handle and counter buffer offset in the `pCounterBuffers` and `pCounterBufferOffsets` arrays, writes to the corresponding transform feedback buffer will start at the byte offset represented by the value stored in the counter buffer location.

Individual lines or triangles of a strip or fan primitive will be extracted and recorded separately.

Incomplete primitives are not recorded.

When using a geometry shader that emits vertices to multiple vertex streams, a primitive will be assembled and output for each stream when there are enough vertices emitted for the output primitive type. All outputs assigned to a given transform feedback buffer are required to come from a single vertex stream.

The sizes of the transform feedback buffers are defined by the vkCmdBindTransformFeedbackBuffersEXT `pSizes` parameter for each of the bound buffers, or the size of the bound buffer, whichever is the lesser. If there is less space remaining in any of the transform feedback buffers than the size of all of the vertex data for that primitive based on the `XfbStride` for that `XfbBuffer` then no vertex data of that primitive is recorded in any transform feedback buffer, and the value for the number of primitives written in the corresponding `VK_QUERY_TYPE_TRANSFORM_FEEDBACK_STREAM_EXT` query for all transform feedback buffers is no longer incremented.

Any outputs made to a `XfbBuffer` that is not bound to a transform feedback buffer is ignored.

To bind transform feedback buffers to a command buffer for use in subsequent drawing commands, call:

```
// Provided by VK_EXT_transform_feedback
void vkCmdBindTransformFeedbackBuffersEXT(
    VkCommandBuffer                             commandBuffer,
    uint32_t                                    firstBinding,
    uint32_t                                    bindingCount,
    const VkBuffer*                             pBuffers,
    const VkDeviceSize*                         pOffsets,
    const VkDeviceSize*                         pSizes);
```

- `commandBuffer` is the command buffer into which the command is recorded.
- `firstBinding` is the index of the first transform feedback binding whose state is updated by the command.
- `bindingCount` is the number of transform feedback bindings whose state is updated by the command.
- `pBuffers` is a pointer to an array of buffer handles.
- `pOffsets` is a pointer to an array of buffer offsets.
- `pSizes` is `NULL` or a pointer to an array of VkDeviceSize buffer sizes, specifying the maximum number of bytes to capture to the corresponding transform feedback buffer. If `pSizes` is `NULL`, or the value of the `pSizes` array element is `VK_WHOLE_SIZE`, then the maximum number of bytes captured will be the size of the corresponding buffer minus the buffer offset.

The values taken from elements i of `pBuffers`, `pOffsets` and `pSizes` replace the current state for the transform feedback binding `firstBinding` + i, for i in [0, `bindingCount`). The transform feedback binding is updated to start at the offset indicated by `pOffsets`[i] from the start of the buffer `pBuffers`[i].

## Valid Usage

- VUID-vkCmdBindTransformFeedbackBuffersEXT-transformFeedback-02355
  `VkPhysicalDeviceTransformFeedbackFeaturesEXT::transformFeedback` **must** be enabled
- VUID-vkCmdBindTransformFeedbackBuffersEXT-firstBinding-02356
  `firstBinding` **must** be less than `VkPhysicalDeviceTransformFeedbackPropertiesEXT::maxTransformFeedbackBuffers`
- VUID-vkCmdBindTransformFeedbackBuffersEXT-firstBinding-02357
  The sum of `firstBinding` and `bindingCount` **must** be less than or equal to `VkPhysicalDeviceTransformFeedbackPropertiesEXT::maxTransformFeedbackBuffers`
- VUID-vkCmdBindTransformFeedbackBuffersEXT-pOffsets-02358
  All elements of `pOffsets` **must** be less than the size of the corresponding element in `pBuffers`
- VUID-vkCmdBindTransformFeedbackBuffersEXT-pOffsets-02359
  All elements of `pOffsets` **must** be a multiple of 4
- VUID-vkCmdBindTransformFeedbackBuffersEXT-pBuffers-02360
  All elements of `pBuffers` **must** have been created with the `VK_BUFFER_USAGE_TRANSFORM_FEEDBACK_BUFFER_BIT_EXT` flag
- VUID-vkCmdBindTransformFeedbackBuffersEXT-pSize-02361
  If the optional `pSize` array is specified, each element of `pSizes` **must** either be `VK_WHOLE_SIZE`, or be less than or equal to `VkPhysicalDeviceTransformFeedbackPropertiesEXT::maxTransformFeedbackBufferSize`
- VUID-vkCmdBindTransformFeedbackBuffersEXT-pSizes-02362
  All elements of `pSizes` **must** be either `VK_WHOLE_SIZE`, or less than or equal to the size of the corresponding buffer in `pBuffers`
- VUID-vkCmdBindTransformFeedbackBuffersEXT-pOffsets-02363
  All elements of `pOffsets` plus `pSizes`, where the `pSizes`, element is not `VK_WHOLE_SIZE`, **must** be less than or equal to the size of the corresponding buffer in `pBuffers`
- VUID-vkCmdBindTransformFeedbackBuffersEXT-pBuffers-02364
  Each element of `pBuffers` that is non-sparse **must** be bound completely and contiguously to a single `VkDeviceMemory` object
- VUID-vkCmdBindTransformFeedbackBuffersEXT-None-02365
  Transform feedback **must** not be active when the `vkCmdBindTransformFeedbackBuffersEXT` command is recorded

## Valid Usage (Implicit)

- VUID-vkCmdBindTransformFeedbackBuffersEXT-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdBindTransformFeedbackBuffersEXT-pBuffers-parameter
  `pBuffers` **must** be a valid pointer to an array of `bindingCount` valid VkBuffer handles

- VUID-vkCmdBindTransformFeedbackBuffersEXT-pOffsets-parameter
  `pOffsets` **must** be a valid pointer to an array of `bindingCount` VkDeviceSize values
- VUID-vkCmdBindTransformFeedbackBuffersEXT-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdBindTransformFeedbackBuffersEXT-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- VUID-vkCmdBindTransformFeedbackBuffersEXT-videocoding
  This command **must** only be called outside of a video coding scope
- VUID-vkCmdBindTransformFeedbackBuffersEXT-bindingCount-arraylength
  `bindingCount` **must** be greater than `0`
- VUID-vkCmdBindTransformFeedbackBuffersEXT-commonparent
  Both of `commandBuffer`, and the elements of `pBuffers` **must** have been created, allocated, or retrieved from the same VkDevice

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics | State |

Transform feedback for specific transform feedback buffers is made active by calling:

```
// Provided by VK_EXT_transform_feedback
void vkCmdBeginTransformFeedbackEXT(
    VkCommandBuffer                             commandBuffer,
    uint32_t                                    firstCounterBuffer,
    uint32_t                                    counterBufferCount,
    const VkBuffer*                             pCounterBuffers,
    const VkDeviceSize*                         pCounterBufferOffsets);
```

- `commandBuffer` is the command buffer into which the command is recorded.
- `firstCounterBuffer` is the index of the first transform feedback buffer corresponding to `pCounterBuffers`[0] and `pCounterBufferOffsets`[0].

- `counterBufferCount` is the size of the `pCounterBuffers` and `pCounterBufferOffsets` arrays.
- `pCounterBuffers` is `NULL` or a pointer to an array of VkBuffer handles to counter buffers. Each buffer contains a 4 byte integer value representing the byte offset from the start of the corresponding transform feedback buffer from where to start capturing vertex data. If the byte offset stored to the counter buffer location was done using vkCmdEndTransformFeedbackEXT it can be used to resume transform feedback from the previous location. If `pCounterBuffers` is `NULL`, then transform feedback will start capturing vertex data to byte offset zero in all bound transform feedback buffers. For each element of `pCounterBuffers` that is VK_NULL_HANDLE, transform feedback will start capturing vertex data to byte zero in the corresponding bound transform feedback buffer.
- `pCounterBufferOffsets` is `NULL` or a pointer to an array of VkDeviceSize values specifying offsets within each of the `pCounterBuffers` where the counter values were previously written. The location in each counter buffer at these offsets **must** be large enough to contain 4 bytes of data. This data is the number of bytes captured by the previous transform feedback to this buffer. If `pCounterBufferOffsets` is `NULL`, then it is assumed the offsets are zero.

The active transform feedback buffers will capture primitives emitted from the corresponding `XfbBuffer` in the bound graphics pipeline. Any `XfbBuffer` emitted that does not output to an active transform feedback buffer will not be captured.

### Valid Usage

- VUID-vkCmdBeginTransformFeedbackEXT-transformFeedback-02366
  `VkPhysicalDeviceTransformFeedbackFeaturesEXT::transformFeedback` **must** be enabled
- VUID-vkCmdBeginTransformFeedbackEXT-None-02367
  Transform feedback **must** not be active
- VUID-vkCmdBeginTransformFeedbackEXT-firstCounterBuffer-02368
  `firstCounterBuffer` **must** be less than `VkPhysicalDeviceTransformFeedbackPropertiesEXT::maxTransformFeedbackBuffers`
- VUID-vkCmdBeginTransformFeedbackEXT-firstCounterBuffer-02369
  The sum of `firstCounterBuffer` and `counterBufferCount` **must** be less than or equal to `VkPhysicalDeviceTransformFeedbackPropertiesEXT::maxTransformFeedbackBuffers`
- VUID-vkCmdBeginTransformFeedbackEXT-counterBufferCount-02607
  If `counterBufferCount` is not `0`, and `pCounterBuffers` is not `NULL`, `pCounterBuffers` **must** be a valid pointer to an array of `counterBufferCount` `VkBuffer` handles that are either valid or VK_NULL_HANDLE
- VUID-vkCmdBeginTransformFeedbackEXT-pCounterBufferOffsets-02370
  For each buffer handle in the array, if it is not VK_NULL_HANDLE it **must** reference a buffer large enough to hold 4 bytes at the corresponding offset from the `pCounterBufferOffsets` array
- VUID-vkCmdBeginTransformFeedbackEXT-pCounterBuffer-02371
  If `pCounterBuffer` is `NULL`, then `pCounterBufferOffsets` **must** also be `NULL`
- VUID-vkCmdBeginTransformFeedbackEXT-pCounterBuffers-02372
  For each buffer handle in the `pCounterBuffers` array that is not VK_NULL_HANDLE it **must**

have been created with a `usage` value containing `VK_BUFFER_USAGE_TRANSFORM_FEEDBACK_COUNTER_BUFFER_BIT_EXT`

- VUID-vkCmdBeginTransformFeedbackEXT-firstCounterBuffer-09630
  The sum of `firstCounterBuffer` and `counterBufferCount` **must** be less than or equal to the number of transform feedback buffers currently bound by vkCmdBindTransformFeedbackBuffersEXT
- VUID-vkCmdBeginTransformFeedbackEXT-None-06233
  If the `shaderObject` feature is not enabled, a valid graphics pipeline **must** be bound to `VK_PIPELINE_BIND_POINT_GRAPHICS`
- VUID-vkCmdBeginTransformFeedbackEXT-None-04128
  The last pre-rasterization shader stage of the bound graphics pipeline **must** have been declared with the `Xfb` execution mode
- VUID-vkCmdBeginTransformFeedbackEXT-None-02373
  Transform feedback **must** not be made active in a render pass instance with multiview enabled

## Valid Usage (Implicit)

- VUID-vkCmdBeginTransformFeedbackEXT-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdBeginTransformFeedbackEXT-pCounterBufferOffsets-parameter
  If `counterBufferCount` is not `0`, and `pCounterBufferOffsets` is not `NULL`, `pCounterBufferOffsets` **must** be a valid pointer to an array of `counterBufferCount` VkDeviceSize values
- VUID-vkCmdBeginTransformFeedbackEXT-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdBeginTransformFeedbackEXT-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- VUID-vkCmdBeginTransformFeedbackEXT-renderpass
  This command **must** only be called inside of a render pass instance
- VUID-vkCmdBeginTransformFeedbackEXT-videocoding
  This command **must** only be called outside of a video coding scope
- VUID-vkCmdBeginTransformFeedbackEXT-commonparent
  Both of `commandBuffer`, and the elements of `pCounterBuffers` that are valid handles of non-ignored parameters **must** have been created, allocated, or retrieved from the same VkDevice

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally

synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Inside | Outside | Graphics | State |

Transform feedback for specific transform feedback buffers is made inactive by calling:

```
// Provided by VK_EXT_transform_feedback
void vkCmdEndTransformFeedbackEXT(
    VkCommandBuffer                             commandBuffer,
    uint32_t                                    firstCounterBuffer,
    uint32_t                                    counterBufferCount,
    const VkBuffer*                             pCounterBuffers,
    const VkDeviceSize*                         pCounterBufferOffsets);
```

- `commandBuffer` is the command buffer into which the command is recorded.
- `firstCounterBuffer` is the index of the first transform feedback buffer corresponding to `pCounterBuffers`[0] and `pCounterBufferOffsets`[0].
- `counterBufferCount` is the size of the `pCounterBuffers` and `pCounterBufferOffsets` arrays.
- `pCounterBuffers` is `NULL` or a pointer to an array of VkBuffer handles to counter buffers. The counter buffers are used to record the current byte positions of each transform feedback buffer where the next vertex output data would be captured. This **can** be used by a subsequent vkCmdBeginTransformFeedbackEXT call to resume transform feedback capture from this position. It can also be used by vkCmdDrawIndirectByteCountEXT to determine the vertex count of the draw call.
- `pCounterBufferOffsets` is `NULL` or a pointer to an array of VkDeviceSize values specifying offsets within each of the `pCounterBuffers` where the counter values can be written. The location in each counter buffer at these offsets **must** be large enough to contain 4 bytes of data. The data stored at this location is the byte offset from the start of the transform feedback buffer binding where the next vertex data would be written. If `pCounterBufferOffsets` is `NULL`, then it is assumed the offsets are zero.

## Valid Usage

- VUID-vkCmdEndTransformFeedbackEXT-transformFeedback-02374
  `VkPhysicalDeviceTransformFeedbackFeaturesEXT::transformFeedback` **must** be enabled
- VUID-vkCmdEndTransformFeedbackEXT-None-02375
  Transform feedback **must** be active

- VUID-vkCmdEndTransformFeedbackEXT-firstCounterBuffer-02376
  `firstCounterBuffer` **must** be less than `VkPhysicalDeviceTransformFeedbackPropertiesEXT::maxTransformFeedbackBuffers`
- VUID-vkCmdEndTransformFeedbackEXT-firstCounterBuffer-02377
  The sum of `firstCounterBuffer` and `counterBufferCount` **must** be less than or equal to `VkPhysicalDeviceTransformFeedbackPropertiesEXT::maxTransformFeedbackBuffers`
- VUID-vkCmdEndTransformFeedbackEXT-counterBufferCount-02608
  If `counterBufferCount` is not `0`, and `pCounterBuffers` is not `NULL`, `pCounterBuffers` **must** be a valid pointer to an array of `counterBufferCount` `VkBuffer` handles that are either valid or VK_NULL_HANDLE
- VUID-vkCmdEndTransformFeedbackEXT-pCounterBufferOffsets-02378
  For each buffer handle in the array, if it is not VK_NULL_HANDLE it **must** reference a buffer large enough to hold 4 bytes at the corresponding offset from the `pCounterBufferOffsets` array
- VUID-vkCmdEndTransformFeedbackEXT-pCounterBuffer-02379
  If `pCounterBuffer` is `NULL`, then `pCounterBufferOffsets` **must** also be `NULL`
- VUID-vkCmdEndTransformFeedbackEXT-pCounterBuffers-02380
  For each buffer handle in the `pCounterBuffers` array that is not VK_NULL_HANDLE it **must** have been created with a `usage` value containing `VK_BUFFER_USAGE_TRANSFORM_FEEDBACK_COUNTER_BUFFER_BIT_EXT`

## Valid Usage (Implicit)

- VUID-vkCmdEndTransformFeedbackEXT-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdEndTransformFeedbackEXT-pCounterBufferOffsets-parameter
  If `counterBufferCount` is not `0`, and `pCounterBufferOffsets` is not `NULL`, `pCounterBufferOffsets` **must** be a valid pointer to an array of `counterBufferCount` VkDeviceSize values
- VUID-vkCmdEndTransformFeedbackEXT-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdEndTransformFeedbackEXT-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- VUID-vkCmdEndTransformFeedbackEXT-renderpass
  This command **must** only be called inside of a render pass instance
- VUID-vkCmdEndTransformFeedbackEXT-videocoding
  This command **must** only be called outside of a video coding scope
- VUID-vkCmdEndTransformFeedbackEXT-commonparent
  Both of `commandBuffer`, and the elements of `pCounterBuffers` that are valid handles of non-ignored parameters **must** have been created, allocated, or retrieved from the same VkDevice

### Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

### Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Inside | Outside | Graphics | State |

## 27.2. Viewport Swizzle

Each primitive sent to a given viewport has a swizzle and **optional** negation applied to its clip coordinates. The swizzle that is applied depends on the viewport index, and is controlled by the `VkPipelineViewportSwizzleStateCreateInfoNV` pipeline state:

```
// Provided by VK_NV_viewport_swizzle
typedef struct VkPipelineViewportSwizzleStateCreateInfoNV {
    VkStructureType                                 sType;
    const void*                                     pNext;
    VkPipelineViewportSwizzleStateCreateFlagsNV     flags;
    uint32_t                                        viewportCount;
    const VkViewportSwizzleNV*                      pViewportSwizzles;
} VkPipelineViewportSwizzleStateCreateInfoNV;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is reserved for future use.
- `viewportCount` is the number of viewport swizzles used by the pipeline.
- `pViewportSwizzles` is a pointer to an array of VkViewportSwizzleNV structures, defining the viewport swizzles.

### Valid Usage

- VUID-VkPipelineViewportSwizzleStateCreateInfoNV-viewportCount-01215
  `viewportCount` **must** be greater than or equal to the `viewportCount` set in `VkPipelineViewportStateCreateInfo`

### Valid Usage (Implicit)

- VUID-VkPipelineViewportSwizzleStateCreateInfoNV-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PIPELINE_VIEWPORT_SWIZZLE_STATE_CREATE_INFO_NV`
- VUID-VkPipelineViewportSwizzleStateCreateInfoNV-flags-zerobitmask
  `flags` **must** be `0`
- VUID-VkPipelineViewportSwizzleStateCreateInfoNV-pViewportSwizzles-parameter
  `pViewportSwizzles` **must** be a valid pointer to an array of `viewportCount` valid VkViewportSwizzleNV structures
- VUID-VkPipelineViewportSwizzleStateCreateInfoNV-viewportCount-arraylength
  `viewportCount` **must** be greater than `0`

```
// Provided by VK_NV_viewport_swizzle
typedef VkFlags VkPipelineViewportSwizzleStateCreateFlagsNV;
```

`VkPipelineViewportSwizzleStateCreateFlagsNV` is a bitmask type for setting a mask, but is currently reserved for future use.

The `VkPipelineViewportSwizzleStateCreateInfoNV` state is set by adding this structure to the `pNext` chain of a `VkPipelineViewportStateCreateInfo` structure and setting the graphics pipeline state with vkCreateGraphicsPipelines.

To dynamically set the viewport swizzle state, call:

```
// Provided by VK_EXT_extended_dynamic_state3 with VK_NV_viewport_swizzle,
VK_EXT_shader_object with VK_NV_viewport_swizzle
void vkCmdSetViewportSwizzleNV(
    VkCommandBuffer                             commandBuffer,
    uint32_t                                    firstViewport,
    uint32_t                                    viewportCount,
    const VkViewportSwizzleNV*                  pViewportSwizzles);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `firstViewport` is the index of the first viewport whose parameters are updated by the command.
- `viewportCount` is the number of viewports whose parameters are updated by the command.
- `pViewportSwizzles` is a pointer to an array of VkViewportSwizzleNV structures specifying viewport swizzles.

This command sets the viewport swizzle state for subsequent drawing commands when drawing using shader objects, or when the graphics pipeline is created with `VK_DYNAMIC_STATE_VIEWPORT_SWIZZLE_NV` set in VkPipelineDynamicStateCreateInfo::`pDynamicStates`. Otherwise, this state is specified by the VkPipelineViewportSwizzleStateCreateInfoNV::`viewportCount`, and VkPipelineViewportSwizzleStateCreateInfoNV::`pViewportSwizzles` values used

to create the currently active pipeline.

## Valid Usage

- VUID-vkCmdSetViewportSwizzleNV-None-09423
  At least one of the following **must** be true:
  - The `extendedDynamicState3ViewportSwizzle` feature is enabled
  - The `shaderObject` feature is enabled

## Valid Usage (Implicit)

- VUID-vkCmdSetViewportSwizzleNV-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdSetViewportSwizzleNV-pViewportSwizzles-parameter
  `pViewportSwizzles` **must** be a valid pointer to an array of `viewportCount` valid VkViewportSwizzleNV structures
- VUID-vkCmdSetViewportSwizzleNV-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdSetViewportSwizzleNV-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- VUID-vkCmdSetViewportSwizzleNV-videocoding
  This command **must** only be called outside of a video coding scope
- VUID-vkCmdSetViewportSwizzleNV-viewportCount-arraylength
  `viewportCount` **must** be greater than `0`

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics | State |

Each viewport specified from 0 to `viewportCount` - 1 has its x,y,z,w swizzle state set to the

corresponding x, y, z and w in the VkViewportSwizzleNV structure. Each component is of type VkViewportCoordinateSwizzleNV, which determines the type of swizzle for that component. The value of x computes the new x component of the position as:

```
if (x == VK_VIEWPORT_COORDINATE_SWIZZLE_POSITIVE_X_NV) x' = x;
if (x == VK_VIEWPORT_COORDINATE_SWIZZLE_NEGATIVE_X_NV) x' = -x;
if (x == VK_VIEWPORT_COORDINATE_SWIZZLE_POSITIVE_Y_NV) x' = y;
if (x == VK_VIEWPORT_COORDINATE_SWIZZLE_NEGATIVE_Y_NV) x' = -y;
if (x == VK_VIEWPORT_COORDINATE_SWIZZLE_POSITIVE_Z_NV) x' = z;
if (x == VK_VIEWPORT_COORDINATE_SWIZZLE_NEGATIVE_Z_NV) x' = -z;
if (x == VK_VIEWPORT_COORDINATE_SWIZZLE_POSITIVE_W_NV) x' = w;
if (x == VK_VIEWPORT_COORDINATE_SWIZZLE_NEGATIVE_W_NV) x' = -w;
```

Similar selections are performed for the y, z, and w coordinates. This swizzling is applied before clipping and perspective divide. If the swizzle for an active viewport index is not specified, the swizzle for x is VK_VIEWPORT_COORDINATE_SWIZZLE_POSITIVE_X_NV, y is VK_VIEWPORT_COORDINATE_SWIZZLE_POSITIVE_Y_NV, z is VK_VIEWPORT_COORDINATE_SWIZZLE_POSITIVE_Z_NV and w is VK_VIEWPORT_COORDINATE_SWIZZLE_POSITIVE_W_NV.

Viewport swizzle parameters are specified by setting the pNext pointer of VkGraphicsPipelineCreateInfo to point to a VkPipelineViewportSwizzleStateCreateInfoNV structure. VkPipelineViewportSwizzleStateCreateInfoNV uses VkViewportSwizzleNV to set the viewport swizzle parameters.

The VkViewportSwizzleNV structure is defined as:

```
// Provided by VK_NV_viewport_swizzle
typedef struct VkViewportSwizzleNV {
    VkViewportCoordinateSwizzleNV    x;
    VkViewportCoordinateSwizzleNV    y;
    VkViewportCoordinateSwizzleNV    z;
    VkViewportCoordinateSwizzleNV    w;
} VkViewportSwizzleNV;
```

- x is a VkViewportCoordinateSwizzleNV value specifying the swizzle operation to apply to the x component of the primitive
- y is a VkViewportCoordinateSwizzleNV value specifying the swizzle operation to apply to the y component of the primitive
- z is a VkViewportCoordinateSwizzleNV value specifying the swizzle operation to apply to the z component of the primitive
- w is a VkViewportCoordinateSwizzleNV value specifying the swizzle operation to apply to the w component of the primitive

### Valid Usage (Implicit)

- VUID-VkViewportSwizzleNV-x-parameter

x **must** be a valid VkViewportCoordinateSwizzleNV value

- VUID-VkViewportSwizzleNV-y-parameter
  y **must** be a valid VkViewportCoordinateSwizzleNV value
- VUID-VkViewportSwizzleNV-z-parameter
  z **must** be a valid VkViewportCoordinateSwizzleNV value
- VUID-VkViewportSwizzleNV-w-parameter
  w **must** be a valid VkViewportCoordinateSwizzleNV value

Possible values of the VkViewportSwizzleNV::x, y, z, and w members, specifying swizzling of the corresponding components of primitives, are:

```
// Provided by VK_NV_viewport_swizzle
typedef enum VkViewportCoordinateSwizzleNV {
    VK_VIEWPORT_COORDINATE_SWIZZLE_POSITIVE_X_NV = 0,
    VK_VIEWPORT_COORDINATE_SWIZZLE_NEGATIVE_X_NV = 1,
    VK_VIEWPORT_COORDINATE_SWIZZLE_POSITIVE_Y_NV = 2,
    VK_VIEWPORT_COORDINATE_SWIZZLE_NEGATIVE_Y_NV = 3,
    VK_VIEWPORT_COORDINATE_SWIZZLE_POSITIVE_Z_NV = 4,
    VK_VIEWPORT_COORDINATE_SWIZZLE_NEGATIVE_Z_NV = 5,
    VK_VIEWPORT_COORDINATE_SWIZZLE_POSITIVE_W_NV = 6,
    VK_VIEWPORT_COORDINATE_SWIZZLE_NEGATIVE_W_NV = 7,
} VkViewportCoordinateSwizzleNV;
```

These values are described in detail in Viewport Swizzle.

## 27.3. Flat Shading

*Flat shading* a vertex output attribute means to assign all vertices of the primitive the same value for that output. The output values assigned are those of the *provoking vertex* of the primitive. Flat shading is applied to those vertex attributes that match fragment input attributes which are decorated as Flat.

If neither mesh, geometry nor tessellation shading is active, the provoking vertex is determined by the primitive topology defined by VkPipelineInputAssemblyStateCreateInfo:topology used to execute the drawing command.

If a shader using MeshNV Execution Model is active, the provoking vertex is determined by the primitive topology defined by the OutputPoints, OutputLinesNV, or OutputTrianglesNV execution mode.

If a shader using MeshEXT Execution Model is active, the provoking vertex is determined by the primitive topology defined by the OutputPoints, OutputLinesEXT, or OutputTrianglesEXT execution mode.

If geometry shading is active, the provoking vertex is determined by the primitive topology defined by the OutputPoints, OutputLineStrip, or OutputTriangleStrip execution mode.

If tessellation shading is active but geometry shading is not, the provoking vertex **may** be any of the vertices in each primitive.

For a given primitive topology, the pipeline's provoking vertex mode determines which vertex is the provoking vertex. To specify the provoking vertex mode, include a VkPipelineRasterizationProvokingVertexStateCreateInfoEXT structure in the VkPipelineRasterizationStateCreateInfo::pNext chain when creating the pipeline.

The VkPipelineRasterizationProvokingVertexStateCreateInfoEXT structure is defined as:

```
// Provided by VK_EXT_provoking_vertex
typedef struct VkPipelineRasterizationProvokingVertexStateCreateInfoEXT {
    VkStructureType             sType;
    const void*                 pNext;
    VkProvokingVertexModeEXT    provokingVertexMode;
} VkPipelineRasterizationProvokingVertexStateCreateInfoEXT;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- provokingVertexMode is a VkProvokingVertexModeEXT value selecting the provoking vertex mode.

If this struct is not provided when creating the pipeline, the pipeline will use the VK_PROVOKING_VERTEX_MODE_FIRST_VERTEX_EXT mode.

If the provokingVertexModePerPipeline limit is VK_FALSE, then all pipelines bound within a render pass instance **must** have the same provokingVertexMode.

### Valid Usage

- VUID-VkPipelineRasterizationProvokingVertexStateCreateInfoEXT-provokingVertexMode-04883
  If provokingVertexMode is VK_PROVOKING_VERTEX_MODE_LAST_VERTEX_EXT, then the provokingVertexLast feature **must** be enabled

### Valid Usage (Implicit)

- VUID-VkPipelineRasterizationProvokingVertexStateCreateInfoEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PIPELINE_RASTERIZATION_PROVOKING_VERTEX_STATE_CREATE_INFO_EXT
- VUID-VkPipelineRasterizationProvokingVertexStateCreateInfoEXT-provokingVertexMode-parameter
  provokingVertexMode **must** be a valid VkProvokingVertexModeEXT value

Possible values of VkPipelineRasterizationProvokingVertexStateCreateInfoEXT::provokingVertexMode

are:

```
// Provided by VK_EXT_provoking_vertex
typedef enum VkProvokingVertexModeEXT {
    VK_PROVOKING_VERTEX_MODE_FIRST_VERTEX_EXT = 0,
    VK_PROVOKING_VERTEX_MODE_LAST_VERTEX_EXT = 1,
} VkProvokingVertexModeEXT;
```

- `VK_PROVOKING_VERTEX_MODE_FIRST_VERTEX_EXT` specifies that the provoking vertex is the first non-adjacency vertex in the list of vertices used by a primitive.
- `VK_PROVOKING_VERTEX_MODE_LAST_VERTEX_EXT` specifies that the provoking vertex is the last non-adjacency vertex in the list of vertices used by a primitive.

These modes are described more precisely in Primitive Topologies.

To dynamically set the `provokingVertexMode` state, call:

```
// Provided by VK_EXT_extended_dynamic_state3 with VK_EXT_provoking_vertex,
VK_EXT_provoking_vertex with VK_EXT_shader_object
void vkCmdSetProvokingVertexModeEXT(
    VkCommandBuffer                             commandBuffer,
    VkProvokingVertexModeEXT                    provokingVertexMode);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `provokingVertexMode` specifies the `provokingVertexMode` state.

This command sets the `provokingVertexMode` state for subsequent drawing commands when drawing using shader objects, or when the graphics pipeline is created with `VK_DYNAMIC_STATE_PROVOKING_VERTEX_MODE_EXT` set in VkPipelineDynamicStateCreateInfo::`pDynamicStates`. Otherwise, this state is specified by the VkPipelineRasterizationProvokingVertexStateCreateInfoEXT::`provokingVertexMode` value used to create the currently active pipeline.

### Valid Usage

- VUID-vkCmdSetProvokingVertexModeEXT-None-09423
  At least one of the following **must** be true:
  - The `extendedDynamicState3ProvokingVertexMode` feature is enabled
  - The `shaderObject` feature is enabled
- VUID-vkCmdSetProvokingVertexModeEXT-provokingVertexMode-07447
  If `provokingVertexMode` is `VK_PROVOKING_VERTEX_MODE_LAST_VERTEX_EXT`, then the `provokingVertexLast` feature **must** be enabled

### Valid Usage (Implicit)

- VUID-vkCmdSetProvokingVertexModeEXT-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdSetProvokingVertexModeEXT-provokingVertexMode-parameter
  `provokingVertexMode` **must** be a valid VkProvokingVertexModeEXT value
- VUID-vkCmdSetProvokingVertexModeEXT-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdSetProvokingVertexModeEXT-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- VUID-vkCmdSetProvokingVertexModeEXT-videocoding
  This command **must** only be called outside of a video coding scope

### Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

### Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics | State |

## 27.4. Primitive Clipping

Primitives are culled against the *cull volume* and then clipped to the *clip volume*. In clip coordinates, the *view volume* is defined by:

$$-w_c \leq x_c \leq w_c$$
$$-w_c \leq y_c \leq w_c$$
$$z_m \leq z_c \leq w_c$$

where if VkPipelineViewportDepthClipControlCreateInfoEXT::`negativeOneToOne` is `VK_TRUE` $z_m$ is equal to $-w_c$ otherwise $z_m$ is equal to zero.

This view volume **can** be further restricted by as many as `VkPhysicalDeviceLimits`::`maxClipDistances` application-defined half-spaces.

The cull volume is the intersection of up to VkPhysicalDeviceLimits::maxCullDistances application-defined half-spaces (if no application-defined cull half-spaces are enabled, culling against the cull volume is skipped).

A shader **must** write a single cull distance for each enabled cull half-space to elements of the CullDistance array. If the cull distance for any enabled cull half-space is negative for all of the vertices of the primitive under consideration, the primitive is discarded. Otherwise the primitive is clipped against the clip volume as defined below.

The clip volume is the intersection of up to VkPhysicalDeviceLimits::maxClipDistances application-defined half-spaces with the view volume (if no application-defined clip half-spaces are enabled, the clip volume is the view volume).

A shader **must** write a single clip distance for each enabled clip half-space to elements of the ClipDistance array. Clip half-space i is then given by the set of points satisfying the inequality

$$c_i(\mathbf{P}) \geq 0$$

where $c_i(\mathbf{P})$ is the clip distance i at point **P**. For point primitives, $c_i(\mathbf{P})$ is simply the clip distance for the vertex in question. For line and triangle primitives, per-vertex clip distances are interpolated using a weighted mean, with weights derived according to the algorithms described in sections Basic Line Segment Rasterization and Basic Polygon Rasterization, using the perspective interpolation equations.

The number of application-defined clip and cull half-spaces that are enabled is determined by the explicit size of the built-in arrays ClipDistance and CullDistance, respectively, declared as an output in the interface of the entry point of the final shader stage before clipping.

If VkPipelineRasterizationDepthClipStateCreateInfoEXT is present in the graphics pipeline state then depth clipping is disabled if VkPipelineRasterizationDepthClipStateCreateInfoEXT::depthClipEnable is VK_FALSE. Otherwise, if VkPipelineRasterizationDepthClipStateCreateInfoEXT is not present, depth clipping is disabled when VkPipelineRasterizationStateCreateInfo::depthClampEnable is VK_TRUE.

To dynamically set enable or disable depth clamping, call:

```
// Provided by VK_EXT_extended_dynamic_state3, VK_EXT_shader_object
void vkCmdSetDepthClampEnableEXT(
    VkCommandBuffer                             commandBuffer,
    VkBool32                                    depthClampEnable);
```

- commandBuffer is the command buffer into which the command will be recorded.
- depthClampEnable specifies whether depth clamping is enabled.

This command sets whether depth clamping is enabled or disabled for subsequent drawing commands when drawing using shader objects, or when the graphics pipeline is created with VK_DYNAMIC_STATE_DEPTH_CLAMP_ENABLE_EXT set in VkPipelineDynamicStateCreateInfo::pDynamicStates.

Otherwise, this state is specified by the VkPipelineRasterizationStateCreateInfo::depthClampEnable value used to create the currently active pipeline.

If the depth clamping state is changed dynamically, and the pipeline was not created with VK_DYNAMIC_STATE_DEPTH_CLIP_ENABLE_EXT enabled, then depth clipping is enabled when depth clamping is disabled and vice versa.

## Valid Usage

- VUID-vkCmdSetDepthClampEnableEXT-None-09423
  At least one of the following **must** be true:
  - The extendedDynamicState3DepthClampEnable feature is enabled
  - The shaderObject feature is enabled
- VUID-vkCmdSetDepthClampEnableEXT-depthClamp-07449
  If the depthClamp feature is not enabled, depthClampEnable **must** be VK_FALSE

## Valid Usage (Implicit)

- VUID-vkCmdSetDepthClampEnableEXT-commandBuffer-parameter
  commandBuffer **must** be a valid VkCommandBuffer handle
- VUID-vkCmdSetDepthClampEnableEXT-commandBuffer-recording
  commandBuffer **must** be in the recording state
- VUID-vkCmdSetDepthClampEnableEXT-commandBuffer-cmdpool
  The VkCommandPool that commandBuffer was allocated from **must** support graphics operations
- VUID-vkCmdSetDepthClampEnableEXT-videocoding
  This command **must** only be called outside of a video coding scope

## Host Synchronization

- Host access to commandBuffer **must** be externally synchronized
- Host access to the VkCommandPool that commandBuffer was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics | State |

To dynamically set enable or disable depth clipping, call:

```
// Provided by VK_EXT_depth_clip_enable with VK_EXT_extended_dynamic_state3,
VK_EXT_depth_clip_enable with VK_EXT_shader_object
void vkCmdSetDepthClipEnableEXT(
    VkCommandBuffer                             commandBuffer,
    VkBool32                                    depthClipEnable);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `depthClipEnable` specifies whether depth clipping is enabled.

This command sets whether depth clipping is enabled or disabled for subsequent drawing commands when drawing using shader objects, or when the graphics pipeline is created with `VK_DYNAMIC_STATE_DEPTH_CLIP_ENABLE_EXT` set in VkPipelineDynamicStateCreateInfo::`pDynamicStates`. Otherwise, this state is specified by the VkPipelineRasterizationDepthClipStateCreateInfoEXT::`depthClipEnable` value used to create the currently active pipeline, or is set to the inverse of VkPipelineRasterizationStateCreateInfo::`depthClampEnable` if `VkPipelineRasterizationDepthClipStateCreateInfoEXT` is not specified.

## Valid Usage

- VUID-vkCmdSetDepthClipEnableEXT-None-09423
  At least one of the following **must** be true:
  - The `extendedDynamicState3DepthClipEnable` feature is enabled
  - The `shaderObject` feature is enabled
- VUID-vkCmdSetDepthClipEnableEXT-depthClipEnable-07451
  The `depthClipEnable` feature **must** be enabled

## Valid Usage (Implicit)

- VUID-vkCmdSetDepthClipEnableEXT-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdSetDepthClipEnableEXT-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdSetDepthClipEnableEXT-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- VUID-vkCmdSetDepthClipEnableEXT-videocoding
  This command **must** only be called outside of a video coding scope

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics | State |

When depth clipping is disabled, the plane equation

$$z_m \leq z_c \leq w_c$$

(see the clip volume definition above) is ignored by view volume clipping (effectively, there is no near or far plane clipping).

If the primitive under consideration is a point or line segment, then clipping passes it unchanged if its vertices lie entirely within the clip volume.

Possible values of VkPhysicalDevicePointClippingProperties::`pointClippingBehavior`, specifying clipping behavior of a point primitive whose vertex lies outside the clip volume, are:

```
// Provided by VK_VERSION_1_1
typedef enum VkPointClippingBehavior {
    VK_POINT_CLIPPING_BEHAVIOR_ALL_CLIP_PLANES = 0,
    VK_POINT_CLIPPING_BEHAVIOR_USER_CLIP_PLANES_ONLY = 1,
  // Provided by VK_KHR_maintenance2
    VK_POINT_CLIPPING_BEHAVIOR_ALL_CLIP_PLANES_KHR =
VK_POINT_CLIPPING_BEHAVIOR_ALL_CLIP_PLANES,
  // Provided by VK_KHR_maintenance2
    VK_POINT_CLIPPING_BEHAVIOR_USER_CLIP_PLANES_ONLY_KHR =
VK_POINT_CLIPPING_BEHAVIOR_USER_CLIP_PLANES_ONLY,
} VkPointClippingBehavior;
```

or the equivalent

```
// Provided by VK_KHR_maintenance2
typedef VkPointClippingBehavior VkPointClippingBehaviorKHR;
```

- VK_POINT_CLIPPING_BEHAVIOR_ALL_CLIP_PLANES specifies that the primitive is discarded if the vertex lies outside any clip plane, including the planes bounding the view volume.
- VK_POINT_CLIPPING_BEHAVIOR_USER_CLIP_PLANES_ONLY specifies that the primitive is discarded only if the vertex lies outside any user clip plane.

If either of a line segment's vertices lie outside of the clip volume, the line segment **may** be clipped, with new vertex coordinates computed for each vertex that lies outside the clip volume. A clipped line segment endpoint lies on both the original line segment and the boundary of the clip volume.

This clipping produces a value, $0 \leq t \leq 1$, for each clipped vertex. If the coordinates of a clipped vertex are $\mathbf{P}$ and the unclipped line segment's vertex coordinates are $\mathbf{P}_1$ and $\mathbf{P}_2$, then t satisfies the following equation

$$\mathbf{P} = t\,\mathbf{P}_1 + (1-t)\,\mathbf{P}_2.$$

t is used to clip vertex output attributes as described in Clipping Shader Outputs.

If the primitive is a polygon, it passes unchanged if every one of its edges lies entirely inside the clip volume, and is either clipped or discarded otherwise. If the edges of the polygon intersect the boundary of the clip volume, the intersecting edges are reconnected by new edges that lie along the boundary of the clip volume - in some cases requiring the introduction of new vertices into a polygon.

If a polygon intersects an edge of the clip volume's boundary, the clipped polygon **must** include a point on this boundary edge.

Primitives rendered with application-defined half-spaces **must** satisfy a complementarity criterion. Suppose a series of primitives is drawn where each vertex i has a single specified clip distance $d_i$ (or a number of similarly specified clip distances, if multiple half-spaces are enabled). Next, suppose that the same series of primitives are drawn again with each such clip distance replaced by $-d_i$ (and the graphics pipeline is otherwise the same). In this case, primitives **must** not be missing any pixels, and pixels **must** not be drawn twice in regions where those primitives are cut by the clip planes.

The VkPipelineViewportDepthClipControlCreateInfoEXT structure is defined as:

```
// Provided by VK_EXT_depth_clip_control
typedef struct VkPipelineViewportDepthClipControlCreateInfoEXT {
    VkStructureType    sType;
    const void*        pNext;
    VkBool32           negativeOneToOne;
} VkPipelineViewportDepthClipControlCreateInfoEXT;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- negativeOneToOne sets the $z_m$ in the *view volume* to $-w_c$

### Valid Usage

- VUID-VkPipelineViewportDepthClipControlCreateInfoEXT-negativeOneToOne-06470
  If `depthClipControl` is not enabled, `negativeOneToOne` **must** be `VK_FALSE`

### Valid Usage (Implicit)

- VUID-VkPipelineViewportDepthClipControlCreateInfoEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PIPELINE_VIEWPORT_DEPTH_CLIP_CONTROL_CREATE_INFO_EXT`

To dynamically set `negativeOneToOne`, call:

```
// Provided by VK_EXT_depth_clip_control with VK_EXT_extended_dynamic_state3,
VK_EXT_depth_clip_control with VK_EXT_shader_object
void vkCmdSetDepthClipNegativeOneToOneEXT(
    VkCommandBuffer                             commandBuffer,
    VkBool32                                    negativeOneToOne);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `negativeOneToOne` specifies the `negativeOneToOne` state.

This command sets the `negativeOneToOne` state for subsequent drawing commands when drawing using shader objects, or when the graphics pipeline is created with `VK_DYNAMIC_STATE_DEPTH_CLIP_NEGATIVE_ONE_TO_ONE_EXT` set in VkPipelineDynamicStateCreateInfo::`pDynamicStates`. Otherwise, this state is specified by the VkPipelineViewportDepthClipControlCreateInfoEXT::`negativeOneToOne` value used to create the currently active pipeline.

### Valid Usage

- VUID-vkCmdSetDepthClipNegativeOneToOneEXT-None-09423
  At least one of the following **must** be true:
  - The `extendedDynamicState3DepthClipNegativeOneToOne` feature is enabled
  - The `shaderObject` feature is enabled
- VUID-vkCmdSetDepthClipNegativeOneToOneEXT-depthClipControl-07453
  The `depthClipControl` feature **must** be enabled

### Valid Usage (Implicit)

- VUID-vkCmdSetDepthClipNegativeOneToOneEXT-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdSetDepthClipNegativeOneToOneEXT-commandBuffer-recording

commandBuffer **must** be in the recording state

- VUID-vkCmdSetDepthClipNegativeOneToOneEXT-commandBuffer-cmdpool
  The VkCommandPool that commandBuffer was allocated from **must** support graphics operations
- VUID-vkCmdSetDepthClipNegativeOneToOneEXT-videocoding
  This command **must** only be called outside of a video coding scope

### Host Synchronization

- Host access to commandBuffer **must** be externally synchronized
- Host access to the VkCommandPool that commandBuffer was allocated from **must** be externally synchronized

### Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics | State |

## 27.5. Clipping Shader Outputs

Next, vertex output attributes are clipped. The output values associated with a vertex that lies within the clip volume are unaffected by clipping. If a primitive is clipped, however, the output values assigned to vertices produced by clipping are clipped.

Let the output values assigned to the two vertices $\mathbf{P}_1$ and $\mathbf{P}_2$ of an unclipped edge be $\mathbf{c}_1$ and $\mathbf{c}_2$. The value of t (see Primitive Clipping) for a clipped point $\mathbf{P}$ is used to obtain the output value associated with $\mathbf{P}$ as

$$\mathbf{c} = t\,\mathbf{c}_1 + (1-t)\,\mathbf{c}_2.$$

(Multiplying an output value by a scalar means multiplying each of *x*, *y*, *z*, and *w* by the scalar.)

Since this computation is performed in clip space before division by $w_c$, clipped output values are perspective-correct.

Polygon clipping creates a clipped vertex along an edge of the clip volume's boundary. This situation is handled by noting that polygon clipping proceeds by clipping against one half-space at a time. Output value clipping is done in the same way, so that clipped points always occur at the intersection of polygon edges (possibly already clipped) with the clip volume's boundary.

For vertex output attributes whose matching fragment input attributes are decorated with

NoPerspective, the value of t used to obtain the output value associated with **P** will be adjusted to produce results that vary linearly in framebuffer space.

Output attributes of integer or unsigned integer type **must** always be flat shaded. Flat shaded attributes are constant over the primitive being rasterized (see Basic Line Segment Rasterization and Basic Polygon Rasterization), and no interpolation is performed. The output value **c** is taken from either $\mathbf{c}_1$ or $\mathbf{c}_2$, since flat shading has already occurred and the two values are identical.

## 27.6. Controlling Viewport W Scaling

If viewport **W** scaling is enabled, the **W** component of the clip coordinate is modified by the provided coefficients from the corresponding viewport as follows.

$$w_c' = x_{coeff}\, x_c + y_{coeff}\, y_c + w_c$$

The VkPipelineViewportWScalingStateCreateInfoNV structure is defined as:

```
// Provided by VK_NV_clip_space_w_scaling
typedef struct VkPipelineViewportWScalingStateCreateInfoNV {
    VkStructureType                 sType;
    const void*                     pNext;
    VkBool32                        viewportWScalingEnable;
    uint32_t                        viewportCount;
    const VkViewportWScalingNV*     pViewportWScalings;
} VkPipelineViewportWScalingStateCreateInfoNV;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- viewportWScalingEnable controls whether viewport **W** scaling is enabled.
- viewportCount is the number of viewports used by **W** scaling, and **must** match the number of viewports in the pipeline if viewport **W** scaling is enabled.
- pViewportWScalings is a pointer to an array of VkViewportWScalingNV structures defining the **W** scaling parameters for the corresponding viewports. If the viewport **W** scaling state is dynamic, this member is ignored.

### Valid Usage (Implicit)

- VUID-VkPipelineViewportWScalingStateCreateInfoNV-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PIPELINE_VIEWPORT_W_SCALING_STATE_CREATE_INFO_NV
- VUID-VkPipelineViewportWScalingStateCreateInfoNV-viewportCount-arraylength
  viewportCount **must** be greater than 0

The VkPipelineViewportWScalingStateCreateInfoNV state is set by adding this structure to the pNext chain of a VkPipelineViewportStateCreateInfo structure and setting the graphics pipeline state with

vkCreateGraphicsPipelines.

To dynamically set the viewportWScalingEnable state, call:

```
// Provided by VK_EXT_extended_dynamic_state3 with VK_NV_clip_space_w_scaling,
VK_EXT_shader_object with VK_NV_clip_space_w_scaling
void vkCmdSetViewportWScalingEnableNV(
    VkCommandBuffer                             commandBuffer,
    VkBool32                                    viewportWScalingEnable);
```

- commandBuffer is the command buffer into which the command will be recorded.
- viewportWScalingEnable specifies the viewportWScalingEnable state.

This command sets the viewportWScalingEnable state for subsequent drawing commands when drawing using shader objects, or when the graphics pipeline is created with VK_DYNAMIC_STATE_VIEWPORT_W_SCALING_ENABLE_NV set in VkPipelineDynamicStateCreateInfo::pDynamicStates. Otherwise, this state is specified by the VkPipelineViewportWScalingStateCreateInfoNV::viewportWScalingEnable value used to create the currently active pipeline.

## Valid Usage

- VUID-vkCmdSetViewportWScalingEnableNV-None-09423
  At least one of the following **must** be true:
  - The extendedDynamicState3ViewportWScalingEnable feature is enabled
  - The shaderObject feature is enabled

## Valid Usage (Implicit)

- VUID-vkCmdSetViewportWScalingEnableNV-commandBuffer-parameter
  commandBuffer **must** be a valid VkCommandBuffer handle
- VUID-vkCmdSetViewportWScalingEnableNV-commandBuffer-recording
  commandBuffer **must** be in the recording state
- VUID-vkCmdSetViewportWScalingEnableNV-commandBuffer-cmdpool
  The VkCommandPool that commandBuffer was allocated from **must** support graphics operations
- VUID-vkCmdSetViewportWScalingEnableNV-videocoding
  This command **must** only be called outside of a video coding scope

## Host Synchronization

- Host access to commandBuffer **must** be externally synchronized

- Host access to the VkCommandPool that commandBuffer was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics | State |

To dynamically set the viewport **W** scaling parameters, call:

```
// Provided by VK_NV_clip_space_w_scaling
void vkCmdSetViewportWScalingNV(
    VkCommandBuffer                             commandBuffer,
    uint32_t                                    firstViewport,
    uint32_t                                    viewportCount,
    const VkViewportWScalingNV*                 pViewportWScalings);
```

- commandBuffer is the command buffer into which the command will be recorded.
- firstViewport is the index of the first viewport whose parameters are updated by the command.
- viewportCount is the number of viewports whose parameters are updated by the command.
- pViewportWScalings is a pointer to an array of VkViewportWScalingNV structures specifying viewport parameters.

The viewport parameters taken from element i of pViewportWScalings replace the current state for the viewport index firstViewport + i, for i in [0, viewportCount).

This command sets the viewport **W** scaling for subsequent drawing commands when drawing using shader objects, or when the graphics pipeline is created with VK_DYNAMIC_STATE_VIEWPORT_W_SCALING_NV set in VkPipelineDynamicStateCreateInfo::pDynamicStates. Otherwise, this state is specified by the VkPipelineViewportWScalingStateCreateInfoNV::pViewportWScalings values used to create the currently active pipeline.

## Valid Usage

- VUID-vkCmdSetViewportWScalingNV-firstViewport-01324
  The sum of firstViewport and viewportCount **must** be between 1 and VkPhysicalDeviceLimits::maxViewports, inclusive

## Valid Usage (Implicit)

- VUID-vkCmdSetViewportWScalingNV-commandBuffer-parameter

commandBuffer **must** be a valid VkCommandBuffer handle

- VUID-vkCmdSetViewportWScalingNV-pViewportWScalings-parameter
  pViewportWScalings **must** be a valid pointer to an array of viewportCount VkViewportWScalingNV structures
- VUID-vkCmdSetViewportWScalingNV-commandBuffer-recording
  commandBuffer **must** be in the recording state
- VUID-vkCmdSetViewportWScalingNV-commandBuffer-cmdpool
  The VkCommandPool that commandBuffer was allocated from **must** support graphics operations
- VUID-vkCmdSetViewportWScalingNV-videocoding
  This command **must** only be called outside of a video coding scope
- VUID-vkCmdSetViewportWScalingNV-viewportCount-arraylength
  viewportCount **must** be greater than 0

## Host Synchronization

- Host access to commandBuffer **must** be externally synchronized
- Host access to the VkCommandPool that commandBuffer was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics | State |

Both VkPipelineViewportWScalingStateCreateInfoNV and vkCmdSetViewportWScalingNV use VkViewportWScalingNV to set the viewport transformation parameters.

The VkViewportWScalingNV structure is defined as:

```
// Provided by VK_NV_clip_space_w_scaling
typedef struct VkViewportWScalingNV {
    float    xcoeff;
    float    ycoeff;
} VkViewportWScalingNV;
```

- xcoeff and ycoeff are the viewport's W scaling factor for x and y respectively.

## 27.7. Coordinate Transformations

*Clip coordinates* for a vertex result from shader execution, which yields a vertex coordinate `Position`.

Perspective division on clip coordinates yields *normalized device coordinates*, followed by a *viewport* transformation (see Controlling the Viewport) to convert these coordinates into *framebuffer coordinates*.

If a vertex in clip coordinates has a position given by

$$\begin{pmatrix} x_c \\ y_c \\ z_c \\ w_c \end{pmatrix}$$

then the vertex's normalized device coordinates are

$$\begin{pmatrix} x_d \\ y_d \\ z_d \end{pmatrix} = \begin{pmatrix} \frac{x_c}{w_c} \\ \frac{y_c}{w_c} \\ \frac{z_c}{w_c} \end{pmatrix}$$

## 27.8. Render Pass Transform

A *render pass transform* **can** be enabled for render pass instances. The clip coordinates ($x_c$, $y_c$) that result from vertex shader execution are transformed by a rotation of 0, 90, 180, or 270 degrees in the XY plane, centered at the origin.

When *Render pass transform* is enabled, the transform applies to all primitives for all subpasses of the render pass. The transformed vertex in clip coordinates has a position given by

$$\begin{pmatrix} x_{c_{trans}} \\ y_{c_{trans}} \\ z_{c_{trans}} \end{pmatrix} = \begin{pmatrix} x_c \cos\theta - y_c \sin\theta \\ x_c \sin\theta + y_c \cos\theta \\ z_c \end{pmatrix}$$

where

- $\theta$ is 0 degrees for `VK_SURFACE_TRANSFORM_IDENTITY_BIT_KHR`
- $\theta$ is 90 degrees for `VK_SURFACE_TRANSFORM_ROTATE_90_BIT_KHR`
- $\theta$ is 180 degrees for `VK_SURFACE_TRANSFORM_ROTATE_180_BIT_KHR`
- $\theta$ is 270 degrees for `VK_SURFACE_TRANSFORM_ROTATE_270_BIT_KHR`

The transformed vertex's normalized device coordinates are

$$\begin{pmatrix} x_d \\ y_d \\ z_d \end{pmatrix} = \begin{pmatrix} \frac{x_{c_{trans}}}{w_c} \\ \frac{y_{c_{trans}}}{w_c} \\ \frac{z_{c_{trans}}}{w_c} \end{pmatrix}$$

When render pass transform is enabled for a render pass instance, the following additional features are enabled:

- Each VkViewport specified by either VkPipelineViewportStateCreateInfo::`pViewports` or vkCmdSetViewport will have its width/height ($p_x$, $p_y$) and its center ($o_x$, $o_y$) similarly transformed by the implementation.
- Each scissor specified by VkPipelineViewportStateCreateInfo::`pScissors` or vkCmdSetScissor will have its ($offset_x$, $offset_y$) and ($extent_x$, $extent_y$) similarly transformed by the implementation.
- The `renderArea` specified in VkCommandBufferInheritanceRenderPassTransformInfoQCOM and VkRenderPassBeginInfo will be similarly transformed by the implementation.
- The (x, y) components of shader variables with built-in decorations `FragCoord`, `SamplePosition`, or `PointCoord` will be similarly transformed by the implementation.
- The (x,y) components of the `offset` operand of the `InterpolateAtOffset` extended instruction will be similarly transformed by the implementation.
- The values returned by SPIR-V derivative instructions `OpDPdx`, `OpDPdy`, `OpDPdxCourse`, `OpDPdyCourse`, `OpDPdxFine`, `OpDPdyFine` will be similarly transformed by the implementation.

The net result of the above, is that applications **can** act as if rendering to a framebuffer oriented with the VkSurfaceCapabilitiesKHR::`currentTransform`. In other words, applications **can** act as if the presentation engine will be performing the transformation of the swapchain image after rendering and prior to presentation to the user. In fact, the transformation of the various items cited above are being handled by the implementation as the rendering takes place.

## 27.9. Controlling the Viewport

The viewport transformation is determined by the selected viewport's width and height in pixels, $p_x$ and $p_y$, respectively, and its center ($o_x$, $o_y$) (also in pixels), as well as its depth range min and max determining a depth range scale value $p_z$ and a depth range bias value $o_z$ (defined below). The vertex's framebuffer coordinates ($x_f$, $y_f$, $z_f$) are given by

$$x_f = (p_x / 2)\, x_d + o_x$$

$$y_f = (p_y / 2)\, y_d + o_y$$

$$z_f = p_z \times z_d + o_z$$

Multiple viewports are available, numbered zero up to VkPhysicalDeviceLimits::maxViewports minus one. The number of viewports used by a pipeline is controlled by the viewportCount member of the VkPipelineViewportStateCreateInfo structure used in pipeline creation.

$x_f$ and $y_f$ have limited precision, where the number of fractional bits retained is specified by VkPhysicalDeviceLimits::subPixelPrecisionBits. When rasterizing line segments, the number of fractional bits is specified by VkPhysicalDeviceLineRasterizationPropertiesKHR::lineSubPixelPrecisionBits.

The VkPipelineViewportStateCreateInfo structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkPipelineViewportStateCreateInfo {
    VkStructureType                       sType;
    const void*                           pNext;
    VkPipelineViewportStateCreateFlags    flags;
    uint32_t                              viewportCount;
    const VkViewport*                     pViewports;
    uint32_t                              scissorCount;
    const VkRect2D*                       pScissors;
} VkPipelineViewportStateCreateInfo;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- flags is reserved for future use.
- viewportCount is the number of viewports used by the pipeline.
- pViewports is a pointer to an array of VkViewport structures, defining the viewport transforms. If the viewport state is dynamic, this member is ignored.
- scissorCount is the number of scissors and **must** match the number of viewports.
- pScissors is a pointer to an array of VkRect2D structures defining the rectangular bounds of the scissor for the corresponding viewport. If the scissor state is dynamic, this member is ignored.

## Valid Usage

- VUID-VkPipelineViewportStateCreateInfo-viewportCount-01216
  If the multiViewport feature is not enabled, viewportCount **must** not be greater than 1
- VUID-VkPipelineViewportStateCreateInfo-scissorCount-01217
  If the multiViewport feature is not enabled, scissorCount **must** not be greater than 1
- VUID-VkPipelineViewportStateCreateInfo-viewportCount-01218
  viewportCount **must** be less than or equal to VkPhysicalDeviceLimits::maxViewports
- VUID-VkPipelineViewportStateCreateInfo-scissorCount-01219
  scissorCount **must** be less than or equal to VkPhysicalDeviceLimits::maxViewports
- VUID-VkPipelineViewportStateCreateInfo-x-02821
  The x and y members of offset member of any element of pScissors **must** be greater than

or equal to `0`

- VUID-VkPipelineViewportStateCreateInfo-offset-02822
  Evaluation of (`offset.x` + `extent.width`) **must** not cause a signed integer addition overflow for any element of `pScissors`
- VUID-VkPipelineViewportStateCreateInfo-offset-02823
  Evaluation of (`offset.y` + `extent.height`) **must** not cause a signed integer addition overflow for any element of `pScissors`
- VUID-VkPipelineViewportStateCreateInfo-scissorCount-04134
  If `scissorCount` and `viewportCount` are both not dynamic, then `scissorCount` and `viewportCount` **must** be identical
- VUID-VkPipelineViewportStateCreateInfo-viewportCount-04135
  If the graphics pipeline is being created with `VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT` set then `viewportCount` **must** be `0`, otherwise `viewportCount` **must** be greater than `0`
- VUID-VkPipelineViewportStateCreateInfo-scissorCount-04136
  If the graphics pipeline is being created with `VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT` set then `scissorCount` **must** be `0`, otherwise `scissorCount` **must** be greater than `0`
- VUID-VkPipelineViewportStateCreateInfo-viewportWScalingEnable-01726
  If the `viewportWScalingEnable` member of a VkPipelineViewportWScalingStateCreateInfoNV structure included in the `pNext` chain is `VK_TRUE`, the `viewportCount` member of the VkPipelineViewportWScalingStateCreateInfoNV structure **must** be greater than or equal to VkPipelineViewportStateCreateInfo::`viewportCount`

## Valid Usage (Implicit)

- VUID-VkPipelineViewportStateCreateInfo-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PIPELINE_VIEWPORT_STATE_CREATE_INFO`
- VUID-VkPipelineViewportStateCreateInfo-pNext-pNext
  Each `pNext` member of any structure (including this one) in the `pNext` chain **must** be either `NULL` or a pointer to a valid instance of VkPipelineViewportCoarseSampleOrderStateCreateInfoNV, VkPipelineViewportDepthClipControlCreateInfoEXT, VkPipelineViewportExclusiveScissorStateCreateInfoNV, VkPipelineViewportShadingRateImageStateCreateInfoNV, VkPipelineViewportSwizzleStateCreateInfoNV, or VkPipelineViewportWScalingStateCreateInfoNV
- VUID-VkPipelineViewportStateCreateInfo-sType-unique
  The `sType` value of each struct in the `pNext` chain **must** be unique
- VUID-VkPipelineViewportStateCreateInfo-flags-zerobitmask
  `flags` **must** be `0`

To dynamically set the viewport count and viewports, call:

```
// Provided by VK_EXT_extended_dynamic_state, VK_EXT_shader_object
void vkCmdSetViewportWithCountEXT(
    VkCommandBuffer                             commandBuffer,
    uint32_t                                    viewportCount,
    const VkViewport*                           pViewports);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `viewportCount` specifies the viewport count.
- `pViewports` specifies the viewports to use for drawing.

This command sets the viewport count and viewports state for subsequent drawing commands when drawing using shader objects, or when the graphics pipeline is created with `VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT` set in VkPipelineDynamicStateCreateInfo::`pDynamicStates`. Otherwise, this state is specified by the corresponding VkPipelineViewportStateCreateInfo::`viewportCount` and `pViewports` values used to create the currently active pipeline.

## Valid Usage

- VUID-vkCmdSetViewportWithCount-None-08971
  At least one of the following **must** be true:
  - the `extendedDynamicState` feature is enabled
  - the `shaderObject` feature is enabled
- VUID-vkCmdSetViewportWithCount-viewportCount-03394
  `viewportCount` **must** be between `1` and `VkPhysicalDeviceLimits`::`maxViewports`, inclusive
- VUID-vkCmdSetViewportWithCount-viewportCount-03395
  If the `multiViewport` feature is not enabled, `viewportCount` **must** be `1`
- VUID-vkCmdSetViewportWithCount-commandBuffer-04819
  `commandBuffer` **must** not have VkCommandBufferInheritanceViewportScissorInfoNV::`viewportScissor2D` enabled

## Valid Usage (Implicit)

- VUID-vkCmdSetViewportWithCount-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdSetViewportWithCount-pViewports-parameter
  `pViewports` **must** be a valid pointer to an array of `viewportCount` valid VkViewport structures
- VUID-vkCmdSetViewportWithCount-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdSetViewportWithCount-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations

- VUID-vkCmdSetViewportWithCount-videocoding
  This command **must** only be called outside of a video coding scope
- VUID-vkCmdSetViewportWithCount-viewportCount-arraylength
  `viewportCount` **must** be greater than `0`

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics | State |

To dynamically set the scissor count and scissor rectangular bounds, call:

```
// Provided by VK_EXT_extended_dynamic_state, VK_EXT_shader_object
void vkCmdSetScissorWithCountEXT(
    VkCommandBuffer                             commandBuffer,
    uint32_t                                    scissorCount,
    const VkRect2D*                             pScissors);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `scissorCount` specifies the scissor count.
- `pScissors` specifies the scissors to use for drawing.

This command sets the scissor count and scissor rectangular bounds state for subsequent drawing commands when drawing using shader objects, or when the graphics pipeline is created with `VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT` set in VkPipelineDynamicStateCreateInfo::`pDynamicStates`. Otherwise, this state is specified by the corresponding VkPipelineViewportStateCreateInfo::`scissorCount` and `pScissors` values used to create the currently active pipeline.

## Valid Usage

- VUID-vkCmdSetScissorWithCount-None-08971
  At least one of the following **must** be true:
  - the `extendedDynamicState` feature is enabled

- the `shaderObject` feature is enabled

- VUID-vkCmdSetScissorWithCount-scissorCount-03397
  `scissorCount` **must** be between `1` and `VkPhysicalDeviceLimits`::`maxViewports`, inclusive
- VUID-vkCmdSetScissorWithCount-scissorCount-03398
  If the `multiViewport` feature is not enabled, `scissorCount` **must** be `1`
- VUID-vkCmdSetScissorWithCount-x-03399
  The `x` and `y` members of `offset` member of any element of `pScissors` **must** be greater than or equal to `0`
- VUID-vkCmdSetScissorWithCount-offset-03400
  Evaluation of (`offset.x` + `extent.width`) **must** not cause a signed integer addition overflow for any element of `pScissors`
- VUID-vkCmdSetScissorWithCount-offset-03401
  Evaluation of (`offset.y` + `extent.height`) **must** not cause a signed integer addition overflow for any element of `pScissors`
- VUID-vkCmdSetScissorWithCount-commandBuffer-04820
  `commandBuffer` **must** not have VkCommandBufferInheritanceViewportScissorInfoNV::`viewportScissor2D` enabled

## Valid Usage (Implicit)

- VUID-vkCmdSetScissorWithCount-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdSetScissorWithCount-pScissors-parameter
  `pScissors` **must** be a valid pointer to an array of `scissorCount` VkRect2D structures
- VUID-vkCmdSetScissorWithCount-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdSetScissorWithCount-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- VUID-vkCmdSetScissorWithCount-videocoding
  This command **must** only be called outside of a video coding scope
- VUID-vkCmdSetScissorWithCount-scissorCount-arraylength
  `scissorCount` **must** be greater than `0`

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

### Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics | State |

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkPipelineViewportStateCreateFlags;
```

VkPipelineViewportStateCreateFlags is a bitmask type for setting a mask, but is currently reserved for future use.

A *pre-rasterization shader stage* **can** direct each primitive to zero or more viewports. The destination viewports for a primitive are selected by the last active pre-rasterization shader stage that has an output variable decorated with ViewportIndex (selecting a single viewport) or ViewportMaskNV (selecting multiple viewports). The viewport transform uses the viewport corresponding to either the value assigned to ViewportIndex or one of the bits set in ViewportMaskNV, and taken from an implementation-dependent vertex of each primitive. If ViewportIndex or any of the bits in ViewportMaskNV are outside the range zero to viewportCount minus one for a primitive, or if the last active pre-rasterization shader stage did not assign a value to either ViewportIndex or ViewportMaskNV for all vertices of a primitive due to flow control, the values resulting from the viewport transformation of the vertices of such primitives are undefined. If the last pre-rasterization shader stage does not have an output decorated with ViewportIndex or ViewportMaskNV, the viewport numbered zero is used by the viewport transformation.

A single vertex **can** be used in more than one individual primitive, in primitives such as VK_PRIMITIVE_TOPOLOGY_TRIANGLE_STRIP. In this case, the viewport transformation is applied separately for each primitive.

To dynamically set the viewport transformation parameters, call:

```
// Provided by VK_VERSION_1_0
void vkCmdSetViewport(
    VkCommandBuffer                             commandBuffer,
    uint32_t                                    firstViewport,
    uint32_t                                    viewportCount,
    const VkViewport*                           pViewports);
```

- commandBuffer is the command buffer into which the command will be recorded.
- firstViewport is the index of the first viewport whose parameters are updated by the command.
- viewportCount is the number of viewports whose parameters are updated by the command.
- pViewports is a pointer to an array of VkViewport structures specifying viewport parameters.

This command sets the viewport transformation parameters state for subsequent drawing commands when drawing using shader objects, or when the graphics pipeline is created with VK_DYNAMIC_STATE_VIEWPORT set in VkPipelineDynamicStateCreateInfo::pDynamicStates. Otherwise, this state is specified by the VkPipelineViewportStateCreateInfo::pViewports values used to create the currently active pipeline.

The viewport parameters taken from element i of pViewports replace the current state for the viewport index firstViewport + i, for i in [0, viewportCount).

## Valid Usage

- VUID-vkCmdSetViewport-firstViewport-01223
  The sum of firstViewport and viewportCount **must** be between 1 and VkPhysicalDeviceLimits::maxViewports, inclusive
- VUID-vkCmdSetViewport-firstViewport-01224
  If the multiViewport feature is not enabled, firstViewport **must** be 0
- VUID-vkCmdSetViewport-viewportCount-01225
  If the multiViewport feature is not enabled, viewportCount **must** be 1
- VUID-vkCmdSetViewport-commandBuffer-04821
  commandBuffer **must** not have VkCommandBufferInheritanceViewportScissorInfoNV::viewportScissor2D enabled

## Valid Usage (Implicit)

- VUID-vkCmdSetViewport-commandBuffer-parameter
  commandBuffer **must** be a valid VkCommandBuffer handle
- VUID-vkCmdSetViewport-pViewports-parameter
  pViewports **must** be a valid pointer to an array of viewportCount valid VkViewport structures
- VUID-vkCmdSetViewport-commandBuffer-recording
  commandBuffer **must** be in the recording state
- VUID-vkCmdSetViewport-commandBuffer-cmdpool
  The VkCommandPool that commandBuffer was allocated from **must** support graphics operations
- VUID-vkCmdSetViewport-videocoding
  This command **must** only be called outside of a video coding scope
- VUID-vkCmdSetViewport-viewportCount-arraylength
  viewportCount **must** be greater than 0

## Host Synchronization

- Host access to commandBuffer **must** be externally synchronized

- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

### Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics | State |

Both VkPipelineViewportStateCreateInfo and vkCmdSetViewport use `VkViewport` to set the viewport transformation parameters.

The `VkViewport` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkViewport {
    float    x;
    float    y;
    float    width;
    float    height;
    float    minDepth;
    float    maxDepth;
} VkViewport;
```

- `x` and `y` are the viewport's upper left corner (x,y).
- `width` and `height` are the viewport's width and height, respectively.
- `minDepth` and `maxDepth` are the depth range for the viewport.

**NOTE** Despite their names, `minDepth` **can** be less than, equal to, or greater than `maxDepth`.

The framebuffer depth coordinate $z_f$ **may** be represented using either a fixed-point or floating-point representation. However, a floating-point representation **must** be used if the depth/stencil attachment has a floating-point depth component. If an m-bit fixed-point representation is used, we assume that it represents each value $\frac{k}{2^m - 1}$, where k □ { 0, 1, ..., $2^m$-1 }, as k (e.g. 1.0 is represented in binary as a string of all ones).

The viewport parameters shown in the above equations are found from these values as

$o_x$ = `x` + `width` / 2

$o_y$ = `y` + `height` / 2

$o_z$ = `minDepth` (or (`maxDepth` + `minDepth`) / 2 if VkPipelineViewportDepthClipControlCreateInfoEXT::`negativeOneToOne` is `VK_TRUE`)

$p_x$ = `width`

$p_y$ = `height`

$p_z$ = `maxDepth` - `minDepth` (or (`maxDepth` - `minDepth`) / 2 if VkPipelineViewportDepthClipControlCreateInfoEXT::`negativeOneToOne` is `VK_TRUE`)

If a render pass transform is enabled, the values ($p_x$,$p_y$) and ($o_x$, $o_y$) defining the viewport are transformed as described in render pass transform before participating in the viewport transform.

The application **can** specify a negative term for `height`, which has the effect of negating the y coordinate in clip space before performing the transform. When using a negative `height`, the application **should** also adjust the `y` value to point to the lower left corner of the viewport instead of the upper left corner. Using the negative `height` allows the application to avoid having to negate the y component of the `Position` output from the last pre-rasterization shader stage.

The width and height of the implementation-dependent maximum viewport dimensions **must** be greater than or equal to the width and height of the largest image which **can** be created and attached to a framebuffer.

The floating-point viewport bounds are represented with an implementation-dependent precision.

## Valid Usage

- VUID-VkViewport-width-01770
  `width` **must** be greater than `0.0`
- VUID-VkViewport-width-01771
  `width` **must** be less than or equal to `VkPhysicalDeviceLimits`::`maxViewportDimensions`[0]
- VUID-VkViewport-apiVersion-07917
  If the VK_KHR_maintenance1 extension is not enabled, the VK_AMD_negative_viewport_height extension is not enabled, and VkPhysicalDeviceProperties::`apiVersion` is less than Vulkan 1.1, `height` **must** be greater than `0.0`
- VUID-VkViewport-height-01773
  The absolute value of `height` **must** be less than or equal to `VkPhysicalDeviceLimits`::`maxViewportDimensions`[1]
- VUID-VkViewport-x-01774
  `x` **must** be greater than or equal to `viewportBoundsRange`[0]
- VUID-VkViewport-x-01232

(x + width) **must** be less than or equal to viewportBoundsRange[1]

- VUID-VkViewport-y-01775
  y **must** be greater than or equal to viewportBoundsRange[0]
- VUID-VkViewport-y-01776
  y **must** be less than or equal to viewportBoundsRange[1]
- VUID-VkViewport-y-01777
  (y + height) **must** be greater than or equal to viewportBoundsRange[0]
- VUID-VkViewport-y-01233
  (y + height) **must** be less than or equal to viewportBoundsRange[1]
- VUID-VkViewport-minDepth-01234
  If the VK_EXT_depth_range_unrestricted extension is not enabled, minDepth **must** be between 0.0 and 1.0, inclusive
- VUID-VkViewport-maxDepth-01235
  If the VK_EXT_depth_range_unrestricted extension is not enabled, maxDepth **must** be between 0.0 and 1.0, inclusive

# Chapter 28. Rasterization

Rasterization is the process by which a primitive is converted to a two-dimensional image. Each discrete location of this image contains associated data such as depth, color, or other attributes.

Rasterizing a primitive begins by determining which squares of an integer grid in framebuffer coordinates are occupied by the primitive, and assigning one or more depth values to each such square. This process is described below for points, lines, and polygons.

A grid square, including its (x,y) framebuffer coordinates, z (depth), and associated data added by fragment shaders, is called a fragment. A fragment is located by its upper left corner, which lies on integer grid coordinates.

Rasterization operations also refer to a fragment's sample locations, which are offset by fractional values from its upper left corner. The rasterization rules for points, lines, and triangles involve testing whether each sample location is inside the primitive. Fragments need not actually be square, and rasterization rules are not affected by the aspect ratio of fragments. Display of non-square grids, however, will cause rasterized points and line segments to appear fatter in one direction than the other.

We assume that fragments are square, since it simplifies antialiasing and texturing. After rasterization, fragments are processed by fragment operations.

Several factors affect rasterization, including the members of VkPipelineRasterizationStateCreateInfo and VkPipelineMultisampleStateCreateInfo.

The `VkPipelineRasterizationStateCreateInfo` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkPipelineRasterizationStateCreateInfo {
    VkStructureType                             sType;
    const void*                                 pNext;
    VkPipelineRasterizationStateCreateFlags     flags;
    VkBool32                                    depthClampEnable;
    VkBool32                                    rasterizerDiscardEnable;
    VkPolygonMode                               polygonMode;
    VkCullModeFlags                             cullMode;
    VkFrontFace                                 frontFace;
    VkBool32                                    depthBiasEnable;
    float                                       depthBiasConstantFactor;
    float                                       depthBiasClamp;
    float                                       depthBiasSlopeFactor;
    float                                       lineWidth;
} VkPipelineRasterizationStateCreateInfo;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is reserved for future use.

- `depthClampEnable` controls whether to clamp the fragment's depth values as described in Depth Test. If the pipeline is not created with VkPipelineRasterizationDepthClipStateCreateInfoEXT present then enabling depth clamp will also disable clipping primitives to the z planes of the frustum as described in Primitive Clipping. Otherwise depth clipping is controlled by the state set in VkPipelineRasterizationDepthClipStateCreateInfoEXT.
- `rasterizerDiscardEnable` controls whether primitives are discarded immediately before the rasterization stage.
- `polygonMode` is the triangle rendering mode. See VkPolygonMode.
- `cullMode` is the triangle facing direction used for primitive culling. See VkCullModeFlagBits.
- `frontFace` is a VkFrontFace value specifying the front-facing triangle orientation to be used for culling.
- `depthBiasEnable` controls whether to bias fragment depth values.
- `depthBiasConstantFactor` is a scalar factor controlling the constant depth value added to each fragment.
- `depthBiasClamp` is the maximum (or minimum) depth bias of a fragment.
- `depthBiasSlopeFactor` is a scalar factor applied to a fragment's slope in depth bias calculations.
- `lineWidth` is the width of rasterized line segments.

The application **can** also add a `VkPipelineRasterizationStateRasterizationOrderAMD` structure to the `pNext` chain of a VkPipelineRasterizationStateCreateInfo structure. This structure enables selecting the rasterization order to use when rendering with the corresponding graphics pipeline as described in Rasterization Order.

## Valid Usage

- VUID-VkPipelineRasterizationStateCreateInfo-depthClampEnable-00782
  If the `depthClamp` feature is not enabled, `depthClampEnable` **must** be `VK_FALSE`
- VUID-VkPipelineRasterizationStateCreateInfo-polygonMode-01507
  If the `fillModeNonSolid` feature is not enabled, `polygonMode` **must** be `VK_POLYGON_MODE_FILL` or `VK_POLYGON_MODE_FILL_RECTANGLE_NV`
- VUID-VkPipelineRasterizationStateCreateInfo-polygonMode-01414
  If the `VK_NV_fill_rectangle` extension is not enabled, `polygonMode` **must** not be `VK_POLYGON_MODE_FILL_RECTANGLE_NV`
- VUID-VkPipelineRasterizationStateCreateInfo-pointPolygons-04458
  If the `VK_KHR_portability_subset` extension is enabled, and VkPhysicalDevicePortabilitySubsetFeaturesKHR::`pointPolygons` is `VK_FALSE`, and `rasterizerDiscardEnable` is `VK_FALSE`, `polygonMode` **must** not be `VK_POLYGON_MODE_POINT`

## Valid Usage (Implicit)

- VUID-VkPipelineRasterizationStateCreateInfo-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PIPELINE_RASTERIZATION_STATE_CREATE_INFO`

- VUID-VkPipelineRasterizationStateCreateInfo-pNext-pNext
  Each pNext member of any structure (including this one) in the pNext chain **must** be either NULL or a pointer to a valid instance of VkDepthBiasRepresentationInfoEXT, VkPipelineRasterizationConservativeStateCreateInfoEXT, VkPipelineRasterizationDepthClipStateCreateInfoEXT, VkPipelineRasterizationLineStateCreateInfoKHR, VkPipelineRasterizationProvokingVertexStateCreateInfoEXT, VkPipelineRasterizationStateRasterizationOrderAMD, or VkPipelineRasterizationStateStreamCreateInfoEXT
- VUID-VkPipelineRasterizationStateCreateInfo-sType-unique
  The sType value of each struct in the pNext chain **must** be unique
- VUID-VkPipelineRasterizationStateCreateInfo-flags-zerobitmask
  flags **must** be 0
- VUID-VkPipelineRasterizationStateCreateInfo-polygonMode-parameter
  polygonMode **must** be a valid VkPolygonMode value
- VUID-VkPipelineRasterizationStateCreateInfo-cullMode-parameter
  cullMode **must** be a valid combination of VkCullModeFlagBits values
- VUID-VkPipelineRasterizationStateCreateInfo-frontFace-parameter
  frontFace **must** be a valid VkFrontFace value

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkPipelineRasterizationStateCreateFlags;
```

VkPipelineRasterizationStateCreateFlags is a bitmask type for setting a mask, but is currently reserved for future use.

If the pNext chain of VkPipelineRasterizationStateCreateInfo includes a VkPipelineRasterizationDepthClipStateCreateInfoEXT structure, then that structure controls whether depth clipping is enabled or disabled.

The VkPipelineRasterizationDepthClipStateCreateInfoEXT structure is defined as:

```
// Provided by VK_EXT_depth_clip_enable
typedef struct VkPipelineRasterizationDepthClipStateCreateInfoEXT {
    VkStructureType                                         sType;
    const void*                                             pNext;
    VkPipelineRasterizationDepthClipStateCreateFlagsEXT     flags;
    VkBool32                                                depthClipEnable;
} VkPipelineRasterizationDepthClipStateCreateInfoEXT;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- flags is reserved for future use.

- depthClipEnable controls whether depth clipping is enabled as described in Primitive Clipping.

### Valid Usage (Implicit)

- VUID-VkPipelineRasterizationDepthClipStateCreateInfoEXT-sType-sType sType **must** be VK_STRUCTURE_TYPE_PIPELINE_RASTERIZATION_DEPTH_CLIP_STATE_CREATE_INFO_EXT
- VUID-VkPipelineRasterizationDepthClipStateCreateInfoEXT-flags-zerobitmask flags **must** be 0

```
// Provided by VK_EXT_depth_clip_enable
typedef VkFlags VkPipelineRasterizationDepthClipStateCreateFlagsEXT;
```

VkPipelineRasterizationDepthClipStateCreateFlagsEXT is a bitmask type for setting a mask, but is currently reserved for future use.

The VkPipelineMultisampleStateCreateInfo structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkPipelineMultisampleStateCreateInfo {
    VkStructureType                         sType;
    const void*                             pNext;
    VkPipelineMultisampleStateCreateFlags   flags;
    VkSampleCountFlagBits                   rasterizationSamples;
    VkBool32                                sampleShadingEnable;
    float                                   minSampleShading;
    const VkSampleMask*                     pSampleMask;
    VkBool32                                alphaToCoverageEnable;
    VkBool32                                alphaToOneEnable;
} VkPipelineMultisampleStateCreateInfo;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- flags is reserved for future use.
- rasterizationSamples is a VkSampleCountFlagBits value specifying the number of samples used in rasterization. This value is ignored for the purposes of setting the number of samples used in rasterization if the pipeline is created with the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT dynamic state set, but if VK_DYNAMIC_STATE_SAMPLE_MASK_EXT dynamic state is not set, it is still used to define the size of the pSampleMask array as described below.
- sampleShadingEnable **can** be used to enable Sample Shading.
- minSampleShading specifies a minimum fraction of sample shading if sampleShadingEnable is set to VK_TRUE.
- pSampleMask is a pointer to an array of VkSampleMask values used in the sample mask test.

- alphaToCoverageEnable controls whether a temporary coverage value is generated based on the alpha component of the fragment's first color output as specified in the Multisample Coverage section.
- alphaToOneEnable controls whether the alpha component of the fragment's first color output is replaced with one as described in Multisample Coverage.

Each bit in the sample mask is associated with a unique sample index as defined for the coverage mask. Each bit b for mask word w in the sample mask corresponds to sample index i, where i = 32 × w + b. pSampleMask has a length equal to ⌈ rasterizationSamples / 32 ⌉ words.

If pSampleMask is NULL, it is treated as if the mask has all bits set to 1.

## Valid Usage

- VUID-VkPipelineMultisampleStateCreateInfo-sampleShadingEnable-00784
  If the sampleRateShading feature is not enabled, sampleShadingEnable **must** be VK_FALSE
- VUID-VkPipelineMultisampleStateCreateInfo-alphaToOneEnable-00785
  If the alphaToOne feature is not enabled, alphaToOneEnable **must** be VK_FALSE
- VUID-VkPipelineMultisampleStateCreateInfo-minSampleShading-00786
  minSampleShading **must** be in the range [0,1]
- VUID-VkPipelineMultisampleStateCreateInfo-rasterizationSamples-01415
  If the VK_NV_framebuffer_mixed_samples extension is enabled, and if the subpass has any color attachments and rasterizationSamples is greater than the number of color samples, then sampleShadingEnable **must** be VK_FALSE

## Valid Usage (Implicit)

- VUID-VkPipelineMultisampleStateCreateInfo-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PIPELINE_MULTISAMPLE_STATE_CREATE_INFO
- VUID-VkPipelineMultisampleStateCreateInfo-pNext-pNext
  Each pNext member of any structure (including this one) in the pNext chain **must** be either NULL or a pointer to a valid instance of VkPipelineCoverageModulationStateCreateInfoNV, VkPipelineCoverageReductionStateCreateInfoNV, VkPipelineCoverageToColorStateCreateInfoNV, or VkPipelineSampleLocationsStateCreateInfoEXT
- VUID-VkPipelineMultisampleStateCreateInfo-sType-unique
  The sType value of each struct in the pNext chain **must** be unique
- VUID-VkPipelineMultisampleStateCreateInfo-flags-zerobitmask
  flags **must** be 0
- VUID-VkPipelineMultisampleStateCreateInfo-rasterizationSamples-parameter
  rasterizationSamples **must** be a valid VkSampleCountFlagBits value
- VUID-VkPipelineMultisampleStateCreateInfo-pSampleMask-parameter
  If pSampleMask is not NULL, pSampleMask **must** be a valid pointer to an array of

$\lceil \frac{rasterizationSamples}{32} \rceil$ VkSampleMask values

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkPipelineMultisampleStateCreateFlags;
```

VkPipelineMultisampleStateCreateFlags is a bitmask type for setting a mask, but is currently reserved for future use.

The elements of the sample mask array are of type VkSampleMask, each representing 32 bits of coverage information:

```
// Provided by VK_VERSION_1_0
typedef uint32_t VkSampleMask;
```

Rasterization only generates fragments which cover one or more pixels inside the framebuffer. Pixels outside the framebuffer are never considered covered in the fragment. Fragments which would be produced by application of any of the primitive rasterization rules described below but which lie outside the framebuffer are not produced, nor are they processed by any later stage of the pipeline, including any of the fragment operations.

Surviving fragments are processed by fragment shaders. Fragment shaders determine associated data for fragments, and **can** also modify or replace their assigned depth values.

## 28.1. Discarding Primitives Before Rasterization

Primitives are discarded before rasterization if the rasterizerDiscardEnable member of VkPipelineRasterizationStateCreateInfo is enabled. When enabled, primitives are discarded after they are processed by the last active shader stage in the pipeline before rasterization.

To dynamically enable whether primitives are discarded before the rasterization stage, call:

```
// Provided by VK_EXT_extended_dynamic_state2, VK_EXT_shader_object
void vkCmdSetRasterizerDiscardEnableEXT(
    VkCommandBuffer                             commandBuffer,
    VkBool32                                    rasterizerDiscardEnable);
```

- commandBuffer is the command buffer into which the command will be recorded.
- rasterizerDiscardEnable controls whether primitives are discarded immediately before the rasterization stage.

This command sets the discard enable for subsequent drawing commands when drawing using shader objects, or when the graphics pipeline is created with VK_DYNAMIC_STATE_RASTERIZER_DISCARD_ENABLE set in VkPipelineDynamicStateCreateInfo::pDynamicStates. Otherwise, this state is specified by the VkPipelineRasterizationStateCreateInfo::rasterizerDiscardEnable value used to create the currently active pipeline.

### Valid Usage

- VUID-vkCmdSetRasterizerDiscardEnable-None-08970
  At least one of the following **must** be true:
  - the `extendedDynamicState2` feature is enabled
  - the `shaderObject` feature is enabled

### Valid Usage (Implicit)

- VUID-vkCmdSetRasterizerDiscardEnable-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdSetRasterizerDiscardEnable-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdSetRasterizerDiscardEnable-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- VUID-vkCmdSetRasterizerDiscardEnable-videocoding
  This command **must** only be called outside of a video coding scope

### Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

### Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics | State |

## 28.2. Controlling the Vertex Stream Used for Rasterization

By default vertex data output from the last pre-rasterization shader stage are directed to vertex stream zero. Geometry shaders **can** emit primitives to multiple independent vertex streams. Each vertex emitted by the geometry shader is directed at one of the vertex streams. As vertices are received on each vertex stream, they are arranged into primitives of the type specified by the

geometry shader output primitive type. The shading language instructions OpEndPrimitive and OpEndStreamPrimitive **can** be used to end the primitive being assembled on a given vertex stream and start a new empty primitive of the same type. An implementation supports up to VkPhysicalDeviceTransformFeedbackPropertiesEXT::maxTransformFeedbackStreams streams, which is at least 1. The individual streams are numbered 0 through maxTransformFeedbackStreams minus 1. There is no requirement on the order of the streams to which vertices are emitted, and the number of vertices emitted to each vertex stream **can** be completely independent, subject only to the VkPhysicalDeviceTransformFeedbackPropertiesEXT::maxTransformFeedbackStreamDataSize and VkPhysicalDeviceTransformFeedbackPropertiesEXT::maxTransformFeedbackBufferDataSize limits. The primitives output from all vertex streams are passed to the transform feedback stage to be captured to transform feedback buffers in the manner specified by the last pre-rasterization shader stage shader's XfbBuffer, XfbStride, and Offsets decorations on the output interface variables in the graphics pipeline. To use a vertex stream other than zero, or to use multiple streams, the GeometryStreams capability **must** be specified.

By default, the primitives output from vertex stream zero are rasterized. If the implementation supports the VkPhysicalDeviceTransformFeedbackPropertiesEXT::transformFeedbackRasterizationStreamSelect property it is possible to rasterize a vertex stream other than zero.

By default, geometry shaders that emit vertices to multiple vertex streams are limited to using only the OutputPoints output primitive type. If the implementation supports the VkPhysicalDeviceTransformFeedbackPropertiesEXT::transformFeedbackStreamsLinesTriangles property it is possible to emit OutputLineStrip or OutputTriangleStrip in addition to OutputPoints.

The vertex stream used for rasterization is specified by adding a VkPipelineRasterizationStateStreamCreateInfoEXT structure to the pNext chain of a VkPipelineRasterizationStateCreateInfo structure.

The VkPipelineRasterizationStateStreamCreateInfoEXT structure is defined as:

```
// Provided by VK_EXT_transform_feedback
typedef struct VkPipelineRasterizationStateStreamCreateInfoEXT {
    VkStructureType                                     sType;
    const void*                                         pNext;
    VkPipelineRasterizationStateStreamCreateFlagsEXT    flags;
    uint32_t                                            rasterizationStream;
} VkPipelineRasterizationStateStreamCreateInfoEXT;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- flags is reserved for future use.
- rasterizationStream is the vertex stream selected for rasterization.

If this structure is not present, rasterizationStream is assumed to be zero.

### Valid Usage

- VUID-VkPipelineRasterizationStateStreamCreateInfoEXT-geometryStreams-02324 `VkPhysicalDeviceTransformFeedbackFeaturesEXT::geometryStreams` **must** be enabled
- VUID-VkPipelineRasterizationStateStreamCreateInfoEXT-rasterizationStream-02325 `rasterizationStream` **must** be less than VkPhysicalDeviceTransformFeedbackPropertiesEXT::`maxTransformFeedbackStreams`
- VUID-VkPipelineRasterizationStateStreamCreateInfoEXT-rasterizationStream-02326 `rasterizationStream` **must** be zero if `VkPhysicalDeviceTransformFeedbackPropertiesEXT::transformFeedbackRasterizationStreamSelect` is `VK_FALSE`

### Valid Usage (Implicit)

- VUID-VkPipelineRasterizationStateStreamCreateInfoEXT-sType-sType `sType` **must** be `VK_STRUCTURE_TYPE_PIPELINE_RASTERIZATION_STATE_STREAM_CREATE_INFO_EXT`
- VUID-VkPipelineRasterizationStateStreamCreateInfoEXT-flags-zerobitmask `flags` **must** be `0`

```
// Provided by VK_EXT_transform_feedback
typedef VkFlags VkPipelineRasterizationStateStreamCreateFlagsEXT;
```

`VkPipelineRasterizationStateStreamCreateFlagsEXT` is a bitmask type for setting a mask, but is currently reserved for future use.

To dynamically set the `rasterizationStream` state, call:

```
// Provided by VK_EXT_extended_dynamic_state3 with VK_EXT_transform_feedback,
VK_EXT_shader_object with VK_EXT_transform_feedback
void vkCmdSetRasterizationStreamEXT(
    VkCommandBuffer                             commandBuffer,
    uint32_t                                    rasterizationStream);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `rasterizationStream` specifies the `rasterizationStream` state.

This command sets the `rasterizationStream` state for subsequent drawing commands when drawing using shader objects, or when the graphics pipeline is created with `VK_DYNAMIC_STATE_RASTERIZATION_STREAM_EXT` set in VkPipelineDynamicStateCreateInfo::`pDynamicStates`. Otherwise, this state is specified by the VkPipelineRasterizationStateStreamCreateInfoEXT::`rasterizationStream` value used to create the currently active pipeline.

## Valid Usage

- VUID-vkCmdSetRasterizationStreamEXT-None-09423
  At least one of the following **must** be true:
  - The `extendedDynamicState3RasterizationStream` feature is enabled
  - The `shaderObject` feature is enabled
- VUID-vkCmdSetRasterizationStreamEXT-transformFeedback-07411
  The `transformFeedback` feature **must** be enabled
- VUID-vkCmdSetRasterizationStreamEXT-rasterizationStream-07412
  `rasterizationStream` **must** be less than VkPhysicalDeviceTransformFeedbackPropertiesEXT::`maxTransformFeedbackStreams`
- VUID-vkCmdSetRasterizationStreamEXT-rasterizationStream-07413
  `rasterizationStream` **must** be zero if `VkPhysicalDeviceTransformFeedbackPropertiesEXT::transformFeedbackRasterizationStreamSelect` is `VK_FALSE`

## Valid Usage (Implicit)

- VUID-vkCmdSetRasterizationStreamEXT-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdSetRasterizationStreamEXT-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdSetRasterizationStreamEXT-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- VUID-vkCmdSetRasterizationStreamEXT-videocoding
  This command **must** only be called outside of a video coding scope

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics | State |

## 28.3. Rasterization Order

Within a subpass of a render pass instance, for a given (x,y,layer,sample) sample location, the following operations are guaranteed to execute in *rasterization order*, for each separate primitive that includes that sample location:

1. Fragment operations, in the order defined
2. Blending, logic operations, and color writes

Execution of these operations for each primitive in a subpass occurs in an order determined by the application.

The rasterization order to use for a graphics pipeline is specified by adding a `VkPipelineRasterizationStateRasterizationOrderAMD` structure to the `pNext` chain of a VkPipelineRasterizationStateCreateInfo structure.

The `VkPipelineRasterizationStateRasterizationOrderAMD` structure is defined as:

```
// Provided by VK_AMD_rasterization_order
typedef struct VkPipelineRasterizationStateRasterizationOrderAMD {
    VkStructureType            sType;
    const void*                pNext;
    VkRasterizationOrderAMD    rasterizationOrder;
} VkPipelineRasterizationStateRasterizationOrderAMD;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `rasterizationOrder` is a VkRasterizationOrderAMD value specifying the primitive rasterization order to use.

### Valid Usage (Implicit)

- VUID-VkPipelineRasterizationStateRasterizationOrderAMD-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PIPELINE_RASTERIZATION_STATE_RASTERIZATION_ORDER_AMD`
- VUID-VkPipelineRasterizationStateRasterizationOrderAMD-rasterizationOrder-parameter
  `rasterizationOrder` **must** be a valid VkRasterizationOrderAMD value

If the `VK_AMD_rasterization_order` device extension is not enabled or the application does not request a particular rasterization order through specifying a `VkPipelineRasterizationStateRasterizationOrderAMD` structure then the rasterization order used by the graphics pipeline defaults to `VK_RASTERIZATION_ORDER_STRICT_AMD`.

Possible values of VkPipelineRasterizationStateRasterizationOrderAMD::`rasterizationOrder`, specifying the primitive rasterization order, are:

```
// Provided by VK_AMD_rasterization_order
typedef enum VkRasterizationOrderAMD {
    VK_RASTERIZATION_ORDER_STRICT_AMD = 0,
    VK_RASTERIZATION_ORDER_RELAXED_AMD = 1,
} VkRasterizationOrderAMD;
```

- `VK_RASTERIZATION_ORDER_STRICT_AMD` specifies that operations for each primitive in a subpass **must** occur in primitive order.
- `VK_RASTERIZATION_ORDER_RELAXED_AMD` specifies that operations for each primitive in a subpass **may** not occur in primitive order.

## 28.4. Multisampling

Multisampling is a mechanism to antialias all Vulkan primitives: points, lines, and polygons. The technique is to sample all primitives multiple times at each pixel. Each sample in each framebuffer attachment has storage for a color, depth, and/or stencil value, such that per-fragment operations apply to each sample independently. The color sample values **can** be later *resolved* to a single color (see Resolving Multisample Images and the Render Pass chapter for more details on how to resolve multisample images to non-multisample images).

Vulkan defines rasterization rules for single-sample modes in a way that is equivalent to a multisample mode with a single sample in the center of each fragment.

Each fragment includes a coverage mask with a single bit for each sample in the fragment, and a number of depth values and associated data for each sample.

It is understood that each pixel has `rasterizationSamples` locations associated with it. These locations are exact positions, rather than regions or areas, and each is referred to as a sample point. The sample points associated with a pixel **must** be located inside or on the boundary of the unit square that is considered to bound the pixel. Furthermore, the relative locations of sample points **may** be identical for each pixel in the framebuffer, or they **may** differ.

If the render pass has a fragment density map attachment, each fragment only has `rasterizationSamples` locations associated with it regardless of how many pixels are covered in the fragment area. Fragment sample locations are defined as if the fragment had an area of (1,1) and its sample points **must** be located within these bounds. Their actual location in the framebuffer is calculated by scaling the sample location by the fragment area. Attachments with storage for multiple samples per pixel are located at the pixel sample locations. Otherwise, the fragment's sample locations are generally used for evaluation of associated data and fragment operations.

If the current pipeline includes a fragment shader with one or more variables in its interface decorated with `Sample` and `Input`, the data associated with those variables will be assigned independently for each sample. The values for each sample **must** be evaluated at the location of the sample. The data associated with any other variables not decorated with `Sample` and `Input` need not be evaluated independently for each sample.

A *coverage mask* is generated for each fragment, based on which samples within that fragment are

determined to be within the area of the primitive that generated the fragment.

Single pixel fragments and multi-pixel fragments defined by a fragment density map have one set of samples. Multi-pixel fragments defined by a shading rate image have one set of samples per pixel. Multi-pixel fragments defined by setting the fragment shading rate have one set of samples per pixel. Each set of samples has a number of samples determined by VkPipelineMultisampleStateCreateInfo::rasterizationSamples. Each sample in a set is assigned a unique *sample index* i in the range [0, rasterizationSamples).

To dynamically set the rasterizationSamples, call:

```
// Provided by VK_EXT_extended_dynamic_state3, VK_EXT_shader_object
void vkCmdSetRasterizationSamplesEXT(
    VkCommandBuffer                             commandBuffer,
    VkSampleCountFlagBits                       rasterizationSamples);
```

- commandBuffer is the command buffer into which the command will be recorded.
- rasterizationSamples specifies rasterizationSamples.

This command sets the rasterizationSamples for subsequent drawing commands when drawing using shader objects, or when the graphics pipeline is created with VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT set in VkPipelineDynamicStateCreateInfo::pDynamicStates. Otherwise, this state is specified by the VkPipelineMultisampleStateCreateInfo::rasterizationSamples value used to create the currently active pipeline.

## Valid Usage

- VUID-vkCmdSetRasterizationSamplesEXT-None-09423
  At least one of the following **must** be true:
  - The extendedDynamicState3RasterizationSamples feature is enabled
  - The shaderObject feature is enabled

## Valid Usage (Implicit)

- VUID-vkCmdSetRasterizationSamplesEXT-commandBuffer-parameter
  commandBuffer **must** be a valid VkCommandBuffer handle
- VUID-vkCmdSetRasterizationSamplesEXT-rasterizationSamples-parameter
  rasterizationSamples **must** be a valid VkSampleCountFlagBits value
- VUID-vkCmdSetRasterizationSamplesEXT-commandBuffer-recording
  commandBuffer **must** be in the recording state
- VUID-vkCmdSetRasterizationSamplesEXT-commandBuffer-cmdpool
  The VkCommandPool that commandBuffer was allocated from **must** support graphics operations
- VUID-vkCmdSetRasterizationSamplesEXT-videocoding

This command **must** only be called outside of a video coding scope

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics | State |

Each sample in a fragment is also assigned a unique *coverage index* j in the range [0, n × `rasterizationSamples`), where n is the number of sets in the fragment. If the fragment contains a single set of samples, the *coverage index* is always equal to the *sample index*. If a shading rate image is used and a fragment covers multiple pixels, the coverage index is determined as defined by VkPipelineViewportCoarseSampleOrderStateCreateInfoNV or vkCmdSetCoarseSampleOrderNV.

If the fragment shading rate is set, the coverage index j is determined as a function of the *pixel index* p, the *sample index* i, and the number of rasterization samples r as:

$$j = i + r \times ((f_w \times f_h) - 1 - p)$$

where the pixel index p is determined as a function of the pixel's framebuffer location (x,y) and the fragment size $(f_w,f_h)$:

$$p_x = x \% f_w$$

$$p_y = y \% f_h$$

$$p = p_x + (p_y \times f_w)$$

The table below illustrates the pixel index for multi-pixel fragments:

*Table 35. Pixel indices - 1 wide*

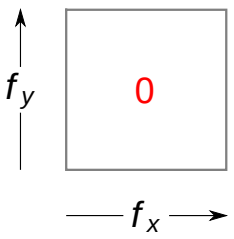
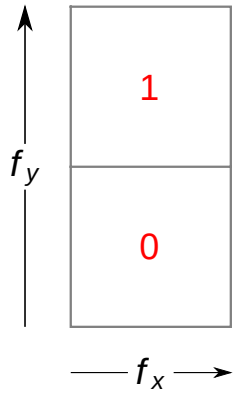
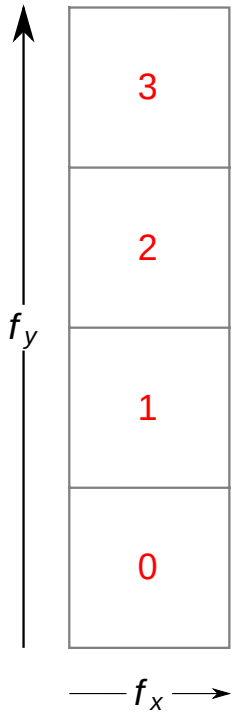
| 1x1 | 1x2 | 1x4 |
|---|---|---|
|   |   |   |

*Table 36. Pixel indices - 2 wide*

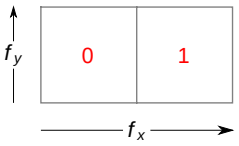
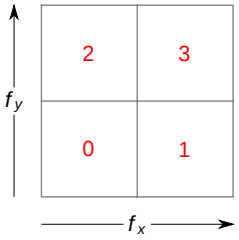
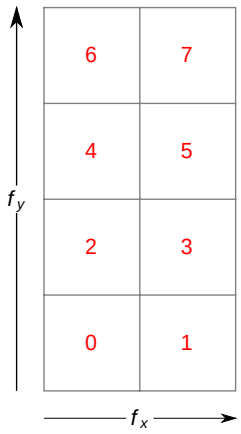
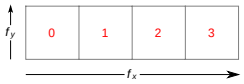
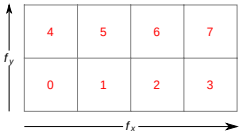
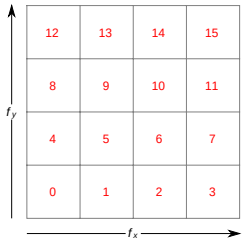
| 2x1 | 2x2 | 2x4 |
|---|---|---|
|   |   |   |

*Table 37. Pixel indices - 4 wide*

| 4x1 | 4x2 | 4x4 |
|---|---|---|
|   |   |   |

The coverage mask includes B bits packed into W words, defined as:

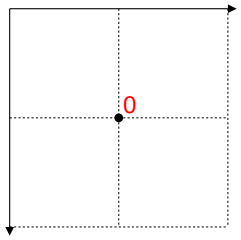
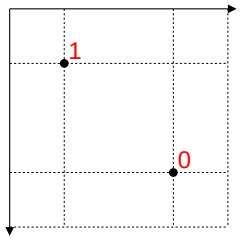
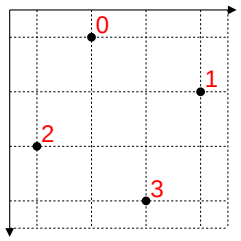
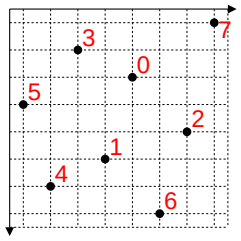
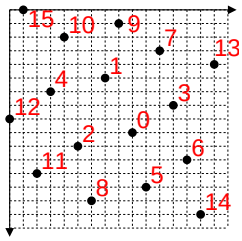
$$B = n \times \texttt{rasterizationSamples}$$

$$W = \lceil B/32 \rceil$$

Bit b in coverage mask word w is `1` if the sample with coverage index $j = 32 \times w + b$ is covered, and `0` otherwise.

If the `standardSampleLocations` member of VkPhysicalDeviceLimits is `VK_TRUE`, then the sample counts `VK_SAMPLE_COUNT_1_BIT`, `VK_SAMPLE_COUNT_2_BIT`, `VK_SAMPLE_COUNT_4_BIT`, `VK_SAMPLE_COUNT_8_BIT`, and `VK_SAMPLE_COUNT_16_BIT` have sample locations as listed in the following table, with the ith entry in the table corresponding to sample index i. `VK_SAMPLE_COUNT_32_BIT` and `VK_SAMPLE_COUNT_64_BIT` do not have standard sample locations. Locations are defined relative to an origin in the upper left corner of the fragment.

*Table 38. Standard sample locations*

| Sample count | Sample Locations | |
|---|---|---|
| VK_SAMPLE_COUNT_1_BIT | (0.5,0.5) |  |
| VK_SAMPLE_COUNT_2_BIT | (0.75,0.75)<br>(0.25,0.25) |  |
| VK_SAMPLE_COUNT_4_BIT | (0.375, 0.125)<br>(0.875, 0.375)<br>(0.125, 0.625)<br>(0.625, 0.875) |  |
| VK_SAMPLE_COUNT_8_BIT | (0.5625, 0.3125)<br>(0.4375, 0.6875)<br>(0.8125, 0.5625)<br>(0.3125, 0.1875)<br>(0.1875, 0.8125)<br>(0.0625, 0.4375)<br>(0.6875, 0.9375)<br>(0.9375, 0.0625) |  |
| VK_SAMPLE_COUNT_16_BIT | (0.5625, 0.5625)<br>(0.4375, 0.3125)<br>(0.3125, 0.625)<br>(0.75, 0.4375)<br>(0.1875, 0.375)<br>(0.625, 0.8125)<br>(0.8125, 0.6875)<br>(0.6875, 0.1875)<br>(0.375, 0.875)<br>(0.5, 0.0625)<br>(0.25, 0.125)<br>(0.125, 0.75)<br>(0.0, 0.5)<br>(0.9375, 0.25)<br>(0.875, 0.9375)<br>(0.0625, 0.0) |  |

Color images created with multiple samples per pixel use a compression technique where there are

two arrays of data associated with each pixel. The first array contains one element per sample where each element stores an index to the second array defining the *fragment mask* of the pixel. The second array contains one element per *color fragment* and each element stores a unique color value in the format of the image. With this compression technique it is not always necessary to actually use unique storage locations for each color sample: when multiple samples share the same color value the fragment mask **may** have two samples referring to the same color fragment. The number of color fragments is determined by the `samples` member of the VkImageCreateInfo structure used to create the image. The `VK_AMD_shader_fragment_mask` device extension provides shader instructions enabling the application to get direct access to the fragment mask and the individual color fragment values.

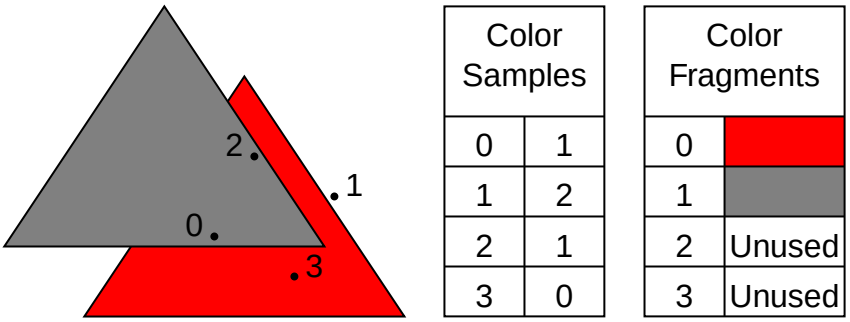



*Figure 19. Fragment Mask*

## 28.5. Custom Sample Locations

Applications **can** also control the sample locations used for rasterization.

If the `pNext` chain of the VkPipelineMultisampleStateCreateInfo structure specified at pipeline creation time includes a `VkPipelineSampleLocationsStateCreateInfoEXT` structure, then that structure controls the sample locations used when rasterizing primitives with the pipeline.

The `VkPipelineSampleLocationsStateCreateInfoEXT` structure is defined as:

```
// Provided by VK_EXT_sample_locations
typedef struct VkPipelineSampleLocationsStateCreateInfoEXT {
    VkStructureType             sType;
    const void*                 pNext;
    VkBool32                    sampleLocationsEnable;
    VkSampleLocationsInfoEXT    sampleLocationsInfo;
} VkPipelineSampleLocationsStateCreateInfoEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `sampleLocationsEnable` controls whether custom sample locations are used. If `sampleLocationsEnable` is `VK_FALSE`, the default sample locations are used and the values specified in `sampleLocationsInfo` are ignored.
- `sampleLocationsInfo` is the sample locations to use during rasterization if `sampleLocationsEnable` is `VK_TRUE` and the graphics pipeline is not created with `VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT`.

### Valid Usage (Implicit)

- VUID-VkPipelineSampleLocationsStateCreateInfoEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PIPELINE_SAMPLE_LOCATIONS_STATE_CREATE_INFO_EXT`
- VUID-VkPipelineSampleLocationsStateCreateInfoEXT-sampleLocationsInfo-parameter
  `sampleLocationsInfo` **must** be a valid VkSampleLocationsInfoEXT structure

The `VkSampleLocationsInfoEXT` structure is defined as:

```
// Provided by VK_EXT_sample_locations
typedef struct VkSampleLocationsInfoEXT {
    VkStructureType                 sType;
    const void*                     pNext;
    VkSampleCountFlagBits           sampleLocationsPerPixel;
    VkExtent2D                      sampleLocationGridSize;
    uint32_t                        sampleLocationsCount;
    const VkSampleLocationEXT*      pSampleLocations;
} VkSampleLocationsInfoEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `sampleLocationsPerPixel` is a VkSampleCountFlagBits value specifying the number of sample locations per pixel.
- `sampleLocationGridSize` is the size of the sample location grid to select custom sample locations for.
- `sampleLocationsCount` is the number of sample locations in `pSampleLocations`.
- `pSampleLocations` is a pointer to an array of `sampleLocationsCount` VkSampleLocationEXT structures.

This structure **can** be used either to specify the sample locations to be used for rendering or to specify the set of sample locations an image subresource has been last rendered with for the purposes of layout transitions of depth/stencil images created with `VK_IMAGE_CREATE_SAMPLE_LOCATIONS_COMPATIBLE_DEPTH_BIT_EXT`.

The sample locations in `pSampleLocations` specify `sampleLocationsPerPixel` number of sample locations for each pixel in the grid of the size specified in `sampleLocationGridSize`. The sample location for sample i at the pixel grid location (x,y) is taken from `pSampleLocations`[(x + y × `sampleLocationGridSize.width`) × `sampleLocationsPerPixel` + i].

If the render pass has a fragment density map, the implementation will choose the sample locations for the fragment and the contents of `pSampleLocations` **may** be ignored.

### Valid Usage

- VUID-VkSampleLocationsInfoEXT-sampleLocationsPerPixel-01526
  `sampleLocationsPerPixel` **must** be a valid VkSampleCountFlagBits value that is set in VkPhysicalDeviceSampleLocationsPropertiesEXT::`sampleLocationSampleCounts`
- VUID-VkSampleLocationsInfoEXT-sampleLocationsCount-01527
  `sampleLocationsCount` **must** equal `sampleLocationsPerPixel` × `sampleLocationGridSize.width` × `sampleLocationGridSize.height`

### Valid Usage (Implicit)

- VUID-VkSampleLocationsInfoEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_SAMPLE_LOCATIONS_INFO_EXT`
- VUID-VkSampleLocationsInfoEXT-pSampleLocations-parameter
  If `sampleLocationsCount` is not `0`, `pSampleLocations` **must** be a valid pointer to an array of `sampleLocationsCount` VkSampleLocationEXT structures

The `VkSampleLocationEXT` structure is defined as:

```
// Provided by VK_EXT_sample_locations
typedef struct VkSampleLocationEXT {
    float    x;
    float    y;
} VkSampleLocationEXT;
```

- `x` is the horizontal coordinate of the sample's location.
- `y` is the vertical coordinate of the sample's location.

The domain space of the sample location coordinates has an upper-left origin within the pixel in framebuffer space.

The values specified in a `VkSampleLocationEXT` structure are always clamped to the implementation-dependent sample location coordinate range [`sampleLocationCoordinateRange`[0],`sampleLocationCoordinateRange`[1]] that **can** be queried using VkPhysicalDeviceSampleLocationsPropertiesEXT.

To dynamically set the `sampleLocationsEnable` state, call:

```
// Provided by VK_EXT_extended_dynamic_state3 with VK_EXT_sample_locations,
VK_EXT_sample_locations with VK_EXT_shader_object
void vkCmdSetSampleLocationsEnableEXT(
    VkCommandBuffer                             commandBuffer,
    VkBool32                                    sampleLocationsEnable);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `sampleLocationsEnable` specifies the `sampleLocationsEnable` state.

This command sets the `sampleLocationsEnable` state for subsequent drawing commands when drawing using shader objects, or when the graphics pipeline is created with `VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT` set in VkPipelineDynamicStateCreateInfo::`pDynamicStates`. Otherwise, this state is specified by the VkPipelineSampleLocationsStateCreateInfoEXT::`sampleLocationsEnable` value used to create the currently active pipeline.

## Valid Usage

- VUID-vkCmdSetSampleLocationsEnableEXT-None-09423
  At least one of the following **must** be true:
  - The `extendedDynamicState3SampleLocationsEnable` feature is enabled
  - The `shaderObject` feature is enabled

## Valid Usage (Implicit)

- VUID-vkCmdSetSampleLocationsEnableEXT-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdSetSampleLocationsEnableEXT-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdSetSampleLocationsEnableEXT-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- VUID-vkCmdSetSampleLocationsEnableEXT-videocoding
  This command **must** only be called outside of a video coding scope

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics | State |

To dynamically set the sample locations used for rasterization, call:

```
// Provided by VK_EXT_sample_locations
void vkCmdSetSampleLocationsEXT(
    VkCommandBuffer                             commandBuffer,
    const VkSampleLocationsInfoEXT*             pSampleLocationsInfo);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `pSampleLocationsInfo` is the sample locations state to set.

This command sets the custom sample locations for subsequent drawing commands when drawing using shader objects, or when the graphics pipeline is created with `VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT` set in VkPipelineDynamicStateCreateInfo::`pDynamicStates`, and when the VkPipelineSampleLocationsStateCreateInfoEXT::`sampleLocationsEnable` property of the bound graphics pipeline is `VK_TRUE`. Otherwise, this state is specified by the VkPipelineSampleLocationsStateCreateInfoEXT::`sampleLocationsInfo` values used to create the currently active pipeline.

## Valid Usage

- VUID-vkCmdSetSampleLocationsEXT-variableSampleLocations-01530
  If VkPhysicalDeviceSampleLocationsPropertiesEXT::`variableSampleLocations` is `VK_FALSE` then the current render pass **must** have been begun by specifying a VkRenderPassSampleLocationsBeginInfoEXT structure whose `pPostSubpassSampleLocations` member contains an element with a `subpassIndex` matching the current subpass index and the `sampleLocationsInfo` member of that element **must** match the sample locations state pointed to by `pSampleLocationsInfo`

## Valid Usage (Implicit)

- VUID-vkCmdSetSampleLocationsEXT-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdSetSampleLocationsEXT-pSampleLocationsInfo-parameter
  `pSampleLocationsInfo` **must** be a valid pointer to a valid VkSampleLocationsInfoEXT structure
- VUID-vkCmdSetSampleLocationsEXT-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdSetSampleLocationsEXT-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- VUID-vkCmdSetSampleLocationsEXT-videocoding
  This command **must** only be called outside of a video coding scope

### Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

### Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics | State |

## 28.6. Fragment Shading Rates

The features advertised by VkPhysicalDeviceFragmentShadingRateFeaturesKHR allow an application to control the shading rate of a given fragment shader invocation.

The fragment shading rate strongly interacts with Multisampling, and the set of available rates for an implementation **may** be restricted by sample rate.

To query available shading rates, call:

```
// Provided by VK_KHR_fragment_shading_rate
VkResult vkGetPhysicalDeviceFragmentShadingRatesKHR(
    VkPhysicalDevice                            physicalDevice,
    uint32_t*                                   pFragmentShadingRateCount,
    VkPhysicalDeviceFragmentShadingRateKHR*     pFragmentShadingRates);
```

- `physicalDevice` is the handle to the physical device whose properties will be queried.
- `pFragmentShadingRateCount` is a pointer to an integer related to the number of fragment shading rates available or queried, as described below.
- `pFragmentShadingRates` is either `NULL` or a pointer to an array of VkPhysicalDeviceFragmentShadingRateKHR structures.

If `pFragmentShadingRates` is `NULL`, then the number of fragment shading rates available is returned in `pFragmentShadingRateCount`. Otherwise, `pFragmentShadingRateCount` **must** point to a variable set by the application to the number of elements in the `pFragmentShadingRates` array, and on return the variable is overwritten with the number of structures actually written to `pFragmentShadingRates`. If `pFragmentShadingRateCount` is less than the number of fragment shading rates available, at most `pFragmentShadingRateCount` structures will be written, and `VK_INCOMPLETE` will be returned instead of `VK_SUCCESS`, to indicate that not all the available fragment shading rates were returned.

The returned array of fragment shading rates **must** be ordered from largest `fragmentSize.width` value to smallest, and each set of fragment shading rates with the same `fragmentSize.width` value **must** be ordered from largest `fragmentSize.height` to smallest. Any two entries in the array **must** not have the same `fragmentSize` values.

For any entry in the array, the following rules also apply:

- The value of `fragmentSize.width` **must** be less than or equal to `maxFragmentSize.width`.
- The value of `fragmentSize.width` **must** be greater than or equal to `1`.
- The value of `fragmentSize.width` **must** be a power-of-two.
- The value of `fragmentSize.height` **must** be less than or equal to `maxFragmentSize.height`.
- The value of `fragmentSize.height` **must** be greater than or equal to `1`.
- The value of `fragmentSize.height` **must** be a power-of-two.
- The highest sample count in `sampleCounts` **must** be less than or equal to `maxFragmentShadingRateRasterizationSamples`.
- The product of `fragmentSize.width`, `fragmentSize.height`, and the highest sample count in `sampleCounts` **must** be less than or equal to `maxFragmentShadingRateCoverageSamples`.

Implementations **must** support at least the following shading rates:

| `sampleCounts` | `fragmentSize` |
|---|---|
| `VK_SAMPLE_COUNT_1_BIT` \| `VK_SAMPLE_COUNT_4_BIT` | {2,2} |
| `VK_SAMPLE_COUNT_1_BIT` \| `VK_SAMPLE_COUNT_4_BIT` | {2,1} |
| ~0 | {1,1} |

If `framebufferColorSampleCounts`, includes `VK_SAMPLE_COUNT_2_BIT`, the required rates **must** also include `VK_SAMPLE_COUNT_2_BIT`.

**NOTE** Including the {1,1} fragment size is done for completeness; it has no actual effect on the support of rendering without setting the fragment size. All sample counts and render pass transforms are supported for this rate.

The returned set of fragment shading rates **must** be returned in the native (rotated) coordinate system. For rasterization using render pass `transform` not equal to `VK_SURFACE_TRANSFORM_IDENTITY_BIT_KHR`, the application **must** transform the returned fragment shading rates into the current (unrotated) coordinate system to get the supported rates for that transform.

**NOTE** For example, consider an implementation returning support for 4x2, but not 2x4 in the set of supported fragment shading rates. This means that for transforms `VK_SURFACE_TRANSFORM_ROTATE_90_BIT_KHR` and `VK_SURFACE_TRANSFORM_ROTATE_270_BIT_KHR`, 2x4 is a supported rate, but 4x2 is an unsupported rate.

### Valid Usage (Implicit)

- VUID-vkGetPhysicalDeviceFragmentShadingRatesKHR-physicalDevice-parameter
  `physicalDevice` **must** be a valid VkPhysicalDevice handle
- VUID-vkGetPhysicalDeviceFragmentShadingRatesKHR-pFragmentShadingRateCount-parameter
  `pFragmentShadingRateCount` **must** be a valid pointer to a `uint32_t` value
- VUID-vkGetPhysicalDeviceFragmentShadingRatesKHR-pFragmentShadingRates-parameter
  If the value referenced by `pFragmentShadingRateCount` is not `0`, and `pFragmentShadingRates` is not `NULL`, `pFragmentShadingRates` **must** be a valid pointer to an array of `pFragmentShadingRateCount` VkPhysicalDeviceFragmentShadingRateKHR structures

### Return Codes

**Success**

- `VK_SUCCESS`
- `VK_INCOMPLETE`

**Failure**

- `VK_ERROR_OUT_OF_HOST_MEMORY`

The `VkPhysicalDeviceFragmentShadingRateKHR` structure is defined as

```
// Provided by VK_KHR_fragment_shading_rate
typedef struct VkPhysicalDeviceFragmentShadingRateKHR {
    VkStructureType       sType;
    void*                 pNext;
    VkSampleCountFlags    sampleCounts;
    VkExtent2D            fragmentSize;
} VkPhysicalDeviceFragmentShadingRateKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `sampleCounts` is a bitmask of sample counts for which the shading rate described by `fragmentSize` is supported.
- `fragmentSize` is a VkExtent2D describing the width and height of a supported shading rate.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceFragmentShadingRateKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_FRAGMENT_SHADING_RATE_KHR`

- VUID-VkPhysicalDeviceFragmentShadingRateKHR-pNext-pNext
  `pNext` **must** be `NULL`

Fragment shading rates **can** be set at three points, with the three rates combined to determine the final shading rate.

### 28.6.1. Pipeline Fragment Shading Rate

The *pipeline fragment shading rate* **can** be set on a per-draw basis by either setting the rate in a graphics pipeline, or dynamically via vkCmdSetFragmentShadingRateKHR.

The `VkPipelineFragmentShadingRateStateCreateInfoKHR` structure is defined as:

```
// Provided by VK_KHR_fragment_shading_rate
typedef struct VkPipelineFragmentShadingRateStateCreateInfoKHR {
    VkStructureType                         sType;
    const void*                             pNext;
    VkExtent2D                              fragmentSize;
    VkFragmentShadingRateCombinerOpKHR      combinerOps[2];
} VkPipelineFragmentShadingRateStateCreateInfoKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `fragmentSize` specifies a VkExtent2D structure containing the fragment size used to define the pipeline fragment shading rate for drawing commands using this pipeline.
- `combinerOps` specifies a VkFragmentShadingRateCombinerOpKHR value determining how the pipeline, primitive, and attachment shading rates are combined for fragments generated by drawing commands using the created pipeline.

If the `pNext` chain of VkGraphicsPipelineCreateInfo includes a `VkPipelineFragmentShadingRateStateCreateInfoKHR` structure, then that structure includes parameters controlling the pipeline fragment shading rate.

If this structure is not present, `fragmentSize` is considered to be equal to (1,1), and both elements of `combinerOps` are considered to be equal to `VK_FRAGMENT_SHADING_RATE_COMBINER_OP_KEEP_KHR`.

**Valid Usage (Implicit)**

- VUID-VkPipelineFragmentShadingRateStateCreateInfoKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PIPELINE_FRAGMENT_SHADING_RATE_STATE_CREATE_INFO_KHR`

To dynamically set the pipeline fragment shading rate and combiner operation, call:

```
// Provided by VK_KHR_fragment_shading_rate
void vkCmdSetFragmentShadingRateKHR(
    VkCommandBuffer                             commandBuffer,
    const VkExtent2D*                           pFragmentSize,
    const VkFragmentShadingRateCombinerOpKHR    combinerOps[2]);
```

- commandBuffer is the command buffer into which the command will be recorded.
- pFragmentSize specifies the pipeline fragment shading rate for subsequent drawing commands.
- combinerOps specifies a VkFragmentShadingRateCombinerOpKHR determining how the pipeline, primitive, and attachment shading rates are combined for fragments generated by subsequent drawing commands.

This command sets the pipeline fragment shading rate and combiner operation for subsequent drawing commands when drawing using shader objects, or when the graphics pipeline is created with VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR set in VkPipelineDynamicStateCreateInfo::pDynamicStates. Otherwise, this state is specified by the VkPipelineFragmentShadingRateStateCreateInfoKHR values used to create the currently active pipeline.

## Valid Usage

- VUID-vkCmdSetFragmentShadingRateKHR-pipelineFragmentShadingRate-04507
  If pipelineFragmentShadingRate is not enabled, pFragmentSize->width **must** be 1
- VUID-vkCmdSetFragmentShadingRateKHR-pipelineFragmentShadingRate-04508
  If pipelineFragmentShadingRate is not enabled, pFragmentSize->height **must** be 1
- VUID-vkCmdSetFragmentShadingRateKHR-pipelineFragmentShadingRate-04509
  One of pipelineFragmentShadingRate, primitiveFragmentShadingRate, or attachmentFragmentShadingRate **must** be enabled
- VUID-vkCmdSetFragmentShadingRateKHR-primitiveFragmentShadingRate-04510
  If the primitiveFragmentShadingRate feature is not enabled, combinerOps[0] **must** be VK_FRAGMENT_SHADING_RATE_COMBINER_OP_KEEP_KHR
- VUID-vkCmdSetFragmentShadingRateKHR-attachmentFragmentShadingRate-04511
  If the attachmentFragmentShadingRate feature is not enabled, combinerOps[1] **must** be VK_FRAGMENT_SHADING_RATE_COMBINER_OP_KEEP_KHR
- VUID-vkCmdSetFragmentShadingRateKHR-fragmentSizeNonTrivialCombinerOps-04512
  If the fragmentSizeNonTrivialCombinerOps limit is not supported, elements of combinerOps **must** be either VK_FRAGMENT_SHADING_RATE_COMBINER_OP_KEEP_KHR or VK_FRAGMENT_SHADING_RATE_COMBINER_OP_REPLACE_KHR
- VUID-vkCmdSetFragmentShadingRateKHR-pFragmentSize-04513
  pFragmentSize->width **must** be greater than or equal to 1
- VUID-vkCmdSetFragmentShadingRateKHR-pFragmentSize-04514
  pFragmentSize->height **must** be greater than or equal to 1
- VUID-vkCmdSetFragmentShadingRateKHR-pFragmentSize-04515

`pFragmentSize->width` **must** be a power-of-two value

- VUID-vkCmdSetFragmentShadingRateKHR-pFragmentSize-04516
  `pFragmentSize->height` **must** be a power-of-two value
- VUID-vkCmdSetFragmentShadingRateKHR-pFragmentSize-04517
  `pFragmentSize->width` **must** be less than or equal to `4`
- VUID-vkCmdSetFragmentShadingRateKHR-pFragmentSize-04518
  `pFragmentSize->height` **must** be less than or equal to `4`

## Valid Usage (Implicit)

- VUID-vkCmdSetFragmentShadingRateKHR-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdSetFragmentShadingRateKHR-pFragmentSize-parameter
  `pFragmentSize` **must** be a valid pointer to a valid VkExtent2D structure
- VUID-vkCmdSetFragmentShadingRateKHR-combinerOps-parameter
  Each element of `combinerOps` **must** be a valid VkFragmentShadingRateCombinerOpKHR value
- VUID-vkCmdSetFragmentShadingRateKHR-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdSetFragmentShadingRateKHR-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- VUID-vkCmdSetFragmentShadingRateKHR-videocoding
  This command **must** only be called outside of a video coding scope

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics | State |

### 28.6.2. Primitive Fragment Shading Rate

The *primitive fragment shading rate* **can** be set via the `PrimitiveShadingRateKHR` built-in in the last active pre-rasterization shader stage. If the last pre-rasterization shader stage is using the `MeshEXT Execution Model`, the rate associated with a given primitive is sourced from the value written to the per-primitive `PrimitiveShadingRateKHR`. Otherwise the rate associated with a given primitive is sourced from the value written to `PrimitiveShadingRateKHR` by that primitive's provoking vertex.

### 28.6.3. Attachment Fragment Shading Rate

The *attachment shading rate* **can** be set by including VkFragmentShadingRateAttachmentInfoKHR in a subpass to define a *fragment shading rate attachment*. Each pixel in the framebuffer is assigned an attachment fragment shading rate by the corresponding texel in the fragment shading rate attachment, according to:

$$x' = \mathrm{floor}(x / \mathrm{region}_x)$$

$$y' = \mathrm{floor}(y / \mathrm{region}_y)$$

where x' and y' are the coordinates of a texel in the fragment shading rate attachment, x and y are the coordinates of the pixel in the framebuffer, and $\mathrm{region}_x$ and $\mathrm{region}_y$ are the size of the region each texel corresponds to, as defined by the `shadingRateAttachmentTexelSize` member of VkFragmentShadingRateAttachmentInfoKHR.

If multiview is enabled and the shading rate attachment has multiple layers, the shading rate attachment texel is selected using layer = `ViewIndex`. If multiview is disabled, and both the shading rate attachment and the framebuffer have multiple layers, the shading rate attachment texel is selected using layer = `Layer`. Otherwise, layer = 0.

The texel is read from the fragment shading rate attachment image as a texture input operation without a sampler, using integer coordinates i = x', j = y', k = 0, l = layer, and s = 0. The fragment size is encoded into the first component of the result of that operation as follows:

$$\mathrm{size}_w = 2^{((\mathrm{texel}/4)\&3)}$$

$$\mathrm{size}_h = 2^{(\mathrm{texel}\&3)}$$

where texel is the value in the first component of the returned value, and $\mathrm{size}_w$ and $\mathrm{size}_h$ are the width and height of the fragment size, decoded from the texel.

If no fragment shading rate attachment is specified, this size is calculated as $\mathrm{size}_w = \mathrm{size}_h = 1$. Applications **must** not specify a width or height greater than 4 by this method.

The *Fragment Shading Rate* enumeration in SPIR-V adheres to the above encoding.

### 28.6.4. Combining the Fragment Shading Rates

The final rate ($C_{xy}'$) used for fragment shading **must** be one of the rates returned by vkGetPhysicalDeviceFragmentShadingRatesKHR for the sample count and render pass transform used by rasterization.

If any of the following conditions are met, $C_{xy}'$ **must** be set to {1,1} by the implementation:

- If Sample Shading is enabled.
- The `fragmentShadingRateWithSampleMask` limit is not supported, and VkPipelineMultisampleStateCreateInfo::`pSampleMask` contains a zero value in any bit used by fragment operations.
- The `fragmentShadingRateWithShaderSampleMask` is not supported, and the fragment shader has `SampleMask` in the input or output interface.
- The `fragmentShadingRateWithShaderDepthStencilWrites` limit is not supported, and the fragment shader declares the `FragDepth` or `FragStencilRefEXT` built-in.
- The `fragmentShadingRateWithConservativeRasterization` limit is not supported, and VkPipelineRasterizationConservativeStateCreateInfoEXT::`conservativeRasterizationMode` is not `VK_CONSERVATIVE_RASTERIZATION_MODE_DISABLED_EXT`.
- The `fragmentShadingRateWithFragmentShaderInterlock` limit is not supported, and the fragment shader declares any of the fragment shader interlock execution modes.
- The `fragmentShadingRateWithCustomSampleLocations` limit is not supported, and VkPipelineSampleLocationsStateCreateInfoEXT::`sampleLocationsEnable` is `VK_TRUE`.
- The fragment shader declares any of the `TileImageColorReadAccessEXT`, `TileImageDepthReadAccessEXT`, or `TileImageStencilReadAccessEXT` capabilities.

Otherwise, each of the specified shading rates are combined and then used to derive the value of $C_{xy}'$. As there are three ways to specify shading rates, two combiner operations are specified - between the pipeline and primitive shading rates, and between the result of that and the attachment shading rate.

The equation used for each combiner operation is defined by `VkFragmentShadingRateCombinerOpKHR`:

```
// Provided by VK_KHR_fragment_shading_rate
typedef enum VkFragmentShadingRateCombinerOpKHR {
    VK_FRAGMENT_SHADING_RATE_COMBINER_OP_KEEP_KHR = 0,
    VK_FRAGMENT_SHADING_RATE_COMBINER_OP_REPLACE_KHR = 1,
    VK_FRAGMENT_SHADING_RATE_COMBINER_OP_MIN_KHR = 2,
    VK_FRAGMENT_SHADING_RATE_COMBINER_OP_MAX_KHR = 3,
    VK_FRAGMENT_SHADING_RATE_COMBINER_OP_MUL_KHR = 4,
} VkFragmentShadingRateCombinerOpKHR;
```

- `VK_FRAGMENT_SHADING_RATE_COMBINER_OP_KEEP_KHR` specifies a combiner operation of combine($A_{xy}$,$B_{xy}$) = $A_{xy}$.
- `VK_FRAGMENT_SHADING_RATE_COMBINER_OP_REPLACE_KHR` specifies a combiner operation of

combine($A_{xy}$,$B_{xy}$) = $B_{xy}$.

- VK_FRAGMENT_SHADING_RATE_COMBINER_OP_MIN_KHR specifies a combiner operation of combine($A_{xy}$,$B_{xy}$) = min($A_{xy}$,$B_{xy}$).
- VK_FRAGMENT_SHADING_RATE_COMBINER_OP_MAX_KHR specifies a combiner operation of combine($A_{xy}$,$B_{xy}$) = max($A_{xy}$,$B_{xy}$).
- VK_FRAGMENT_SHADING_RATE_COMBINER_OP_MUL_KHR specifies a combiner operation of combine($A_{xy}$,$B_{xy}$) = $A_{xy}$*$B_{xy}$.

where combine($A_{xy}$,$B_{xy}$) is the combine operation, and $A_{xy}$ and $B_{xy}$ are the inputs to the operation.

If fragmentShadingRateStrictMultiplyCombiner is VK_FALSE, using VK_FRAGMENT_SHADING_RATE_COMBINER_OP_MUL_KHR with values of 1 for both A and B in the same dimension results in the value 2 being produced for that dimension. See the definition of fragmentShadingRateStrictMultiplyCombiner for more information.

These operations are performed in a component-wise fashion.

This is used to generate a combined fragment area using the equation:

$$C_{xy} = \text{combine}(A_{xy},B_{xy})$$

where $C_{xy}$ is the combined fragment area result, and $A_{xy}$ and $B_{xy}$ are the fragment areas of the fragment shading rates being combined.

Two combine operations are performed, first with $A_{xy}$ equal to the pipeline fragment shading rate and $B_{xy}$ equal to the primitive fragment shading rate, with the combine() operation selected by combinerOps[0]. A second combination is then performed, with $A_{xy}$ equal to the result of the first combination and $B_{xy}$ equal to the attachment fragment shading rate, with the combine() operation selected by combinerOps[1]. The result of the second combination is used as the final fragment shading rate, reported via the ShadingRateKHR built-in.

Implementations **should** clamp the inputs to the combiner operations $A_{xy}$ and $B_{xy}$, and **must** do so if VkPhysicalDeviceMaintenance6PropertiesKHR::fragmentShadingRateClampCombinerInputs is set to VK_TRUE. All implementations **must** clamp the result of the second combiner operation.

A fragment shading rate $R_{xy}$ representing any of $A_{xy}$, $B_{xy}$ or $C_{xy}$ is clamped as follows. If $R_{xy}$ is one of the rates returned by vkGetPhysicalDeviceFragmentShadingRatesKHR for the sample count and render pass transform used by rasterization, the clamped shading rate $R_{xy}'$ is $R_{xy}$. Otherwise, the clamped shading rate is selected from the rates returned by vkGetPhysicalDeviceFragmentShadingRatesKHR for the sample count and render pass transform used by rasterization. From this list of supported rates, the following steps are applied in order, to select a single value:

1. Keep only rates where $R_x' \le R_x$ and $R_y' \le R_y$.
   - Implementations **may** also keep rates where $R_x' \le R_y$ and $R_y' \le R_x$.
2. Keep only rates with the highest area ($R_x' \times R_y'$).

3. Keep only rates with the lowest aspect ratio ($R_x' + R_y'$).
4. In cases where a wide (e.g. 4x1) and tall (e.g. 1x4) rate remain, the implementation **may** choose either rate. However, it **must** choose this rate consistently for the same shading rates, render pass transform, and combiner operations for the lifetime of the VkDevice.

### 28.6.5. Extended Fragment Shading Rates

The features advertised by VkPhysicalDeviceFragmentShadingRateEnumsFeaturesNV provide support for additional fragment shading rates beyond those specifying one fragment shader invocation covering all pixels in a fragment whose size is indicated by the fragment shading rate.

If the fragmentShadingRateEnums feature is enabled, fragment shading rates may be specified using the VkFragmentShadingRateNV enumerated type defined as:

```
// Provided by VK_NV_fragment_shading_rate_enums
typedef enum VkFragmentShadingRateNV {
    VK_FRAGMENT_SHADING_RATE_1_INVOCATION_PER_PIXEL_NV = 0,
    VK_FRAGMENT_SHADING_RATE_1_INVOCATION_PER_1X2_PIXELS_NV = 1,
    VK_FRAGMENT_SHADING_RATE_1_INVOCATION_PER_2X1_PIXELS_NV = 4,
    VK_FRAGMENT_SHADING_RATE_1_INVOCATION_PER_2X2_PIXELS_NV = 5,
    VK_FRAGMENT_SHADING_RATE_1_INVOCATION_PER_2X4_PIXELS_NV = 6,
    VK_FRAGMENT_SHADING_RATE_1_INVOCATION_PER_4X2_PIXELS_NV = 9,
    VK_FRAGMENT_SHADING_RATE_1_INVOCATION_PER_4X4_PIXELS_NV = 10,
    VK_FRAGMENT_SHADING_RATE_2_INVOCATIONS_PER_PIXEL_NV = 11,
    VK_FRAGMENT_SHADING_RATE_4_INVOCATIONS_PER_PIXEL_NV = 12,
    VK_FRAGMENT_SHADING_RATE_8_INVOCATIONS_PER_PIXEL_NV = 13,
    VK_FRAGMENT_SHADING_RATE_16_INVOCATIONS_PER_PIXEL_NV = 14,
    VK_FRAGMENT_SHADING_RATE_NO_INVOCATIONS_NV = 15,
} VkFragmentShadingRateNV;
```

- VK_FRAGMENT_SHADING_RATE_1_INVOCATION_PER_PIXEL_NV specifies a fragment size of 1x1 pixels.
- VK_FRAGMENT_SHADING_RATE_1_INVOCATION_PER_1X2_PIXELS_NV specifies a fragment size of 1x2 pixels.
- VK_FRAGMENT_SHADING_RATE_1_INVOCATION_PER_2X1_PIXELS_NV specifies a fragment size of 2x1 pixels.
- VK_FRAGMENT_SHADING_RATE_1_INVOCATION_PER_2X2_PIXELS_NV specifies a fragment size of 2x2 pixels.
- VK_FRAGMENT_SHADING_RATE_1_INVOCATION_PER_2X4_PIXELS_NV specifies a fragment size of 2x4 pixels.
- VK_FRAGMENT_SHADING_RATE_1_INVOCATION_PER_4X2_PIXELS_NV specifies a fragment size of 4x2 pixels.
- VK_FRAGMENT_SHADING_RATE_1_INVOCATION_PER_4X4_PIXELS_NV specifies a fragment size of 4x4 pixels.
- VK_FRAGMENT_SHADING_RATE_2_INVOCATIONS_PER_PIXEL_NV specifies a fragment size of 1x1 pixels, with two fragment shader invocations per fragment.

- VK_FRAGMENT_SHADING_RATE_4_INVOCATIONS_PER_PIXEL_NV specifies a fragment size of 1x1 pixels, with four fragment shader invocations per fragment.
- VK_FRAGMENT_SHADING_RATE_8_INVOCATIONS_PER_PIXEL_NV specifies a fragment size of 1x1 pixels, with eight fragment shader invocations per fragment.
- VK_FRAGMENT_SHADING_RATE_16_INVOCATIONS_PER_PIXEL_NV specifies a fragment size of 1x1 pixels, with sixteen fragment shader invocations per fragment.
- VK_FRAGMENT_SHADING_RATE_NO_INVOCATIONS_NV specifies that any portions of a primitive that use that shading rate should be discarded without invoking any fragment shader.

To use the shading rates VK_FRAGMENT_SHADING_RATE_2_INVOCATIONS_PER_PIXEL_NV, VK_FRAGMENT_SHADING_RATE_4_INVOCATIONS_PER_PIXEL_NV, VK_FRAGMENT_SHADING_RATE_8_INVOCATIONS_PER_PIXEL_NV, and VK_FRAGMENT_SHADING_RATE_16_INVOCATIONS_PER_PIXEL_NV as a pipeline, primitive, or attachment shading rate, the supersampleFragmentShadingRates feature **must** be enabled. To use the shading rate VK_FRAGMENT_SHADING_RATE_NO_INVOCATIONS_NV as a pipeline, primitive, or attachment shading rate, the noInvocationFragmentShadingRates feature **must** be enabled.

When using fragment shading rate enums, the pipeline fragment shading rate **can** be set on a per-draw basis by either setting the rate in a graphics pipeline, or dynamically via vkCmdSetFragmentShadingRateEnumNV.

The VkPipelineFragmentShadingRateEnumStateCreateInfoNV structure is defined as:

```
// Provided by VK_NV_fragment_shading_rate_enums
typedef struct VkPipelineFragmentShadingRateEnumStateCreateInfoNV {
    VkStructureType                         sType;
    const void*                             pNext;
    VkFragmentShadingRateTypeNV             shadingRateType;
    VkFragmentShadingRateNV                 shadingRate;
    VkFragmentShadingRateCombinerOpKHR      combinerOps[2];
} VkPipelineFragmentShadingRateEnumStateCreateInfoNV;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- shadingRateType specifies a VkFragmentShadingRateTypeNV value indicating whether fragment shading rates are specified using fragment sizes or VkFragmentShadingRateNV enums.
- shadingRate specifies a VkFragmentShadingRateNV value indicating the pipeline fragment shading rate.
- combinerOps specifies VkFragmentShadingRateCombinerOpKHR values determining how the pipeline, primitive, and attachment shading rates are combined for fragments generated by drawing commands using the created pipeline.

If the pNext chain of VkGraphicsPipelineCreateInfo includes a VkPipelineFragmentShadingRateEnumStateCreateInfoNV structure, then that structure includes parameters controlling the pipeline fragment shading rate.

If this structure is not present, shadingRateType is considered to be equal to VK_FRAGMENT_SHADING_RATE_TYPE_FRAGMENT_SIZE_NV, shadingRate is considered to be equal to VK_FRAGMENT_SHADING_RATE_1_INVOCATION_PER_PIXEL_NV, and both elements of combinerOps are considered to be equal to VK_FRAGMENT_SHADING_RATE_COMBINER_OP_KEEP_KHR.

### Valid Usage (Implicit)

- VUID-VkPipelineFragmentShadingRateEnumStateCreateInfoNV-sType-sType sType **must** be VK_STRUCTURE_TYPE_PIPELINE_FRAGMENT_SHADING_RATE_ENUM_STATE_CREATE_INFO_NV

The VkFragmentShadingRateTypeNV enumerated type specifies whether a graphics pipeline gets its pipeline fragment shading rates and combiners from the VkPipelineFragmentShadingRateEnumStateCreateInfoNV structure or the VkPipelineFragmentShadingRateStateCreateInfoKHR structure.

```
// Provided by VK_NV_fragment_shading_rate_enums
typedef enum VkFragmentShadingRateTypeNV {
    VK_FRAGMENT_SHADING_RATE_TYPE_FRAGMENT_SIZE_NV = 0,
    VK_FRAGMENT_SHADING_RATE_TYPE_ENUMS_NV = 1,
} VkFragmentShadingRateTypeNV;
```

- VK_FRAGMENT_SHADING_RATE_TYPE_FRAGMENT_SIZE_NV specifies that a graphics pipeline should obtain its pipeline fragment shading rate and shading rate combiner state from the VkPipelineFragmentShadingRateStateCreateInfoKHR structure and that any state specified by the VkPipelineFragmentShadingRateEnumStateCreateInfoNV structure should be ignored.
- VK_FRAGMENT_SHADING_RATE_TYPE_ENUMS_NV specifies that a graphics pipeline should obtain its pipeline fragment shading rate and shading rate combiner state from the VkPipelineFragmentShadingRateEnumStateCreateInfoNV structure and that any state specified by the VkPipelineFragmentShadingRateStateCreateInfoKHR structure should be ignored.

To dynamically set the pipeline fragment shading rate and combiner operation, call:

```
// Provided by VK_NV_fragment_shading_rate_enums
void vkCmdSetFragmentShadingRateEnumNV(
    VkCommandBuffer                             commandBuffer,
    VkFragmentShadingRateNV                     shadingRate,
    const VkFragmentShadingRateCombinerOpKHR    combinerOps[2]);
```

- commandBuffer is the command buffer into which the command will be recorded.
- shadingRate specifies a VkFragmentShadingRateNV enum indicating the pipeline fragment shading rate for subsequent drawing commands.
- combinerOps specifies a VkFragmentShadingRateCombinerOpKHR determining how the pipeline, primitive, and attachment shading rates are combined for fragments generated by subsequent drawing commands.

This command sets the pipeline fragment shading rate and combiner operation for subsequent drawing commands when drawing using shader objects, or when the graphics pipeline is created with VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR set in VkPipelineDynamicStateCreateInfo::pDynamicStates. Otherwise, this state is specified by the VkPipelineFragmentShadingRateEnumStateCreateInfoNV values used to create the currently active pipeline.

| | |
|---|---|
| **NOTE** | This command allows specifying additional shading rates beyond those supported by vkCmdSetFragmentShadingRateKHR. For more information, refer to the VK_NV_fragment_shading_rate_enums appendix. |

## Valid Usage

- VUID-vkCmdSetFragmentShadingRateEnumNV-pipelineFragmentShadingRate-04576
  If pipelineFragmentShadingRate is not enabled, shadingRate **must** be VK_FRAGMENT_SHADING_RATE_1_INVOCATION_PER_PIXEL_NV
- VUID-vkCmdSetFragmentShadingRateEnumNV-supersampleFragmentShadingRates-04577
  If supersampleFragmentShadingRates is not enabled, shadingRate **must** not be VK_FRAGMENT_SHADING_RATE_2_INVOCATIONS_PER_PIXEL_NV, VK_FRAGMENT_SHADING_RATE_4_INVOCATIONS_PER_PIXEL_NV, VK_FRAGMENT_SHADING_RATE_8_INVOCATIONS_PER_PIXEL_NV, or VK_FRAGMENT_SHADING_RATE_16_INVOCATIONS_PER_PIXEL_NV
- VUID-vkCmdSetFragmentShadingRateEnumNV-noInvocationFragmentShadingRates-04578
  If noInvocationFragmentShadingRates is not enabled, shadingRate **must** not be VK_FRAGMENT_SHADING_RATE_NO_INVOCATIONS_NV
- VUID-vkCmdSetFragmentShadingRateEnumNV-fragmentShadingRateEnums-04579
  The fragmentShadingRateEnums feature **must** be enabled
- VUID-vkCmdSetFragmentShadingRateEnumNV-pipelineFragmentShadingRate-04580
  One of the pipelineFragmentShadingRate, primitiveFragmentShadingRate, or attachmentFragmentShadingRate features **must** be enabled
- VUID-vkCmdSetFragmentShadingRateEnumNV-primitiveFragmentShadingRate-04581
  If the primitiveFragmentShadingRate feature is not enabled, combinerOps[0] **must** be VK_FRAGMENT_SHADING_RATE_COMBINER_OP_KEEP_KHR
- VUID-vkCmdSetFragmentShadingRateEnumNV-attachmentFragmentShadingRate-04582
  If the attachmentFragmentShadingRate feature is not enabled, combinerOps[1] **must** be VK_FRAGMENT_SHADING_RATE_COMBINER_OP_KEEP_KHR
- VUID-vkCmdSetFragmentShadingRateEnumNV-fragmentSizeNonTrivialCombinerOps-04583
  If the fragmentSizeNonTrivialCombinerOps limit is not supported, elements of combinerOps **must** be either VK_FRAGMENT_SHADING_RATE_COMBINER_OP_KEEP_KHR or VK_FRAGMENT_SHADING_RATE_COMBINER_OP_REPLACE_KHR

## Valid Usage (Implicit)

- VUID-vkCmdSetFragmentShadingRateEnumNV-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdSetFragmentShadingRateEnumNV-shadingRate-parameter
  `shadingRate` **must** be a valid VkFragmentShadingRateNV value
- VUID-vkCmdSetFragmentShadingRateEnumNV-combinerOps-parameter
  Each element of `combinerOps` **must** be a valid VkFragmentShadingRateCombinerOpKHR value
- VUID-vkCmdSetFragmentShadingRateEnumNV-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdSetFragmentShadingRateEnumNV-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- VUID-vkCmdSetFragmentShadingRateEnumNV-videocoding
  This command **must** only be called outside of a video coding scope

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics | State |

When the `supersampleFragmentShadingRates` or `noInvocationFragmentShadingRates` features are enabled, the behavior of the shading rate combiner operations is extended to support the shading rates enabled by those features. Primitive and attachment shading rate values are interpreted as VkFragmentShadingRateNV values and the behavior of the combiners is modified as follows:

- For `VK_FRAGMENT_SHADING_RATE_COMBINER_OP_MIN_KHR`, `VK_FRAGMENT_SHADING_RATE_COMBINER_OP_MAX_KHR`, and `VK_FRAGMENT_SHADING_RATE_COMBINER_OP_MUL_KHR`, if either $A_{xy}$ or $B_{xy}$ is `VK_FRAGMENT_SHADING_RATE_NO_INVOCATIONS_NV`, combine($A_{xy}$,$B_{xy}$) produces a shading rate of `VK_FRAGMENT_SHADING_RATE_NO_INVOCATIONS_NV`, regardless of the other input shading rate.
- For `VK_FRAGMENT_SHADING_RATE_COMBINER_OP_MIN_KHR`, combine($A_{xy}$,$B_{xy}$) produces a shading rate

whose fragment size is the smaller of the fragment sizes of $A_{xy}$ and $B_{xy}$ and whose invocation count is the larger of the invocation counts of $A_{xy}$ and $B_{xy}$.

- For VK_FRAGMENT_SHADING_RATE_COMBINER_OP_MAX_KHR, combine($A_{xy}$,$B_{xy}$) produces a shading rate whose fragment size is the larger of the fragment sizes of $A_{xy}$ and $B_{xy}$ and whose invocation count is the smaller of the invocation counts of $A_{xy}$ and $B_{xy}$.
- For VK_FRAGMENT_SHADING_RATE_COMBINER_OP_MUL_KHR, combine($A_{xy}$,$B_{xy}$) produces a shading rate whose fragment size and invocation count is the product of the fragment sizes and invocation counts, respectively, of $A_{xy}$ and $B_{xy}$. If the resulting shading rate has both multiple pixels and multiple invocations per fragment, an implementation **may** adjust the shading rate by reducing both the pixel and invocation counts.

If the final shading rate from the combiners is VK_FRAGMENT_SHADING_RATE_NO_INVOCATIONS_NV, no fragments will be generated for any portion of a primitive using that shading rate.

If the final shading rate from the combiners specifies multiple fragment shader invocations per fragment, the fragment will be processed with multiple unique samples as in sample shading, where the total number the total number of invocations is taken from the shading rate and then clamped to rasterizationSamples and maxFragmentShadingRateInvocationCount.

## 28.7. Shading Rate Image

The shadingRateImage feature allows pipelines to use a shading rate image to control the fragment area and the minimum number of fragment shader invocations launched for each fragment. When the shading rate image is enabled, the rasterizer determines a base shading rate for each region of the framebuffer covered by a primitive by fetching a value from the shading rate image and translating it to a shading rate using a per-viewport shading rate palette. This base shading rate is then adjusted to derive a final shading rate. The final shading rate specifies the fragment area and fragment shader invocation count to use for fragments generated in the region.

If the pNext chain of VkPipelineViewportStateCreateInfo includes a VkPipelineViewportShadingRateImageStateCreateInfoNV structure, then that structure includes parameters controlling the shading rate.

The VkPipelineViewportShadingRateImageStateCreateInfoNV structure is defined as:

```
// Provided by VK_NV_shading_rate_image
typedef struct VkPipelineViewportShadingRateImageStateCreateInfoNV {
    VkStructureType                 sType;
    const void*                     pNext;
    VkBool32                        shadingRateImageEnable;
    uint32_t                        viewportCount;
    const VkShadingRatePaletteNV*   pShadingRatePalettes;
} VkPipelineViewportShadingRateImageStateCreateInfoNV;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.

- shadingRateImageEnable specifies whether shading rate image and palettes are used during rasterization.
- viewportCount specifies the number of per-viewport palettes used to translate values stored in shading rate images.
- pShadingRatePalettes is a pointer to an array of VkShadingRatePaletteNV structures defining the palette for each viewport. If the shading rate palette state is dynamic, this member is ignored.

If this structure is not present, shadingRateImageEnable is considered to be VK_FALSE, and the shading rate image and palettes are not used.

### Valid Usage

- VUID-VkPipelineViewportShadingRateImageStateCreateInfoNV-viewportCount-02054
  If the multiViewport feature is not enabled, viewportCount **must** be 0 or 1
- VUID-VkPipelineViewportShadingRateImageStateCreateInfoNV-viewportCount-02055
  viewportCount **must** be less than or equal to VkPhysicalDeviceLimits::maxViewports
- VUID-VkPipelineViewportShadingRateImageStateCreateInfoNV-shadingRateImageEnable-02056
  If shadingRateImageEnable is VK_TRUE, viewportCount **must** be greater or equal to the viewportCount member of VkPipelineViewportStateCreateInfo

### Valid Usage (Implicit)

- VUID-VkPipelineViewportShadingRateImageStateCreateInfoNV-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PIPELINE_VIEWPORT_SHADING_RATE_IMAGE_STATE_CREATE_INFO_NV

When shading rate image usage is enabled in the bound pipeline, the pipeline uses a shading rate image specified by the command:

```
// Provided by VK_NV_shading_rate_image
void vkCmdBindShadingRateImageNV(
    VkCommandBuffer                             commandBuffer,
    VkImageView                                 imageView,
    VkImageLayout                               imageLayout);
```

- commandBuffer is the command buffer into which the command will be recorded.
- imageView is an image view handle specifying the shading rate image. imageView **may** be set to VK_NULL_HANDLE, which is equivalent to specifying a view of an image filled with zero values.
- imageLayout is the layout that the image subresources accessible from imageView will be in when the shading rate image is accessed.

## Valid Usage

- VUID-vkCmdBindShadingRateImageNV-None-02058
  The `shadingRateImage` feature **must** be enabled
- VUID-vkCmdBindShadingRateImageNV-imageView-02059
  If `imageView` is not VK_NULL_HANDLE, it **must** be a valid VkImageView handle of type `VK_IMAGE_VIEW_TYPE_2D` or `VK_IMAGE_VIEW_TYPE_2D_ARRAY`
- VUID-vkCmdBindShadingRateImageNV-imageView-02060
  If `imageView` is not VK_NULL_HANDLE, it **must** have a format of `VK_FORMAT_R8_UINT`
- VUID-vkCmdBindShadingRateImageNV-imageView-02061
  If `imageView` is not VK_NULL_HANDLE, it **must** have been created with a `usage` value including `VK_IMAGE_USAGE_SHADING_RATE_IMAGE_BIT_NV`
- VUID-vkCmdBindShadingRateImageNV-imageView-02062
  If `imageView` is not VK_NULL_HANDLE, `imageLayout` **must** match the actual VkImageLayout of each subresource accessible from `imageView` at the time the subresource is accessed
- VUID-vkCmdBindShadingRateImageNV-imageLayout-02063
  If `imageView` is not VK_NULL_HANDLE, `imageLayout` **must** be `VK_IMAGE_LAYOUT_SHADING_RATE_OPTIMAL_NV` or `VK_IMAGE_LAYOUT_GENERAL`

## Valid Usage (Implicit)

- VUID-vkCmdBindShadingRateImageNV-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdBindShadingRateImageNV-imageView-parameter
  If `imageView` is not VK_NULL_HANDLE, `imageView` **must** be a valid VkImageView handle
- VUID-vkCmdBindShadingRateImageNV-imageLayout-parameter
  `imageLayout` **must** be a valid VkImageLayout value
- VUID-vkCmdBindShadingRateImageNV-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdBindShadingRateImageNV-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- VUID-vkCmdBindShadingRateImageNV-videocoding
  This command **must** only be called outside of a video coding scope
- VUID-vkCmdBindShadingRateImageNV-commonparent
  Both of `commandBuffer`, and `imageView` that are valid handles of non-ignored parameters **must** have been created, allocated, or retrieved from the same VkDevice

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized

- Host access to the VkCommandPool that commandBuffer was allocated from **must** be externally synchronized

### Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics | State |

When the shading rate image is enabled in the current pipeline, rasterizing a primitive covering the pixel with coordinates (*x*,*y*) will fetch a shading rate index value from the shading rate image bound by vkCmdBindShadingRateImageNV. If the shading rate image view has a type of VK_IMAGE_VIEW_TYPE_2D, the lookup will use texel coordinates (*u*,*v*) where $u = \lfloor \frac{x}{twidth} \rfloor$, $v = \lfloor \frac{y}{theight} \rfloor$, and $twidth$ and $theight$ are the width and height of the implementation-dependent shading rate texel size. If the shading rate image view has a type of VK_IMAGE_VIEW_TYPE_2D_ARRAY, the lookup will use texel coordinates (*u*,*v*) to extract a texel from the layer *l*, where *l* is the layer of the framebuffer being rendered to. If *l* is greater than or equal to the number of layers in the image view, layer zero will be used.

If the bound shading rate image view is not VK_NULL_HANDLE and contains a texel with coordinates (*u*,*v*) in layer *l* (if applicable), the single unsigned integer component for that texel will be used as the shading rate index. If the (*u*,*v*) coordinate is outside the extents of the subresource used by the shading rate image view, or if the image view is VK_NULL_HANDLE, the shading rate index is zero. If the shading rate image view has multiple mipmap levels, the base level identified by VkImageSubresourceRange::baseMipLevel will be used.

A shading rate index is mapped to a base shading rate using a lookup table called the shading rate image palette. There is a separate palette for each viewport. The number of entries in each palette is given by the implementation-dependent shading rate image palette size.

To dynamically set the shadingRateImageEnable state, call:

```
// Provided by VK_EXT_extended_dynamic_state3 with VK_NV_shading_rate_image,
VK_EXT_shader_object with VK_NV_shading_rate_image
void vkCmdSetShadingRateImageEnableNV(
    VkCommandBuffer                             commandBuffer,
    VkBool32                                    shadingRateImageEnable);
```

- commandBuffer is the command buffer into which the command will be recorded.
- shadingRateImageEnable specifies the shadingRateImageEnable state.

This command sets the shadingRateImageEnable state for subsequent drawing commands when drawing using shader objects, or when the graphics pipeline is created with VK_DYNAMIC_STATE_SHADING_RATE_IMAGE_ENABLE_NV set in VkPipelineDynamicStateCreateInfo

::pDynamicStates. Otherwise, this state is specified by the VkPipelineViewportShadingRateImageStateCreateInfoNV::shadingRateImageEnable value used to create the currently active pipeline.

## Valid Usage

- VUID-vkCmdSetShadingRateImageEnableNV-None-09423
  At least one of the following **must** be true:
  - The extendedDynamicState3ShadingRateImageEnable feature is enabled
  - The shaderObject feature is enabled

## Valid Usage (Implicit)

- VUID-vkCmdSetShadingRateImageEnableNV-commandBuffer-parameter
  commandBuffer **must** be a valid VkCommandBuffer handle
- VUID-vkCmdSetShadingRateImageEnableNV-commandBuffer-recording
  commandBuffer **must** be in the recording state
- VUID-vkCmdSetShadingRateImageEnableNV-commandBuffer-cmdpool
  The VkCommandPool that commandBuffer was allocated from **must** support graphics operations
- VUID-vkCmdSetShadingRateImageEnableNV-videocoding
  This command **must** only be called outside of a video coding scope

## Host Synchronization

- Host access to commandBuffer **must** be externally synchronized
- Host access to the VkCommandPool that commandBuffer was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics | State |

To dynamically set the per-viewport shading rate image palettes, call:

```
// Provided by VK_NV_shading_rate_image
void vkCmdSetViewportShadingRatePaletteNV(
    VkCommandBuffer                             commandBuffer,
    uint32_t                                    firstViewport,
    uint32_t                                    viewportCount,
    const VkShadingRatePaletteNV*               pShadingRatePalettes);
```

- commandBuffer is the command buffer into which the command will be recorded.
- firstViewport is the index of the first viewport whose shading rate palette is updated by the command.
- viewportCount is the number of viewports whose shading rate palettes are updated by the command.
- pShadingRatePalettes is a pointer to an array of VkShadingRatePaletteNV structures defining the palette for each viewport.

This command sets the per-viewport shading rate image palettes for subsequent drawing commands when drawing using shader objects, or when the graphics pipeline is created with VK_DYNAMIC_STATE_VIEWPORT_SHADING_RATE_PALETTE_NV set in VkPipelineDynamicStateCreateInfo::pDynamicStates. Otherwise, this state is specified by the VkPipelineViewportShadingRateImageStateCreateInfoNV::pShadingRatePalettes values used to create the currently active pipeline.

### Valid Usage

- VUID-vkCmdSetViewportShadingRatePaletteNV-None-02064
  The shadingRateImage feature **must** be enabled
- VUID-vkCmdSetViewportShadingRatePaletteNV-firstViewport-02067
  The sum of firstViewport and viewportCount **must** be between 1 and VkPhysicalDeviceLimits::maxViewports, inclusive
- VUID-vkCmdSetViewportShadingRatePaletteNV-firstViewport-02068
  If the multiViewport feature is not enabled, firstViewport **must** be 0
- VUID-vkCmdSetViewportShadingRatePaletteNV-viewportCount-02069
  If the multiViewport feature is not enabled, viewportCount **must** be 1

### Valid Usage (Implicit)

- VUID-vkCmdSetViewportShadingRatePaletteNV-commandBuffer-parameter
  commandBuffer **must** be a valid VkCommandBuffer handle
- VUID-vkCmdSetViewportShadingRatePaletteNV-pShadingRatePalettes-parameter
  pShadingRatePalettes **must** be a valid pointer to an array of viewportCount valid VkShadingRatePaletteNV structures
- VUID-vkCmdSetViewportShadingRatePaletteNV-commandBuffer-recording
  commandBuffer **must** be in the recording state

- VUID-vkCmdSetViewportShadingRatePaletteNV-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- VUID-vkCmdSetViewportShadingRatePaletteNV-videocoding
  This command **must** only be called outside of a video coding scope
- VUID-vkCmdSetViewportShadingRatePaletteNV-viewportCount-arraylength
  `viewportCount` **must** be greater than `0`

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics | State |

The `VkShadingRatePaletteNV` structure specifies to contents of a single shading rate image palette and is defined as:

```
// Provided by VK_NV_shading_rate_image
typedef struct VkShadingRatePaletteNV {
    uint32_t                                    shadingRatePaletteEntryCount;
    const VkShadingRatePaletteEntryNV*          pShadingRatePaletteEntries;
} VkShadingRatePaletteNV;
```

- `shadingRatePaletteEntryCount` specifies the number of entries in the shading rate image palette.
- `pShadingRatePaletteEntries` is a pointer to an array of VkShadingRatePaletteEntryNV enums defining the shading rate for each palette entry.

## Valid Usage

- VUID-VkShadingRatePaletteNV-shadingRatePaletteEntryCount-02071
  `shadingRatePaletteEntryCount` **must** be between `1` and `VkPhysicalDeviceShadingRateImagePropertiesNV::shadingRatePaletteSize`, inclusive

### Valid Usage (Implicit)

- VUID-VkShadingRatePaletteNV-pShadingRatePaletteEntries-parameter `pShadingRatePaletteEntries` **must** be a valid pointer to an array of `shadingRatePaletteEntryCount` valid VkShadingRatePaletteEntryNV values
- VUID-VkShadingRatePaletteNV-shadingRatePaletteEntryCount-arraylength `shadingRatePaletteEntryCount` **must** be greater than `0`

To determine the base shading rate image, a shading rate index *i* is mapped to array element *i* in the array `pShadingRatePaletteEntries` for the palette corresponding to the viewport used for the fragment. If *i* is greater than or equal to the palette size `shadingRatePaletteEntryCount`, the base shading rate is undefined.

The supported shading rate image palette entries are defined by VkShadingRatePaletteEntryNV:

```
// Provided by VK_NV_shading_rate_image
typedef enum VkShadingRatePaletteEntryNV {
    VK_SHADING_RATE_PALETTE_ENTRY_NO_INVOCATIONS_NV = 0,
    VK_SHADING_RATE_PALETTE_ENTRY_16_INVOCATIONS_PER_PIXEL_NV = 1,
    VK_SHADING_RATE_PALETTE_ENTRY_8_INVOCATIONS_PER_PIXEL_NV = 2,
    VK_SHADING_RATE_PALETTE_ENTRY_4_INVOCATIONS_PER_PIXEL_NV = 3,
    VK_SHADING_RATE_PALETTE_ENTRY_2_INVOCATIONS_PER_PIXEL_NV = 4,
    VK_SHADING_RATE_PALETTE_ENTRY_1_INVOCATION_PER_PIXEL_NV = 5,
    VK_SHADING_RATE_PALETTE_ENTRY_1_INVOCATION_PER_2X1_PIXELS_NV = 6,
    VK_SHADING_RATE_PALETTE_ENTRY_1_INVOCATION_PER_1X2_PIXELS_NV = 7,
    VK_SHADING_RATE_PALETTE_ENTRY_1_INVOCATION_PER_2X2_PIXELS_NV = 8,
    VK_SHADING_RATE_PALETTE_ENTRY_1_INVOCATION_PER_4X2_PIXELS_NV = 9,
    VK_SHADING_RATE_PALETTE_ENTRY_1_INVOCATION_PER_2X4_PIXELS_NV = 10,
    VK_SHADING_RATE_PALETTE_ENTRY_1_INVOCATION_PER_4X4_PIXELS_NV = 11,
} VkShadingRatePaletteEntryNV;
```

The following table indicates the width and height (in pixels) of each fragment generated using the indicated shading rate, as well as the maximum number of fragment shader invocations launched for each fragment. When processing regions of a primitive that have a shading rate of `VK_SHADING_RATE_PALETTE_ENTRY_NO_INVOCATIONS_NV`, no fragments will be generated in that region.

| Shading Rate | Width | Height | Invocations |
|---|---|---|---|
| `VK_SHADING_RATE_PALETTE_ENTRY_NO_INVOCATIONS_NV` | 0 | 0 | 0 |
| `VK_SHADING_RATE_PALETTE_ENTRY_16_INVOCATIONS_PER_PIXEL_NV` | 1 | 1 | 16 |
| `VK_SHADING_RATE_PALETTE_ENTRY_8_INVOCATIONS_PER_PIXEL_NV` | 1 | 1 | 8 |

| Shading Rate | Width | Height | Invocations |
|---|---|---|---|
| VK_SHADING_RATE_PALETTE_ENTRY_4_INVOCATIONS_PER_PIXEL_NV | 1 | 1 | 4 |
| VK_SHADING_RATE_PALETTE_ENTRY_2_INVOCATIONS_PER_PIXEL_NV | 1 | 1 | 2 |
| VK_SHADING_RATE_PALETTE_ENTRY_1_INVOCATION_PER_PIXEL_NV | 1 | 1 | 1 |
| VK_SHADING_RATE_PALETTE_ENTRY_1_INVOCATION_PER_2X1_PIXELS_NV | 2 | 1 | 1 |
| VK_SHADING_RATE_PALETTE_ENTRY_1_INVOCATION_PER_1X2_PIXELS_NV | 1 | 2 | 1 |
| VK_SHADING_RATE_PALETTE_ENTRY_1_INVOCATION_PER_2X2_PIXELS_NV | 2 | 2 | 1 |
| VK_SHADING_RATE_PALETTE_ENTRY_1_INVOCATION_PER_4X2_PIXELS_NV | 4 | 2 | 1 |
| VK_SHADING_RATE_PALETTE_ENTRY_1_INVOCATION_PER_2X4_PIXELS_NV | 2 | 4 | 1 |
| VK_SHADING_RATE_PALETTE_ENTRY_1_INVOCATION_PER_4X4_PIXELS_NV | 4 | 4 | 1 |

When the shading rate image is disabled, a shading rate of VK_SHADING_RATE_PALETTE_ENTRY_1_INVOCATION_PER_PIXEL_NV will be used as the base shading rate.

Once a base shading rate has been established, it is adjusted to produce a final shading rate. First, if the base shading rate uses multiple pixels for each fragment, the implementation **may** reduce the fragment area to ensure that the total number of coverage samples for all pixels in a fragment does not exceed an implementation-dependent maximum.

If sample shading is active in the current pipeline and would result in processing $n$ ($n > 1$) unique samples per fragment when the shading rate image is disabled, the shading rate is adjusted in an implementation-dependent manner to increase the number of fragment shader invocations spawned by the primitive. If the shading rate indicates $fs$ pixels per fragment and $fs$ is greater than $n$, the fragment area is adjusted so each fragment has approximately $f\frac{s}{n}$ pixels. Otherwise, if the shading rate indicates $ipf$ invocations per fragment, the fragment area will be adjusted to a single pixel with approximately $ipf \times \frac{n}{f}s$ invocations per fragment.

If sample shading occurs due to the use of a fragment shader input variable decorated with SampleId or SamplePosition, the shading rate is ignored. Each fragment will have a single pixel and will spawn up to rasterizationSamples fragment shader invocations, as when using sample shading without a shading rate image.

Finally, if the shading rate specifies multiple fragment shader invocations per fragment, the total number of invocations in the shading rate is clamped to be no larger than `rasterizationSamples`.

When the final shading rate for a primitive covering pixel (*x*,*y*) has a fragment area of $fw \times fh$, the fragment for that pixel will cover all pixels with coordinates (*x*',*y*') that satisfy the equations:

$$\lfloor \frac{x}{fw} \rfloor = \lfloor \frac{x'}{fw} \rfloor$$

$$\lfloor \frac{y}{fh} \rfloor = \lfloor \frac{y'}{fh} \rfloor$$

This combined fragment is considered to have multiple coverage samples; the total number of samples in this fragment is given by $samples = fw \times fh \times rs$ where $rs$ indicates the value of `VkPipelineMultisampleStateCreateInfo::rasterizationSamples` specified at pipeline creation time. The set of coverage samples in the fragment is the union of the per-pixel coverage samples in each of the fragment's pixels The location and order of coverage samples within each pixel in the combined fragment are assigned as described in Multisampling and Custom Sample Locations. Each coverage sample in the set of pixels belonging to the combined fragment is assigned a unique coverage index in the range [0,*samples*-1]. If the `shadingRateCoarseSampleOrder` feature is supported, the order of coverage samples **can** be specified for each combination of fragment area and coverage sample count. If this feature is not supported, the sample order is implementation-dependent.

If the `pNext` chain of VkPipelineViewportStateCreateInfo includes a `VkPipelineViewportCoarseSampleOrderStateCreateInfoNV` structure, then that structure includes parameters controlling the order of coverage samples in fragments larger than one pixel.

The `VkPipelineViewportCoarseSampleOrderStateCreateInfoNV` structure is defined as:

```
// Provided by VK_NV_shading_rate_image
typedef struct VkPipelineViewportCoarseSampleOrderStateCreateInfoNV {
    VkStructureType                         sType;
    const void*                             pNext;
    VkCoarseSampleOrderTypeNV               sampleOrderType;
    uint32_t                                customSampleOrderCount;
    const VkCoarseSampleOrderCustomNV*      pCustomSampleOrders;
} VkPipelineViewportCoarseSampleOrderStateCreateInfoNV;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `sampleOrderType` specifies the mechanism used to order coverage samples in fragments larger than one pixel.
- `customSampleOrderCount` specifies the number of custom sample orderings to use when ordering coverage samples.
- `pCustomSampleOrders` is a pointer to an array of `customSampleOrderCount` VkCoarseSampleOrderCustomNV structures, each structure specifying the coverage sample

order for a single combination of fragment area and coverage sample count.

If this structure is not present, `sampleOrderType` is considered to be `VK_COARSE_SAMPLE_ORDER_TYPE_DEFAULT_NV`.

If `sampleOrderType` is `VK_COARSE_SAMPLE_ORDER_TYPE_CUSTOM_NV`, the coverage sample order used for any combination of fragment area and coverage sample count not enumerated in `pCustomSampleOrders` will be identical to that used for `VK_COARSE_SAMPLE_ORDER_TYPE_DEFAULT_NV`.

If the pipeline was created with `VK_DYNAMIC_STATE_VIEWPORT_COARSE_SAMPLE_ORDER_NV`, the contents of this structure (if present) are ignored, and the coverage sample order is instead specified by vkCmdSetCoarseSampleOrderNV.

### Valid Usage

- VUID-VkPipelineViewportCoarseSampleOrderStateCreateInfoNV-sampleOrderType-02072
  If `sampleOrderType` is not `VK_COARSE_SAMPLE_ORDER_TYPE_CUSTOM_NV`, `customSamplerOrderCount` **must** be `0`
- VUID-VkPipelineViewportCoarseSampleOrderStateCreateInfoNV-pCustomSampleOrders-02234
  The array `pCustomSampleOrders` **must** not contain two structures with matching values for both the `shadingRate` and `sampleCount` members

### Valid Usage (Implicit)

- VUID-VkPipelineViewportCoarseSampleOrderStateCreateInfoNV-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PIPELINE_VIEWPORT_COARSE_SAMPLE_ORDER_STATE_CREATE_INFO_NV`
- VUID-VkPipelineViewportCoarseSampleOrderStateCreateInfoNV-sampleOrderType-parameter
  `sampleOrderType` **must** be a valid VkCoarseSampleOrderTypeNV value
- VUID-VkPipelineViewportCoarseSampleOrderStateCreateInfoNV-pCustomSampleOrders-parameter
  If `customSampleOrderCount` is not `0`, `pCustomSampleOrders` **must** be a valid pointer to an array of `customSampleOrderCount` valid VkCoarseSampleOrderCustomNV structures

The type VkCoarseSampleOrderTypeNV specifies the technique used to order coverage samples in fragments larger than one pixel, and is defined as:

```
// Provided by VK_NV_shading_rate_image
typedef enum VkCoarseSampleOrderTypeNV {
    VK_COARSE_SAMPLE_ORDER_TYPE_DEFAULT_NV = 0,
    VK_COARSE_SAMPLE_ORDER_TYPE_CUSTOM_NV = 1,
    VK_COARSE_SAMPLE_ORDER_TYPE_PIXEL_MAJOR_NV = 2,
    VK_COARSE_SAMPLE_ORDER_TYPE_SAMPLE_MAJOR_NV = 3,
} VkCoarseSampleOrderTypeNV;
```

- `VK_COARSE_SAMPLE_ORDER_TYPE_DEFAULT_NV` specifies that coverage samples will be ordered in an implementation-dependent manner.
- `VK_COARSE_SAMPLE_ORDER_TYPE_CUSTOM_NV` specifies that coverage samples will be ordered according to the array of custom orderings provided in either the `pCustomSampleOrders` member of `VkPipelineViewportCoarseSampleOrderStateCreateInfoNV` or the `pCustomSampleOrders` member of vkCmdSetCoarseSampleOrderNV.
- `VK_COARSE_SAMPLE_ORDER_TYPE_PIXEL_MAJOR_NV` specifies that coverage samples will be ordered sequentially, sorted first by pixel coordinate (in row-major order) and then by sample index.
- `VK_COARSE_SAMPLE_ORDER_TYPE_SAMPLE_MAJOR_NV` specifies that coverage samples will be ordered sequentially, sorted first by sample index and then by pixel coordinate (in row-major order).

When using a coarse sample order of `VK_COARSE_SAMPLE_ORDER_TYPE_PIXEL_MAJOR_NV` for a fragment with an upper-left corner of $(fx, fy)$ with a width of $fw \times fh$ and $fsc$ samples per pixel, coverage index $cs$ of the fragment will be assigned to sample index $fs$ of pixel $(px, py)$ as follows:

$$
\begin{aligned}
px &= fx + (\lfloor c\frac{s}{f}sc \rfloor \% fw) \\
py &= fy + \lfloor c\frac{s}{fsc \times fw} \rfloor \\
fs &= cs \% fsc
\end{aligned}
$$

When using a coarse sample order of `VK_COARSE_SAMPLE_ORDER_TYPE_SAMPLE_MAJOR_NV`, coverage index $cs$ will be assigned as follows:

$$
\begin{aligned}
px &= fx + cs \% fw \\
py &= (fy + \lfloor c\frac{s}{f}w \rfloor \% fh) \\
fs &= \lfloor c\frac{s}{fw \times fh} \rfloor
\end{aligned}
$$

The `VkCoarseSampleOrderCustomNV` structure is defined as:

```
// Provided by VK_NV_shading_rate_image
typedef struct VkCoarseSampleOrderCustomNV {
    VkShadingRatePaletteEntryNV        shadingRate;
    uint32_t                           sampleCount;
    uint32_t                           sampleLocationCount;
    const VkCoarseSampleLocationNV*    pSampleLocations;
} VkCoarseSampleOrderCustomNV;
```

- `shadingRate` is a shading rate palette entry that identifies the fragment width and height for the combination of fragment area and per-pixel coverage sample count to control.
- `sampleCount` identifies the per-pixel coverage sample count for the combination of fragment area and coverage sample count to control.
- `sampleLocationCount` specifies the number of sample locations in the custom ordering.
- `pSampleLocations` is a pointer to an array of VkCoarseSampleLocationNV structures specifying the location of each sample in the custom ordering.

The `VkCoarseSampleOrderCustomNV` structure is used with a coverage sample ordering type of `VK_COARSE_SAMPLE_ORDER_TYPE_CUSTOM_NV` to specify the order of coverage samples for one combination of fragment width, fragment height, and coverage sample count.

When using a custom sample ordering, element *j* in `pSampleLocations` specifies a specific pixel location and sample index that corresponds to coverage index *j* in the multi-pixel fragment.

### Valid Usage

- VUID-VkCoarseSampleOrderCustomNV-shadingRate-02073
  `shadingRate` **must** be a shading rate that generates fragments with more than one pixel
- VUID-VkCoarseSampleOrderCustomNV-sampleCount-02074
  `sampleCount` **must** correspond to a sample count enumerated in VkSampleCountFlags whose corresponding bit is set in VkPhysicalDeviceLimits::`framebufferNoAttachmentsSampleCounts`
- VUID-VkCoarseSampleOrderCustomNV-sampleLocationCount-02075
  `sampleLocationCount` **must** be equal to the product of `sampleCount`, the fragment width for `shadingRate`, and the fragment height for `shadingRate`
- VUID-VkCoarseSampleOrderCustomNV-sampleLocationCount-02076
  `sampleLocationCount` **must** be less than or equal to the value of `VkPhysicalDeviceShadingRateImagePropertiesNV`::`shadingRateMaxCoarseSamples`
- VUID-VkCoarseSampleOrderCustomNV-pSampleLocations-02077
  The array `pSampleLocations` **must** contain exactly one entry for every combination of valid values for `pixelX`, `pixelY`, and `sample` in the structure VkCoarseSampleOrderCustomNV

### Valid Usage (Implicit)

- VUID-VkCoarseSampleOrderCustomNV-shadingRate-parameter
  `shadingRate` **must** be a valid VkShadingRatePaletteEntryNV value
- VUID-VkCoarseSampleOrderCustomNV-pSampleLocations-parameter
  `pSampleLocations` **must** be a valid pointer to an array of `sampleLocationCount` VkCoarseSampleLocationNV structures
- VUID-VkCoarseSampleOrderCustomNV-sampleLocationCount-arraylength
  `sampleLocationCount` **must** be greater than `0`

The `VkCoarseSampleLocationNV` structure identifies a specific pixel and sample index for one of the coverage samples in a fragment that is larger than one pixel. This structure is defined as:

```
// Provided by VK_NV_shading_rate_image
typedef struct VkCoarseSampleLocationNV {
    uint32_t    pixelX;
    uint32_t    pixelY;
    uint32_t    sample;
} VkCoarseSampleLocationNV;
```

- `pixelX` is added to the x coordinate of the upper-leftmost pixel of each fragment to identify the pixel containing the coverage sample.
- `pixelY` is added to the y coordinate of the upper-leftmost pixel of each fragment to identify the pixel containing the coverage sample.
- `sample` is the number of the coverage sample in the pixel identified by `pixelX` and `pixelY`.

### Valid Usage

- VUID-VkCoarseSampleLocationNV-pixelX-02078
  `pixelX` **must** be less than the width (in pixels) of the fragment
- VUID-VkCoarseSampleLocationNV-pixelY-02079
  `pixelY` **must** be less than the height (in pixels) of the fragment
- VUID-VkCoarseSampleLocationNV-sample-02080
  `sample` **must** be less than the number of coverage samples in each pixel belonging to the fragment

To dynamically set the order of coverage samples in fragments larger than one pixel, call:

```
// Provided by VK_NV_shading_rate_image
void vkCmdSetCoarseSampleOrderNV(
    VkCommandBuffer                             commandBuffer,
    VkCoarseSampleOrderTypeNV                   sampleOrderType,
    uint32_t                                    customSampleOrderCount,
    const VkCoarseSampleOrderCustomNV*          pCustomSampleOrders);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `sampleOrderType` specifies the mechanism used to order coverage samples in fragments larger than one pixel.
- `customSampleOrderCount` specifies the number of custom sample orderings to use when ordering coverage samples.
- `pCustomSampleOrders` is a pointer to an array of VkCoarseSampleOrderCustomNV structures, each structure specifying the coverage sample order for a single combination of fragment area and coverage sample count.

If `sampleOrderType` is `VK_COARSE_SAMPLE_ORDER_TYPE_CUSTOM_NV`, the coverage sample order used for any combination of fragment area and coverage sample count not enumerated in `pCustomSampleOrders` will be identical to that used for `VK_COARSE_SAMPLE_ORDER_TYPE_DEFAULT_NV`.

This command sets the order of coverage samples for subsequent drawing commands when drawing using shader objects, or when the graphics pipeline is created with `VK_DYNAMIC_STATE_VIEWPORT_COARSE_SAMPLE_ORDER_NV` set in VkPipelineDynamicStateCreateInfo::`pDynamicStates`. Otherwise, this state is specified by the VkPipelineViewportCoarseSampleOrderStateCreateInfoNV values used to create the currently active pipeline.

## Valid Usage

- VUID-vkCmdSetCoarseSampleOrderNV-sampleOrderType-02081
  If `sampleOrderType` is not `VK_COARSE_SAMPLE_ORDER_TYPE_CUSTOM_NV`, `customSamplerOrderCount` **must** be `0`
- VUID-vkCmdSetCoarseSampleOrderNV-pCustomSampleOrders-02235
  The array `pCustomSampleOrders` **must** not contain two structures with matching values for both the `shadingRate` and `sampleCount` members

## Valid Usage (Implicit)

- VUID-vkCmdSetCoarseSampleOrderNV-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdSetCoarseSampleOrderNV-sampleOrderType-parameter
  `sampleOrderType` **must** be a valid VkCoarseSampleOrderTypeNV value
- VUID-vkCmdSetCoarseSampleOrderNV-pCustomSampleOrders-parameter
  If `customSampleOrderCount` is not `0`, `pCustomSampleOrders` **must** be a valid pointer to an array

of customSampleOrderCount valid VkCoarseSampleOrderCustomNV structures

- VUID-vkCmdSetCoarseSampleOrderNV-commandBuffer-recording
  commandBuffer **must** be in the recording state
- VUID-vkCmdSetCoarseSampleOrderNV-commandBuffer-cmdpool
  The VkCommandPool that commandBuffer was allocated from **must** support graphics operations
- VUID-vkCmdSetCoarseSampleOrderNV-videocoding
  This command **must** only be called outside of a video coding scope

### Host Synchronization

- Host access to commandBuffer **must** be externally synchronized
- Host access to the VkCommandPool that commandBuffer was allocated from **must** be externally synchronized

### Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics | State |

If the final shading rate for a primitive covering pixel (*x*,*y*) results in *n* invocations per pixel ($n > 1$), *n* separate fragment shader invocations will be generated for the fragment. Each coverage sample in the fragment will be assigned to one of the *n* fragment shader invocations in an implementation-dependent manner. The outputs from the fragment output interface of each shader invocation will be broadcast to all of the framebuffer samples associated with the invocation. If none of the coverage samples associated with a fragment shader invocation is covered by a primitive, the implementation **may** discard the fragment shader invocation for those samples.

If the final shading rate for a primitive covering pixel (*x*,*y*) results in a fragment containing multiple pixels, a single set of fragment shader invocations will be generated for all pixels in the combined fragment. Outputs from the fragment output interface will be broadcast to all covered framebuffer samples belonging to the fragment. If the fragment shader executes code discarding the fragment, none of the samples of the fragment will be updated.

## 28.8. Sample Shading

Sample shading **can** be used to specify a minimum number of unique samples to process for each fragment. If sample shading is enabled, an implementation **must** invoke the fragment shader at least max(□ VkPipelineMultisampleStateCreateInfo::minSampleShading × VkPipelineMultisampleStateCreateInfo::rasterizationSamples □, 1) times per fragment. If VkPipelineMultisampleStateCreateInfo::sampleShadingEnable is set to VK_TRUE, sample shading is

enabled.

If a fragment shader entry point statically uses an input variable decorated with a `BuiltIn` of `SampleId` or `SamplePosition`, sample shading is enabled and a value of `1.0` is used instead of `minSampleShading`. If a fragment shader entry point statically uses an input variable decorated with `Sample`, sample shading **may** be enabled and a value of `1.0` will be used instead of `minSampleShading` if it is. If the `VK_AMD_mixed_attachment_samples` extension is enabled and the subpass uses color attachments, the `samples` value used to create each color attachment is used instead of `rasterizationSamples`.

**NOTE** If a shader decorates an input variable with `Sample` and that value meaningfully impacts the output of a shader, sample shading will be enabled to ensure that the input is in fact interpolated per-sample. This is inherent to the specification and not spelled out here - if an application simply declares such a variable it is implementation-defined whether sample shading is enabled or not. It is possible to see the effects of this by using atomics in the shader or using a pipeline statistics query to query the number of fragment invocations, even if the shader itself does not use any per-sample variables.

If there are fewer fragment invocations than covered samples, implementations **may** include those samples in fragment shader invocations in any manner as long as covered samples are all shaded at least once, and each invocation that is not a helper invocation covers at least one sample.

## 28.9. Barycentric Interpolation

When the `fragmentShaderBarycentric` feature is enabled, the `PerVertexKHR` interpolation decoration **can** be used with fragment shader inputs to indicate that the decorated inputs do not have associated data in the fragment. Such inputs **can** only be accessed in a fragment shader using an array index whose value (0, 1, or 2) identifies one of the vertices of the primitive that produced the fragment. Reads of per-vertex values for missing vertices, such as the third vertex of a line primitive, will return values from the valid vertex with the highest index. This means that the per-vertex values of indices 1 and 2 for point primitives will be equal to those of index 0, and the per-vertex values of index 2 for line primitives will be equal to those of index 1.

When tessellation, geometry shading, and mesh shading are not active, fragment shader inputs decorated with `PerVertexKHR` will take values from one of the vertices of the primitive that produced the fragment, identified by the extra index provided in SPIR-V code accessing the input. If the *n* vertices passed to a draw call are numbered 0 through *n*-1, and the point, line, and triangle primitives produced by the draw call are numbered with consecutive integers beginning with zero, the following table indicates the original vertex numbers used when the provoking vertex mode is `VK_PROVOKING_VERTEX_MODE_FIRST_VERTEX_EXT` for index values of 0, 1, and 2. If an input decorated with `PerVertexKHR` is accessed with any other vertex index value, or is accessed while rasterizing a polygon when the VkPipelineRasterizationStateCreateInfo::`polygonMode` property of the currently active pipeline is not `VK_POLYGON_MODE_FILL`, an undefined value is returned.

| Primitive Topology | Vertex 0 | Vertex 1 | Vertex 2 |
|---|---|---|---|
| `VK_PRIMITIVE_TOPOLOGY_POINT_LIST` | i | i | i |

| Primitive Topology | Vertex 0 | Vertex 1 | Vertex 2 |
|---|---|---|---|
| VK_PRIMITIVE_TOPOLOGY_LINE_LIST | 2i | 2i+1 | 2i+1 |
| VK_PRIMITIVE_TOPOLOGY_LINE_STRIP | i | i+1 | i+1 |
| VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST | 3i | 3i+1 | 3i+2 |
| VK_PRIMITIVE_TOPOLOGY_TRIANGLE_STRIP (even) | i | i+1 | i+2 |
| VK_PRIMITIVE_TOPOLOGY_TRIANGLE_STRIP (odd) | i | i+2 | i+1 |
| VK_PRIMITIVE_TOPOLOGY_TRIANGLE_FAN | i+1 | i+2 | 0 |
| VK_PRIMITIVE_TOPOLOGY_LINE_LIST_WITH_ADJACENCY | 4i+1 | 4i+2 | 4i+2 |
| VK_PRIMITIVE_TOPOLOGY_LINE_STRIP_WITH_ADJACENCY | i+1 | i+2 | i+2 |
| VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST_WITH_ADJACENCY | 6i | 6i+2 | 6i+4 |
| VK_PRIMITIVE_TOPOLOGY_TRIANGLE_STRIP_WITH_ADJACENCY (even) | 2i | 2i+2 | 2i+4 |
| VK_PRIMITIVE_TOPOLOGY_TRIANGLE_STRIP_WITH_ADJACENCY (odd) | 2i | 2i+4 | 2i+2 |

When the provoking vertex mode is VK_PROVOKING_VERTEX_MODE_LAST_VERTEX_EXT, the original vertex numbers used are the same as above except as indicated in the table below.

| Primitive Topology | Vertex 0 | Vertex 1 | Vertex 2 |
|---|---|---|---|
| VK_PRIMITIVE_TOPOLOGY_TRIANGLE_STRIP (odd, and triStripVertexOrderIndependentOfProvokingVertex of VkPhysicalDeviceFragmentShaderBarycentricPropertiesKHR is VK_FALSE) | i+1 | i | i+2 |
| VK_PRIMITIVE_TOPOLOGY_TRIANGLE_FAN | 0 | i+1 | i+2 |

| Primitive Topology | Vertex 0 | Vertex 1 | Vertex 2 |
|---|---|---|---|
| VK_PRIMITIVE_TOPOLOGY_TRIANGLE_STRIP_WITH_ADJACENCY (odd) | 2i+2 | 2i | 2i+4 |

When geometry or mesh shading is active, primitives processed by fragment shaders are assembled from the vertices emitted by the geometry or mesh shader. In this case, the vertices used for fragment shader inputs decorated with PerVertexKHR are derived by treating the primitives produced by the shader as though they were specified by a draw call and consulting the table above.

When using tessellation without geometry shading, the tessellator produces primitives in an implementation-dependent manner. While there is no defined vertex ordering for inputs decorated with PerVertexKHR, the vertex ordering used in this case will be consistent with the ordering used to derive the values of inputs decorated with BaryCoordKHR or BaryCoordNoPerspKHR.

Fragment shader inputs decorated with BaryCoordKHR or BaryCoordNoPerspKHR hold three-component vectors with barycentric weights that indicate the location of the fragment relative to the screen-space locations of vertices of its primitive. For point primitives, such variables are always assigned the value (1,0,0). For line primitives, the built-ins are obtained by interpolating an attribute whose values for the vertices numbered 0 and 1 are (1,0,0) and (0,1,0), respectively. For polygon primitives, the built-ins are obtained by interpolating an attribute whose values for the vertices numbered 0, 1, and 2 are (1,0,0), (0,1,0), and (0,0,1), respectively. For BaryCoordKHR, the values are obtained using perspective interpolation. For BaryCoordNoPerspKHR, the values are obtained using linear interpolation. The values of BaryCoordKHR and BaryCoordNoPerspKHR are undefined while rasterizing a polygon when the VkPipelineRasterizationStateCreateInfo::polygonMode property of the currently active pipeline is not VK_POLYGON_MODE_FILL.

## 28.10. Points

A point is drawn by generating a set of fragments in the shape of a square centered around the vertex of the point. Each vertex has an associated point size controlling the width/height of that square. The point size is taken from the (potentially clipped) shader built-in PointSize written by:

- the geometry shader, if active;
- the tessellation evaluation shader, if active and no geometry shader is active;
- the vertex shader, otherwise

and clamped to the implementation-dependent point size range [pointSizeRange[0], pointSizeRange[1]]. The value written to PointSize **must** be greater than zero. If maintenance5 is enabled, and a value is not written to PointSize, the point size takes a default value of 1.0.

Not all point sizes need be supported, but the size 1.0 **must** be supported. The range of supported sizes and the size of evenly-spaced gradations within that range are implementation-dependent. The range and gradations are obtained from the pointSizeRange and pointSizeGranularity members of VkPhysicalDeviceLimits. If, for instance, the size range is from 0.1 to 2.0 and the gradation size is 0.1, then the sizes 0.1, 0.2, ..., 1.9, 2.0 are supported. Additional point sizes **may** also be supported.

There is no requirement that these sizes be equally spaced. If an unsupported size is requested, the nearest supported size is used instead.

Further, if the render pass has a fragment density map attachment, point size **may** be rounded by the implementation to a multiple of the fragment's width or height.

### 28.10.1. Basic Point Rasterization

Point rasterization produces a fragment for each fragment area group of framebuffer pixels with one or more sample points that intersect a region centered at the point's $(x_f,y_f)$. This region is a square with side equal to the current point size. Coverage bits that correspond to sample points that intersect the region are 1, other coverage bits are 0. All fragments produced in rasterizing a point are assigned the same associated data, which are those of the vertex corresponding to the point. However, the fragment shader built-in `PointCoord` contains point sprite texture coordinates. The s and t point sprite texture coordinates vary from zero to one across the point horizontally left-to-right and vertically top-to-bottom, respectively. The following formulas are used to evaluate s and t:

$$s = \frac{1}{2} + \frac{(x_p - x_f)}{\text{size}}$$

$$t = \frac{1}{2} + \frac{(y_p - y_f)}{\text{size}}$$

where size is the point's size; $(x_p,y_p)$ is the location at which the point sprite coordinates are evaluated - this **may** be the framebuffer coordinates of the fragment center, or the location of a sample; and $(x_f,y_f)$ is the exact, unrounded framebuffer coordinate of the vertex for the point.

## 28.11. Line Segments

Line segment rasterization options are controlled by the VkPipelineRasterizationLineStateCreateInfoKHR structure.

The `VkPipelineRasterizationLineStateCreateInfoKHR` structure is defined as:

```
// Provided by VK_KHR_line_rasterization
typedef struct VkPipelineRasterizationLineStateCreateInfoKHR {
    VkStructureType                 sType;
    const void*                     pNext;
    VkLineRasterizationModeKHR      lineRasterizationMode;
    VkBool32                        stippledLineEnable;
    uint32_t                        lineStippleFactor;
    uint16_t                        lineStipplePattern;
} VkPipelineRasterizationLineStateCreateInfoKHR;
```

or the equivalent

```
// Provided by VK_EXT_line_rasterization
typedef VkPipelineRasterizationLineStateCreateInfoKHR
VkPipelineRasterizationLineStateCreateInfoEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `lineRasterizationMode` is a VkLineRasterizationModeKHR value selecting the style of line rasterization.
- `stippledLineEnable` enables stippled line rasterization.
- `lineStippleFactor` is the repeat factor used in stippled line rasterization.
- `lineStipplePattern` is the bit pattern used in stippled line rasterization.

If `stippledLineEnable` is `VK_FALSE`, the values of `lineStippleFactor` and `lineStipplePattern` are ignored.

## Valid Usage

- VUID-VkPipelineRasterizationLineStateCreateInfoKHR-lineRasterizationMode-02768
  If `lineRasterizationMode` is `VK_LINE_RASTERIZATION_MODE_RECTANGULAR_KHR`, then the `rectangularLines` feature **must** be enabled
- VUID-VkPipelineRasterizationLineStateCreateInfoKHR-lineRasterizationMode-02769
  If `lineRasterizationMode` is `VK_LINE_RASTERIZATION_MODE_BRESENHAM_KHR`, then the `bresenhamLines` feature **must** be enabled
- VUID-VkPipelineRasterizationLineStateCreateInfoKHR-lineRasterizationMode-02770
  If `lineRasterizationMode` is `VK_LINE_RASTERIZATION_MODE_RECTANGULAR_SMOOTH_KHR`, then the `smoothLines` feature **must** be enabled
- VUID-VkPipelineRasterizationLineStateCreateInfoKHR-stippledLineEnable-02771
  If `stippledLineEnable` is `VK_TRUE` and `lineRasterizationMode` is `VK_LINE_RASTERIZATION_MODE_RECTANGULAR_KHR`, then the `stippledRectangularLines` feature **must** be enabled
- VUID-VkPipelineRasterizationLineStateCreateInfoKHR-stippledLineEnable-02772
  If `stippledLineEnable` is `VK_TRUE` and `lineRasterizationMode` is `VK_LINE_RASTERIZATION_MODE_BRESENHAM_KHR`, then the `stippledBresenhamLines` feature **must** be enabled
- VUID-VkPipelineRasterizationLineStateCreateInfoKHR-stippledLineEnable-02773
  If `stippledLineEnable` is `VK_TRUE` and `lineRasterizationMode` is `VK_LINE_RASTERIZATION_MODE_RECTANGULAR_SMOOTH_KHR`, then the `stippledSmoothLines` feature **must** be enabled
- VUID-VkPipelineRasterizationLineStateCreateInfoKHR-stippledLineEnable-02774
  If `stippledLineEnable` is `VK_TRUE` and `lineRasterizationMode` is `VK_LINE_RASTERIZATION_MODE_DEFAULT_KHR`, then the `stippledRectangularLines` feature **must** be enabled and VkPhysicalDeviceLimits::`strictLines` **must** be `VK_TRUE`

### Valid Usage (Implicit)

- VUID-VkPipelineRasterizationLineStateCreateInfoKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PIPELINE_RASTERIZATION_LINE_STATE_CREATE_INFO_KHR`
- VUID-VkPipelineRasterizationLineStateCreateInfoKHR-lineRasterizationMode-parameter
  `lineRasterizationMode` **must** be a valid VkLineRasterizationModeKHR value

Possible values of VkPipelineRasterizationLineStateCreateInfoKHR::`lineRasterizationMode` are:

```
// Provided by VK_KHR_line_rasterization
typedef enum VkLineRasterizationModeKHR {
    VK_LINE_RASTERIZATION_MODE_DEFAULT_KHR = 0,
    VK_LINE_RASTERIZATION_MODE_RECTANGULAR_KHR = 1,
    VK_LINE_RASTERIZATION_MODE_BRESENHAM_KHR = 2,
    VK_LINE_RASTERIZATION_MODE_RECTANGULAR_SMOOTH_KHR = 3,
    VK_LINE_RASTERIZATION_MODE_DEFAULT_EXT = VK_LINE_RASTERIZATION_MODE_DEFAULT_KHR,
    VK_LINE_RASTERIZATION_MODE_RECTANGULAR_EXT =
VK_LINE_RASTERIZATION_MODE_RECTANGULAR_KHR,
    VK_LINE_RASTERIZATION_MODE_BRESENHAM_EXT =
VK_LINE_RASTERIZATION_MODE_BRESENHAM_KHR,
    VK_LINE_RASTERIZATION_MODE_RECTANGULAR_SMOOTH_EXT =
VK_LINE_RASTERIZATION_MODE_RECTANGULAR_SMOOTH_KHR,
} VkLineRasterizationModeKHR;
```

or the equivalent

```
// Provided by VK_EXT_line_rasterization
typedef VkLineRasterizationModeKHR VkLineRasterizationModeEXT;
```

- `VK_LINE_RASTERIZATION_MODE_DEFAULT_KHR` is equivalent to `VK_LINE_RASTERIZATION_MODE_RECTANGULAR_KHR` if VkPhysicalDeviceLimits::`strictLines` is `VK_TRUE`, otherwise lines are drawn as non-`strictLines` parallelograms. Both of these modes are defined in Basic Line Segment Rasterization.
- `VK_LINE_RASTERIZATION_MODE_RECTANGULAR_KHR` specifies lines drawn as if they were rectangles extruded from the line
- `VK_LINE_RASTERIZATION_MODE_BRESENHAM_KHR` specifies lines drawn by determining which pixel diamonds the line intersects and exits, as defined in Bresenham Line Segment Rasterization.
- `VK_LINE_RASTERIZATION_MODE_RECTANGULAR_SMOOTH_KHR` specifies lines drawn if they were rectangles extruded from the line, with alpha falloff, as defined in Smooth Lines.

To dynamically set the `lineRasterizationMode` state, call:

```
// Provided by VK_EXT_extended_dynamic_state3 with VK_EXT_line_rasterization,
VK_EXT_line_rasterization with VK_EXT_shader_object
void vkCmdSetLineRasterizationModeEXT(
    VkCommandBuffer                             commandBuffer,
    VkLineRasterizationModeEXT                  lineRasterizationMode);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `lineRasterizationMode` specifies the `lineRasterizationMode` state.

This command sets the `lineRasterizationMode` state for subsequent drawing commands when drawing using shader objects, or when the graphics pipeline is created with `VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT` set in VkPipelineDynamicStateCreateInfo::`pDynamicStates`. Otherwise, this state is specified by the VkPipelineRasterizationLineStateCreateInfoKHR::`lineRasterizationMode` value used to create the currently active pipeline.

## Valid Usage

- VUID-vkCmdSetLineRasterizationModeEXT-None-09423
  At least one of the following **must** be true:
  - The `extendedDynamicState3LineRasterizationMode` feature is enabled
  - The `shaderObject` feature is enabled
- VUID-vkCmdSetLineRasterizationModeEXT-lineRasterizationMode-07418
  If `lineRasterizationMode` is `VK_LINE_RASTERIZATION_MODE_RECTANGULAR_KHR`, then the `rectangularLines` feature **must** be enabled
- VUID-vkCmdSetLineRasterizationModeEXT-lineRasterizationMode-07419
  If `lineRasterizationMode` is `VK_LINE_RASTERIZATION_MODE_BRESENHAM_KHR`, then the `bresenhamLines` feature **must** be enabled
- VUID-vkCmdSetLineRasterizationModeEXT-lineRasterizationMode-07420
  If `lineRasterizationMode` is `VK_LINE_RASTERIZATION_MODE_RECTANGULAR_SMOOTH_KHR`, then the `smoothLines` feature **must** be enabled

## Valid Usage (Implicit)

- VUID-vkCmdSetLineRasterizationModeEXT-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdSetLineRasterizationModeEXT-lineRasterizationMode-parameter
  `lineRasterizationMode` **must** be a valid VkLineRasterizationModeEXT value
- VUID-vkCmdSetLineRasterizationModeEXT-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdSetLineRasterizationModeEXT-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics

operations

- VUID-vkCmdSetLineRasterizationModeEXT-videocoding
  This command **must** only be called outside of a video coding scope

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics | State |

To dynamically set the `stippledLineEnable` state, call:

```
// Provided by VK_EXT_extended_dynamic_state3 with VK_EXT_line_rasterization,
VK_EXT_line_rasterization with VK_EXT_shader_object
void vkCmdSetLineStippleEnableEXT(
    VkCommandBuffer                             commandBuffer,
    VkBool32                                    stippledLineEnable);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `stippledLineEnable` specifies the `stippledLineEnable` state.

This command sets the `stippledLineEnable` state for subsequent drawing commands when drawing using shader objects, or when the graphics pipeline is created with `VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT` set in VkPipelineDynamicStateCreateInfo::`pDynamicStates`. Otherwise, this state is specified by the VkPipelineRasterizationLineStateCreateInfoKHR::`stippledLineEnable` value used to create the currently active pipeline.

## Valid Usage

- VUID-vkCmdSetLineStippleEnableEXT-None-09423
  At least one of the following **must** be true:
  - The `extendedDynamicState3LineStippleEnable` feature is enabled
  - The `shaderObject` feature is enabled

## Valid Usage (Implicit)

- VUID-vkCmdSetLineStippleEnableEXT-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdSetLineStippleEnableEXT-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdSetLineStippleEnableEXT-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- VUID-vkCmdSetLineStippleEnableEXT-videocoding
  This command **must** only be called outside of a video coding scope

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics | State |

To dynamically set the line width, call:

```
// Provided by VK_VERSION_1_0
void vkCmdSetLineWidth(
    VkCommandBuffer                                 commandBuffer,
    float                                           lineWidth);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `lineWidth` is the width of rasterized line segments.

This command sets the line width for subsequent drawing commands when drawing using shader objects, or when the graphics pipeline is created with `VK_DYNAMIC_STATE_LINE_WIDTH` set in VkPipelineDynamicStateCreateInfo::`pDynamicStates`. Otherwise, this state is specified by the VkPipelineRasterizationStateCreateInfo::`lineWidth` value used to create the currently active pipeline.

## Valid Usage

- VUID-vkCmdSetLineWidth-lineWidth-00788
  If the `wideLines` feature is not enabled, `lineWidth` **must** be `1.0`

## Valid Usage (Implicit)

- VUID-vkCmdSetLineWidth-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdSetLineWidth-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdSetLineWidth-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- VUID-vkCmdSetLineWidth-videocoding
  This command **must** only be called outside of a video coding scope

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics | State |

Not all line widths need be supported for line segment rasterization, but width 1.0 antialiased segments **must** be provided. The range and gradations are obtained from the `lineWidthRange` and `lineWidthGranularity` members of VkPhysicalDeviceLimits. If, for instance, the size range is from 0.1 to 2.0 and the gradation size is 0.1, then the sizes 0.1, 0.2, ..., 1.9, 2.0 are supported. Additional line widths **may** also be supported. There is no requirement that these widths be equally spaced. If an unsupported width is requested, the nearest supported width is used instead.

Further, if the render pass has a fragment density map attachment, line width **may** be rounded by the implementation to a multiple of the fragment's width or height.

### 28.11.1. Basic Line Segment Rasterization

If the `lineRasterizationMode` member of VkPipelineRasterizationLineStateCreateInfoKHR is `VK_LINE_RASTERIZATION_MODE_RECTANGULAR_KHR`, rasterized line segments produce fragments which intersect a rectangle centered on the line segment. Two of the edges are parallel to the specified line segment; each is at a distance of one-half the current width from that segment in directions perpendicular to the direction of the line. The other two edges pass through the line endpoints and are perpendicular to the direction of the specified line segment. Coverage bits that correspond to sample points that intersect the rectangle are 1, other coverage bits are 0.

Next we specify how the data associated with each rasterized fragment are obtained. Let $\mathbf{p}_r = (x_d, y_d)$ be the framebuffer coordinates at which associated data are evaluated. This **may** be the center of a fragment or the location of a sample within the fragment. When `rasterizationSamples` is `VK_SAMPLE_COUNT_1_BIT`, the fragment center **must** be used. Let $\mathbf{p}_a = (x_a, y_a)$ and $\mathbf{p}_b = (x_b, y_b)$ be initial and final endpoints of the line segment, respectively. Set

$$t = \frac{(\mathbf{p}_r - \mathbf{p}_a) \cdot (\mathbf{p}_b - \mathbf{p}_a)}{\| \mathbf{p}_b - \mathbf{p}_a \|^2}$$

(Note that t = 0 at $\mathbf{p}_a$ and t = 1 at $\mathbf{p}_b$. Also note that this calculation projects the vector from $\mathbf{p}_a$ to $\mathbf{p}_r$ onto the line, and thus computes the normalized distance of the fragment along the line.)

If `strictLines` is `VK_TRUE`, line segments are rasterized using perspective or linear interpolation.

*Perspective interpolation* for a line segment interpolates two values in a manner that is correct when taking the perspective of the viewport into consideration, by way of the line segment's clip coordinates. An interpolated value f can be determined by

$$f = \frac{(1-t) f_a / w_a + t f_b / w_b}{(1-t) / w_a + t / w_b}$$

where $f_a$ and $f_b$ are the data associated with the starting and ending endpoints of the segment, respectively; $w_a$ and $w_b$ are the clip w coordinates of the starting and ending endpoints of the segment, respectively.

*Linear interpolation* for a line segment directly interpolates two values, and an interpolated value f can be determined by

$$f = (1 - t) f_a + t f_b$$

where $f_a$ and $f_b$ are the data associated with the starting and ending endpoints of the segment, respectively.

The clip coordinate w for a sample is determined using perspective interpolation. The depth value z for a sample is determined using linear interpolation. Interpolation of fragment shader input values are determined by Interpolation decorations.

The above description documents the preferred method of line rasterization, and **must** be used when `lineRasterizationMode` is `VK_LINE_RASTERIZATION_MODE_RECTANGULAR_KHR`.

By default, when `strictLines` is `VK_FALSE`, or the `relaxedLineRasterization` feature is enabled, and when the `lineRasterizationMode` is `VK_LINE_RASTERIZATION_MODE_DEFAULT_KHR`, the edges of the lines are generated as a parallelogram surrounding the original line. The major axis is chosen by noting the axis in which there is the greatest distance between the line start and end points. If the difference is equal in both directions then the X axis is chosen as the major axis. Edges 2 and 3 are aligned to the minor axis and are centered on the endpoints of the line as in Non strict lines, and each is `lineWidth` long. Edges 0 and 1 are parallel to the line and connect the endpoints of edges 2 and 3. Coverage bits that correspond to sample points that intersect the parallelogram are 1, other coverage bits are 0.

Samples that fall exactly on the edge of the parallelogram follow the polygon rasterization rules.

Interpolation occurs as if the parallelogram was decomposed into two triangles where each pair of vertices at each end of the line has identical attributes.

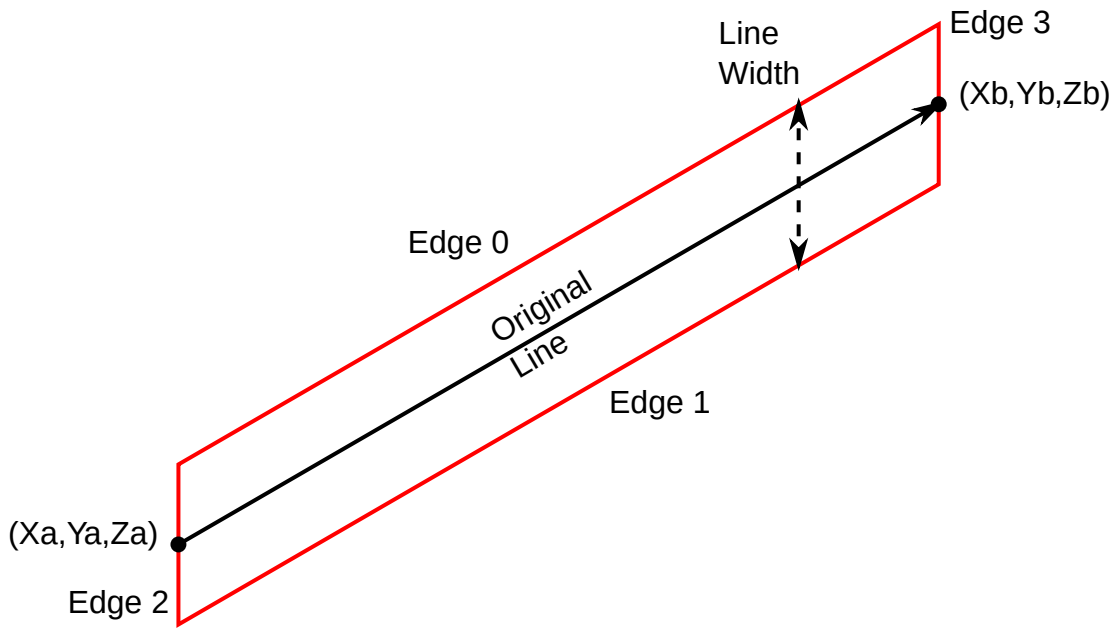



*Figure 20. Non strict lines*

When `strictLines` is `VK_FALSE` or when the `relaxedLineRasterization` feature is enabled, and `lineRasterizationMode` is `VK_LINE_RASTERIZATION_MODE_DEFAULT_EXT` implementations **may** deviate from the non-strict line algorithm described above in the following ways:

- Implementations **may** instead interpolate each fragment according to the formula in Basic Line Segment Rasterization using the original line segment endpoints.
- Rasterization of non-antialiased non-strict line segments **may** be performed using the rules defined in Bresenham Line Segment Rasterization.

If `VkPhysicalDeviceMaintenance5PropertiesKHR::nonStrictSinglePixelWideLinesUseParallelogram` is `VK_TRUE`, the `lineRasterizationMode` is `VK_LINE_RASTERIZATION_MODE_DEFAULT_EXT`, and `strictLines` is `VK_FALSE`, non-strict lines of width 1.0 are rasterized as parallelograms, otherwise they are rasterized using Bresenham's algorithm.

If `VkPhysicalDeviceMaintenance5PropertiesKHR::nonStrictWideLinesUseParallelogram` is `VK_TRUE`, the `lineRasterizationMode` is `VK_LINE_RASTERIZATION_MODE_DEFAULT_EXT`, and `strictLines` is `VK_FALSE`, non-strict lines of width greater than 1.0 are rasterized as parallelograms, otherwise they are rasterized

using Bresenham's algorithm.

### 28.11.2. Bresenham Line Segment Rasterization

If `lineRasterizationMode` is `VK_LINE_RASTERIZATION_MODE_BRESENHAM_KHR`, then the following rules replace the line rasterization rules defined in Basic Line Segment Rasterization.

Non-strict lines **may** also follow these rasterization rules for non-antialiased lines.

If the `relaxedLineRasterization` feature is enabled, and `lineRasterizationMode` is `VK_LINE_RASTERIZATION_MODE_DEFAULT_EXT` implementations **must** follow these rasterization rules for non-antialised lines of width 1.0.

Line segment rasterization begins by characterizing the segment as either *x-major* or *y-major*. x-major line segments have slope in the closed interval [-1,1]; all other line segments are y-major (slope is determined by the segment's endpoints). We specify rasterization only for x-major segments except in cases where the modifications for y-major segments are not self-evident.

Ideally, Vulkan uses a *diamond-exit* rule to determine those fragments that are produced by rasterizing a line segment. For each fragment f with center at framebuffer coordinates $x_f$ and $y_f$, define a diamond-shaped region that is the intersection of four half planes:

$$R_f = \{(x, y) \mid |x - x_f| + |y - y_f| < \frac{1}{2}\}$$

Essentially, a line segment starting at $p_a$ and ending at $p_b$ produces those fragments f for which the segment intersects $R_f$, except if $p_b$ is contained in $R_f$.

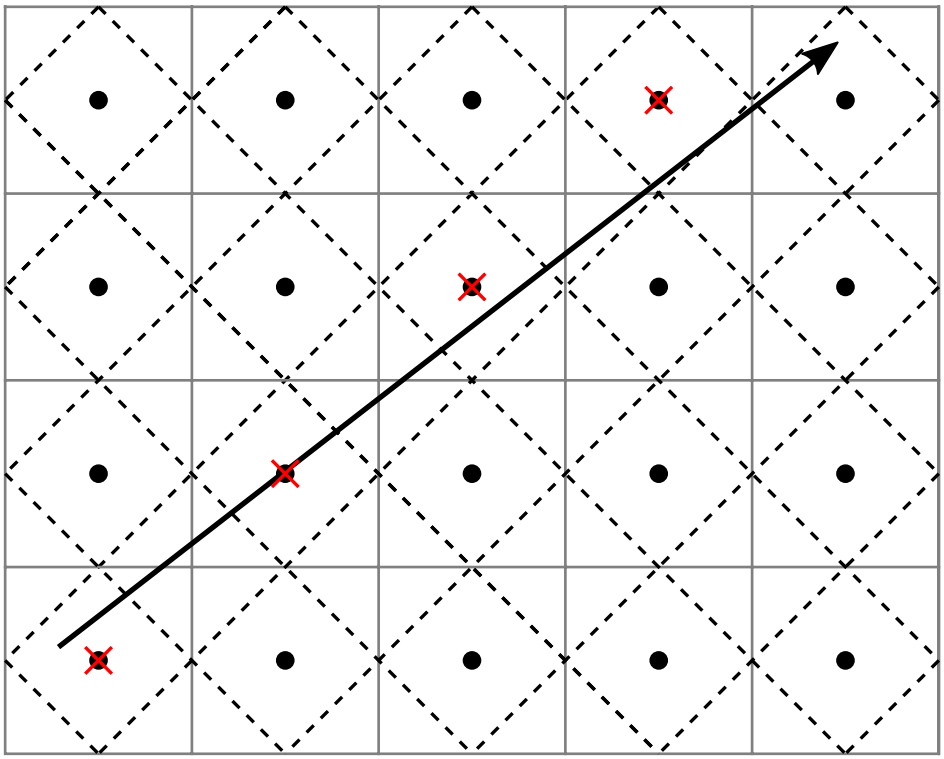


*Figure 21. Visualization of Bresenham's algorithm*

To avoid difficulties when an endpoint lies on a boundary of $R_f$ we (in principle) perturb the supplied endpoints by a tiny amount. Let $p_a$ and $p_b$ have framebuffer coordinates $(x_a, y_a)$ and $(x_b, y_b)$, respectively. Obtain the perturbed endpoints $p_a'$ given by $(x_a, y_a) - (\varepsilon, \varepsilon^2)$ and $p_b'$ given by $(x_b, y_b) - (\varepsilon, \varepsilon^2)$. Rasterizing the line segment starting at $p_a$ and ending at $p_b$ produces those fragments f for which the segment starting at $p_a'$ and ending on $p_b'$ intersects $R_f$, except if $p_b'$ is contained in $R_f$. $\varepsilon$ is chosen to be so small that rasterizing the line segment produces the same fragments when $\delta$ is substituted for $\varepsilon$ for any $0 < \delta \leq \varepsilon$.

When $p_a$ and $p_b$ lie on fragment centers, this characterization of fragments reduces to Bresenham's algorithm with one modification: lines produced in this description are "half-open", meaning that the final fragment (corresponding to $p_b$) is not drawn. This means that when rasterizing a series of connected line segments, shared endpoints will be produced only once rather than twice (as would occur with Bresenham's algorithm).

Implementations **may** use other line segment rasterization algorithms, subject to the following rules:

- The coordinates of a fragment produced by the algorithm **must** not deviate by more than one unit in either x or y framebuffer coordinates from a corresponding fragment produced by the diamond-exit rule.
- The total number of fragments produced by the algorithm **must** not differ from that produced by the diamond-exit rule by more than one.
- For an x-major line, two fragments that lie in the same framebuffer-coordinate column **must** not be produced (for a y-major line, two fragments that lie in the same framebuffer-coordinate row **must** not be produced).
- If two line segments share a common endpoint, and both segments are either x-major (both left-to-right or both right-to-left) or y-major (both bottom-to-top or both top-to-bottom), then rasterizing both segments **must** not produce duplicate fragments. Fragments also **must** not be omitted so as to interrupt continuity of the connected segments.

The actual width w of Bresenham lines is determined by rounding the line width to the nearest integer, clamping it to the implementation-dependent `lineWidthRange` (with both values rounded to the nearest integer), then clamping it to be no less than 1.

Bresenham line segments of width other than one are rasterized by offsetting them in the minor direction (for an x-major line, the minor direction is y, and for a y-major line, the minor direction is x) and producing a row or column of fragments in the minor direction. If the line segment has endpoints given by $(x_0, y_0)$ and $(x_1, y_1)$ in framebuffer coordinates, the segment with endpoints $(x_0, y_0 - \frac{w-1}{2})$ and $(x_1, y_1 - \frac{w-1}{2})$ is rasterized, but instead of a single fragment, a column of fragments of height w (a row of fragments of length w for a y-major segment) is produced at each x (y for y-major) location. The lowest fragment of this column is the fragment that would be produced by rasterizing the segment of width 1 with the modified coordinates.

The preferred method of attribute interpolation for a wide line is to generate the same attribute values for all fragments in the row or column described above, as if the adjusted line was used for interpolation and those values replicated to the other fragments, except for `FragCoord` which is interpolated as usual. Implementations **may** instead interpolate each fragment according to the formula in Basic Line Segment Rasterization, using the original line segment endpoints.

When Bresenham lines are being rasterized, sample locations **may** all be treated as being at the pixel center (this **may** affect attribute and depth interpolation).

| | |
|---|---|
| **NOTE** | The sample locations described above are **not** used for determining coverage, they are only used for things like attribute interpolation. The rasterization rules that determine coverage are defined in terms of whether the line intersects **pixels**, as opposed to the point sampling rules used for other primitive types. So these rules are independent of the sample locations. One consequence of this is that Bresenham lines cover the same pixels regardless of the number of rasterization samples, and cover all samples in those pixels (unless masked out or killed). |

### 28.11.3. Line Stipple

If the `stippledLineEnable` member of VkPipelineRasterizationLineStateCreateInfoKHR is `VK_TRUE`, then lines are rasterized with a *line stipple* determined by `lineStippleFactor` and `lineStipplePattern`. `lineStipplePattern` is an unsigned 16-bit integer that determines which fragments are to be drawn or discarded when the line is rasterized. `lineStippleFactor` is a count that is used to modify the effective line stipple by causing each bit in `lineStipplePattern` to be used `lineStippleFactor` times.

Line stippling discards certain fragments that are produced by rasterization. The masking is achieved using three parameters: the 16-bit line stipple pattern $p$, the line stipple factor $r$, and an integer stipple counter $s$. Let

$$b = \lfloor \frac{s}{r} \rfloor \bmod 16$$

Then a fragment is produced if the $b$'th bit of $p$ is 1, and discarded otherwise. The bits of $p$ are numbered with 0 being the least significant and 15 being the most significant.

The initial value of $s$ is zero. For `VK_LINE_RASTERIZATION_MODE_BRESENHAM_KHR` lines, $s$ is incremented after production of each fragment of a line segment (fragments are produced in order, beginning at the starting point and working towards the ending point). For `VK_LINE_RASTERIZATION_MODE_RECTANGULAR_KHR` and `VK_LINE_RASTERIZATION_MODE_RECTANGULAR_SMOOTH_KHR` lines, the rectangular region is subdivided into adjacent unit-length rectangles, and s is incremented once for each rectangle. Rectangles with a value of $s$ such that the $b$'th bit of $p$ is zero are discarded. If the last rectangle in a line segment is shorter than unit-length, then the remainder **may** carry over to the next line segment in the line strip using the same value of $s$ (this is the preferred behavior, for the stipple pattern to appear more consistent through the strip).

$s$ is reset to 0 at the start of each strip (for line strips), and before every line segment in a group of independent segments.

If the line segment has been clipped, then the value of $s$ at the beginning of the line segment is implementation-dependent.

To dynamically set the line stipple state, call:

```
// Provided by VK_KHR_line_rasterization
void vkCmdSetLineStippleKHR(
    VkCommandBuffer                             commandBuffer,
    uint32_t                                    lineStippleFactor,
    uint16_t                                    lineStipplePattern);
```

or the equivalent command

```
// Provided by VK_EXT_line_rasterization
void vkCmdSetLineStippleEXT(
    VkCommandBuffer                             commandBuffer,
    uint32_t                                    lineStippleFactor,
    uint16_t                                    lineStipplePattern);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `lineStippleFactor` is the repeat factor used in stippled line rasterization.
- `lineStipplePattern` is the bit pattern used in stippled line rasterization.

This command sets the line stipple state for subsequent drawing commands when drawing using shader objects, or when the graphics pipeline is created with `VK_DYNAMIC_STATE_LINE_STIPPLE_EXT` set in VkPipelineDynamicStateCreateInfo::`pDynamicStates`. Otherwise, this state is specified by the VkPipelineRasterizationLineStateCreateInfoKHR::`lineStippleFactor` and VkPipelineRasterizationLineStateCreateInfoKHR::`lineStipplePattern` values used to create the currently active pipeline.

### Valid Usage

- VUID-vkCmdSetLineStippleKHR-lineStippleFactor-02776
  `lineStippleFactor` **must** be in the range [1,256]

### Valid Usage (Implicit)

- VUID-vkCmdSetLineStippleKHR-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdSetLineStippleKHR-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdSetLineStippleKHR-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- VUID-vkCmdSetLineStippleKHR-videocoding
  This command **must** only be called outside of a video coding scope

### Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

### Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics | State |

### 28.11.4. Smooth Lines

If the `lineRasterizationMode` member of VkPipelineRasterizationLineStateCreateInfoKHR is `VK_LINE_RASTERIZATION_MODE_RECTANGULAR_SMOOTH_KHR`, then lines are considered to be rectangles using the same geometry as for `VK_LINE_RASTERIZATION_MODE_RECTANGULAR_KHR` lines. The rules for determining which pixels are covered are implementation-dependent, and **may** include nearby pixels where no sample locations are covered or where the rectangle does not intersect the pixel at all. For each pixel that is considered covered, the fragment computes a coverage value that approximates the area of the intersection of the rectangle with the pixel square, and this coverage value is multiplied into the color location 0's alpha value after fragment shading, as described in Multisample Coverage.

| | |
|---|---|
| **NOTE** | The details of the rasterization rules and area calculation are left intentionally vague, to allow implementations to generate coverage and values that are aesthetically pleasing. |

## 28.12. Polygons

A polygon results from the decomposition of a triangle strip, triangle fan or a series of independent triangles. Like points and line segments, polygon rasterization is controlled by several variables in the VkPipelineRasterizationStateCreateInfo structure.

### 28.12.1. Basic Polygon Rasterization

The first step of polygon rasterization is to determine whether the triangle is *back-facing* or *front-facing*. This determination is made based on the sign of the (clipped or unclipped) polygon's area computed in framebuffer coordinates. One way to compute this area is:

$$a = -\frac{1}{2}\sum_{i=0}^{n-1} x_f^i y_f^{i \oplus 1} - x_f^{i \oplus 1} y_f^i$$

where $x_f^i$ and $y_f^i$ are the x and y framebuffer coordinates of the ith vertex of the n-vertex polygon (vertices are numbered starting at zero for the purposes of this computation) and i 1 is (i + 1) mod n.

The interpretation of the sign of a is determined by the VkPipelineRasterizationStateCreateInfo::frontFace property of the currently active pipeline. Possible values are:

```
// Provided by VK_VERSION_1_0
typedef enum VkFrontFace {
    VK_FRONT_FACE_COUNTER_CLOCKWISE = 0,
    VK_FRONT_FACE_CLOCKWISE = 1,
} VkFrontFace;
```

- VK_FRONT_FACE_COUNTER_CLOCKWISE specifies that a triangle with positive area is considered front-facing.
- VK_FRONT_FACE_CLOCKWISE specifies that a triangle with negative area is considered front-facing.

Any triangle which is not front-facing is back-facing, including zero-area triangles.

To dynamically set the front face orientation, call:

```
// Provided by VK_EXT_extended_dynamic_state, VK_EXT_shader_object
void vkCmdSetFrontFaceEXT(
    VkCommandBuffer                             commandBuffer,
    VkFrontFace                                 frontFace);
```

- commandBuffer is the command buffer into which the command will be recorded.
- frontFace is a VkFrontFace value specifying the front-facing triangle orientation to be used for culling.

This command sets the front face orientation for subsequent drawing commands when drawing using shader objects, or when the graphics pipeline is created with VK_DYNAMIC_STATE_FRONT_FACE set in VkPipelineDynamicStateCreateInfo::pDynamicStates. Otherwise, this state is specified by the VkPipelineRasterizationStateCreateInfo::frontFace value used to create the currently active pipeline.

### Valid Usage

- VUID-vkCmdSetFrontFace-None-08971
  At least one of the following **must** be true:
  - the extendedDynamicState feature is enabled
  - the shaderObject feature is enabled

## Valid Usage (Implicit)

- VUID-vkCmdSetFrontFace-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdSetFrontFace-frontFace-parameter
  `frontFace` **must** be a valid VkFrontFace value
- VUID-vkCmdSetFrontFace-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdSetFrontFace-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- VUID-vkCmdSetFrontFace-videocoding
  This command **must** only be called outside of a video coding scope

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics | State |

Once the orientation of triangles is determined, they are culled according to the VkPipelineRasterizationStateCreateInfo::`cullMode` property of the currently active pipeline. Possible values are:

```
// Provided by VK_VERSION_1_0
typedef enum VkCullModeFlagBits {
    VK_CULL_MODE_NONE = 0,
    VK_CULL_MODE_FRONT_BIT = 0x00000001,
    VK_CULL_MODE_BACK_BIT = 0x00000002,
    VK_CULL_MODE_FRONT_AND_BACK = 0x00000003,
} VkCullModeFlagBits;
```

- `VK_CULL_MODE_NONE` specifies that no triangles are discarded
- `VK_CULL_MODE_FRONT_BIT` specifies that front-facing triangles are discarded

- VK_CULL_MODE_BACK_BIT specifies that back-facing triangles are discarded
- VK_CULL_MODE_FRONT_AND_BACK specifies that all triangles are discarded.

Following culling, fragments are produced for any triangles which have not been discarded.

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkCullModeFlags;
```

VkCullModeFlags is a bitmask type for setting a mask of zero or more VkCullModeFlagBits.

To dynamically set the cull mode, call:

```
// Provided by VK_EXT_extended_dynamic_state, VK_EXT_shader_object
void vkCmdSetCullModeEXT(
    VkCommandBuffer                             commandBuffer,
    VkCullModeFlags                             cullMode);
```

- commandBuffer is the command buffer into which the command will be recorded.
- cullMode specifies the cull mode property to use for drawing.

This command sets the cull mode for subsequent drawing commands when drawing using shader objects, or when the graphics pipeline is created with VK_DYNAMIC_STATE_CULL_MODE set in VkPipelineDynamicStateCreateInfo::pDynamicStates. Otherwise, this state is specified by the VkPipelineRasterizationStateCreateInfo::cullMode value used to create the currently active pipeline.

### Valid Usage

- VUID-vkCmdSetCullMode-None-08971
  At least one of the following **must** be true:
  - the extendedDynamicState feature is enabled
  - the shaderObject feature is enabled

### Valid Usage (Implicit)

- VUID-vkCmdSetCullMode-commandBuffer-parameter
  commandBuffer **must** be a valid VkCommandBuffer handle
- VUID-vkCmdSetCullMode-cullMode-parameter
  cullMode **must** be a valid combination of VkCullModeFlagBits values
- VUID-vkCmdSetCullMode-commandBuffer-recording
  commandBuffer **must** be in the recording state
- VUID-vkCmdSetCullMode-commandBuffer-cmdpool
  The VkCommandPool that commandBuffer was allocated from **must** support graphics operations

- VUID-vkCmdSetCullMode-videocoding
  This command **must** only be called outside of a video coding scope

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics | State |

The rule for determining which fragments are produced by polygon rasterization is called *point sampling*. The two-dimensional projection obtained by taking the x and y framebuffer coordinates of the polygon's vertices is formed. Fragments are produced for any fragment area groups of pixels for which any sample points lie inside of this polygon. Coverage bits that correspond to sample points that satisfy the point sampling criteria are 1, other coverage bits are 0. Special treatment is given to a sample whose sample location lies on a polygon edge. In such a case, if two polygons lie on either side of a common edge (with identical endpoints) on which a sample point lies, then exactly one of the polygons **must** result in a covered sample for that fragment during rasterization. As for the data associated with each fragment produced by rasterizing a polygon, we begin by specifying how these values are produced for fragments in a triangle.

*Barycentric coordinates* are a set of three numbers, a, b, and c, each in the range [0,1], with a + b + c = 1. These coordinates uniquely specify any point p within the triangle or on the triangle's boundary as

$$p = a\, p_a + b\, p_b + c\, p_c$$

where $p_a$, $p_b$, and $p_c$ are the vertices of the triangle. a, b, and c are determined by:

$$a = \frac{\mathrm{A}(p p_b p_c)}{\mathrm{A}(p_a p_b p_c)}, \quad b = \frac{\mathrm{A}(p p_a p_c)}{\mathrm{A}(p_a p_b p_c)}, \quad c = \frac{\mathrm{A}(p p_a p_b)}{\mathrm{A}(p_a p_b p_c)},$$

where A(lmn) denotes the area in framebuffer coordinates of the triangle with vertices l, m, and n.

Denote an associated datum at $p_a$, $p_b$, or $p_c$ as $f_a$, $f_b$, or $f_c$, respectively.

*Perspective interpolation* for a triangle interpolates three values in a manner that is correct when taking the perspective of the viewport into consideration, by way of the triangle's clip coordinates.

An interpolated value f can be determined by

$$f = \frac{a f_a / w_a + b f_b / w_b + c f_c / w_c}{a / w_a + b / w_b + c / w_c}$$

where $w_a$, $w_b$, and $w_c$ are the clip w coordinates of $p_a$, $p_b$, and $p_c$, respectively. a, b, and c are the barycentric coordinates of the location at which the data are produced.

*Linear interpolation* for a triangle directly interpolates three values, and an interpolated value f can be determined by

$$f = a f_a + b f_b + c f_c$$

where $f_a$, $f_b$, and $f_c$ are the data associated with $p_a$, $p_b$, and $p_c$, respectively.

The clip coordinate w for a sample is determined using perspective interpolation. The depth value z for a sample is determined using linear interpolation. Interpolation of fragment shader input values are determined by Interpolation decorations.

For a polygon with more than three edges, such as are produced by clipping a triangle, a convex combination of the values of the datum at the polygon's vertices **must** be used to obtain the value assigned to each fragment produced by the rasterization algorithm. That is, it **must** be the case that at every fragment

$$f = \sum_{i=1}^{n} a_i f_i$$

where n is the number of vertices in the polygon and $f_i$ is the value of f at vertex i. For each i, $0 \le a_i \le 1$ and $\sum_{i=1}^{n} a_i = 1$. The values of $a_i$ **may** differ from fragment to fragment, but at vertex i, $a_i = 1$ and $a_j = 0$ for $j \neq i$.

| | |
|---|---|
| **NOTE** | One algorithm that achieves the required behavior is to triangulate a polygon (without adding any vertices) and then treat each triangle individually as already discussed. A scan-line rasterizer that linearly interpolates data along each edge and then linearly interpolates data across each horizontal span from edge to edge also satisfies the restrictions (in this case the numerator and denominator of perspective interpolation are iterated independently, and a division is performed for each fragment). |

### 28.12.2. Polygon Mode

Possible values of the VkPipelineRasterizationStateCreateInfo::`polygonMode` property of the currently active pipeline, specifying the method of rasterization for polygons, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkPolygonMode {
    VK_POLYGON_MODE_FILL = 0,
    VK_POLYGON_MODE_LINE = 1,
    VK_POLYGON_MODE_POINT = 2,
  // Provided by VK_NV_fill_rectangle
    VK_POLYGON_MODE_FILL_RECTANGLE_NV = 1000153000,
} VkPolygonMode;
```

- VK_POLYGON_MODE_POINT specifies that polygon vertices are drawn as points.
- VK_POLYGON_MODE_LINE specifies that polygon edges are drawn as line segments.
- VK_POLYGON_MODE_FILL specifies that polygons are rendered using the polygon rasterization rules in this section.
- VK_POLYGON_MODE_FILL_RECTANGLE_NV specifies that polygons are rendered using polygon rasterization rules, modified to consider a sample within the primitive if the sample location is inside the axis-aligned bounding box of the triangle after projection. Note that the barycentric weights used in attribute interpolation **can** extend outside the range [0,1] when these primitives are shaded. Special treatment is given to a sample position on the boundary edge of the bounding box. In such a case, if two rectangles lie on either side of a common edge (with identical endpoints) on which a sample position lies, then exactly one of the triangles **must** produce a fragment that covers that sample during rasterization.

  Polygons rendered in VK_POLYGON_MODE_FILL_RECTANGLE_NV mode **may** be clipped by the frustum or by user clip planes. If clipping is applied, the triangle is culled rather than clipped.

  Area calculation and facingness are determined for VK_POLYGON_MODE_FILL_RECTANGLE_NV mode using the triangle's vertices.

These modes affect only the final rasterization of polygons: in particular, a polygon's vertices are shaded and the polygon is clipped and possibly culled before these modes are applied.

If VkPhysicalDeviceMaintenance5PropertiesKHR::polygonModePointSize is set to VK_TRUE, the point size of the final rasterization of polygons is taken from PointSize when polygon mode is VK_POLYGON_MODE_POINT.

Otherwise, if VkPhysicalDeviceMaintenance5PropertiesKHR::polygonModePointSize is set to VK_FALSE, the point size of the final rasterization of polygons is 1.0 when polygon mode is VK_POLYGON_MODE_POINT.

To dynamically set the polygon mode, call:

```
// Provided by VK_EXT_extended_dynamic_state3, VK_EXT_shader_object
void vkCmdSetPolygonModeEXT(
    VkCommandBuffer                             commandBuffer,
    VkPolygonMode                               polygonMode);
```

- commandBuffer is the command buffer into which the command will be recorded.

- `polygonMode` specifies polygon mode.

This command sets the polygon mode for subsequent drawing commands when drawing using shader objects, or when the graphics pipeline is created with `VK_DYNAMIC_STATE_POLYGON_MODE_EXT` set in VkPipelineDynamicStateCreateInfo::`pDynamicStates`. Otherwise, this state is specified by the VkPipelineRasterizationStateCreateInfo::`polygonMode` value used to create the currently active pipeline.

## Valid Usage

- VUID-vkCmdSetPolygonModeEXT-None-09423
  At least one of the following **must** be true:
  - The `extendedDynamicState3PolygonMode` feature is enabled
  - The `shaderObject` feature is enabled
- VUID-vkCmdSetPolygonModeEXT-fillModeNonSolid-07424
  If the `fillModeNonSolid` feature is not enabled, `polygonMode` **must** be `VK_POLYGON_MODE_FILL` or `VK_POLYGON_MODE_FILL_RECTANGLE_NV`
- VUID-vkCmdSetPolygonModeEXT-polygonMode-07425
  If the `VK_NV_fill_rectangle` extension is not enabled, `polygonMode` **must** not be `VK_POLYGON_MODE_FILL_RECTANGLE_NV`

## Valid Usage (Implicit)

- VUID-vkCmdSetPolygonModeEXT-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdSetPolygonModeEXT-polygonMode-parameter
  `polygonMode` **must** be a valid VkPolygonMode value
- VUID-vkCmdSetPolygonModeEXT-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdSetPolygonModeEXT-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- VUID-vkCmdSetPolygonModeEXT-videocoding
  This command **must** only be called outside of a video coding scope

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

### Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics | State |

## 28.12.3. Depth Bias

The depth values of all fragments generated by the rasterization of a polygon **can** be biased (offset) by a single depth bias value $o$ that is computed for that polygon.

#### Depth Bias Enable

The depth bias computation is enabled by the `depthBiasEnable` set with vkCmdSetDepthBiasEnableEXT and `vkCmdSetDepthBiasEnableEXT`, or the corresponding VkPipelineRasterizationStateCreateInfo::`depthBiasEnable` value used to create the currently active pipeline. If the depth bias enable is `VK_FALSE`, no bias is applied and the fragment's depth values are unchanged.

To dynamically enable whether to bias fragment depth values, call:

```
// Provided by VK_EXT_extended_dynamic_state2, VK_EXT_shader_object
void vkCmdSetDepthBiasEnableEXT(
    VkCommandBuffer                             commandBuffer,
    VkBool32                                    depthBiasEnable);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `depthBiasEnable` controls whether to bias fragment depth values.

This command sets the depth bias enable for subsequent drawing commands when drawing using shader objects, or when the graphics pipeline is created with `VK_DYNAMIC_STATE_DEPTH_BIAS_ENABLE` set in VkPipelineDynamicStateCreateInfo::`pDynamicStates`. Otherwise, this state is specified by the VkPipelineRasterizationStateCreateInfo::`depthBiasEnable` value used to create the currently active pipeline.

### Valid Usage

- VUID-vkCmdSetDepthBiasEnable-None-08970
  At least one of the following **must** be true:
  - the `extendedDynamicState2` feature is enabled
  - the `shaderObject` feature is enabled

## Valid Usage (Implicit)

- VUID-vkCmdSetDepthBiasEnable-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdSetDepthBiasEnable-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdSetDepthBiasEnable-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- VUID-vkCmdSetDepthBiasEnable-videocoding
  This command **must** only be called outside of a video coding scope

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics | State |

**Depth Bias Computation**

The depth bias depends on three parameters:

- `depthBiasSlopeFactor` scales the maximum depth slope m of the polygon
- `depthBiasConstantFactor` scales the parameter r of the depth attachment
- the scaled terms are summed to produce a value which is then clamped to a minimum or maximum value specified by `depthBiasClamp`

`depthBiasSlopeFactor`, `depthBiasConstantFactor`, and `depthBiasClamp` **can** each be positive, negative, or zero. These parameters are set as described for vkCmdSetDepthBias and vkCmdSetDepthBias2EXT below.

The maximum depth slope m of a triangle is

$$m = \sqrt{\left(\frac{\partial z_f}{\partial x_f}\right)^2 + \left(\frac{\partial z_f}{\partial y_f}\right)^2}$$

where ($x_f$, $y_f$, $z_f$) is a point on the triangle. m **may** be approximated as

$$m = \max\left(\left|\frac{\partial z_f}{\partial x_f}\right|, \left|\frac{\partial z_f}{\partial y_f}\right|\right).$$

In a pipeline with a depth bias representation of `VK_DEPTH_BIAS_REPRESENTATION_FLOAT_EXT`, r, for the given primitive is defined as

$$r = 1$$

Otherwise r is the minimum resolvable difference that depends on the depth attachment representation. If `VkDepthBiasRepresentationInfoEXT`::`depthBiasExact` is `VK_FALSE` it is the smallest difference in framebuffer coordinate z values that is guaranteed to remain distinct throughout polygon rasterization and in the depth attachment. All pairs of fragments generated by the rasterization of two polygons with otherwise identical vertices, but $z_f$ values that differ by r, will have distinct depth values.

For fixed-point depth attachment representations, or in a pipeline with a depth bias representation of `VK_DEPTH_BIAS_REPRESENTATION_LEAST_REPRESENTABLE_VALUE_FORCE_UNORM_EXT`, r is constant throughout the range of the entire depth attachment. If `VkDepthBiasRepresentationInfoEXT`::`depthBiasExact` is `VK_TRUE`, then its value **must** be

$$r = 2^{-n}$$

Otherwise its value is implementation-dependent but **must** be at most

$$r = 2 \times 2^{-n}$$

where n is the number of bits used for the depth aspect when using a fixed-point attachment, or the number of mantissa bits plus one when using a floating-point attachment.

Otherwise for floating-point depth attachment, there is no single minimum resolvable difference. In this case, the minimum resolvable difference for a given polygon is dependent on the maximum exponent, e, in the range of z values spanned by the primitive. If n is the number of bits in the floating-point mantissa, the minimum resolvable difference, r, for the given primitive is defined as

$$r = 2^{e-n}$$

If a triangle is rasterized using the `VK_POLYGON_MODE_FILL_RECTANGLE_NV` polygon mode, then this minimum resolvable difference **may** not be resolvable for samples outside of the triangle, where the depth is extrapolated.

If no depth attachment is present, r is undefined.

The bias value o for a polygon is

$$o = \mathrm{dbclamp}(m \times depthBiasSlopeFactor + r \times depthBiasConstantFactor)$$

$$\text{where} \quad \mathrm{dbclamp}(x) = \begin{cases} x & depthBiasClamp = 0 \text{ or } NaN \\ \min(x, depthBiasClamp) & depthBiasClamp > 0 \\ \max(x, depthBiasClamp) & depthBiasClamp < 0 \end{cases}$$

m is computed as described above. If the depth attachment uses a fixed-point representation, m is a function of depth values in the range [0,1], and o is applied to depth values in the same range.

Depth bias is applied to triangle topology primitives received by the rasterizer regardless of polygon mode. Depth bias **may** also be applied to line and point topology primitives received by the rasterizer.

To dynamically set the depth bias parameters, call:

```
// Provided by VK_VERSION_1_0
void vkCmdSetDepthBias(
    VkCommandBuffer                             commandBuffer,
    float                                       depthBiasConstantFactor,
    float                                       depthBiasClamp,
    float                                       depthBiasSlopeFactor);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `depthBiasConstantFactor` is a scalar factor controlling the constant depth value added to each fragment.
- `depthBiasClamp` is the maximum (or minimum) depth bias of a fragment.
- `depthBiasSlopeFactor` is a scalar factor applied to a fragment's slope in depth bias calculations.

This command sets the depth bias parameters for subsequent drawing commands when drawing using shader objects, or when the graphics pipeline is created with `VK_DYNAMIC_STATE_DEPTH_BIAS` set in VkPipelineDynamicStateCreateInfo::`pDynamicStates`. Otherwise, this state is specified by the corresponding VkPipelineRasterizationStateCreateInfo::`depthBiasConstantFactor`, `depthBiasClamp`, and `depthBiasSlopeFactor` values used to create the currently active pipeline.

Calling this function is equivalent to calling `vkCmdSetDepthBias2EXT` without a `VkDepthBiasRepresentationInfoEXT` in the pNext chain of `VkDepthBiasInfoEXT`.

### Valid Usage

- VUID-vkCmdSetDepthBias-depthBiasClamp-00790
  If the `depthBiasClamp` feature is not enabled, `depthBiasClamp` **must** be `0.0`

## Valid Usage (Implicit)

- VUID-vkCmdSetDepthBias-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdSetDepthBias-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdSetDepthBias-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- VUID-vkCmdSetDepthBias-videocoding
  This command **must** only be called outside of a video coding scope

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics | State |

The `VkDepthBiasRepresentationInfoEXT` structure is defined as:

```
// Provided by VK_EXT_depth_bias_control
typedef struct VkDepthBiasRepresentationInfoEXT {
    VkStructureType                 sType;
    const void*                     pNext;
    VkDepthBiasRepresentationEXT    depthBiasRepresentation;
    VkBool32                        depthBiasExact;
} VkDepthBiasRepresentationInfoEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `depthBiasRepresentation` is a VkDepthBiasRepresentationEXT value specifying the depth bias representation.
- `depthBiasExact` specifies that the implementation is not allowed to scale the depth bias value to

ensure a minimum resolvable distance.

### Valid Usage

- VUID-VkDepthBiasRepresentationInfoEXT-leastRepresentableValueForceUnormRepresentation-08947
  If the `leastRepresentableValueForceUnormRepresentation` feature is not enabled, `depthBiasRepresentation` **must** not be `VK_DEPTH_BIAS_REPRESENTATION_LEAST_REPRESENTABLE_VALUE_FORCE_UNORM_EXT`
- VUID-VkDepthBiasRepresentationInfoEXT-floatRepresentation-08948
  If the `floatRepresentation` feature is not enabled, `depthBiasRepresentation` **must** not be `VK_DEPTH_BIAS_REPRESENTATION_FLOAT_EXT`
- VUID-VkDepthBiasRepresentationInfoEXT-depthBiasExact-08949
  If the `depthBiasExact` feature is not enabled, `depthBiasExact` **must** be `VK_FALSE`

### Valid Usage (Implicit)

- VUID-VkDepthBiasRepresentationInfoEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_DEPTH_BIAS_REPRESENTATION_INFO_EXT`
- VUID-VkDepthBiasRepresentationInfoEXT-depthBiasRepresentation-parameter
  `depthBiasRepresentation` **must** be a valid VkDepthBiasRepresentationEXT value

Possible values of VkDepthBiasRepresentationInfoEXT::`depthBiasRepresentation`, specifying the depth bias representation are:

```
// Provided by VK_EXT_depth_bias_control
typedef enum VkDepthBiasRepresentationEXT {
    VK_DEPTH_BIAS_REPRESENTATION_LEAST_REPRESENTABLE_VALUE_FORMAT_EXT = 0,
    VK_DEPTH_BIAS_REPRESENTATION_LEAST_REPRESENTABLE_VALUE_FORCE_UNORM_EXT = 1,
    VK_DEPTH_BIAS_REPRESENTATION_FLOAT_EXT = 2,
} VkDepthBiasRepresentationEXT;
```

- `VK_DEPTH_BIAS_REPRESENTATION_LEAST_REPRESENTABLE_VALUE_FORMAT_EXT` specifies that the depth bias representation is a factor of the format's r as described in Depth Bias Computation.
- `VK_DEPTH_BIAS_REPRESENTATION_LEAST_REPRESENTABLE_VALUE_FORCE_UNORM_EXT` specifies that the depth bias representation is a factor of a constant r defined by the bit-size or mantissa of the format as described in Depth Bias Computation.
- `VK_DEPTH_BIAS_REPRESENTATION_FLOAT_EXT` specifies that the depth bias representation is a factor of constant r equal to 1.

The `VkDepthBiasInfoEXT` structure is defined as:

```
// Provided by VK_EXT_depth_bias_control
typedef struct VkDepthBiasInfoEXT {
    VkStructureType    sType;
    const void*        pNext;
    float              depthBiasConstantFactor;
    float              depthBiasClamp;
    float              depthBiasSlopeFactor;
} VkDepthBiasInfoEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `depthBiasConstantFactor` is a scalar factor controlling the constant depth value added to each fragment.
- `depthBiasClamp` is the maximum (or minimum) depth bias of a fragment.
- `depthBiasSlopeFactor` is a scalar factor applied to a fragment's slope in depth bias calculations.

If `pNext` does not contain a VkDepthBiasRepresentationInfoEXT structure, then this command is equivalent to including a VkDepthBiasRepresentationInfoEXT with `depthBiasExact` set to `VK_FALSE` and `depthBiasRepresentation` set to `VK_DEPTH_BIAS_REPRESENTATION_LEAST_REPRESENTABLE_VALUE_FORMAT_EXT`.

### Valid Usage

- VUID-VkDepthBiasInfoEXT-depthBiasClamp-08950
  If the `depthBiasClamp` feature is not enabled, `depthBiasClamp` **must** be `0.0`

### Valid Usage (Implicit)

- VUID-VkDepthBiasInfoEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_DEPTH_BIAS_INFO_EXT`
- VUID-VkDepthBiasInfoEXT-pNext-pNext
  `pNext` **must** be `NULL` or a pointer to a valid instance of VkDepthBiasRepresentationInfoEXT
- VUID-VkDepthBiasInfoEXT-sType-unique
  The `sType` value of each struct in the `pNext` chain **must** be unique

To dynamically set the depth bias parameters, call:

```
// Provided by VK_EXT_depth_bias_control
void vkCmdSetDepthBias2EXT(
    VkCommandBuffer                             commandBuffer,
    const VkDepthBiasInfoEXT*                   pDepthBiasInfo);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `pDepthBiasInfo` is a pointer to a VkDepthBiasInfoEXT structure specifying depth bias parameters.

This command is functionally identical to vkCmdSetDepthBias, but includes extensible sub-structures that include `sType` and `pNext` parameters, allowing them to be more easily extended.

### Valid Usage (Implicit)

- VUID-vkCmdSetDepthBias2EXT-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdSetDepthBias2EXT-pDepthBiasInfo-parameter
  `pDepthBiasInfo` **must** be a valid pointer to a valid VkDepthBiasInfoEXT structure
- VUID-vkCmdSetDepthBias2EXT-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdSetDepthBias2EXT-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- VUID-vkCmdSetDepthBias2EXT-videocoding
  This command **must** only be called outside of a video coding scope

### Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

### Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics | State |

## 28.12.4. Conservative Rasterization

If the `pNext` chain of VkPipelineRasterizationStateCreateInfo includes a `VkPipelineRasterizationConservativeStateCreateInfoEXT` structure, then that structure includes parameters controlling conservative rasterization.

`VkPipelineRasterizationConservativeStateCreateInfoEXT` is defined as:

```
// Provided by VK_EXT_conservative_rasterization
typedef struct VkPipelineRasterizationConservativeStateCreateInfoEXT {
    VkStructureType                                             sType;
    const void*                                                 pNext;
    VkPipelineRasterizationConservativeStateCreateFlagsEXT      flags;
    VkConservativeRasterizationModeEXT
conservativeRasterizationMode;
    float
extraPrimitiveOverestimationSize;
} VkPipelineRasterizationConservativeStateCreateInfoEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is reserved for future use.
- `conservativeRasterizationMode` is the conservative rasterization mode to use.
- `extraPrimitiveOverestimationSize` is the extra size in pixels to increase the generating primitive during conservative rasterization at each of its edges in `X` and `Y` equally in screen space beyond the base overestimation specified in `VkPhysicalDeviceConservativeRasterizationPropertiesEXT::primitiveOverestimationSize`. If `conservativeRasterizationMode` is not `VK_CONSERVATIVE_RASTERIZATION_MODE_OVERESTIMATE_EXT`, this value is ignored.

If this structure is not included in the `pNext` chain, `conservativeRasterizationMode` is considered to be `VK_CONSERVATIVE_RASTERIZATION_MODE_DISABLED_EXT`, and conservative rasterization is disabled.

Polygon rasterization **can** be made conservative by setting `conservativeRasterizationMode` to `VK_CONSERVATIVE_RASTERIZATION_MODE_OVERESTIMATE_EXT` or `VK_CONSERVATIVE_RASTERIZATION_MODE_UNDERESTIMATE_EXT` in `VkPipelineRasterizationConservativeStateCreateInfoEXT`.

| | |
|---|---|
| **NOTE** | If `conservativePointAndLineRasterization` is supported, conservative rasterization can be applied to line and point primitives, otherwise it must be disabled. |

## Valid Usage

- VUID-VkPipelineRasterizationConservativeStateCreateInfoEXT-extraPrimitiveOverestimationSize-01769
  `extraPrimitiveOverestimationSize` **must** be in the range of `0.0` to `VkPhysicalDeviceConservativeRasterizationPropertiesEXT::maxExtraPrimitiveOverestimationSize` inclusive

## Valid Usage (Implicit)

- VUID-VkPipelineRasterizationConservativeStateCreateInfoEXT-sType-sType
  `sType` **must** be

VK_STRUCTURE_TYPE_PIPELINE_RASTERIZATION_CONSERVATIVE_STATE_CREATE_INFO_EXT

- VUID-VkPipelineRasterizationConservativeStateCreateInfoEXT-flags-zerobitmask flags **must** be 0
- VUID-VkPipelineRasterizationConservativeStateCreateInfoEXT-conservativeRasterizationMode-parameter conservativeRasterizationMode **must** be a valid VkConservativeRasterizationModeEXT value

```
// Provided by VK_EXT_conservative_rasterization
typedef VkFlags VkPipelineRasterizationConservativeStateCreateFlagsEXT;
```

VkPipelineRasterizationConservativeStateCreateFlagsEXT is a bitmask type for setting a mask, but is currently reserved for future use.

Possible values of VkPipelineRasterizationConservativeStateCreateInfoEXT::conservativeRasterizationMode, specifying the conservative rasterization mode are:

```
// Provided by VK_EXT_conservative_rasterization
typedef enum VkConservativeRasterizationModeEXT {
    VK_CONSERVATIVE_RASTERIZATION_MODE_DISABLED_EXT = 0,
    VK_CONSERVATIVE_RASTERIZATION_MODE_OVERESTIMATE_EXT = 1,
    VK_CONSERVATIVE_RASTERIZATION_MODE_UNDERESTIMATE_EXT = 2,
} VkConservativeRasterizationModeEXT;
```

- VK_CONSERVATIVE_RASTERIZATION_MODE_DISABLED_EXT specifies that conservative rasterization is disabled and rasterization proceeds as normal.
- VK_CONSERVATIVE_RASTERIZATION_MODE_OVERESTIMATE_EXT specifies that conservative rasterization is enabled in overestimation mode.
- VK_CONSERVATIVE_RASTERIZATION_MODE_UNDERESTIMATE_EXT specifies that conservative rasterization is enabled in underestimation mode.

To dynamically set the conservativeRasterizationMode, call:

```
// Provided by VK_EXT_conservative_rasterization with VK_EXT_extended_dynamic_state3,
VK_EXT_conservative_rasterization with VK_EXT_shader_object
void vkCmdSetConservativeRasterizationModeEXT(
    VkCommandBuffer                             commandBuffer,
    VkConservativeRasterizationModeEXT          conservativeRasterizationMode);
```

- commandBuffer is the command buffer into which the command will be recorded.
- conservativeRasterizationMode specifies the conservativeRasterizationMode state.

This command sets the conservativeRasterizationMode state for subsequent drawing commands when drawing using shader objects, or when the graphics pipeline is created with

VK_DYNAMIC_STATE_CONSERVATIVE_RASTERIZATION_MODE_EXT set in VkPipelineDynamicStateCreateInfo::pDynamicStates. Otherwise, this state is specified by the VkPipelineRasterizationConservativeStateCreateInfoEXT::conservativeRasterizationMode value used to create the currently active pipeline.

## Valid Usage

- VUID-vkCmdSetConservativeRasterizationModeEXT-None-09423
  At least one of the following **must** be true:
  - The extendedDynamicState3ConservativeRasterizationMode feature is enabled
  - The shaderObject feature is enabled

## Valid Usage (Implicit)

- VUID-vkCmdSetConservativeRasterizationModeEXT-commandBuffer-parameter
  commandBuffer **must** be a valid VkCommandBuffer handle
- VUID-vkCmdSetConservativeRasterizationModeEXT-conservativeRasterizationMode-parameter
  conservativeRasterizationMode **must** be a valid VkConservativeRasterizationModeEXT value
- VUID-vkCmdSetConservativeRasterizationModeEXT-commandBuffer-recording
  commandBuffer **must** be in the recording state
- VUID-vkCmdSetConservativeRasterizationModeEXT-commandBuffer-cmdpool
  The VkCommandPool that commandBuffer was allocated from **must** support graphics operations
- VUID-vkCmdSetConservativeRasterizationModeEXT-videocoding
  This command **must** only be called outside of a video coding scope

## Host Synchronization

- Host access to commandBuffer **must** be externally synchronized
- Host access to the VkCommandPool that commandBuffer was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics | State |

To dynamically set the extraPrimitiveOverestimationSize, call:

```
// Provided by VK_EXT_conservative_rasterization with VK_EXT_extended_dynamic_state3,
VK_EXT_conservative_rasterization with VK_EXT_shader_object
void vkCmdSetExtraPrimitiveOverestimationSizeEXT(
    VkCommandBuffer                             commandBuffer,
    float                                       extraPrimitiveOverestimationSize);
```

- commandBuffer is the command buffer into which the command will be recorded.
- extraPrimitiveOverestimationSize specifies the extraPrimitiveOverestimationSize.

This command sets the extraPrimitiveOverestimationSize for subsequent drawing commands when drawing using shader objects, or when the graphics pipeline is created with VK_DYNAMIC_STATE_EXTRA_PRIMITIVE_OVERESTIMATION_SIZE_EXT set in VkPipelineDynamicStateCreateInfo::pDynamicStates. Otherwise, this state is specified by the VkPipelineRasterizationConservativeStateCreateInfoEXT::extraPrimitiveOverestimationSize value used to create the currently active pipeline.

## Valid Usage

- VUID-vkCmdSetExtraPrimitiveOverestimationSizeEXT-None-09423
  At least one of the following **must** be true:
  - The extendedDynamicState3ExtraPrimitiveOverestimationSize feature is enabled
  - The shaderObject feature is enabled
- VUID-vkCmdSetExtraPrimitiveOverestimationSizeEXT-extraPrimitiveOverestimationSize-07428
  extraPrimitiveOverestimationSize **must** be in the range of 0.0 to VkPhysicalDeviceConservativeRasterizationPropertiesEXT::maxExtraPrimitiveOverestimationSize inclusive

## Valid Usage (Implicit)

- VUID-vkCmdSetExtraPrimitiveOverestimationSizeEXT-commandBuffer-parameter
  commandBuffer **must** be a valid VkCommandBuffer handle
- VUID-vkCmdSetExtraPrimitiveOverestimationSizeEXT-commandBuffer-recording
  commandBuffer **must** be in the recording state
- VUID-vkCmdSetExtraPrimitiveOverestimationSizeEXT-commandBuffer-cmdpool
  The VkCommandPool that commandBuffer was allocated from **must** support graphics operations
- VUID-vkCmdSetExtraPrimitiveOverestimationSizeEXT-videocoding
  This command **must** only be called outside of a video coding scope

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics | State |

When overestimate conservative rasterization is enabled, rather than evaluating coverage at individual sample locations, a determination is made whether any portion of the pixel (including its edges and corners) is covered by the primitive. If any portion of the pixel is covered, then all bits of the coverage mask for the fragment corresponding to that pixel are enabled. If the render pass has a fragment density map attachment and any bit of the coverage mask for the fragment is enabled, then all bits of the coverage mask for the fragment are enabled.

For the purposes of evaluating which pixels are covered by the primitive, implementations **can** increase the size of the primitive by up to `VkPhysicalDeviceConservativeRasterizationPropertiesEXT::primitiveOverestimationSize` pixels at each of the primitive edges. This **may** increase the number of fragments generated by this primitive and represents an overestimation of the pixel coverage.

This overestimation size can be increased further by setting the `extraPrimitiveOverestimationSize` value above `0.0` in steps of `VkPhysicalDeviceConservativeRasterizationPropertiesEXT::extraPrimitiveOverestimationSizeGranularity` up to and including `VkPhysicalDeviceConservativeRasterizationPropertiesEXT::extraPrimitiveOverestimationSize`. This **may** further increase the number of fragments generated by this primitive.

The actual precision of the overestimation size used for conservative rasterization **may** vary between implementations and produce results that only approximate the `primitiveOverestimationSize` and `extraPrimitiveOverestimationSizeGranularity` properties. Implementations **may** especially vary these approximations when the render pass has a fragment density map and the fragment area covers multiple pixels.

For triangles if `VK_CONSERVATIVE_RASTERIZATION_MODE_OVERESTIMATE_EXT` is enabled, fragments will be generated if the primitive area covers any portion of any pixel inside the fragment area, including their edges or corners. The tie-breaking rule described in Basic Polygon Rasterization does not apply during conservative rasterization and coverage is set for all fragments generated from shared edges of polygons. Degenerate triangles that evaluate to zero area after rasterization, even for pixels containing a vertex or edge of the zero-area polygon, will be culled if `VkPhysicalDeviceConservativeRasterizationPropertiesEXT::degenerateTrianglesRasterized` is `VK_FALSE` or will generate fragments if `degenerateTrianglesRasterized` is `VK_TRUE`. The fragment input values

for these degenerate triangles take their attribute and depth values from the provoking vertex. Degenerate triangles are considered backfacing and the application **can** enable backface culling if desired. Triangles that are zero area before rasterization **may** be culled regardless.

For lines if VK_CONSERVATIVE_RASTERIZATION_MODE_OVERESTIMATE_EXT is enabled, and the implementation sets VkPhysicalDeviceConservativeRasterizationPropertiesEXT::conservativePointAndLineRasterization to VK_TRUE, fragments will be generated if the line covers any portion of any pixel inside the fragment area, including their edges or corners. Degenerate lines that evaluate to zero length after rasterization will be culled if VkPhysicalDeviceConservativeRasterizationPropertiesEXT::degenerateLinesRasterized is VK_FALSE or will generate fragments if degenerateLinesRasterized is VK_TRUE. The fragments input values for these degenerate lines take their attribute and depth values from the provoking vertex. Lines that are zero length before rasterization **may** be culled regardless.

For points if VK_CONSERVATIVE_RASTERIZATION_MODE_OVERESTIMATE_EXT is enabled, and the implementation sets VkPhysicalDeviceConservativeRasterizationPropertiesEXT::conservativePointAndLineRasterization to VK_TRUE, fragments will be generated if the point square covers any portion of any pixel inside the fragment area, including their edges or corners.

When underestimate conservative rasterization is enabled, rather than evaluating coverage at individual sample locations, a determination is made whether all of the pixel (including its edges and corners) is covered by the primitive. If the entire pixel is covered, then a fragment is generated with all bits of its coverage mask corresponding to the pixel enabled, otherwise the pixel is not considered covered even if some portion of the pixel is covered. The fragment is discarded if no pixels inside the fragment area are considered covered. If the render pass has a fragment density map attachment and any pixel inside the fragment area is not considered covered, then the fragment is discarded even if some pixels are considered covered.

For triangles, if VK_CONSERVATIVE_RASTERIZATION_MODE_UNDERESTIMATE_EXT is enabled, fragments will only be generated if any pixel inside the fragment area is fully covered by the generating primitive, including its edges and corners.

For lines, if VK_CONSERVATIVE_RASTERIZATION_MODE_UNDERESTIMATE_EXT is enabled, fragments will be generated if any pixel inside the fragment area, including its edges and corners, are entirely covered by the line.

For points, if VK_CONSERVATIVE_RASTERIZATION_MODE_UNDERESTIMATE_EXT is enabled, fragments will only be generated if the point square covers the entirety of any pixel square inside the fragment area, including its edges or corners.

If the render pass has a fragment density map and VK_CONSERVATIVE_RASTERIZATION_MODE_UNDERESTIMATE_EXT is enabled, fragments will only be generated if the entirety of all pixels inside the fragment area are covered by the generating primitive, line, or point.

For both overestimate and underestimate conservative rasterization modes a fragment has all of its pixel squares fully covered by the generating primitive **must** set FullyCoveredEXT to VK_TRUE if the implementation enables the VkPhysicalDeviceConservativeRasterizationPropertiesEXT::fullyCoveredFragmentShaderInputVariable feature.

When the use of a shading rate image or setting the fragment shading rate results in fragments covering multiple pixels, coverage for conservative rasterization is still evaluated on a per-pixel basis and may result in fragments with partial coverage. For fragment shader inputs decorated with `FullyCoveredEXT`, a fragment is considered fully covered if and only if all pixels in the fragment are fully covered by the generating primitive.

# Chapter 29. Fragment Operations

Fragments produced by rasterization go through a number of operations to determine whether or how values produced by fragment shading are written to the framebuffer.

The following fragment operations adhere to rasterization order, and are typically performed in this order:

1. Discard rectangles test
2. Scissor test
3. Exclusive scissor test
4. Sample mask test
5. Certain Fragment shading operations:
    - Sample Mask Accesses
    - Tile Image Reads
    - Depth Replacement
    - Stencil Reference Replacement
    - Interlocked Operations
6. Multisample coverage
7. Depth bounds test
8. Stencil test
9. Depth test
10. Representative fragment test
11. Sample counting
12. Coverage to color
13. Coverage reduction
14. Coverage modulation

The coverage mask generated by rasterization describes the initial coverage of each sample covered by the fragment. Fragment operations will update the coverage mask to add or subtract coverage where appropriate. If a fragment operation results in all bits of the coverage mask being `0`, the fragment is discarded, and no further operations are performed. Fragments can also be programmatically discarded in a fragment shader by executing one of

- `OpTerminateInvocation`
- `OpDemoteToHelperInvocationEXT`
- `OpKill`.

When one of the fragment operations in this chapter is described as “replacing” a fragment shader output, that output is replaced unconditionally, even if no fragment shader previously wrote to that output.

If there is a fragment shader and it declares the PostDepthCoverage execution mode, the sample mask test is instead performed after the depth test.

If VkPhysicalDeviceMaintenance5PropertiesKHR::earlyFragmentMultisampleCoverageAfterSampleCounting is set to VK_TRUE and there is a fragment shader which declares the EarlyFragmentTests execution mode, fragment shading and multisample coverage operations **must** be performed after sample counting.

Otherwise, if VkPhysicalDeviceMaintenance5PropertiesKHR::earlyFragmentMultisampleCoverageAfterSampleCounting is set to VK_FALSE and there is a fragment shader which declares the EarlyFragmentTests execution mode, fragment shading and multisample coverage operations **should** instead be performed after sample counting, but **may** be performed before sample counting.

If VkPhysicalDeviceMaintenance5PropertiesKHR::earlyFragmentSampleMaskTestBeforeSampleCounting is set to VK_TRUE and there is a fragment shader which declares the EarlyFragmentTests execution mode sample mask test operations **must** follow the order of fragment operations from above.

Otherwise, if VkPhysicalDeviceMaintenance5PropertiesKHR::earlyFragmentSampleMaskTestBeforeSampleCounting is set to VK_FALSE and there is a fragment shader which declares the EarlyFragmentTests execution mode, sample mask test operations **should** follow the order of fragment operations from above but **may** instead be performed after sample counting.

If there is a fragment shader which declares the EarlyAndLateFragmentTestsAMD execution mode, and it does not declare the DepthReplacing or StencilRefReplacingEXT execution mode, fragment shading and multisample coverage operations are instead be performed after sample counting.

For a pipeline with the following properties:

- a fragment shader is specified
- the fragment shader either specifies EarlyAndLateFragmentTestsAMD or does not write to storage resources;
- the fragment shader specifies the StencilRefReplacingEXT execution mode;
- either
  - the fragment shader specifies the StencilRefUnchangedFrontAMD execution mode;
  - the fragment shader specifies the StencilRefLessFrontAMD execution mode and the pipeline uses a VkPipelineDepthStencilStateCreateInfo::front.compareOp of VK_COMPARE_OP_GREATER or VK_COMPARE_OP_GREATER_OR_EQUAL; or
  - the fragment shader specifies the StencilRefGreaterFrontAMD execution mode and the pipeline uses a VkPipelineDepthStencilStateCreateInfo::front.compareOp of VK_COMPARE_OP_LESS or VK_COMPARE_OP_LESS_OR_EQUAL; and
- either
  - the fragment shader specifies the StencilRefUnchangedBackAMD execution mode;
  - the fragment shader specifies the StencilRefLessBackAMD execution mode and the pipeline uses a VkPipelineDepthStencilStateCreateInfo::back.compareOp of VK_COMPARE_OP_GREATER or VK_COMPARE_OP_GREATER_OR_EQUAL; or

- the fragment shader specifies the `StencilRefGreaterBackAMD` execution mode and the pipeline uses a VkPipelineDepthStencilStateCreateInfo::`back.compareOp` of `VK_COMPARE_OP_LESS` or `VK_COMPARE_OP_LESS_OR_EQUAL`

an additional stencil test **may** be performed before fragment shading, using the stencil reference value specified by VkPipelineDepthStencilStateCreateInfo::`front.reference` or VkPipelineDepthStencilStateCreateInfo::`back.reference`.

For a pipeline with the following properties:

- a fragment shader is specified
- the fragment shader either specifies `EarlyAndLateFragmentTestsAMD` or does not write to storage resources;
- the fragment shader specifies the `DepthReplacing` execution mode; and
- either
  - the fragment shader specifies the `DepthUnchanged` execution mode;
  - the fragment shader specifies the `DepthLess` execution mode and the pipeline uses a VkPipelineDepthStencilStateCreateInfo::`depthCompareOp` of `VK_COMPARE_OP_GREATER` or `VK_COMPARE_OP_GREATER_OR_EQUAL`; or
  - the fragment shader specifies the `DepthGreater` execution mode and the pipeline uses a VkPipelineDepthStencilStateCreateInfo::`depthCompareOp` of `VK_COMPARE_OP_LESS` or `VK_COMPARE_OP_LESS_OR_EQUAL`

the implementation **may** perform depth bounds test before fragment shading and perform an additional depth test immediately after that using the interpolated depth value generated by rasterization.

Once all fragment operations have completed, fragment shader outputs for covered color attachment samples pass through framebuffer operations.

## 29.1. Discard Rectangles Test

The discard rectangle test compares the framebuffer coordinates $(x_f,y_f)$ of each sample covered by a fragment against a set of *discard rectangles*.

Each discard rectangle is defined by a VkRect2D. These values are either set by the VkPipelineDiscardRectangleStateCreateInfoEXT structure during pipeline creation, or dynamically by the vkCmdSetDiscardRectangleEXT command.

A given sample is considered inside a discard rectangle if the $x_f$ is in the range [VkRect2D::`offset.x`, VkRect2D::`offset.x` + VkRect2D::`extent.x`), and $y_f$ is in the range [VkRect2D::`offset.y`, VkRect2D::`offset.y` + VkRect2D::`extent.y`). If the test is set to be inclusive, samples that are not inside any of the discard rectangles will have their coverage set to `0`. If the test is set to be exclusive, samples that are inside any of the discard rectangles will have their coverage set to `0`.

If no discard rectangles are specified, the coverage mask is unmodified by this operation.

The VkPipelineDiscardRectangleStateCreateInfoEXT structure is defined as:

```
// Provided by VK_EXT_discard_rectangles
typedef struct VkPipelineDiscardRectangleStateCreateInfoEXT {
    VkStructureType                                     sType;
    const void*                                         pNext;
    VkPipelineDiscardRectangleStateCreateFlagsEXT       flags;
    VkDiscardRectangleModeEXT                           discardRectangleMode;
    uint32_t                                            discardRectangleCount;
    const VkRect2D*                                     pDiscardRectangles;
} VkPipelineDiscardRectangleStateCreateInfoEXT;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- flags is reserved for future use.
- discardRectangleMode is a VkDiscardRectangleModeEXT value determining whether the discard rectangle test is inclusive or exclusive.
- discardRectangleCount is the number of discard rectangles to use.
- pDiscardRectangles is a pointer to an array of VkRect2D structures defining discard rectangles.

If the VK_DYNAMIC_STATE_DISCARD_RECTANGLE_EXT dynamic state is enabled for a pipeline, the pDiscardRectangles member is ignored. If the VK_DYNAMIC_STATE_DISCARD_RECTANGLE_ENABLE_EXT dynamic state is not enabled for the pipeline the presence of this structure in the VkGraphicsPipelineCreateInfo chain, and a discardRectangleCount greater than zero, implicitly enables discard rectangles in the pipeline, otherwise discard rectangles **must** enabled or disabled by vkCmdSetDiscardRectangleEnableEXT. If the VK_DYNAMIC_STATE_DISCARD_RECTANGLE_MODE_EXT dynamic state is enabled for the pipeline, the discardRectangleMode member is ignored, and the discard rectangle mode **must** be set by vkCmdSetDiscardRectangleModeEXT.

When this structure is included in the pNext chain of VkGraphicsPipelineCreateInfo, it defines parameters of the discard rectangle test. If the VK_DYNAMIC_STATE_DISCARD_RECTANGLE_EXT dynamic state is not enabled, and this structure is not included in the pNext chain, it is equivalent to specifying this structure with a discardRectangleCount of 0.

### Valid Usage

- VUID-VkPipelineDiscardRectangleStateCreateInfoEXT-discardRectangleCount-00582
  discardRectangleCount **must** be less than or equal to VkPhysicalDeviceDiscardRectanglePropertiesEXT::maxDiscardRectangles

### Valid Usage (Implicit)

- VUID-VkPipelineDiscardRectangleStateCreateInfoEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PIPELINE_DISCARD_RECTANGLE_STATE_CREATE_INFO_EXT

- VUID-VkPipelineDiscardRectangleStateCreateInfoEXT-flags-zerobitmask
  `flags` **must** be `0`
- VUID-VkPipelineDiscardRectangleStateCreateInfoEXT-discardRectangleMode-parameter
  `discardRectangleMode` **must** be a valid VkDiscardRectangleModeEXT value

```
// Provided by VK_EXT_discard_rectangles
typedef VkFlags VkPipelineDiscardRectangleStateCreateFlagsEXT;
```

`VkPipelineDiscardRectangleStateCreateFlagsEXT` is a bitmask type for setting a mask, but is currently reserved for future use.

`VkDiscardRectangleModeEXT` values are:

```
// Provided by VK_EXT_discard_rectangles
typedef enum VkDiscardRectangleModeEXT {
    VK_DISCARD_RECTANGLE_MODE_INCLUSIVE_EXT = 0,
    VK_DISCARD_RECTANGLE_MODE_EXCLUSIVE_EXT = 1,
} VkDiscardRectangleModeEXT;
```

- `VK_DISCARD_RECTANGLE_MODE_INCLUSIVE_EXT` specifies that the discard rectangle test is inclusive.
- `VK_DISCARD_RECTANGLE_MODE_EXCLUSIVE_EXT` specifies that the discard rectangle test is exclusive.

To dynamically set the discard rectangles, call:

```
// Provided by VK_EXT_discard_rectangles
void vkCmdSetDiscardRectangleEXT(
    VkCommandBuffer                             commandBuffer,
    uint32_t                                    firstDiscardRectangle,
    uint32_t                                    discardRectangleCount,
    const VkRect2D*                             pDiscardRectangles);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `firstDiscardRectangle` is the index of the first discard rectangle whose state is updated by the command.
- `discardRectangleCount` is the number of discard rectangles whose state are updated by the command.
- `pDiscardRectangles` is a pointer to an array of VkRect2D structures specifying discard rectangles.

The discard rectangle taken from element i of `pDiscardRectangles` replace the current state for the discard rectangle at index `firstDiscardRectangle` + i, for i in [0, `discardRectangleCount`).

This command sets the discard rectangles for subsequent drawing commands when drawing using shader objects, or when the graphics pipeline is created with `VK_DYNAMIC_STATE_DISCARD_RECTANGLE_EXT` set in VkPipelineDynamicStateCreateInfo::`pDynamicStates`.

Otherwise, this state is specified by the VkPipelineDiscardRectangleStateCreateInfoEXT::pDiscardRectangles values used to create the currently active pipeline.

## Valid Usage

- VUID-vkCmdSetDiscardRectangleEXT-firstDiscardRectangle-00585
  The sum of firstDiscardRectangle and discardRectangleCount **must** be less than or equal to VkPhysicalDeviceDiscardRectanglePropertiesEXT::maxDiscardRectangles
- VUID-vkCmdSetDiscardRectangleEXT-x-00587
  The x and y member of offset in each VkRect2D element of pDiscardRectangles **must** be greater than or equal to 0
- VUID-vkCmdSetDiscardRectangleEXT-offset-00588
  Evaluation of (offset.x + extent.width) in each VkRect2D element of pDiscardRectangles **must** not cause a signed integer addition overflow
- VUID-vkCmdSetDiscardRectangleEXT-offset-00589
  Evaluation of (offset.y + extent.height) in each VkRect2D element of pDiscardRectangles **must** not cause a signed integer addition overflow
- VUID-vkCmdSetDiscardRectangleEXT-viewportScissor2D-04788
  If this command is recorded in a secondary command buffer with VkCommandBufferInheritanceViewportScissorInfoNV::viewportScissor2D enabled, then this function **must** not be called

## Valid Usage (Implicit)

- VUID-vkCmdSetDiscardRectangleEXT-commandBuffer-parameter
  commandBuffer **must** be a valid VkCommandBuffer handle
- VUID-vkCmdSetDiscardRectangleEXT-pDiscardRectangles-parameter
  pDiscardRectangles **must** be a valid pointer to an array of discardRectangleCount VkRect2D structures
- VUID-vkCmdSetDiscardRectangleEXT-commandBuffer-recording
  commandBuffer **must** be in the recording state
- VUID-vkCmdSetDiscardRectangleEXT-commandBuffer-cmdpool
  The VkCommandPool that commandBuffer was allocated from **must** support graphics operations
- VUID-vkCmdSetDiscardRectangleEXT-videocoding
  This command **must** only be called outside of a video coding scope
- VUID-vkCmdSetDiscardRectangleEXT-discardRectangleCount-arraylength
  discardRectangleCount **must** be greater than 0

## Host Synchronization

- Host access to commandBuffer **must** be externally synchronized

- Host access to the VkCommandPool that commandBuffer was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics | State |

To dynamically set whether discard rectangles are enabled, call:

```
// Provided by VK_EXT_discard_rectangles
void vkCmdSetDiscardRectangleEnableEXT(
    VkCommandBuffer                             commandBuffer,
    VkBool32                                    discardRectangleEnable);
```

- commandBuffer is the command buffer into which the command will be recorded.
- discardRectangleEnable specifies whether discard rectangles are enabled or not.

This command sets the discard rectangle enable for subsequent drawing commands when drawing using shader objects, or when the graphics pipeline is created with VK_DYNAMIC_STATE_DISCARD_RECTANGLE_ENABLE_EXT set in VkPipelineDynamicStateCreateInfo::pDynamicStates. Otherwise, this state is implied by the VkPipelineDiscardRectangleStateCreateInfoEXT::discardRectangleCount value used to create the currently active pipeline, where a non-zero discardRectangleCount implicitly enables discard rectangles, otherwise they are disabled.

## Valid Usage

- VUID-vkCmdSetDiscardRectangleEnableEXT-specVersion-07851
  The VK_EXT_discard_rectangles extension **must** be enabled, and the implementation **must** support at least specVersion 2 of this extension

## Valid Usage (Implicit)

- VUID-vkCmdSetDiscardRectangleEnableEXT-commandBuffer-parameter
  commandBuffer **must** be a valid VkCommandBuffer handle
- VUID-vkCmdSetDiscardRectangleEnableEXT-commandBuffer-recording
  commandBuffer **must** be in the recording state
- VUID-vkCmdSetDiscardRectangleEnableEXT-commandBuffer-cmdpool
  The VkCommandPool that commandBuffer was allocated from **must** support graphics

operations

- VUID-vkCmdSetDiscardRectangleEnableEXT-videocoding
  This command **must** only be called outside of a video coding scope

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics | State |

To dynamically set the discard rectangle mode, call:

```
// Provided by VK_EXT_discard_rectangles
void vkCmdSetDiscardRectangleModeEXT(
    VkCommandBuffer                             commandBuffer,
    VkDiscardRectangleModeEXT                   discardRectangleMode);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `discardRectangleMode` specifies the discard rectangle mode for all discard rectangles, either inclusive or exclusive.

This command sets the discard rectangle mode for subsequent drawing commands when drawing using shader objects, or when the graphics pipeline is created with `VK_DYNAMIC_STATE_DISCARD_RECTANGLE_MODE_EXT` set in VkPipelineDynamicStateCreateInfo::`pDynamicStates`. Otherwise, this state is specified by the VkPipelineDiscardRectangleStateCreateInfoEXT::`discardRectangleMode` value used to create the currently active pipeline.

## Valid Usage

- VUID-vkCmdSetDiscardRectangleModeEXT-specVersion-07852
  The `VK_EXT_discard_rectangles` extension **must** be enabled, and the implementation **must** support at least `specVersion` `2` of this extension

### Valid Usage (Implicit)

- VUID-vkCmdSetDiscardRectangleModeEXT-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdSetDiscardRectangleModeEXT-discardRectangleMode-parameter
  `discardRectangleMode` **must** be a valid VkDiscardRectangleModeEXT value
- VUID-vkCmdSetDiscardRectangleModeEXT-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdSetDiscardRectangleModeEXT-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- VUID-vkCmdSetDiscardRectangleModeEXT-videocoding
  This command **must** only be called outside of a video coding scope

### Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

### Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics | State |

## 29.2. Scissor Test

The scissor test compares the framebuffer coordinates $(x_f,y_f)$ of each sample covered by a fragment against a *scissor rectangle* at the index equal to the fragment's `ViewportIndex`.

Each scissor rectangle is defined by a VkRect2D. These values are either set by the VkPipelineViewportStateCreateInfo structure during pipeline creation, or dynamically by the vkCmdSetScissor command.

A given sample is considered inside a scissor rectangle if $x_f$ is in the range [VkRect2D::`offset.x`, VkRect2D::`offset.x` + VkRect2D::`extent.x`), and $y_f$ is in the range [VkRect2D::`offset.y`, VkRect2D::`offset.y` + VkRect2D::`extent.y`). Samples with coordinates outside the scissor rectangle at the corresponding `ViewportIndex` will have their coverage set to `0`.

If a render pass transform is enabled, the (`offset.x` and `offset.y`) and (`extent.width` and

extent.height) values are transformed as described in render pass transform before participating in the scissor test.

To dynamically set the scissor rectangles, call:

```
// Provided by VK_VERSION_1_0
void vkCmdSetScissor(
    VkCommandBuffer                             commandBuffer,
    uint32_t                                    firstScissor,
    uint32_t                                    scissorCount,
    const VkRect2D*                             pScissors);
```

- commandBuffer is the command buffer into which the command will be recorded.
- firstScissor is the index of the first scissor whose state is updated by the command.
- scissorCount is the number of scissors whose rectangles are updated by the command.
- pScissors is a pointer to an array of VkRect2D structures defining scissor rectangles.

The scissor rectangles taken from element i of pScissors replace the current state for the scissor index firstScissor + i, for i in [0, scissorCount).

This command sets the scissor rectangles for subsequent drawing commands when drawing using shader objects, or when the graphics pipeline is created with VK_DYNAMIC_STATE_SCISSOR set in VkPipelineDynamicStateCreateInfo::pDynamicStates. Otherwise, this state is specified by the VkPipelineViewportStateCreateInfo::pScissors values used to create the currently active pipeline.

### Valid Usage

- VUID-vkCmdSetScissor-firstScissor-00592
  The sum of firstScissor and scissorCount **must** be between 1 and VkPhysicalDeviceLimits::maxViewports, inclusive
- VUID-vkCmdSetScissor-firstScissor-00593
  If the multiViewport feature is not enabled, firstScissor **must** be 0
- VUID-vkCmdSetScissor-scissorCount-00594
  If the multiViewport feature is not enabled, scissorCount **must** be 1
- VUID-vkCmdSetScissor-x-00595
  The x and y members of offset member of any element of pScissors **must** be greater than or equal to 0
- VUID-vkCmdSetScissor-offset-00596
  Evaluation of (offset.x + extent.width) **must** not cause a signed integer addition overflow for any element of pScissors
- VUID-vkCmdSetScissor-offset-00597
  Evaluation of (offset.y + extent.height) **must** not cause a signed integer addition overflow for any element of pScissors
- VUID-vkCmdSetScissor-viewportScissor2D-04789

If this command is recorded in a secondary command buffer with VkCommandBufferInheritanceViewportScissorInfoNV::viewportScissor2D enabled, then this function **must** not be called

### Valid Usage (Implicit)

- VUID-vkCmdSetScissor-commandBuffer-parameter
  commandBuffer **must** be a valid VkCommandBuffer handle
- VUID-vkCmdSetScissor-pScissors-parameter
  pScissors **must** be a valid pointer to an array of scissorCount VkRect2D structures
- VUID-vkCmdSetScissor-commandBuffer-recording
  commandBuffer **must** be in the recording state
- VUID-vkCmdSetScissor-commandBuffer-cmdpool
  The VkCommandPool that commandBuffer was allocated from **must** support graphics operations
- VUID-vkCmdSetScissor-videocoding
  This command **must** only be called outside of a video coding scope
- VUID-vkCmdSetScissor-scissorCount-arraylength
  scissorCount **must** be greater than 0

### Host Synchronization

- Host access to commandBuffer **must** be externally synchronized
- Host access to the VkCommandPool that commandBuffer was allocated from **must** be externally synchronized

### Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics | State |

## 29.3. Exclusive Scissor Test

The exclusive scissor test compares the framebuffer coordinates $(x_f,y_f)$ of each sample covered by a fragment against an *exclusive scissor rectangle* at the index equal to the fragment's ViewportIndex.

Each exclusive scissor rectangle is defined by a VkRect2D. These values are either set by the VkPipelineViewportExclusiveScissorStateCreateInfoNV structure during pipeline creation, or

dynamically by the vkCmdSetExclusiveScissorNV command.

A given sample is considered inside an exclusive scissor rectangle if $x_f$ is in the range [VkRect2D::offset.x, VkRect2D::offset.x + VkRect2D::extent.x), and $y_f$ is in the range [VkRect2D::offset.y, VkRect2D::offset.y + VkRect2D::extent.y). Samples with coordinates inside the exclusive scissor rectangle at the corresponding ViewportIndex will have their coverage set to 0.

If no exclusive scissor rectangles are specified, the coverage mask is unmodified by this operation.

The VkPipelineViewportExclusiveScissorStateCreateInfoNV structure is defined as:

```
// Provided by VK_NV_scissor_exclusive
typedef struct VkPipelineViewportExclusiveScissorStateCreateInfoNV {
    VkStructureType    sType;
    const void*        pNext;
    uint32_t           exclusiveScissorCount;
    const VkRect2D*    pExclusiveScissors;
} VkPipelineViewportExclusiveScissorStateCreateInfoNV;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- exclusiveScissorCount is the number of exclusive scissor rectangles.
- pExclusiveScissors is a pointer to an array of VkRect2D structures defining exclusive scissor rectangles.

If the VK_DYNAMIC_STATE_EXCLUSIVE_SCISSOR_NV dynamic state is enabled for a pipeline, the pExclusiveScissors member is ignored.

When this structure is included in the pNext chain of VkGraphicsPipelineCreateInfo, it defines parameters of the exclusive scissor test. If this structure is not included in the pNext chain, it is equivalent to specifying this structure with an exclusiveScissorCount of 0.

### Valid Usage

- VUID-VkPipelineViewportExclusiveScissorStateCreateInfoNV-exclusiveScissorCount-02027
  If the multiViewport feature is not enabled, exclusiveScissorCount **must** be 0 or 1
- VUID-VkPipelineViewportExclusiveScissorStateCreateInfoNV-exclusiveScissorCount-02028
  exclusiveScissorCount **must** be less than or equal to VkPhysicalDeviceLimits::maxViewports
- VUID-VkPipelineViewportExclusiveScissorStateCreateInfoNV-exclusiveScissorCount-02029
  exclusiveScissorCount **must** be 0 or greater than or equal to the viewportCount member of VkPipelineViewportStateCreateInfo

### Valid Usage (Implicit)

- VUID-VkPipelineViewportExclusiveScissorStateCreateInfoNV-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PIPELINE_VIEWPORT_EXCLUSIVE_SCISSOR_STATE_CREATE_INFO_NV`

To dynamically set the exclusive scissor rectangles, call:

```
// Provided by VK_NV_scissor_exclusive
void vkCmdSetExclusiveScissorNV(
    VkCommandBuffer                             commandBuffer,
    uint32_t                                    firstExclusiveScissor,
    uint32_t                                    exclusiveScissorCount,
    const VkRect2D*                             pExclusiveScissors);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `firstExclusiveScissor` is the index of the first exclusive scissor rectangle whose state is updated by the command.
- `exclusiveScissorCount` is the number of exclusive scissor rectangles updated by the command.
- `pExclusiveScissors` is a pointer to an array of VkRect2D structures defining exclusive scissor rectangles.

The scissor rectangles taken from element i of `pExclusiveScissors` replace the current state for the scissor index `firstExclusiveScissor` + i, for i in [0, `exclusiveScissorCount`).

This command sets the exclusive scissor rectangles for subsequent drawing commands when drawing using shader objects, or when the graphics pipeline is created with `VK_DYNAMIC_STATE_EXCLUSIVE_SCISSOR_NV` set in VkPipelineDynamicStateCreateInfo::`pDynamicStates`. Otherwise, this state is specified by the VkPipelineViewportExclusiveScissorStateCreateInfoNV::`pExclusiveScissors` values used to create the currently active pipeline.

### Valid Usage

- VUID-vkCmdSetExclusiveScissorNV-None-02031
  The `exclusiveScissor` feature **must** be enabled
- VUID-vkCmdSetExclusiveScissorNV-firstExclusiveScissor-02034
  The sum of `firstExclusiveScissor` and `exclusiveScissorCount` **must** be between `1` and `VkPhysicalDeviceLimits::maxViewports`, inclusive
- VUID-vkCmdSetExclusiveScissorNV-firstExclusiveScissor-02035
  If the `multiViewport` feature is not enabled, `firstExclusiveScissor` **must** be `0`
- VUID-vkCmdSetExclusiveScissorNV-exclusiveScissorCount-02036
  If the `multiViewport` feature is not enabled, `exclusiveScissorCount` **must** be `1`
- VUID-vkCmdSetExclusiveScissorNV-x-02037

The `x` and `y` members of `offset` in each member of `pExclusiveScissors` **must** be greater than or equal to `0`

- VUID-vkCmdSetExclusiveScissorNV-offset-02038
  Evaluation of (`offset.x` + `extent.width`) for each member of `pExclusiveScissors` **must** not cause a signed integer addition overflow
- VUID-vkCmdSetExclusiveScissorNV-offset-02039
  Evaluation of (`offset.y` + `extent.height`) for each member of `pExclusiveScissors` **must** not cause a signed integer addition overflow

## Valid Usage (Implicit)

- VUID-vkCmdSetExclusiveScissorNV-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdSetExclusiveScissorNV-pExclusiveScissors-parameter
  `pExclusiveScissors` **must** be a valid pointer to an array of `exclusiveScissorCount` VkRect2D structures
- VUID-vkCmdSetExclusiveScissorNV-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdSetExclusiveScissorNV-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- VUID-vkCmdSetExclusiveScissorNV-videocoding
  This command **must** only be called outside of a video coding scope
- VUID-vkCmdSetExclusiveScissorNV-exclusiveScissorCount-arraylength
  `exclusiveScissorCount` **must** be greater than `0`

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics | State |

To dynamically set whether an exclusive scissor is enabled or not, call:

```
// Provided by VK_NV_scissor_exclusive
void vkCmdSetExclusiveScissorEnableNV(
    VkCommandBuffer                             commandBuffer,
    uint32_t                                    firstExclusiveScissor,
    uint32_t                                    exclusiveScissorCount,
    const VkBool32*                             pExclusiveScissorEnables);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `firstExclusiveScissor` is the index of the first exclusive scissor rectangle whose state is updated by the command.
- `exclusiveScissorCount` is the number of exclusive scissor rectangles updated by the command.
- `pExclusiveScissorEnables` is a pointer to an array of VkBool32 values defining whether the exclusive scissor is enabled.

The exclusive scissor enables taken from element i of `pExclusiveScissorEnables` replace the current state for the scissor index `firstExclusiveScissor` + i, for i in [0, `exclusiveScissorCount`).

This command sets the exclusive scissor enable for subsequent drawing commands when drawing using shader objects, or when the graphics pipeline is created with `VK_DYNAMIC_STATE_EXCLUSIVE_SCISSOR_ENABLE_NV` set in VkPipelineDynamicStateCreateInfo::`pDynamicStates`. Otherwise, this state is implied by the VkPipelineViewportExclusiveScissorStateCreateInfoNV::`exclusiveScissorCount` value used to create the currently active pipeline, where all `exclusiveScissorCount` exclusive scissors are implicitly enabled and the remainder up to `VkPhysicalDeviceLimits`::`maxViewports` are implicitly disabled.

## Valid Usage

- VUID-vkCmdSetExclusiveScissorEnableNV-exclusiveScissor-07853
  The `exclusiveScissor` feature **must** be enabled, and the implementation **must** support at least `specVersion` 2 of the `VK_NV_scissor_exclusive` extension

## Valid Usage (Implicit)

- VUID-vkCmdSetExclusiveScissorEnableNV-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdSetExclusiveScissorEnableNV-pExclusiveScissorEnables-parameter
  `pExclusiveScissorEnables` **must** be a valid pointer to an array of `exclusiveScissorCount` VkBool32 values
- VUID-vkCmdSetExclusiveScissorEnableNV-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdSetExclusiveScissorEnableNV-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations

- VUID-vkCmdSetExclusiveScissorEnableNV-videocoding
  This command **must** only be called outside of a video coding scope
- VUID-vkCmdSetExclusiveScissorEnableNV-exclusiveScissorCount-arraylength
  `exclusiveScissorCount` **must** be greater than `0`

### Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

### Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics | State |

## 29.4. Sample Mask Test

The sample mask test compares the coverage mask for a fragment with the *sample mask* defined by VkPipelineMultisampleStateCreateInfo::`pSampleMask`.

To dynamically set the sample mask, call:

```
// Provided by VK_EXT_extended_dynamic_state3, VK_EXT_shader_object
void vkCmdSetSampleMaskEXT(
    VkCommandBuffer                             commandBuffer,
    VkSampleCountFlagBits                       samples,
    const VkSampleMask*                         pSampleMask);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `samples` specifies the number of sample bits in the `pSampleMask`.
- `pSampleMask` is a pointer to an array of VkSampleMask values, where the array size is based on the `samples` parameter.

This command sets the sample mask for subsequent drawing commands when drawing using shader objects, or when the graphics pipeline is created with `VK_DYNAMIC_STATE_SAMPLE_MASK_EXT` set in VkPipelineDynamicStateCreateInfo::`pDynamicStates`. Otherwise, this state is specified by the VkPipelineMultisampleStateCreateInfo::`pSampleMask` value used to create the currently active pipeline.

## Valid Usage

- VUID-vkCmdSetSampleMaskEXT-None-09423
  At least one of the following **must** be true:
  - The `extendedDynamicState3SampleMask` feature is enabled
  - The `shaderObject` feature is enabled

## Valid Usage (Implicit)

- VUID-vkCmdSetSampleMaskEXT-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdSetSampleMaskEXT-samples-parameter
  `samples` **must** be a valid VkSampleCountFlagBits value
- VUID-vkCmdSetSampleMaskEXT-pSampleMask-parameter
  `pSampleMask` **must** be a valid pointer to an array of $\lceil \frac{samples}{32} \rceil$ VkSampleMask values
- VUID-vkCmdSetSampleMaskEXT-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdSetSampleMaskEXT-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- VUID-vkCmdSetSampleMaskEXT-videocoding
  This command **must** only be called outside of a video coding scope

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics | State |

Each bit of the coverage mask is associated with a sample index as described in the rasterization chapter. If the bit in VkPipelineMultisampleStateCreateInfo::`pSampleMask` which is associated with that same sample index is set to `0`, the coverage mask bit is set to `0`.

## 29.5. Fragment Shading

Fragment shaders are invoked for each fragment, or as helper invocations.

Most operations in the fragment shader are not performed in rasterization order, with exceptions called out in the following sections.

For fragment shaders invoked by fragments, the following rules apply:

- A fragment shader **must** not be executed if a fragment operation that executes before fragment shading discards the fragment.
- A fragment shader **may** not be executed if:
  - An implementation determines that another fragment shader, invoked by a subsequent primitive in primitive order, overwrites all results computed by the shader (including writes to storage resources).
  - Any other fragment operation discards the fragment, and the shader does not write to any storage resources.
  - If a fragment shader statically computes the same values for different framebuffer locations, and does not write to any storage resources, multiple fragments **may** be shaded by one fragment shader invocation. This **may** affect `VK_QUERY_PIPELINE_STATISTIC_FRAGMENT_SHADER_INVOCATIONS_BIT` results, but **must** otherwise not be visible behavior to applications.
- Otherwise, at least one fragment shader **must** be executed.
  - If sample shading is enabled and multiple invocations per fragment are **required**, additional invocations **must** be executed as specified.
  - If a shading rate image is used and multiple invocations per fragment are **required**, additional invocations **must** be executed as specified.
  - Each covered sample **must** be included in at least one fragment shader invocation.

If no fragment shader is included in the pipeline, no fragment shader is executed, and undefined values **may** be written to all color attachment outputs during this fragment operation.

**NOTE**

Multiple fragment shader invocations may be executed for the same fragment for any number of implementation-dependent reasons. When there is more than one fragment shader invocation per fragment, the association of samples to invocations is implementation-dependent. Stores and atomics performed by these additional invocations have the normal effect.

For example, if the subpass includes multiple views in its view mask, a fragment shader may be invoked separately for each view.

Similarly, if the render pass has a fragment density map attachment, more than one fragment shader invocation may be invoked for each covered sample. Such additional invocations are only produced if `VkPhysicalDeviceFragmentDensityMapPropertiesEXT::fragmentDensityInvocations` is `VK_TRUE`. Implementations may generate these additional fragment shader

invocations in order to make transitions between fragment areas with different fragment densities more smooth.

### 29.5.1. Sample Mask

Reading from the `SampleMask` built-in in the `Input` storage class will return the coverage mask for the current fragment as calculated by fragment operations that executed prior to fragment shading.

If sample shading is enabled, fragment shaders will only see values of `1` for samples being shaded - other bits will be `0`.

Each bit of the coverage mask is associated with a sample index as described in the rasterization chapter. If the bit in `SampleMask` which is associated with that same sample index is set to `0`, that coverage mask bit is set to `0`.

Values written to the `SampleMask` built-in in the `Output` storage class will be used by the multisample coverage operation, with the same encoding as the input built-in.

### 29.5.2. Fragment Shader Tile Image Reads

If the `VK_EXT_shader_tile_image` extension is enabled, implementations divide the framebuffer into a grid of tiles. A tile image is a view of a framebuffer attachment tile for fragments with locations within the tile.

Within a render pass instance initiated by vkCmdBeginRenderingKHR, fragment shader invocations **can** read the framebuffer color, depth, and stencil values at the fragment location via tile images.

**NOTE**

Even though fragment shader invocation can only read from the corresponding fragment location, the abstraction of a tile image is introduced for the following reasons:

- Tile dimensions will be exposed in a future extension
- Future functionality such as executing compute dispatches within render passes via tile shaders can leverage tile images.

Enabling shaderTileImageColorReadAccess, shaderTileImageDepthReadAccess, shaderTileImageStencilReadAccess enables fragment shader invocations to read from color, depth, and stencil, respectively.

Color values are read from tile image variables with `OpColorAttachmentReadEXT`. Tile image variables are linked to specific color attachments using `Location` decoration. See Fragment Tile Image Interface for more details.

Depth values are read with `OpDepthAttachmentReadEXT`.

Stencil values are read with `OpStencilAttachmentReadEXT`.

The sample to read is specified by a sample index value specified as the `Sample` operand to

OpColorAttachmentReadEXT, OpDepthAttachmentReadEXT, or OpStencilAttachmentReadEXT.

If sample shading is disabled, a fragment invocation **can** read from all sample locations associated with the fragment regardless of the fragment's coverage. This functionality is supported for VkPipelineMultisampleStateCreateInfo::rasterizationSamples > 1 when VkPhysicalDeviceShaderTileImagePropertiesEXT::shaderTileImageReadSampleFromPixelRateInvocation is VK_TRUE.

If sample shading is enabled, and minSampleShading is 1.0, a fragment invocation **must** only read from the coverage index sample. Tile image access **must** not be used if the value of minSampleShading is not 1.0.

If the fragment shader declares the EarlyFragmentTests execution mode, depth reads are allowed only if depth writes are disabled and stencil reads are allowed only if stencil writes are disabled.

If VkPhysicalDeviceShaderTileImagePropertiesEXT::shaderTileImageReadFromHelperInvocation is VK_FALSE, values read from helper invocations are undefined otherwise the values read are subject to the coherency guarantees described below.

OpDepthAttachmentReadEXT returns an undefined value if no depth attachment is present. OpStencilAttachmentReadEXT returns an undefined value if no stencil attachment is present.

Tile image reads from color, depth and stencil attachments are said to be coherent when the accesses happen in raster order and without data race with respect to accesses to the attachments from framebuffer-space pipeline stages. The samples which qualify for coherent access and the enabling conditions are described below.

- Let Rc be the set of components being read from an attachment A in a draw call
- Let Wc be the set of components being written to A by the draw call

The samples which qualify for coherent tile image reads from an attachment A are:

- All samples in a pixel when Rc is disjoint with Wc.
- The samples with coverage in a fragment when Rc is not disjoint with Wc. The samples with coverage are determined by the coverage mask for the fragment as calculated by fragment operations that executed prior to fragment shading, including early fragment tests if enabled for the draw call.

A fragment shader **can** declare NonCoherentColorAttachmentReadEXT, NonCoherentDepthAttachmentReadEXT, or NonCoherentStencilAttachmentReadEXT execution modes to enable non-coherent tile image reads which require an explicit vkCmdPipelineBarrier2KHR call for the writes to an attachment to be made visible via tile image reads.

When VkPhysicalDeviceShaderTileImagePropertiesEXT::shaderTileImageCoherentReadAccelerated is VK_TRUE, the implementation prefers that coherent tile image reads are used, otherwise the implementation prefers that non-coherent tile image reads are used.

| | |
|---|---|
| **NOTE** | In practice, the most common tile image reads usage patterns fall under one of the following: |

- Programmable blending - each fragment reads from a single sample (SampleID) at its location. Per-sample shading is typically enabled when multisampled rendertargets are used.
- G-buffer generation and shading in one render pass - in the shading phase a fragment reads from a single sample at its location.
- Programmable resolve - a fragment reads from all samples at its location (per-sample shading is disabled). This requires the use of a "full-screen triangle" instead of a rectangle composed of two triangles in order to avoid data races along the shared edge of the triangles.
- 1:1 texturing with LOD - in use cases such a deferred screen space decals a fragment reads a single sample (SampleID) from depth buffer, but requires being able to read from helper threads to derive the texture LOD. This use case is supported as long as the attachment components being read are not overwritten by color, depth, or stencil attachment writes.

All of the above use cases are supported by coherent tile image reads, but only the latter three are supported when non-coherent reads are used as there is no mechanism to synchronize non-coherent reads with writes within a draw call.

### 29.5.3. Depth Replacement

Writing to the `FragDepth` built-in will replace the fragment's calculated depth values for each sample in the input `SampleMask`. Depth testing performed after the fragment shader for this fragment will use this new value as $z_f$.

### 29.5.4. Stencil Reference Replacement

Writing to the `FragStencilRefEXT` built-in will replace the fragment's stencil reference value for each sample in the input `SampleMask`. Stencil testing performed after the fragment shader for this fragment will use this new value as $s_r$.

### 29.5.5. Interlocked Operations

`OpBeginInvocationInterlockEXT` and `OpEndInvocationInterlockEXT` define a section of a fragment shader which imposes additional ordering constraints on operations performed within them. These operations are defined as *interlocked operations*. How interlocked operations are ordered against other fragment shader invocations depends on the specified execution modes.

If the `ShadingRateInterlockOrderedEXT` execution mode is specified, any interlocked operations in a fragment shader **must** happen before interlocked operations in fragment shader invocations that execute later in rasterization order and cover at least one sample in the same fragment area, and **must** happen after interlocked operations in a fragment shader that executes earlier in rasterization order and cover at least one sample in the same fragment area.

If the `ShadingRateInterlockUnorderedEXT` execution mode is specified, any interlocked operations in a fragment shader **must** happen before or after interlocked operations in fragment shader invocations that execute earlier or later in rasterization order and cover at least one sample in the

same fragment area.

If the PixelInterlockOrderedEXT execution mode is specified, any interlocked operations in a fragment shader **must** happen before interlocked operations in fragment shader invocations that execute later in rasterization order and cover at least one sample in the same pixel, and **must** happen after interlocked operations in a fragment shader that executes earlier in rasterization order and cover at least one sample in the same pixel.

If the PixelInterlockUnorderedEXT execution mode is specified, any interlocked operations in a fragment shader **must** happen before or after interlocked operations in fragment shader invocations that execute earlier or later in rasterization order and cover at least one sample in the same pixel.

If the SampleInterlockOrderedEXT execution mode is specified, any interlocked operations in a fragment shader **must** happen before interlocked operations in fragment shader invocations that execute later in rasterization order and cover at least one of the same samples, and **must** happen after interlocked operations in a fragment shader that executes earlier in rasterization order and cover at least one of the same samples.

If the SampleInterlockUnorderedEXT execution mode is specified, any interlocked operations in a fragment shader **must** happen before or after interlocked operations in fragment shader invocations that execute earlier or later in rasterization order and cover at least one of the same samples.

## 29.6. Multisample Coverage

If a fragment shader is active and its entry point's interface includes a built-in output variable decorated with SampleMask, but not OverrideCoverageNV, the coverage mask is ANDed with the bits of the SampleMask built-in to generate a new coverage mask. If the SampleMask built-in is also decorated with OverrideCoverageNV, the coverage mask is replaced with the mask bits set in the shader. If sample shading is enabled, bits written to SampleMask corresponding to samples that are not being shaded by the fragment shader invocation are ignored. If no fragment shader is active, or if the active fragment shader does not include SampleMask in its interface, the coverage mask is not modified.

Next, the fragment alpha value and coverage mask are modified based on the line coverage factor if the lineRasterizationMode member of the VkPipelineRasterizationStateCreateInfo structure is VK_LINE_RASTERIZATION_MODE_RECTANGULAR_SMOOTH_KHR, and the alphaToCoverageEnable and alphaToOneEnable members of the VkPipelineMultisampleStateCreateInfo structure.

To dynamically set the alphaToCoverageEnable state, call:

```
// Provided by VK_EXT_extended_dynamic_state3, VK_EXT_shader_object
void vkCmdSetAlphaToCoverageEnableEXT(
    VkCommandBuffer                             commandBuffer,
    VkBool32                                    alphaToCoverageEnable);
```

- commandBuffer is the command buffer into which the command will be recorded.

- `alphaToCoverageEnable` specifies the `alphaToCoverageEnable` state.

This command sets the `alphaToCoverageEnable` state for subsequent drawing commands when drawing using shader objects, or when the graphics pipeline is created with `VK_DYNAMIC_STATE_ALPHA_TO_COVERAGE_ENABLE_EXT` set in VkPipelineDynamicStateCreateInfo::`pDynamicStates`. Otherwise, this state is specified by the VkPipelineMultisampleStateCreateInfo::`alphaToCoverageEnable` value used to create the currently active pipeline.

## Valid Usage

- VUID-vkCmdSetAlphaToCoverageEnableEXT-None-09423
  At least one of the following **must** be true:
  - The `extendedDynamicState3AlphaToCoverageEnable` feature is enabled
  - The `shaderObject` feature is enabled

## Valid Usage (Implicit)

- VUID-vkCmdSetAlphaToCoverageEnableEXT-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdSetAlphaToCoverageEnableEXT-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdSetAlphaToCoverageEnableEXT-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- VUID-vkCmdSetAlphaToCoverageEnableEXT-videocoding
  This command **must** only be called outside of a video coding scope

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics | State |

To dynamically set the `alphaToOneEnable` state, call:

```
// Provided by VK_EXT_extended_dynamic_state3, VK_EXT_shader_object
void vkCmdSetAlphaToOneEnableEXT(
    VkCommandBuffer                             commandBuffer,
    VkBool32                                    alphaToOneEnable);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `alphaToOneEnable` specifies the `alphaToOneEnable` state.

This command sets the `alphaToOneEnable` state for subsequent drawing commands when drawing using shader objects, or when the graphics pipeline is created with `VK_DYNAMIC_STATE_ALPHA_TO_ONE_ENABLE_EXT` set in VkPipelineDynamicStateCreateInfo::`pDynamicStates`. Otherwise, this state is specified by the VkPipelineMultisampleStateCreateInfo::`alphaToOneEnable` value used to create the currently active pipeline.

## Valid Usage

- VUID-vkCmdSetAlphaToOneEnableEXT-None-09423
  At least one of the following **must** be true:
  - The `extendedDynamicState3AlphaToOneEnable` feature is enabled
  - The `shaderObject` feature is enabled
- VUID-vkCmdSetAlphaToOneEnableEXT-alphaToOne-07607
  If the `alphaToOne` feature is not enabled, `alphaToOneEnable` **must** be `VK_FALSE`

## Valid Usage (Implicit)

- VUID-vkCmdSetAlphaToOneEnableEXT-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdSetAlphaToOneEnableEXT-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdSetAlphaToOneEnableEXT-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- VUID-vkCmdSetAlphaToOneEnableEXT-videocoding
  This command **must** only be called outside of a video coding scope

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

**Command Properties**

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics | State |

All alpha values in this section refer only to the alpha component of the fragment shader output that has a `Location` and `Index` decoration of zero (see the Fragment Output Interface section). If that shader output has an integer or unsigned integer type, then these operations are skipped.

If the `lineRasterizationMode` member of the VkPipelineRasterizationStateCreateInfo structure is `VK_LINE_RASTERIZATION_MODE_RECTANGULAR_SMOOTH_KHR` and the fragment came from a line segment, then the alpha value is replaced by multiplying it by the coverage factor for the fragment computed during smooth line rasterization.

If `alphaToCoverageEnable` is enabled, a temporary coverage mask is generated where each bit is determined by the fragment's alpha value, which is ANDed with the fragment coverage mask.

No specific algorithm is specified for converting the alpha value to a temporary coverage mask. It is intended that the number of 1's in this value be proportional to the alpha value (clamped to [0,1]), with all 1's corresponding to a value of 1.0 and all 0's corresponding to 0.0. The algorithm **may** be different at different framebuffer coordinates.

**NOTE** Using different algorithms at different framebuffer coordinates **may** help to avoid artifacts caused by regular coverage sample locations.

Finally, if `alphaToOneEnable` is enabled, each alpha value is replaced by the maximum representable alpha value for fixed-point color attachments, or by 1.0 for floating-point attachments. Otherwise, the alpha values are not changed.

## 29.7. Depth and Stencil Operations

Pipeline state controlling the depth bounds tests, stencil test, and depth test is specified through the members of the `VkPipelineDepthStencilStateCreateInfo` structure.

The `VkPipelineDepthStencilStateCreateInfo` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkPipelineDepthStencilStateCreateInfo {
    VkStructureType                             sType;
    const void*                                 pNext;
    VkPipelineDepthStencilStateCreateFlags      flags;
    VkBool32                                    depthTestEnable;
    VkBool32                                    depthWriteEnable;
    VkCompareOp                                 depthCompareOp;
    VkBool32                                    depthBoundsTestEnable;
    VkBool32                                    stencilTestEnable;
    VkStencilOpState                            front;
    VkStencilOpState                            back;
    float                                       minDepthBounds;
    float                                       maxDepthBounds;
} VkPipelineDepthStencilStateCreateInfo;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is a bitmask of VkPipelineDepthStencilStateCreateFlagBits specifying additional depth/stencil state information.
- `depthTestEnable` controls whether depth testing is enabled.
- `depthWriteEnable` controls whether depth writes are enabled when `depthTestEnable` is `VK_TRUE`. Depth writes are always disabled when `depthTestEnable` is `VK_FALSE`.
- `depthCompareOp` is a VkCompareOp value specifying the comparison operator to use in the Depth Comparison step of the depth test.
- `depthBoundsTestEnable` controls whether depth bounds testing is enabled.
- `stencilTestEnable` controls whether stencil testing is enabled.
- `front` and `back` are VkStencilOpState values controlling the corresponding parameters of the stencil test.
- `minDepthBounds` is the minimum depth bound used in the depth bounds test.
- `maxDepthBounds` is the maximum depth bound used in the depth bounds test.

## Valid Usage

- VUID-VkPipelineDepthStencilStateCreateInfo-depthBoundsTestEnable-00598
  If the `depthBounds` feature is not enabled, `depthBoundsTestEnable` **must** be `VK_FALSE`
- VUID-VkPipelineDepthStencilStateCreateInfo-separateStencilMaskRef-04453
  If the `VK_KHR_portability_subset` extension is enabled, and VkPhysicalDevicePortabilitySubsetFeaturesKHR::`separateStencilMaskRef` is `VK_FALSE`, and the value of VkPipelineDepthStencilStateCreateInfo::`stencilTestEnable` is `VK_TRUE`, and the value of VkPipelineRasterizationStateCreateInfo::`cullMode` is `VK_CULL_MODE_NONE`, the value of `reference` in each of the VkStencilOpState structs in `front` and `back` **must** be the same
- VUID-VkPipelineDepthStencilStateCreateInfo-rasterizationOrderDepthAttachmentAccess-

06463
If the `rasterizationOrderDepthAttachmentAccess` feature is not enabled, `flags` **must** not include `VK_PIPELINE_DEPTH_STENCIL_STATE_CREATE_RASTERIZATION_ORDER_ATTACHMENT_DEPTH_ACCESS_BIT_EXT`

- VUID-VkPipelineDepthStencilStateCreateInfo-rasterizationOrderStencilAttachmentAccess-06464
If the `rasterizationOrderStencilAttachmentAccess` feature is not enabled, `flags` **must** not include `VK_PIPELINE_DEPTH_STENCIL_STATE_CREATE_RASTERIZATION_ORDER_ATTACHMENT_STENCIL_ACCESS_BIT_EXT`

## Valid Usage (Implicit)

- VUID-VkPipelineDepthStencilStateCreateInfo-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_PIPELINE_DEPTH_STENCIL_STATE_CREATE_INFO`
- VUID-VkPipelineDepthStencilStateCreateInfo-pNext-pNext
`pNext` **must** be `NULL`
- VUID-VkPipelineDepthStencilStateCreateInfo-flags-parameter
`flags` **must** be a valid combination of VkPipelineDepthStencilStateCreateFlagBits values
- VUID-VkPipelineDepthStencilStateCreateInfo-depthCompareOp-parameter
`depthCompareOp` **must** be a valid VkCompareOp value
- VUID-VkPipelineDepthStencilStateCreateInfo-front-parameter
`front` **must** be a valid VkStencilOpState structure
- VUID-VkPipelineDepthStencilStateCreateInfo-back-parameter
`back` **must** be a valid VkStencilOpState structure

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkPipelineDepthStencilStateCreateFlags;
```

`VkPipelineDepthStencilStateCreateFlags` is a bitmask type for setting a mask of zero or more VkPipelineDepthStencilStateCreateFlagBits.

Bits which **can** be set in the VkPipelineDepthStencilStateCreateInfo::`flags` parameter are:

```
// Provided by VK_EXT_rasterization_order_attachment_access
typedef enum VkPipelineDepthStencilStateCreateFlagBits {
  // Provided by VK_EXT_rasterization_order_attachment_access

VK_PIPELINE_DEPTH_STENCIL_STATE_CREATE_RASTERIZATION_ORDER_ATTACHMENT_DEPTH_ACCESS_BIT
_EXT = 0x00000001,
  // Provided by VK_EXT_rasterization_order_attachment_access

VK_PIPELINE_DEPTH_STENCIL_STATE_CREATE_RASTERIZATION_ORDER_ATTACHMENT_STENCIL_ACCESS_B
IT_EXT = 0x00000002,
  // Provided by VK_ARM_rasterization_order_attachment_access

VK_PIPELINE_DEPTH_STENCIL_STATE_CREATE_RASTERIZATION_ORDER_ATTACHMENT_DEPTH_ACCESS_BIT
_ARM =
VK_PIPELINE_DEPTH_STENCIL_STATE_CREATE_RASTERIZATION_ORDER_ATTACHMENT_DEPTH_ACCESS_BIT
_EXT,
  // Provided by VK_ARM_rasterization_order_attachment_access

VK_PIPELINE_DEPTH_STENCIL_STATE_CREATE_RASTERIZATION_ORDER_ATTACHMENT_STENCIL_ACCESS_B
IT_ARM =
VK_PIPELINE_DEPTH_STENCIL_STATE_CREATE_RASTERIZATION_ORDER_ATTACHMENT_STENCIL_ACCESS_B
IT_EXT,
} VkPipelineDepthStencilStateCreateFlagBits;
```

- `VK_PIPELINE_DEPTH_STENCIL_STATE_CREATE_RASTERIZATION_ORDER_ATTACHMENT_DEPTH_ACCESS_BIT_EXT` indicates that access to the depth aspects of depth/stencil and input attachments will have implicit framebuffer-local memory dependencies.
- `VK_PIPELINE_DEPTH_STENCIL_STATE_CREATE_RASTERIZATION_ORDER_ATTACHMENT_STENCIL_ACCESS_BIT_EXT` indicates that access to the stencil aspects of depth/stencil and input attachments will have implicit framebuffer-local memory dependencies.

When `VK_PIPELINE_DEPTH_STENCIL_STATE_CREATE_RASTERIZATION_ORDER_ATTACHMENT_DEPTH_ACCESS_BIT_EXT` is included in a pipeline, it forms a framebuffer-local memory dependency for each fragment generated by draw commands for that pipeline with the following scopes:

- The first synchronization scope includes `VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT` and `VK_PIPELINE_STAGE_LATE_FRAGMENT_TESTS_BIT` pipeline stages executed by all previous fragments (as defined by primitive order) in the corresponding framebuffer regions including those generated by the same draw command.
- The second synchronization scope includes `VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT` stage executed by the generated fragment.
- The first access scope includes all writes to the depth aspect of depth/stencil attachments.
- The second access scope includes all reads from the depth aspect of input attachments.

When `VK_PIPELINE_DEPTH_STENCIL_STATE_CREATE_RASTERIZATION_ORDER_ATTACHMENT_STENCIL_ACCESS_BIT_EXT` is

included in a pipeline, it forms a framebuffer-local memory dependency for each fragment generated by draw commands for that pipeline with the following scopes:

- The first synchronization scope includes VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT VK_PIPELINE_STAGE_LATE_FRAGMENT_TESTS_BIT pipeline stages executed by all previous fragments (as defined by primitive order) in the corresponding framebuffer regions including those generated by the same draw command.
- The second synchronization scope includes VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT and VK_PIPELINE_STAGE_LATE_FRAGMENT_TESTS_BIT pipeline stages executed by the generated fragment.
- The first access scope includes all writes to the stencil aspect of depth/stencil attachments.
- The second access scope includes all reads from the stencil aspect of input attachments.

## 29.8. Depth Bounds Test

The depth bounds test compares the depth value $z_a$ in the depth/stencil attachment at each sample's framebuffer coordinates $(x_f,y_f)$ and sample index i against a set of *depth bounds*.

The depth bounds are determined by two floating-point values defining a minimum ( minDepthBounds) and maximum (maxDepthBounds) depth value. These values are either set by the VkPipelineDepthStencilStateCreateInfo structure during pipeline creation, or dynamically by vkCmdSetDepthBoundsTestEnableEXT and vkCmdSetDepthBounds.

A given sample is considered within the depth bounds if $z_a$ is in the range [minDepthBounds ,maxDepthBounds]. Samples with depth attachment values outside of the depth bounds will have their coverage set to 0.

If the depth bounds test is disabled, or if there is no depth attachment, the coverage mask is unmodified by this operation.

To dynamically enable or disable the depth bounds test, call:

```
// Provided by VK_EXT_extended_dynamic_state, VK_EXT_shader_object
void vkCmdSetDepthBoundsTestEnableEXT(
    VkCommandBuffer                             commandBuffer,
    VkBool32                                    depthBoundsTestEnable);
```

- commandBuffer is the command buffer into which the command will be recorded.
- depthBoundsTestEnable specifies if the depth bounds test is enabled.

This command sets the depth bounds enable for subsequent drawing commands when drawing using shader objects, or when the graphics pipeline is created with VK_DYNAMIC_STATE_DEPTH_BOUNDS_TEST_ENABLE set in VkPipelineDynamicStateCreateInfo ::pDynamicStates. Otherwise, this state is specified by the VkPipelineDepthStencilStateCreateInfo ::depthBoundsTestEnable value used to create the currently active pipeline.

## Valid Usage

- VUID-vkCmdSetDepthBoundsTestEnable-None-08971
  At least one of the following **must** be true:
  - the `extendedDynamicState` feature is enabled
  - the `shaderObject` feature is enabled
- VUID-vkCmdSetDepthBoundsTestEnable-depthBounds-10010
  If the `depthBounds` feature is not enabled, `depthBoundsTestEnable` **must** be `VK_FALSE`

## Valid Usage (Implicit)

- VUID-vkCmdSetDepthBoundsTestEnable-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdSetDepthBoundsTestEnable-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdSetDepthBoundsTestEnable-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- VUID-vkCmdSetDepthBoundsTestEnable-videocoding
  This command **must** only be called outside of a video coding scope

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics | State |

To dynamically set the depth bounds range, call:

```
// Provided by VK_VERSION_1_0
void vkCmdSetDepthBounds(
    VkCommandBuffer                             commandBuffer,
    float                                       minDepthBounds,
    float                                       maxDepthBounds);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `minDepthBounds` is the minimum depth bound.
- `maxDepthBounds` is the maximum depth bound.

This command sets the depth bounds range for subsequent drawing commands when drawing using shader objects, or when the graphics pipeline is created with `VK_DYNAMIC_STATE_DEPTH_BOUNDS` set in VkPipelineDynamicStateCreateInfo::`pDynamicStates`. Otherwise, this state is specified by the VkPipelineDepthStencilStateCreateInfo::`minDepthBounds` and VkPipelineDepthStencilStateCreateInfo::`maxDepthBounds` values used to create the currently active pipeline.

## Valid Usage

- VUID-vkCmdSetDepthBounds-minDepthBounds-00600
  If the `VK_EXT_depth_range_unrestricted` extension is not enabled `minDepthBounds` **must** be between `0.0` and `1.0`, inclusive
- VUID-vkCmdSetDepthBounds-maxDepthBounds-00601
  If the `VK_EXT_depth_range_unrestricted` extension is not enabled `maxDepthBounds` **must** be between `0.0` and `1.0`, inclusive

## Valid Usage (Implicit)

- VUID-vkCmdSetDepthBounds-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdSetDepthBounds-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdSetDepthBounds-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- VUID-vkCmdSetDepthBounds-videocoding
  This command **must** only be called outside of a video coding scope

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally

synchronized

**Command Properties**

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary Secondary | Both | Outside | Graphics | State |

## 29.9. Stencil Test

The stencil test compares the stencil attachment value $s_a$ in the depth/stencil attachment at each sample's framebuffer coordinates ($x_f$,$y_f$) and sample index i against a *stencil reference value*.

If the render pass has a fragment density map attachment and the fragment covers multiple pixels, there is an implementation-dependent association of coverage samples to stencil attachment samples within the fragment. However, if all samples in the fragment are covered, and the stencil attachment value is updated as a result of this test, all stencil attachment samples will be updated.

If the stencil test is not enabled, as specified by vkCmdSetStencilTestEnableEXT or VkPipelineDepthStencilStateCreateInfo::`stencilTestEnable`, or if there is no stencil attachment, the coverage mask is unmodified by this operation.

The stencil test is controlled by one of two sets of stencil-related state, the front stencil state and the back stencil state. Stencil tests and writes use the back stencil state when processing fragments generated by back-facing polygons, and the front stencil state when processing fragments generated by front-facing polygons or any other primitives.

The comparison operation performed is determined by the VkCompareOp value set by vkCmdSetStencilOpEXT::`compareOp`, or by VkStencilOpState::`compareOp` during pipeline creation.

The compare mask $s_c$ and stencil reference value $s_r$ of the front or the back stencil state set determine arguments of the comparison operation. $s_c$ is set by the VkPipelineDepthStencilStateCreateInfo structure during pipeline creation, or by the vkCmdSetStencilCompareMask command. $s_r$ is set by VkPipelineDepthStencilStateCreateInfo or by vkCmdSetStencilReference.

$s_r$ and $s_a$ are each independently combined with $s_c$ using a bitwise `AND` operation to create masked reference and attachment values $s'_r$ and $s'_a$. $s'_r$ and $s'_a$ are used as the *reference* and *test* values, respectively, in the operation specified by the VkCompareOp.

If the comparison evaluates to false, the coverage for the sample is set to `0`.

A new stencil value $s_g$ is generated according to a stencil operation defined by VkStencilOp parameters set by vkCmdSetStencilOpEXT or VkPipelineDepthStencilStateCreateInfo. If the stencil test fails, `failOp` defines the stencil operation used. If the stencil test passes however, the stencil op used is based on the depth test - if it passes, VkPipelineDepthStencilStateCreateInfo::`passOp` is used,

otherwise VkPipelineDepthStencilStateCreateInfo::depthFailOp is used.

The stencil attachment value $s_a$ is then updated with the generated stencil value $s_g$ according to the write mask $s_w$ defined by writeMask in VkPipelineDepthStencilStateCreateInfo::front and VkPipelineDepthStencilStateCreateInfo::back as:

$$s_a = (s_a \& \neg s_w) \mid (s_g \& s_w)$$

To dynamically enable or disable the stencil test, call:

```
// Provided by VK_EXT_extended_dynamic_state, VK_EXT_shader_object
void vkCmdSetStencilTestEnableEXT(
    VkCommandBuffer                             commandBuffer,
    VkBool32                                    stencilTestEnable);
```

- commandBuffer is the command buffer into which the command will be recorded.
- stencilTestEnable specifies if the stencil test is enabled.

This command sets the stencil test enable for subsequent drawing commands when drawing using shader objects, or when the graphics pipeline is created with VK_DYNAMIC_STATE_STENCIL_TEST_ENABLE set in VkPipelineDynamicStateCreateInfo::pDynamicStates. Otherwise, this state is specified by the VkPipelineDepthStencilStateCreateInfo::stencilTestEnable value used to create the currently active pipeline.

### Valid Usage

- VUID-vkCmdSetStencilTestEnable-None-08971
  At least one of the following **must** be true:
  - the extendedDynamicState feature is enabled
  - the shaderObject feature is enabled

### Valid Usage (Implicit)

- VUID-vkCmdSetStencilTestEnable-commandBuffer-parameter
  commandBuffer **must** be a valid VkCommandBuffer handle
- VUID-vkCmdSetStencilTestEnable-commandBuffer-recording
  commandBuffer **must** be in the recording state
- VUID-vkCmdSetStencilTestEnable-commandBuffer-cmdpool
  The VkCommandPool that commandBuffer was allocated from **must** support graphics operations
- VUID-vkCmdSetStencilTestEnable-videocoding
  This command **must** only be called outside of a video coding scope

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics | State |

To dynamically set the stencil operation, call:

```
// Provided by VK_EXT_extended_dynamic_state, VK_EXT_shader_object
void vkCmdSetStencilOpEXT(
    VkCommandBuffer                             commandBuffer,
    VkStencilFaceFlags                          faceMask,
    VkStencilOp                                 failOp,
    VkStencilOp                                 passOp,
    VkStencilOp                                 depthFailOp,
    VkCompareOp                                 compareOp);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `faceMask` is a bitmask of VkStencilFaceFlagBits specifying the set of stencil state for which to update the stencil operation.
- `failOp` is a VkStencilOp value specifying the action performed on samples that fail the stencil test.
- `passOp` is a VkStencilOp value specifying the action performed on samples that pass both the depth and stencil tests.
- `depthFailOp` is a VkStencilOp value specifying the action performed on samples that pass the stencil test and fail the depth test.
- `compareOp` is a VkCompareOp value specifying the comparison operator used in the stencil test.

This command sets the stencil operation for subsequent drawing commands when when drawing using shader objects, or when the graphics pipeline is created with `VK_DYNAMIC_STATE_STENCIL_OP` set in VkPipelineDynamicStateCreateInfo::`pDynamicStates`. Otherwise, this state is specified by the corresponding `VkPipelineDepthStencilStateCreateInfo`::`failOp`, `passOp`, `depthFailOp`, and `compareOp` values used to create the currently active pipeline, for both front and back faces.

## Valid Usage

- VUID-vkCmdSetStencilOp-None-08971
  At least one of the following **must** be true:
  - the `extendedDynamicState` feature is enabled
  - the `shaderObject` feature is enabled

## Valid Usage (Implicit)

- VUID-vkCmdSetStencilOp-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdSetStencilOp-faceMask-parameter
  `faceMask` **must** be a valid combination of VkStencilFaceFlagBits values
- VUID-vkCmdSetStencilOp-faceMask-requiredbitmask
  `faceMask` **must** not be `0`
- VUID-vkCmdSetStencilOp-failOp-parameter
  `failOp` **must** be a valid VkStencilOp value
- VUID-vkCmdSetStencilOp-passOp-parameter
  `passOp` **must** be a valid VkStencilOp value
- VUID-vkCmdSetStencilOp-depthFailOp-parameter
  `depthFailOp` **must** be a valid VkStencilOp value
- VUID-vkCmdSetStencilOp-compareOp-parameter
  `compareOp` **must** be a valid VkCompareOp value
- VUID-vkCmdSetStencilOp-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdSetStencilOp-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- VUID-vkCmdSetStencilOp-videocoding
  This command **must** only be called outside of a video coding scope

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics | State |

The `VkStencilOpState` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkStencilOpState {
    VkStencilOp    failOp;
    VkStencilOp    passOp;
    VkStencilOp    depthFailOp;
    VkCompareOp    compareOp;
    uint32_t       compareMask;
    uint32_t       writeMask;
    uint32_t       reference;
} VkStencilOpState;
```

- `failOp` is a VkStencilOp value specifying the action performed on samples that fail the stencil test.
- `passOp` is a VkStencilOp value specifying the action performed on samples that pass both the depth and stencil tests.
- `depthFailOp` is a VkStencilOp value specifying the action performed on samples that pass the stencil test and fail the depth test.
- `compareOp` is a VkCompareOp value specifying the comparison operator used in the stencil test.
- `compareMask` selects the bits of the unsigned integer stencil values participating in the stencil test.
- `writeMask` selects the bits of the unsigned integer stencil values updated by the stencil test in the stencil framebuffer attachment.
- `reference` is an integer stencil reference value that is used in the unsigned stencil comparison.

## Valid Usage (Implicit)

- VUID-VkStencilOpState-failOp-parameter
  `failOp` **must** be a valid VkStencilOp value
- VUID-VkStencilOpState-passOp-parameter
  `passOp` **must** be a valid VkStencilOp value
- VUID-VkStencilOpState-depthFailOp-parameter
  `depthFailOp` **must** be a valid VkStencilOp value
- VUID-VkStencilOpState-compareOp-parameter
  `compareOp` **must** be a valid VkCompareOp value

To dynamically set the stencil compare mask, call:

```
// Provided by VK_VERSION_1_0
void vkCmdSetStencilCompareMask(
    VkCommandBuffer                             commandBuffer,
    VkStencilFaceFlags                          faceMask,
    uint32_t                                    compareMask);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `faceMask` is a bitmask of VkStencilFaceFlagBits specifying the set of stencil state for which to update the compare mask.
- `compareMask` is the new value to use as the stencil compare mask.

This command sets the stencil compare mask for subsequent drawing commands when drawing using shader objects, or when the graphics pipeline is created with `VK_DYNAMIC_STATE_STENCIL_COMPARE_MASK` set in VkPipelineDynamicStateCreateInfo::`pDynamicStates`. Otherwise, this state is specified by the VkStencilOpState::`compareMask` value used to create the currently active pipeline, for both front and back faces.

## Valid Usage (Implicit)

- VUID-vkCmdSetStencilCompareMask-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdSetStencilCompareMask-faceMask-parameter
  `faceMask` **must** be a valid combination of VkStencilFaceFlagBits values
- VUID-vkCmdSetStencilCompareMask-faceMask-requiredbitmask
  `faceMask` **must** not be `0`
- VUID-vkCmdSetStencilCompareMask-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdSetStencilCompareMask-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- VUID-vkCmdSetStencilCompareMask-videocoding
  This command **must** only be called outside of a video coding scope

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

### Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics | State |

VkStencilFaceFlagBits values are:

```
// Provided by VK_VERSION_1_0
typedef enum VkStencilFaceFlagBits {
    VK_STENCIL_FACE_FRONT_BIT = 0x00000001,
    VK_STENCIL_FACE_BACK_BIT = 0x00000002,
    VK_STENCIL_FACE_FRONT_AND_BACK = 0x00000003,
  // VK_STENCIL_FRONT_AND_BACK is a deprecated alias
    VK_STENCIL_FRONT_AND_BACK = VK_STENCIL_FACE_FRONT_AND_BACK,
} VkStencilFaceFlagBits;
```

- VK_STENCIL_FACE_FRONT_BIT specifies that only the front set of stencil state is updated.
- VK_STENCIL_FACE_BACK_BIT specifies that only the back set of stencil state is updated.
- VK_STENCIL_FACE_FRONT_AND_BACK is the combination of VK_STENCIL_FACE_FRONT_BIT and VK_STENCIL_FACE_BACK_BIT, and specifies that both sets of stencil state are updated.

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkStencilFaceFlags;
```

VkStencilFaceFlags is a bitmask type for setting a mask of zero or more VkStencilFaceFlagBits.

To dynamically set the stencil write mask, call:

```
// Provided by VK_VERSION_1_0
void vkCmdSetStencilWriteMask(
    VkCommandBuffer                             commandBuffer,
    VkStencilFaceFlags                          faceMask,
    uint32_t                                    writeMask);
```

- commandBuffer is the command buffer into which the command will be recorded.
- faceMask is a bitmask of VkStencilFaceFlagBits specifying the set of stencil state for which to update the write mask, as described above for vkCmdSetStencilCompareMask.
- writeMask is the new value to use as the stencil write mask.

This command sets the stencil write mask for subsequent drawing commands when drawing using shader objects, or when the graphics pipeline is created with VK_DYNAMIC_STATE_STENCIL_WRITE_MASK

set in VkPipelineDynamicStateCreateInfo::pDynamicStates. Otherwise, this state is specified by the writeMask value used to create the currently active pipeline, for both VkPipelineDepthStencilStateCreateInfo::front and VkPipelineDepthStencilStateCreateInfo::back faces.

## Valid Usage (Implicit)

- VUID-vkCmdSetStencilWriteMask-commandBuffer-parameter
  commandBuffer **must** be a valid VkCommandBuffer handle
- VUID-vkCmdSetStencilWriteMask-faceMask-parameter
  faceMask **must** be a valid combination of VkStencilFaceFlagBits values
- VUID-vkCmdSetStencilWriteMask-faceMask-requiredbitmask
  faceMask **must** not be 0
- VUID-vkCmdSetStencilWriteMask-commandBuffer-recording
  commandBuffer **must** be in the recording state
- VUID-vkCmdSetStencilWriteMask-commandBuffer-cmdpool
  The VkCommandPool that commandBuffer was allocated from **must** support graphics operations
- VUID-vkCmdSetStencilWriteMask-videocoding
  This command **must** only be called outside of a video coding scope

## Host Synchronization

- Host access to commandBuffer **must** be externally synchronized
- Host access to the VkCommandPool that commandBuffer was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics | State |

To dynamically set the stencil reference value, call:

```
// Provided by VK_VERSION_1_0
void vkCmdSetStencilReference(
    VkCommandBuffer                                 commandBuffer,
    VkStencilFaceFlags                              faceMask,
    uint32_t                                        reference);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `faceMask` is a bitmask of VkStencilFaceFlagBits specifying the set of stencil state for which to update the reference value, as described above for vkCmdSetStencilCompareMask.
- `reference` is the new value to use as the stencil reference value.

This command sets the stencil reference value for subsequent drawing commands when drawing using shader objects, or when the graphics pipeline is created with `VK_DYNAMIC_STATE_STENCIL_REFERENCE` set in VkPipelineDynamicStateCreateInfo::`pDynamicStates`. Otherwise, this state is specified by the VkPipelineDepthStencilStateCreateInfo::`reference` value used to create the currently active pipeline, for both front and back faces.

## Valid Usage (Implicit)

- VUID-vkCmdSetStencilReference-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdSetStencilReference-faceMask-parameter
  `faceMask` **must** be a valid combination of VkStencilFaceFlagBits values
- VUID-vkCmdSetStencilReference-faceMask-requiredbitmask
  `faceMask` **must** not be `0`
- VUID-vkCmdSetStencilReference-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdSetStencilReference-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- VUID-vkCmdSetStencilReference-videocoding
  This command **must** only be called outside of a video coding scope

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics | State |

Possible values of the `failOp`, `passOp`, and `depthFailOp` members of VkStencilOpState, specifying what

happens to the stored stencil value if this or certain subsequent tests fail or pass, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkStencilOp {
    VK_STENCIL_OP_KEEP = 0,
    VK_STENCIL_OP_ZERO = 1,
    VK_STENCIL_OP_REPLACE = 2,
    VK_STENCIL_OP_INCREMENT_AND_CLAMP = 3,
    VK_STENCIL_OP_DECREMENT_AND_CLAMP = 4,
    VK_STENCIL_OP_INVERT = 5,
    VK_STENCIL_OP_INCREMENT_AND_WRAP = 6,
    VK_STENCIL_OP_DECREMENT_AND_WRAP = 7,
} VkStencilOp;
```

- `VK_STENCIL_OP_KEEP` keeps the current value.
- `VK_STENCIL_OP_ZERO` sets the value to 0.
- `VK_STENCIL_OP_REPLACE` sets the value to `reference`.
- `VK_STENCIL_OP_INCREMENT_AND_CLAMP` increments the current value and clamps to the maximum representable unsigned value.
- `VK_STENCIL_OP_DECREMENT_AND_CLAMP` decrements the current value and clamps to 0.
- `VK_STENCIL_OP_INVERT` bitwise-inverts the current value.
- `VK_STENCIL_OP_INCREMENT_AND_WRAP` increments the current value and wraps to 0 when the maximum value would have been exceeded.
- `VK_STENCIL_OP_DECREMENT_AND_WRAP` decrements the current value and wraps to the maximum possible value when the value would go below 0.

For purposes of increment and decrement, the stencil bits are considered as an unsigned integer.

# 29.10. Depth Test

The depth test compares the depth value $z_a$ in the depth/stencil attachment at each sample's framebuffer coordinates $(x_f,y_f)$ and sample index i against the sample's depth value $z_f$. If there is no depth attachment then the depth test is skipped.

If the render pass has a fragment density map attachment and the fragment covers multiple pixels, there is an implementation-dependent association of rasterization samples to depth attachment samples within the fragment. However, if all samples in the fragment are covered, and the depth attachment value is updated as a result of this test, all depth attachment samples will be updated.

The depth test occurs in three stages, as detailed in the following sections.

## 29.10.1. Depth Clamping and Range Adjustment

If VkPipelineRasterizationStateCreateInfo::`depthClampEnable` is enabled, $z_f$ is clamped to $[z_{min}, z_{max}]$, where $z_{min} = \min(n,f)$, $z_{max} = \max(n,f)]$, and n and f are the `minDepth` and `maxDepth` depth range values

of the viewport used by this fragment, respectively.

If VkPhysicalDeviceDepthClampZeroOneFeaturesEXT::depthClampZeroOne is enabled:

- If the depth attachment has a floating-point format and VK_EXT_depth_range_unrestricted is enabled then $z_f$ is unchanged.
- Otherwise, $z_f$ is clamped to the range [0, 1].

Otherwise:

- If $z_f$ is not in the range $[z_{min}, z_{max}]$, then $z_f$ is undefined following this step.
- If the depth attachment has a fixed-point format and $z_f$ is not in the range [0, 1], then $z_f$ is undefined following this step.

### 29.10.2. Depth Comparison

If the depth test is not enabled, as specified by vkCmdSetDepthTestEnableEXT or VkPipelineDepthStencilStateCreateInfo::depthTestEnable, then this step is skipped.

The comparison operation performed is determined by the VkCompareOp value set by vkCmdSetDepthCompareOpEXT, or by VkPipelineDepthStencilStateCreateInfo::depthCompareOp during pipeline creation. $z_f$ and $z_a$ are used as the *reference* and *test* values, respectively, in the operation specified by the VkCompareOp.

If the comparison evaluates to false, the coverage for the sample is set to 0.

### 29.10.3. Depth Attachment Writes

If depth writes are enabled, as specified by vkCmdSetDepthWriteEnableEXT or VkPipelineDepthStencilStateCreateInfo::depthWriteEnable, and the comparison evaluated to true, the depth attachment value $z_a$ is set to the sample's depth value $z_f$. If there is no depth attachment, no value is written.

To dynamically enable or disable the depth test, call:

```
// Provided by VK_EXT_extended_dynamic_state, VK_EXT_shader_object
void vkCmdSetDepthTestEnableEXT(
    VkCommandBuffer                             commandBuffer,
    VkBool32                                    depthTestEnable);
```

- commandBuffer is the command buffer into which the command will be recorded.
- depthTestEnable specifies if the depth test is enabled.

This command sets the depth test enable for subsequent drawing commands when drawing using shader objects, or when the graphics pipeline is created with VK_DYNAMIC_STATE_DEPTH_TEST_ENABLE set in VkPipelineDynamicStateCreateInfo::pDynamicStates. Otherwise, this state is specified by the VkPipelineDepthStencilStateCreateInfo::depthTestEnable value used to create the currently active pipeline.

## Valid Usage

- VUID-vkCmdSetDepthTestEnable-None-08971
  At least one of the following **must** be true:
  - the `extendedDynamicState` feature is enabled
  - the `shaderObject` feature is enabled

## Valid Usage (Implicit)

- VUID-vkCmdSetDepthTestEnable-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdSetDepthTestEnable-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdSetDepthTestEnable-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- VUID-vkCmdSetDepthTestEnable-videocoding
  This command **must** only be called outside of a video coding scope

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics | State |

To dynamically set the depth compare operator, call:

```
// Provided by VK_EXT_extended_dynamic_state, VK_EXT_shader_object
void vkCmdSetDepthCompareOpEXT(
    VkCommandBuffer                                 commandBuffer,
    VkCompareOp                                     depthCompareOp);
```

- `commandBuffer` is the command buffer into which the command will be recorded.

- `depthCompareOp` is a VkCompareOp value specifying the comparison operator used for the Depth Comparison step of the depth test.

This command sets the depth comparison operator for subsequent drawing commands when drawing using shader objects, or when the graphics pipeline is created with `VK_DYNAMIC_STATE_DEPTH_COMPARE_OP` set in VkPipelineDynamicStateCreateInfo::`pDynamicStates`. Otherwise, this state is specified by the VkPipelineDepthStencilStateCreateInfo::`depthCompareOp` value used to create the currently active pipeline.

## Valid Usage

- VUID-vkCmdSetDepthCompareOp-None-08971
  At least one of the following **must** be true:
  - the `extendedDynamicState` feature is enabled
  - the `shaderObject` feature is enabled

## Valid Usage (Implicit)

- VUID-vkCmdSetDepthCompareOp-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdSetDepthCompareOp-depthCompareOp-parameter
  `depthCompareOp` **must** be a valid VkCompareOp value
- VUID-vkCmdSetDepthCompareOp-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdSetDepthCompareOp-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- VUID-vkCmdSetDepthCompareOp-videocoding
  This command **must** only be called outside of a video coding scope

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics | State |

To dynamically set the depth write enable, call:

```
// Provided by VK_EXT_extended_dynamic_state, VK_EXT_shader_object
void vkCmdSetDepthWriteEnableEXT(
    VkCommandBuffer                                 commandBuffer,
    VkBool32                                        depthWriteEnable);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `depthWriteEnable` specifies if depth writes are enabled.

This command sets the depth write enable for subsequent drawing commands when drawing using shader objects, or when the graphics pipeline is created with `VK_DYNAMIC_STATE_DEPTH_WRITE_ENABLE` set in VkPipelineDynamicStateCreateInfo::`pDynamicStates`. Otherwise, this state is specified by the VkPipelineDepthStencilStateCreateInfo::`depthWriteEnable` value used to create the currently active pipeline.

## Valid Usage

- VUID-vkCmdSetDepthWriteEnable-None-08971
  At least one of the following **must** be true:
  - the `extendedDynamicState` feature is enabled
  - the `shaderObject` feature is enabled

## Valid Usage (Implicit)

- VUID-vkCmdSetDepthWriteEnable-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdSetDepthWriteEnable-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdSetDepthWriteEnable-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- VUID-vkCmdSetDepthWriteEnable-videocoding
  This command **must** only be called outside of a video coding scope

### Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

### Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics | State |

## 29.11. Representative Fragment Test

The representative fragment test allows implementations to reduce the amount of rasterization and fragment processing work performed for each point, line, or triangle primitive. For any primitive that produces one or more fragments that pass all prior early fragment tests, the implementation **may** choose one or more "representative" fragments for processing and discard all other fragments. For draw calls rendering multiple points, lines, or triangles arranged in lists, strips, or fans, the representative fragment test is performed independently for each of those primitives. The set of fragments discarded by the representative fragment test is implementation-dependent. In some cases, the representative fragment test may not discard any fragments for a given primitive.

If the `pNext` chain of VkGraphicsPipelineCreateInfo includes a `VkPipelineRepresentativeFragmentTestStateCreateInfoNV` structure, then that structure includes parameters controlling the representative fragment test.

The `VkPipelineRepresentativeFragmentTestStateCreateInfoNV` structure is defined as:

```
// Provided by VK_NV_representative_fragment_test
typedef struct VkPipelineRepresentativeFragmentTestStateCreateInfoNV {
    VkStructureType    sType;
    const void*        pNext;
    VkBool32           representativeFragmentTestEnable;
} VkPipelineRepresentativeFragmentTestStateCreateInfoNV;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `representativeFragmentTestEnable` controls whether the representative fragment test is enabled.

If this structure is not included in the `pNext` chain, `representativeFragmentTestEnable` is considered to be `VK_FALSE`, and the representative fragment test is disabled.

If the active fragment shader does not specify the `EarlyFragmentTests` execution mode, the representative fragment shader test has no effect, even if enabled.

### Valid Usage (Implicit)

- VUID-VkPipelineRepresentativeFragmentTestStateCreateInfoNV-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PIPELINE_REPRESENTATIVE_FRAGMENT_TEST_STATE_CREATE_INFO_NV`

To dynamically set the `representativeFragmentTestEnable` state, call:

```
// Provided by VK_EXT_extended_dynamic_state3 with VK_NV_representative_fragment_test,
VK_EXT_shader_object with VK_NV_representative_fragment_test
void vkCmdSetRepresentativeFragmentTestEnableNV(
    VkCommandBuffer                             commandBuffer,
    VkBool32                                    representativeFragmentTestEnable);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `representativeFragmentTestEnable` specifies the `representativeFragmentTestEnable` state.

This command sets the `representativeFragmentTestEnable` state for subsequent drawing commands when drawing using shader objects, or when the graphics pipeline is created with `VK_DYNAMIC_STATE_REPRESENTATIVE_FRAGMENT_TEST_ENABLE_NV` set in VkPipelineDynamicStateCreateInfo::`pDynamicStates`. Otherwise, this state is specified by the VkPipelineRepresentativeFragmentTestStateCreateInfoNV::`representativeFragmentTestEnable` value used to create the currently active pipeline.

### Valid Usage

- VUID-vkCmdSetRepresentativeFragmentTestEnableNV-None-09423
  At least one of the following **must** be true:
  - The `extendedDynamicState3RepresentativeFragmentTestEnable` feature is enabled
  - The `shaderObject` feature is enabled

### Valid Usage (Implicit)

- VUID-vkCmdSetRepresentativeFragmentTestEnableNV-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdSetRepresentativeFragmentTestEnableNV-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdSetRepresentativeFragmentTestEnableNV-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations

- VUID-vkCmdSetRepresentativeFragmentTestEnableNV-videocoding
  This command **must** only be called outside of a video coding scope

### Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

### Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics | State |

## 29.12. Sample Counting

Occlusion queries use query pool entries to track the number of samples that pass all the per-fragment tests. The mechanism of collecting an occlusion query value is described in Occlusion Queries.

The occlusion query sample counter increments by one for each sample with a coverage value of 1 in each fragment that survives all the per-fragment tests, including scissor, exclusive scissor, sample mask, alpha to coverage, stencil, and depth tests.

## 29.13. Fragment Coverage to Color

The `VkPipelineCoverageToColorStateCreateInfoNV` structure is defined as:

```
// Provided by VK_NV_fragment_coverage_to_color
typedef struct VkPipelineCoverageToColorStateCreateInfoNV {
    VkStructureType                                 sType;
    const void*                                     pNext;
    VkPipelineCoverageToColorStateCreateFlagsNV     flags;
    VkBool32                                        coverageToColorEnable;
    uint32_t                                        coverageToColorLocation;
} VkPipelineCoverageToColorStateCreateInfoNV;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is reserved for future use.

- coverageToColorEnable controls whether the fragment coverage value replaces a fragment color output.
- coverageToColorLocation controls which fragment shader color output value is replaced.

If the pNext chain of VkPipelineMultisampleStateCreateInfo includes a VkPipelineCoverageToColorStateCreateInfoNV structure, then that structure controls whether the fragment coverage is substituted for a fragment color output and, if so, which output is replaced.

If coverageToColorEnable is VK_TRUE, the coverage mask replaces the first component of the color value corresponding to the fragment shader output location with Location equal to coverageToColorLocation and Index equal to zero. If the color attachment format has fewer bits than the coverage mask, the low bits of the sample coverage mask are taken without any clamping. If the color attachment format has more bits than the coverage mask, the high bits of the sample coverage mask are filled with zeros.

If coverageToColorEnable is VK_FALSE, these operations are skipped. If this structure is not included in the pNext chain, it is as if coverageToColorEnable is VK_FALSE.

### Valid Usage

- VUID-VkPipelineCoverageToColorStateCreateInfoNV-coverageToColorEnable-01404
  If coverageToColorEnable is VK_TRUE, then the render pass subpass indicated by VkGraphicsPipelineCreateInfo::renderPass and VkGraphicsPipelineCreateInfo::subpass **must** have a color attachment at the location selected by coverageToColorLocation, with a VkFormat of VK_FORMAT_R8_UINT, VK_FORMAT_R8_SINT, VK_FORMAT_R16_UINT, VK_FORMAT_R16_SINT, VK_FORMAT_R32_UINT, or VK_FORMAT_R32_SINT

### Valid Usage (Implicit)

- VUID-VkPipelineCoverageToColorStateCreateInfoNV-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PIPELINE_COVERAGE_TO_COLOR_STATE_CREATE_INFO_NV
- VUID-VkPipelineCoverageToColorStateCreateInfoNV-flags-zerobitmask
  flags **must** be 0

```
// Provided by VK_NV_fragment_coverage_to_color
typedef VkFlags VkPipelineCoverageToColorStateCreateFlagsNV;
```

VkPipelineCoverageToColorStateCreateFlagsNV is a bitmask type for setting a mask, but is currently reserved for future use.

To dynamically set the coverageToColorEnable state, call:

```
// Provided by VK_EXT_extended_dynamic_state3 with VK_NV_fragment_coverage_to_color,
VK_EXT_shader_object with VK_NV_fragment_coverage_to_color
void vkCmdSetCoverageToColorEnableNV(
    VkCommandBuffer                             commandBuffer,
    VkBool32                                    coverageToColorEnable);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `coverageToColorEnable` specifies the `coverageToColorEnable` state.

This command sets the `coverageToColorEnable` state for subsequent drawing commands when drawing using shader objects, or when the graphics pipeline is created with `VK_DYNAMIC_STATE_COVERAGE_TO_COLOR_ENABLE_NV` set in VkPipelineDynamicStateCreateInfo::`pDynamicStates`. Otherwise, this state is specified by the VkPipelineCoverageToColorStateCreateInfoNV::`coverageToColorEnable` value used to create the currently active pipeline.

## Valid Usage

- VUID-vkCmdSetCoverageToColorEnableNV-None-09423
  At least one of the following **must** be true:
  - The `extendedDynamicState3CoverageToColorEnable` feature is enabled
  - The `shaderObject` feature is enabled

## Valid Usage (Implicit)

- VUID-vkCmdSetCoverageToColorEnableNV-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdSetCoverageToColorEnableNV-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdSetCoverageToColorEnableNV-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- VUID-vkCmdSetCoverageToColorEnableNV-videocoding
  This command **must** only be called outside of a video coding scope

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics | State |

To dynamically set the `coverageToColorLocation` state, call:

```
// Provided by VK_EXT_extended_dynamic_state3 with VK_NV_fragment_coverage_to_color,
VK_EXT_shader_object with VK_NV_fragment_coverage_to_color
void vkCmdSetCoverageToColorLocationNV(
    VkCommandBuffer                             commandBuffer,
    uint32_t                                    coverageToColorLocation);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `coverageToColorLocation` specifies the `coverageToColorLocation` state.

This command sets the `coverageToColorLocation` state for subsequent drawing commands when drawing using shader objects, or when the graphics pipeline is created with `VK_DYNAMIC_STATE_COVERAGE_TO_COLOR_LOCATION_NV` set in VkPipelineDynamicStateCreateInfo::`pDynamicStates`. Otherwise, this state is specified by the VkPipelineCoverageToColorStateCreateInfoNV::`coverageToColorLocation` value used to create the currently active pipeline.

## Valid Usage

- VUID-vkCmdSetCoverageToColorLocationNV-None-09423
  At least one of the following **must** be true:
  - The `extendedDynamicState3CoverageToColorLocation` feature is enabled
  - The `shaderObject` feature is enabled

## Valid Usage (Implicit)

- VUID-vkCmdSetCoverageToColorLocationNV-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdSetCoverageToColorLocationNV-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdSetCoverageToColorLocationNV-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- VUID-vkCmdSetCoverageToColorLocationNV-videocoding

This command **must** only be called outside of a video coding scope

### Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

### Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics | State |

# 29.14. Coverage Reduction

Coverage reduction takes the coverage information for a fragment and converts that to a boolean coverage value for each color sample in each pixel covered by the fragment.

## 29.14.1. Pixel Coverage

Coverage for each pixel is first extracted from the total fragment coverage mask. This consists of `rasterizationSamples` unique coverage samples for each pixel in the fragment area, each with a unique sample index. If the fragment only contains a single pixel, coverage for the pixel is equivalent to the fragment coverage.

If the render pass has a fragment density map attachment and the fragment covers multiple pixels, pixel coverage is generated in an implementation-dependent manner. If all samples in the fragment are covered, all samples will be covered in each pixel coverage.

If a shading rate image is used, and the fragment covers multiple pixels, each pixel's coverage consists of the coverage samples corresponding to that pixel, and each sample retains its unique sample index i.

If the fragment shading rate is set, and the fragment covers multiple pixels, each pixel's coverage consists of the coverage samples with a pixel index matching that pixel, and each sample retains its unique sample index i.

## 29.14.2. Color Sample Coverage

Once pixel coverage is determined, coverage for each individual color sample corresponding to that pixel is determined.

If the number of rasterizationSamples is identical to the number of samples in the color attachments, a color sample is covered if the pixel coverage sample with the same sample index i is covered.

Otherwise, the coverage for each color sample is computed from the pixel coverage as follows.

If the VK_AMD_mixed_attachment_samples extension is enabled, for color samples present in the color attachments, a color sample is covered if the pixel coverage sample with the same sample index i is covered; additional pixel coverage samples are discarded.

If the pNext chain of VkSubpassDescription2 or VkRenderingInfo includes a VkMultisampledRenderToSingleSampledInfoEXT structure with the multisampledRenderToSingleSampledEnable field equal to VK_TRUE, sample coverage is calculated as if the attachment has VkMultisampledRenderToSingleSampledInfoEXT::rasterizationSamples samples.

When the VK_NV_coverage_reduction_mode extension is enabled, the pipeline state controlling coverage reduction is specified through the members of the VkPipelineCoverageReductionStateCreateInfoNV structure.

The VkPipelineCoverageReductionStateCreateInfoNV structure is defined as:

```
// Provided by VK_NV_coverage_reduction_mode
typedef struct VkPipelineCoverageReductionStateCreateInfoNV {
    VkStructureType                                     sType;
    const void*                                         pNext;
    VkPipelineCoverageReductionStateCreateFlagsNV       flags;
    VkCoverageReductionModeNV                           coverageReductionMode;
} VkPipelineCoverageReductionStateCreateInfoNV;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- flags is reserved for future use.
- coverageReductionMode is a VkCoverageReductionModeNV value controlling how color sample coverage is generated from pixel coverage.

If this structure is not included in the pNext chain, or if the extension is not enabled, the default coverage reduction mode is inferred as follows:

- If the VK_NV_framebuffer_mixed_samples extension is enabled, then it is as if the coverageReductionMode is VK_COVERAGE_REDUCTION_MODE_MERGE_NV.
- If the VK_AMD_mixed_attachment_samples extension is enabled, then it is as if the coverageReductionMode is VK_COVERAGE_REDUCTION_MODE_TRUNCATE_NV.
- If both VK_NV_framebuffer_mixed_samples and VK_AMD_mixed_attachment_samples are enabled, then the default coverage reduction mode is implementation-dependent.

### Valid Usage (Implicit)

- VUID-VkPipelineCoverageReductionStateCreateInfoNV-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PIPELINE_COVERAGE_REDUCTION_STATE_CREATE_INFO_NV`
- VUID-VkPipelineCoverageReductionStateCreateInfoNV-flags-zerobitmask
  `flags` **must** be `0`
- VUID-VkPipelineCoverageReductionStateCreateInfoNV-coverageReductionMode-parameter
  `coverageReductionMode` **must** be a valid VkCoverageReductionModeNV value

```
// Provided by VK_NV_coverage_reduction_mode
typedef VkFlags VkPipelineCoverageReductionStateCreateFlagsNV;
```

`VkPipelineCoverageReductionStateCreateFlagsNV` is a bitmask type for setting a mask, but is currently reserved for future use.

Possible values of VkPipelineCoverageReductionStateCreateInfoNV::`coverageReductionMode`, specifying how color sample coverage is generated from pixel coverage, are:

```
// Provided by VK_NV_coverage_reduction_mode
typedef enum VkCoverageReductionModeNV {
    VK_COVERAGE_REDUCTION_MODE_MERGE_NV = 0,
    VK_COVERAGE_REDUCTION_MODE_TRUNCATE_NV = 1,
} VkCoverageReductionModeNV;
```

- `VK_COVERAGE_REDUCTION_MODE_MERGE_NV` specifies that each color sample will be associated with an implementation-dependent subset of samples in the pixel coverage. If any of those associated samples are covered, the color sample is covered.
- `VK_COVERAGE_REDUCTION_MODE_TRUNCATE_NV` specifies that for color samples present in the color attachments, a color sample is covered if the pixel coverage sample with the same sample index i is covered; other pixel coverage samples are discarded.

To dynamically set the `coverageReductionMode` state, call:

```
// Provided by VK_EXT_extended_dynamic_state3 with VK_NV_coverage_reduction_mode,
VK_EXT_shader_object with VK_NV_coverage_reduction_mode
void vkCmdSetCoverageReductionModeNV(
    VkCommandBuffer                             commandBuffer,
    VkCoverageReductionModeNV                   coverageReductionMode);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `coverageReductionMode` specifies the `coverageReductionMode` state.

This command sets the `coverageReductionMode` state for subsequent drawing commands when drawing using shader objects, or when the graphics pipeline is created with `VK_DYNAMIC_STATE_COVERAGE_REDUCTION_MODE_NV` set in VkPipelineDynamicStateCreateInfo::`pDynamicStates`. Otherwise, this state is specified by the VkPipelineCoverageReductionStateCreateInfoNV::`coverageReductionMode` value used to create the currently active pipeline.

## Valid Usage

- VUID-vkCmdSetCoverageReductionModeNV-None-09423
  At least one of the following **must** be true:
  - The `extendedDynamicState3CoverageReductionMode` feature is enabled
  - The `shaderObject` feature is enabled

## Valid Usage (Implicit)

- VUID-vkCmdSetCoverageReductionModeNV-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdSetCoverageReductionModeNV-coverageReductionMode-parameter
  `coverageReductionMode` **must** be a valid VkCoverageReductionModeNV value
- VUID-vkCmdSetCoverageReductionModeNV-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdSetCoverageReductionModeNV-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- VUID-vkCmdSetCoverageReductionModeNV-videocoding
  This command **must** only be called outside of a video coding scope

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics | State |

To query the set of mixed sample combinations of coverage reduction mode, rasterization samples and color, depth, stencil attachment sample counts that are supported by a physical device, call:

```
// Provided by VK_NV_coverage_reduction_mode
VkResult vkGetPhysicalDeviceSupportedFramebufferMixedSamplesCombinationsNV(
    VkPhysicalDevice                            physicalDevice,
    uint32_t*                                   pCombinationCount,
    VkFramebufferMixedSamplesCombinationNV*     pCombinations);
```

- `physicalDevice` is the physical device from which to query the set of combinations.
- `pCombinationCount` is a pointer to an integer related to the number of combinations available or queried, as described below.
- `pCombinations` is either `NULL` or a pointer to an array of VkFramebufferMixedSamplesCombinationNV values, indicating the supported combinations of coverage reduction mode, rasterization samples, and color, depth, stencil attachment sample counts.

If `pCombinations` is `NULL`, then the number of supported combinations for the given `physicalDevice` is returned in `pCombinationCount`. Otherwise, `pCombinationCount` **must** point to a variable set by the application to the number of elements in the `pCombinations` array, and on return the variable is overwritten with the number of values actually written to `pCombinations`. If the value of `pCombinationCount` is less than the number of combinations supported for the given `physicalDevice`, at most `pCombinationCount` values will be written to `pCombinations`, and `VK_INCOMPLETE` will be returned instead of `VK_SUCCESS`, to indicate that not all the supported values were returned.

## Valid Usage (Implicit)

- VUID-vkGetPhysicalDeviceSupportedFramebufferMixedSamplesCombinationsNV-physicalDevice-parameter
  `physicalDevice` **must** be a valid VkPhysicalDevice handle
- VUID-vkGetPhysicalDeviceSupportedFramebufferMixedSamplesCombinationsNV-pCombinationCount-parameter
  `pCombinationCount` **must** be a valid pointer to a `uint32_t` value
- VUID-vkGetPhysicalDeviceSupportedFramebufferMixedSamplesCombinationsNV-pCombinations-parameter
  If the value referenced by `pCombinationCount` is not `0`, and `pCombinations` is not `NULL`, `pCombinations` **must** be a valid pointer to an array of `pCombinationCount` VkFramebufferMixedSamplesCombinationNV structures

## Return Codes

**Success**

- `VK_SUCCESS`
- `VK_INCOMPLETE`

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY

The VkFramebufferMixedSamplesCombinationNV structure is defined as:

```
// Provided by VK_NV_coverage_reduction_mode
typedef struct VkFramebufferMixedSamplesCombinationNV {
    VkStructureType              sType;
    void*                        pNext;
    VkCoverageReductionModeNV    coverageReductionMode;
    VkSampleCountFlagBits        rasterizationSamples;
    VkSampleCountFlags           depthStencilSamples;
    VkSampleCountFlags           colorSamples;
} VkFramebufferMixedSamplesCombinationNV;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- coverageReductionMode is a VkCoverageReductionModeNV value specifying the coverage reduction mode.
- rasterizationSamples is a VkSampleCountFlagBits specifying the number of rasterization samples in the supported combination.
- depthStencilSamples specifies the number of samples in the depth stencil attachment in the supported combination. A value of 0 indicates the combination does not have a depth stencil attachment.
- colorSamples specifies the number of color samples in a color attachment in the supported combination. A value of 0 indicates the combination does not have a color attachment.

**Valid Usage (Implicit)**

- VUID-VkFramebufferMixedSamplesCombinationNV-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_FRAMEBUFFER_MIXED_SAMPLES_COMBINATION_NV
- VUID-VkFramebufferMixedSamplesCombinationNV-pNext-pNext
  pNext **must** be NULL

### 29.14.3. Coverage Modulation

As part of coverage reduction, fragment color values **can** also be modulated (multiplied) by a value that is a function of fraction of covered rasterization samples associated with that color sample.

Pipeline state controlling coverage modulation is specified through the members of the VkPipelineCoverageModulationStateCreateInfoNV structure.

The VkPipelineCoverageModulationStateCreateInfoNV structure is defined as:

```
// Provided by VK_NV_framebuffer_mixed_samples
typedef struct VkPipelineCoverageModulationStateCreateInfoNV {
    VkStructureType                                     sType;
    const void*                                         pNext;
    VkPipelineCoverageModulationStateCreateFlagsNV      flags;
    VkCoverageModulationModeNV                          coverageModulationMode;
    VkBool32                                            coverageModulationTableEnable;
    uint32_t                                            coverageModulationTableCount;
    const float*                                        pCoverageModulationTable;
} VkPipelineCoverageModulationStateCreateInfoNV;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- flags is reserved for future use.
- coverageModulationMode is a VkCoverageModulationModeNV value controlling which color components are modulated.
- coverageModulationTableEnable controls whether the modulation factor is looked up from a table in pCoverageModulationTable.
- coverageModulationTableCount is the number of elements in pCoverageModulationTable.
- pCoverageModulationTable is a table of modulation factors containing a value for each number of covered samples.

If coverageModulationTableEnable is VK_FALSE, then for each color sample the associated bits of the pixel coverage are counted and divided by the number of associated bits to produce a modulation factor R in the range (0,1] (a value of zero would have been killed due to a color coverage of 0). Specifically:

- N = value of rasterizationSamples
- M = value of VkAttachmentDescription::samples for any color attachments
- R = popcount(associated coverage bits) / (N / M)

If coverageModulationTableEnable is VK_TRUE, the value R is computed using a programmable lookup table. The lookup table has N / M elements, and the element of the table is selected by:

- R = pCoverageModulationTable[popcount(associated coverage bits)-1]

Note that the table does not have an entry for popcount(associated coverage bits) = 0, because such samples would have been killed.

The values of pCoverageModulationTable **may** be rounded to an implementation-dependent precision, which is at least as fine as 1 / N, and clamped to [0,1].

For each color attachment with a floating-point or normalized color format, each fragment output color value is replicated to M values which **can** each be modulated (multiplied) by that color

sample's associated value of R. Which components are modulated is controlled by coverageModulationMode.

If this structure is not included in the pNext chain, it is as if coverageModulationMode is VK_COVERAGE_MODULATION_MODE_NONE_NV.

If the coverage reduction mode is VK_COVERAGE_REDUCTION_MODE_TRUNCATE_NV, each color sample is associated with only a single coverage sample. In this case, it is as if coverageModulationMode is VK_COVERAGE_MODULATION_MODE_NONE_NV.

### Valid Usage

- VUID-VkPipelineCoverageModulationStateCreateInfoNV-coverageModulationTableEnable-01405
  If coverageModulationTableEnable is VK_TRUE, coverageModulationTableCount **must** be equal to the number of rasterization samples divided by the number of color samples in the subpass

### Valid Usage (Implicit)

- VUID-VkPipelineCoverageModulationStateCreateInfoNV-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PIPELINE_COVERAGE_MODULATION_STATE_CREATE_INFO_NV
- VUID-VkPipelineCoverageModulationStateCreateInfoNV-flags-zerobitmask
  flags **must** be 0
- VUID-VkPipelineCoverageModulationStateCreateInfoNV-coverageModulationMode-parameter
  coverageModulationMode **must** be a valid VkCoverageModulationModeNV value

```
// Provided by VK_NV_framebuffer_mixed_samples
typedef VkFlags VkPipelineCoverageModulationStateCreateFlagsNV;
```

VkPipelineCoverageModulationStateCreateFlagsNV is a bitmask type for setting a mask, but is currently reserved for future use.

Possible values of VkPipelineCoverageModulationStateCreateInfoNV::coverageModulationMode, specifying which color components are modulated, are:

```
// Provided by VK_NV_framebuffer_mixed_samples
typedef enum VkCoverageModulationModeNV {
    VK_COVERAGE_MODULATION_MODE_NONE_NV = 0,
    VK_COVERAGE_MODULATION_MODE_RGB_NV = 1,
    VK_COVERAGE_MODULATION_MODE_ALPHA_NV = 2,
    VK_COVERAGE_MODULATION_MODE_RGBA_NV = 3,
} VkCoverageModulationModeNV;
```

- VK_COVERAGE_MODULATION_MODE_NONE_NV specifies that no components are multiplied by the modulation factor.
- VK_COVERAGE_MODULATION_MODE_RGB_NV specifies that the red, green, and blue components are multiplied by the modulation factor.
- VK_COVERAGE_MODULATION_MODE_ALPHA_NV specifies that the alpha component is multiplied by the modulation factor.
- VK_COVERAGE_MODULATION_MODE_RGBA_NV specifies that all components are multiplied by the modulation factor.

To dynamically set the coverageModulationMode state, call:

```
// Provided by VK_EXT_extended_dynamic_state3 with VK_NV_framebuffer_mixed_samples,
VK_EXT_shader_object with VK_NV_framebuffer_mixed_samples
void vkCmdSetCoverageModulationModeNV(
    VkCommandBuffer                             commandBuffer,
    VkCoverageModulationModeNV                  coverageModulationMode);
```

- commandBuffer is the command buffer into which the command will be recorded.
- coverageModulationMode specifies the coverageModulationMode state.

This command sets the coverageModulationMode state for subsequent drawing commands when drawing using shader objects, or the graphics pipeline is created with VK_DYNAMIC_STATE_COVERAGE_MODULATION_MODE_NV set in VkPipelineDynamicStateCreateInfo::pDynamicStates. Otherwise, this state is specified by the VkPipelineCoverageModulationStateCreateInfoNV::coverageModulationMode value used to create the currently active pipeline.

## Valid Usage

- VUID-vkCmdSetCoverageModulationModeNV-None-09423
  At least one of the following **must** be true:
  - The extendedDynamicState3CoverageModulationMode feature is enabled
  - The shaderObject feature is enabled

## Valid Usage (Implicit)

- VUID-vkCmdSetCoverageModulationModeNV-commandBuffer-parameter
  commandBuffer **must** be a valid VkCommandBuffer handle
- VUID-vkCmdSetCoverageModulationModeNV-coverageModulationMode-parameter
  coverageModulationMode **must** be a valid VkCoverageModulationModeNV value
- VUID-vkCmdSetCoverageModulationModeNV-commandBuffer-recording
  commandBuffer **must** be in the recording state
- VUID-vkCmdSetCoverageModulationModeNV-commandBuffer-cmdpool

The VkCommandPool that commandBuffer was allocated from **must** support graphics operations

- VUID-vkCmdSetCoverageModulationModeNV-videocoding
  This command **must** only be called outside of a video coding scope

## Host Synchronization

- Host access to commandBuffer **must** be externally synchronized
- Host access to the VkCommandPool that commandBuffer was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics | State |

To dynamically set the coverageModulationTableEnable state, call:

```
// Provided by VK_EXT_extended_dynamic_state3 with VK_NV_framebuffer_mixed_samples,
VK_EXT_shader_object with VK_NV_framebuffer_mixed_samples
void vkCmdSetCoverageModulationTableEnableNV(
    VkCommandBuffer                                 commandBuffer,
    VkBool32                                        coverageModulationTableEnable);
```

- commandBuffer is the command buffer into which the command will be recorded.
- coverageModulationTableEnable specifies the coverageModulationTableEnable state.

This command sets the coverageModulationTableEnable state for subsequent drawing commands when drawing using shader objects, or when the graphics pipeline is created with VK_DYNAMIC_STATE_COVERAGE_MODULATION_TABLE_ENABLE_NV set in VkPipelineDynamicStateCreateInfo::pDynamicStates. Otherwise, this state is specified by the VkPipelineCoverageModulationStateCreateInfoNV::coverageModulationTableEnable value used to create the currently active pipeline.

## Valid Usage

- VUID-vkCmdSetCoverageModulationTableEnableNV-None-09423
  At least one of the following **must** be true:
  - The extendedDynamicState3CoverageModulationTableEnable feature is enabled
  - The shaderObject feature is enabled

## Valid Usage (Implicit)

- VUID-vkCmdSetCoverageModulationTableEnableNV-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdSetCoverageModulationTableEnableNV-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdSetCoverageModulationTableEnableNV-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- VUID-vkCmdSetCoverageModulationTableEnableNV-videocoding
  This command **must** only be called outside of a video coding scope

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics | State |

To dynamically set the `pCoverageModulationTable` state, call:

```
// Provided by VK_EXT_extended_dynamic_state3 with VK_NV_framebuffer_mixed_samples,
VK_EXT_shader_object with VK_NV_framebuffer_mixed_samples
void vkCmdSetCoverageModulationTableNV(
    VkCommandBuffer                             commandBuffer,
    uint32_t                                    coverageModulationTableCount,
    const float*                                pCoverageModulationTable);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `coverageModulationTableCount` specifies the number of elements in `pCoverageModulationTable`.
- `pCoverageModulationTable` specifies the table of modulation factors containing a value for each number of covered samples.

This command sets the table of modulation factors for subsequent drawing commands when drawing using shader objects, or when the graphics pipeline is created with

VK_DYNAMIC_STATE_COVERAGE_MODULATION_TABLE_NV set in VkPipelineDynamicStateCreateInfo::pDynamicStates. Otherwise, this state is specified by the VkPipelineCoverageModulationStateCreateInfoNV::coverageModulationTableCount, and VkPipelineCoverageModulationStateCreateInfoNV::pCoverageModulationTable values used to create the currently active pipeline.

## Valid Usage

- VUID-vkCmdSetCoverageModulationTableNV-None-09423
  At least one of the following **must** be true:
  - The extendedDynamicState3CoverageModulationTable feature is enabled
  - The shaderObject feature is enabled

## Valid Usage (Implicit)

- VUID-vkCmdSetCoverageModulationTableNV-commandBuffer-parameter
  commandBuffer **must** be a valid VkCommandBuffer handle
- VUID-vkCmdSetCoverageModulationTableNV-pCoverageModulationTable-parameter
  pCoverageModulationTable **must** be a valid pointer to an array of coverageModulationTableCount float values
- VUID-vkCmdSetCoverageModulationTableNV-commandBuffer-recording
  commandBuffer **must** be in the recording state
- VUID-vkCmdSetCoverageModulationTableNV-commandBuffer-cmdpool
  The VkCommandPool that commandBuffer was allocated from **must** support graphics operations
- VUID-vkCmdSetCoverageModulationTableNV-videocoding
  This command **must** only be called outside of a video coding scope
- VUID-vkCmdSetCoverageModulationTableNV-coverageModulationTableCount-arraylength
  coverageModulationTableCount **must** be greater than 0

## Host Synchronization

- Host access to commandBuffer **must** be externally synchronized
- Host access to the VkCommandPool that commandBuffer was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary Secondary | Both | Outside | Graphics | State |

# Chapter 30. The Framebuffer

## 30.1. Blending

Blending combines the incoming *source* fragment's R, G, B, and A values with the *destination* R, G, B, and A values of each sample stored in the framebuffer at the fragment's $(x_f,y_f)$ location. Blending is performed for each color sample covered by the fragment, rather than just once for each fragment.

Source and destination values are combined according to the blend operation, quadruplets of source and destination weighting factors determined by the blend factors, and a blend constant, to obtain a new set of R, G, B, and A values, as described below.

Blending is computed and applied separately to each color attachment used by the subpass, with separate controls for each attachment.

Prior to performing the blend operation, signed and unsigned normalized fixed-point color components undergo an implied conversion to floating-point as specified by Conversion from Normalized Fixed-Point to Floating-Point. Blending computations are treated as if carried out in floating-point, and basic blend operations are performed with a precision and dynamic range no lower than that used to represent destination components. Advanced blending operations are performed with a precision and dynamic range no lower than the smaller of that used to represent destination components or that used to represent 16-bit floating-point values.

| | |
|---|---|
| **NOTE** | Blending is only defined for floating-point, UNORM, SNORM, and sRGB formats. Within those formats, the implementation may only support blending on some subset of them. Which formats support blending is indicated by `VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT`. |

The pipeline blend state is included in the `VkPipelineColorBlendStateCreateInfo` structure during graphics pipeline creation:

The `VkPipelineColorBlendStateCreateInfo` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkPipelineColorBlendStateCreateInfo {
    VkStructureType                               sType;
    const void*                                   pNext;
    VkPipelineColorBlendStateCreateFlags          flags;
    VkBool32                                      logicOpEnable;
    VkLogicOp                                     logicOp;
    uint32_t                                      attachmentCount;
    const VkPipelineColorBlendAttachmentState*    pAttachments;
    float                                         blendConstants[4];
} VkPipelineColorBlendStateCreateInfo;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.

- flags is a bitmask of VkPipelineColorBlendStateCreateFlagBits specifying additional color blending information.
- logicOpEnable controls whether to apply Logical Operations.
- logicOp selects which logical operation to apply.
- attachmentCount is the number of VkPipelineColorBlendAttachmentState elements in pAttachments. It is ignored if the pipeline is created with VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT, VK_DYNAMIC_STATE_COLOR_BLEND_EQUATION_EXT, and VK_DYNAMIC_STATE_COLOR_WRITE_MASK_EXT dynamic states set, and either VK_DYNAMIC_STATE_COLOR_BLEND_ADVANCED_EXT set or advancedBlendCoherentOperations is not enabled on the device.
- pAttachments is a pointer to an array of VkPipelineColorBlendAttachmentState structures defining blend state for each color attachment. It is ignored if the pipeline is created with VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT, VK_DYNAMIC_STATE_COLOR_BLEND_EQUATION_EXT, and VK_DYNAMIC_STATE_COLOR_WRITE_MASK_EXT dynamic states set, and either VK_DYNAMIC_STATE_COLOR_BLEND_ADVANCED_EXT set or advancedBlendCoherentOperations is not enabled on the device.
- blendConstants is a pointer to an array of four values used as the R, G, B, and A components of the blend constant that are used in blending, depending on the blend factor.

## Valid Usage

- VUID-VkPipelineColorBlendStateCreateInfo-pAttachments-00605
  If the independentBlend feature is not enabled, all elements of pAttachments **must** be identical
- VUID-VkPipelineColorBlendStateCreateInfo-logicOpEnable-00606
  If the logicOp feature is not enabled, logicOpEnable **must** be VK_FALSE
- VUID-VkPipelineColorBlendStateCreateInfo-logicOpEnable-00607
  If logicOpEnable is VK_TRUE, logicOp **must** be a valid VkLogicOp value
- VUID-VkPipelineColorBlendStateCreateInfo-rasterizationOrderColorAttachmentAccess-06465
  If the rasterizationOrderColorAttachmentAccess feature is not enabled, flags **must** not include VK_PIPELINE_COLOR_BLEND_STATE_CREATE_RASTERIZATION_ORDER_ATTACHMENT_ACCESS_BIT_EXT
- VUID-VkPipelineColorBlendStateCreateInfo-pAttachments-07353
  If attachmentCount is not 0 , and any of VK_DYNAMIC_STATE_COLOR_BLEND_ADVANCED_EXT, VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT, VK_DYNAMIC_STATE_COLOR_BLEND_EQUATION_EXT, or VK_DYNAMIC_STATE_COLOR_WRITE_MASK_EXT are not set, pAttachments **must** be a valid pointer to an array of attachmentCount valid VkPipelineColorBlendAttachmentState structures

## Valid Usage (Implicit)

- VUID-VkPipelineColorBlendStateCreateInfo-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PIPELINE_COLOR_BLEND_STATE_CREATE_INFO

- VUID-VkPipelineColorBlendStateCreateInfo-pNext-pNext
  Each pNext member of any structure (including this one) in the pNext chain **must** be either NULL or a pointer to a valid instance of VkPipelineColorBlendAdvancedStateCreateInfoEXT or VkPipelineColorWriteCreateInfoEXT
- VUID-VkPipelineColorBlendStateCreateInfo-sType-unique
  The sType value of each struct in the pNext chain **must** be unique
- VUID-VkPipelineColorBlendStateCreateInfo-flags-parameter
  flags **must** be a valid combination of VkPipelineColorBlendStateCreateFlagBits values
- VUID-VkPipelineColorBlendStateCreateInfo-pAttachments-parameter
  If attachmentCount is not 0, and pAttachments is not NULL, pAttachments **must** be a valid pointer to an array of attachmentCount valid VkPipelineColorBlendAttachmentState structures

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkPipelineColorBlendStateCreateFlags;
```

VkPipelineColorBlendStateCreateFlags is a bitmask type for setting a mask of zero or more VkPipelineColorBlendStateCreateFlagBits.

Bits which **can** be set in the VkPipelineColorBlendStateCreateInfo::flags parameter are:

```
// Provided by VK_EXT_rasterization_order_attachment_access
typedef enum VkPipelineColorBlendStateCreateFlagBits {
  // Provided by VK_EXT_rasterization_order_attachment_access
    VK_PIPELINE_COLOR_BLEND_STATE_CREATE_RASTERIZATION_ORDER_ATTACHMENT_ACCESS_BIT_EXT
= 0x00000001,
  // Provided by VK_ARM_rasterization_order_attachment_access
    VK_PIPELINE_COLOR_BLEND_STATE_CREATE_RASTERIZATION_ORDER_ATTACHMENT_ACCESS_BIT_ARM
= VK_PIPELINE_COLOR_BLEND_STATE_CREATE_RASTERIZATION_ORDER_ATTACHMENT_ACCESS_BIT_EXT,
} VkPipelineColorBlendStateCreateFlagBits;
```

- VK_PIPELINE_COLOR_BLEND_STATE_CREATE_RASTERIZATION_ORDER_ATTACHMENT_ACCESS_BIT_EXT indicates that access to color and input attachments will have implicit framebuffer-local memory dependencies, allowing applications to express custom blending operations in a fragment shader.

When VK_PIPELINE_COLOR_BLEND_STATE_CREATE_RASTERIZATION_ORDER_ATTACHMENT_ACCESS_BIT_EXT is included in a pipeline, it forms a framebuffer-local memory dependency for each fragment generated by draw commands for that pipeline with the following scopes:

- The first synchronization scope includes the VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT pipeline stage executed by all previous fragments (as defined by primitive order) in the corresponding framebuffer regions including those generated by the same draw command.
- The second synchronization scope includes the VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT pipeline stage executed by the generated fragment.

- The first access scope includes all writes to color attachments.
- The second access scope includes all reads from input attachments.

The VkPipelineColorBlendAttachmentState structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkPipelineColorBlendAttachmentState {
    VkBool32                 blendEnable;
    VkBlendFactor            srcColorBlendFactor;
    VkBlendFactor            dstColorBlendFactor;
    VkBlendOp                colorBlendOp;
    VkBlendFactor            srcAlphaBlendFactor;
    VkBlendFactor            dstAlphaBlendFactor;
    VkBlendOp                alphaBlendOp;
    VkColorComponentFlags    colorWriteMask;
} VkPipelineColorBlendAttachmentState;
```

- blendEnable controls whether blending is enabled for the corresponding color attachment. If blending is not enabled, the source fragment's color for that attachment is passed through unmodified.
- srcColorBlendFactor selects which blend factor is used to determine the source factors ($S_r$,$S_g$,$S_b$).
- dstColorBlendFactor selects which blend factor is used to determine the destination factors ($D_r$,$D_g$,$D_b$).
- colorBlendOp selects which blend operation is used to calculate the RGB values to write to the color attachment.
- srcAlphaBlendFactor selects which blend factor is used to determine the source factor $S_a$.
- dstAlphaBlendFactor selects which blend factor is used to determine the destination factor $D_a$.
- alphaBlendOp selects which blend operation is used to calculate the alpha values to write to the color attachment.
- colorWriteMask is a bitmask of VkColorComponentFlagBits specifying which of the R, G, B, and/or A components are enabled for writing, as described for the Color Write Mask.

## Valid Usage

- VUID-VkPipelineColorBlendAttachmentState-srcColorBlendFactor-00608
  If the dualSrcBlend feature is not enabled, srcColorBlendFactor **must** not be VK_BLEND_FACTOR_SRC1_COLOR, VK_BLEND_FACTOR_ONE_MINUS_SRC1_COLOR, VK_BLEND_FACTOR_SRC1_ALPHA, or VK_BLEND_FACTOR_ONE_MINUS_SRC1_ALPHA
- VUID-VkPipelineColorBlendAttachmentState-dstColorBlendFactor-00609
  If the dualSrcBlend feature is not enabled, dstColorBlendFactor **must** not be VK_BLEND_FACTOR_SRC1_COLOR, VK_BLEND_FACTOR_ONE_MINUS_SRC1_COLOR, VK_BLEND_FACTOR_SRC1_ALPHA, or VK_BLEND_FACTOR_ONE_MINUS_SRC1_ALPHA
- VUID-VkPipelineColorBlendAttachmentState-srcAlphaBlendFactor-00610

If the dualSrcBlend feature is not enabled, srcAlphaBlendFactor **must** not be VK_BLEND_FACTOR_SRC1_COLOR, VK_BLEND_FACTOR_ONE_MINUS_SRC1_COLOR, VK_BLEND_FACTOR_SRC1_ALPHA, or VK_BLEND_FACTOR_ONE_MINUS_SRC1_ALPHA

- VUID-VkPipelineColorBlendAttachmentState-dstAlphaBlendFactor-00611
  If the dualSrcBlend feature is not enabled, dstAlphaBlendFactor **must** not be VK_BLEND_FACTOR_SRC1_COLOR, VK_BLEND_FACTOR_ONE_MINUS_SRC1_COLOR, VK_BLEND_FACTOR_SRC1_ALPHA, or VK_BLEND_FACTOR_ONE_MINUS_SRC1_ALPHA
- VUID-VkPipelineColorBlendAttachmentState-colorBlendOp-01406
  If either of colorBlendOp or alphaBlendOp is an advanced blend operation, then colorBlendOp **must** equal alphaBlendOp
- VUID-VkPipelineColorBlendAttachmentState-advancedBlendIndependentBlend-01407
  If VkPhysicalDeviceBlendOperationAdvancedPropertiesEXT::advancedBlendIndependentBlend is VK_FALSE and colorBlendOp is an advanced blend operation, then colorBlendOp **must** be the same for all attachments
- VUID-VkPipelineColorBlendAttachmentState-advancedBlendIndependentBlend-01408
  If VkPhysicalDeviceBlendOperationAdvancedPropertiesEXT::advancedBlendIndependentBlend is VK_FALSE and alphaBlendOp is an advanced blend operation, then alphaBlendOp **must** be the same for all attachments
- VUID-VkPipelineColorBlendAttachmentState-advancedBlendAllOperations-01409
  If VkPhysicalDeviceBlendOperationAdvancedPropertiesEXT::advancedBlendAllOperations is VK_FALSE, then colorBlendOp **must** not be VK_BLEND_OP_ZERO_EXT, VK_BLEND_OP_SRC_EXT, VK_BLEND_OP_DST_EXT, VK_BLEND_OP_SRC_OVER_EXT, VK_BLEND_OP_DST_OVER_EXT, VK_BLEND_OP_SRC_IN_EXT, VK_BLEND_OP_DST_IN_EXT, VK_BLEND_OP_SRC_OUT_EXT, VK_BLEND_OP_DST_OUT_EXT, VK_BLEND_OP_SRC_ATOP_EXT, VK_BLEND_OP_DST_ATOP_EXT, VK_BLEND_OP_XOR_EXT, VK_BLEND_OP_INVERT_EXT, VK_BLEND_OP_INVERT_RGB_EXT, VK_BLEND_OP_LINEARDODGE_EXT, VK_BLEND_OP_LINEARBURN_EXT, VK_BLEND_OP_VIVIDLIGHT_EXT, VK_BLEND_OP_LINEARLIGHT_EXT, VK_BLEND_OP_PINLIGHT_EXT, VK_BLEND_OP_HARDMIX_EXT, VK_BLEND_OP_PLUS_EXT, VK_BLEND_OP_PLUS_CLAMPED_EXT, VK_BLEND_OP_PLUS_CLAMPED_ALPHA_EXT, VK_BLEND_OP_PLUS_DARKER_EXT, VK_BLEND_OP_MINUS_EXT, VK_BLEND_OP_MINUS_CLAMPED_EXT, VK_BLEND_OP_CONTRAST_EXT, VK_BLEND_OP_INVERT_OVG_EXT, VK_BLEND_OP_RED_EXT, VK_BLEND_OP_GREEN_EXT, or VK_BLEND_OP_BLUE_EXT
- VUID-VkPipelineColorBlendAttachmentState-colorBlendOp-01410
  If colorBlendOp or alphaBlendOp is an advanced blend operation, then colorAttachmentCount of the subpass this pipeline is compiled against **must** be less than or equal to VkPhysicalDeviceBlendOperationAdvancedPropertiesEXT::advancedBlendMaxColorAttachments
- VUID-VkPipelineColorBlendAttachmentState-constantAlphaColorBlendFactors-04454
  If the VK_KHR_portability_subset extension is enabled, and VkPhysicalDevicePortabilitySubsetFeaturesKHR::constantAlphaColorBlendFactors is VK_FALSE, srcColorBlendFactor **must** not be VK_BLEND_FACTOR_CONSTANT_ALPHA or VK_BLEND_FACTOR_ONE_MINUS_CONSTANT_ALPHA
- VUID-VkPipelineColorBlendAttachmentState-constantAlphaColorBlendFactors-04455
  If the VK_KHR_portability_subset extension is enabled, and VkPhysicalDevicePortabilitySubsetFeaturesKHR::constantAlphaColorBlendFactors is VK_FALSE, dstColorBlendFactor **must** not be VK_BLEND_FACTOR_CONSTANT_ALPHA or

```
    VK_BLEND_FACTOR_ONE_MINUS_CONSTANT_ALPHA
```

### Valid Usage (Implicit)

- VUID-VkPipelineColorBlendAttachmentState-srcColorBlendFactor-parameter `srcColorBlendFactor` **must** be a valid VkBlendFactor value
- VUID-VkPipelineColorBlendAttachmentState-dstColorBlendFactor-parameter `dstColorBlendFactor` **must** be a valid VkBlendFactor value
- VUID-VkPipelineColorBlendAttachmentState-colorBlendOp-parameter `colorBlendOp` **must** be a valid VkBlendOp value
- VUID-VkPipelineColorBlendAttachmentState-srcAlphaBlendFactor-parameter `srcAlphaBlendFactor` **must** be a valid VkBlendFactor value
- VUID-VkPipelineColorBlendAttachmentState-dstAlphaBlendFactor-parameter `dstAlphaBlendFactor` **must** be a valid VkBlendFactor value
- VUID-VkPipelineColorBlendAttachmentState-alphaBlendOp-parameter `alphaBlendOp` **must** be a valid VkBlendOp value
- VUID-VkPipelineColorBlendAttachmentState-colorWriteMask-parameter `colorWriteMask` **must** be a valid combination of VkColorComponentFlagBits values

To dynamically set `blendEnable`, call:

```
// Provided by VK_EXT_extended_dynamic_state3, VK_EXT_shader_object
void vkCmdSetColorBlendEnableEXT(
    VkCommandBuffer                             commandBuffer,
    uint32_t                                    firstAttachment,
    uint32_t                                    attachmentCount,
    const VkBool32*                             pColorBlendEnables);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `firstAttachment` the first color attachment the color blending enable applies.
- `attachmentCount` the number of color blending enables in the `pColorBlendEnables` array.
- `pColorBlendEnables` an array of booleans to indicate whether color blending is enabled for the corresponding attachment.

This command sets the color blending enable of the specified color attachments for subsequent drawing commands when drawing using shader objects, or when the graphics pipeline is created with `VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT` set in VkPipelineDynamicStateCreateInfo::`pDynamicStates`. Otherwise, this state is specified by the VkPipelineColorBlendAttachmentState::`blendEnable` values used to create the currently active pipeline.

## Valid Usage

- VUID-vkCmdSetColorBlendEnableEXT-None-09423
  At least one of the following **must** be true:
  - The `extendedDynamicState3ColorBlendEnable` feature is enabled
  - The `shaderObject` feature is enabled

## Valid Usage (Implicit)

- VUID-vkCmdSetColorBlendEnableEXT-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdSetColorBlendEnableEXT-pColorBlendEnables-parameter
  `pColorBlendEnables` **must** be a valid pointer to an array of `attachmentCount` VkBool32 values
- VUID-vkCmdSetColorBlendEnableEXT-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdSetColorBlendEnableEXT-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- VUID-vkCmdSetColorBlendEnableEXT-videocoding
  This command **must** only be called outside of a video coding scope
- VUID-vkCmdSetColorBlendEnableEXT-attachmentCount-arraylength
  `attachmentCount` **must** be greater than `0`

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics | State |

To dynamically set color blend factors and operations, call:

```
// Provided by VK_EXT_extended_dynamic_state3, VK_EXT_shader_object
void vkCmdSetColorBlendEquationEXT(
    VkCommandBuffer                             commandBuffer,
    uint32_t                                    firstAttachment,
    uint32_t                                    attachmentCount,
    const VkColorBlendEquationEXT*              pColorBlendEquations);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `firstAttachment` the first color attachment the color blend factors and operations apply to.
- `attachmentCount` the number of VkColorBlendEquationEXT elements in the `pColorBlendEquations` array.
- `pColorBlendEquations` an array of VkColorBlendEquationEXT structs that specify the color blend factors and operations for the corresponding attachments.

This command sets the color blending factors and operations of the specified attachments for subsequent drawing commands when drawing using shader objects, or when the graphics pipeline is created with `VK_DYNAMIC_STATE_COLOR_BLEND_EQUATION_EXT` set in VkPipelineDynamicStateCreateInfo::`pDynamicStates`. Otherwise, this state is specified by the VkPipelineColorBlendAttachmentState::`srcColorBlendFactor`, VkPipelineColorBlendAttachmentState::`dstColorBlendFactor`, VkPipelineColorBlendAttachmentState::`colorBlendOp`, VkPipelineColorBlendAttachmentState::`srcAlphaBlendFactor`, VkPipelineColorBlendAttachmentState::`dstAlphaBlendFactor`, and VkPipelineColorBlendAttachmentState::`alphaBlendOp` values used to create the currently active pipeline.

## Valid Usage

- VUID-vkCmdSetColorBlendEquationEXT-None-09423
  At least one of the following **must** be true:
  - The `extendedDynamicState3ColorBlendEquation` feature is enabled
  - The `shaderObject` feature is enabled

## Valid Usage (Implicit)

- VUID-vkCmdSetColorBlendEquationEXT-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdSetColorBlendEquationEXT-pColorBlendEquations-parameter
  `pColorBlendEquations` **must** be a valid pointer to an array of `attachmentCount` valid VkColorBlendEquationEXT structures
- VUID-vkCmdSetColorBlendEquationEXT-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdSetColorBlendEquationEXT-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics

operations

- VUID-vkCmdSetColorBlendEquationEXT-videocoding
  This command **must** only be called outside of a video coding scope
- VUID-vkCmdSetColorBlendEquationEXT-attachmentCount-arraylength
  `attachmentCount` **must** be greater than `0`

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics | State |

The `VkColorBlendEquationEXT` structure is defined as:

```
// Provided by VK_EXT_extended_dynamic_state3, VK_EXT_shader_object
typedef struct VkColorBlendEquationEXT {
    VkBlendFactor    srcColorBlendFactor;
    VkBlendFactor    dstColorBlendFactor;
    VkBlendOp        colorBlendOp;
    VkBlendFactor    srcAlphaBlendFactor;
    VkBlendFactor    dstAlphaBlendFactor;
    VkBlendOp        alphaBlendOp;
} VkColorBlendEquationEXT;
```

- `srcColorBlendFactor` selects which blend factor is used to determine the source factors ($S_r$,$S_g$,$S_b$).
- `dstColorBlendFactor` selects which blend factor is used to determine the destination factors ($D_r$,$D_g$,$D_b$).
- `colorBlendOp` selects which blend operation is used to calculate the RGB values to write to the color attachment.
- `srcAlphaBlendFactor` selects which blend factor is used to determine the source factor $S_a$.
- `dstAlphaBlendFactor` selects which blend factor is used to determine the destination factor $D_a$.
- `alphaBlendOp` selects which blend operation is use to calculate the alpha values to write to the color attachment.

## Valid Usage

- VUID-VkColorBlendEquationEXT-dualSrcBlend-07357
  If the `dualSrcBlend` feature is not enabled, `srcColorBlendFactor` **must** not be `VK_BLEND_FACTOR_SRC1_COLOR`, `VK_BLEND_FACTOR_ONE_MINUS_SRC1_COLOR`, `VK_BLEND_FACTOR_SRC1_ALPHA`, or `VK_BLEND_FACTOR_ONE_MINUS_SRC1_ALPHA`
- VUID-VkColorBlendEquationEXT-dualSrcBlend-07358
  If the `dualSrcBlend` feature is not enabled, `dstColorBlendFactor` **must** not be `VK_BLEND_FACTOR_SRC1_COLOR`, `VK_BLEND_FACTOR_ONE_MINUS_SRC1_COLOR`, `VK_BLEND_FACTOR_SRC1_ALPHA`, or `VK_BLEND_FACTOR_ONE_MINUS_SRC1_ALPHA`
- VUID-VkColorBlendEquationEXT-dualSrcBlend-07359
  If the `dualSrcBlend` feature is not enabled, `srcAlphaBlendFactor` **must** not be `VK_BLEND_FACTOR_SRC1_COLOR`, `VK_BLEND_FACTOR_ONE_MINUS_SRC1_COLOR`, `VK_BLEND_FACTOR_SRC1_ALPHA`, or `VK_BLEND_FACTOR_ONE_MINUS_SRC1_ALPHA`
- VUID-VkColorBlendEquationEXT-dualSrcBlend-07360
  If the `dualSrcBlend` feature is not enabled, `dstAlphaBlendFactor` **must** not be `VK_BLEND_FACTOR_SRC1_COLOR`, `VK_BLEND_FACTOR_ONE_MINUS_SRC1_COLOR`, `VK_BLEND_FACTOR_SRC1_ALPHA`, or `VK_BLEND_FACTOR_ONE_MINUS_SRC1_ALPHA`
- VUID-VkColorBlendEquationEXT-colorBlendOp-07361
  `colorBlendOp` and `alphaBlendOp` **must** not be `VK_BLEND_OP_ZERO_EXT`, `VK_BLEND_OP_SRC_EXT`, `VK_BLEND_OP_DST_EXT`, `VK_BLEND_OP_SRC_OVER_EXT`, `VK_BLEND_OP_DST_OVER_EXT`, `VK_BLEND_OP_SRC_IN_EXT`, `VK_BLEND_OP_DST_IN_EXT`, `VK_BLEND_OP_SRC_OUT_EXT`, `VK_BLEND_OP_DST_OUT_EXT`, `VK_BLEND_OP_SRC_ATOP_EXT`, `VK_BLEND_OP_DST_ATOP_EXT`, `VK_BLEND_OP_XOR_EXT`, `VK_BLEND_OP_MULTIPLY_EXT`, `VK_BLEND_OP_SCREEN_EXT`, `VK_BLEND_OP_OVERLAY_EXT`, `VK_BLEND_OP_DARKEN_EXT`, `VK_BLEND_OP_LIGHTEN_EXT`, `VK_BLEND_OP_COLORDODGE_EXT`, `VK_BLEND_OP_COLORBURN_EXT`, `VK_BLEND_OP_HARDLIGHT_EXT`, `VK_BLEND_OP_SOFTLIGHT_EXT`, `VK_BLEND_OP_DIFFERENCE_EXT`, `VK_BLEND_OP_EXCLUSION_EXT`, `VK_BLEND_OP_INVERT_EXT`, `VK_BLEND_OP_INVERT_RGB_EXT`, `VK_BLEND_OP_LINEARDODGE_EXT`, `VK_BLEND_OP_LINEARBURN_EXT`, `VK_BLEND_OP_VIVIDLIGHT_EXT`, `VK_BLEND_OP_LINEARLIGHT_EXT`, `VK_BLEND_OP_PINLIGHT_EXT`, `VK_BLEND_OP_HARDMIX_EXT`, `VK_BLEND_OP_HSL_HUE_EXT`, `VK_BLEND_OP_HSL_SATURATION_EXT`, `VK_BLEND_OP_HSL_COLOR_EXT`, `VK_BLEND_OP_HSL_LUMINOSITY_EXT`, `VK_BLEND_OP_PLUS_EXT`, `VK_BLEND_OP_PLUS_CLAMPED_EXT`, `VK_BLEND_OP_PLUS_CLAMPED_ALPHA_EXT`, `VK_BLEND_OP_PLUS_DARKER_EXT`, `VK_BLEND_OP_MINUS_EXT`, `VK_BLEND_OP_MINUS_CLAMPED_EXT`, `VK_BLEND_OP_CONTRAST_EXT`, `VK_BLEND_OP_INVERT_OVG_EXT`, `VK_BLEND_OP_RED_EXT`, `VK_BLEND_OP_GREEN_EXT`, or `VK_BLEND_OP_BLUE_EXT`
- VUID-VkColorBlendEquationEXT-constantAlphaColorBlendFactors-07362
  If the `VK_KHR_portability_subset` extension is enabled, and VkPhysicalDevicePortabilitySubsetFeaturesKHR::`constantAlphaColorBlendFactors` is `VK_FALSE`, `srcColorBlendFactor` **must** not be `VK_BLEND_FACTOR_CONSTANT_ALPHA` or `VK_BLEND_FACTOR_ONE_MINUS_CONSTANT_ALPHA`
- VUID-VkColorBlendEquationEXT-constantAlphaColorBlendFactors-07363
  If the `VK_KHR_portability_subset` extension is enabled, and VkPhysicalDevicePortabilitySubsetFeaturesKHR::`constantAlphaColorBlendFactors` is `VK_FALSE`, `dstColorBlendFactor` **must** not be `VK_BLEND_FACTOR_CONSTANT_ALPHA` or `VK_BLEND_FACTOR_ONE_MINUS_CONSTANT_ALPHA`

### Valid Usage (Implicit)

- VUID-VkColorBlendEquationEXT-srcColorBlendFactor-parameter
  srcColorBlendFactor **must** be a valid VkBlendFactor value
- VUID-VkColorBlendEquationEXT-dstColorBlendFactor-parameter
  dstColorBlendFactor **must** be a valid VkBlendFactor value
- VUID-VkColorBlendEquationEXT-colorBlendOp-parameter
  colorBlendOp **must** be a valid VkBlendOp value
- VUID-VkColorBlendEquationEXT-srcAlphaBlendFactor-parameter
  srcAlphaBlendFactor **must** be a valid VkBlendFactor value
- VUID-VkColorBlendEquationEXT-dstAlphaBlendFactor-parameter
  dstAlphaBlendFactor **must** be a valid VkBlendFactor value
- VUID-VkColorBlendEquationEXT-alphaBlendOp-parameter
  alphaBlendOp **must** be a valid VkBlendOp value

To dynamically set the color write masks, call:

```
// Provided by VK_EXT_extended_dynamic_state3, VK_EXT_shader_object
void vkCmdSetColorWriteMaskEXT(
    VkCommandBuffer                                 commandBuffer,
    uint32_t                                        firstAttachment,
    uint32_t                                        attachmentCount,
    const VkColorComponentFlags*                    pColorWriteMasks);
```

- commandBuffer is the command buffer into which the command will be recorded.
- firstAttachment the first color attachment the color write masks apply to.
- attachmentCount the number of VkColorComponentFlags values in the pColorWriteMasks array.
- pColorWriteMasks an array of VkColorComponentFlags values that specify the color write masks of the corresponding attachments.

This command sets the color write masks of the specified attachments for subsequent drawing commands when drawing using shader objects, or when the graphics pipeline is created with VK_DYNAMIC_STATE_COLOR_WRITE_MASK_EXT set in VkPipelineDynamicStateCreateInfo::pDynamicStates. Otherwise, this state is specified by the VkPipelineColorBlendAttachmentState::colorWriteMask values used to create the currently active pipeline.

| | |
|---|---|
| **NOTE** | Formats with bits that are shared between components specified by VkColorComponentFlagBits, such as VK_FORMAT_E5B9G9R9_UFLOAT_PACK32, cannot have their channels individually masked by this functionality; either all components that share bits have to be enabled, or none of them. |

## Valid Usage

- VUID-vkCmdSetColorWriteMaskEXT-None-09423
  At least one of the following **must** be true:
  - The `extendedDynamicState3ColorWriteMask` feature is enabled
  - The `shaderObject` feature is enabled

## Valid Usage (Implicit)

- VUID-vkCmdSetColorWriteMaskEXT-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdSetColorWriteMaskEXT-pColorWriteMasks-parameter
  `pColorWriteMasks` **must** be a valid pointer to an array of `attachmentCount` valid combinations of VkColorComponentFlagBits values
- VUID-vkCmdSetColorWriteMaskEXT-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdSetColorWriteMaskEXT-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- VUID-vkCmdSetColorWriteMaskEXT-videocoding
  This command **must** only be called outside of a video coding scope
- VUID-vkCmdSetColorWriteMaskEXT-attachmentCount-arraylength
  `attachmentCount` **must** be greater than `0`

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics | State |

### 30.1.1. Blend Factors

The source and destination color and alpha blending factors are selected from the enum:

```
// Provided by VK_VERSION_1_0
typedef enum VkBlendFactor {
    VK_BLEND_FACTOR_ZERO = 0,
    VK_BLEND_FACTOR_ONE = 1,
    VK_BLEND_FACTOR_SRC_COLOR = 2,
    VK_BLEND_FACTOR_ONE_MINUS_SRC_COLOR = 3,
    VK_BLEND_FACTOR_DST_COLOR = 4,
    VK_BLEND_FACTOR_ONE_MINUS_DST_COLOR = 5,
    VK_BLEND_FACTOR_SRC_ALPHA = 6,
    VK_BLEND_FACTOR_ONE_MINUS_SRC_ALPHA = 7,
    VK_BLEND_FACTOR_DST_ALPHA = 8,
    VK_BLEND_FACTOR_ONE_MINUS_DST_ALPHA = 9,
    VK_BLEND_FACTOR_CONSTANT_COLOR = 10,
    VK_BLEND_FACTOR_ONE_MINUS_CONSTANT_COLOR = 11,
    VK_BLEND_FACTOR_CONSTANT_ALPHA = 12,
    VK_BLEND_FACTOR_ONE_MINUS_CONSTANT_ALPHA = 13,
    VK_BLEND_FACTOR_SRC_ALPHA_SATURATE = 14,
    VK_BLEND_FACTOR_SRC1_COLOR = 15,
    VK_BLEND_FACTOR_ONE_MINUS_SRC1_COLOR = 16,
    VK_BLEND_FACTOR_SRC1_ALPHA = 17,
    VK_BLEND_FACTOR_ONE_MINUS_SRC1_ALPHA = 18,
} VkBlendFactor;
```

The semantics of the enum values are described in the table below:

*Table 39. Blend Factors*

| VkBlendFactor | RGB Blend Factors ($S_r$,$S_g$,$S_b$) or ($D_r$,$D_g$,$D_b$) | Alpha Blend Factor ($S_a$ or $D_a$) |
|---|---|---|
| VK_BLEND_FACTOR_ZERO | (0,0,0) | 0 |
| VK_BLEND_FACTOR_ONE | (1,1,1) | 1 |
| VK_BLEND_FACTOR_SRC_COLOR | ($R_{s0}$,$G_{s0}$,$B_{s0}$) | $A_{s0}$ |
| VK_BLEND_FACTOR_ONE_MINUS_SRC_COLOR | (1-$R_{s0}$,1-$G_{s0}$,1-$B_{s0}$) | 1-$A_{s0}$ |
| VK_BLEND_FACTOR_DST_COLOR | ($R_d$,$G_d$,$B_d$) | $A_d$ |
| VK_BLEND_FACTOR_ONE_MINUS_DST_COLOR | (1-$R_d$,1-$G_d$,1-$B_d$) | 1-$A_d$ |
| VK_BLEND_FACTOR_SRC_ALPHA | ($A_{s0}$,$A_{s0}$,$A_{s0}$) | $A_{s0}$ |
| VK_BLEND_FACTOR_ONE_MINUS_SRC_ALPHA | (1-$A_{s0}$,1-$A_{s0}$,1-$A_{s0}$) | 1-$A_{s0}$ |
| VK_BLEND_FACTOR_DST_ALPHA | ($A_d$,$A_d$,$A_d$) | $A_d$ |
| VK_BLEND_FACTOR_ONE_MINUS_DST_ALPHA | (1-$A_d$,1-$A_d$,1-$A_d$) | 1-$A_d$ |

| VkBlendFactor | RGB Blend Factors ($S_r$,$S_g$,$S_b$) or ($D_r$,$D_g$,$D_b$) | Alpha Blend Factor ($S_a$ or $D_a$) |
|---|---|---|
| VK_BLEND_FACTOR_CONSTANT_COLOR | ($R_c$,$G_c$,$B_c$) | $A_c$ |
| VK_BLEND_FACTOR_ONE_MINUS_CONSTANT_COLOR | (1-$R_c$,1-$G_c$,1-$B_c$) | 1-$A_c$ |
| VK_BLEND_FACTOR_CONSTANT_ALPHA | ($A_c$,$A_c$,$A_c$) | $A_c$ |
| VK_BLEND_FACTOR_ONE_MINUS_CONSTANT_ALPHA | (1-$A_c$,1-$A_c$,1-$A_c$) | 1-$A_c$ |
| VK_BLEND_FACTOR_SRC_ALPHA_SATURATE | (f,f,f); f = min($A_{s0}$,1-$A_d$) | 1 |
| VK_BLEND_FACTOR_SRC1_COLOR | ($R_{s1}$,$G_{s1}$,$B_{s1}$) | $A_{s1}$ |
| VK_BLEND_FACTOR_ONE_MINUS_SRC1_COLOR | (1-$R_{s1}$,1-$G_{s1}$,1-$B_{s1}$) | 1-$A_{s1}$ |
| VK_BLEND_FACTOR_SRC1_ALPHA | ($A_{s1}$,$A_{s1}$,$A_{s1}$) | $A_{s1}$ |
| VK_BLEND_FACTOR_ONE_MINUS_SRC1_ALPHA | (1-$A_{s1}$,1-$A_{s1}$,1-$A_{s1}$) | 1-$A_{s1}$ |

In this table, the following conventions are used:

- $R_{s0}$,$G_{s0}$,$B_{s0}$ and $A_{s0}$ represent the first source color R, G, B, and A components, respectively, for the fragment output location corresponding to the color attachment being blended.
- $R_{s1}$,$G_{s1}$,$B_{s1}$ and $A_{s1}$ represent the second source color R, G, B, and A components, respectively, used in dual source blending modes, for the fragment output location corresponding to the color attachment being blended.
- $R_d$,$G_d$,$B_d$ and $A_d$ represent the R, G, B, and A components of the destination color. That is, the color currently in the corresponding color attachment for this fragment/sample.
- $R_c$,$G_c$,$B_c$ and $A_c$ represent the blend constant R, G, B, and A components, respectively.

To dynamically set and change the blend constants, call:

```
// Provided by VK_VERSION_1_0
void vkCmdSetBlendConstants(
    VkCommandBuffer                             commandBuffer,
    const float                                 blendConstants[4]);
```

- commandBuffer is the command buffer into which the command will be recorded.
- blendConstants is a pointer to an array of four values specifying the $R_c$, $G_c$, $B_c$, and $A_c$ components of the blend constant color used in blending, depending on the blend factor.

This command sets blend constants for subsequent drawing commands when when drawing using shader objects, or the graphics pipeline is created with VK_DYNAMIC_STATE_BLEND_CONSTANTS set in VkPipelineDynamicStateCreateInfo::pDynamicStates. Otherwise, this state is specified by the VkPipelineColorBlendStateCreateInfo::blendConstants values used to create the currently active pipeline.

### Valid Usage (Implicit)

- VUID-vkCmdSetBlendConstants-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdSetBlendConstants-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdSetBlendConstants-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- VUID-vkCmdSetBlendConstants-videocoding
  This command **must** only be called outside of a video coding scope

### Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

### Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics | State |

## 30.1.2. Dual-Source Blending

Blend factors that use the secondary color input ($R_{s1}$,$G_{s1}$,$B_{s1}$,$A_{s1}$) (`VK_BLEND_FACTOR_SRC1_COLOR`, `VK_BLEND_FACTOR_ONE_MINUS_SRC1_COLOR`, `VK_BLEND_FACTOR_SRC1_ALPHA`, and `VK_BLEND_FACTOR_ONE_MINUS_SRC1_ALPHA`) **may** consume implementation resources that could otherwise be used for rendering to multiple color attachments. Therefore, the number of color attachments that **can** be used in a framebuffer **may** be lower when using dual-source blending.

Dual-source blending is only supported if the `dualSrcBlend` feature is enabled.

The maximum number of color attachments that **can** be used in a subpass when using dual-source blending functions is implementation-dependent and is reported as the `maxFragmentDualSrcAttachments` member of `VkPhysicalDeviceLimits`.

Color outputs **can** be bound to the first and second inputs of the blender using the `Index` decoration, as described in Fragment Output Interface. If the second color input to the blender is not written in the shader, or if no output is bound to the second input of a blender, the value of the second input is undefined.

### 30.1.3. Blend Operations

Once the source and destination blend factors have been selected, they along with the source and destination components are passed to the blending operations. RGB and alpha components **can** use different operations. Possible values of VkBlendOp, specifying the operations, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkBlendOp {
    VK_BLEND_OP_ADD = 0,
    VK_BLEND_OP_SUBTRACT = 1,
    VK_BLEND_OP_REVERSE_SUBTRACT = 2,
    VK_BLEND_OP_MIN = 3,
    VK_BLEND_OP_MAX = 4,
  // Provided by VK_EXT_blend_operation_advanced
    VK_BLEND_OP_ZERO_EXT = 1000148000,
  // Provided by VK_EXT_blend_operation_advanced
    VK_BLEND_OP_SRC_EXT = 1000148001,
  // Provided by VK_EXT_blend_operation_advanced
    VK_BLEND_OP_DST_EXT = 1000148002,
  // Provided by VK_EXT_blend_operation_advanced
    VK_BLEND_OP_SRC_OVER_EXT = 1000148003,
  // Provided by VK_EXT_blend_operation_advanced
    VK_BLEND_OP_DST_OVER_EXT = 1000148004,
  // Provided by VK_EXT_blend_operation_advanced
    VK_BLEND_OP_SRC_IN_EXT = 1000148005,
  // Provided by VK_EXT_blend_operation_advanced
    VK_BLEND_OP_DST_IN_EXT = 1000148006,
  // Provided by VK_EXT_blend_operation_advanced
    VK_BLEND_OP_SRC_OUT_EXT = 1000148007,
  // Provided by VK_EXT_blend_operation_advanced
    VK_BLEND_OP_DST_OUT_EXT = 1000148008,
  // Provided by VK_EXT_blend_operation_advanced
    VK_BLEND_OP_SRC_ATOP_EXT = 1000148009,
  // Provided by VK_EXT_blend_operation_advanced
    VK_BLEND_OP_DST_ATOP_EXT = 1000148010,
  // Provided by VK_EXT_blend_operation_advanced
    VK_BLEND_OP_XOR_EXT = 1000148011,
  // Provided by VK_EXT_blend_operation_advanced
    VK_BLEND_OP_MULTIPLY_EXT = 1000148012,
  // Provided by VK_EXT_blend_operation_advanced
    VK_BLEND_OP_SCREEN_EXT = 1000148013,
  // Provided by VK_EXT_blend_operation_advanced
    VK_BLEND_OP_OVERLAY_EXT = 1000148014,
  // Provided by VK_EXT_blend_operation_advanced
    VK_BLEND_OP_DARKEN_EXT = 1000148015,
  // Provided by VK_EXT_blend_operation_advanced
    VK_BLEND_OP_LIGHTEN_EXT = 1000148016,
  // Provided by VK_EXT_blend_operation_advanced
    VK_BLEND_OP_COLORDODGE_EXT = 1000148017,
  // Provided by VK_EXT_blend_operation_advanced
```

```
    VK_BLEND_OP_COLORBURN_EXT = 1000148018,
  // Provided by VK_EXT_blend_operation_advanced
    VK_BLEND_OP_HARDLIGHT_EXT = 1000148019,
  // Provided by VK_EXT_blend_operation_advanced
    VK_BLEND_OP_SOFTLIGHT_EXT = 1000148020,
  // Provided by VK_EXT_blend_operation_advanced
    VK_BLEND_OP_DIFFERENCE_EXT = 1000148021,
  // Provided by VK_EXT_blend_operation_advanced
    VK_BLEND_OP_EXCLUSION_EXT = 1000148022,
  // Provided by VK_EXT_blend_operation_advanced
    VK_BLEND_OP_INVERT_EXT = 1000148023,
  // Provided by VK_EXT_blend_operation_advanced
    VK_BLEND_OP_INVERT_RGB_EXT = 1000148024,
  // Provided by VK_EXT_blend_operation_advanced
    VK_BLEND_OP_LINEARDODGE_EXT = 1000148025,
  // Provided by VK_EXT_blend_operation_advanced
    VK_BLEND_OP_LINEARBURN_EXT = 1000148026,
  // Provided by VK_EXT_blend_operation_advanced
    VK_BLEND_OP_VIVIDLIGHT_EXT = 1000148027,
  // Provided by VK_EXT_blend_operation_advanced
    VK_BLEND_OP_LINEARLIGHT_EXT = 1000148028,
  // Provided by VK_EXT_blend_operation_advanced
    VK_BLEND_OP_PINLIGHT_EXT = 1000148029,
  // Provided by VK_EXT_blend_operation_advanced
    VK_BLEND_OP_HARDMIX_EXT = 1000148030,
  // Provided by VK_EXT_blend_operation_advanced
    VK_BLEND_OP_HSL_HUE_EXT = 1000148031,
  // Provided by VK_EXT_blend_operation_advanced
    VK_BLEND_OP_HSL_SATURATION_EXT = 1000148032,
  // Provided by VK_EXT_blend_operation_advanced
    VK_BLEND_OP_HSL_COLOR_EXT = 1000148033,
  // Provided by VK_EXT_blend_operation_advanced
    VK_BLEND_OP_HSL_LUMINOSITY_EXT = 1000148034,
  // Provided by VK_EXT_blend_operation_advanced
    VK_BLEND_OP_PLUS_EXT = 1000148035,
  // Provided by VK_EXT_blend_operation_advanced
    VK_BLEND_OP_PLUS_CLAMPED_EXT = 1000148036,
  // Provided by VK_EXT_blend_operation_advanced
    VK_BLEND_OP_PLUS_CLAMPED_ALPHA_EXT = 1000148037,
  // Provided by VK_EXT_blend_operation_advanced
    VK_BLEND_OP_PLUS_DARKER_EXT = 1000148038,
  // Provided by VK_EXT_blend_operation_advanced
    VK_BLEND_OP_MINUS_EXT = 1000148039,
  // Provided by VK_EXT_blend_operation_advanced
    VK_BLEND_OP_MINUS_CLAMPED_EXT = 1000148040,
  // Provided by VK_EXT_blend_operation_advanced
    VK_BLEND_OP_CONTRAST_EXT = 1000148041,
  // Provided by VK_EXT_blend_operation_advanced
    VK_BLEND_OP_INVERT_OVG_EXT = 1000148042,
  // Provided by VK_EXT_blend_operation_advanced
    VK_BLEND_OP_RED_EXT = 1000148043,
```

```
  // Provided by VK_EXT_blend_operation_advanced
    VK_BLEND_OP_GREEN_EXT = 1000148044,
  // Provided by VK_EXT_blend_operation_advanced
    VK_BLEND_OP_BLUE_EXT = 1000148045,
} VkBlendOp;
```

The semantics of the basic blend operations are described in the table below:

*Table 40. Basic Blend Operations*

| VkBlendOp | RGB Components | Alpha Component |
|---|---|---|
| VK_BLEND_OP_ADD | $R = R_{s0} \times S_r + R_d \times D_r$<br>$G = G_{s0} \times S_g + G_d \times D_g$<br>$B = B_{s0} \times S_b + B_d \times D_b$ | $A = A_{s0} \times S_a + A_d \times D_a$ |
| VK_BLEND_OP_SUBTRACT | $R = R_{s0} \times S_r - R_d \times D_r$<br>$G = G_{s0} \times S_g - G_d \times D_g$<br>$B = B_{s0} \times S_b - B_d \times D_b$ | $A = A_{s0} \times S_a - A_d \times D_a$ |
| VK_BLEND_OP_REVERSE_SUBTRACT | $R = R_d \times D_r - R_{s0} \times S_r$<br>$G = G_d \times D_g - G_{s0} \times S_g$<br>$B = B_d \times D_b - B_{s0} \times S_b$ | $A = A_d \times D_a - A_{s0} \times S_a$ |
| VK_BLEND_OP_MIN | $R = \min(R_{s0},R_d)$<br>$G = \min(G_{s0},G_d)$<br>$B = \min(B_{s0},B_d)$ | $A = \min(A_{s0},A_d)$ |
| VK_BLEND_OP_MAX | $R = \max(R_{s0},R_d)$<br>$G = \max(G_{s0},G_d)$<br>$B = \max(B_{s0},B_d)$ | $A = \max(A_{s0},A_d)$ |

In this table, the following conventions are used:

- $R_{s0}$, $G_{s0}$, $B_{s0}$ and $A_{s0}$ represent the first source color R, G, B, and A components, respectively.
- $R_d$, $G_d$, $B_d$ and $A_d$ represent the R, G, B, and A components of the destination color. That is, the color currently in the corresponding color attachment for this fragment/sample.
- $S_r$, $S_g$, $S_b$ and $S_a$ represent the source blend factor R, G, B, and A components, respectively.
- $D_r$, $D_g$, $D_b$ and $D_a$ represent the destination blend factor R, G, B, and A components, respectively.

The blending operation produces a new set of values R, G, B and A, which are written to the framebuffer attachment. If blending is not enabled for this attachment, then R, G, B and A are assigned $R_{s0}$, $G_{s0}$, $B_{s0}$ and $A_{s0}$, respectively.

If the color attachment is fixed-point, the components of the source and destination values and blend factors are each clamped to [0,1] or [-1,1] respectively for an unsigned normalized or signed normalized color attachment prior to evaluating the blend operations. If the color attachment is floating-point, no clamping occurs.

If the numeric format of a framebuffer attachment uses sRGB encoding, the R, G, and B destination color values (after conversion from fixed-point to floating-point) are considered to be encoded for the sRGB color space and hence are linearized prior to their use in blending. Each R, G, and B component is converted from nonlinear to linear as described in the “sRGB EOTF” section of the Khronos Data Format Specification. If the format is not sRGB, no linearization is performed.

If the numeric format of a framebuffer attachment uses sRGB encoding, then the final R, G and B values are converted into the nonlinear sRGB representation before being written to the framebuffer attachment as described in the “sRGB $EOTF^{-1}$” section of the Khronos Data Format

Specification.

If the numeric format of a framebuffer color attachment is not sRGB encoded then the resulting $c_s$ values for R, G and B are unmodified. The value of A is never sRGB encoded. That is, the alpha component is always stored in memory as linear.

If the framebuffer color attachment is VK_ATTACHMENT_UNUSED, no writes are performed through that attachment. Writes are not performed to framebuffer color attachments greater than or equal to the VkSubpassDescription::colorAttachmentCount or VkSubpassDescription2::colorAttachmentCount value.

### 30.1.4. Advanced Blend Operations

The *advanced blend operations* are those listed in tables f/X/Y/Z Advanced Blend Operations, Hue-Saturation-Luminosity Advanced Blend Operations, and Additional RGB Blend Operations.

If the pNext chain of VkPipelineColorBlendStateCreateInfo includes a VkPipelineColorBlendAdvancedStateCreateInfoEXT structure, then that structure includes parameters that affect advanced blend operations.

The VkPipelineColorBlendAdvancedStateCreateInfoEXT structure is defined as:

```
// Provided by VK_EXT_blend_operation_advanced
typedef struct VkPipelineColorBlendAdvancedStateCreateInfoEXT {
    VkStructureType      sType;
    const void*          pNext;
    VkBool32             srcPremultiplied;
    VkBool32             dstPremultiplied;
    VkBlendOverlapEXT    blendOverlap;
} VkPipelineColorBlendAdvancedStateCreateInfoEXT;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- srcPremultiplied specifies whether the source color of the blend operation is treated as premultiplied.
- dstPremultiplied specifies whether the destination color of the blend operation is treated as premultiplied.
- blendOverlap is a VkBlendOverlapEXT value specifying how the source and destination sample's coverage is correlated.

If this structure is not present, srcPremultiplied and dstPremultiplied are both considered to be VK_TRUE, and blendOverlap is considered to be VK_BLEND_OVERLAP_UNCORRELATED_EXT.

**Valid Usage**

- VUID-VkPipelineColorBlendAdvancedStateCreateInfoEXT-srcPremultiplied-01424
  If the non-premultiplied source color property is not supported, srcPremultiplied **must** be

VK_TRUE

- VUID-VkPipelineColorBlendAdvancedStateCreateInfoEXT-dstPremultiplied-01425
  If the non-premultiplied destination color property is not supported, dstPremultiplied **must** be VK_TRUE
- VUID-VkPipelineColorBlendAdvancedStateCreateInfoEXT-blendOverlap-01426
  If the correlated overlap property is not supported, blendOverlap **must** be VK_BLEND_OVERLAP_UNCORRELATED_EXT

### Valid Usage (Implicit)

- VUID-VkPipelineColorBlendAdvancedStateCreateInfoEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PIPELINE_COLOR_BLEND_ADVANCED_STATE_CREATE_INFO_EXT
- VUID-VkPipelineColorBlendAdvancedStateCreateInfoEXT-blendOverlap-parameter
  blendOverlap **must** be a valid VkBlendOverlapEXT value

To dynamically set the advanced blend state, call:

```
// Provided by VK_EXT_blend_operation_advanced with VK_EXT_extended_dynamic_state3,
VK_EXT_blend_operation_advanced with VK_EXT_shader_object
void vkCmdSetColorBlendAdvancedEXT(
    VkCommandBuffer                             commandBuffer,
    uint32_t                                    firstAttachment,
    uint32_t                                    attachmentCount,
    const VkColorBlendAdvancedEXT*              pColorBlendAdvanced);
```

- commandBuffer is the command buffer into which the command will be recorded.
- firstAttachment the first color attachment the advanced blend parameters apply to.
- attachmentCount the number of VkColorBlendAdvancedEXT elements in the pColorBlendAdvanced array.
- pColorBlendAdvanced an array of VkColorBlendAdvancedEXT structs that specify the advanced color blend parameters for the corresponding attachments.

This command sets the advanced blend operation parameters of the specified attachments for subsequent drawing commands when drawing using shader objects, or when the graphics pipeline is created with VK_DYNAMIC_STATE_COLOR_BLEND_ADVANCED_EXT set in VkPipelineDynamicStateCreateInfo::pDynamicStates. Otherwise, this state is specified by the VkPipelineColorBlendAdvancedStateCreateInfoEXT::srcPremultiplied, VkPipelineColorBlendAdvancedStateCreateInfoEXT::dstPremultiplied, and VkPipelineColorBlendAdvancedStateCreateInfoEXT::blendOverlap values used to create the currently active pipeline.

## Valid Usage

- VUID-vkCmdSetColorBlendAdvancedEXT-None-09423
  At least one of the following **must** be true:
  - The `extendedDynamicState3ColorBlendAdvanced` feature is enabled
  - The `shaderObject` feature is enabled

## Valid Usage (Implicit)

- VUID-vkCmdSetColorBlendAdvancedEXT-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdSetColorBlendAdvancedEXT-pColorBlendAdvanced-parameter
  `pColorBlendAdvanced` **must** be a valid pointer to an array of `attachmentCount` valid VkColorBlendAdvancedEXT structures
- VUID-vkCmdSetColorBlendAdvancedEXT-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdSetColorBlendAdvancedEXT-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- VUID-vkCmdSetColorBlendAdvancedEXT-videocoding
  This command **must** only be called outside of a video coding scope
- VUID-vkCmdSetColorBlendAdvancedEXT-attachmentCount-arraylength
  `attachmentCount` **must** be greater than `0`

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics | State |

The `VkColorBlendAdvancedEXT` structure is defined as:

```
// Provided by VK_EXT_extended_dynamic_state3, VK_EXT_shader_object
typedef struct VkColorBlendAdvancedEXT {
    VkBlendOp            advancedBlendOp;
    VkBool32             srcPremultiplied;
    VkBool32             dstPremultiplied;
    VkBlendOverlapEXT    blendOverlap;
    VkBool32             clampResults;
} VkColorBlendAdvancedEXT;
```

- `advancedBlendOp` selects which blend operation is used to calculate the RGB values to write to the color attachment.
- `srcPremultiplied` specifies whether the source color of the blend operation is treated as premultiplied.
- `dstPremultiplied` specifies whether the destination color of the blend operation is treated as premultiplied.
- `blendOverlap` is a VkBlendOverlapEXT value specifying how the source and destination sample's coverage is correlated.
- `clampResults` specifies that results **must** be clamped to the [0,1] range before writing to the attachment, which is useful when the attachment format is not normalized fixed-point.

### Valid Usage

- VUID-VkColorBlendAdvancedEXT-srcPremultiplied-07505
  If the non-premultiplied source color property is not supported, `srcPremultiplied` **must** be `VK_TRUE`
- VUID-VkColorBlendAdvancedEXT-dstPremultiplied-07506
  If the non-premultiplied destination color property is not supported, `dstPremultiplied` **must** be `VK_TRUE`
- VUID-VkColorBlendAdvancedEXT-blendOverlap-07507
  If the correlated overlap property is not supported, `blendOverlap` **must** be `VK_BLEND_OVERLAP_UNCORRELATED_EXT`

### Valid Usage (Implicit)

- VUID-VkColorBlendAdvancedEXT-advancedBlendOp-parameter
  `advancedBlendOp` **must** be a valid VkBlendOp value
- VUID-VkColorBlendAdvancedEXT-blendOverlap-parameter
  `blendOverlap` **must** be a valid VkBlendOverlapEXT value

When using one of the operations in table f/X/Y/Z Advanced Blend Operations or Hue-Saturation-Luminosity Advanced Blend Operations, blending is performed according to the following equations:

$$
\begin{aligned}
R &= f(R_s{}', R_d{}') * p_0(A_s, A_d) + Y * R_s{}' * p_1(A_s, A_d) + Z * R_d{}' * p_2(A_s, A_d) \\
G &= f(G_s{}', G_d{}') * p_0(A_s, A_d) + Y * G_s{}' * p_1(A_s, A_d) + Z * G_d{}' * p_2(A_s, A_d) \\
B &= f(B_s{}', B_d{}') * p_0(A_s, A_d) + Y * B_s{}' * p_1(A_s, A_d) + Z * B_d{}' * p_2(A_s, A_d) \\
A &= X * p_0(A_s, A_d) + Y * p_1(A_s, A_d) + Z * p_2(A_s, A_d)
\end{aligned}
$$

where the function f and terms X, Y, and Z are specified in the table. The R, G, and B components of the source color used for blending are derived according to `srcPremultiplied`. If `srcPremultiplied` is set to `VK_TRUE`, the fragment color components are considered to have been premultiplied by the A component prior to blending. The base source color ($R_s'$,$G_s'$,$B_s'$) is obtained by dividing through by the A component:

$$
(R_s{}', G_s{}', B_s{}') = \begin{cases} (0, 0, 0) & A_s = 0 \\ (\frac{R_s}{A_s}, \frac{G_s}{A_s}, \frac{B_s}{A_s}) & \text{otherwise} \end{cases}
$$

If `srcPremultiplied` is `VK_FALSE`, the fragment color components are used as the base color:

$$
(R_s{}', G_s{}', B_s{}') = (R_s, G_s, B_s)
$$

The R, G, and B components of the destination color used for blending are derived according to `dstPremultiplied`. If `dstPremultiplied` is set to `VK_TRUE`, the destination components are considered to have been premultiplied by the A component prior to blending. The base destination color ($R_d'$,$G_d'$,$B_d'$) is obtained by dividing through by the A component:

$$
(R_d{}', G_d{}', B_d{}') = \begin{cases} (0, 0, 0) & A_d = 0 \\ (\frac{R_d}{A_d}, \frac{G_d}{A_d}, \frac{B_d}{A_d}) & \text{otherwise} \end{cases}
$$

If `dstPremultiplied` is `VK_FALSE`, the destination color components are used as the base color:

$$
(R_d{}', G_d{}', B_d{}') = (R_d, G_d, B_d)
$$

When blending using advanced blend operations, we expect that the R, G, and B components of premultiplied source and destination color inputs be stored as the product of non-premultiplied R, G, and B component values and the A component of the color. If any R, G, or B component of a premultiplied input color is non-zero and the A component is zero, the color is considered ill-formed, and the corresponding component of the blend result is undefined.

All of the advanced blend operation formulas in this chapter compute the result as a premultiplied color. If `dstPremultiplied` is `VK_FALSE`, that result color's R, G, and B components are divided by the A component before being written to the framebuffer. If any R, G, or B component of the color is non-zero and the A component is zero, the result is considered ill-formed, and the corresponding component of the blend result is undefined. If all components are zero, that value is unchanged.

If the A component of any input or result color is less than zero, the color is considered ill-formed, and all components of the blend result are undefined.

The weighting functions $p_0$, $p_1$, and $p_2$ are defined in table Advanced Blend Overlap Modes. In these

functions, the A components of the source and destination colors are taken to indicate the portion of the pixel covered by the fragment (source) and the fragments previously accumulated in the pixel (destination). The functions $p_0$, $p_1$, and $p_2$ approximate the relative portion of the pixel covered by the intersection of the source and destination, covered only by the source, and covered only by the destination, respectively.

Possible values of VkPipelineColorBlendAdvancedStateCreateInfoEXT::`blendOverlap`, specifying the blend overlap functions, are:

```
// Provided by VK_EXT_blend_operation_advanced
typedef enum VkBlendOverlapEXT {
    VK_BLEND_OVERLAP_UNCORRELATED_EXT = 0,
    VK_BLEND_OVERLAP_DISJOINT_EXT = 1,
    VK_BLEND_OVERLAP_CONJOINT_EXT = 2,
} VkBlendOverlapEXT;
```

- `VK_BLEND_OVERLAP_UNCORRELATED_EXT` specifies that there is no correlation between the source and destination coverage.
- `VK_BLEND_OVERLAP_CONJOINT_EXT` specifies that the source and destination coverage are considered to have maximal overlap.
- `VK_BLEND_OVERLAP_DISJOINT_EXT` specifies that the source and destination coverage are considered to have minimal overlap.

*Table 41. Advanced Blend Overlap Modes*

| Overlap Mode | Weighting Equations |
|---|---|
| `VK_BLEND_OVERLAP_UNCORRELATED_EXT` | $p_0(A_s, A_d) = A_s A_d$<br>$p_1(A_s, A_d) = A_s(1 - A_d)$<br>$p_2(A_s, A_d) = A_d(1 - A_s)$ |
| `VK_BLEND_OVERLAP_CONJOINT_EXT` | $p_0(A_s, A_d) = min(A_s, A_d)$<br>$p_1(A_s, A_d) = max(A_s - A_d, 0)$<br>$p_2(A_s, A_d) = max(A_d - A_s, 0)$ |
| `VK_BLEND_OVERLAP_DISJOINT_EXT` | $p_0(A_s, A_d) = max(A_s + A_d - 1, 0)$<br>$p_1(A_s, A_d) = min(A_s, 1 - A_d)$<br>$p_2(A_s, A_d) = min(A_d, 1 - A_s)$ |

*Table 42. f/X/Y/Z Advanced Blend Operations*

| Mode | Blend Coefficients |
|---|---|
| `VK_BLEND_OP_ZERO_EXT` | $(X, Y, Z) = (0, 0, 0)$<br>$f(C_s, C_d) = 0$ |
| `VK_BLEND_OP_SRC_EXT` | $(X, Y, Z) = (1, 1, 0)$<br>$f(C_s, C_d) = C_s$ |

| Mode | Blend Coefficients |
|---|---|
| VK_BLEND_OP_DST_EXT | $(X, Y, Z) = (1, 0, 1)$<br>$f(C_s, C_d) = C_d$ |
| VK_BLEND_OP_SRC_OVER_EXT | $(X, Y, Z) = (1, 1, 1)$<br>$f(C_s, C_d) = C_s$ |
| VK_BLEND_OP_DST_OVER_EXT | $(X, Y, Z) = (1, 1, 1)$<br>$f(C_s, C_d) = C_d$ |
| VK_BLEND_OP_SRC_IN_EXT | $(X, Y, Z) = (1, 0, 0)$<br>$f(C_s, C_d) = C_s$ |
| VK_BLEND_OP_DST_IN_EXT | $(X, Y, Z) = (1, 0, 0)$<br>$f(C_s, C_d) = C_d$ |
| VK_BLEND_OP_SRC_OUT_EXT | $(X, Y, Z) = (0, 1, 0)$<br>$f(C_s, C_d) = 0$ |
| VK_BLEND_OP_DST_OUT_EXT | $(X, Y, Z) = (0, 0, 1)$<br>$f(C_s, C_d) = 0$ |
| VK_BLEND_OP_SRC_ATOP_EXT | $(X, Y, Z) = (1, 0, 1)$<br>$f(C_s, C_d) = C_s$ |
| VK_BLEND_OP_DST_ATOP_EXT | $(X, Y, Z) = (1, 1, 0)$<br>$f(C_s, C_d) = C_d$ |
| VK_BLEND_OP_XOR_EXT | $(X, Y, Z) = (0, 1, 1)$<br>$f(C_s, C_d) = 0$ |
| VK_BLEND_OP_MULTIPLY_EXT | $(X, Y, Z) = (1, 1, 1)$<br>$f(C_s, C_d) = C_s C_d$ |
| VK_BLEND_OP_SCREEN_EXT | $(X, Y, Z) = (1, 1, 1)$<br>$f(C_s, C_d) = C_s + C_d - C_s C_d$ |
| VK_BLEND_OP_OVERLAY_EXT | $(X, Y, Z) = (1, 1, 1)$<br>$f(C_s, C_d) = \begin{cases} 2C_s C_d & C_d \leq 0.5 \\ 1 - 2(1 - C_s)(1 - C_d) & \text{otherwise} \end{cases}$ |
| VK_BLEND_OP_DARKEN_EXT | $(X, Y, Z) = (1, 1, 1)$<br>$f(C_s, C_d) = min(C_s, C_d)$ |
| VK_BLEND_OP_LIGHTEN_EXT | $(X, Y, Z) = (1, 1, 1)$<br>$f(C_s, C_d) = max(C_s, C_d)$ |
| VK_BLEND_OP_COLORDODGE_EXT | $(X, Y, Z) = (1, 1, 1)$<br>$f(C_s, C_d) = \begin{cases} 0 & C_d \leq 0 \\ min(1, \frac{C_d}{1 - C_s}) & C_d > 0 \text{and} C_s < 1 \\ 1 & C_d > 0 \text{and} C_s \geq 1 \end{cases}$ |

| Mode | Blend Coefficients |
|---|---|
| VK_BLEND_OP_COLORBURN_EXT | $(X, Y, Z) = (1, 1, 1)$ <br> $f(C_s, C_d) = \begin{cases} 1 & C_d \geq 1 \\ 1 - min(1, \frac{1 - C_d}{C_s}) & C_d < 1 \text{and} C_s > 0 \\ 0 & C_d < 1 \text{and} C_s \leq 0 \end{cases}$ |
| VK_BLEND_OP_HARDLIGHT_EXT | $(X, Y, Z) = (1, 1, 1)$ <br> $f(C_s, C_d) = \begin{cases} 2C_sC_d & C_s \leq 0.5 \\ 1 - 2(1 - C_s)(1 - C_d) & \text{otherwise} \end{cases}$ |
| VK_BLEND_OP_SOFTLIGHT_EXT | $(X, Y, Z) = (1, 1, 1)$ <br> $f(C_s, C_d) = \begin{cases} C_d - (1 - 2C_s)C_d(1 - C_d) & C_s \leq 0.5 \\ C_d + (2C_s - 1)C_d((16C_d - 12)C_d + 3) & C_s > 0.5 \text{and} C_d \leq 0.25 \\ C_d + (2C_s - 1)(\sqrt{C_d} - C_d) & C_s > 0.5 \text{and} C_d > 0.25 \end{cases}$ |
| VK_BLEND_OP_DIFFERENCE_EXT | $(X, Y, Z) = (1, 1, 1)$ <br> $f(C_s, C_d) = \lvert C_d - C_s \rvert$ |
| VK_BLEND_OP_EXCLUSION_EXT | $(X, Y, Z) = (1, 1, 1)$ <br> $f(C_s, C_d) = C_s + C_d - 2C_sC_d$ |
| VK_BLEND_OP_INVERT_EXT | $(X, Y, Z) = (1, 0, 1)$ <br> $f(C_s, C_d) = 1 - C_d$ |
| VK_BLEND_OP_INVERT_RGB_EXT | $(X, Y, Z) = (1, 0, 1)$ <br> $f(C_s, C_d) = C_s(1 - C_d)$ |
| VK_BLEND_OP_LINEARDODGE_EXT | $(X, Y, Z) = (1, 1, 1)$ <br> $f(C_s, C_d) = \begin{cases} C_s + C_d & C_s + C_d \leq 1 \\ 1 & \text{otherwise} \end{cases}$ |
| VK_BLEND_OP_LINEARBURN_EXT | $(X, Y, Z) = (1, 1, 1)$ <br> $f(C_s, C_d) = \begin{cases} C_s + C_d - 1 & C_s + C_d > 1 \\ 0 & \text{otherwise} \end{cases}$ |
| VK_BLEND_OP_VIVIDLIGHT_EXT | $(X, Y, Z) = (1, 1, 1)$ <br> $f(C_s, C_d) = \begin{cases} 1 - min(1, \frac{1 - C_d}{2C_s}) & 0 < C_s < 0.5 \\ 0 & C_s \leq 0 \\ min(1, \frac{C_d}{2(1 - C_s)}) & 0.5 \leq C_s < 1 \\ 1 & C_s \geq 1 \end{cases}$ |
| VK_BLEND_OP_LINEARLIGHT_EXT | $(X, Y, Z) = (1, 1, 1)$ <br> $f(C_s, C_d) = \begin{cases} 1 & 2C_s + C_d > 2 \\ 2C_s + C_d - 1 & 1 < 2C_s + C_d \leq 2 \\ 0 & 2C_s + C_d \leq 1 \end{cases}$ |
| VK_BLEND_OP_PINLIGHT_EXT | $(X, Y, Z) = (1, 1, 1)$ <br> $f(C_s, C_d) = \begin{cases} 0 & 2C_s - 1 > C_d \text{and} C_s < 0.5 \\ 2C_s - 1 & 2C_s - 1 > C_d \text{and} C_s \geq 0.5 \\ 2C_s & 2C_s - 1 \leq C_d \text{and} C_s < 0.5C_d \\ C_d & 2C_s - 1 \leq C_d \text{and} C_s \geq 0.5C_d \end{cases}$ |
| VK_BLEND_OP_HARDMIX_EXT | $(X, Y, Z) = (1, 1, 1)$ <br> $f(C_s, C_d) = \begin{cases} 0 & C_s + C_d < 1 \\ 1 & \text{otherwise} \end{cases}$ |

When using one of the HSL blend operations in table Hue-Saturation-Luminosity Advanced Blend

Operations as the blend operation, the RGB color components produced by the function f are effectively obtained by converting both the non-premultiplied source and destination colors to the HSL (hue, saturation, luminosity) color space, generating a new HSL color by selecting H, S, and L components from the source or destination according to the blend operation, and then converting the result back to RGB. In the equations below, a blended RGB color is produced according to the following pseudocode:

```
float minv3(vec3 c) {
  return min(min(c.r, c.g), c.b);
}
float maxv3(vec3 c) {
  return max(max(c.r, c.g), c.b);
}
float lumv3(vec3 c) {
  return dot(c, vec3(0.30, 0.59, 0.11));
}
float satv3(vec3 c) {
  return maxv3(c) - minv3(c);
}

// If any color components are outside [0,1], adjust the color to
// get the components in range.
vec3 ClipColor(vec3 color) {
  float lum = lumv3(color);
  float mincol = minv3(color);
  float maxcol = maxv3(color);
  if (mincol < 0.0) {
    color = lum + ((color-lum)*lum) / (lum-mincol);
  }
  if (maxcol > 1.0) {
    color = lum + ((color-lum)*(1-lum)) / (maxcol-lum);
  }
  return color;
}

// Take the base RGB color <cbase> and override its luminosity
// with that of the RGB color <clum>.
vec3 SetLum(vec3 cbase, vec3 clum) {
  float lbase = lumv3(cbase);
  float llum = lumv3(clum);
  float ldiff = llum - lbase;
  vec3 color = cbase + vec3(ldiff);
  return ClipColor(color);
}

// Take the base RGB color <cbase> and override its saturation with
// that of the RGB color <csat>.  The override the luminosity of the
// result with that of the RGB color <clum>.
vec3 SetLumSat(vec3 cbase, vec3 csat, vec3 clum)
{
```

```
    float minbase = minv3(cbase);
    float sbase = satv3(cbase);
    float ssat = satv3(csat);
    vec3 color;
    if (sbase > 0) {
      // Equivalent (modulo rounding errors) to setting the
      // smallest (R,G,B) component to 0, the largest to <ssat>,
      // and interpolating the "middle" component based on its
      // original value relative to the smallest/largest.
      color = (cbase - minbase) * ssat / sbase;
    } else {
      color = vec3(0.0);
    }
    return SetLum(color, clum);
  }
```

*Table 43. Hue-Saturation-Luminosity Advanced Blend Operations*

| Mode | Result |
|---|---|
| VK_BLEND_OP_HSL_HUE_EXT | $(X, Y, Z) = (1, 1, 1)$<br>$f(C_s, C_d) = SetLumSat(C_s, C_d, C_d)$ |
| VK_BLEND_OP_HSL_SATURATION_EXT | $(X, Y, Z) = (1, 1, 1)$<br>$f(C_s, C_d) = SetLumSat(C_d, C_s, C_d)$ |
| VK_BLEND_OP_HSL_COLOR_EXT | $(X, Y, Z) = (1, 1, 1)$<br>$f(C_s, C_d) = SetLum(C_s, C_d)$ |
| VK_BLEND_OP_HSL_LUMINOSITY_EXT | $(X, Y, Z) = (1, 1, 1)$<br>$f(C_s, C_d) = SetLum(C_d, C_s)$ |

When using one of the operations in table Additional RGB Blend Operations as the blend operation, the source and destination colors used by these blending operations are interpreted according to srcPremultiplied and dstPremultiplied. The blending operations below are evaluated where the RGB source and destination color components are both considered to have been premultiplied by the corresponding A component.

$$(R_s{}', G_s{}', B_s{}') = \begin{cases} (R_s, G_s, B_s) & \text{if srcPremultiplied is VK_TRUE} \\ (R_sA_s, G_sA_s, B_sA_s) & \text{if srcPremultiplied is VK_FALSE} \end{cases}$$

$$(R_d{}', G_d{}', B_d{}') = \begin{cases} (R_d, G_d, B_d) & \text{if dstPremultiplied is VK_TRUE} \\ (R_dA_d, G_dA_d, B_dA_d) & \text{if dstPremultiplied is VK_FALSE} \end{cases}$$

*Table 44. Additional RGB Blend Operations*

| Mode | Result |
|---|---|
| VK_BLEND_OP_PLUS_EXT | $(R, G, B, A) = (R_s{}' + R_d{}', G_s{}' + G_d{}', B_s{}' + B_d{}', A_s + A_d)$ |
| VK_BLEND_OP_PLUS_CLAMPED_EXT | $(R, G, B, A) = (min(1, R_s{}' + R_d{}'), min(1, G_s{}' + G_d{}'), min(1, B_s{}' + B_d{}'), min(1, A_s + A_d))$ |
| VK_BLEND_OP_PLUS_CLAMPED_ALPHA_EXT | $(R, G, B, A) = (min(min(1, A_s + A_d), R_s{}' + R_d{}'), min(min(1, A_s + A_d), G_s{}' + G_d{}'), min(min(1, A_s + A_d), B_s{}' + B_d{}'), min(1, A_s + A_d))$ |
| VK_BLEND_OP_PLUS_DARKER_EXT | $(R, G, B, A) = (max(0, min(1, A_s + A_d) - ((A_s - R_s{}') + (A_d - R_d{}'))), max(0, min(1, A_s + A_d) - ((A_s - G_s{}') + (A_d - G_d{}'))), max(0, min(1, A_s + A_d) - ((A_s - B_s{}') + (A_d - B_d{}'))), min(1, A_s + A_d))$ |
| VK_BLEND_OP_MINUS_EXT | $(R, G, B, A) = (R_d{}' - R_s{}', G_d{}' - G_s{}', B_d{}' - B_s{}', A_d - A_s)$ |
| VK_BLEND_OP_MINUS_CLAMPED_EXT | $(R, G, B, A) = (max(0, R_d{}' - R_s{}'), max(0, G_d{}' - G_s{}'), max(0, B_d{}' - B_s{}'), max(0, A_d - A_s))$ |
| VK_BLEND_OP_CONTRAST_EXT | $(R, G, B, A) = (\frac{A_d}{2} + 2(R_d{}' - \frac{A_d}{2})(R_s{}' - \frac{A_s}{2}), \frac{A_d}{2} + 2(G_d{}' - \frac{A_d}{2})(G_s{}' - \frac{A_s}{2}), \frac{A_d}{2} + 2(B_d{}' - \frac{A_d}{2})(B_s{}' - \frac{A_s}{2}), A_d)$ |
| VK_BLEND_OP_INVERT_OVG_EXT | $(R, G, B, A) = (A_s(1 - R_d{}') + (1 - A_s)R_d{}', A_s(1 - G_d{}') + (1 - A_s)G_d{}', A_s(1 - B_d{}') + (1 - A_s)B_d{}', A_s + A_d - A_sA_d)$ |
| VK_BLEND_OP_RED_EXT | $(R, G, B, A) = (R_s{}', G_d{}', B_d{}', A_d)$ |
| VK_BLEND_OP_GREEN_EXT | $(R, G, B, A) = (R_d{}', G_s{}', B_d{}', A_d)$ |
| VK_BLEND_OP_BLUE_EXT | $(R, G, B, A) = (R_d{}', G_d{}', B_s{}', A_d)$ |

## 30.2. Logical Operations

The application **can** enable a *logical operation* between the fragment's color values and the existing value in the framebuffer attachment. This logical operation is applied prior to updating the framebuffer attachment. Logical operations are applied only for signed and unsigned integer and normalized integer framebuffers. Logical operations are not applied to floating-point or sRGB format color attachments.

Logical operations are controlled by the `logicOpEnable` and `logicOp` members of VkPipelineColorBlendStateCreateInfo. The `logicOpEnable` state can also be controlled by vkCmdSetLogicOpEnableEXT if graphics pipeline is created with `VK_DYNAMIC_STATE_LOGIC_OP_ENABLE_EXT` set in VkPipelineDynamicStateCreateInfo::`pDynamicStates`. The `logicOp` state can also be controlled by vkCmdSetLogicOpEXT if graphics pipeline is created with `VK_DYNAMIC_STATE_LOGIC_OP_EXT` set in VkPipelineDynamicStateCreateInfo::`pDynamicStates`. If `logicOpEnable` is `VK_TRUE`, then a logical operation selected by `logicOp` is applied between each color attachment and the fragment's corresponding output value, and blending of all attachments is treated as if it were disabled. Any attachments using color formats for which logical operations are not supported simply pass through the color values unmodified. The logical operation is applied independently for each of the red, green, blue, and alpha components. The `logicOp` is selected from the following operations:

```
// Provided by VK_VERSION_1_0
typedef enum VkLogicOp {
    VK_LOGIC_OP_CLEAR = 0,
    VK_LOGIC_OP_AND = 1,
    VK_LOGIC_OP_AND_REVERSE = 2,
    VK_LOGIC_OP_COPY = 3,
    VK_LOGIC_OP_AND_INVERTED = 4,
    VK_LOGIC_OP_NO_OP = 5,
    VK_LOGIC_OP_XOR = 6,
    VK_LOGIC_OP_OR = 7,
    VK_LOGIC_OP_NOR = 8,
    VK_LOGIC_OP_EQUIVALENT = 9,
    VK_LOGIC_OP_INVERT = 10,
    VK_LOGIC_OP_OR_REVERSE = 11,
    VK_LOGIC_OP_COPY_INVERTED = 12,
    VK_LOGIC_OP_OR_INVERTED = 13,
    VK_LOGIC_OP_NAND = 14,
    VK_LOGIC_OP_SET = 15,
} VkLogicOp;
```

The logical operations supported by Vulkan are summarized in the following table in which

- ¬ is bitwise invert,
- □ is bitwise and,
- □ is bitwise or,
- □ is bitwise exclusive or,
- s is the fragment's $R_{s0}$, $G_{s0}$, $B_{s0}$ or $A_{s0}$ component value for the fragment output corresponding to the color attachment being updated, and
- d is the color attachment's R, G, B or A component value:

*Table 45. Logical Operations*

| Mode | Operation |
|---|---|
| VK_LOGIC_OP_CLEAR | 0 |
| VK_LOGIC_OP_AND | s □ d |
| VK_LOGIC_OP_AND_REVERSE | s □ ¬ d |
| VK_LOGIC_OP_COPY | s |
| VK_LOGIC_OP_AND_INVERTED | ¬ s □ d |
| VK_LOGIC_OP_NO_OP | d |
| VK_LOGIC_OP_XOR | s □ d |
| VK_LOGIC_OP_OR | s □ d |
| VK_LOGIC_OP_NOR | ¬ (s □ d) |
| VK_LOGIC_OP_EQUIVALENT | ¬ (s □ d) |
| VK_LOGIC_OP_INVERT | ¬ d |
| VK_LOGIC_OP_OR_REVERSE | s □ ¬ d |
| VK_LOGIC_OP_COPY_INVERTED | ¬ s |
| VK_LOGIC_OP_OR_INVERTED | ¬ s □ d |
| VK_LOGIC_OP_NAND | ¬ (s □ d) |
| VK_LOGIC_OP_SET | all 1s |

The result of the logical operation is then written to the color attachment as controlled by the component write mask, described in Blend Operations.

To dynamically set whether logical operations are enabled, call:

```
// Provided by VK_EXT_extended_dynamic_state3, VK_EXT_shader_object
void vkCmdSetLogicOpEnableEXT(
    VkCommandBuffer                                 commandBuffer,
    VkBool32                                        logicOpEnable);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `logicOpEnable` specifies whether logical operations are enabled.

This command sets whether logical operations are enabled for subsequent drawing commands when drawing using shader objects, or when the graphics pipeline is created with `VK_DYNAMIC_STATE_LOGIC_OP_ENABLE_EXT` set in VkPipelineDynamicStateCreateInfo::`pDynamicStates`. Otherwise, this state is specified by the VkPipelineColorBlendStateCreateInfo::`logicOpEnable` value used to create the currently active pipeline.

## Valid Usage

- VUID-vkCmdSetLogicOpEnableEXT-None-09423
  At least one of the following **must** be true:
  - The `extendedDynamicState3LogicOpEnable` feature is enabled
  - The `shaderObject` feature is enabled
- VUID-vkCmdSetLogicOpEnableEXT-logicOp-07366
  If the `logicOp` feature is not enabled, `logicOpEnable` **must** be `VK_FALSE`

## Valid Usage (Implicit)

- VUID-vkCmdSetLogicOpEnableEXT-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdSetLogicOpEnableEXT-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdSetLogicOpEnableEXT-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- VUID-vkCmdSetLogicOpEnableEXT-videocoding
  This command **must** only be called outside of a video coding scope

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics | State |

To dynamically set the logical operation to apply for blend state, call:

```
// Provided by VK_EXT_extended_dynamic_state2, VK_EXT_shader_object
void vkCmdSetLogicOpEXT(
    VkCommandBuffer                                 commandBuffer,
    VkLogicOp                                       logicOp);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `logicOp` specifies the logical operation to apply for blend state.

This command sets the logical operation for blend state for subsequent drawing commands when drawing using shader objects, or when the graphics pipeline is created with `VK_DYNAMIC_STATE_LOGIC_OP_EXT` set in VkPipelineDynamicStateCreateInfo::`pDynamicStates`. Otherwise, this state is specified by the VkPipelineColorBlendStateCreateInfo::`logicOp` value used to create the currently active pipeline.

## Valid Usage

- VUID-vkCmdSetLogicOpEXT-None-09422
  At least one of the following **must** be true:
  - The `extendedDynamicState2LogicOp` feature is enabled
  - The `shaderObject` feature is enabled

## Valid Usage (Implicit)

- VUID-vkCmdSetLogicOpEXT-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdSetLogicOpEXT-logicOp-parameter
  `logicOp` **must** be a valid VkLogicOp value
- VUID-vkCmdSetLogicOpEXT-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdSetLogicOpEXT-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations

- VUID-vkCmdSetLogicOpEXT-videocoding
  This command **must** only be called outside of a video coding scope

### Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

### Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics | State |

## 30.3. Color Write Mask

Bits which **can** be set in VkPipelineColorBlendAttachmentState::`colorWriteMask`, determining whether the final color values R, G, B and A are written to the framebuffer attachment, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkColorComponentFlagBits {
    VK_COLOR_COMPONENT_R_BIT = 0x00000001,
    VK_COLOR_COMPONENT_G_BIT = 0x00000002,
    VK_COLOR_COMPONENT_B_BIT = 0x00000004,
    VK_COLOR_COMPONENT_A_BIT = 0x00000008,
} VkColorComponentFlagBits;
```

- `VK_COLOR_COMPONENT_R_BIT` specifies that the R value is written to the color attachment for the appropriate sample. Otherwise, the value in memory is unmodified.
- `VK_COLOR_COMPONENT_G_BIT` specifies that the G value is written to the color attachment for the appropriate sample. Otherwise, the value in memory is unmodified.
- `VK_COLOR_COMPONENT_B_BIT` specifies that the B value is written to the color attachment for the appropriate sample. Otherwise, the value in memory is unmodified.
- `VK_COLOR_COMPONENT_A_BIT` specifies that the A value is written to the color attachment for the appropriate sample. Otherwise, the value in memory is unmodified.

The color write mask operation is applied regardless of whether blending is enabled.

The color write mask operation is applied only if Color Write Enable is enabled for the respective attachment. Otherwise the color write mask is ignored and writes to all components of the

attachment are disabled.

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkColorComponentFlags;
```

VkColorComponentFlags is a bitmask type for setting a mask of zero or more VkColorComponentFlagBits.

## 30.4. Color Write Enable

The VkPipelineColorWriteCreateInfoEXT structure is defined as:

```
// Provided by VK_EXT_color_write_enable
typedef struct VkPipelineColorWriteCreateInfoEXT {
    VkStructureType    sType;
    const void*        pNext;
    uint32_t           attachmentCount;
    const VkBool32*    pColorWriteEnables;
} VkPipelineColorWriteCreateInfoEXT;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- attachmentCount is the number of VkBool32 elements in pColorWriteEnables.
- pColorWriteEnables is a pointer to an array of per target attachment boolean values specifying whether color writes are enabled for the given attachment.

When this structure is included in the pNext chain of VkPipelineColorBlendStateCreateInfo, it defines per-attachment color write state. If this structure is not included in the pNext chain, it is equivalent to specifying this structure with attachmentCount equal to the attachmentCount member of VkPipelineColorBlendStateCreateInfo, and pColorWriteEnables pointing to an array of as many VK_TRUE values.

If the colorWriteEnable feature is not enabled on the device, all VkBool32 elements in the pColorWriteEnables array **must** be VK_TRUE.

Color Write Enable interacts with the Color Write Mask as follows:

- If colorWriteEnable is VK_TRUE, writes to the attachment are determined by the colorWriteMask.
- If colorWriteEnable is VK_FALSE, the colorWriteMask is ignored and writes to all components of the attachment are disabled. This is equivalent to specifying a colorWriteMask of 0.

### Valid Usage

- VUID-VkPipelineColorWriteCreateInfoEXT-pAttachments-04801
  If the colorWriteEnable feature is not enabled, all elements of pColorWriteEnables **must** be

VK_TRUE

- VUID-VkPipelineColorWriteCreateInfoEXT-attachmentCount-07608
  If the pipeline is being created with VK_DYNAMIC_STATE_COLOR_BLEND_ADVANCED_EXT, VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT, VK_DYNAMIC_STATE_COLOR_BLEND_EQUATION_EXT, or VK_DYNAMIC_STATE_COLOR_WRITE_MASK_EXT dynamic states not set, attachmentCount **must** be equal to the attachmentCount member of the VkPipelineColorBlendStateCreateInfo structure specified during pipeline creation
- VUID-VkPipelineColorWriteCreateInfoEXT-attachmentCount-06655
  attachmentCount **must** be less than or equal to the maxColorAttachments member of VkPhysicalDeviceLimits

### Valid Usage (Implicit)

- VUID-VkPipelineColorWriteCreateInfoEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PIPELINE_COLOR_WRITE_CREATE_INFO_EXT
- VUID-VkPipelineColorWriteCreateInfoEXT-pColorWriteEnables-parameter
  If attachmentCount is not 0, pColorWriteEnables **must** be a valid pointer to an array of attachmentCount VkBool32 values

To dynamically enable or disable writes to a color attachment, call:

```
// Provided by VK_EXT_color_write_enable
void                                        vkCmdSetColorWriteEnableEXT(
    VkCommandBuffer                                 commandBuffer,
    uint32_t                                        attachmentCount,
    const VkBool32*                                 pColorWriteEnables);
```

- commandBuffer is the command buffer into which the command will be recorded.
- attachmentCount is the number of VkBool32 elements in pColorWriteEnables.
- pColorWriteEnables is a pointer to an array of per target attachment boolean values specifying whether color writes are enabled for the given attachment.

This command sets the color write enables for subsequent drawing commands when drawing using shader objects, or when the graphics pipeline is created with VK_DYNAMIC_STATE_COLOR_WRITE_ENABLE_EXT set in VkPipelineDynamicStateCreateInfo::pDynamicStates. Otherwise, this state is specified by the VkPipelineColorWriteCreateInfoEXT::pColorWriteEnables values used to create the currently active pipeline.

### Valid Usage

- VUID-vkCmdSetColorWriteEnableEXT-None-04803
  The colorWriteEnable feature **must** be enabled
- VUID-vkCmdSetColorWriteEnableEXT-attachmentCount-06656

`attachmentCount` **must** be less than or equal to the `maxColorAttachments` member of `VkPhysicalDeviceLimits`

### Valid Usage (Implicit)

- VUID-vkCmdSetColorWriteEnableEXT-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdSetColorWriteEnableEXT-pColorWriteEnables-parameter
  `pColorWriteEnables` **must** be a valid pointer to an array of `attachmentCount` VkBool32 values
- VUID-vkCmdSetColorWriteEnableEXT-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdSetColorWriteEnableEXT-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- VUID-vkCmdSetColorWriteEnableEXT-videocoding
  This command **must** only be called outside of a video coding scope
- VUID-vkCmdSetColorWriteEnableEXT-attachmentCount-arraylength
  `attachmentCount` **must** be greater than `0`

### Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

### Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics | State |

## 30.5. Framebuffer Query Instructions

To query the tile properties from the attachments in framebuffer, call:

```
// Provided by VK_QCOM_tile_properties
VkResult vkGetFramebufferTilePropertiesQCOM(
    VkDevice                                    device,
    VkFramebuffer                               framebuffer,
    uint32_t*                                   pPropertiesCount,
    VkTilePropertiesQCOM*                       pProperties);
```

- `device` is a logical device associated with the framebuffer.
- `framebuffer` is a handle of the framebuffer to query.
- `pPropertiesCount` is a pointer to an integer related to the number of tile properties available or queried, as described below.
- `pProperties` is either `NULL` or a pointer to an array of VkTilePropertiesQCOM structures.

If `pProperties` is `NULL`, then the number of tile properties available is returned in `pPropertiesCount`. Otherwise, `pPropertiesCount` **must** point to a variable set by the application to the number of elements in the `pProperties` array, and on return the variable is overwritten with the number of properties actually written to `pProperties`. If `pPropertiesCount` is less than the number of tile properties available, at most `pPropertiesCount` structures will be written, and `VK_INCOMPLETE` will be returned instead of `VK_SUCCESS`, to indicate that not all the available properties were returned.

The number of tile properties available is determined by the number of merged subpasses, and each tile property is associated with a merged subpass. There will be at most as many properties as there are subpasses within the render pass. To obtain the tile properties for a given merged subpass, the `pProperties` array can be indexed using the `postMergeIndex` value provided in VkRenderPassSubpassFeedbackInfoEXT.

### Valid Usage (Implicit)

- VUID-vkGetFramebufferTilePropertiesQCOM-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkGetFramebufferTilePropertiesQCOM-framebuffer-parameter
  `framebuffer` **must** be a valid VkFramebuffer handle
- VUID-vkGetFramebufferTilePropertiesQCOM-pPropertiesCount-parameter
  `pPropertiesCount` **must** be a valid pointer to a `uint32_t` value
- VUID-vkGetFramebufferTilePropertiesQCOM-pProperties-parameter
  If the value referenced by `pPropertiesCount` is not `0`, and `pProperties` is not `NULL`, `pProperties` **must** be a valid pointer to an array of `pPropertiesCount` VkTilePropertiesQCOM structures
- VUID-vkGetFramebufferTilePropertiesQCOM-framebuffer-parent
  `framebuffer` **must** have been created, allocated, or retrieved from `device`

## Return Codes

### Success

- `VK_SUCCESS`
- `VK_INCOMPLETE`

### Failure

None

# Chapter 31. Dispatching Commands

*Dispatching commands* (commands with `Dispatch` in the name) provoke work in a compute pipeline. Dispatching commands are recorded into a command buffer and when executed by a queue, will produce work which executes according to the bound compute pipeline. A compute pipeline **must** be bound to a command buffer before any dispatching commands are recorded in that command buffer.

To record a dispatch, call:

```
// Provided by VK_VERSION_1_0
void vkCmdDispatch(
    VkCommandBuffer                             commandBuffer,
    uint32_t                                    groupCountX,
    uint32_t                                    groupCountY,
    uint32_t                                    groupCountZ);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `groupCountX` is the number of local workgroups to dispatch in the X dimension.
- `groupCountY` is the number of local workgroups to dispatch in the Y dimension.
- `groupCountZ` is the number of local workgroups to dispatch in the Z dimension.

When the command is executed, a global workgroup consisting of `groupCountX` × `groupCountY` × `groupCountZ` local workgroups is assembled.

### Valid Usage

- VUID-vkCmdDispatch-magFilter-04553
  If a VkSampler created with `magFilter` or `minFilter` equal to `VK_FILTER_LINEAR`, `reductionMode` equal to `VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE`, and `compareEnable` equal to `VK_FALSE` is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT`
- VUID-vkCmdDispatch-magFilter-09598
  If a VkSampler created with `magFilter` or `minFilter` equal to `VK_FILTER_LINEAR` and `reductionMode` equal to either `VK_SAMPLER_REDUCTION_MODE_MIN` or `VK_SAMPLER_REDUCTION_MODE_MAX` is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT`
- VUID-vkCmdDispatch-mipmapMode-04770
  If a VkSampler created with `mipmapMode` equal to `VK_SAMPLER_MIPMAP_MODE_LINEAR`, `reductionMode` equal to `VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE`, and `compareEnable` equal to `VK_FALSE` is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT`

- VUID-vkCmdDispatch-mipmapMode-09599
  If a VkSampler created with mipmapMode equal to VK_SAMPLER_MIPMAP_MODE_LINEAR and reductionMode equal to either VK_SAMPLER_REDUCTION_MODE_MIN or VK_SAMPLER_REDUCTION_MODE_MAX is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT
- VUID-vkCmdDispatch-unnormalizedCoordinates-09635
  If a VkSampler created with unnormalizedCoordinates equal to VK_TRUE is used to sample a VkImageView as a result of this command, then the image view's levelCount and layerCount **must** be 1
- VUID-vkCmdDispatch-unnormalizedCoordinates-09636
  If a VkSampler created with unnormalizedCoordinates equal to VK_TRUE is used to sample a VkImageView as a result of this command, then the image view's viewType **must** be VK_IMAGE_VIEW_TYPE_1D or VK_IMAGE_VIEW_TYPE_2D
- VUID-vkCmdDispatch-None-06479
  If a VkImageView is sampled with depth comparison, the image view's format features **must** contain VK_FORMAT_FEATURE_2_SAMPLED_IMAGE_DEPTH_COMPARISON_BIT
- VUID-vkCmdDispatch-None-02691
  If a VkImageView is accessed using atomic operations as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_STORAGE_IMAGE_ATOMIC_BIT
- VUID-vkCmdDispatch-None-07888
  If a VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER descriptor is accessed using atomic operations as a result of this command, then the storage texel buffer's format features **must** contain VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_ATOMIC_BIT
- VUID-vkCmdDispatch-None-02692
  If a VkImageView is sampled with VK_FILTER_CUBIC_EXT as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_CUBIC_BIT_EXT
- VUID-vkCmdDispatch-None-02693
  If the VK_EXT_filter_cubic extension is not enabled and any VkImageView is sampled with VK_FILTER_CUBIC_EXT as a result of this command, it **must** not have a VkImageViewType of VK_IMAGE_VIEW_TYPE_3D, VK_IMAGE_VIEW_TYPE_CUBE, or VK_IMAGE_VIEW_TYPE_CUBE_ARRAY
- VUID-vkCmdDispatch-filterCubic-02694
  Any VkImageView being sampled with VK_FILTER_CUBIC_EXT as a result of this command **must** have a VkImageViewType and format that supports cubic filtering, as specified by VkFilterCubicImageViewImageFormatPropertiesEXT::filterCubic returned by vkGetPhysicalDeviceImageFormatProperties2
- VUID-vkCmdDispatch-filterCubicMinmax-02695
  Any VkImageView being sampled with VK_FILTER_CUBIC_EXT with a reduction mode of either VK_SAMPLER_REDUCTION_MODE_MIN or VK_SAMPLER_REDUCTION_MODE_MAX as a result of this command **must** have a VkImageViewType and format that supports cubic filtering together with minmax filtering, as specified by VkFilterCubicImageViewImageFormatPropertiesEXT::filterCubicMinmax returned by

vkGetPhysicalDeviceImageFormatProperties2

- VUID-vkCmdDispatch-cubicRangeClamp-09212
  If the `cubicRangeClamp` feature is not enabled, then any VkImageView being sampled with `VK_FILTER_CUBIC_EXT` as a result of this command **must** not have a VkSamplerReductionModeCreateInfo::`reductionMode` equal to `VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE_RANGECLAMP_QCOM`
- VUID-vkCmdDispatch-reductionMode-09213
  Any VkImageView being sampled with a VkSamplerReductionModeCreateInfo::`reductionMode` equal to `VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE_RANGECLAMP_QCOM` as a result of this command **must** sample with `VK_FILTER_CUBIC_EXT`
- VUID-vkCmdDispatch-selectableCubicWeights-09214
  If the `selectableCubicWeights` feature is not enabled, then any VkImageView being sampled with `VK_FILTER_CUBIC_EXT` as a result of this command **must** have VkSamplerCubicWeightsCreateInfoQCOM::`cubicWeights` equal to `VK_CUBIC_FILTER_WEIGHTS_CATMULL_ROM_QCOM`
- VUID-vkCmdDispatch-flags-02696
  Any VkImage created with a VkImageCreateInfo::`flags` containing `VK_IMAGE_CREATE_CORNER_SAMPLED_BIT_NV` sampled as a result of this command **must** only be sampled using a VkSamplerAddressMode of `VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_EDGE`
- VUID-vkCmdDispatch-OpTypeImage-07027
  For any VkImageView being written as a storage image where the image format field of the `OpTypeImage` is `Unknown`, the view's format features **must** contain `VK_FORMAT_FEATURE_2_STORAGE_WRITE_WITHOUT_FORMAT_BIT`
- VUID-vkCmdDispatch-OpTypeImage-07028
  For any VkImageView being read as a storage image where the image format field of the `OpTypeImage` is `Unknown`, the view's format features **must** contain `VK_FORMAT_FEATURE_2_STORAGE_READ_WITHOUT_FORMAT_BIT`
- VUID-vkCmdDispatch-OpTypeImage-07029
  For any VkBufferView being written as a storage texel buffer where the image format field of the `OpTypeImage` is `Unknown`, the view's buffer features **must** contain `VK_FORMAT_FEATURE_2_STORAGE_WRITE_WITHOUT_FORMAT_BIT`
- VUID-vkCmdDispatch-OpTypeImage-07030
  Any VkBufferView being read as a storage texel buffer where the image format field of the `OpTypeImage` is `Unknown` then the view's buffer features **must** contain `VK_FORMAT_FEATURE_2_STORAGE_READ_WITHOUT_FORMAT_BIT`
- VUID-vkCmdDispatch-None-08600
  For each set *n* that is statically used by a bound shader, a descriptor set **must** have been bound to *n* at the same pipeline bind point, with a VkPipelineLayout that is compatible for set *n*, with the VkPipelineLayout used to create the current VkPipeline or the VkDescriptorSetLayout array used to create the current VkShaderEXT , as described in Pipeline Layout Compatibility
- VUID-vkCmdDispatch-None-08601
  For each push constant that is statically used by a bound shader, a push constant value **must** have been set for the same pipeline bind point, with a VkPipelineLayout that is

compatible for push constants, with the VkPipelineLayout used to create the current VkPipeline or the VkDescriptorSetLayout array used to create the current VkShaderEXT , as described in Pipeline Layout Compatibility

- VUID-vkCmdDispatch-None-10068
  For each array of resources that is used by a bound shader, the indices used to access members of the array **must** be less than the descriptor count for the identified binding in the descriptor sets used by this command
- VUID-vkCmdDispatch-maintenance4-08602
  If the `maintenance4` feature is not enabled, then for each push constant that is statically used by a bound shader, a push constant value **must** have been set for the same pipeline bind point, with a VkPipelineLayout that is compatible for push constants, with the VkPipelineLayout used to create the current VkPipeline or the VkDescriptorSetLayout and VkPushConstantRange arrays used to create the current VkShaderEXT , as described in Pipeline Layout Compatibility
- VUID-vkCmdDispatch-None-08114
  Descriptors in each bound descriptor set, specified via vkCmdBindDescriptorSets, **must** be valid as described by descriptor validity if they are statically used by the VkPipeline bound to the pipeline bind point used by this command and the bound VkPipeline was not created with `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`
- VUID-vkCmdDispatch-None-08115
  If the descriptors used by the VkPipeline bound to the pipeline bind point were specified via vkCmdBindDescriptorSets, the bound VkPipeline **must** have been created without `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`
- VUID-vkCmdDispatch-None-08116
  Descriptors in bound descriptor buffers, specified via vkCmdSetDescriptorBufferOffsetsEXT, **must** be valid if they are dynamically used by the VkPipeline bound to the pipeline bind point used by this command and the bound VkPipeline was created with `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`
- VUID-vkCmdDispatch-None-08604
  Descriptors in bound descriptor buffers, specified via vkCmdSetDescriptorBufferOffsetsEXT, **must** be valid if they are dynamically used by any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command
- VUID-vkCmdDispatch-None-08117
  If the descriptors used by the VkPipeline bound to the pipeline bind point were specified via vkCmdSetDescriptorBufferOffsetsEXT, the bound VkPipeline **must** have been created with `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`
- VUID-vkCmdDispatch-None-08119
  If a descriptor is dynamically used with a VkPipeline created with `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`, the descriptor memory **must** be resident
- VUID-vkCmdDispatch-None-08605
  If a descriptor is dynamically used with a VkShaderEXT created with a `VkDescriptorSetLayout` that was created with `VK_DESCRIPTOR_SET_LAYOUT_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`, the descriptor memory **must**

be resident

- VUID-vkCmdDispatch-None-08606
  If the `shaderObject` feature is not enabled, a valid pipeline **must** be bound to the pipeline bind point used by this command
- VUID-vkCmdDispatch-None-08608
  If a pipeline is bound to the pipeline bind point used by this command, there **must** not have been any calls to dynamic state setting commands for any state specified statically in the VkPipeline object bound to the pipeline bind point used by this command, since that pipeline was bound
- VUID-vkCmdDispatch-None-08609
  If the VkPipeline object bound to the pipeline bind point used by this command or any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a VkSampler object that uses unnormalized coordinates, that sampler **must** not be used to sample from any VkImage with a VkImageView of the type `VK_IMAGE_VIEW_TYPE_3D`, `VK_IMAGE_VIEW_TYPE_CUBE`, `VK_IMAGE_VIEW_TYPE_1D_ARRAY`, `VK_IMAGE_VIEW_TYPE_2D_ARRAY` or `VK_IMAGE_VIEW_TYPE_CUBE_ARRAY`, in any shader stage
- VUID-vkCmdDispatch-None-08610
  If the VkPipeline object bound to the pipeline bind point used by this command or any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a VkSampler object that uses unnormalized coordinates, that sampler **must** not be used with any of the SPIR-V `OpImageSample*` or `OpImageSparseSample*` instructions with `ImplicitLod`, `Dref` or `Proj` in their name, in any shader stage
- VUID-vkCmdDispatch-None-08611
  If the VkPipeline object bound to the pipeline bind point used by this command or any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a VkSampler object that uses unnormalized coordinates, that sampler **must** not be used with any of the SPIR-V `OpImageSample*` or `OpImageSparseSample*` instructions that includes a LOD bias or any offset values, in any shader stage
- VUID-vkCmdDispatch-None-08607
  If the `shaderObject` is enabled, either a valid pipeline **must** be bound to the pipeline bind point used by this command, or a valid combination of valid and VK_NULL_HANDLE shader objects **must** be bound to every supported shader stage corresponding to the pipeline bind point used by this command
- VUID-vkCmdDispatch-uniformBuffers-06935
  If any stage of the VkPipeline object bound to the pipeline bind point used by this command accesses a uniform buffer, and that stage was created without enabling either `VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_EXT` or `VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_2_EXT` for `uniformBuffers`, and the `robustBufferAccess` feature is not enabled, that stage **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point
- VUID-vkCmdDispatch-None-08612
  If the `robustBufferAccess` feature is not enabled, and any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a uniform buffer, it **must** not access values outside of the range of the buffer as specified in the

descriptor set bound to the same pipeline bind point

- VUID-vkCmdDispatch-storageBuffers-06936
  If any stage of the VkPipeline object bound to the pipeline bind point used by this command accesses a storage buffer, and that stage was created without enabling either `VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_EXT` or `VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_2_EXT` for `storageBuffers`, and the `robustBufferAccess` feature is not enabled, that stage **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point
- VUID-vkCmdDispatch-None-08613
  If the `robustBufferAccess` feature is not enabled, and any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a storage buffer, it **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point
- VUID-vkCmdDispatch-commandBuffer-02707
  If `commandBuffer` is an unprotected command buffer and `protectedNoFault` is not supported, any resource accessed by bound shaders **must** not be a protected resource
- VUID-vkCmdDispatch-None-06550
  If a bound shader accesses a VkSampler or VkImageView object that enables sampler $Y'C_BC_R$ conversion, that object **must** only be used with `OpImageSample*` or `OpImageSparseSample*` instructions
- VUID-vkCmdDispatch-ConstOffset-06551
  If a bound shader accesses a VkSampler or VkImageView object that enables sampler $Y'C_BC_R$ conversion, that object **must** not use the `ConstOffset` and `Offset` operands
- VUID-vkCmdDispatch-viewType-07752
  If a VkImageView is accessed as a result of this command, then the image view's `viewType` **must** match the `Dim` operand of the `OpTypeImage` as described in Compatibility Between SPIR-V Image Dimensions and Vulkan ImageView Types
- VUID-vkCmdDispatch-format-07753
  If a VkImageView is accessed as a result of this command, then the numeric type of the image view's `format` and the `Sampled Type` operand of the `OpTypeImage` **must** match
- VUID-vkCmdDispatch-OpImageWrite-08795
  If a VkImageView created with a format other than `VK_FORMAT_A8_UNORM_KHR` is accessed using `OpImageWrite` as a result of this command, then the `Type` of the `Texel` operand of that instruction **must** have at least as many components as the image view's format
- VUID-vkCmdDispatch-OpImageWrite-08796
  If a VkImageView created with the format `VK_FORMAT_A8_UNORM_KHR` is accessed using `OpImageWrite` as a result of this command, then the `Type` of the `Texel` operand of that instruction **must** have four components
- VUID-vkCmdDispatch-OpImageWrite-04469
  If a VkBufferView is accessed using `OpImageWrite` as a result of this command, then the `Type` of the `Texel` operand of that instruction **must** have at least as many components as the buffer view's format

- VUID-vkCmdDispatch-SampledType-04470
  If a VkImageView with a VkFormat that has a 64-bit component width is accessed as a result of this command, the `SampledType` of the `OpTypeImage` operand of that instruction **must** have a `Width` of 64
- VUID-vkCmdDispatch-SampledType-04471
  If a VkImageView with a VkFormat that has a component width less than 64-bit is accessed as a result of this command, the `SampledType` of the `OpTypeImage` operand of that instruction **must** have a `Width` of 32
- VUID-vkCmdDispatch-SampledType-04472
  If a VkBufferView with a VkFormat that has a 64-bit component width is accessed as a result of this command, the `SampledType` of the `OpTypeImage` operand of that instruction **must** have a `Width` of 64
- VUID-vkCmdDispatch-SampledType-04473
  If a VkBufferView with a VkFormat that has a component width less than 64-bit is accessed as a result of this command, the `SampledType` of the `OpTypeImage` operand of that instruction **must** have a `Width` of 32
- VUID-vkCmdDispatch-sparseImageInt64Atomics-04474
  If the `sparseImageInt64Atomics` feature is not enabled, VkImage objects created with the `VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT` flag **must** not be accessed by atomic instructions through an `OpTypeImage` with a `SampledType` with a `Width` of 64 by this command
- VUID-vkCmdDispatch-sparseImageInt64Atomics-04475
  If the `sparseImageInt64Atomics` feature is not enabled, VkBuffer objects created with the `VK_BUFFER_CREATE_SPARSE_RESIDENCY_BIT` flag **must** not be accessed by atomic instructions through an `OpTypeImage` with a `SampledType` with a `Width` of 64 by this command
- VUID-vkCmdDispatch-OpImageWeightedSampleQCOM-06971
  If `OpImageWeightedSampleQCOM` is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_2_WEIGHT_SAMPLED_IMAGE_BIT_QCOM`
- VUID-vkCmdDispatch-OpImageWeightedSampleQCOM-06972
  If `OpImageWeightedSampleQCOM` uses a VkImageView as a sample weight image as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_2_WEIGHT_IMAGE_BIT_QCOM`
- VUID-vkCmdDispatch-OpImageBoxFilterQCOM-06973
  If `OpImageBoxFilterQCOM` is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_2_BOX_FILTER_SAMPLED_BIT_QCOM`
- VUID-vkCmdDispatch-OpImageBlockMatchSSDQCOM-06974
  If `OpImageBlockMatchSSDQCOM` is used to read from an VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM`
- VUID-vkCmdDispatch-OpImageBlockMatchSADQCOM-06975
  If `OpImageBlockMatchSADQCOM` is used to read from an VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM`

- VUID-vkCmdDispatch-OpImageBlockMatchSADQCOM-06976
  If `OpImageBlockMatchSADQCOM` or OpImageBlockMatchSSDQCOM is used to read from a reference image as result of this command, then the specified reference coordinates **must** not fail integer texel coordinate validation
- VUID-vkCmdDispatch-OpImageWeightedSampleQCOM-06977
  If `OpImageWeightedSampleQCOM`, `OpImageBoxFilterQCOM`, `OpImageBlockMatchWindowSSDQCOM`, `OpImageBlockMatchWindowSADQCOM`, `OpImageBlockMatchGatherSSDQCOM`, `OpImageBlockMatchGatherSADQCOM`, `OpImageBlockMatchSSDQCOM`, or `OpImageBlockMatchSADQCOM` uses a VkSampler as a result of this command, then the sampler **must** have been created with `VK_SAMPLER_CREATE_IMAGE_PROCESSING_BIT_QCOM`
- VUID-vkCmdDispatch-OpImageWeightedSampleQCOM-06978
  If any command other than `OpImageWeightedSampleQCOM`, `OpImageBoxFilterQCOM`, `OpImageBlockMatchWindowSSDQCOM`, `OpImageBlockMatchWindowSADQCOM`, `OpImageBlockMatchGatherSSDQCOM`, `OpImageBlockMatchGatherSADQCOM`, `OpImageBlockMatchSSDQCOM`, or `OpImageBlockMatchSADQCOM` uses a VkSampler as a result of this command, then the sampler **must** not have been created with `VK_SAMPLER_CREATE_IMAGE_PROCESSING_BIT_QCOM`
- VUID-vkCmdDispatch-OpImageBlockMatchWindow-09215
  If a `OpImageBlockMatchWindow*QCOM` or `OpImageBlockMatchGather*QCOM` instruction is used to read from an VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM`
- VUID-vkCmdDispatch-OpImageBlockMatchWindow-09216
  If a `OpImageBlockMatchWindow*QCOM` or `OpImageBlockMatchGather*QCOM` instruction is used to read from an VkImageView as a result of this command, then the image view's format **must** be a single-component format
- VUID-vkCmdDispatch-OpImageBlockMatchWindow-09217
  If a `OpImageBlockMatchWindow*QCOM` or `OpImageBlockMatchGather*QCOM` read from a reference image as result of this command, then the specified reference coordinates **must** not fail integer texel coordinate validation
- VUID-vkCmdDispatch-None-07288
  Any shader invocation executed by this command **must** terminate
- VUID-vkCmdDispatch-None-09600
  If a descriptor with type equal to any of `VK_DESCRIPTOR_TYPE_SAMPLE_WEIGHT_IMAGE_QCOM`, `VK_DESCRIPTOR_TYPE_BLOCK_MATCH_IMAGE_QCOM`, `VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE`, `VK_DESCRIPTOR_TYPE_STORAGE_IMAGE`, or `VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT` is accessed as a result of this command, the image subresource identified by that descriptor **must** be in the image layout identified when the descriptor was written
- VUID-vkCmdDispatch-commandBuffer-02712
  If `commandBuffer` is a protected command buffer and `protectedNoFault` is not supported, any resource written to by the `VkPipeline` object bound to the pipeline bind point used by this command **must** not be an unprotected resource
- VUID-vkCmdDispatch-commandBuffer-02713
  If `commandBuffer` is a protected command buffer and `protectedNoFault` is not supported, pipeline stages other than the framebuffer-space and compute stages in the `VkPipeline`

object bound to the pipeline bind point used by this command **must** not write to any resource

- VUID-vkCmdDispatch-commandBuffer-04617
  If any of the shader stages of the `VkPipeline` bound to the pipeline bind point used by this command uses the `RayQueryKHR` capability, then `commandBuffer` **must** not be a protected command buffer
- VUID-vkCmdDispatch-groupCountX-00386
  `groupCountX` **must** be less than or equal to `VkPhysicalDeviceLimits::maxComputeWorkGroupCount`[0]
- VUID-vkCmdDispatch-groupCountY-00387
  `groupCountY` **must** be less than or equal to `VkPhysicalDeviceLimits::maxComputeWorkGroupCount`[1]
- VUID-vkCmdDispatch-groupCountZ-00388
  `groupCountZ` **must** be less than or equal to `VkPhysicalDeviceLimits::maxComputeWorkGroupCount`[2]

## Valid Usage (Implicit)

- VUID-vkCmdDispatch-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdDispatch-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdDispatch-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support compute operations
- VUID-vkCmdDispatch-renderpass
  This command **must** only be called outside of a render pass instance
- VUID-vkCmdDispatch-videocoding
  This command **must** only be called outside of a video coding scope

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Outside | Outside | Compute | Action |

To record an indirect dispatching command, call:

```
// Provided by VK_VERSION_1_0
void vkCmdDispatchIndirect(
    VkCommandBuffer                                 commandBuffer,
    VkBuffer                                        buffer,
    VkDeviceSize                                    offset);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `buffer` is the buffer containing dispatch parameters.
- `offset` is the byte offset into `buffer` where parameters begin.

`vkCmdDispatchIndirect` behaves similarly to vkCmdDispatch except that the parameters are read by the device from a buffer during execution. The parameters of the dispatch are encoded in a VkDispatchIndirectCommand structure taken from `buffer` starting at `offset`.

## Valid Usage

- VUID-vkCmdDispatchIndirect-magFilter-04553
  If a VkSampler created with `magFilter` or `minFilter` equal to `VK_FILTER_LINEAR`, `reductionMode` equal to `VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE`, and `compareEnable` equal to `VK_FALSE` is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT`
- VUID-vkCmdDispatchIndirect-magFilter-09598
  If a VkSampler created with `magFilter` or `minFilter` equal to `VK_FILTER_LINEAR` and `reductionMode` equal to either `VK_SAMPLER_REDUCTION_MODE_MIN` or `VK_SAMPLER_REDUCTION_MODE_MAX` is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT`
- VUID-vkCmdDispatchIndirect-mipmapMode-04770
  If a VkSampler created with `mipmapMode` equal to `VK_SAMPLER_MIPMAP_MODE_LINEAR`, `reductionMode` equal to `VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE`, and `compareEnable` equal to `VK_FALSE` is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT`

- VUID-vkCmdDispatchIndirect-mipmapMode-09599
  If a VkSampler created with mipmapMode equal to VK_SAMPLER_MIPMAP_MODE_LINEAR and reductionMode equal to either VK_SAMPLER_REDUCTION_MODE_MIN or VK_SAMPLER_REDUCTION_MODE_MAX is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT
- VUID-vkCmdDispatchIndirect-unnormalizedCoordinates-09635
  If a VkSampler created with unnormalizedCoordinates equal to VK_TRUE is used to sample a VkImageView as a result of this command, then the image view's levelCount and layerCount **must** be 1
- VUID-vkCmdDispatchIndirect-unnormalizedCoordinates-09636
  If a VkSampler created with unnormalizedCoordinates equal to VK_TRUE is used to sample a VkImageView as a result of this command, then the image view's viewType **must** be VK_IMAGE_VIEW_TYPE_1D or VK_IMAGE_VIEW_TYPE_2D
- VUID-vkCmdDispatchIndirect-None-06479
  If a VkImageView is sampled with depth comparison, the image view's format features **must** contain VK_FORMAT_FEATURE_2_SAMPLED_IMAGE_DEPTH_COMPARISON_BIT
- VUID-vkCmdDispatchIndirect-None-02691
  If a VkImageView is accessed using atomic operations as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_STORAGE_IMAGE_ATOMIC_BIT
- VUID-vkCmdDispatchIndirect-None-07888
  If a VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER descriptor is accessed using atomic operations as a result of this command, then the storage texel buffer's format features **must** contain VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_ATOMIC_BIT
- VUID-vkCmdDispatchIndirect-None-02692
  If a VkImageView is sampled with VK_FILTER_CUBIC_EXT as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_CUBIC_BIT_EXT
- VUID-vkCmdDispatchIndirect-None-02693
  If the VK_EXT_filter_cubic extension is not enabled and any VkImageView is sampled with VK_FILTER_CUBIC_EXT as a result of this command, it **must** not have a VkImageViewType of VK_IMAGE_VIEW_TYPE_3D, VK_IMAGE_VIEW_TYPE_CUBE, or VK_IMAGE_VIEW_TYPE_CUBE_ARRAY
- VUID-vkCmdDispatchIndirect-filterCubic-02694
  Any VkImageView being sampled with VK_FILTER_CUBIC_EXT as a result of this command **must** have a VkImageViewType and format that supports cubic filtering, as specified by VkFilterCubicImageViewImageFormatPropertiesEXT::filterCubic returned by vkGetPhysicalDeviceImageFormatProperties2
- VUID-vkCmdDispatchIndirect-filterCubicMinmax-02695
  Any VkImageView being sampled with VK_FILTER_CUBIC_EXT with a reduction mode of either VK_SAMPLER_REDUCTION_MODE_MIN or VK_SAMPLER_REDUCTION_MODE_MAX as a result of this command **must** have a VkImageViewType and format that supports cubic filtering together with minmax filtering, as specified by VkFilterCubicImageViewImageFormatPropertiesEXT::filterCubicMinmax returned by

vkGetPhysicalDeviceImageFormatProperties2

- VUID-vkCmdDispatchIndirect-cubicRangeClamp-09212
  If the `cubicRangeClamp` feature is not enabled, then any VkImageView being sampled with `VK_FILTER_CUBIC_EXT` as a result of this command **must** not have a VkSamplerReductionModeCreateInfo::`reductionMode` equal to `VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE_RANGECLAMP_QCOM`
- VUID-vkCmdDispatchIndirect-reductionMode-09213
  Any VkImageView being sampled with a VkSamplerReductionModeCreateInfo::`reductionMode` equal to `VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE_RANGECLAMP_QCOM` as a result of this command **must** sample with `VK_FILTER_CUBIC_EXT`
- VUID-vkCmdDispatchIndirect-selectableCubicWeights-09214
  If the `selectableCubicWeights` feature is not enabled, then any VkImageView being sampled with `VK_FILTER_CUBIC_EXT` as a result of this command **must** have VkSamplerCubicWeightsCreateInfoQCOM::`cubicWeights` equal to `VK_CUBIC_FILTER_WEIGHTS_CATMULL_ROM_QCOM`
- VUID-vkCmdDispatchIndirect-flags-02696
  Any VkImage created with a VkImageCreateInfo::`flags` containing `VK_IMAGE_CREATE_CORNER_SAMPLED_BIT_NV` sampled as a result of this command **must** only be sampled using a VkSamplerAddressMode of `VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_EDGE`
- VUID-vkCmdDispatchIndirect-OpTypeImage-07027
  For any VkImageView being written as a storage image where the image format field of the `OpTypeImage` is `Unknown`, the view's format features **must** contain `VK_FORMAT_FEATURE_2_STORAGE_WRITE_WITHOUT_FORMAT_BIT`
- VUID-vkCmdDispatchIndirect-OpTypeImage-07028
  For any VkImageView being read as a storage image where the image format field of the `OpTypeImage` is `Unknown`, the view's format features **must** contain `VK_FORMAT_FEATURE_2_STORAGE_READ_WITHOUT_FORMAT_BIT`
- VUID-vkCmdDispatchIndirect-OpTypeImage-07029
  For any VkBufferView being written as a storage texel buffer where the image format field of the `OpTypeImage` is `Unknown`, the view's buffer features **must** contain `VK_FORMAT_FEATURE_2_STORAGE_WRITE_WITHOUT_FORMAT_BIT`
- VUID-vkCmdDispatchIndirect-OpTypeImage-07030
  Any VkBufferView being read as a storage texel buffer where the image format field of the `OpTypeImage` is `Unknown` then the view's buffer features **must** contain `VK_FORMAT_FEATURE_2_STORAGE_READ_WITHOUT_FORMAT_BIT`
- VUID-vkCmdDispatchIndirect-None-08600
  For each set *n* that is statically used by a bound shader, a descriptor set **must** have been bound to *n* at the same pipeline bind point, with a VkPipelineLayout that is compatible for set *n*, with the VkPipelineLayout used to create the current VkPipeline or the VkDescriptorSetLayout array used to create the current VkShaderEXT , as described in Pipeline Layout Compatibility
- VUID-vkCmdDispatchIndirect-None-08601
  For each push constant that is statically used by a bound shader, a push constant value **must** have been set for the same pipeline bind point, with a VkPipelineLayout that is

compatible for push constants, with the VkPipelineLayout used to create the current VkPipeline or the VkDescriptorSetLayout array used to create the current VkShaderEXT , as described in Pipeline Layout Compatibility

- VUID-vkCmdDispatchIndirect-None-10068
  For each array of resources that is used by a bound shader, the indices used to access members of the array **must** be less than the descriptor count for the identified binding in the descriptor sets used by this command
- VUID-vkCmdDispatchIndirect-maintenance4-08602
  If the `maintenance4` feature is not enabled, then for each push constant that is statically used by a bound shader, a push constant value **must** have been set for the same pipeline bind point, with a VkPipelineLayout that is compatible for push constants, with the VkPipelineLayout used to create the current VkPipeline or the VkDescriptorSetLayout and VkPushConstantRange arrays used to create the current VkShaderEXT , as described in Pipeline Layout Compatibility
- VUID-vkCmdDispatchIndirect-None-08114
  Descriptors in each bound descriptor set, specified via vkCmdBindDescriptorSets, **must** be valid as described by descriptor validity if they are statically used by the VkPipeline bound to the pipeline bind point used by this command and the bound VkPipeline was not created with `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`
- VUID-vkCmdDispatchIndirect-None-08115
  If the descriptors used by the VkPipeline bound to the pipeline bind point were specified via vkCmdBindDescriptorSets, the bound VkPipeline **must** have been created without `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`
- VUID-vkCmdDispatchIndirect-None-08116
  Descriptors in bound descriptor buffers, specified via vkCmdSetDescriptorBufferOffsetsEXT, **must** be valid if they are dynamically used by the VkPipeline bound to the pipeline bind point used by this command and the bound VkPipeline was created with `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`
- VUID-vkCmdDispatchIndirect-None-08604
  Descriptors in bound descriptor buffers, specified via vkCmdSetDescriptorBufferOffsetsEXT, **must** be valid if they are dynamically used by any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command
- VUID-vkCmdDispatchIndirect-None-08117
  If the descriptors used by the VkPipeline bound to the pipeline bind point were specified via vkCmdSetDescriptorBufferOffsetsEXT, the bound VkPipeline **must** have been created with `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`
- VUID-vkCmdDispatchIndirect-None-08119
  If a descriptor is dynamically used with a VkPipeline created with `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`, the descriptor memory **must** be resident
- VUID-vkCmdDispatchIndirect-None-08605
  If a descriptor is dynamically used with a VkShaderEXT created with a `VkDescriptorSetLayout` that was created with `VK_DESCRIPTOR_SET_LAYOUT_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`, the descriptor memory **must**

be resident

- VUID-vkCmdDispatchIndirect-None-08606
  If the `shaderObject` feature is not enabled, a valid pipeline **must** be bound to the pipeline bind point used by this command
- VUID-vkCmdDispatchIndirect-None-08608
  If a pipeline is bound to the pipeline bind point used by this command, there **must** not have been any calls to dynamic state setting commands for any state specified statically in the VkPipeline object bound to the pipeline bind point used by this command, since that pipeline was bound
- VUID-vkCmdDispatchIndirect-None-08609
  If the VkPipeline object bound to the pipeline bind point used by this command or any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a VkSampler object that uses unnormalized coordinates, that sampler **must** not be used to sample from any VkImage with a VkImageView of the type `VK_IMAGE_VIEW_TYPE_3D`, `VK_IMAGE_VIEW_TYPE_CUBE`, `VK_IMAGE_VIEW_TYPE_1D_ARRAY`, `VK_IMAGE_VIEW_TYPE_2D_ARRAY` or `VK_IMAGE_VIEW_TYPE_CUBE_ARRAY`, in any shader stage
- VUID-vkCmdDispatchIndirect-None-08610
  If the VkPipeline object bound to the pipeline bind point used by this command or any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a VkSampler object that uses unnormalized coordinates, that sampler **must** not be used with any of the SPIR-V `OpImageSample*` or `OpImageSparseSample*` instructions with `ImplicitLod`, `Dref` or `Proj` in their name, in any shader stage
- VUID-vkCmdDispatchIndirect-None-08611
  If the VkPipeline object bound to the pipeline bind point used by this command or any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a VkSampler object that uses unnormalized coordinates, that sampler **must** not be used with any of the SPIR-V `OpImageSample*` or `OpImageSparseSample*` instructions that includes a LOD bias or any offset values, in any shader stage
- VUID-vkCmdDispatchIndirect-None-08607
  If the `shaderObject` is enabled, either a valid pipeline **must** be bound to the pipeline bind point used by this command, or a valid combination of valid and VK_NULL_HANDLE shader objects **must** be bound to every supported shader stage corresponding to the pipeline bind point used by this command
- VUID-vkCmdDispatchIndirect-uniformBuffers-06935
  If any stage of the VkPipeline object bound to the pipeline bind point used by this command accesses a uniform buffer, and that stage was created without enabling either `VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_EXT` or `VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_2_EXT` for `uniformBuffers`, and the `robustBufferAccess` feature is not enabled, that stage **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point
- VUID-vkCmdDispatchIndirect-None-08612
  If the `robustBufferAccess` feature is not enabled, and any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a uniform buffer, it **must** not access values outside of the range of the buffer as specified in the

descriptor set bound to the same pipeline bind point

- VUID-vkCmdDispatchIndirect-storageBuffers-06936
  If any stage of the VkPipeline object bound to the pipeline bind point used by this command accesses a storage buffer, and that stage was created without enabling either `VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_EXT` or `VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_2_EXT` for `storageBuffers`, and the `robustBufferAccess` feature is not enabled, that stage **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point
- VUID-vkCmdDispatchIndirect-None-08613
  If the `robustBufferAccess` feature is not enabled, and any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a storage buffer, it **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point
- VUID-vkCmdDispatchIndirect-commandBuffer-02707
  If `commandBuffer` is an unprotected command buffer and `protectedNoFault` is not supported, any resource accessed by bound shaders **must** not be a protected resource
- VUID-vkCmdDispatchIndirect-None-06550
  If a bound shader accesses a VkSampler or VkImageView object that enables sampler $Y'C_BC_R$ conversion, that object **must** only be used with `OpImageSample*` or `OpImageSparseSample*` instructions
- VUID-vkCmdDispatchIndirect-ConstOffset-06551
  If a bound shader accesses a VkSampler or VkImageView object that enables sampler $Y'C_BC_R$ conversion, that object **must** not use the `ConstOffset` and `Offset` operands
- VUID-vkCmdDispatchIndirect-viewType-07752
  If a VkImageView is accessed as a result of this command, then the image view's `viewType` **must** match the `Dim` operand of the `OpTypeImage` as described in Compatibility Between SPIR-V Image Dimensions and Vulkan ImageView Types
- VUID-vkCmdDispatchIndirect-format-07753
  If a VkImageView is accessed as a result of this command, then the numeric type of the image view's `format` and the `Sampled Type` operand of the `OpTypeImage` **must** match
- VUID-vkCmdDispatchIndirect-OpImageWrite-08795
  If a VkImageView created with a format other than `VK_FORMAT_A8_UNORM_KHR` is accessed using `OpImageWrite` as a result of this command, then the `Type` of the `Texel` operand of that instruction **must** have at least as many components as the image view's format
- VUID-vkCmdDispatchIndirect-OpImageWrite-08796
  If a VkImageView created with the format `VK_FORMAT_A8_UNORM_KHR` is accessed using `OpImageWrite` as a result of this command, then the `Type` of the `Texel` operand of that instruction **must** have four components
- VUID-vkCmdDispatchIndirect-OpImageWrite-04469
  If a VkBufferView is accessed using `OpImageWrite` as a result of this command, then the `Type` of the `Texel` operand of that instruction **must** have at least as many components as the buffer view's format

- VUID-vkCmdDispatchIndirect-SampledType-04470
  If a VkImageView with a VkFormat that has a 64-bit component width is accessed as a result of this command, the SampledType of the OpTypeImage operand of that instruction **must** have a Width of 64
- VUID-vkCmdDispatchIndirect-SampledType-04471
  If a VkImageView with a VkFormat that has a component width less than 64-bit is accessed as a result of this command, the SampledType of the OpTypeImage operand of that instruction **must** have a Width of 32
- VUID-vkCmdDispatchIndirect-SampledType-04472
  If a VkBufferView with a VkFormat that has a 64-bit component width is accessed as a result of this command, the SampledType of the OpTypeImage operand of that instruction **must** have a Width of 64
- VUID-vkCmdDispatchIndirect-SampledType-04473
  If a VkBufferView with a VkFormat that has a component width less than 64-bit is accessed as a result of this command, the SampledType of the OpTypeImage operand of that instruction **must** have a Width of 32
- VUID-vkCmdDispatchIndirect-sparseImageInt64Atomics-04474
  If the sparseImageInt64Atomics feature is not enabled, VkImage objects created with the VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT flag **must** not be accessed by atomic instructions through an OpTypeImage with a SampledType with a Width of 64 by this command
- VUID-vkCmdDispatchIndirect-sparseImageInt64Atomics-04475
  If the sparseImageInt64Atomics feature is not enabled, VkBuffer objects created with the VK_BUFFER_CREATE_SPARSE_RESIDENCY_BIT flag **must** not be accessed by atomic instructions through an OpTypeImage with a SampledType with a Width of 64 by this command
- VUID-vkCmdDispatchIndirect-OpImageWeightedSampleQCOM-06971
  If OpImageWeightedSampleQCOM is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_WEIGHT_SAMPLED_IMAGE_BIT_QCOM
- VUID-vkCmdDispatchIndirect-OpImageWeightedSampleQCOM-06972
  If OpImageWeightedSampleQCOM uses a VkImageView as a sample weight image as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_WEIGHT_IMAGE_BIT_QCOM
- VUID-vkCmdDispatchIndirect-OpImageBoxFilterQCOM-06973
  If OpImageBoxFilterQCOM is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_BOX_FILTER_SAMPLED_BIT_QCOM
- VUID-vkCmdDispatchIndirect-OpImageBlockMatchSSDQCOM-06974
  If OpImageBlockMatchSSDQCOM is used to read from an VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM
- VUID-vkCmdDispatchIndirect-OpImageBlockMatchSADQCOM-06975
  If OpImageBlockMatchSADQCOM is used to read from an VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM

- VUID-vkCmdDispatchIndirect-OpImageBlockMatchSADQCOM-06976
  If `OpImageBlockMatchSADQCOM` or OpImageBlockMatchSSDQCOM is used to read from a reference image as result of this command, then the specified reference coordinates **must** not fail integer texel coordinate validation
- VUID-vkCmdDispatchIndirect-OpImageWeightedSampleQCOM-06977
  If `OpImageWeightedSampleQCOM`, `OpImageBoxFilterQCOM`, `OpImageBlockMatchWindowSSDQCOM`, `OpImageBlockMatchWindowSADQCOM`, `OpImageBlockMatchGatherSSDQCOM`, `OpImageBlockMatchGatherSADQCOM`, `OpImageBlockMatchSSDQCOM`, or `OpImageBlockMatchSADQCOM` uses a VkSampler as a result of this command, then the sampler **must** have been created with `VK_SAMPLER_CREATE_IMAGE_PROCESSING_BIT_QCOM`
- VUID-vkCmdDispatchIndirect-OpImageWeightedSampleQCOM-06978
  If any command other than `OpImageWeightedSampleQCOM`, `OpImageBoxFilterQCOM`, `OpImageBlockMatchWindowSSDQCOM`, `OpImageBlockMatchWindowSADQCOM`, `OpImageBlockMatchGatherSSDQCOM`, `OpImageBlockMatchGatherSADQCOM`, `OpImageBlockMatchSSDQCOM`, or `OpImageBlockMatchSADQCOM` uses a VkSampler as a result of this command, then the sampler **must** not have been created with `VK_SAMPLER_CREATE_IMAGE_PROCESSING_BIT_QCOM`
- VUID-vkCmdDispatchIndirect-OpImageBlockMatchWindow-09215
  If a `OpImageBlockMatchWindow*QCOM` or `OpImageBlockMatchGather*QCOM` instruction is used to read from an VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM`
- VUID-vkCmdDispatchIndirect-OpImageBlockMatchWindow-09216
  If a `OpImageBlockMatchWindow*QCOM` or `OpImageBlockMatchGather*QCOM` instruction is used to read from an VkImageView as a result of this command, then the image view's format **must** be a single-component format
- VUID-vkCmdDispatchIndirect-OpImageBlockMatchWindow-09217
  If a `OpImageBlockMatchWindow*QCOM` or `OpImageBlockMatchGather*QCOM` read from a reference image as result of this command, then the specified reference coordinates **must** not fail integer texel coordinate validation
- VUID-vkCmdDispatchIndirect-None-07288
  Any shader invocation executed by this command **must** terminate
- VUID-vkCmdDispatchIndirect-None-09600
  If a descriptor with type equal to any of `VK_DESCRIPTOR_TYPE_SAMPLE_WEIGHT_IMAGE_QCOM`, `VK_DESCRIPTOR_TYPE_BLOCK_MATCH_IMAGE_QCOM`, `VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE`, `VK_DESCRIPTOR_TYPE_STORAGE_IMAGE`, or `VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT` is accessed as a result of this command, the image subresource identified by that descriptor **must** be in the image layout identified when the descriptor was written
- VUID-vkCmdDispatchIndirect-buffer-02708
  If `buffer` is non-sparse then it **must** be bound completely and contiguously to a single `VkDeviceMemory` object
- VUID-vkCmdDispatchIndirect-buffer-02709
  `buffer` **must** have been created with the `VK_BUFFER_USAGE_INDIRECT_BUFFER_BIT` bit set
- VUID-vkCmdDispatchIndirect-offset-02710

offset **must** be a multiple of 4

- VUID-vkCmdDispatchIndirect-commandBuffer-02711
  commandBuffer **must** not be a protected command buffer
- VUID-vkCmdDispatchIndirect-offset-00407
  The sum of offset and the size of VkDispatchIndirectCommand **must** be less than or equal to the size of buffer

## Valid Usage (Implicit)

- VUID-vkCmdDispatchIndirect-commandBuffer-parameter
  commandBuffer **must** be a valid VkCommandBuffer handle
- VUID-vkCmdDispatchIndirect-buffer-parameter
  buffer **must** be a valid VkBuffer handle
- VUID-vkCmdDispatchIndirect-commandBuffer-recording
  commandBuffer **must** be in the recording state
- VUID-vkCmdDispatchIndirect-commandBuffer-cmdpool
  The VkCommandPool that commandBuffer was allocated from **must** support compute operations
- VUID-vkCmdDispatchIndirect-renderpass
  This command **must** only be called outside of a render pass instance
- VUID-vkCmdDispatchIndirect-videocoding
  This command **must** only be called outside of a video coding scope
- VUID-vkCmdDispatchIndirect-commonparent
  Both of buffer, and commandBuffer **must** have been created, allocated, or retrieved from the same VkDevice

## Host Synchronization

- Host access to commandBuffer **must** be externally synchronized
- Host access to the VkCommandPool that commandBuffer was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Outside | Outside | Compute | Action |

The VkDispatchIndirectCommand structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkDispatchIndirectCommand {
    uint32_t    x;
    uint32_t    y;
    uint32_t    z;
} VkDispatchIndirectCommand;
```

- `x` is the number of local workgroups to dispatch in the X dimension.
- `y` is the number of local workgroups to dispatch in the Y dimension.
- `z` is the number of local workgroups to dispatch in the Z dimension.

The members of `VkDispatchIndirectCommand` have the same meaning as the corresponding parameters of vkCmdDispatch.

### Valid Usage

- VUID-VkDispatchIndirectCommand-x-00417
  `x` **must** be less than or equal to `VkPhysicalDeviceLimits::maxComputeWorkGroupCount[0]`
- VUID-VkDispatchIndirectCommand-y-00418
  `y` **must** be less than or equal to `VkPhysicalDeviceLimits::maxComputeWorkGroupCount[1]`
- VUID-VkDispatchIndirectCommand-z-00419
  `z` **must** be less than or equal to `VkPhysicalDeviceLimits::maxComputeWorkGroupCount[2]`

To record a dispatch using non-zero base values for the components of `WorkgroupId`, call:

```
// Provided by VK_VERSION_1_1
void vkCmdDispatchBase(
    VkCommandBuffer                             commandBuffer,
    uint32_t                                    baseGroupX,
    uint32_t                                    baseGroupY,
    uint32_t                                    baseGroupZ,
    uint32_t                                    groupCountX,
    uint32_t                                    groupCountY,
    uint32_t                                    groupCountZ);
```

or the equivalent command

```
// Provided by VK_KHR_device_group
void vkCmdDispatchBaseKHR(
    VkCommandBuffer                             commandBuffer,
    uint32_t                                    baseGroupX,
    uint32_t                                    baseGroupY,
    uint32_t                                    baseGroupZ,
    uint32_t                                    groupCountX,
    uint32_t                                    groupCountY,
    uint32_t                                    groupCountZ);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `baseGroupX` is the start value for the X component of `WorkgroupId`.
- `baseGroupY` is the start value for the Y component of `WorkgroupId`.
- `baseGroupZ` is the start value for the Z component of `WorkgroupId`.
- `groupCountX` is the number of local workgroups to dispatch in the X dimension.
- `groupCountY` is the number of local workgroups to dispatch in the Y dimension.
- `groupCountZ` is the number of local workgroups to dispatch in the Z dimension.

When the command is executed, a global workgroup consisting of `groupCountX` × `groupCountY` × `groupCountZ` local workgroups is assembled, with `WorkgroupId` values ranging from [`baseGroup*`, `baseGroup*` + `groupCount*`) in each component. vkCmdDispatch is equivalent to `vkCmdDispatchBase(0,0,0,groupCountX,groupCountY,groupCountZ)`.

### Valid Usage

- VUID-vkCmdDispatchBase-magFilter-04553
  If a VkSampler created with `magFilter` or `minFilter` equal to `VK_FILTER_LINEAR`, `reductionMode` equal to `VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE`, and `compareEnable` equal to `VK_FALSE` is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT`
- VUID-vkCmdDispatchBase-magFilter-09598
  If a VkSampler created with `magFilter` or `minFilter` equal to `VK_FILTER_LINEAR` and `reductionMode` equal to either `VK_SAMPLER_REDUCTION_MODE_MIN` or `VK_SAMPLER_REDUCTION_MODE_MAX` is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT`
- VUID-vkCmdDispatchBase-mipmapMode-04770
  If a VkSampler created with `mipmapMode` equal to `VK_SAMPLER_MIPMAP_MODE_LINEAR`, `reductionMode` equal to `VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE`, and `compareEnable` equal to `VK_FALSE` is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT`

- VUID-vkCmdDispatchBase-mipmapMode-09599
  If a VkSampler created with mipmapMode equal to VK_SAMPLER_MIPMAP_MODE_LINEAR and reductionMode equal to either VK_SAMPLER_REDUCTION_MODE_MIN or VK_SAMPLER_REDUCTION_MODE_MAX is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT
- VUID-vkCmdDispatchBase-unnormalizedCoordinates-09635
  If a VkSampler created with unnormalizedCoordinates equal to VK_TRUE is used to sample a VkImageView as a result of this command, then the image view's levelCount and layerCount **must** be 1
- VUID-vkCmdDispatchBase-unnormalizedCoordinates-09636
  If a VkSampler created with unnormalizedCoordinates equal to VK_TRUE is used to sample a VkImageView as a result of this command, then the image view's viewType **must** be VK_IMAGE_VIEW_TYPE_1D or VK_IMAGE_VIEW_TYPE_2D
- VUID-vkCmdDispatchBase-None-06479
  If a VkImageView is sampled with depth comparison, the image view's format features **must** contain VK_FORMAT_FEATURE_2_SAMPLED_IMAGE_DEPTH_COMPARISON_BIT
- VUID-vkCmdDispatchBase-None-02691
  If a VkImageView is accessed using atomic operations as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_STORAGE_IMAGE_ATOMIC_BIT
- VUID-vkCmdDispatchBase-None-07888
  If a VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER descriptor is accessed using atomic operations as a result of this command, then the storage texel buffer's format features **must** contain VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_ATOMIC_BIT
- VUID-vkCmdDispatchBase-None-02692
  If a VkImageView is sampled with VK_FILTER_CUBIC_EXT as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_CUBIC_BIT_EXT
- VUID-vkCmdDispatchBase-None-02693
  If the VK_EXT_filter_cubic extension is not enabled and any VkImageView is sampled with VK_FILTER_CUBIC_EXT as a result of this command, it **must** not have a VkImageViewType of VK_IMAGE_VIEW_TYPE_3D, VK_IMAGE_VIEW_TYPE_CUBE, or VK_IMAGE_VIEW_TYPE_CUBE_ARRAY
- VUID-vkCmdDispatchBase-filterCubic-02694
  Any VkImageView being sampled with VK_FILTER_CUBIC_EXT as a result of this command **must** have a VkImageViewType and format that supports cubic filtering, as specified by VkFilterCubicImageViewImageFormatPropertiesEXT::filterCubic returned by vkGetPhysicalDeviceImageFormatProperties2
- VUID-vkCmdDispatchBase-filterCubicMinmax-02695
  Any VkImageView being sampled with VK_FILTER_CUBIC_EXT with a reduction mode of either VK_SAMPLER_REDUCTION_MODE_MIN or VK_SAMPLER_REDUCTION_MODE_MAX as a result of this command **must** have a VkImageViewType and format that supports cubic filtering together with minmax filtering, as specified by VkFilterCubicImageViewImageFormatPropertiesEXT::filterCubicMinmax returned by

vkGetPhysicalDeviceImageFormatProperties2

- VUID-vkCmdDispatchBase-cubicRangeClamp-09212
  If the `cubicRangeClamp` feature is not enabled, then any VkImageView being sampled with `VK_FILTER_CUBIC_EXT` as a result of this command **must** not have a VkSamplerReductionModeCreateInfo::`reductionMode` equal to `VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE_RANGECLAMP_QCOM`
- VUID-vkCmdDispatchBase-reductionMode-09213
  Any VkImageView being sampled with a VkSamplerReductionModeCreateInfo::`reductionMode` equal to `VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE_RANGECLAMP_QCOM` as a result of this command **must** sample with `VK_FILTER_CUBIC_EXT`
- VUID-vkCmdDispatchBase-selectableCubicWeights-09214
  If the `selectableCubicWeights` feature is not enabled, then any VkImageView being sampled with `VK_FILTER_CUBIC_EXT` as a result of this command **must** have VkSamplerCubicWeightsCreateInfoQCOM::`cubicWeights` equal to `VK_CUBIC_FILTER_WEIGHTS_CATMULL_ROM_QCOM`
- VUID-vkCmdDispatchBase-flags-02696
  Any VkImage created with a VkImageCreateInfo::`flags` containing `VK_IMAGE_CREATE_CORNER_SAMPLED_BIT_NV` sampled as a result of this command **must** only be sampled using a VkSamplerAddressMode of `VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_EDGE`
- VUID-vkCmdDispatchBase-OpTypeImage-07027
  For any VkImageView being written as a storage image where the image format field of the `OpTypeImage` is `Unknown`, the view's format features **must** contain `VK_FORMAT_FEATURE_2_STORAGE_WRITE_WITHOUT_FORMAT_BIT`
- VUID-vkCmdDispatchBase-OpTypeImage-07028
  For any VkImageView being read as a storage image where the image format field of the `OpTypeImage` is `Unknown`, the view's format features **must** contain `VK_FORMAT_FEATURE_2_STORAGE_READ_WITHOUT_FORMAT_BIT`
- VUID-vkCmdDispatchBase-OpTypeImage-07029
  For any VkBufferView being written as a storage texel buffer where the image format field of the `OpTypeImage` is `Unknown`, the view's buffer features **must** contain `VK_FORMAT_FEATURE_2_STORAGE_WRITE_WITHOUT_FORMAT_BIT`
- VUID-vkCmdDispatchBase-OpTypeImage-07030
  Any VkBufferView being read as a storage texel buffer where the image format field of the `OpTypeImage` is `Unknown` then the view's buffer features **must** contain `VK_FORMAT_FEATURE_2_STORAGE_READ_WITHOUT_FORMAT_BIT`
- VUID-vkCmdDispatchBase-None-08600
  For each set *n* that is statically used by a bound shader, a descriptor set **must** have been bound to *n* at the same pipeline bind point, with a VkPipelineLayout that is compatible for set *n*, with the VkPipelineLayout used to create the current VkPipeline or the VkDescriptorSetLayout array used to create the current VkShaderEXT , as described in Pipeline Layout Compatibility
- VUID-vkCmdDispatchBase-None-08601
  For each push constant that is statically used by a bound shader, a push constant value **must** have been set for the same pipeline bind point, with a VkPipelineLayout that is

compatible for push constants, with the VkPipelineLayout used to create the current VkPipeline or the VkDescriptorSetLayout array used to create the current VkShaderEXT , as described in Pipeline Layout Compatibility

- VUID-vkCmdDispatchBase-None-10068
  For each array of resources that is used by a bound shader, the indices used to access members of the array **must** be less than the descriptor count for the identified binding in the descriptor sets used by this command
- VUID-vkCmdDispatchBase-maintenance4-08602
  If the `maintenance4` feature is not enabled, then for each push constant that is statically used by a bound shader, a push constant value **must** have been set for the same pipeline bind point, with a VkPipelineLayout that is compatible for push constants, with the VkPipelineLayout used to create the current VkPipeline or the VkDescriptorSetLayout and VkPushConstantRange arrays used to create the current VkShaderEXT , as described in Pipeline Layout Compatibility
- VUID-vkCmdDispatchBase-None-08114
  Descriptors in each bound descriptor set, specified via vkCmdBindDescriptorSets, **must** be valid as described by descriptor validity if they are statically used by the VkPipeline bound to the pipeline bind point used by this command and the bound VkPipeline was not created with `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`
- VUID-vkCmdDispatchBase-None-08115
  If the descriptors used by the VkPipeline bound to the pipeline bind point were specified via vkCmdBindDescriptorSets, the bound VkPipeline **must** have been created without `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`
- VUID-vkCmdDispatchBase-None-08116
  Descriptors in bound descriptor buffers, specified via vkCmdSetDescriptorBufferOffsetsEXT, **must** be valid if they are dynamically used by the VkPipeline bound to the pipeline bind point used by this command and the bound VkPipeline was created with `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`
- VUID-vkCmdDispatchBase-None-08604
  Descriptors in bound descriptor buffers, specified via vkCmdSetDescriptorBufferOffsetsEXT, **must** be valid if they are dynamically used by any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command
- VUID-vkCmdDispatchBase-None-08117
  If the descriptors used by the VkPipeline bound to the pipeline bind point were specified via vkCmdSetDescriptorBufferOffsetsEXT, the bound VkPipeline **must** have been created with `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`
- VUID-vkCmdDispatchBase-None-08119
  If a descriptor is dynamically used with a VkPipeline created with `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`, the descriptor memory **must** be resident
- VUID-vkCmdDispatchBase-None-08605
  If a descriptor is dynamically used with a VkShaderEXT created with a `VkDescriptorSetLayout` that was created with `VK_DESCRIPTOR_SET_LAYOUT_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`, the descriptor memory **must**

be resident

- VUID-vkCmdDispatchBase-None-08606
  If the `shaderObject` feature is not enabled, a valid pipeline **must** be bound to the pipeline bind point used by this command
- VUID-vkCmdDispatchBase-None-08608
  If a pipeline is bound to the pipeline bind point used by this command, there **must** not have been any calls to dynamic state setting commands for any state specified statically in the VkPipeline object bound to the pipeline bind point used by this command, since that pipeline was bound
- VUID-vkCmdDispatchBase-None-08609
  If the VkPipeline object bound to the pipeline bind point used by this command or any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a VkSampler object that uses unnormalized coordinates, that sampler **must** not be used to sample from any VkImage with a VkImageView of the type `VK_IMAGE_VIEW_TYPE_3D`, `VK_IMAGE_VIEW_TYPE_CUBE`, `VK_IMAGE_VIEW_TYPE_1D_ARRAY`, `VK_IMAGE_VIEW_TYPE_2D_ARRAY` or `VK_IMAGE_VIEW_TYPE_CUBE_ARRAY`, in any shader stage
- VUID-vkCmdDispatchBase-None-08610
  If the VkPipeline object bound to the pipeline bind point used by this command or any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a VkSampler object that uses unnormalized coordinates, that sampler **must** not be used with any of the SPIR-V `OpImageSample*` or `OpImageSparseSample*` instructions with `ImplicitLod`, `Dref` or `Proj` in their name, in any shader stage
- VUID-vkCmdDispatchBase-None-08611
  If the VkPipeline object bound to the pipeline bind point used by this command or any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a VkSampler object that uses unnormalized coordinates, that sampler **must** not be used with any of the SPIR-V `OpImageSample*` or `OpImageSparseSample*` instructions that includes a LOD bias or any offset values, in any shader stage
- VUID-vkCmdDispatchBase-None-08607
  If the `shaderObject` is enabled, either a valid pipeline **must** be bound to the pipeline bind point used by this command, or a valid combination of valid and VK_NULL_HANDLE shader objects **must** be bound to every supported shader stage corresponding to the pipeline bind point used by this command
- VUID-vkCmdDispatchBase-uniformBuffers-06935
  If any stage of the VkPipeline object bound to the pipeline bind point used by this command accesses a uniform buffer, and that stage was created without enabling either `VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_EXT` or `VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_2_EXT` for `uniformBuffers`, and the `robustBufferAccess` feature is not enabled, that stage **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point
- VUID-vkCmdDispatchBase-None-08612
  If the `robustBufferAccess` feature is not enabled, and any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a uniform buffer, it **must** not access values outside of the range of the buffer as specified in the

descriptor set bound to the same pipeline bind point

- VUID-vkCmdDispatchBase-storageBuffers-06936
  If any stage of the VkPipeline object bound to the pipeline bind point used by this command accesses a storage buffer, and that stage was created without enabling either VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_EXT or VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_2_EXT for storageBuffers, and the robustBufferAccess feature is not enabled, that stage **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point
- VUID-vkCmdDispatchBase-None-08613
  If the robustBufferAccess feature is not enabled, and any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a storage buffer, it **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point
- VUID-vkCmdDispatchBase-commandBuffer-02707
  If commandBuffer is an unprotected command buffer and protectedNoFault is not supported, any resource accessed by bound shaders **must** not be a protected resource
- VUID-vkCmdDispatchBase-None-06550
  If a bound shader accesses a VkSampler or VkImageView object that enables sampler $Y'C_BC_R$ conversion, that object **must** only be used with OpImageSample* or OpImageSparseSample* instructions
- VUID-vkCmdDispatchBase-ConstOffset-06551
  If a bound shader accesses a VkSampler or VkImageView object that enables sampler $Y'C_BC_R$ conversion, that object **must** not use the ConstOffset and Offset operands
- VUID-vkCmdDispatchBase-viewType-07752
  If a VkImageView is accessed as a result of this command, then the image view's viewType **must** match the Dim operand of the OpTypeImage as described in Compatibility Between SPIR-V Image Dimensions and Vulkan ImageView Types
- VUID-vkCmdDispatchBase-format-07753
  If a VkImageView is accessed as a result of this command, then the numeric type of the image view's format and the Sampled Type operand of the OpTypeImage **must** match
- VUID-vkCmdDispatchBase-OpImageWrite-08795
  If a VkImageView created with a format other than VK_FORMAT_A8_UNORM_KHR is accessed using OpImageWrite as a result of this command, then the Type of the Texel operand of that instruction **must** have at least as many components as the image view's format
- VUID-vkCmdDispatchBase-OpImageWrite-08796
  If a VkImageView created with the format VK_FORMAT_A8_UNORM_KHR is accessed using OpImageWrite as a result of this command, then the Type of the Texel operand of that instruction **must** have four components
- VUID-vkCmdDispatchBase-OpImageWrite-04469
  If a VkBufferView is accessed using OpImageWrite as a result of this command, then the Type of the Texel operand of that instruction **must** have at least as many components as the buffer view's format

- VUID-vkCmdDispatchBase-SampledType-04470
  If a VkImageView with a VkFormat that has a 64-bit component width is accessed as a result of this command, the `SampledType` of the `OpTypeImage` operand of that instruction **must** have a `Width` of 64
- VUID-vkCmdDispatchBase-SampledType-04471
  If a VkImageView with a VkFormat that has a component width less than 64-bit is accessed as a result of this command, the `SampledType` of the `OpTypeImage` operand of that instruction **must** have a `Width` of 32
- VUID-vkCmdDispatchBase-SampledType-04472
  If a VkBufferView with a VkFormat that has a 64-bit component width is accessed as a result of this command, the `SampledType` of the `OpTypeImage` operand of that instruction **must** have a `Width` of 64
- VUID-vkCmdDispatchBase-SampledType-04473
  If a VkBufferView with a VkFormat that has a component width less than 64-bit is accessed as a result of this command, the `SampledType` of the `OpTypeImage` operand of that instruction **must** have a `Width` of 32
- VUID-vkCmdDispatchBase-sparseImageInt64Atomics-04474
  If the `sparseImageInt64Atomics` feature is not enabled, VkImage objects created with the `VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT` flag **must** not be accessed by atomic instructions through an `OpTypeImage` with a `SampledType` with a `Width` of 64 by this command
- VUID-vkCmdDispatchBase-sparseImageInt64Atomics-04475
  If the `sparseImageInt64Atomics` feature is not enabled, VkBuffer objects created with the `VK_BUFFER_CREATE_SPARSE_RESIDENCY_BIT` flag **must** not be accessed by atomic instructions through an `OpTypeImage` with a `SampledType` with a `Width` of 64 by this command
- VUID-vkCmdDispatchBase-OpImageWeightedSampleQCOM-06971
  If `OpImageWeightedSampleQCOM` is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_2_WEIGHT_SAMPLED_IMAGE_BIT_QCOM`
- VUID-vkCmdDispatchBase-OpImageWeightedSampleQCOM-06972
  If `OpImageWeightedSampleQCOM` uses a VkImageView as a sample weight image as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_2_WEIGHT_IMAGE_BIT_QCOM`
- VUID-vkCmdDispatchBase-OpImageBoxFilterQCOM-06973
  If `OpImageBoxFilterQCOM` is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_2_BOX_FILTER_SAMPLED_BIT_QCOM`
- VUID-vkCmdDispatchBase-OpImageBlockMatchSSDQCOM-06974
  If `OpImageBlockMatchSSDQCOM` is used to read from an VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM`
- VUID-vkCmdDispatchBase-OpImageBlockMatchSADQCOM-06975
  If `OpImageBlockMatchSADQCOM` is used to read from an VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM`

- VUID-vkCmdDispatchBase-OpImageBlockMatchSADQCOM-06976
  If `OpImageBlockMatchSADQCOM` or OpImageBlockMatchSSDQCOM is used to read from a reference image as result of this command, then the specified reference coordinates **must** not fail integer texel coordinate validation
- VUID-vkCmdDispatchBase-OpImageWeightedSampleQCOM-06977
  If `OpImageWeightedSampleQCOM`, `OpImageBoxFilterQCOM`, `OpImageBlockMatchWindowSSDQCOM`, `OpImageBlockMatchWindowSADQCOM`, `OpImageBlockMatchGatherSSDQCOM`, `OpImageBlockMatchGatherSADQCOM`, `OpImageBlockMatchSSDQCOM`, or `OpImageBlockMatchSADQCOM` uses a VkSampler as a result of this command, then the sampler **must** have been created with `VK_SAMPLER_CREATE_IMAGE_PROCESSING_BIT_QCOM`
- VUID-vkCmdDispatchBase-OpImageWeightedSampleQCOM-06978
  If any command other than `OpImageWeightedSampleQCOM`, `OpImageBoxFilterQCOM`, `OpImageBlockMatchWindowSSDQCOM`, `OpImageBlockMatchWindowSADQCOM`, `OpImageBlockMatchGatherSSDQCOM`, `OpImageBlockMatchGatherSADQCOM`, `OpImageBlockMatchSSDQCOM`, or `OpImageBlockMatchSADQCOM` uses a VkSampler as a result of this command, then the sampler **must** not have been created with `VK_SAMPLER_CREATE_IMAGE_PROCESSING_BIT_QCOM`
- VUID-vkCmdDispatchBase-OpImageBlockMatchWindow-09215
  If a `OpImageBlockMatchWindow*QCOM` or `OpImageBlockMatchGather*QCOM` instruction is used to read from an VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM`
- VUID-vkCmdDispatchBase-OpImageBlockMatchWindow-09216
  If a `OpImageBlockMatchWindow*QCOM` or `OpImageBlockMatchGather*QCOM` instruction is used to read from an VkImageView as a result of this command, then the image view's format **must** be a single-component format
- VUID-vkCmdDispatchBase-OpImageBlockMatchWindow-09217
  If a `OpImageBlockMatchWindow*QCOM` or `OpImageBlockMatchGather*QCOM` read from a reference image as result of this command, then the specified reference coordinates **must** not fail integer texel coordinate validation
- VUID-vkCmdDispatchBase-None-07288
  Any shader invocation executed by this command **must** terminate
- VUID-vkCmdDispatchBase-None-09600
  If a descriptor with type equal to any of `VK_DESCRIPTOR_TYPE_SAMPLE_WEIGHT_IMAGE_QCOM`, `VK_DESCRIPTOR_TYPE_BLOCK_MATCH_IMAGE_QCOM`, `VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE`, `VK_DESCRIPTOR_TYPE_STORAGE_IMAGE`, or `VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT` is accessed as a result of this command, the image subresource identified by that descriptor **must** be in the image layout identified when the descriptor was written
- VUID-vkCmdDispatchBase-commandBuffer-02712
  If `commandBuffer` is a protected command buffer and `protectedNoFault` is not supported, any resource written to by the `VkPipeline` object bound to the pipeline bind point used by this command **must** not be an unprotected resource
- VUID-vkCmdDispatchBase-commandBuffer-02713
  If `commandBuffer` is a protected command buffer and `protectedNoFault` is not supported, pipeline stages other than the framebuffer-space and compute stages in the `VkPipeline`

object bound to the pipeline bind point used by this command **must** not write to any resource

- VUID-vkCmdDispatchBase-commandBuffer-04617
  If any of the shader stages of the `VkPipeline` bound to the pipeline bind point used by this command uses the `RayQueryKHR` capability, then `commandBuffer` **must** not be a protected command buffer
- VUID-vkCmdDispatchBase-baseGroupX-00421
  `baseGroupX` **must** be less than `VkPhysicalDeviceLimits`::`maxComputeWorkGroupCount`[0]
- VUID-vkCmdDispatchBase-baseGroupX-00422
  `baseGroupY` **must** be less than `VkPhysicalDeviceLimits`::`maxComputeWorkGroupCount`[1]
- VUID-vkCmdDispatchBase-baseGroupZ-00423
  `baseGroupZ` **must** be less than `VkPhysicalDeviceLimits`::`maxComputeWorkGroupCount`[2]
- VUID-vkCmdDispatchBase-groupCountX-00424
  `groupCountX` **must** be less than or equal to `VkPhysicalDeviceLimits`::`maxComputeWorkGroupCount`[0] minus `baseGroupX`
- VUID-vkCmdDispatchBase-groupCountY-00425
  `groupCountY` **must** be less than or equal to `VkPhysicalDeviceLimits`::`maxComputeWorkGroupCount`[1] minus `baseGroupY`
- VUID-vkCmdDispatchBase-groupCountZ-00426
  `groupCountZ` **must** be less than or equal to `VkPhysicalDeviceLimits`::`maxComputeWorkGroupCount`[2] minus `baseGroupZ`
- VUID-vkCmdDispatchBase-baseGroupX-00427
  If any of `baseGroupX`, `baseGroupY`, or `baseGroupZ` are not zero, then the bound compute pipeline **must** have been created with the `VK_PIPELINE_CREATE_DISPATCH_BASE` flag or the bound compute shader object **must** have been created with the `VK_SHADER_CREATE_DISPATCH_BASE_BIT_EXT` flag

## Valid Usage (Implicit)

- VUID-vkCmdDispatchBase-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdDispatchBase-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdDispatchBase-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support compute operations
- VUID-vkCmdDispatchBase-renderpass
  This command **must** only be called outside of a render pass instance
- VUID-vkCmdDispatchBase-videocoding
  This command **must** only be called outside of a video coding scope

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Outside | Outside | Compute | Action |

A subpass shading dispatches a compute pipeline work with the work dimension of render area of the calling subpass and work groups are partitioned by specified work group size. Subpass operations like `subpassLoad` are allowed to be used.

To record a subpass shading, call:

```
// Provided by VK_HUAWEI_subpass_shading
void vkCmdSubpassShadingHUAWEI(
    VkCommandBuffer                             commandBuffer);
```

- `commandBuffer` is the command buffer into which the command will be recorded.

When the command is executed, a global workgroup consisting of ceil (render area size / local workgroup size) local workgroups is assembled.

## Valid Usage

- VUID-vkCmdSubpassShadingHUAWEI-magFilter-04553
  If a VkSampler created with `magFilter` or `minFilter` equal to `VK_FILTER_LINEAR`, `reductionMode` equal to `VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE`, and `compareEnable` equal to `VK_FALSE` is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT`
- VUID-vkCmdSubpassShadingHUAWEI-magFilter-09598
  If a VkSampler created with `magFilter` or `minFilter` equal to `VK_FILTER_LINEAR` and `reductionMode` equal to either `VK_SAMPLER_REDUCTION_MODE_MIN` or `VK_SAMPLER_REDUCTION_MODE_MAX` is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT`
- VUID-vkCmdSubpassShadingHUAWEI-mipmapMode-04770

If a VkSampler created with mipmapMode equal to VK_SAMPLER_MIPMAP_MODE_LINEAR, reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE, and compareEnable equal to VK_FALSE is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT

- VUID-vkCmdSubpassShadingHUAWEI-mipmapMode-09599
  If a VkSampler created with mipmapMode equal to VK_SAMPLER_MIPMAP_MODE_LINEAR and reductionMode equal to either VK_SAMPLER_REDUCTION_MODE_MIN or VK_SAMPLER_REDUCTION_MODE_MAX is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT
- VUID-vkCmdSubpassShadingHUAWEI-unnormalizedCoordinates-09635
  If a VkSampler created with unnormalizedCoordinates equal to VK_TRUE is used to sample a VkImageView as a result of this command, then the image view's levelCount and layerCount **must** be 1
- VUID-vkCmdSubpassShadingHUAWEI-unnormalizedCoordinates-09636
  If a VkSampler created with unnormalizedCoordinates equal to VK_TRUE is used to sample a VkImageView as a result of this command, then the image view's viewType **must** be VK_IMAGE_VIEW_TYPE_1D or VK_IMAGE_VIEW_TYPE_2D
- VUID-vkCmdSubpassShadingHUAWEI-None-06479
  If a VkImageView is sampled with depth comparison, the image view's format features **must** contain VK_FORMAT_FEATURE_2_SAMPLED_IMAGE_DEPTH_COMPARISON_BIT
- VUID-vkCmdSubpassShadingHUAWEI-None-02691
  If a VkImageView is accessed using atomic operations as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_STORAGE_IMAGE_ATOMIC_BIT
- VUID-vkCmdSubpassShadingHUAWEI-None-07888
  If a VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER descriptor is accessed using atomic operations as a result of this command, then the storage texel buffer's format features **must** contain VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_ATOMIC_BIT
- VUID-vkCmdSubpassShadingHUAWEI-None-02692
  If a VkImageView is sampled with VK_FILTER_CUBIC_EXT as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_CUBIC_BIT_EXT
- VUID-vkCmdSubpassShadingHUAWEI-None-02693
  If the VK_EXT_filter_cubic extension is not enabled and any VkImageView is sampled with VK_FILTER_CUBIC_EXT as a result of this command, it **must** not have a VkImageViewType of VK_IMAGE_VIEW_TYPE_3D, VK_IMAGE_VIEW_TYPE_CUBE, or VK_IMAGE_VIEW_TYPE_CUBE_ARRAY
- VUID-vkCmdSubpassShadingHUAWEI-filterCubic-02694
  Any VkImageView being sampled with VK_FILTER_CUBIC_EXT as a result of this command **must** have a VkImageViewType and format that supports cubic filtering, as specified by VkFilterCubicImageViewImageFormatPropertiesEXT::filterCubic returned by vkGetPhysicalDeviceImageFormatProperties2
- VUID-vkCmdSubpassShadingHUAWEI-filterCubicMinmax-02695

Any VkImageView being sampled with VK_FILTER_CUBIC_EXT with a reduction mode of either VK_SAMPLER_REDUCTION_MODE_MIN or VK_SAMPLER_REDUCTION_MODE_MAX as a result of this command **must** have a VkImageViewType and format that supports cubic filtering together with minmax filtering, as specified by VkFilterCubicImageViewImageFormatPropertiesEXT::filterCubicMinmax returned by vkGetPhysicalDeviceImageFormatProperties2

- VUID-vkCmdSubpassShadingHUAWEI-cubicRangeClamp-09212
  If the cubicRangeClamp feature is not enabled, then any VkImageView being sampled with VK_FILTER_CUBIC_EXT as a result of this command **must** not have a VkSamplerReductionModeCreateInfo::reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE_RANGECLAMP_QCOM
- VUID-vkCmdSubpassShadingHUAWEI-reductionMode-09213
  Any VkImageView being sampled with a VkSamplerReductionModeCreateInfo::reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE_RANGECLAMP_QCOM as a result of this command **must** sample with VK_FILTER_CUBIC_EXT
- VUID-vkCmdSubpassShadingHUAWEI-selectableCubicWeights-09214
  If the selectableCubicWeights feature is not enabled, then any VkImageView being sampled with VK_FILTER_CUBIC_EXT as a result of this command **must** have VkSamplerCubicWeightsCreateInfoQCOM::cubicWeights equal to VK_CUBIC_FILTER_WEIGHTS_CATMULL_ROM_QCOM
- VUID-vkCmdSubpassShadingHUAWEI-flags-02696
  Any VkImage created with a VkImageCreateInfo::flags containing VK_IMAGE_CREATE_CORNER_SAMPLED_BIT_NV sampled as a result of this command **must** only be sampled using a VkSamplerAddressMode of VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_EDGE
- VUID-vkCmdSubpassShadingHUAWEI-OpTypeImage-07027
  For any VkImageView being written as a storage image where the image format field of the OpTypeImage is Unknown, the view's format features **must** contain VK_FORMAT_FEATURE_2_STORAGE_WRITE_WITHOUT_FORMAT_BIT
- VUID-vkCmdSubpassShadingHUAWEI-OpTypeImage-07028
  For any VkImageView being read as a storage image where the image format field of the OpTypeImage is Unknown, the view's format features **must** contain VK_FORMAT_FEATURE_2_STORAGE_READ_WITHOUT_FORMAT_BIT
- VUID-vkCmdSubpassShadingHUAWEI-OpTypeImage-07029
  For any VkBufferView being written as a storage texel buffer where the image format field of the OpTypeImage is Unknown, the view's buffer features **must** contain VK_FORMAT_FEATURE_2_STORAGE_WRITE_WITHOUT_FORMAT_BIT
- VUID-vkCmdSubpassShadingHUAWEI-OpTypeImage-07030
  Any VkBufferView being read as a storage texel buffer where the image format field of the OpTypeImage is Unknown then the view's buffer features **must** contain VK_FORMAT_FEATURE_2_STORAGE_READ_WITHOUT_FORMAT_BIT
- VUID-vkCmdSubpassShadingHUAWEI-None-08600
  For each set *n* that is statically used by a bound shader, a descriptor set **must** have been bound to *n* at the same pipeline bind point, with a VkPipelineLayout that is compatible for set *n*, with the VkPipelineLayout used to create the current VkPipeline or the

VkDescriptorSetLayout array used to create the current VkShaderEXT , as described in Pipeline Layout Compatibility

- VUID-vkCmdSubpassShadingHUAWEI-None-08601
  For each push constant that is statically used by a bound shader, a push constant value **must** have been set for the same pipeline bind point, with a VkPipelineLayout that is compatible for push constants, with the VkPipelineLayout used to create the current VkPipeline or the VkDescriptorSetLayout array used to create the current VkShaderEXT , as described in Pipeline Layout Compatibility
- VUID-vkCmdSubpassShadingHUAWEI-None-10068
  For each array of resources that is used by a bound shader, the indices used to access members of the array **must** be less than the descriptor count for the identified binding in the descriptor sets used by this command
- VUID-vkCmdSubpassShadingHUAWEI-maintenance4-08602
  If the `maintenance4` feature is not enabled, then for each push constant that is statically used by a bound shader, a push constant value **must** have been set for the same pipeline bind point, with a VkPipelineLayout that is compatible for push constants, with the VkPipelineLayout used to create the current VkPipeline or the VkDescriptorSetLayout and VkPushConstantRange arrays used to create the current VkShaderEXT , as described in Pipeline Layout Compatibility
- VUID-vkCmdSubpassShadingHUAWEI-None-08114
  Descriptors in each bound descriptor set, specified via vkCmdBindDescriptorSets, **must** be valid as described by descriptor validity if they are statically used by the VkPipeline bound to the pipeline bind point used by this command and the bound VkPipeline was not created with `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`
- VUID-vkCmdSubpassShadingHUAWEI-None-08115
  If the descriptors used by the VkPipeline bound to the pipeline bind point were specified via vkCmdBindDescriptorSets, the bound VkPipeline **must** have been created without `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`
- VUID-vkCmdSubpassShadingHUAWEI-None-08116
  Descriptors in bound descriptor buffers, specified via vkCmdSetDescriptorBufferOffsetsEXT, **must** be valid if they are dynamically used by the VkPipeline bound to the pipeline bind point used by this command and the bound VkPipeline was created with `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`
- VUID-vkCmdSubpassShadingHUAWEI-None-08604
  Descriptors in bound descriptor buffers, specified via vkCmdSetDescriptorBufferOffsetsEXT, **must** be valid if they are dynamically used by any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command
- VUID-vkCmdSubpassShadingHUAWEI-None-08117
  If the descriptors used by the VkPipeline bound to the pipeline bind point were specified via vkCmdSetDescriptorBufferOffsetsEXT, the bound VkPipeline **must** have been created with `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`
- VUID-vkCmdSubpassShadingHUAWEI-None-08119
  If a descriptor is dynamically used with a VkPipeline created with

VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT, the descriptor memory **must** be resident

- VUID-vkCmdSubpassShadingHUAWEI-None-08605
  If a descriptor is dynamically used with a VkShaderEXT created with a VkDescriptorSetLayout that was created with VK_DESCRIPTOR_SET_LAYOUT_CREATE_DESCRIPTOR_BUFFER_BIT_EXT, the descriptor memory **must** be resident
- VUID-vkCmdSubpassShadingHUAWEI-None-08606
  If the shaderObject feature is not enabled, a valid pipeline **must** be bound to the pipeline bind point used by this command
- VUID-vkCmdSubpassShadingHUAWEI-None-08608
  If a pipeline is bound to the pipeline bind point used by this command, there **must** not have been any calls to dynamic state setting commands for any state specified statically in the VkPipeline object bound to the pipeline bind point used by this command, since that pipeline was bound
- VUID-vkCmdSubpassShadingHUAWEI-None-08609
  If the VkPipeline object bound to the pipeline bind point used by this command or any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a VkSampler object that uses unnormalized coordinates, that sampler **must** not be used to sample from any VkImage with a VkImageView of the type VK_IMAGE_VIEW_TYPE_3D, VK_IMAGE_VIEW_TYPE_CUBE, VK_IMAGE_VIEW_TYPE_1D_ARRAY, VK_IMAGE_VIEW_TYPE_2D_ARRAY or VK_IMAGE_VIEW_TYPE_CUBE_ARRAY, in any shader stage
- VUID-vkCmdSubpassShadingHUAWEI-None-08610
  If the VkPipeline object bound to the pipeline bind point used by this command or any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a VkSampler object that uses unnormalized coordinates, that sampler **must** not be used with any of the SPIR-V OpImageSample* or OpImageSparseSample* instructions with ImplicitLod, Dref or Proj in their name, in any shader stage
- VUID-vkCmdSubpassShadingHUAWEI-None-08611
  If the VkPipeline object bound to the pipeline bind point used by this command or any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a VkSampler object that uses unnormalized coordinates, that sampler **must** not be used with any of the SPIR-V OpImageSample* or OpImageSparseSample* instructions that includes a LOD bias or any offset values, in any shader stage
- VUID-vkCmdSubpassShadingHUAWEI-None-08607
  If the shaderObject is enabled, either a valid pipeline **must** be bound to the pipeline bind point used by this command, or a valid combination of valid and VK_NULL_HANDLE shader objects **must** be bound to every supported shader stage corresponding to the pipeline bind point used by this command
- VUID-vkCmdSubpassShadingHUAWEI-uniformBuffers-06935
  If any stage of the VkPipeline object bound to the pipeline bind point used by this command accesses a uniform buffer, and that stage was created without enabling either VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_EXT or VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_2_EXT for uniformBuffers, and the robustBufferAccess feature is not enabled, that stage **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same

pipeline bind point

- VUID-vkCmdSubpassShadingHUAWEI-None-08612
  If the `robustBufferAccess` feature is not enabled, and any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a uniform buffer, it **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point

- VUID-vkCmdSubpassShadingHUAWEI-storageBuffers-06936
  If any stage of the VkPipeline object bound to the pipeline bind point used by this command accesses a storage buffer, and that stage was created without enabling either `VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_EXT` or `VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_2_EXT` for `storageBuffers`, and the `robustBufferAccess` feature is not enabled, that stage **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point

- VUID-vkCmdSubpassShadingHUAWEI-None-08613
  If the `robustBufferAccess` feature is not enabled, and any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a storage buffer, it **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point

- VUID-vkCmdSubpassShadingHUAWEI-commandBuffer-02707
  If `commandBuffer` is an unprotected command buffer and `protectedNoFault` is not supported, any resource accessed by bound shaders **must** not be a protected resource

- VUID-vkCmdSubpassShadingHUAWEI-None-06550
  If a bound shader accesses a VkSampler or VkImageView object that enables sampler $Y'C_BC_R$ conversion, that object **must** only be used with `OpImageSample*` or `OpImageSparseSample*` instructions

- VUID-vkCmdSubpassShadingHUAWEI-ConstOffset-06551
  If a bound shader accesses a VkSampler or VkImageView object that enables sampler $Y'C_BC_R$ conversion, that object **must** not use the `ConstOffset` and `Offset` operands

- VUID-vkCmdSubpassShadingHUAWEI-viewType-07752
  If a VkImageView is accessed as a result of this command, then the image view's `viewType` **must** match the `Dim` operand of the `OpTypeImage` as described in Compatibility Between SPIR-V Image Dimensions and Vulkan ImageView Types

- VUID-vkCmdSubpassShadingHUAWEI-format-07753
  If a VkImageView is accessed as a result of this command, then the numeric type of the image view's `format` and the `Sampled Type` operand of the `OpTypeImage` **must** match

- VUID-vkCmdSubpassShadingHUAWEI-OpImageWrite-08795
  If a VkImageView created with a format other than `VK_FORMAT_A8_UNORM_KHR` is accessed using `OpImageWrite` as a result of this command, then the `Type` of the `Texel` operand of that instruction **must** have at least as many components as the image view's format

- VUID-vkCmdSubpassShadingHUAWEI-OpImageWrite-08796
  If a VkImageView created with the format `VK_FORMAT_A8_UNORM_KHR` is accessed using `OpImageWrite` as a result of this command, then the `Type` of the `Texel` operand of that instruction **must** have four components

- VUID-vkCmdSubpassShadingHUAWEI-OpImageWrite-04469
  If a VkBufferView is accessed using OpImageWrite as a result of this command, then the Type of the Texel operand of that instruction **must** have at least as many components as the buffer view's format
- VUID-vkCmdSubpassShadingHUAWEI-SampledType-04470
  If a VkImageView with a VkFormat that has a 64-bit component width is accessed as a result of this command, the SampledType of the OpTypeImage operand of that instruction **must** have a Width of 64
- VUID-vkCmdSubpassShadingHUAWEI-SampledType-04471
  If a VkImageView with a VkFormat that has a component width less than 64-bit is accessed as a result of this command, the SampledType of the OpTypeImage operand of that instruction **must** have a Width of 32
- VUID-vkCmdSubpassShadingHUAWEI-SampledType-04472
  If a VkBufferView with a VkFormat that has a 64-bit component width is accessed as a result of this command, the SampledType of the OpTypeImage operand of that instruction **must** have a Width of 64
- VUID-vkCmdSubpassShadingHUAWEI-SampledType-04473
  If a VkBufferView with a VkFormat that has a component width less than 64-bit is accessed as a result of this command, the SampledType of the OpTypeImage operand of that instruction **must** have a Width of 32
- VUID-vkCmdSubpassShadingHUAWEI-sparseImageInt64Atomics-04474
  If the sparseImageInt64Atomics feature is not enabled, VkImage objects created with the VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT flag **must** not be accessed by atomic instructions through an OpTypeImage with a SampledType with a Width of 64 by this command
- VUID-vkCmdSubpassShadingHUAWEI-sparseImageInt64Atomics-04475
  If the sparseImageInt64Atomics feature is not enabled, VkBuffer objects created with the VK_BUFFER_CREATE_SPARSE_RESIDENCY_BIT flag **must** not be accessed by atomic instructions through an OpTypeImage with a SampledType with a Width of 64 by this command
- VUID-vkCmdSubpassShadingHUAWEI-OpImageWeightedSampleQCOM-06971
  If OpImageWeightedSampleQCOM is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_WEIGHT_SAMPLED_IMAGE_BIT_QCOM
- VUID-vkCmdSubpassShadingHUAWEI-OpImageWeightedSampleQCOM-06972
  If OpImageWeightedSampleQCOM uses a VkImageView as a sample weight image as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_WEIGHT_IMAGE_BIT_QCOM
- VUID-vkCmdSubpassShadingHUAWEI-OpImageBoxFilterQCOM-06973
  If OpImageBoxFilterQCOM is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_BOX_FILTER_SAMPLED_BIT_QCOM
- VUID-vkCmdSubpassShadingHUAWEI-OpImageBlockMatchSSDQCOM-06974
  If OpImageBlockMatchSSDQCOM is used to read from an VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM

- VUID-vkCmdSubpassShadingHUAWEI-OpImageBlockMatchSADQCOM-06975
  If `OpImageBlockMatchSADQCOM` is used to read from an VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM`
- VUID-vkCmdSubpassShadingHUAWEI-OpImageBlockMatchSADQCOM-06976
  If `OpImageBlockMatchSADQCOM` or OpImageBlockMatchSSDQCOM is used to read from a reference image as result of this command, then the specified reference coordinates **must** not fail integer texel coordinate validation
- VUID-vkCmdSubpassShadingHUAWEI-OpImageWeightedSampleQCOM-06977
  If `OpImageWeightedSampleQCOM`, `OpImageBoxFilterQCOM`, `OpImageBlockMatchWindowSSDQCOM`, `OpImageBlockMatchWindowSADQCOM`, `OpImageBlockMatchGatherSSDQCOM`, `OpImageBlockMatchGatherSADQCOM`, `OpImageBlockMatchSSDQCOM`, or `OpImageBlockMatchSADQCOM` uses a VkSampler as a result of this command, then the sampler **must** have been created with `VK_SAMPLER_CREATE_IMAGE_PROCESSING_BIT_QCOM`
- VUID-vkCmdSubpassShadingHUAWEI-OpImageWeightedSampleQCOM-06978
  If any command other than `OpImageWeightedSampleQCOM`, `OpImageBoxFilterQCOM`, `OpImageBlockMatchWindowSSDQCOM`, `OpImageBlockMatchWindowSADQCOM`, `OpImageBlockMatchGatherSSDQCOM`, `OpImageBlockMatchGatherSADQCOM`, `OpImageBlockMatchSSDQCOM`, or `OpImageBlockMatchSADQCOM` uses a VkSampler as a result of this command, then the sampler **must** not have been created with `VK_SAMPLER_CREATE_IMAGE_PROCESSING_BIT_QCOM`
- VUID-vkCmdSubpassShadingHUAWEI-OpImageBlockMatchWindow-09215
  If a `OpImageBlockMatchWindow*QCOM` or `OpImageBlockMatchGather*QCOM` instruction is used to read from an VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM`
- VUID-vkCmdSubpassShadingHUAWEI-OpImageBlockMatchWindow-09216
  If a `OpImageBlockMatchWindow*QCOM` or `OpImageBlockMatchGather*QCOM` instruction is used to read from an VkImageView as a result of this command, then the image view's format **must** be a single-component format
- VUID-vkCmdSubpassShadingHUAWEI-OpImageBlockMatchWindow-09217
  If a `OpImageBlockMatchWindow*QCOM` or `OpImageBlockMatchGather*QCOM` read from a reference image as result of this command, then the specified reference coordinates **must** not fail integer texel coordinate validation
- VUID-vkCmdSubpassShadingHUAWEI-None-07288
  Any shader invocation executed by this command **must** terminate
- VUID-vkCmdSubpassShadingHUAWEI-None-09600
  If a descriptor with type equal to any of `VK_DESCRIPTOR_TYPE_SAMPLE_WEIGHT_IMAGE_QCOM`, `VK_DESCRIPTOR_TYPE_BLOCK_MATCH_IMAGE_QCOM`, `VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE`, `VK_DESCRIPTOR_TYPE_STORAGE_IMAGE`, or `VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT` is accessed as a result of this command, the image subresource identified by that descriptor **must** be in the image layout identified when the descriptor was written
- VUID-vkCmdSubpassShadingHUAWEI-None-04931
  This command **must** be called in a subpass with bind point `VK_PIPELINE_BIND_POINT_SUBPASS_SHADING_HUAWEI`. No draw commands can be called in the

same subpass. Only one vkCmdSubpassShadingHUAWEI command can be called in a subpass

### Valid Usage (Implicit)

- VUID-vkCmdSubpassShadingHUAWEI-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdSubpassShadingHUAWEI-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdSubpassShadingHUAWEI-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- VUID-vkCmdSubpassShadingHUAWEI-renderpass
  This command **must** only be called inside of a render pass instance
- VUID-vkCmdSubpassShadingHUAWEI-videocoding
  This command **must** only be called outside of a video coding scope

### Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

### Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Inside | Outside | Graphics | Action |

## 31.1. Dispatch Command for CUDA PTX Kernels

Compute kernels **can** be provided in SPIR-V or PTX code. When using PTX kernels the dispatch mechanism is different than with regular compute pipelines.

The way to create a PTX assembly file is beyond the scope of this documentation. For mode information, please refer to the CUDA toolkit documentation at https://docs.nvidia.com/cuda/.

Prior to using this command, you **must** initialize a CUDA module, and create a function handle that will serve as the entry point of the kernel to dispatch. See CUDA Modules.

The dispatching of a CUDA kernel is recorded into a command buffer, and when executed by a queue submit, will produce work which executes according to the bound compute pipeline.

To record a CUDA kernel launch, call:

```
// Provided by VK_NV_cuda_kernel_launch
void vkCmdCudaLaunchKernelNV(
    VkCommandBuffer                             commandBuffer,
    const VkCudaLaunchInfoNV*                   pLaunchInfo);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `pLaunchInfo` is a pointer to a VkCudaLaunchInfoNV structure in which the grid (similar to workgroup) dimension, function handle and related arguments are defined.

When the command is executed, a global workgroup consisting of `gridDimX` × `gridDimY` × `gridDimZ` local workgroups is assembled.

## Valid Usage (Implicit)

- VUID-vkCmdCudaLaunchKernelNV-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdCudaLaunchKernelNV-pLaunchInfo-parameter
  `pLaunchInfo` **must** be a valid pointer to a valid VkCudaLaunchInfoNV structure
- VUID-vkCmdCudaLaunchKernelNV-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdCudaLaunchKernelNV-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics, or compute operations
- VUID-vkCmdCudaLaunchKernelNV-videocoding
  This command **must** only be called outside of a video coding scope

## Host Synchronization

- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics<br>Compute | Action |

### 31.1.1. Passing Dispatch Parameters and Arguments

The `VkCudaLaunchInfoNV` structure is very close to the parameters of the CUDA-Driver function cuLaunchKernel documented in section 6.19 Execution Control of CUDA Driver API.

The structure is defined as:

```
// Provided by VK_NV_cuda_kernel_launch
typedef struct VkCudaLaunchInfoNV {
    VkStructureType        sType;
    const void*            pNext;
    VkCudaFunctionNV       function;
    uint32_t               gridDimX;
    uint32_t               gridDimY;
    uint32_t               gridDimZ;
    uint32_t               blockDimX;
    uint32_t               blockDimY;
    uint32_t               blockDimZ;
    uint32_t               sharedMemBytes;
    size_t                 paramCount;
    const void* const *    pParams;
    size_t                 extraCount;
    const void* const *    pExtras;
} VkCudaLaunchInfoNV;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `function` is the CUDA-Driver handle to the function being launched.
- `gridDimX` is the number of local workgroups to dispatch in the X dimension. It **must** be less than or equal to `VkPhysicalDeviceLimits`::`maxComputeWorkGroupCount`[0]
- `gridDimY` is the number of local workgroups to dispatch in the Y dimension. It **must** be less than or equal to `VkPhysicalDeviceLimits`::`maxComputeWorkGroupCount`[1]
- `gridDimZ` is the number of local workgroups to dispatch in the Z dimension. It **must** be less than or equal to `VkPhysicalDeviceLimits`::`maxComputeWorkGroupCount`[2]
- `blockDimX` is block size in the X dimension.
- `blockDimY` is block size in the Y dimension.
- `blockDimZ` is block size in the Z dimension.
- `sharedMemBytes` is the dynamic shared-memory size per thread block in bytes.
- `paramCount` is the length of the `pParams` table.
- `pParams` is a pointer to an array of `paramCount` pointers, corresponding to the arguments of `function`.
- `extraCount` is reserved for future use.
- `pExtras` is reserved for future use.

Kernel parameters of function are specified via pParams, very much the same way as described in cuLaunchKernel

If function has N parameters, then pParams **must** be an array of N pointers and paramCount **must** be set to N. Each of kernelParams[0] through kernelParams[N-1] **must** point to a region of memory from which the actual kernel parameter will be copied. The number of kernel parameters and their offsets and sizes are not specified here as that information is stored in the VkCudaFunctionNV object.

The application-owned memory pointed to by pParams and kernelParams[0] through kernelParams[N-1] are consumed immediately, and **may** be altered or freed after vkCmdCudaLaunchKernelNV has returned.

## Valid Usage

- VUID-VkCudaLaunchInfoNV-gridDimX-09406
  gridDimX **must** be less than or equal to VkPhysicalDeviceLimits::maxComputeWorkGroupCount[0]
- VUID-VkCudaLaunchInfoNV-gridDimY-09407
  gridDimY **must** be less than or equal to VkPhysicalDeviceLimits::maxComputeWorkGroupCount[1]
- VUID-VkCudaLaunchInfoNV-gridDimZ-09408
  gridDimZ **must** be less than or equal to VkPhysicalDeviceLimits::maxComputeWorkGroupCount[2]
- VUID-VkCudaLaunchInfoNV-paramCount-09409
  paramCount **must** be the total amount of parameters listed in the pParams table
- VUID-VkCudaLaunchInfoNV-pParams-09410
  pParams **must** be a pointer to a table of paramCount parameters, corresponding to the arguments of function
- VUID-VkCudaLaunchInfoNV-extraCount-09411
  extraCount **must** be 0
- VUID-VkCudaLaunchInfoNV-pExtras-09412
  pExtras **must** be NULL

## Valid Usage (Implicit)

- VUID-VkCudaLaunchInfoNV-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_CUDA_LAUNCH_INFO_NV
- VUID-VkCudaLaunchInfoNV-pNext-pNext
  pNext **must** be NULL
- VUID-VkCudaLaunchInfoNV-function-parameter
  function **must** be a valid VkCudaFunctionNV handle

### 31.1.2. Resource Sharing from Vulkan to the CUDA Kernel

Given that one key limitation of this extension is that Vulkan **cannot** access, nor bind any global resource of CUDA modules, the only way to exchange data with the kernel **must** be to *pass resources via the arguments of the function*.

You can use VK_KHR_buffer_device_address to write/read to/from a VkBuffer object. VK_KHR_buffer_device_address allows you to get the device address of the buffer to pass it as an argument into `pParams`. Application-side pointer arithmetic on the device address is legal, but will not be bounds-checked on the device.

The corresponding argument of the CUDA function **should** be declared as a pointer of the same type as the referenced buffer. CUDA code **may** simply read or write to this buffer in the typical C way.

You **may** also use VK_NVX_image_view_handle as another convenient way to read/write from/to a VkImage.

The corresponding argument of the CUDA function **must** be typed as `cudaSurfaceObject_t`.

- You **may** read from it by using CUDA surface-read functions such as `surf3Dread`, `surf2Dread`, and `surf1Dread`
- You **may** write to it by using CUDA surface-write functions such as `surf3Dwrite`, `surf2Dwrite`, and `surf1Dwrite`

Please refer to CUDA surface object documentation for more details

On Vulkan side, here is an example on how to setup VkImageViewHandleInfoNVX to query the handle for `cudaSurfaceObject_t`:

```
VkImageViewHandleInfoNVX imageViewHandleInfo =
{VK_STRUCTURE_TYPE_IMAGE_VIEW_HANDLE_INFO_NVX};
imageViewHandleInfo.sampler = VK_NULL_HANDLE;
imageViewHandleInfo.descriptorType = VK_DESCRIPTOR_TYPE_STORAGE_IMAGE;
imageViewHandleInfo.imageView = imageViewIn; // the VkImageView we want to access
uint32_t myViewHandleIn = vkGetImageViewHandleNVX(m_device, &imageViewHandleInfo);
imageViewHandleInfo.imageView = imageViewOut; // the VkImageView we want to access
uint32_t myViewHandleOut = vkGetImageViewHandleNVX(m_device, &imageViewHandleInfo);
```

Here is an example of how to declare parameters for `pParams`

```
VkCudaLaunchInfoNV launchInfo = { VK_STRUCTURE_TYPE_CUDA_LAUNCH_INFO_NV };

int block_size = 8;
float dt = 1.0f / 60.0f;

const void* params[] =
{
  &dt,
```

```
  &uint32_t myViewHandleIn,
  &uint32_t myViewHandleOut
};

launchInfo.function = cudaFunction; // CUDA function previously created
launchInfo.gridDimX = (volumeTexDimensionNonBoundary / block_size);
launchInfo.gridDimY = (volumeTexDimensionNonBoundary / block_size);
launchInfo.gridDimZ = (volumeTexDimensionNonBoundary / block_size);
launchInfo.blockDimX = block_size;
launchInfo.blockDimY = block_size;
launchInfo.blockDimZ = block_size;
launchInfo.sharedMemBytes = 0;
launchInfo.paramCount = 3;
launchInfo.pParams = &params[0];
launchInfo.extraCount = 0;
launchInfo.pExtras = nullptr;

vkCmdCudaLaunchKernelNV(commandBuffer, &launchInfo);
```

In the CUDA kernel source code, here is an example on how arguments match `pParams` and how we can use Surface object:

```
extern "C"  __global__ void cudaFunction(
  float dt,
  cudaSurfaceObject_t volumeTexIn,
  cudaSurfaceObject_t volumeTexOut
  )
{
  int i = 1 + blockIdx.x * blockDim.x + threadIdx.x;
  int j = 1 + blockIdx.y * blockDim.y + threadIdx.y;
  int k = 1 + blockIdx.z * blockDim.z + threadIdx.z;

  float val;
  surf3Dread(&val, volumeTexIn, i * sizeof(float), j, k);
  ...
  float result = ...;
  // write result
  surf3Dwrite(result, volumeTexOut, i * sizeof(float), j, k);
}
```

# Chapter 32. Device-Generated Commands

This chapter discusses the generation of command buffer content on the device, for which these principle steps are to be taken:

- Define via `VkIndirectCommandsLayoutNV` the sequence of commands which should be generated.
- Optionally make use of device-bindable Shader Groups for graphics pipelines.
- Retrieve device addresses by vkGetBufferDeviceAddressEXT for setting buffers on the device.
- Retrieve device addresses of compute pipelines by vkGetPipelineIndirectDeviceAddressNV to be able to bind it in device generated rendering.
- Fill one or more `VkBuffer` with the appropriate content that gets interpreted by `VkIndirectCommandsLayoutNV`.
- Create a `preprocess` `VkBuffer` using the allocation information from vkGetGeneratedCommandsMemoryRequirementsNV.
- Optionally preprocess the input data using vkCmdPreprocessGeneratedCommandsNV in a separate action.
- Generate and execute the actual commands via vkCmdExecuteGeneratedCommandsNV passing all required data.

vkCmdPreprocessGeneratedCommandsNV executes in a separate logical pipeline from either graphics or compute. When preprocessing commands in a separate step they **must** be explicitly synchronized against the command execution. When not preprocessing, the preprocessing is automatically synchronized against the command execution.

## 32.1. Indirect Commands Layout

The device-side command generation happens through an iterative processing of an atomic sequence comprised of command tokens, which are represented by:

```
// Provided by VK_NV_device_generated_commands
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkIndirectCommandsLayoutNV)
```

Each indirect command layout **must** have exactly one action command token and it **must** be the last token in the sequence.

### 32.1.1. Creation and Deletion

Indirect command layouts are created by:

```
// Provided by VK_NV_device_generated_commands
VkResult vkCreateIndirectCommandsLayoutNV(
    VkDevice                                    device,
    const VkIndirectCommandsLayoutCreateInfoNV* pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkIndirectCommandsLayoutNV*                 pIndirectCommandsLayout);
```

- `device` is the logical device that creates the indirect command layout.
- `pCreateInfo` is a pointer to a VkIndirectCommandsLayoutCreateInfoNV structure containing parameters affecting creation of the indirect command layout.
- `pAllocator` controls host memory allocation as described in the Memory Allocation chapter.
- `pIndirectCommandsLayout` is a pointer to a `VkIndirectCommandsLayoutNV` handle in which the resulting indirect command layout is returned.

## Valid Usage

- VUID-vkCreateIndirectCommandsLayoutNV-deviceGeneratedCommands-02929
  The `VkPhysicalDeviceDeviceGeneratedCommandsFeaturesNV::deviceGeneratedCommands` feature **must** be enabled

## Valid Usage (Implicit)

- VUID-vkCreateIndirectCommandsLayoutNV-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkCreateIndirectCommandsLayoutNV-pCreateInfo-parameter
  `pCreateInfo` **must** be a valid pointer to a valid VkIndirectCommandsLayoutCreateInfoNV structure
- VUID-vkCreateIndirectCommandsLayoutNV-pAllocator-parameter
  If `pAllocator` is not `NULL`, `pAllocator` **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkCreateIndirectCommandsLayoutNV-pIndirectCommandsLayout-parameter
  `pIndirectCommandsLayout` **must** be a valid pointer to a VkIndirectCommandsLayoutNV handle

## Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`

The VkIndirectCommandsLayoutCreateInfoNV structure is defined as:

```
// Provided by VK_NV_device_generated_commands
typedef struct VkIndirectCommandsLayoutCreateInfoNV {
    VkStructureType                             sType;
    const void*                                 pNext;
    VkIndirectCommandsLayoutUsageFlagsNV        flags;
    VkPipelineBindPoint                         pipelineBindPoint;
    uint32_t                                    tokenCount;
    const VkIndirectCommandsLayoutTokenNV*      pTokens;
    uint32_t                                    streamCount;
    const uint32_t*                             pStreamStrides;
} VkIndirectCommandsLayoutCreateInfoNV;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- pipelineBindPoint is the VkPipelineBindPoint that this layout targets.
- flags is a bitmask of VkIndirectCommandsLayoutUsageFlagBitsNV specifying usage hints of this layout.
- tokenCount is the length of the individual command sequence.
- pTokens is an array describing each command token in detail. See VkIndirectCommandsTokenTypeNV and VkIndirectCommandsLayoutTokenNV below for details.
- streamCount is the number of streams used to provide the token inputs.
- pStreamStrides is an array defining the byte stride for each input stream.

The following code illustrates some of the flags:

```
void cmdProcessAllSequences(cmd, pipeline, indirectCommandsLayout,
pIndirectCommandsTokens, sequencesCount, indexbuffer, indexbufferOffset)
{
  for (s = 0; s < sequencesCount; s++)
  {
    sUsed = s;

    if (indirectCommandsLayout.flags &
VK_INDIRECT_COMMANDS_LAYOUT_USAGE_INDEXED_SEQUENCES_BIT_NV) {
      sUsed = indexbuffer.load_uint32( sUsed * sizeof(uint32_t) + indexbufferOffset);
    }

    if (indirectCommandsLayout.flags &
VK_INDIRECT_COMMANDS_LAYOUT_USAGE_UNORDERED_SEQUENCES_BIT_NV) {
      sUsed = incoherent_implementation_dependent_permutation[ sUsed ];
    }

    cmdProcessSequence( cmd, pipeline, indirectCommandsLayout,
```

```
pIndirectCommandsTokens, sUsed );
  }
}
```

When tokens are consumed, an offset is computed based on token offset and stream stride. The resulting offset is required to be aligned. The alignment for a specific token is equal to the scalar alignment of the data type as defined in Alignment Requirements, or VkPhysicalDeviceDeviceGeneratedCommandsPropertiesNV::minIndirectCommandsBufferOffsetAlignment, whichever is lower.

| | |
|---|---|
| **NOTE** | A minIndirectCommandsBufferOffsetAlignment of 4 allows VkDeviceAddress to be packed as uvec2 with scalar layout instead of uint64_t with 8 byte alignment. This enables direct compatibility with D3D12 command signature layouts. |

## Valid Usage

- VUID-VkIndirectCommandsLayoutCreateInfoNV-pipelineBindPoint-02930
  The pipelineBindPoint **must** be VK_PIPELINE_BIND_POINT_GRAPHICS or VK_PIPELINE_BIND_POINT_COMPUTE
- VUID-VkIndirectCommandsLayoutCreateInfoNV-tokenCount-02931
  tokenCount **must** be greater than 0 and less than or equal to VkPhysicalDeviceDeviceGeneratedCommandsPropertiesNV::maxIndirectCommandsTokenCount
- VUID-VkIndirectCommandsLayoutCreateInfoNV-pTokens-02932
  If pTokens contains an entry of VK_INDIRECT_COMMANDS_TOKEN_TYPE_SHADER_GROUP_NV it **must** be the first element of the array and there **must** be only a single element of such token type
- VUID-VkIndirectCommandsLayoutCreateInfoNV-pTokens-09585
  If pTokens contains an entry of VK_INDIRECT_COMMANDS_TOKEN_TYPE_PIPELINE_NV it **must** be the first element of the array and there **must** be only a single element of such token type
- VUID-VkIndirectCommandsLayoutCreateInfoNV-pTokens-02933
  If pTokens contains an entry of VK_INDIRECT_COMMANDS_TOKEN_TYPE_STATE_FLAGS_NV there **must** be only a single element of such token type
- VUID-VkIndirectCommandsLayoutCreateInfoNV-pTokens-02934
  All state tokens in pTokens **must** occur before any action command tokens (VK_INDIRECT_COMMANDS_TOKEN_TYPE_DRAW_NV, VK_INDIRECT_COMMANDS_TOKEN_TYPE_DRAW_INDEXED_NV, VK_INDIRECT_COMMANDS_TOKEN_TYPE_DRAW_TASKS_NV, VK_INDIRECT_COMMANDS_TOKEN_TYPE_DRAW_MESH_TASKS_NV, VK_INDIRECT_COMMANDS_TOKEN_TYPE_DISPATCH_NV )
- VUID-VkIndirectCommandsLayoutCreateInfoNV-pTokens-02935
  The content of pTokens **must** include one single action command token that is compatible with the pipelineBindPoint
- VUID-VkIndirectCommandsLayoutCreateInfoNV-streamCount-02936
  streamCount **must** be greater than 0 and less or equal to

VkPhysicalDeviceDeviceGeneratedCommandsPropertiesNV::maxIndirectCommandsStreamCount

- VUID-VkIndirectCommandsLayoutCreateInfoNV-pStreamStrides-02937
  each element of pStreamStrides **must** be greater than 0 and less than or equal to VkPhysicalDeviceDeviceGeneratedCommandsPropertiesNV::maxIndirectCommandsStreamStride. Furthermore the alignment of each token input **must** be ensured
- VUID-VkIndirectCommandsLayoutCreateInfoNV-pipelineBindPoint-09088
  If pipelineBindPoint is VK_PIPELINE_BIND_POINT_COMPUTE then the VkPhysicalDeviceDeviceGeneratedCommandsComputeFeaturesNV::deviceGeneratedCompute feature **must** be enabled
- VUID-VkIndirectCommandsLayoutCreateInfoNV-pipelineBindPoint-09089
  If pipelineBindPoint is VK_PIPELINE_BIND_POINT_COMPUTE then the state tokens in pTokens **must** only include VK_INDIRECT_COMMANDS_TOKEN_TYPE_DISPATCH_NV, VK_INDIRECT_COMMANDS_TOKEN_TYPE_PIPELINE_NV, or VK_INDIRECT_COMMANDS_TOKEN_TYPE_PUSH_CONSTANT_NV
- VUID-VkIndirectCommandsLayoutCreateInfoNV-pipelineBindPoint-09090
  If pipelineBindPoint is VK_PIPELINE_BIND_POINT_COMPUTE and pTokens includes VK_INDIRECT_COMMANDS_TOKEN_TYPE_PIPELINE_NV, then the VkPhysicalDeviceDeviceGeneratedCommandsComputeFeaturesNV::deviceGeneratedComputePipelines feature **must** be enabled

## Valid Usage (Implicit)

- VUID-VkIndirectCommandsLayoutCreateInfoNV-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_INDIRECT_COMMANDS_LAYOUT_CREATE_INFO_NV
- VUID-VkIndirectCommandsLayoutCreateInfoNV-pNext-pNext
  pNext **must** be NULL
- VUID-VkIndirectCommandsLayoutCreateInfoNV-flags-parameter
  flags **must** be a valid combination of VkIndirectCommandsLayoutUsageFlagBitsNV values
- VUID-VkIndirectCommandsLayoutCreateInfoNV-pipelineBindPoint-parameter
  pipelineBindPoint **must** be a valid VkPipelineBindPoint value
- VUID-VkIndirectCommandsLayoutCreateInfoNV-pTokens-parameter
  pTokens **must** be a valid pointer to an array of tokenCount valid VkIndirectCommandsLayoutTokenNV structures
- VUID-VkIndirectCommandsLayoutCreateInfoNV-pStreamStrides-parameter
  pStreamStrides **must** be a valid pointer to an array of streamCount uint32_t values
- VUID-VkIndirectCommandsLayoutCreateInfoNV-tokenCount-arraylength
  tokenCount **must** be greater than 0
- VUID-VkIndirectCommandsLayoutCreateInfoNV-streamCount-arraylength
  streamCount **must** be greater than 0

Bits which **can** be set in VkIndirectCommandsLayoutCreateInfoNV::flags, specifying usage hints of

an indirect command layout, are:

```
// Provided by VK_NV_device_generated_commands
typedef enum VkIndirectCommandsLayoutUsageFlagBitsNV {
    VK_INDIRECT_COMMANDS_LAYOUT_USAGE_EXPLICIT_PREPROCESS_BIT_NV = 0x00000001,
    VK_INDIRECT_COMMANDS_LAYOUT_USAGE_INDEXED_SEQUENCES_BIT_NV = 0x00000002,
    VK_INDIRECT_COMMANDS_LAYOUT_USAGE_UNORDERED_SEQUENCES_BIT_NV = 0x00000004,
} VkIndirectCommandsLayoutUsageFlagBitsNV;
```

- `VK_INDIRECT_COMMANDS_LAYOUT_USAGE_EXPLICIT_PREPROCESS_BIT_NV` specifies that the layout is always used with the manual preprocessing step through calling vkCmdPreprocessGeneratedCommandsNV and executed by vkCmdExecuteGeneratedCommandsNV with `isPreprocessed` set to `VK_TRUE`.
- `VK_INDIRECT_COMMANDS_LAYOUT_USAGE_INDEXED_SEQUENCES_BIT_NV` specifies that the input data for the sequences is not implicitly indexed from 0..sequencesUsed, but an application-provided `VkBuffer` encoding the index is provided.
- `VK_INDIRECT_COMMANDS_LAYOUT_USAGE_UNORDERED_SEQUENCES_BIT_NV` specifies that the processing of sequences **can** happen at an implementation-dependent order, which is not: guaranteed to be coherent using the same input data. This flag is ignored when the `pipelineBindPoint` is `VK_PIPELINE_BIND_POINT_COMPUTE` as it is implied that the dispatch sequence is always unordered.

```
// Provided by VK_NV_device_generated_commands
typedef VkFlags VkIndirectCommandsLayoutUsageFlagsNV;
```

`VkIndirectCommandsLayoutUsageFlagsNV` is a bitmask type for setting a mask of zero or more VkIndirectCommandsLayoutUsageFlagBitsNV.

Indirect command layouts are destroyed by:

```
// Provided by VK_NV_device_generated_commands
void vkDestroyIndirectCommandsLayoutNV(
    VkDevice                                    device,
    VkIndirectCommandsLayoutNV                  indirectCommandsLayout,
    const VkAllocationCallbacks*                pAllocator);
```

- `device` is the logical device that destroys the layout.
- `indirectCommandsLayout` is the layout to destroy.
- `pAllocator` controls host memory allocation as described in the Memory Allocation chapter.

### Valid Usage

- VUID-vkDestroyIndirectCommandsLayoutNV-indirectCommandsLayout-02938
  All submitted commands that refer to `indirectCommandsLayout` **must** have completed execution

- VUID-vkDestroyIndirectCommandsLayoutNV-indirectCommandsLayout-02939
  If `VkAllocationCallbacks` were provided when `indirectCommandsLayout` was created, a compatible set of callbacks **must** be provided here
- VUID-vkDestroyIndirectCommandsLayoutNV-indirectCommandsLayout-02940
  If no `VkAllocationCallbacks` were provided when `indirectCommandsLayout` was created, `pAllocator` **must** be `NULL`
- VUID-vkDestroyIndirectCommandsLayoutNV-deviceGeneratedCommands-02941
  The `VkPhysicalDeviceDeviceGeneratedCommandsFeaturesNV::deviceGeneratedCommands` feature **must** be enabled

### Valid Usage (Implicit)

- VUID-vkDestroyIndirectCommandsLayoutNV-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkDestroyIndirectCommandsLayoutNV-indirectCommandsLayout-parameter
  If `indirectCommandsLayout` is not VK_NULL_HANDLE, `indirectCommandsLayout` **must** be a valid VkIndirectCommandsLayoutNV handle
- VUID-vkDestroyIndirectCommandsLayoutNV-pAllocator-parameter
  If `pAllocator` is not `NULL`, `pAllocator` **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkDestroyIndirectCommandsLayoutNV-indirectCommandsLayout-parent
  If `indirectCommandsLayout` is a valid handle, it **must** have been created, allocated, or retrieved from `device`

### Host Synchronization

- Host access to `indirectCommandsLayout` **must** be externally synchronized

## 32.1.2. Token Input Streams

The `VkIndirectCommandsStreamNV` structure specifies the input data for one or more tokens at processing time.

```
// Provided by VK_NV_device_generated_commands
typedef struct VkIndirectCommandsStreamNV {
    VkBuffer        buffer;
    VkDeviceSize    offset;
} VkIndirectCommandsStreamNV;
```

- `buffer` specifies the VkBuffer storing the functional arguments for each sequence. These arguments **can** be written by the device.
- `offset` specified an offset into `buffer` where the arguments start.

### Valid Usage

- VUID-VkIndirectCommandsStreamNV-buffer-02942
  The `buffer`'s usage flag **must** have the `VK_BUFFER_USAGE_INDIRECT_BUFFER_BIT` bit set
- VUID-VkIndirectCommandsStreamNV-offset-02943
  The `offset` **must** be aligned to `VkPhysicalDeviceDeviceGeneratedCommandsPropertiesNV::minIndirectCommandsBufferOffsetAlignment`
- VUID-VkIndirectCommandsStreamNV-buffer-02975
  If `buffer` is non-sparse then it **must** be bound completely and contiguously to a single `VkDeviceMemory` object

### Valid Usage (Implicit)

- VUID-VkIndirectCommandsStreamNV-buffer-parameter
  `buffer` **must** be a valid VkBuffer handle

The input streams **can** contain raw `uint32_t` values, existing indirect commands such as:

- VkDrawIndirectCommand
- VkDrawIndexedIndirectCommand
- VkDrawMeshTasksIndirectCommandNV
- VkDrawMeshTasksIndirectCommandEXT
- VkDispatchIndirectCommand

or additional commands as listed below. How the data is used is described in the next section.

The `VkBindShaderGroupIndirectCommandNV` structure specifies the input data for the `VK_INDIRECT_COMMANDS_TOKEN_TYPE_SHADER_GROUP_NV` token.

```
// Provided by VK_NV_device_generated_commands
typedef struct VkBindShaderGroupIndirectCommandNV {
    uint32_t    groupIndex;
} VkBindShaderGroupIndirectCommandNV;
```

- `groupIndex` specifies which shader group of the current bound graphics pipeline is used.

### Valid Usage

- VUID-VkBindShaderGroupIndirectCommandNV-None-02944
  The current bound graphics pipeline, as well as the pipelines it may reference, **must** have been created with `VK_PIPELINE_CREATE_INDIRECT_BINDABLE_BIT_NV`
- VUID-VkBindShaderGroupIndirectCommandNV-index-02945

The index **must** be within range of the accessible shader groups of the current bound graphics pipeline. See vkCmdBindPipelineShaderGroupNV for further details

The VkBindIndexBufferIndirectCommandNV structure specifies the input data for the VK_INDIRECT_COMMANDS_TOKEN_TYPE_INDEX_BUFFER_NV token.

```
// Provided by VK_NV_device_generated_commands
typedef struct VkBindIndexBufferIndirectCommandNV {
    VkDeviceAddress    bufferAddress;
    uint32_t           size;
    VkIndexType        indexType;
} VkBindIndexBufferIndirectCommandNV;
```

- bufferAddress specifies a physical address of the VkBuffer used as index buffer.
- size is the byte size range which is available for this operation from the provided address.
- indexType is a VkIndexType value specifying how indices are treated. Instead of the Vulkan enum values, a custom uint32_t value **can** be mapped to VkIndexType by specifying the VkIndirectCommandsLayoutTokenNV::pIndexTypes and VkIndirectCommandsLayoutTokenNV::pIndexTypeValues arrays.

## Valid Usage

- VUID-VkBindIndexBufferIndirectCommandNV-None-02946
  The buffer's usage flag from which the address was acquired **must** have the VK_BUFFER_USAGE_INDEX_BUFFER_BIT bit set
- VUID-VkBindIndexBufferIndirectCommandNV-bufferAddress-02947
  The bufferAddress **must** be aligned to the indexType used
- VUID-VkBindIndexBufferIndirectCommandNV-None-02948
  Each element of the buffer from which the address was acquired and that is non-sparse **must** be bound completely and contiguously to a single VkDeviceMemory object

## Valid Usage (Implicit)

- VUID-VkBindIndexBufferIndirectCommandNV-indexType-parameter
  indexType **must** be a valid VkIndexType value

The VkBindVertexBufferIndirectCommandNV structure specifies the input data for the VK_INDIRECT_COMMANDS_TOKEN_TYPE_VERTEX_BUFFER_NV token.

```
// Provided by VK_NV_device_generated_commands
typedef struct VkBindVertexBufferIndirectCommandNV {
    VkDeviceAddress    bufferAddress;
    uint32_t           size;
    uint32_t           stride;
} VkBindVertexBufferIndirectCommandNV;
```

- `bufferAddress` specifies a physical address of the VkBuffer used as vertex input binding.
- `size` is the byte size range which is available for this operation from the provided address.
- `stride` is the byte size stride for this vertex input binding as in `VkVertexInputBindingDescription::stride`. It is only used if `VkIndirectCommandsLayoutTokenNV::vertexDynamicStride` was set, otherwise the stride is inherited from the current bound graphics pipeline.

### Valid Usage

- VUID-VkBindVertexBufferIndirectCommandNV-None-02949
  The buffer's usage flag from which the address was acquired **must** have the `VK_BUFFER_USAGE_VERTEX_BUFFER_BIT` bit set
- VUID-VkBindVertexBufferIndirectCommandNV-None-02950
  Each element of the buffer from which the address was acquired and that is non-sparse **must** be bound completely and contiguously to a single `VkDeviceMemory` object

The `VkSetStateFlagsIndirectCommandNV` structure specifies the input data for the `VK_INDIRECT_COMMANDS_TOKEN_TYPE_STATE_FLAGS_NV` token. Which state is changed depends on the VkIndirectStateFlagBitsNV specified at `VkIndirectCommandsLayoutNV` creation time.

```
// Provided by VK_NV_device_generated_commands
typedef struct VkSetStateFlagsIndirectCommandNV {
    uint32_t    data;
} VkSetStateFlagsIndirectCommandNV;
```

- `data` encodes packed state that this command alters.
  - Bit `0`: If set represents `VK_FRONT_FACE_CLOCKWISE`, otherwise `VK_FRONT_FACE_COUNTER_CLOCKWISE`

A subset of the graphics pipeline state **can** be altered using indirect state flags:

```
// Provided by VK_NV_device_generated_commands
typedef enum VkIndirectStateFlagBitsNV {
    VK_INDIRECT_STATE_FLAG_FRONTFACE_BIT_NV = 0x00000001,
} VkIndirectStateFlagBitsNV;
```

- `VK_INDIRECT_STATE_FLAG_FRONTFACE_BIT_NV` allows to toggle the VkFrontFace rasterization state for subsequent drawing commands.

```
// Provided by VK_NV_device_generated_commands
typedef VkFlags VkIndirectStateFlagsNV;
```

VkIndirectStateFlagsNV is a bitmask type for setting a mask of zero or more VkIndirectStateFlagBitsNV.

The VkBindPipelineIndirectCommandNV structure specifies the input data for the VK_INDIRECT_COMMANDS_TOKEN_TYPE_PIPELINE_NV token.

```
// Provided by VK_NV_device_generated_commands_compute
typedef struct VkBindPipelineIndirectCommandNV {
    VkDeviceAddress    pipelineAddress;
} VkBindPipelineIndirectCommandNV;
```

- pipelineAddress specifies the pipeline address of the compute pipeline that will be used in device generated rendering.

**Valid Usage**

- VUID-VkBindPipelineIndirectCommandNV-deviceGeneratedComputePipelines-09091
  The VkPhysicalDeviceDeviceGeneratedCommandsComputeFeaturesNV::deviceGeneratedComputePipelines feature **must** be enabled
- VUID-VkBindPipelineIndirectCommandNV-None-09092
  The referenced pipeline **must** have been created with VK_PIPELINE_CREATE_INDIRECT_BINDABLE_BIT_NV
- VUID-VkBindPipelineIndirectCommandNV-None-09093
  The referenced pipeline **must** have been updated with vkCmdUpdatePipelineIndirectBufferNV
- VUID-VkBindPipelineIndirectCommandNV-None-09094
  The referenced pipeline's address **must** have been queried with vkGetPipelineIndirectDeviceAddressNV

### 32.1.3. Tokenized Command Processing

The processing is in principle illustrated below:

```
void cmdProcessSequence(cmd, pipeline, indirectCommandsLayout,
pIndirectCommandsStreams, s)
{
  for (t = 0; t < indirectCommandsLayout.tokenCount; t++)
  {
    uint32_t stream  = indirectCommandsLayout.pTokens[t].stream;
    uint32_t offset  = indirectCommandsLayout.pTokens[t].offset;
    uint32_t stride  = indirectCommandsLayout.pStreamStrides[stream];
```

```
    stream            = pIndirectCommandsStreams[stream];
    const void* input = stream.buffer.pointer( stream.offset + stride * s + offset )

    // further details later
    indirectCommandsLayout.pTokens[t].command (cmd, pipeline, input, s);
  }
}

void cmdProcessAllSequences(cmd, pipeline, indirectCommandsLayout,
pIndirectCommandsStreams, sequencesCount)
{
  for (s = 0; s < sequencesCount; s++)
  {
    cmdProcessSequence(cmd, pipeline, indirectCommandsLayout,
pIndirectCommandsStreams, s);
  }
}
```

The processing of each sequence is considered stateless, therefore all state changes **must** occur before any action command tokens within the sequence. A single sequence is strictly targeting the VkPipelineBindPoint it was created with.

The primary input data for each token is provided through `VkBuffer` content at preprocessing using vkCmdPreprocessGeneratedCommandsNV or execution time using vkCmdExecuteGeneratedCommandsNV, however some functional arguments, for example binding sets, are specified at layout creation time. The input size is different for each token.

Possible values of those elements of the VkIndirectCommandsLayoutCreateInfoNV::`pTokens` array specifying command tokens (other elements of the array specify command parameters) are:

```
// Provided by VK_NV_device_generated_commands
typedef enum VkIndirectCommandsTokenTypeNV {
    VK_INDIRECT_COMMANDS_TOKEN_TYPE_SHADER_GROUP_NV = 0,
    VK_INDIRECT_COMMANDS_TOKEN_TYPE_STATE_FLAGS_NV = 1,
    VK_INDIRECT_COMMANDS_TOKEN_TYPE_INDEX_BUFFER_NV = 2,
    VK_INDIRECT_COMMANDS_TOKEN_TYPE_VERTEX_BUFFER_NV = 3,
    VK_INDIRECT_COMMANDS_TOKEN_TYPE_PUSH_CONSTANT_NV = 4,
    VK_INDIRECT_COMMANDS_TOKEN_TYPE_DRAW_INDEXED_NV = 5,
    VK_INDIRECT_COMMANDS_TOKEN_TYPE_DRAW_NV = 6,
    VK_INDIRECT_COMMANDS_TOKEN_TYPE_DRAW_TASKS_NV = 7,
  // Provided by VK_EXT_mesh_shader with VK_NV_device_generated_commands
    VK_INDIRECT_COMMANDS_TOKEN_TYPE_DRAW_MESH_TASKS_NV = 1000328000,
  // Provided by VK_NV_device_generated_commands_compute
    VK_INDIRECT_COMMANDS_TOKEN_TYPE_PIPELINE_NV = 1000428003,
  // Provided by VK_NV_device_generated_commands_compute
    VK_INDIRECT_COMMANDS_TOKEN_TYPE_DISPATCH_NV = 1000428004,
} VkIndirectCommandsTokenTypeNV;
```

*Table 46. Supported indirect command tokens*

| Token type | Equivalent command |
|---|---|
| VK_INDIRECT_COMMANDS_TOKEN_TYPE_SHADER_GROUP_NV | vkCmdBindPipelineShaderGroupNV |
| VK_INDIRECT_COMMANDS_TOKEN_TYPE_STATE_FLAGS_NV | - |
| VK_INDIRECT_COMMANDS_TOKEN_TYPE_INDEX_BUFFER_NV | vkCmdBindIndexBuffer |
| VK_INDIRECT_COMMANDS_TOKEN_TYPE_VERTEX_BUFFER_NV | vkCmdBindVertexBuffers |
| VK_INDIRECT_COMMANDS_TOKEN_TYPE_PUSH_CONSTANT_NV | vkCmdPushConstants |
| VK_INDIRECT_COMMANDS_TOKEN_TYPE_DRAW_INDEXED_NV | vkCmdDrawIndexedIndirect |
| VK_INDIRECT_COMMANDS_TOKEN_TYPE_DRAW_NV | vkCmdDrawIndirect |
| VK_INDIRECT_COMMANDS_TOKEN_TYPE_DRAW_TASKS_NV | vkCmdDrawMeshTasksIndirectNV |
| VK_INDIRECT_COMMANDS_TOKEN_TYPE_DRAW_MESH_TASKS_NV | vkCmdDrawMeshTasksIndirectEXT |
| VK_INDIRECT_COMMANDS_TOKEN_TYPE_PIPELINE_NV | vkCmdBindPipeline |
| VK_INDIRECT_COMMANDS_TOKEN_TYPE_DISPATCH_NV | vkCmdDispatchIndirect |

The VkIndirectCommandsLayoutTokenNV structure specifies details to the function arguments that need to be known at layout creation time:

```
// Provided by VK_NV_device_generated_commands
typedef struct VkIndirectCommandsLayoutTokenNV {
    VkStructureType                 sType;
    const void*                     pNext;
    VkIndirectCommandsTokenTypeNV   tokenType;
    uint32_t                        stream;
    uint32_t                        offset;
    uint32_t                        vertexBindingUnit;
    VkBool32                        vertexDynamicStride;
    VkPipelineLayout                pushconstantPipelineLayout;
    VkShaderStageFlags              pushconstantShaderStageFlags;
    uint32_t                        pushconstantOffset;
    uint32_t                        pushconstantSize;
    VkIndirectStateFlagsNV          indirectStateFlags;
    uint32_t                        indexTypeCount;
    const VkIndexType*              pIndexTypes;
    const uint32_t*                 pIndexTypeValues;
} VkIndirectCommandsLayoutTokenNV;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- tokenType is a VkIndirectCommandsTokenTypeNV specifying the token command type.

- stream is the index of the input stream containing the token argument data.
- offset is a relative starting offset within the input stream memory for the token argument data.
- vertexBindingUnit is used for the vertex buffer binding command.
- vertexDynamicStride sets if the vertex buffer stride is provided by the binding command rather than the current bound graphics pipeline state.
- pushconstantPipelineLayout is the VkPipelineLayout used for the push constant command.
- pushconstantShaderStageFlags are the shader stage flags used for the push constant command.
- pushconstantOffset is the offset used for the push constant command.
- pushconstantSize is the size used for the push constant command.
- indirectStateFlags is a VkIndirectStateFlagsNV bitfield indicating the active states for the state flag command.
- indexTypeCount is the optional size of the pIndexTypes and pIndexTypeValues array pairings. If not zero, it allows to register a custom uint32_t value to be treated as specific VkIndexType.
- pIndexTypes is the used VkIndexType for the corresponding uint32_t value entry in pIndexTypeValues.

## Valid Usage

- VUID-VkIndirectCommandsLayoutTokenNV-stream-02951
  stream **must** be smaller than VkIndirectCommandsLayoutCreateInfoNV::streamCount
- VUID-VkIndirectCommandsLayoutTokenNV-offset-02952
  offset **must** be less than or equal to VkPhysicalDeviceDeviceGeneratedCommandsPropertiesNV::maxIndirectCommandsTokenOffset
- VUID-VkIndirectCommandsLayoutTokenNV-offset-06888
  offset **must** be aligned to the scalar alignment of tokenType or minIndirectCommandsBufferOffsetAlignment, whichever is lower
- VUID-VkIndirectCommandsLayoutTokenNV-tokenType-02976
  If tokenType is VK_INDIRECT_COMMANDS_TOKEN_TYPE_VERTEX_BUFFER_NV, vertexBindingUnit **must** stay within device supported limits for the appropriate commands
- VUID-VkIndirectCommandsLayoutTokenNV-tokenType-02977
  If tokenType is VK_INDIRECT_COMMANDS_TOKEN_TYPE_PUSH_CONSTANT_NV, pushconstantPipelineLayout **must** be valid
- VUID-VkIndirectCommandsLayoutTokenNV-tokenType-02978
  If tokenType is VK_INDIRECT_COMMANDS_TOKEN_TYPE_PUSH_CONSTANT_NV, pushconstantOffset **must** be a multiple of 4
- VUID-VkIndirectCommandsLayoutTokenNV-tokenType-02979
  If tokenType is VK_INDIRECT_COMMANDS_TOKEN_TYPE_PUSH_CONSTANT_NV, pushconstantSize **must** be a multiple of 4
- VUID-VkIndirectCommandsLayoutTokenNV-tokenType-02980
  If tokenType is VK_INDIRECT_COMMANDS_TOKEN_TYPE_PUSH_CONSTANT_NV, pushconstantOffset **must** be less than VkPhysicalDeviceLimits::maxPushConstantsSize

- VUID-VkIndirectCommandsLayoutTokenNV-tokenType-02981
  If `tokenType` is `VK_INDIRECT_COMMANDS_TOKEN_TYPE_PUSH_CONSTANT_NV`, `pushconstantSize` **must** be less than or equal to `VkPhysicalDeviceLimits::maxPushConstantsSize` minus `pushconstantOffset`
- VUID-VkIndirectCommandsLayoutTokenNV-tokenType-02982
  If `tokenType` is `VK_INDIRECT_COMMANDS_TOKEN_TYPE_PUSH_CONSTANT_NV`, for each byte in the range specified by `pushconstantOffset` and `pushconstantSize` and for each shader stage in `pushconstantShaderStageFlags`, there **must** be a push constant range in `pushconstantPipelineLayout` that includes that byte and that stage
- VUID-VkIndirectCommandsLayoutTokenNV-tokenType-02983
  If `tokenType` is `VK_INDIRECT_COMMANDS_TOKEN_TYPE_PUSH_CONSTANT_NV`, for each byte in the range specified by `pushconstantOffset` and `pushconstantSize` and for each push constant range that overlaps that byte, `pushconstantShaderStageFlags` **must** include all stages in that push constant range's VkPushConstantRange::`stageFlags`
- VUID-VkIndirectCommandsLayoutTokenNV-tokenType-02984
  If `tokenType` is `VK_INDIRECT_COMMANDS_TOKEN_TYPE_STATE_FLAGS_NV`, `indirectStateFlags` **must** not be `0`

## Valid Usage (Implicit)

- VUID-VkIndirectCommandsLayoutTokenNV-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_INDIRECT_COMMANDS_LAYOUT_TOKEN_NV`
- VUID-VkIndirectCommandsLayoutTokenNV-pNext-pNext
  `pNext` **must** be `NULL`
- VUID-VkIndirectCommandsLayoutTokenNV-tokenType-parameter
  `tokenType` **must** be a valid VkIndirectCommandsTokenTypeNV value
- VUID-VkIndirectCommandsLayoutTokenNV-pushconstantPipelineLayout-parameter
  If `pushconstantPipelineLayout` is not VK_NULL_HANDLE, `pushconstantPipelineLayout` **must** be a valid VkPipelineLayout handle
- VUID-VkIndirectCommandsLayoutTokenNV-pushconstantShaderStageFlags-parameter
  `pushconstantShaderStageFlags` **must** be a valid combination of VkShaderStageFlagBits values
- VUID-VkIndirectCommandsLayoutTokenNV-indirectStateFlags-parameter
  `indirectStateFlags` **must** be a valid combination of VkIndirectStateFlagBitsNV values
- VUID-VkIndirectCommandsLayoutTokenNV-pIndexTypes-parameter
  If `indexTypeCount` is not `0`, `pIndexTypes` **must** be a valid pointer to an array of `indexTypeCount` valid VkIndexType values
- VUID-VkIndirectCommandsLayoutTokenNV-pIndexTypeValues-parameter
  If `indexTypeCount` is not `0`, `pIndexTypeValues` **must** be a valid pointer to an array of `indexTypeCount` `uint32_t` values

The following code provides detailed information on how an individual sequence is processed. For

valid usage, all restrictions from the regular commands apply.

```
void cmdProcessSequence(cmd, pipeline, indirectCommandsLayout,
pIndirectCommandsStreams, s)
{
  for (uint32_t t = 0; t < indirectCommandsLayout.tokenCount; t++){
    token = indirectCommandsLayout.pTokens[t];

    uint32_t stride   = indirectCommandsLayout.pStreamStrides[token.stream];
    stream            = pIndirectCommandsStreams[token.stream];
    uint32_t offset   = stream.offset + stride * s + token.offset;
    const void* input = stream.buffer.pointer( offset )

    switch(input.type){
    case VK_INDIRECT_COMMANDS_TOKEN_TYPE_SHADER_GROUP_NV:
      VkBindShaderGroupIndirectCommandNV* bind = input;

      vkCmdBindPipelineShaderGroupNV(cmd, indirectCommandsLayout.pipelineBindPoint,
        pipeline, bind->groupIndex);
    break;

    case VK_INDIRECT_COMMANDS_TOKEN_TYPE_STATE_FLAGS_NV:
      VkSetStateFlagsIndirectCommandNV* state = input;

      if (token.indirectStateFlags & VK_INDIRECT_STATE_FLAG_FRONTFACE_BIT_NV){
        if (state.data & (1 << 0)){
          set VK_FRONT_FACE_CLOCKWISE;
        } else {
          set VK_FRONT_FACE_COUNTER_CLOCKWISE;
        }
      }
    break;

    case VK_INDIRECT_COMMANDS_TOKEN_TYPE_PUSH_CONSTANT_NV:
      uint32_t* data = input;

      vkCmdPushConstants(cmd,
        token.pushconstantPipelineLayout
        token.pushconstantStageFlags,
        token.pushconstantOffset,
        token.pushconstantSize, data);
    break;

    case VK_INDIRECT_COMMANDS_TOKEN_TYPE_INDEX_BUFFER_NV:
      VkBindIndexBufferIndirectCommandNV* data = input;

      // the indexType may optionally be remapped
      // from a custom uint32_t value, via
      // VkIndirectCommandsLayoutTokenNV::pIndexTypeValues
```

```
      vkCmdBindIndexBuffer(cmd,
        deriveBuffer(data->bufferAddress),
        deriveOffset(data->bufferAddress),
        data->indexType);
    break;

    case VK_INDIRECT_COMMANDS_TOKEN_TYPE_VERTEX_BUFFER_NV:
      VkBindVertexBufferIndirectCommandNV* data = input;

      // if token.vertexDynamicStride is VK_TRUE
      // then the stride for this binding is set
      // using data->stride as well

      vkCmdBindVertexBuffers(cmd,
        token.vertexBindingUnit, 1,
        &deriveBuffer(data->bufferAddress),
        &deriveOffset(data->bufferAddress));
    break;

    case VK_INDIRECT_COMMANDS_TOKEN_TYPE_DRAW_INDEXED_NV:
      vkCmdDrawIndexedIndirect(cmd,
        stream.buffer, offset, 1, 0);
    break;

    case VK_INDIRECT_COMMANDS_TOKEN_TYPE_DRAW_NV:
      vkCmdDrawIndirect(cmd,
        stream.buffer,
        offset, 1, 0);
    break;

    // only available if VK_NV_mesh_shader is supported
    case VK_INDIRECT_COMMANDS_TOKEN_TYPE_DRAW_TASKS_NV:
      vkCmdDrawMeshTasksIndirectNV(cmd,
        stream.buffer, offset, 1, 0);
    break;

    // only available if VK_EXT_mesh_shader is supported
    case VK_INDIRECT_COMMANDS_TOKEN_TYPE_DRAW_MESH_TASKS_NV:
      vkCmdDrawMeshTasksIndirectEXT(cmd,
        stream.buffer, offset, 1, 0);
    break;

    case VK_INDIRECT_COMMANDS_TOKEN_TYPE_PIPELINE_NV:
      VkBindPipelineIndirectCommandNV *data = input;
      VkPipeline computePipeline = deriveFromDeviceAddress(data->pipelineAddress);
      vkCmdBindPipeline(cmd, VK_PIPELINE_BIND_POINT_COMPUTE, computePipeline);
    break;

    case VK_INDIRECT_COMMANDS_TOKEN_TYPE_DISPATCH_NV:
      vkCmdDispatchIndirect(cmd, stream.buffer, offset);
    break;
```

```
    }
  }
}
```

## 32.2. Indirect Commands Generation and Execution

The generation of commands on the device requires a `preprocess` buffer. To retrieve the memory size and alignment requirements of a particular execution state call:

```
// Provided by VK_NV_device_generated_commands
void vkGetGeneratedCommandsMemoryRequirementsNV(
    VkDevice                                    device,
    const VkGeneratedCommandsMemoryRequirementsInfoNV* pInfo,
    VkMemoryRequirements2*                      pMemoryRequirements);
```

- `device` is the logical device that owns the buffer.
- `pInfo` is a pointer to a VkGeneratedCommandsMemoryRequirementsInfoNV structure containing parameters required for the memory requirements query.
- `pMemoryRequirements` is a pointer to a VkMemoryRequirements2 structure in which the memory requirements of the buffer object are returned.

### Valid Usage

- VUID-vkGetGeneratedCommandsMemoryRequirementsNV-deviceGeneratedCommands-02906
  The `VkPhysicalDeviceDeviceGeneratedCommandsFeaturesNV::deviceGeneratedCommands` feature **must** be enabled
- VUID-vkGetGeneratedCommandsMemoryRequirementsNV-pInfo-09074
  If `pInfo->pipelineBindPoint` is of type `VK_PIPELINE_BIND_POINT_COMPUTE`, then the `VkPhysicalDeviceDeviceGeneratedCommandsComputeFeaturesNV::deviceGeneratedCompute` feature **must** be enabled

### Valid Usage (Implicit)

- VUID-vkGetGeneratedCommandsMemoryRequirementsNV-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkGetGeneratedCommandsMemoryRequirementsNV-pInfo-parameter
  `pInfo` **must** be a valid pointer to a valid VkGeneratedCommandsMemoryRequirementsInfoNV structure
- VUID-vkGetGeneratedCommandsMemoryRequirementsNV-pMemoryRequirements-parameter
  `pMemoryRequirements` **must** be a valid pointer to a VkMemoryRequirements2 structure

```
// Provided by VK_NV_device_generated_commands
typedef struct VkGeneratedCommandsMemoryRequirementsInfoNV {
    VkStructureType                 sType;
    const void*                     pNext;
    VkPipelineBindPoint             pipelineBindPoint;
    VkPipeline                      pipeline;
    VkIndirectCommandsLayoutNV      indirectCommandsLayout;
    uint32_t                        maxSequencesCount;
} VkGeneratedCommandsMemoryRequirementsInfoNV;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `pipelineBindPoint` is the VkPipelineBindPoint of the `pipeline` that this buffer memory is intended to be used with during the execution.
- `pipeline` is the VkPipeline that this buffer memory is intended to be used with during the execution.
- `indirectCommandsLayout` is the VkIndirectCommandsLayoutNV that this buffer memory is intended to be used with.
- `maxSequencesCount` is the maximum number of sequences that this buffer memory in combination with the other state provided **can** be used with.

## Valid Usage

- VUID-VkGeneratedCommandsMemoryRequirementsInfoNV-maxSequencesCount-02907
  `maxSequencesCount` **must** be less or equal to VkPhysicalDeviceDeviceGeneratedCommandsPropertiesNV::`maxIndirectSequenceCount`
- VUID-VkGeneratedCommandsMemoryRequirementsInfoNV-pipelineBindPoint-09075
  If `pipelineBindPoint` is of type `VK_PIPELINE_BIND_POINT_GRAPHICS`, then `pipeline` **must** be a valid VkPipeline handle
- VUID-VkGeneratedCommandsMemoryRequirementsInfoNV-pipelineBindPoint-09076
  If `pipelineBindPoint` is of type `VK_PIPELINE_BIND_POINT_COMPUTE`, and the `indirectCommandsLayout` was not created with a `VK_INDIRECT_COMMANDS_TOKEN_TYPE_PIPELINE_NV` token, then the `pipeline` **must** be a valid VkPipeline handle
- VUID-VkGeneratedCommandsMemoryRequirementsInfoNV-pipelineBindPoint-09077
  If `pipelineBindPoint` is of type `VK_PIPELINE_BIND_POINT_COMPUTE`, and the `indirectCommandsLayout` contains a `VK_INDIRECT_COMMANDS_TOKEN_TYPE_PIPELINE_NV` token, then the `pipeline` **must** be VK_NULL_HANDLE

## Valid Usage (Implicit)

- VUID-VkGeneratedCommandsMemoryRequirementsInfoNV-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_GENERATED_COMMANDS_MEMORY_REQUIREMENTS_INFO_NV`

- VUID-VkGeneratedCommandsMemoryRequirementsInfoNV-pNext-pNext
  `pNext` **must** be `NULL`
- VUID-VkGeneratedCommandsMemoryRequirementsInfoNV-pipelineBindPoint-parameter
  `pipelineBindPoint` **must** be a valid VkPipelineBindPoint value
- VUID-VkGeneratedCommandsMemoryRequirementsInfoNV-pipeline-parameter
  If `pipeline` is not VK_NULL_HANDLE, `pipeline` **must** be a valid VkPipeline handle
- VUID-VkGeneratedCommandsMemoryRequirementsInfoNV-indirectCommandsLayout-parameter
  `indirectCommandsLayout` **must** be a valid VkIndirectCommandsLayoutNV handle
- VUID-VkGeneratedCommandsMemoryRequirementsInfoNV-commonparent
  Both of `indirectCommandsLayout`, and `pipeline` that are valid handles of non-ignored parameters **must** have been created, allocated, or retrieved from the same VkDevice

To bind a compute pipeline in Device-Generated Commands, an application **must** query the pipeline's device address.

To query a compute pipeline's 64-bit device address, call:

```
// Provided by VK_NV_device_generated_commands_compute
VkDeviceAddress vkGetPipelineIndirectDeviceAddressNV(
    VkDevice                                    device,
    const VkPipelineIndirectDeviceAddressInfoNV* pInfo);
```

- `device` is the logical device on which the pipeline was created.
- `pInfo` is a pointer to a VkPipelineIndirectDeviceAddressInfoNV structure specifying the pipeline to retrieve the address for.

### Valid Usage

- VUID-vkGetPipelineIndirectDeviceAddressNV-deviceGeneratedComputePipelines-09078
  The `VkPhysicalDeviceDeviceGeneratedCommandsComputeFeaturesNV::deviceGeneratedComputePipelines` feature **must** be enabled

### Valid Usage (Implicit)

- VUID-vkGetPipelineIndirectDeviceAddressNV-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkGetPipelineIndirectDeviceAddressNV-pInfo-parameter
  `pInfo` **must** be a valid pointer to a valid VkPipelineIndirectDeviceAddressInfoNV structure

The `VkPipelineIndirectDeviceAddressInfoNV` structure is defined as:

```
// Provided by VK_NV_device_generated_commands_compute
typedef struct VkPipelineIndirectDeviceAddressInfoNV {
    VkStructureType        sType;
    const void*            pNext;
    VkPipelineBindPoint    pipelineBindPoint;
    VkPipeline             pipeline;
} VkPipelineIndirectDeviceAddressInfoNV;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `pipelineBindPoint` is a VkPipelineBindPoint value specifying the type of pipeline whose device address is being queried.
- `pipeline` specifies the pipeline whose device address is being queried.

### Valid Usage

- VUID-VkPipelineIndirectDeviceAddressInfoNV-pipelineBindPoint-09079
  The provided `pipelineBindPoint` **must** be of type `VK_PIPELINE_BIND_POINT_COMPUTE`
- VUID-VkPipelineIndirectDeviceAddressInfoNV-pipeline-09080
  `pipeline` **must** have been created with flag `VK_PIPELINE_CREATE_INDIRECT_BINDABLE_BIT_NV` set
- VUID-VkPipelineIndirectDeviceAddressInfoNV-pipeline-09081
  `pipeline` **must** have been created with a VkComputePipelineIndirectBufferInfoNV structure specifying a valid address where its metadata will be saved

### Valid Usage (Implicit)

- VUID-VkPipelineIndirectDeviceAddressInfoNV-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PIPELINE_INDIRECT_DEVICE_ADDRESS_INFO_NV`
- VUID-VkPipelineIndirectDeviceAddressInfoNV-pNext-pNext
  `pNext` **must** be `NULL`
- VUID-VkPipelineIndirectDeviceAddressInfoNV-pipelineBindPoint-parameter
  `pipelineBindPoint` **must** be a valid VkPipelineBindPoint value
- VUID-VkPipelineIndirectDeviceAddressInfoNV-pipeline-parameter
  `pipeline` **must** be a valid VkPipeline handle

To determine the memory requirements for a compute pipeline's metadata, call:

```
// Provided by VK_NV_device_generated_commands_compute
void vkGetPipelineIndirectMemoryRequirementsNV(
    VkDevice                                    device,
    const VkComputePipelineCreateInfo*          pCreateInfo,
    VkMemoryRequirements2*                      pMemoryRequirements);
```

- `device` is the logical device that owns the buffer.
- `pCreateInfo` is a VkComputePipelineCreateInfo structure specifying the creation parameters of the compute pipeline whose memory requirements are being queried.
- `pMemoryRequirements` is a pointer to a VkMemoryRequirements2 structure in which the requested pipeline's memory requirements are returned.

If `pCreateInfo->pNext` chain includes a pointer to a VkComputePipelineIndirectBufferInfoNV structure, then the contents of that structure are ignored.

### Valid Usage

- VUID-vkGetPipelineIndirectMemoryRequirementsNV-deviceGeneratedComputePipelines-09082
  The `VkPhysicalDeviceDeviceGeneratedCommandsComputeFeaturesNV::deviceGeneratedComputePipelines` feature **must** be enabled
- VUID-vkGetPipelineIndirectMemoryRequirementsNV-pCreateInfo-09083
  `pCreateInfo->flags` **must** include `VK_PIPELINE_CREATE_INDIRECT_BINDABLE_BIT_NV`

### Valid Usage (Implicit)

- VUID-vkGetPipelineIndirectMemoryRequirementsNV-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkGetPipelineIndirectMemoryRequirementsNV-pCreateInfo-parameter
  `pCreateInfo` **must** be a valid pointer to a valid VkComputePipelineCreateInfo structure
- VUID-vkGetPipelineIndirectMemoryRequirementsNV-pMemoryRequirements-parameter
  `pMemoryRequirements` **must** be a valid pointer to a VkMemoryRequirements2 structure

The actual generation of commands as well as their execution on the device is handled as single action with:

```
// Provided by VK_NV_device_generated_commands
void vkCmdExecuteGeneratedCommandsNV(
    VkCommandBuffer                             commandBuffer,
    VkBool32                                    isPreprocessed,
    const VkGeneratedCommandsInfoNV*            pGeneratedCommandsInfo);
```

- `commandBuffer` is the command buffer into which the command is recorded.

- isPreprocessed represents whether the input data has already been preprocessed on the device. If it is VK_FALSE this command will implicitly trigger the preprocessing step, otherwise not.
- pGeneratedCommandsInfo is a pointer to a VkGeneratedCommandsInfoNV structure containing parameters affecting the generation of commands.

If the VK_INDIRECT_COMMANDS_LAYOUT_USAGE_UNORDERED_SEQUENCES_BIT_NV flag was used to create the VkGeneratedCommandsInfoNV::indirectCommandsLayout then the order of execution of individual draws through this command **may** execute in any order, and **may** not necessarily be in the same order as specified in VkGeneratedCommandsInfoNV::pStreams.

The order of execution of individual dispatches through this command **may** execute in any order and **may** not necessarily be in the same order as specified in VkGeneratedCommandsInfoNV::pStreams.

## Valid Usage

- VUID-vkCmdExecuteGeneratedCommandsNV-magFilter-04553
  If a VkSampler created with magFilter or minFilter equal to VK_FILTER_LINEAR, reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE, and compareEnable equal to VK_FALSE is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT
- VUID-vkCmdExecuteGeneratedCommandsNV-magFilter-09598
  If a VkSampler created with magFilter or minFilter equal to VK_FILTER_LINEAR and reductionMode equal to either VK_SAMPLER_REDUCTION_MODE_MIN or VK_SAMPLER_REDUCTION_MODE_MAX is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT
- VUID-vkCmdExecuteGeneratedCommandsNV-mipmapMode-04770
  If a VkSampler created with mipmapMode equal to VK_SAMPLER_MIPMAP_MODE_LINEAR, reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE, and compareEnable equal to VK_FALSE is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT
- VUID-vkCmdExecuteGeneratedCommandsNV-mipmapMode-09599
  If a VkSampler created with mipmapMode equal to VK_SAMPLER_MIPMAP_MODE_LINEAR and reductionMode equal to either VK_SAMPLER_REDUCTION_MODE_MIN or VK_SAMPLER_REDUCTION_MODE_MAX is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT
- VUID-vkCmdExecuteGeneratedCommandsNV-unnormalizedCoordinates-09635
  If a VkSampler created with unnormalizedCoordinates equal to VK_TRUE is used to sample a VkImageView as a result of this command, then the image view's levelCount and layerCount **must** be 1
- VUID-vkCmdExecuteGeneratedCommandsNV-unnormalizedCoordinates-09636

If a VkSampler created with unnormalizedCoordinates equal to VK_TRUE is used to sample a VkImageView as a result of this command, then the image view's viewType **must** be VK_IMAGE_VIEW_TYPE_1D or VK_IMAGE_VIEW_TYPE_2D

- VUID-vkCmdExecuteGeneratedCommandsNV-None-06479
  If a VkImageView is sampled with depth comparison, the image view's format features **must** contain VK_FORMAT_FEATURE_2_SAMPLED_IMAGE_DEPTH_COMPARISON_BIT
- VUID-vkCmdExecuteGeneratedCommandsNV-None-02691
  If a VkImageView is accessed using atomic operations as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_STORAGE_IMAGE_ATOMIC_BIT
- VUID-vkCmdExecuteGeneratedCommandsNV-None-07888
  If a VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER descriptor is accessed using atomic operations as a result of this command, then the storage texel buffer's format features **must** contain VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_ATOMIC_BIT
- VUID-vkCmdExecuteGeneratedCommandsNV-None-02692
  If a VkImageView is sampled with VK_FILTER_CUBIC_EXT as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_CUBIC_BIT_EXT
- VUID-vkCmdExecuteGeneratedCommandsNV-None-02693
  If the VK_EXT_filter_cubic extension is not enabled and any VkImageView is sampled with VK_FILTER_CUBIC_EXT as a result of this command, it **must** not have a VkImageViewType of VK_IMAGE_VIEW_TYPE_3D, VK_IMAGE_VIEW_TYPE_CUBE, or VK_IMAGE_VIEW_TYPE_CUBE_ARRAY
- VUID-vkCmdExecuteGeneratedCommandsNV-filterCubic-02694
  Any VkImageView being sampled with VK_FILTER_CUBIC_EXT as a result of this command **must** have a VkImageViewType and format that supports cubic filtering, as specified by VkFilterCubicImageViewImageFormatPropertiesEXT::filterCubic returned by vkGetPhysicalDeviceImageFormatProperties2
- VUID-vkCmdExecuteGeneratedCommandsNV-filterCubicMinmax-02695
  Any VkImageView being sampled with VK_FILTER_CUBIC_EXT with a reduction mode of either VK_SAMPLER_REDUCTION_MODE_MIN or VK_SAMPLER_REDUCTION_MODE_MAX as a result of this command **must** have a VkImageViewType and format that supports cubic filtering together with minmax filtering, as specified by VkFilterCubicImageViewImageFormatPropertiesEXT::filterCubicMinmax returned by vkGetPhysicalDeviceImageFormatProperties2
- VUID-vkCmdExecuteGeneratedCommandsNV-cubicRangeClamp-09212
  If the cubicRangeClamp feature is not enabled, then any VkImageView being sampled with VK_FILTER_CUBIC_EXT as a result of this command **must** not have a VkSamplerReductionModeCreateInfo::reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE_RANGECLAMP_QCOM
- VUID-vkCmdExecuteGeneratedCommandsNV-reductionMode-09213
  Any VkImageView being sampled with a VkSamplerReductionModeCreateInfo::reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE_RANGECLAMP_QCOM as a result of this command **must** sample with VK_FILTER_CUBIC_EXT
- VUID-vkCmdExecuteGeneratedCommandsNV-selectableCubicWeights-09214

If the selectableCubicWeights feature is not enabled, then any VkImageView being sampled with VK_FILTER_CUBIC_EXT as a result of this command **must** have VkSamplerCubicWeightsCreateInfoQCOM::cubicWeights equal to VK_CUBIC_FILTER_WEIGHTS_CATMULL_ROM_QCOM

- VUID-vkCmdExecuteGeneratedCommandsNV-flags-02696
  Any VkImage created with a VkImageCreateInfo::flags containing VK_IMAGE_CREATE_CORNER_SAMPLED_BIT_NV sampled as a result of this command **must** only be sampled using a VkSamplerAddressMode of VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_EDGE
- VUID-vkCmdExecuteGeneratedCommandsNV-OpTypeImage-07027
  For any VkImageView being written as a storage image where the image format field of the OpTypeImage is Unknown, the view's format features **must** contain VK_FORMAT_FEATURE_2_STORAGE_WRITE_WITHOUT_FORMAT_BIT
- VUID-vkCmdExecuteGeneratedCommandsNV-OpTypeImage-07028
  For any VkImageView being read as a storage image where the image format field of the OpTypeImage is Unknown, the view's format features **must** contain VK_FORMAT_FEATURE_2_STORAGE_READ_WITHOUT_FORMAT_BIT
- VUID-vkCmdExecuteGeneratedCommandsNV-OpTypeImage-07029
  For any VkBufferView being written as a storage texel buffer where the image format field of the OpTypeImage is Unknown, the view's buffer features **must** contain VK_FORMAT_FEATURE_2_STORAGE_WRITE_WITHOUT_FORMAT_BIT
- VUID-vkCmdExecuteGeneratedCommandsNV-OpTypeImage-07030
  Any VkBufferView being read as a storage texel buffer where the image format field of the OpTypeImage is Unknown then the view's buffer features **must** contain VK_FORMAT_FEATURE_2_STORAGE_READ_WITHOUT_FORMAT_BIT
- VUID-vkCmdExecuteGeneratedCommandsNV-None-08600
  For each set *n* that is statically used by a bound shader, a descriptor set **must** have been bound to *n* at the same pipeline bind point, with a VkPipelineLayout that is compatible for set *n*, with the VkPipelineLayout used to create the current VkPipeline or the VkDescriptorSetLayout array used to create the current VkShaderEXT , as described in Pipeline Layout Compatibility
- VUID-vkCmdExecuteGeneratedCommandsNV-None-08601
  For each push constant that is statically used by a bound shader, a push constant value **must** have been set for the same pipeline bind point, with a VkPipelineLayout that is compatible for push constants, with the VkPipelineLayout used to create the current VkPipeline or the VkDescriptorSetLayout array used to create the current VkShaderEXT , as described in Pipeline Layout Compatibility
- VUID-vkCmdExecuteGeneratedCommandsNV-None-10068
  For each array of resources that is used by a bound shader, the indices used to access members of the array **must** be less than the descriptor count for the identified binding in the descriptor sets used by this command
- VUID-vkCmdExecuteGeneratedCommandsNV-maintenance4-08602
  If the maintenance4 feature is not enabled, then for each push constant that is statically used by a bound shader, a push constant value **must** have been set for the same pipeline bind point, with a VkPipelineLayout that is compatible for push constants, with the

VkPipelineLayout used to create the current VkPipeline or the VkDescriptorSetLayout and VkPushConstantRange arrays used to create the current VkShaderEXT , as described in Pipeline Layout Compatibility

- VUID-vkCmdExecuteGeneratedCommandsNV-None-08114
  Descriptors in each bound descriptor set, specified via vkCmdBindDescriptorSets, **must** be valid as described by descriptor validity if they are statically used by the VkPipeline bound to the pipeline bind point used by this command and the bound VkPipeline was not created with `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`
- VUID-vkCmdExecuteGeneratedCommandsNV-None-08115
  If the descriptors used by the VkPipeline bound to the pipeline bind point were specified via vkCmdBindDescriptorSets, the bound VkPipeline **must** have been created without `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`
- VUID-vkCmdExecuteGeneratedCommandsNV-None-08116
  Descriptors in bound descriptor buffers, specified via vkCmdSetDescriptorBufferOffsetsEXT, **must** be valid if they are dynamically used by the VkPipeline bound to the pipeline bind point used by this command and the bound VkPipeline was created with `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`
- VUID-vkCmdExecuteGeneratedCommandsNV-None-08604
  Descriptors in bound descriptor buffers, specified via vkCmdSetDescriptorBufferOffsetsEXT, **must** be valid if they are dynamically used by any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command
- VUID-vkCmdExecuteGeneratedCommandsNV-None-08117
  If the descriptors used by the VkPipeline bound to the pipeline bind point were specified via vkCmdSetDescriptorBufferOffsetsEXT, the bound VkPipeline **must** have been created with `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`
- VUID-vkCmdExecuteGeneratedCommandsNV-None-08119
  If a descriptor is dynamically used with a VkPipeline created with `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`, the descriptor memory **must** be resident
- VUID-vkCmdExecuteGeneratedCommandsNV-None-08605
  If a descriptor is dynamically used with a VkShaderEXT created with a `VkDescriptorSetLayout` that was created with `VK_DESCRIPTOR_SET_LAYOUT_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`, the descriptor memory **must** be resident
- VUID-vkCmdExecuteGeneratedCommandsNV-None-08606
  If the `shaderObject` feature is not enabled, a valid pipeline **must** be bound to the pipeline bind point used by this command
- VUID-vkCmdExecuteGeneratedCommandsNV-None-08608
  If a pipeline is bound to the pipeline bind point used by this command, there **must** not have been any calls to dynamic state setting commands for any state specified statically in the VkPipeline object bound to the pipeline bind point used by this command, since that pipeline was bound
- VUID-vkCmdExecuteGeneratedCommandsNV-None-08609
  If the VkPipeline object bound to the pipeline bind point used by this command or any

VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a VkSampler object that uses unnormalized coordinates, that sampler **must** not be used to sample from any VkImage with a VkImageView of the type VK_IMAGE_VIEW_TYPE_3D, VK_IMAGE_VIEW_TYPE_CUBE, VK_IMAGE_VIEW_TYPE_1D_ARRAY, VK_IMAGE_VIEW_TYPE_2D_ARRAY or VK_IMAGE_VIEW_TYPE_CUBE_ARRAY, in any shader stage

- VUID-vkCmdExecuteGeneratedCommandsNV-None-08610
  If the VkPipeline object bound to the pipeline bind point used by this command or any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a VkSampler object that uses unnormalized coordinates, that sampler **must** not be used with any of the SPIR-V OpImageSample* or OpImageSparseSample* instructions with ImplicitLod, Dref or Proj in their name, in any shader stage
- VUID-vkCmdExecuteGeneratedCommandsNV-None-08611
  If the VkPipeline object bound to the pipeline bind point used by this command or any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a VkSampler object that uses unnormalized coordinates, that sampler **must** not be used with any of the SPIR-V OpImageSample* or OpImageSparseSample* instructions that includes a LOD bias or any offset values, in any shader stage
- VUID-vkCmdExecuteGeneratedCommandsNV-None-08607
  If the shaderObject is enabled, either a valid pipeline **must** be bound to the pipeline bind point used by this command, or a valid combination of valid and VK_NULL_HANDLE shader objects **must** be bound to every supported shader stage corresponding to the pipeline bind point used by this command
- VUID-vkCmdExecuteGeneratedCommandsNV-uniformBuffers-06935
  If any stage of the VkPipeline object bound to the pipeline bind point used by this command accesses a uniform buffer, and that stage was created without enabling either VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_EXT or VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_2_EXT for uniformBuffers, and the robustBufferAccess feature is not enabled, that stage **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point
- VUID-vkCmdExecuteGeneratedCommandsNV-None-08612
  If the robustBufferAccess feature is not enabled, and any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a uniform buffer, it **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point
- VUID-vkCmdExecuteGeneratedCommandsNV-storageBuffers-06936
  If any stage of the VkPipeline object bound to the pipeline bind point used by this command accesses a storage buffer, and that stage was created without enabling either VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_EXT or VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_2_EXT for storageBuffers, and the robustBufferAccess feature is not enabled, that stage **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point
- VUID-vkCmdExecuteGeneratedCommandsNV-None-08613
  If the robustBufferAccess feature is not enabled, and any VkShaderEXT bound to a stage

corresponding to the pipeline bind point used by this command accesses a storage buffer, it **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point

- VUID-vkCmdExecuteGeneratedCommandsNV-commandBuffer-02707
  If `commandBuffer` is an unprotected command buffer and `protectedNoFault` is not supported, any resource accessed by bound shaders **must** not be a protected resource
- VUID-vkCmdExecuteGeneratedCommandsNV-None-06550
  If a bound shader accesses a VkSampler or VkImageView object that enables sampler $Y'C_BC_R$ conversion, that object **must** only be used with `OpImageSample*` or `OpImageSparseSample*` instructions
- VUID-vkCmdExecuteGeneratedCommandsNV-ConstOffset-06551
  If a bound shader accesses a VkSampler or VkImageView object that enables sampler $Y'C_BC_R$ conversion, that object **must** not use the `ConstOffset` and `Offset` operands
- VUID-vkCmdExecuteGeneratedCommandsNV-viewType-07752
  If a VkImageView is accessed as a result of this command, then the image view's `viewType` **must** match the `Dim` operand of the `OpTypeImage` as described in Compatibility Between SPIR-V Image Dimensions and Vulkan ImageView Types
- VUID-vkCmdExecuteGeneratedCommandsNV-format-07753
  If a VkImageView is accessed as a result of this command, then the numeric type of the image view's `format` and the `Sampled Type` operand of the `OpTypeImage` **must** match
- VUID-vkCmdExecuteGeneratedCommandsNV-OpImageWrite-08795
  If a VkImageView created with a format other than `VK_FORMAT_A8_UNORM_KHR` is accessed using `OpImageWrite` as a result of this command, then the `Type` of the `Texel` operand of that instruction **must** have at least as many components as the image view's format
- VUID-vkCmdExecuteGeneratedCommandsNV-OpImageWrite-08796
  If a VkImageView created with the format `VK_FORMAT_A8_UNORM_KHR` is accessed using `OpImageWrite` as a result of this command, then the `Type` of the `Texel` operand of that instruction **must** have four components
- VUID-vkCmdExecuteGeneratedCommandsNV-OpImageWrite-04469
  If a VkBufferView is accessed using `OpImageWrite` as a result of this command, then the `Type` of the `Texel` operand of that instruction **must** have at least as many components as the buffer view's format
- VUID-vkCmdExecuteGeneratedCommandsNV-SampledType-04470
  If a VkImageView with a VkFormat that has a 64-bit component width is accessed as a result of this command, the `SampledType` of the `OpTypeImage` operand of that instruction **must** have a `Width` of 64
- VUID-vkCmdExecuteGeneratedCommandsNV-SampledType-04471
  If a VkImageView with a VkFormat that has a component width less than 64-bit is accessed as a result of this command, the `SampledType` of the `OpTypeImage` operand of that instruction **must** have a `Width` of 32
- VUID-vkCmdExecuteGeneratedCommandsNV-SampledType-04472
  If a VkBufferView with a VkFormat that has a 64-bit component width is accessed as a result of this command, the `SampledType` of the `OpTypeImage` operand of that instruction

**must** have a `Width` of 64

- VUID-vkCmdExecuteGeneratedCommandsNV-SampledType-04473
  If a VkBufferView with a VkFormat that has a component width less than 64-bit is accessed as a result of this command, the `SampledType` of the `OpTypeImage` operand of that instruction **must** have a `Width` of 32
- VUID-vkCmdExecuteGeneratedCommandsNV-sparseImageInt64Atomics-04474
  If the `sparseImageInt64Atomics` feature is not enabled, VkImage objects created with the `VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT` flag **must** not be accessed by atomic instructions through an `OpTypeImage` with a `SampledType` with a `Width` of 64 by this command
- VUID-vkCmdExecuteGeneratedCommandsNV-sparseImageInt64Atomics-04475
  If the `sparseImageInt64Atomics` feature is not enabled, VkBuffer objects created with the `VK_BUFFER_CREATE_SPARSE_RESIDENCY_BIT` flag **must** not be accessed by atomic instructions through an `OpTypeImage` with a `SampledType` with a `Width` of 64 by this command
- VUID-vkCmdExecuteGeneratedCommandsNV-OpImageWeightedSampleQCOM-06971
  If `OpImageWeightedSampleQCOM` is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_2_WEIGHT_SAMPLED_IMAGE_BIT_QCOM`
- VUID-vkCmdExecuteGeneratedCommandsNV-OpImageWeightedSampleQCOM-06972
  If `OpImageWeightedSampleQCOM` uses a VkImageView as a sample weight image as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_2_WEIGHT_IMAGE_BIT_QCOM`
- VUID-vkCmdExecuteGeneratedCommandsNV-OpImageBoxFilterQCOM-06973
  If `OpImageBoxFilterQCOM` is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_2_BOX_FILTER_SAMPLED_BIT_QCOM`
- VUID-vkCmdExecuteGeneratedCommandsNV-OpImageBlockMatchSSDQCOM-06974
  If `OpImageBlockMatchSSDQCOM` is used to read from an VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM`
- VUID-vkCmdExecuteGeneratedCommandsNV-OpImageBlockMatchSADQCOM-06975
  If `OpImageBlockMatchSADQCOM` is used to read from an VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM`
- VUID-vkCmdExecuteGeneratedCommandsNV-OpImageBlockMatchSADQCOM-06976
  If `OpImageBlockMatchSADQCOM` or OpImageBlockMatchSSDQCOM is used to read from a reference image as result of this command, then the specified reference coordinates **must** not fail integer texel coordinate validation
- VUID-vkCmdExecuteGeneratedCommandsNV-OpImageWeightedSampleQCOM-06977
  If `OpImageWeightedSampleQCOM`, `OpImageBoxFilterQCOM`, `OpImageBlockMatchWindowSSDQCOM`, `OpImageBlockMatchWindowSADQCOM`, `OpImageBlockMatchGatherSSDQCOM`, `OpImageBlockMatchGatherSADQCOM`, `OpImageBlockMatchSSDQCOM`, or `OpImageBlockMatchSADQCOM` uses a VkSampler as a result of this command, then the sampler **must** have been created with `VK_SAMPLER_CREATE_IMAGE_PROCESSING_BIT_QCOM`

- VUID-vkCmdExecuteGeneratedCommandsNV-OpImageWeightedSampleQCOM-06978
  If any command other than `OpImageWeightedSampleQCOM`, `OpImageBoxFilterQCOM`, `OpImageBlockMatchWindowSSDQCOM`, `OpImageBlockMatchWindowSADQCOM`, `OpImageBlockMatchGatherSSDQCOM`, `OpImageBlockMatchGatherSADQCOM`, `OpImageBlockMatchSSDQCOM`, or `OpImageBlockMatchSADQCOM` uses a VkSampler as a result of this command, then the sampler **must** not have been created with `VK_SAMPLER_CREATE_IMAGE_PROCESSING_BIT_QCOM`
- VUID-vkCmdExecuteGeneratedCommandsNV-OpImageBlockMatchWindow-09215
  If a `OpImageBlockMatchWindow*QCOM` or `OpImageBlockMatchGather*QCOM` instruction is used to read from an VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM`
- VUID-vkCmdExecuteGeneratedCommandsNV-OpImageBlockMatchWindow-09216
  If a `OpImageBlockMatchWindow*QCOM` or `OpImageBlockMatchGather*QCOM` instruction is used to read from an VkImageView as a result of this command, then the image view's format **must** be a single-component format
- VUID-vkCmdExecuteGeneratedCommandsNV-OpImageBlockMatchWindow-09217
  If a `OpImageBlockMatchWindow*QCOM` or `OpImageBlockMatchGather*QCOM` read from a reference image as result of this command, then the specified reference coordinates **must** not fail integer texel coordinate validation
- VUID-vkCmdExecuteGeneratedCommandsNV-None-07288
  Any shader invocation executed by this command **must** terminate
- VUID-vkCmdExecuteGeneratedCommandsNV-None-09600
  If a descriptor with type equal to any of `VK_DESCRIPTOR_TYPE_SAMPLE_WEIGHT_IMAGE_QCOM`, `VK_DESCRIPTOR_TYPE_BLOCK_MATCH_IMAGE_QCOM`, `VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE`, `VK_DESCRIPTOR_TYPE_STORAGE_IMAGE`, or `VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT` is accessed as a result of this command, the image subresource identified by that descriptor **must** be in the image layout identified when the descriptor was written
- VUID-vkCmdExecuteGeneratedCommandsNV-renderPass-02684
  The current render pass **must** be compatible with the `renderPass` member of the `VkGraphicsPipelineCreateInfo` structure specified when creating the `VkPipeline` bound to `VK_PIPELINE_BIND_POINT_GRAPHICS`
- VUID-vkCmdExecuteGeneratedCommandsNV-subpass-02685
  The subpass index of the current render pass **must** be equal to the `subpass` member of the `VkGraphicsPipelineCreateInfo` structure specified when creating the `VkPipeline` bound to `VK_PIPELINE_BIND_POINT_GRAPHICS`
- VUID-vkCmdExecuteGeneratedCommandsNV-None-07748
  If any shader statically accesses an input attachment, a valid descriptor **must** be bound to the pipeline via a descriptor set
- VUID-vkCmdExecuteGeneratedCommandsNV-OpTypeImage-07468
  If any shader executed by this pipeline accesses an `OpTypeImage` variable with a `Dim` operand of `SubpassData`, it **must** be decorated with an `InputAttachmentIndex` that corresponds to a valid input attachment in the current subpass
- VUID-vkCmdExecuteGeneratedCommandsNV-None-07469
  Input attachment views accessed in a subpass **must** be created with the same VkFormat

as the corresponding subpass definition, and be created with a VkImageView that is compatible with the attachment referenced by the subpass' pInputAttachments [InputAttachmentIndex] in the currently bound VkFramebuffer as specified by Fragment Input Attachment Compatibility

- VUID-vkCmdExecuteGeneratedCommandsNV-pDepthInputAttachmentIndex-09595
  Input attachment views accessed in a dynamic render pass with a InputAttachmentIndex referenced by VkRenderingInputAttachmentIndexInfoKHR, or no InputAttachmentIndex if VkRenderingInputAttachmentIndexInfoKHR:pDepthInputAttachmentIndex or VkRenderingInputAttachmentIndexInfoKHR:pStencilInputAttachmentIndex are NULL, **must** be created with a VkImageView that is compatible with the corresponding color, depth, or stencil attachment in VkRenderingInfo
- VUID-vkCmdExecuteGeneratedCommandsNV-pDepthInputAttachmentIndex-09596
  Input attachment views accessed in a dynamic render pass via a shader object **must** have an InputAttachmentIndex if both VkRenderingInputAttachmentIndexInfoKHR:pDepthInputAttachmentIndex and VkRenderingInputAttachmentIndexInfoKHR:pStencilInputAttachmentIndex are non-NULL
- VUID-vkCmdExecuteGeneratedCommandsNV-InputAttachmentIndex-09597
  If an input attachment view accessed in a dynamic render pass via a shader object has an InputAttachmentIndex, the InputAttachmentIndex **must** match an index in VkRenderingInputAttachmentIndexInfoKHR
- VUID-vkCmdExecuteGeneratedCommandsNV-None-06537
  Memory backing image subresources used as attachments in the current render pass **must** not be written in any way other than as an attachment by this command
- VUID-vkCmdExecuteGeneratedCommandsNV-None-09000
  If a color attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass, it is not in the VK_IMAGE_LAYOUT_ATTACHMENT_FEEDBACK_LOOP_OPTIMAL_EXT image layout, and either:
  - the VK_PIPELINE_CREATE_COLOR_ATTACHMENT_FEEDBACK_LOOP_BIT_EXT is set on the currently bound pipeline or
  - the last call to vkCmdSetAttachmentFeedbackLoopEnableEXT included VK_IMAGE_ASPECT_COLOR_BIT and
    - there is no currently bound graphics pipeline or
    - the currently bound graphics pipeline was created with VK_DYNAMIC_STATE_ATTACHMENT_FEEDBACK_LOOP_ENABLE_EXT

  it **must** not be accessed in any way other than as an attachment by this command
- VUID-vkCmdExecuteGeneratedCommandsNV-None-09001
  If a depth attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass, it is not in the VK_IMAGE_LAYOUT_ATTACHMENT_FEEDBACK_LOOP_OPTIMAL_EXT image layout, and either:
  - the VK_PIPELINE_CREATE_DEPTH_STENCIL_ATTACHMENT_FEEDBACK_LOOP_BIT_EXT is set on the currently bound pipeline or

- the last call to vkCmdSetAttachmentFeedbackLoopEnableEXT included VK_IMAGE_ASPECT_DEPTH_BIT and
  - there is no currently bound graphics pipeline or
  - the currently bound graphics pipeline was created with VK_DYNAMIC_STATE_ATTACHMENT_FEEDBACK_LOOP_ENABLE_EXT

it **must** not be accessed in any way other than as an attachment by this command

- VUID-vkCmdExecuteGeneratedCommandsNV-None-09002
  If a stencil attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass, it is not in the VK_IMAGE_LAYOUT_ATTACHMENT_FEEDBACK_LOOP_OPTIMAL_EXT image layout, and either:
  - the VK_PIPELINE_CREATE_DEPTH_STENCIL_ATTACHMENT_FEEDBACK_LOOP_BIT_EXT is set on the currently bound pipeline or
  - the last call to vkCmdSetAttachmentFeedbackLoopEnableEXT included VK_IMAGE_ASPECT_STENCIL_BIT and
    - there is no currently bound graphics pipeline or
    - the currently bound graphics pipeline was created with VK_DYNAMIC_STATE_ATTACHMENT_FEEDBACK_LOOP_ENABLE_EXT

  it **must** not be accessed in any way other than as an attachment by this command

- VUID-vkCmdExecuteGeneratedCommandsNV-None-09003
  If an attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass, it **must** not be accessed in any way other than as an attachment, storage image, or sampled image by this command
- VUID-vkCmdExecuteGeneratedCommandsNV-None-06539
  If any previously recorded command in the current subpass accessed an image subresource used as an attachment in this subpass in any way other than as an attachment, this command **must** not write to that image subresource as an attachment
- VUID-vkCmdExecuteGeneratedCommandsNV-None-06886
  If the current render pass instance uses a depth/stencil attachment with a read-only layout for the depth aspect, depth writes **must** be disabled
- VUID-vkCmdExecuteGeneratedCommandsNV-None-06887
  If the current render pass instance uses a depth/stencil attachment with a read-only layout for the stencil aspect, both front and back writeMask are not zero, and stencil test is enabled, all stencil ops **must** be VK_STENCIL_OP_KEEP
- VUID-vkCmdExecuteGeneratedCommandsNV-None-07831
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT dynamic state enabled then vkCmdSetViewport **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdExecuteGeneratedCommandsNV-None-07832
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SCISSOR dynamic state enabled then vkCmdSetScissor **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdExecuteGeneratedCommandsNV-None-07833
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_WIDTH dynamic state enabled then vkCmdSetLineWidth **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdExecuteGeneratedCommandsNV-None-08617
  If a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetPolygonModeEXT in the current command buffer set polygonMode to VK_POLYGON_MODE_LINE, vkCmdSetLineWidth **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdExecuteGeneratedCommandsNV-None-08618
  If a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetPrimitiveTopologyEXT in the current command buffer set primitiveTopology to any line topology, vkCmdSetLineWidth **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdExecuteGeneratedCommandsNV-None-08619
  If a shader object that outputs line primitives is bound to the VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT or VK_SHADER_STAGE_GEOMETRY_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, vkCmdSetLineWidth **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdExecuteGeneratedCommandsNV-None-07834
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_BIAS dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of depthBiasEnable is VK_TRUE, then vkCmdSetDepthBounds or vkCmdSetDepthBias2EXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdExecuteGeneratedCommandsNV-None-07835
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_BLEND_CONSTANTS dynamic state enabled then vkCmdSetBlendConstants **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdExecuteGeneratedCommandsNV-None-08621
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetColorBlendEnableEXT in the current command buffer set any element of pColorBlendEnables to VK_TRUE, and the most recent call to vkCmdSetColorBlendEquationEXT in the current command buffer set the same element of pColorBlendEquations to a VkColorBlendEquationEXT structure with any VkBlendFactor

member with a value of VK_BLEND_FACTOR_CONSTANT_COLOR, VK_BLEND_FACTOR_ONE_MINUS_CONSTANT_COLOR, VK_BLEND_FACTOR_CONSTANT_ALPHA, or VK_BLEND_FACTOR_ONE_MINUS_CONSTANT_ALPHA, vkCmdSetBlendConstants **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdExecuteGeneratedCommandsNV-None-07836
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_BOUNDS dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of depthBoundsTestEnable is VK_TRUE, then vkCmdSetDepthBounds **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdExecuteGeneratedCommandsNV-None-07837
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_STENCIL_COMPARE_MASK dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of stencilTestEnable is VK_TRUE, then vkCmdSetStencilCompareMask **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdExecuteGeneratedCommandsNV-None-07838
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_STENCIL_WRITE_MASK dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of stencilTestEnable is VK_TRUE, then vkCmdSetStencilWriteMask **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdExecuteGeneratedCommandsNV-None-07839
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_STENCIL_REFERENCE dynamic state enabled, the current value of and rasterizerDiscardEnable is VK_FALSE, the current value of stencilTestEnable is VK_TRUE, then vkCmdSetStencilReference **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdExecuteGeneratedCommandsNV-maxMultiviewInstanceIndex-02688
  If the draw is recorded in a render pass instance with multiview enabled, the maximum instance index **must** be less than or equal to VkPhysicalDeviceMultiviewProperties::maxMultiviewInstanceIndex
- VUID-vkCmdExecuteGeneratedCommandsNV-sampleLocationsEnable-02689
  If the bound graphics pipeline was created with VkPipelineSampleLocationsStateCreateInfoEXT::sampleLocationsEnable set to VK_TRUE and the current subpass has a depth/stencil attachment, then that attachment **must** have been created with the VK_IMAGE_CREATE_SAMPLE_LOCATIONS_COMPATIBLE_DEPTH_BIT_EXT bit set
- VUID-vkCmdExecuteGeneratedCommandsNV-None-06666
  If the VK_EXT_sample_locations extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the

VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of sampleLocationsEnable is VK_TRUE, then vkCmdSetSampleLocationsEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdExecuteGeneratedCommandsNV-None-07840
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_CULL_MODE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetCullModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdExecuteGeneratedCommandsNV-None-07841
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_FRONT_FACE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetFrontFaceEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdExecuteGeneratedCommandsNV-None-07843
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_TEST_ENABLE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, vkCmdSetDepthTestEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdExecuteGeneratedCommandsNV-None-07844
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_WRITE_ENABLE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetDepthWriteEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdExecuteGeneratedCommandsNV-None-07845
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_COMPARE_OP dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of depthTestEnable is VK_TRUE, then vkCmdSetDepthCompareOpEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdExecuteGeneratedCommandsNV-None-07846
  If the depthBounds feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_BOUNDS_TEST_ENABLE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetDepthBoundsTestEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdExecuteGeneratedCommandsNV-None-07847
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_STENCIL_TEST_ENABLE dynamic state enabled, and

the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetStencilTestEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdExecuteGeneratedCommandsNV-None-07848
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_STENCIL_OP dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, the current value of stencilTestEnable is VK_TRUE, then vkCmdSetStencilOpEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdExecuteGeneratedCommandsNV-viewportCount-03417
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, but not the VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT dynamic state enabled, then vkCmdSetViewportWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the viewportCount parameter of vkCmdSetViewportWithCountEXT **must** match the VkPipelineViewportStateCreateInfo::scissorCount of the pipeline

- VUID-vkCmdExecuteGeneratedCommandsNV-scissorCount-03418
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT dynamic state enabled, but not the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, then vkCmdSetScissorWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the scissorCount parameter of vkCmdSetScissorWithCountEXT **must** match the VkPipelineViewportStateCreateInfo::viewportCount of the pipeline

- VUID-vkCmdExecuteGeneratedCommandsNV-viewportCount-03419
  If the bound graphics pipeline state was created with both the VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT and VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic states enabled then both vkCmdSetViewportWithCountEXT and vkCmdSetScissorWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the viewportCount parameter of vkCmdSetViewportWithCountEXT **must** match the scissorCount parameter of vkCmdSetScissorWithCountEXT

- VUID-vkCmdExecuteGeneratedCommandsNV-None-08635
  If a shader object is bound to any graphics stage, then both vkCmdSetViewportWithCountEXT and vkCmdSetScissorWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the viewportCount parameter of vkCmdSetViewportWithCountEXT **must** match the scissorCount parameter of vkCmdSetScissorWithCountEXT

- VUID-vkCmdExecuteGeneratedCommandsNV-viewportCount-04137
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, but not the VK_DYNAMIC_STATE_VIEWPORT_W_SCALING_NV dynamic state enabled, then the bound graphics pipeline **must** have been created with VkPipelineViewportWScalingStateCreateInfoNV::viewportCount greater or equal to the viewportCount parameter in the last call to

vkCmdSetViewportWithCountEXT

- VUID-vkCmdExecuteGeneratedCommandsNV-viewportCount-04138
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT and VK_DYNAMIC_STATE_VIEWPORT_W_SCALING_NV dynamic states enabled then the viewportCount parameter in the last call to vkCmdSetViewportWScalingNV **must** be greater than or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdExecuteGeneratedCommandsNV-None-09232
  If the VK_NV_clip_space_w_scaling extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetViewportWScalingEnableNV in the current command buffer set viewportWScalingEnable to VK_TRUE, then vkCmdSetViewportWScalingNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdExecuteGeneratedCommandsNV-None-08636
  If the VK_NV_clip_space_w_scaling extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetViewportWScalingEnableNV in the current command buffer set viewportWScalingEnable to VK_TRUE, then the viewportCount parameter in the last call to vkCmdSetViewportWScalingNV **must** be greater than or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdExecuteGeneratedCommandsNV-viewportCount-04139
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, but not the VK_DYNAMIC_STATE_VIEWPORT_SHADING_RATE_PALETTE_NV dynamic state enabled, then the bound graphics pipeline **must** have been created with VkPipelineViewportShadingRateImageStateCreateInfoNV::viewportCount greater or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdExecuteGeneratedCommandsNV-viewportCount-04140
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT and VK_DYNAMIC_STATE_VIEWPORT_SHADING_RATE_PALETTE_NV dynamic states enabled then the viewportCount parameter in the last call to vkCmdSetViewportShadingRatePaletteNV **must** be greater than or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdExecuteGeneratedCommandsNV-shadingRateImage-09233
  If the shadingRateImage feature is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetCoarseSampleOrderNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdExecuteGeneratedCommandsNV-shadingRateImage-09234
  If the shadingRateImage feature is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetShadingRateImageEnableNV in the current command buffer set shadingRateImageEnable to VK_TRUE, then vkCmdSetViewportShadingRatePaletteNV **must**

have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdExecuteGeneratedCommandsNV-None-08637
  If the `shadingRateImage` feature is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set `rasterizerDiscardEnable` to `VK_FALSE`, and the most recent call to vkCmdSetShadingRateImageEnableNV in the current command buffer set `shadingRateImageEnable` to `VK_TRUE`, then the `viewportCount` parameter in the last call to vkCmdSetViewportShadingRatePaletteNV **must** be greater than or equal to the `viewportCount` parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdExecuteGeneratedCommandsNV-VkPipelineVieportCreateInfo-04141
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT` dynamic state enabled and a VkPipelineViewportSwizzleStateCreateInfoNV structure chained from VkPipelineViewportStateCreateInfo, then the bound graphics pipeline **must** have been created with VkPipelineViewportSwizzleStateCreateInfoNV::`viewportCount` greater or equal to the `viewportCount` parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdExecuteGeneratedCommandsNV-VkPipelineVieportCreateInfo-04142
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT` dynamic state enabled and a VkPipelineViewportExclusiveScissorStateCreateInfoNV structure chained from VkPipelineViewportStateCreateInfo, then the bound graphics pipeline **must** have been created with VkPipelineViewportExclusiveScissorStateCreateInfoNV::`exclusiveScissorCount` greater or equal to the `viewportCount` parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdExecuteGeneratedCommandsNV-None-07878
  If the `exclusiveScissor` feature is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_EXCLUSIVE_SCISSOR_ENABLE_NV` dynamic state enabled, then vkCmdSetExclusiveScissorEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdExecuteGeneratedCommandsNV-None-07879
  If the `exclusiveScissor` feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_EXCLUSIVE_SCISSOR_NV` dynamic state enabled, and the most recent call to vkCmdSetExclusiveScissorEnableNV in the current command buffer set any element of `pExclusiveScissorEnables` to `VK_TRUE`, then vkCmdSetExclusiveScissorNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdExecuteGeneratedCommandsNV-None-04876
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_RASTERIZER_DISCARD_ENABLE` dynamic state enabled, then vkCmdSetRasterizerDiscardEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdExecuteGeneratedCommandsNV-None-04877

If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_BIAS_ENABLE dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetDepthBiasEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdExecuteGeneratedCommandsNV-logicOp-04878
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_LOGIC_OP_EXT dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of logicOpEnable is VK_TRUE, then vkCmdSetLogicOpEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdExecuteGeneratedCommandsNV-primitiveFragmentShadingRateWithMultipleViewports-04552
  If the primitiveFragmentShadingRateWithMultipleViewports limit is not supported, the bound graphics pipeline was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, and any of the shader stages of the bound graphics pipeline write to the PrimitiveShadingRateKHR built-in, then vkCmdSetViewportWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the viewportCount parameter of vkCmdSetViewportWithCountEXT **must** be 1
- VUID-vkCmdExecuteGeneratedCommandsNV-primitiveFragmentShadingRateWithMultipleViewports-08642
  If the primitiveFragmentShadingRateWithMultipleViewports limit is not supported, and any shader object bound to a graphics stage writes to the PrimitiveShadingRateKHR built-in, then vkCmdSetViewportWithCountEXT **must** have been called in the current command buffer prior to this drawing command, and the viewportCount parameter of vkCmdSetViewportWithCountEXT **must** be 1
- VUID-vkCmdExecuteGeneratedCommandsNV-blendEnable-04727
  If rasterization is not disabled in the bound graphics pipeline, then for each color attachment in the subpass, if the corresponding image view's format features do not contain VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT, then the blendEnable member of the corresponding element of the pAttachments member of pColorBlendState **must** be VK_FALSE
- VUID-vkCmdExecuteGeneratedCommandsNV-None-08643
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then for each color attachment in the render pass, if the corresponding image view's format features do not contain VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT, then the corresponding member of pColorBlendEnables in the most recent call to vkCmdSetColorBlendEnableEXT in the current command buffer that affected that attachment index **must** have been VK_FALSE
- VUID-vkCmdExecuteGeneratedCommandsNV-multisampledRenderToSingleSampled-07284
  If rasterization is not disabled in the bound graphics pipeline, and none of the following is enabled:
  - the VK_AMD_mixed_attachment_samples extension

- the VK_NV_framebuffer_mixed_samples extension
- the multisampledRenderToSingleSampled feature

then rasterizationSamples for the currently bound graphics pipeline **must** be the same as the current subpass color and/or depth/stencil attachments

- VUID-vkCmdExecuteGeneratedCommandsNV-None-08644
  If a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and none of the following is enabled:
  - the VK_AMD_mixed_attachment_samples extension
  - the VK_NV_framebuffer_mixed_samples extension
  - the multisampledRenderToSingleSampled feature

  then the most recent call to vkCmdSetRasterizationSamplesEXT in the current command buffer **must** have set rasterizationSamples to be the same as the number of samples for the current render pass color and/or depth/stencil attachments
- VUID-vkCmdExecuteGeneratedCommandsNV-None-08876
  If a shader object is bound to any graphics stage, the current render pass instance **must** have been begun with vkCmdBeginRenderingKHR
- VUID-vkCmdExecuteGeneratedCommandsNV-imageView-06172
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the imageView member of pDepthAttachment is not VK_NULL_HANDLE, and the layout member of pDepthAttachment is VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL, this command **must** not write any values to the depth attachment
- VUID-vkCmdExecuteGeneratedCommandsNV-imageView-06173
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the imageView member of pStencilAttachment is not VK_NULL_HANDLE, and the layout member of pStencilAttachment is VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL, this command **must** not write any values to the stencil attachment
- VUID-vkCmdExecuteGeneratedCommandsNV-imageView-06174
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the imageView member of pDepthAttachment is not VK_NULL_HANDLE, and the layout member of pDepthAttachment is VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_STENCIL_ATTACHMENT_OPTIMAL, this command **must** not write any values to the depth attachment
- VUID-vkCmdExecuteGeneratedCommandsNV-imageView-06175
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the imageView member of pStencilAttachment is not VK_NULL_HANDLE, and the layout member of pStencilAttachment is VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_STENCIL_READ_ONLY_OPTIMAL, this command **must** not write any values to the stencil attachment
- VUID-vkCmdExecuteGeneratedCommandsNV-imageView-06176
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the imageView member of pDepthAttachment is not VK_NULL_HANDLE, and the layout member

of pDepthAttachment is VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL, this command **must** not write any values to the depth attachment

- VUID-vkCmdExecuteGeneratedCommandsNV-imageView-06177
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the imageView member of pStencilAttachment is not VK_NULL_HANDLE, and the layout member of pStencilAttachment is VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL, this command **must** not write any values to the stencil attachment

- VUID-vkCmdExecuteGeneratedCommandsNV-viewMask-06178
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the currently bound graphics pipeline **must** have been created with a VkPipelineRenderingCreateInfo::viewMask equal to VkRenderingInfo::viewMask

- VUID-vkCmdExecuteGeneratedCommandsNV-colorAttachmentCount-06179
  If the dynamicRenderingUnusedAttachments feature is not enabled and the current render pass instance was begun with vkCmdBeginRenderingKHR, the currently bound graphics pipeline **must** have been created with a VkPipelineRenderingCreateInfo::colorAttachmentCount equal to VkRenderingInfo::colorAttachmentCount

- VUID-vkCmdExecuteGeneratedCommandsNV-dynamicRenderingUnusedAttachments-08910
  If the dynamicRenderingUnusedAttachments feature is not enabled, and the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingInfo::colorAttachmentCount greater than 0, then each element of the VkRenderingInfo::pColorAttachments array with an imageView not equal to VK_NULL_HANDLE **must** have been created with a VkFormat equal to the corresponding element of VkPipelineRenderingCreateInfo::pColorAttachmentFormats used to create the currently bound graphics pipeline

- VUID-vkCmdExecuteGeneratedCommandsNV-dynamicRenderingUnusedAttachments-08912
  If the dynamicRenderingUnusedAttachments feature is not enabled, and the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingInfo::colorAttachmentCount greater than 0, then each element of the VkRenderingInfo::pColorAttachments array with an imageView equal to VK_NULL_HANDLE **must** have the corresponding element of VkPipelineRenderingCreateInfo::pColorAttachmentFormats used to create the currently bound pipeline equal to VK_FORMAT_UNDEFINED

- VUID-vkCmdExecuteGeneratedCommandsNV-dynamicRenderingUnusedAttachments-08911
  If the dynamicRenderingUnusedAttachments feature is enabled, and the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingInfo::colorAttachmentCount greater than 0, then each element of the VkRenderingInfo::pColorAttachments array with an imageView not equal to VK_NULL_HANDLE **must** have been created with a VkFormat equal to the corresponding element of VkPipelineRenderingCreateInfo::pColorAttachmentFormats used to create the currently bound graphics pipeline, or the corresponding element of VkPipelineRenderingCreateInfo::pColorAttachmentFormats, if it exists, **must** be VK_FORMAT_UNDEFINED

- VUID-vkCmdExecuteGeneratedCommandsNV-colorAttachmentCount-09362

If the current render pass instance was begun with vkCmdBeginRenderingKHR, with a VkRenderingInfo::colorAttachmentCount equal to 1, there is no shader object bound to any graphics stage, and a color attachment with a resolve mode of VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID, each element of the VkRenderingInfo::pColorAttachments array with a resolveImageView not equal to VK_NULL_HANDLE **must** have been created with an image created with a VkExternalFormatANDROID::externalFormat value equal to the VkExternalFormatANDROID::externalFormat value used to create the currently bound graphics pipeline

- VUID-vkCmdExecuteGeneratedCommandsNV-None-09363
  If there is no shader object bound to any graphics stage, the current render pass instance was begun with vkCmdBeginRenderingKHR and a VkRenderingInfo::colorAttachmentCount equal to 1, and a color attachment with a resolve mode of VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID, each element of the VkRenderingInfo::pColorAttachments array with a imageView not equal to VK_NULL_HANDLE **must** have been created with an image created with a VkExternalFormatANDROID::externalFormat value equal to the VkExternalFormatANDROID::externalFormat value used to create the currently bound graphics pipeline
- VUID-vkCmdExecuteGeneratedCommandsNV-None-09364
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is no shader object bound to any graphics stage, and the currently bound graphics pipeline was created with a non-zero VkExternalFormatANDROID::externalFormat value and with the VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT dynamic state enabled, then vkCmdSetColorBlendEnableEXT **must** have set the blend enable to VK_FALSE prior to this drawing command
- VUID-vkCmdExecuteGeneratedCommandsNV-None-09365
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is no shader object bound to any graphics stage, and the currently bound graphics pipeline was created with a non-zero VkExternalFormatANDROID::externalFormat value and with the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT dynamic state enabled, then vkCmdSetRasterizationSamplesEXT **must** have set rasterizationSamples to VK_SAMPLE_COUNT_1_BIT prior to this drawing command
- VUID-vkCmdExecuteGeneratedCommandsNV-None-09366
  If there is a shader object bound to any graphics stage, and the current render pass includes a color attachment that uses the VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID resolve mode, then vkCmdSetColorBlendEnableEXT **must** have set blend enable to VK_FALSE prior to this drawing command
- VUID-vkCmdExecuteGeneratedCommandsNV-rasterizationSamples-09367
  If there is a shader object bound to any graphics stage, and the current render pass includes a color attachment that uses the VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID resolve mode, then vkCmdSetRasterizationSamplesEXT **must** have set rasterizationSamples to VK_SAMPLE_COUNT_1_BIT prior to this drawing command

- VUID-vkCmdExecuteGeneratedCommandsNV-None-09368
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is no shader object bound to any graphics stage, and the currently bound graphics pipeline was created with a non-zero VkExternalFormatANDROID::externalFormat value and with the VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR dynamic state enabled, then vkCmdSetFragmentShadingRateKHR **must** have set pFragmentSize->width to 1 prior to this drawing command
- VUID-vkCmdExecuteGeneratedCommandsNV-None-09369
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is no shader object bound to any graphics stage, and the currently bound graphics pipeline was created with a non-zero VkExternalFormatANDROID::externalFormat value and with the VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR dynamic state enabled, then vkCmdSetFragmentShadingRateKHR **must** have set pFragmentSize->height to 1 prior to this drawing command
- VUID-vkCmdExecuteGeneratedCommandsNV-pFragmentSize-09370
  If there is a shader object bound to any graphics stage, and the current render pass includes a color attachment that uses the VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID resolve mode, then vkCmdSetFragmentShadingRateKHR **must** have set pFragmentSize->width to 1 prior to this drawing command
- VUID-vkCmdExecuteGeneratedCommandsNV-pFragmentSize-09371
  If there is a shader object bound to any graphics stage, and the current render pass includes a color attachment that uses the VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID resolve mode, then vkCmdSetFragmentShadingRateKHR **must** have set pFragmentSize->height to 1 prior to this drawing command
- VUID-vkCmdExecuteGeneratedCommandsNV-None-07749
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_WRITE_ENABLE_EXT dynamic state enabled then vkCmdSetColorWriteEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdExecuteGeneratedCommandsNV-None-08646
  If the colorWriteEnable feature is enabled on the device, and a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetColorWriteEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdExecuteGeneratedCommandsNV-attachmentCount-07750
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_WRITE_ENABLE_EXT dynamic state enabled then the attachmentCount parameter of vkCmdSetColorWriteEnableEXT **must** be greater than or equal to the VkPipelineColorBlendStateCreateInfo::attachmentCount of the currently bound graphics pipeline
- VUID-vkCmdExecuteGeneratedCommandsNV-None-08647

If the colorWriteEnable feature is enabled on the device, and a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then the attachmentCount parameter of most recent call to vkCmdSetColorWriteEnableEXT in the current command buffer **must** be greater than or equal to the number of color attachments in the current render pass instance

- VUID-vkCmdExecuteGeneratedCommandsNV-None-07751
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_DISCARD_RECTANGLE_EXT dynamic state enabled then vkCmdSetDiscardRectangleEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command for each discard rectangle in VkPipelineDiscardRectangleStateCreateInfoEXT::discardRectangleCount
- VUID-vkCmdExecuteGeneratedCommandsNV-None-07880
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_DISCARD_RECTANGLE_ENABLE_EXT dynamic state enabled then vkCmdSetDiscardRectangleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdExecuteGeneratedCommandsNV-rasterizerDiscardEnable-09236
  If the VK_EXT_discard_rectangles extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetDiscardRectangleEnableEXT in the current command buffer set discardRectangleEnable to VK_TRUE, then vkCmdSetDiscardRectangleEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdExecuteGeneratedCommandsNV-None-08648
  If the VK_EXT_discard_rectangles extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetDiscardRectangleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdExecuteGeneratedCommandsNV-None-07881
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_DISCARD_RECTANGLE_MODE_EXT dynamic state enabled then vkCmdSetDiscardRectangleModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdExecuteGeneratedCommandsNV-None-08649
  If the VK_EXT_discard_rectangles extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetDiscardRectangleEnableEXT in the current command buffer set discardRectangleEnable to VK_TRUE, then vkCmdSetDiscardRectangleModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdExecuteGeneratedCommandsNV-dynamicRenderingUnusedAttachments-

08913
If the current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is not enabled, and VkRenderingInfo::pDepthAttachment->imageView was VK_NULL_HANDLE, the value of VkPipelineRenderingCreateInfo::depthAttachmentFormat used to create the currently bound graphics pipeline **must** be equal to VK_FORMAT_UNDEFINED

- VUID-vkCmdExecuteGeneratedCommandsNV-dynamicRenderingUnusedAttachments-08914
If current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is not enabled, and VkRenderingInfo::pDepthAttachment->imageView was not VK_NULL_HANDLE, the value of VkPipelineRenderingCreateInfo::depthAttachmentFormat used to create the currently bound graphics pipeline **must** be equal to the VkFormat used to create VkRenderingInfo::pDepthAttachment->imageView

- VUID-vkCmdExecuteGeneratedCommandsNV-dynamicRenderingUnusedAttachments-08915
If the current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is enabled, VkRenderingInfo::pDepthAttachment->imageView was not VK_NULL_HANDLE, and the value of VkPipelineRenderingCreateInfo::depthAttachmentFormat used to create the currently bound graphics pipeline was not equal to the VkFormat used to create VkRenderingInfo::pDepthAttachment->imageView, the value of the format **must** be VK_FORMAT_UNDEFINED

- VUID-vkCmdExecuteGeneratedCommandsNV-dynamicRenderingUnusedAttachments-08916
If the current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is not enabled, and VkRenderingInfo::pStencilAttachment->imageView was VK_NULL_HANDLE, the value of VkPipelineRenderingCreateInfo::stencilAttachmentFormat used to create the currently bound graphics pipeline **must** be equal to VK_FORMAT_UNDEFINED

- VUID-vkCmdExecuteGeneratedCommandsNV-dynamicRenderingUnusedAttachments-08917
If current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is not enabled, and VkRenderingInfo::pStencilAttachment->imageView was not VK_NULL_HANDLE, the value of VkPipelineRenderingCreateInfo::stencilAttachmentFormat used to create the currently bound graphics pipeline **must** be equal to the VkFormat used to create VkRenderingInfo::pStencilAttachment->imageView

- VUID-vkCmdExecuteGeneratedCommandsNV-dynamicRenderingUnusedAttachments-08918
If the current render pass instance was begun with vkCmdBeginRenderingKHR, the dynamicRenderingUnusedAttachments feature is enabled, VkRenderingInfo::pStencilAttachment->imageView was not VK_NULL_HANDLE, and the value of VkPipelineRenderingCreateInfo::stencilAttachmentFormat used to create the currently bound graphics pipeline was not equal to the VkFormat used to create VkRenderingInfo::pStencilAttachment->imageView, the value of the format **must** be VK_FORMAT_UNDEFINED

- VUID-vkCmdExecuteGeneratedCommandsNV-imageView-06183
  If the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingFragmentShadingRateAttachmentInfoKHR::imageView was not VK_NULL_HANDLE, the currently bound graphics pipeline **must** have been created with VK_PIPELINE_CREATE_RENDERING_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR
- VUID-vkCmdExecuteGeneratedCommandsNV-imageView-06184
  If the current render pass instance was begun with vkCmdBeginRenderingKHR and VkRenderingFragmentDensityMapAttachmentInfoEXT::imageView was not VK_NULL_HANDLE, the currently bound graphics pipeline **must** have been created with VK_PIPELINE_CREATE_RENDERING_FRAGMENT_DENSITY_MAP_ATTACHMENT_BIT_EXT
- VUID-vkCmdExecuteGeneratedCommandsNV-colorAttachmentCount-06185
  If the currently bound pipeline was created with a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and the current render pass instance was begun with vkCmdBeginRenderingKHR with a VkRenderingInfo::colorAttachmentCount parameter greater than 0, then each element of the VkRenderingInfo::pColorAttachments array with a imageView not equal to VK_NULL_HANDLE **must** have been created with a sample count equal to the corresponding element of the pColorAttachmentSamples member of VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV used to create the currently bound graphics pipeline
- VUID-vkCmdExecuteGeneratedCommandsNV-pDepthAttachment-06186
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the currently bound pipeline was created with a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and VkRenderingInfo::pDepthAttachment->imageView was not VK_NULL_HANDLE, the value of the depthStencilAttachmentSamples member of VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV used to create the currently bound graphics pipeline **must** be equal to the sample count used to create VkRenderingInfo::pDepthAttachment->imageView
- VUID-vkCmdExecuteGeneratedCommandsNV-pStencilAttachment-06187
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the currently bound pipeline was created with a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and VkRenderingInfo::pStencilAttachment->imageView was not VK_NULL_HANDLE, the value of the depthStencilAttachmentSamples member of VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV used to create the currently bound graphics pipeline **must** be equal to the sample count used to create VkRenderingInfo::pStencilAttachment->imageView
- VUID-vkCmdExecuteGeneratedCommandsNV-multisampledRenderToSingleSampled-07285
  If the currently bound pipeline was created without a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and the multisampledRenderToSingleSampled feature is not enabled, and the current render pass instance was begun with vkCmdBeginRenderingKHR with a VkRenderingInfo::colorAttachmentCount parameter greater than 0, then each element of the VkRenderingInfo::pColorAttachments array with a imageView not equal to VK_NULL_HANDLE **must** have been created with a sample count equal to the value of rasterizationSamples for the currently bound graphics pipeline

- VUID-vkCmdExecuteGeneratedCommandsNV-multisampledRenderToSingleSampled-07286
  If the currently bound pipeline was created without a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and the `multisampledRenderToSingleSampled` feature is not enabled, and VkRenderingInfo::`pDepthAttachment->imageView` was not VK_NULL_HANDLE, the value of `rasterizationSamples` for the currently bound graphics pipeline **must** be equal to the sample count used to create VkRenderingInfo::`pDepthAttachment->imageView`
- VUID-vkCmdExecuteGeneratedCommandsNV-multisampledRenderToSingleSampled-07287
  If the currently bound pipeline was created without a VkAttachmentSampleCountInfoAMD or VkAttachmentSampleCountInfoNV structure, and the `multisampledRenderToSingleSampled` feature is not enabled, and VkRenderingInfo::`pStencilAttachment->imageView` was not VK_NULL_HANDLE, the value of `rasterizationSamples` for the currently bound graphics pipeline **must** be equal to the sample count used to create VkRenderingInfo::`pStencilAttachment->imageView`
- VUID-vkCmdExecuteGeneratedCommandsNV-pNext-07935
  If this command has been called inside a render pass instance started with vkCmdBeginRenderingKHR, and the `pNext` chain of VkRenderingInfo includes a VkMultisampledRenderToSingleSampledInfoEXT structure with `multisampledRenderToSingleSampledEnable` equal to `VK_TRUE`, then the value of `rasterizationSamples` for the currently bound graphics pipeline **must** be equal to VkMultisampledRenderToSingleSampledInfoEXT::`rasterizationSamples`
- VUID-vkCmdExecuteGeneratedCommandsNV-renderPass-06198
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, the currently bound pipeline **must** have been created with a VkGraphicsPipelineCreateInfo::`renderPass` equal to VK_NULL_HANDLE
- VUID-vkCmdExecuteGeneratedCommandsNV-pColorAttachments-08963
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is a graphics pipeline bound with a fragment shader that statically writes to a color attachment, the color write mask is not zero, color writes are enabled, and the corresponding element of the VkRenderingInfo::`pColorAttachments->imageView` was not VK_NULL_HANDLE, then the corresponding element of VkPipelineRenderingCreateInfo::`pColorAttachmentFormats` used to create the pipeline **must** not be `VK_FORMAT_UNDEFINED`
- VUID-vkCmdExecuteGeneratedCommandsNV-pDepthAttachment-08964
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is a graphics pipeline bound, depth test is enabled, depth write is enabled, and the VkRenderingInfo::`pDepthAttachment->imageView` was not VK_NULL_HANDLE, then the VkPipelineRenderingCreateInfo::`depthAttachmentFormat` used to create the pipeline **must** not be `VK_FORMAT_UNDEFINED`
- VUID-vkCmdExecuteGeneratedCommandsNV-pStencilAttachment-08965
  If the current render pass instance was begun with vkCmdBeginRenderingKHR, there is a graphics pipeline bound, stencil test is enabled and the VkRenderingInfo::`pStencilAttachment->imageView` was not VK_NULL_HANDLE, then the VkPipelineRenderingCreateInfo::`stencilAttachmentFormat` used to create the pipeline **must**

not be VK_FORMAT_UNDEFINED

- VUID-vkCmdExecuteGeneratedCommandsNV-primitivesGeneratedQueryWithRasterizerDiscard-06708
  If the primitivesGeneratedQueryWithRasterizerDiscard feature is not enabled and the VK_QUERY_TYPE_PRIMITIVES_GENERATED_EXT query is active, rasterization discard **must** not be enabled

- VUID-vkCmdExecuteGeneratedCommandsNV-primitivesGeneratedQueryWithNonZeroStreams-06709
  If the primitivesGeneratedQueryWithNonZeroStreams feature is not enabled and the VK_QUERY_TYPE_PRIMITIVES_GENERATED_EXT query is active, the bound graphics pipeline **must** not have been created with a non-zero value in VkPipelineRasterizationStateStreamCreateInfoEXT::rasterizationStream

- VUID-vkCmdExecuteGeneratedCommandsNV-None-07619
  If a shader object is bound to the VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_TESSELLATION_DOMAIN_ORIGIN_EXT dynamic state enabled, then vkCmdSetTessellationDomainOriginEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdExecuteGeneratedCommandsNV-None-07620
  If the depthClamp feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_CLAMP_ENABLE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetDepthClampEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdExecuteGeneratedCommandsNV-None-07621
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_POLYGON_MODE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetPolygonModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdExecuteGeneratedCommandsNV-None-07622
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetRasterizationSamplesEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdExecuteGeneratedCommandsNV-None-07623
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_SAMPLE_MASK_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetSampleMaskEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdExecuteGeneratedCommandsNV-alphaToCoverageEnable-08919
  If the bound graphics pipeline state was created with the

VK_DYNAMIC_STATE_ALPHA_TO_COVERAGE_ENABLE_EXT dynamic state enabled, and alphaToCoverageEnable was VK_TRUE in the last call to vkCmdSetAlphaToCoverageEnableEXT, then the Fragment Output Interface **must** contain a variable for the alpha Component word in Location 0 at Index 0

- VUID-vkCmdExecuteGeneratedCommandsNV-alphaToCoverageEnable-08920
  If a shader object is bound to any graphics stage, and the most recent call to vkCmdSetAlphaToCoverageEnableEXT in the current command buffer set alphaToCoverageEnable to VK_TRUE, then the Fragment Output Interface **must** contain a variable for the alpha Component word in Location 0 at Index 0

- VUID-vkCmdExecuteGeneratedCommandsNV-None-07624
  If a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_ALPHA_TO_COVERAGE_ENABLE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetAlphaToCoverageEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdExecuteGeneratedCommandsNV-None-07625
  If the alphaToOne feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_ALPHA_TO_ONE_ENABLE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetAlphaToOneEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdExecuteGeneratedCommandsNV-None-07626
  If the logicOp feature is enabled, a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_LOGIC_OP_ENABLE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetLogicOpEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdExecuteGeneratedCommandsNV-None-07627
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT dynamic state enabled then vkCmdSetColorBlendEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdExecuteGeneratedCommandsNV-None-08657
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and both the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE and there are color attachments bound, then vkCmdSetColorBlendEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdExecuteGeneratedCommandsNV-None-07628
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_EQUATION_EXT dynamic state enabled then vkCmdSetColorBlendEquationEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdExecuteGeneratedCommandsNV-None-08658
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetColorBlendEnableEXT for any attachment set that attachment's value in pColorBlendEnables to VK_TRUE, then vkCmdSetColorBlendEquationEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdExecuteGeneratedCommandsNV-None-07629
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_WRITE_MASK_EXT dynamic state enabled then vkCmdSetColorWriteMaskEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdExecuteGeneratedCommandsNV-None-08659
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and both the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE and there are color attachments bound, then vkCmdSetColorWriteMaskEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdExecuteGeneratedCommandsNV-None-07630
  If the geometryStreams feature is enabled, and a shader object is bound to the VK_SHADER_STAGE_GEOMETRY_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_RASTERIZATION_STREAM_EXT dynamic state enabled, then vkCmdSetRasterizationStreamEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdExecuteGeneratedCommandsNV-None-07631
  If the VK_EXT_conservative_rasterization extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_CONSERVATIVE_RASTERIZATION_MODE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetConservativeRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdExecuteGeneratedCommandsNV-None-07632
  If the VK_EXT_conservative_rasterization extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_EXTRA_PRIMITIVE_OVERESTIMATION_SIZE_EXT dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of conservativeRasterizationMode is VK_CONSERVATIVE_RASTERIZATION_MODE_OVERESTIMATE_EXT, then vkCmdSetExtraPrimitiveOverestimationSizeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdExecuteGeneratedCommandsNV-None-07633
  If the depthClipEnable feature is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_CLIP_ENABLE_EXT dynamic state, then vkCmdSetDepthClipEnableEXT **must** have been called and not subsequently invalidated

in the current command buffer prior to this drawing command

- VUID-vkCmdExecuteGeneratedCommandsNV-None-07634
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT dynamic state enabled then vkCmdSetSampleLocationsEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdExecuteGeneratedCommandsNV-None-08664
  If the VK_EXT_sample_locations extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetSampleLocationsEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdExecuteGeneratedCommandsNV-None-07635
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_ADVANCED_EXT dynamic state enabled then vkCmdSetColorBlendAdvancedEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdExecuteGeneratedCommandsNV-rasterizerDiscardEnable-09416
  If the VK_EXT_blend_operation_advanced extension is enabled, and a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then at least one of vkCmdSetColorBlendEquationEXT and vkCmdSetColorBlendAdvancedEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdExecuteGeneratedCommandsNV-None-07636
  If the VK_EXT_provoking_vertex extension is enabled, a shader object is bound to the VK_SHADER_STAGE_VERTEX_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_PROVOKING_VERTEX_MODE_EXT dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetProvokingVertexModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdExecuteGeneratedCommandsNV-None-07637
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT dynamic state enabled then vkCmdSetLineRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdExecuteGeneratedCommandsNV-None-08666
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetPolygonModeEXT in the current command buffer set polygonMode to VK_POLYGON_MODE_LINE, then vkCmdSetLineRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing

command

- VUID-vkCmdExecuteGeneratedCommandsNV-None-08667
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object is bound to the VK_SHADER_STAGE_VERTEX_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetPrimitiveTopologyEXT in the current command buffer set primitiveTopology to any line topology, then vkCmdSetLineRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdExecuteGeneratedCommandsNV-None-08668
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object that outputs line primitives is bound to the VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT or VK_SHADER_STAGE_GEOMETRY_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetLineRasterizationModeEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdExecuteGeneratedCommandsNV-None-07638
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT dynamic state enabled then vkCmdSetLineStippleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdExecuteGeneratedCommandsNV-None-08669
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetPolygonModeEXT in the current command buffer set polygonMode to VK_POLYGON_MODE_LINE, then vkCmdSetLineStippleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdExecuteGeneratedCommandsNV-None-08670
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object is bound to the VK_SHADER_STAGE_VERTEX_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetPrimitiveTopologyEXT in the current command buffer set primitiveTopology to any line topology, then vkCmdSetLineStippleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdExecuteGeneratedCommandsNV-None-08671
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object that outputs line primitives is bound to the VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT or VK_SHADER_STAGE_GEOMETRY_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command

buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetLineStippleEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdExecuteGeneratedCommandsNV-None-07849
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_KHR dynamic state enabled then vkCmdSetLineStippleKHR **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdExecuteGeneratedCommandsNV-None-08672
  If the VK_KHR_line_rasterization or VK_EXT_line_rasterization extension is enabled, and a shader object is bound to any graphics stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the most recent call to vkCmdSetLineStippleEnableEXT in the current command buffer set stippledLineEnable to VK_TRUE, then vkCmdSetLineStippleEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdExecuteGeneratedCommandsNV-None-07639
  If the depthClipControl feature is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_DEPTH_CLIP_NEGATIVE_ONE_TO_ONE_EXT dynamic state enabled, then vkCmdSetDepthClipNegativeOneToOneEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdExecuteGeneratedCommandsNV-None-07640
  If the VK_NV_clip_space_w_scaling extension is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_VIEWPORT_W_SCALING_ENABLE_NV dynamic state enabled, then vkCmdSetViewportWScalingEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdExecuteGeneratedCommandsNV-None-07641
  If the VK_NV_viewport_swizzle extension is enabled, and a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_VIEWPORT_SWIZZLE_NV dynamic state enabled, then vkCmdSetViewportSwizzleNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdExecuteGeneratedCommandsNV-None-07642
  If the VK_NV_fragment_coverage_to_color extension is enabled, a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_TO_COLOR_ENABLE_NV dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetCoverageToColorEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdExecuteGeneratedCommandsNV-None-07643
  If the VK_NV_fragment_coverage_to_color extension is enabled, a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_TO_COLOR_LOCATION_NV dynamic state enabled, the

current value of rasterizerDiscardEnable is VK_FALSE, and the current value of coverageToColorEnable is VK_TRUE, then vkCmdSetCoverageToColorLocationNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdExecuteGeneratedCommandsNV-None-07644
  If the VK_NV_framebuffer_mixed_samples extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_MODULATION_MODE_NV dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetCoverageModulationModeNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdExecuteGeneratedCommandsNV-None-07645
  If the VK_NV_framebuffer_mixed_samples extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_MODULATION_TABLE_ENABLE_NV dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of coverageModulationMode is any value other than VK_COVERAGE_MODULATION_MODE_NONE_NV, then vkCmdSetCoverageModulationTableEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdExecuteGeneratedCommandsNV-None-07646
  If the VK_NV_framebuffer_mixed_samples extension is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_MODULATION_TABLE_NV dynamic state enabled, the current value of rasterizerDiscardEnable is VK_FALSE, and the current value of coverageModulationTableEnable is VK_TRUE, then vkCmdSetCoverageModulationTableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdExecuteGeneratedCommandsNV-None-07647
  If the shadingRateImage feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_SHADING_RATE_IMAGE_ENABLE_NV dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetShadingRateImageEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdExecuteGeneratedCommandsNV-pipelineFragmentShadingRate-09238
  If the pipelineFragmentShadingRate feature is enabled, a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetFragmentShadingRateKHR **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdExecuteGeneratedCommandsNV-None-07648
  If the representativeFragmentTest feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_REPRESENTATIVE_FRAGMENT_TEST_ENABLE_NV dynamic state enabled, and the

current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetRepresentativeFragmentTestEnableNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdExecuteGeneratedCommandsNV-None-07649
  If the coverageReductionMode feature is enabled, a shader object is bound to any graphics stage or a graphics pipeline is bound which was created with the VK_DYNAMIC_STATE_COVERAGE_REDUCTION_MODE_NV dynamic state enabled, and the current value of rasterizerDiscardEnable is VK_FALSE, then vkCmdSetCoverageReductionModeNV **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdExecuteGeneratedCommandsNV-pColorBlendEnables-07470
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT state enabled and the last call to vkCmdSetColorBlendEnableEXT set pColorBlendEnables for any attachment to VK_TRUE, then for those attachments in the subpass the corresponding image view's format features **must** contain VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT

- VUID-vkCmdExecuteGeneratedCommandsNV-rasterizationSamples-07471
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, and the current subpass does not use any color and/or depth/stencil attachments, then the rasterizationSamples in the last call to vkCmdSetRasterizationSamplesEXT **must** follow the rules for a zero-attachment subpass

- VUID-vkCmdExecuteGeneratedCommandsNV-samples-07472
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_MASK_EXT state enabled and the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state disabled, then the samples parameter in the last call to vkCmdSetSampleMaskEXT **must** be greater or equal to the VkPipelineMultisampleStateCreateInfo::rasterizationSamples parameter used to create the bound graphics pipeline

- VUID-vkCmdExecuteGeneratedCommandsNV-samples-07473
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_MASK_EXT state and VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT states enabled, then the samples parameter in the last call to vkCmdSetSampleMaskEXT **must** be greater or equal to the rasterizationSamples parameter in the last call to vkCmdSetRasterizationSamplesEXT

- VUID-vkCmdExecuteGeneratedCommandsNV-rasterizationSamples-07474
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, and neither the VK_AMD_mixed_attachment_samples nor the VK_NV_framebuffer_mixed_samples extensions are enabled, then the rasterizationSamples in the last call to vkCmdSetRasterizationSamplesEXT **must** be the same as the current subpass color and/or depth/stencil attachments

- VUID-vkCmdExecuteGeneratedCommandsNV-None-09211
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, or a shader object is bound to

any graphics stage, and the current render pass instance includes a VkMultisampledRenderToSingleSampledInfoEXT structure with multisampledRenderToSingleSampledEnable equal to VK_TRUE, then the rasterizationSamples in the last call to vkCmdSetRasterizationSamplesEXT **must** be the same as the rasterizationSamples member of that structure

- VUID-vkCmdExecuteGeneratedCommandsNV-firstAttachment-07476
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT dynamic state enabled then vkCmdSetColorBlendEnableEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the firstAttachment and attachmentCount parameters of vkCmdSetColorBlendEnableEXT calls **must** specify an enable for all active color attachments in the current subpass

- VUID-vkCmdExecuteGeneratedCommandsNV-rasterizerDiscardEnable-09417
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetColorBlendEnableEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the firstAttachment and attachmentCount parameters of vkCmdSetColorBlendEnableEXT calls **must** specify an enable for all active color attachments in the current subpass

- VUID-vkCmdExecuteGeneratedCommandsNV-firstAttachment-07477
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_EQUATION_EXT dynamic state enabled then vkCmdSetColorBlendEquationEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the firstAttachment and attachmentCount parameters of vkCmdSetColorBlendEquationEXT calls **must** specify the blend equations for all active color attachments in the current subpass where blending is enabled

- VUID-vkCmdExecuteGeneratedCommandsNV-rasterizerDiscardEnable-09418
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and both the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE and there are color attachments bound, then vkCmdSetColorBlendEquationEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the firstAttachment and attachmentCount parameters of vkCmdSetColorBlendEquationEXT calls **must** specify the blend equations for all active color attachments in the current subpass where blending is enabled

- VUID-vkCmdExecuteGeneratedCommandsNV-firstAttachment-07478
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_WRITE_MASK_EXT dynamic state enabled then vkCmdSetColorWriteMaskEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the firstAttachment and attachmentCount parameters of vkCmdSetColorWriteMaskEXT calls **must** specify the color write mask for all active color attachments in the current subpass

- VUID-vkCmdExecuteGeneratedCommandsNV-rasterizerDiscardEnable-09419

If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, then vkCmdSetColorWriteMaskEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the firstAttachment and attachmentCount parameters of vkCmdSetColorWriteMaskEXT calls **must** specify the color write mask for all active color attachments in the current subpass

- VUID-vkCmdExecuteGeneratedCommandsNV-firstAttachment-07479
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_ADVANCED_EXT dynamic state enabled then vkCmdSetColorBlendAdvancedEXT **must** have been called in the current command buffer prior to this drawing command, and the attachments specified by the firstAttachment and attachmentCount parameters of vkCmdSetColorBlendAdvancedEXT calls **must** specify the advanced blend equations for all active color attachments in the current subpass where blending is enabled

- VUID-vkCmdExecuteGeneratedCommandsNV-advancedBlendMaxColorAttachments-07480
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COLOR_BLEND_ADVANCED_EXT and VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT dynamic states enabled and the last calls to vkCmdSetColorBlendEnableEXT and vkCmdSetColorBlendAdvancedEXT have enabled advanced blending, then the number of active color attachments in the current subpass **must** not exceed advancedBlendMaxColorAttachments

- VUID-vkCmdExecuteGeneratedCommandsNV-primitivesGeneratedQueryWithNonZeroStreams-07481
  If the primitivesGeneratedQueryWithNonZeroStreams feature is not enabled and the VK_QUERY_TYPE_PRIMITIVES_GENERATED_EXT query is active, and the bound graphics pipeline was created with VK_DYNAMIC_STATE_RASTERIZATION_STREAM_EXT state enabled, the last call to vkCmdSetRasterizationStreamEXT **must** have set the rasterizationStream to zero

- VUID-vkCmdExecuteGeneratedCommandsNV-sampleLocationsPerPixel-07482
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state enabled and the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state disabled, then the sampleLocationsPerPixel member of pSampleLocationsInfo in the last call to vkCmdSetSampleLocationsEXT **must** equal the rasterizationSamples member of the VkPipelineMultisampleStateCreateInfo structure the bound graphics pipeline has been created with

- VUID-vkCmdExecuteGeneratedCommandsNV-sampleLocationsPerPixel-07483
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state enabled and the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, then the sampleLocationsPerPixel member of pSampleLocationsInfo in the last call to vkCmdSetSampleLocationsEXT **must** equal the rasterizationSamples parameter of the last call to vkCmdSetRasterizationSamplesEXT

- VUID-vkCmdExecuteGeneratedCommandsNV-sampleLocationsEnable-07484

If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, or the bound graphics pipeline was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, and sampleLocationsEnable was VK_TRUE in the last call to vkCmdSetSampleLocationsEnableEXT, and the current subpass has a depth/stencil attachment, then that attachment **must** have been created with the VK_IMAGE_CREATE_SAMPLE_LOCATIONS_COMPATIBLE_DEPTH_BIT_EXT bit set

- VUID-vkCmdExecuteGeneratedCommandsNV-sampleLocationsEnable-07485
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state enabled and the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, and if sampleLocationsEnable was VK_TRUE in the last call to vkCmdSetSampleLocationsEnableEXT, then the sampleLocationsInfo.sampleLocationGridSize.width in the last call to vkCmdSetSampleLocationsEXT **must** evenly divide VkMultisamplePropertiesEXT::sampleLocationGridSize.width as returned by vkGetPhysicalDeviceMultisamplePropertiesEXT with a samples parameter equaling rasterizationSamples

- VUID-vkCmdExecuteGeneratedCommandsNV-sampleLocationsEnable-07486
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state enabled and the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, and if sampleLocationsEnable was VK_TRUE in the last call to vkCmdSetSampleLocationsEnableEXT, then the sampleLocationsInfo.sampleLocationGridSize.height in the last call to vkCmdSetSampleLocationsEXT **must** evenly divide VkMultisamplePropertiesEXT::sampleLocationGridSize.height as returned by vkGetPhysicalDeviceMultisamplePropertiesEXT with a samples parameter equaling rasterizationSamples

- VUID-vkCmdExecuteGeneratedCommandsNV-sampleLocationsEnable-07487
  If a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, and if sampleLocationsEnable was VK_TRUE in the last call to vkCmdSetSampleLocationsEnableEXT, the fragment shader code **must** not statically use the extended instruction InterpolateAtSample

- VUID-vkCmdExecuteGeneratedCommandsNV-sampleLocationsEnable-07936
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state disabled and the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, the sampleLocationsEnable member of a VkPipelineSampleLocationsStateCreateInfoEXT::sampleLocationsEnable in the bound graphics pipeline is VK_TRUE or VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, then, sampleLocationsInfo.sampleLocationGridSize.width **must** evenly divide VkMultisamplePropertiesEXT::sampleLocationGridSize.width as returned by vkGetPhysicalDeviceMultisamplePropertiesEXT with a samples parameter equaling the value of rasterizationSamples in the last call to vkCmdSetRasterizationSamplesEXT

- VUID-vkCmdExecuteGeneratedCommandsNV-sampleLocationsEnable-07937
  If the bound graphics pipeline state was created with the

VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state disabled and the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, the sampleLocationsEnable member of a VkPipelineSampleLocationsStateCreateInfoEXT::sampleLocationsEnable in the bound graphics pipeline is VK_TRUE or VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, then, sampleLocationsInfo.sampleLocationGridSize.height **must** evenly divide VkMultisamplePropertiesEXT::sampleLocationGridSize.height as returned by vkGetPhysicalDeviceMultisamplePropertiesEXT with a samples parameter equaling the value of rasterizationSamples in the last call to vkCmdSetRasterizationSamplesEXT

- VUID-vkCmdExecuteGeneratedCommandsNV-sampleLocationsEnable-07938
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT state disabled and the VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT state enabled, the sampleLocationsEnable member of a VkPipelineSampleLocationsStateCreateInfoEXT::sampleLocationsEnable in the bound graphics pipeline is VK_TRUE or VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT state enabled, then, sampleLocationsInfo.sampleLocationsPerPixel **must** equal rasterizationSamples in the last call to vkCmdSetRasterizationSamplesEXT

- VUID-vkCmdExecuteGeneratedCommandsNV-coverageModulationTableEnable-07488
  If a shader object is bound to any graphics stage or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COVERAGE_MODULATION_TABLE_ENABLE_NV state enabled, and the last call to vkCmdSetCoverageModulationTableEnableNV set coverageModulationTableEnable to VK_TRUE, then the coverageModulationTableCount parameter in the last call to vkCmdSetCoverageModulationTableNV **must** equal the current rasterizationSamples divided by the number of color samples in the current subpass

- VUID-vkCmdExecuteGeneratedCommandsNV-rasterizationSamples-07489
  If the VK_NV_framebuffer_mixed_samples extension is enabled, and if current subpass has a depth/stencil attachment and depth test, stencil test, or depth bounds test are enabled in the currently bound pipeline state, then the current rasterizationSamples **must** be the same as the sample count of the depth/stencil attachment

- VUID-vkCmdExecuteGeneratedCommandsNV-coverageToColorEnable-07490
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COVERAGE_TO_COLOR_ENABLE_NV state enabled and the last call to vkCmdSetCoverageToColorEnableNV set the coverageToColorEnable to VK_TRUE, then the current subpass **must** have a color attachment at the location selected by the last call to vkCmdSetCoverageToColorLocationNV coverageToColorLocation, with a VkFormat of VK_FORMAT_R8_UINT, VK_FORMAT_R8_SINT, VK_FORMAT_R16_UINT, VK_FORMAT_R16_SINT, VK_FORMAT_R32_UINT, or VK_FORMAT_R32_SINT

- VUID-vkCmdExecuteGeneratedCommandsNV-rasterizerDiscardEnable-09420
  If the VK_NV_fragment_coverage_to_color extension is enabled, and a shader object is bound to the VK_SHADER_STAGE_FRAGMENT_BIT stage, and the most recent call to vkCmdSetRasterizerDiscardEnableEXT in the current command buffer set rasterizerDiscardEnable to VK_FALSE, and the last call to vkCmdSetCoverageToColorEnableNV set the coverageToColorEnable to VK_TRUE, then the current subpass **must** have a color attachment at the location selected by the last call to vkCmdSetCoverageToColorLocationNV coverageToColorLocation, with a VkFormat of VK_FORMAT_R8_UINT, VK_FORMAT_R8_SINT, VK_FORMAT_R16_UINT, VK_FORMAT_R16_SINT,

VK_FORMAT_R32_UINT, or VK_FORMAT_R32_SINT

- VUID-vkCmdExecuteGeneratedCommandsNV-coverageReductionMode-07491
  If this VK_NV_coverage_reduction_mode extension is enabled, the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_COVERAGE_TO_COLOR_ENABLE_NV and VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT states enabled, the current coverage reduction mode coverageReductionMode, then the current rasterizationSamples, and the sample counts for the color and depth/stencil attachments (if the subpass has them) **must** be a valid combination returned by vkGetPhysicalDeviceSupportedFramebufferMixedSamplesCombinationsNV
- VUID-vkCmdExecuteGeneratedCommandsNV-viewportCount-07492
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT dynamic state enabled, but not the VK_DYNAMIC_STATE_VIEWPORT_SWIZZLE_NV dynamic state enabled, then the bound graphics pipeline **must** have been created with VkPipelineViewportSwizzleStateCreateInfoNV::viewportCount greater or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdExecuteGeneratedCommandsNV-viewportCount-07493
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT and VK_DYNAMIC_STATE_VIEWPORT_SWIZZLE_NV dynamic states enabled then the viewportCount parameter in the last call to vkCmdSetViewportSwizzleNV **must** be greater than or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdExecuteGeneratedCommandsNV-viewportCount-09421
  If the VK_NV_viewport_swizzle extension is enabled, and a shader object is bound to any graphics stage, then the viewportCount parameter in the last call to vkCmdSetViewportSwizzleNV **must** be greater than or equal to the viewportCount parameter in the last call to vkCmdSetViewportWithCountEXT
- VUID-vkCmdExecuteGeneratedCommandsNV-rasterizationSamples-07494
  If the VK_NV_framebuffer_mixed_samples extension is enabled, and if the current subpass has any color attachments and rasterizationSamples of the last call to vkCmdSetRasterizationSamplesEXT is greater than the number of color samples, then the pipeline sampleShadingEnable **must** be VK_FALSE
- VUID-vkCmdExecuteGeneratedCommandsNV-stippledLineEnable-07495
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT or VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT dynamic states enabled, and if the current stippledLineEnable state is VK_TRUE and the current lineRasterizationMode state is VK_LINE_RASTERIZATION_MODE_RECTANGULAR_KHR, then the stippledRectangularLines feature **must** be enabled
- VUID-vkCmdExecuteGeneratedCommandsNV-stippledLineEnable-07496
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT or VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT dynamic states enabled, and if the current stippledLineEnable state is VK_TRUE and the current lineRasterizationMode state is VK_LINE_RASTERIZATION_MODE_BRESENHAM_KHR, then the stippledBresenhamLines feature **must**

be enabled

- VUID-vkCmdExecuteGeneratedCommandsNV-stippledLineEnable-07497
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT` or `VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT` dynamic states enabled, and if the current `stippledLineEnable` state is `VK_TRUE` and the current `lineRasterizationMode` state is `VK_LINE_RASTERIZATION_MODE_RECTANGULAR_SMOOTH_KHR`, then the `stippledSmoothLines` feature **must** be enabled

- VUID-vkCmdExecuteGeneratedCommandsNV-stippledLineEnable-07498
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT` or `VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT` dynamic states enabled, and if the current `stippledLineEnable` state is `VK_TRUE` and the current `lineRasterizationMode` state is `VK_LINE_RASTERIZATION_MODE_DEFAULT_KHR`, then the `stippledRectangularLines` feature **must** be enabled and VkPhysicalDeviceLimits::`strictLines` **must** be `VK_TRUE`

- VUID-vkCmdExecuteGeneratedCommandsNV-conservativePointAndLineRasterization-07499
  If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_CONSERVATIVE_RASTERIZATION_MODE_EXT` dynamic state enabled, `conservativePointAndLineRasterization` is not supported, and the effective primitive topology output by the last pre-rasterization shader stage is a line or point, then the `conservativeRasterizationMode` set by the last call to vkCmdSetConservativeRasterizationModeEXT **must** be `VK_CONSERVATIVE_RASTERIZATION_MODE_DISABLED_EXT`

- VUID-vkCmdExecuteGeneratedCommandsNV-stage-07073
  If the currently bound pipeline was created with the VkPipelineShaderStageCreateInfo::`stage` member of an element of VkGraphicsPipelineCreateInfo::`pStages` set to `VK_SHADER_STAGE_VERTEX_BIT`, `VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT`, `VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT` or `VK_SHADER_STAGE_GEOMETRY_BIT`, then Mesh Shader Queries **must** not be active

- VUID-vkCmdExecuteGeneratedCommandsNV-None-08877
  If a shader object is bound to the `VK_SHADER_STAGE_FRAGMENT_BIT` stage or a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_ATTACHMENT_FEEDBACK_LOOP_ENABLE_EXT` dynamic state enabled, and the current value of `rasterizerDiscardEnable` is `VK_FALSE`, then vkCmdSetAttachmentFeedbackLoopEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdExecuteGeneratedCommandsNV-None-07850
  If dynamic state was inherited from VkCommandBufferInheritanceViewportScissorInfoNV, it **must** be set in the current command buffer prior to this drawing command

- VUID-vkCmdExecuteGeneratedCommandsNV-None-08684
  If there is no bound graphics pipeline, `vkCmdBindShadersEXT` **must** have been called in the current command buffer with `pStages` with an element of `VK_SHADER_STAGE_VERTEX_BIT`

- VUID-vkCmdExecuteGeneratedCommandsNV-None-08685

If there is no bound graphics pipeline, and the tessellationShader feature is enabled, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT

- VUID-vkCmdExecuteGeneratedCommandsNV-None-08686
  If there is no bound graphics pipeline, and the tessellationShader feature is enabled, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT
- VUID-vkCmdExecuteGeneratedCommandsNV-None-08687
  If there is no bound graphics pipeline, and the geometryShader feature is enabled, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_GEOMETRY_BIT
- VUID-vkCmdExecuteGeneratedCommandsNV-None-08688
  If there is no bound graphics pipeline, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_FRAGMENT_BIT
- VUID-vkCmdExecuteGeneratedCommandsNV-None-08689
  If there is no bound graphics pipeline, and the taskShader feature is enabled, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_TASK_BIT_EXT
- VUID-vkCmdExecuteGeneratedCommandsNV-None-08690
  If there is no bound graphics pipeline, and the meshShader feature is enabled, vkCmdBindShadersEXT **must** have been called in the current command buffer with pStages with an element of VK_SHADER_STAGE_MESH_BIT_EXT
- VUID-vkCmdExecuteGeneratedCommandsNV-None-08693
  If there is no bound graphics pipeline, and at least one of the taskShader and meshShader features is enabled, one of the VK_SHADER_STAGE_VERTEX_BIT or VK_SHADER_STAGE_MESH_BIT_EXT stages **must** have a valid VkShaderEXT bound, and the other **must** have no VkShaderEXT bound
- VUID-vkCmdExecuteGeneratedCommandsNV-None-08694
  If there is no bound graphics pipeline, and both the taskShader and meshShader features are enabled, and a valid VkShaderEXT is bound the to the VK_SHADER_STAGE_MESH_BIT_EXT stage, and that VkShaderEXT was created without the VK_SHADER_CREATE_NO_TASK_SHADER_BIT_EXT flag, a valid VkShaderEXT **must** be bound to the VK_SHADER_STAGE_TASK_BIT_EXT stage
- VUID-vkCmdExecuteGeneratedCommandsNV-None-08695
  If there is no bound graphics pipeline, and both the taskShader and meshShader features are enabled, and a valid VkShaderEXT is bound the to the VK_SHADER_STAGE_MESH_BIT_EXT stage, and that VkShaderEXT was created with the VK_SHADER_CREATE_NO_TASK_SHADER_BIT_EXT flag, there **must** be no VkShaderEXT bound to the VK_SHADER_STAGE_TASK_BIT_EXT stage
- VUID-vkCmdExecuteGeneratedCommandsNV-None-08696
  If there is no bound graphics pipeline, and a valid VkShaderEXT is bound to the VK_SHADER_STAGE_VERTEX_BIT stage, there **must** be no VkShaderEXT bound to either the VK_SHADER_STAGE_TASK_BIT_EXT stage or the VK_SHADER_STAGE_MESH_BIT_EXT stage
- VUID-vkCmdExecuteGeneratedCommandsNV-None-08698
  If any graphics shader is bound which was created with the

VK_SHADER_CREATE_LINK_STAGE_BIT_EXT flag, then all shaders created with the VK_SHADER_CREATE_LINK_STAGE_BIT_EXT flag in the same vkCreateShadersEXT call **must** also be bound

- VUID-vkCmdExecuteGeneratedCommandsNV-None-08699
  If any graphics shader is bound which was created with the VK_SHADER_CREATE_LINK_STAGE_BIT_EXT flag, any stages in between stages whose shaders which did not create a shader with the VK_SHADER_CREATE_LINK_STAGE_BIT_EXT flag as part of the same vkCreateShadersEXT call **must** not have any VkShaderEXT bound
- VUID-vkCmdExecuteGeneratedCommandsNV-None-08878
  All bound graphics shader objects **must** have been created with identical or identically defined push constant ranges
- VUID-vkCmdExecuteGeneratedCommandsNV-None-08879
  All bound graphics shader objects **must** have been created with identical or identically defined arrays of descriptor set layouts
- VUID-vkCmdExecuteGeneratedCommandsNV-colorAttachmentCount-09372
  If the current render pass instance was begun with vkCmdBeginRenderingKHR and a VkRenderingInfo::colorAttachmentCount equal to 1, a color attachment with a resolve mode of VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID, and a fragment shader is bound, it **must** not declare the DepthReplacing or StencilRefReplacingEXT execution modes
- VUID-vkCmdExecuteGeneratedCommandsNV-pDynamicStates-08715
  If the bound graphics pipeline state includes a fragment shader stage, was created with VK_DYNAMIC_STATE_DEPTH_WRITE_ENABLE set in VkPipelineDynamicStateCreateInfo::pDynamicStates, and the fragment shader declares the EarlyFragmentTests execution mode and uses OpDepthAttachmentReadEXT, the depthWriteEnable parameter in the last call to vkCmdSetDepthWriteEnableEXT **must** be VK_FALSE
- VUID-vkCmdExecuteGeneratedCommandsNV-pDynamicStates-08716
  If the bound graphics pipeline state includes a fragment shader stage, was created with VK_DYNAMIC_STATE_STENCIL_WRITE_MASK set in VkPipelineDynamicStateCreateInfo::pDynamicStates, and the fragment shader declares the EarlyFragmentTests execution mode and uses OpStencilAttachmentReadEXT, the writeMask parameter in the last call to vkCmdSetStencilWriteMask **must** be 0
- VUID-vkCmdExecuteGeneratedCommandsNV-None-09116
  If a shader object is bound to any graphics stage or the currently bound graphics pipeline was created with VK_DYNAMIC_STATE_COLOR_WRITE_MASK_EXT, and the format of any color attachment is VK_FORMAT_E5B9G9R9_UFLOAT_PACK32, the corresponding element of the pColorWriteMasks parameter of vkCmdSetColorWriteMaskEXT **must** either include all of VK_COLOR_COMPONENT_R_BIT, VK_COLOR_COMPONENT_G_BIT, and VK_COLOR_COMPONENT_B_BIT, or none of them
- VUID-vkCmdExecuteGeneratedCommandsNV-maxFragmentDualSrcAttachments-09239
  If blending is enabled for any attachment where either the source or destination blend factors for that attachment use the secondary color input, the maximum value of Location for any output attachment statically used in the Fragment Execution Model executed by this command **must** be less than maxFragmentDualSrcAttachments
- VUID-vkCmdExecuteGeneratedCommandsNV-None-09548

If the current render pass was begun with vkCmdBeginRenderingKHR, and there is no shader object bound to any graphics stage, the value of each element of VkRenderingAttachmentLocationInfoKHR::pColorAttachmentLocations set by vkCmdSetRenderingAttachmentLocationsKHR **must** match the value set for the corresponding element in the currently bound pipeline

- VUID-vkCmdExecuteGeneratedCommandsNV-None-09549
  If the current render pass was begun with vkCmdBeginRenderingKHR, and there is no shader object bound to any graphics stage, input attachment index mappings in the currently bound pipeline **must** match those set for the current render pass instance via VkRenderingInputAttachmentIndexInfoKHR

- VUID-vkCmdExecuteGeneratedCommandsNV-None-09642
  If the current render pass was begun with vkCmdBeginRenderingKHR with the VK_RENDERING_ENABLE_LEGACY_DITHERING_BIT_EXT flag, the bound graphics pipeline **must** have been created with VK_PIPELINE_CREATE_2_ENABLE_LEGACY_DITHERING_BIT_EXT

- VUID-vkCmdExecuteGeneratedCommandsNV-None-09643
  If the bound graphics pipeline was created with VK_PIPELINE_CREATE_2_ENABLE_LEGACY_DITHERING_BIT_EXT, the current render pass **must** have begun with vkCmdBeginRenderingKHR with the VK_RENDERING_ENABLE_LEGACY_DITHERING_BIT_EXT flag

- VUID-vkCmdExecuteGeneratedCommandsNV-None-04007
  All vertex input bindings accessed via vertex input variables declared in the vertex shader entry point's interface **must** have either valid or VK_NULL_HANDLE buffers bound

- VUID-vkCmdExecuteGeneratedCommandsNV-None-04008
  If the nullDescriptor feature is not enabled, all vertex input bindings accessed via vertex input variables declared in the vertex shader entry point's interface **must** not be VK_NULL_HANDLE

- VUID-vkCmdExecuteGeneratedCommandsNV-None-02721
  If robustBufferAccess is not enabled, and that pipeline was created without enabling VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_EXT for vertexInputs, then for a given vertex buffer binding, any attribute data fetched **must** be entirely contained within the corresponding vertex buffer binding, as described in Vertex Input Description

- VUID-vkCmdExecuteGeneratedCommandsNV-None-07842
  If there is a shader object bound to the VK_SHADER_STAGE_VERTEX_BIT stage or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_PRIMITIVE_TOPOLOGY dynamic state enabled then vkCmdSetPrimitiveTopologyEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdExecuteGeneratedCommandsNV-dynamicPrimitiveTopologyUnrestricted-07500
  If the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_PRIMITIVE_TOPOLOGY dynamic state enabled and the dynamicPrimitiveTopologyUnrestricted is VK_FALSE, then the primitiveTopology parameter of vkCmdSetPrimitiveTopologyEXT **must** be of the same topology class as the pipeline

VkPipelineInputAssemblyStateCreateInfo::topology state

- VUID-vkCmdExecuteGeneratedCommandsNV-pStrides-04913
  If the bound graphics pipeline was created with the VK_DYNAMIC_STATE_VERTEX_INPUT_BINDING_STRIDE_EXT dynamic state enabled, but without the VK_DYNAMIC_STATE_VERTEX_INPUT_EXT dynamic state enabled, then vkCmdBindVertexBuffers2EXT **must** have been called and not subsequently invalidated in the current command buffer prior to this draw command, and the pStrides parameter of vkCmdBindVertexBuffers2EXT **must** not be NULL

- VUID-vkCmdExecuteGeneratedCommandsNV-None-04914
  If there is a shader object bound to the VK_SHADER_STAGE_VERTEX_BIT stage or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VERTEX_INPUT_EXT dynamic state enabled then vkCmdSetVertexInputEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this draw command

- VUID-vkCmdExecuteGeneratedCommandsNV-Input-07939
  If there is a shader object bound to the VK_SHADER_STAGE_VERTEX_BIT stage or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VERTEX_INPUT_EXT dynamic state enabled then all variables with the Input storage class decorated with Location in the Vertex Execution Model OpEntryPoint **must** contain a location in VkVertexInputAttributeDescription2EXT::location

- VUID-vkCmdExecuteGeneratedCommandsNV-Input-08734
  If there is a shader object bound to the VK_SHADER_STAGE_VERTEX_BIT stage or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VERTEX_INPUT_EXT dynamic state enabled and either legacyVertexAttributes is not enabled or the SPIR-V Type associated with a given Input variable of the corresponding Location in the Vertex Execution Model OpEntryPoint is 64-bit, then the numeric type associated with all Input variables of the corresponding Location in the Vertex Execution Model OpEntryPoint **must** be the same as VkVertexInputAttributeDescription2EXT::format

- VUID-vkCmdExecuteGeneratedCommandsNV-format-08936
  If there is a shader object bound to the VK_SHADER_STAGE_VERTEX_BIT stage or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VERTEX_INPUT_EXT dynamic state enabled and VkVertexInputAttributeDescription2EXT::format has a 64-bit component, then the scalar width associated with all Input variables of the corresponding Location in the Vertex Execution Model OpEntryPoint **must** be 64-bit

- VUID-vkCmdExecuteGeneratedCommandsNV-format-08937
  If there is a shader object bound to the VK_SHADER_STAGE_VERTEX_BIT stage or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VERTEX_INPUT_EXT dynamic state enabled and the scalar width associated with a Location decorated Input variable in the Vertex Execution Model OpEntryPoint is 64-bit, then the corresponding VkVertexInputAttributeDescription2EXT::format **must** have a 64-bit component

- VUID-vkCmdExecuteGeneratedCommandsNV-None-09203
  If there is a shader object bound to the VK_SHADER_STAGE_VERTEX_BIT stage or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_VERTEX_INPUT_EXT dynamic state enabled and VkVertexInputAttributeDescription2EXT::format has a 64-bit component, then all Input variables at the corresponding Location in the Vertex Execution Model OpEntryPoint **must** not use components that are not present in the format

- VUID-vkCmdExecuteGeneratedCommandsNV-None-04875
  If there is a shader object bound to the VK_SHADER_STAGE_VERTEX_BIT stage and the most recent call to vkCmdSetPrimitiveTopologyEXT in the current command buffer set primitiveTopology to VK_PRIMITIVE_TOPOLOGY_PATCH_LIST, or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_PATCH_CONTROL_POINTS_EXT dynamic state enabled then vkCmdSetPatchControlPointsEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdExecuteGeneratedCommandsNV-None-04879
  If there is a shader object bound to the VK_SHADER_STAGE_VERTEX_BIT stage or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_PRIMITIVE_RESTART_ENABLE dynamic state enabled then vkCmdSetPrimitiveRestartEnableEXT **must** have been called and not subsequently invalidated in the current command buffer prior to this drawing command
- VUID-vkCmdExecuteGeneratedCommandsNV-None-09637
  If the primitiveTopologyListRestart feature is not enabled, the topology is VK_PRIMITIVE_TOPOLOGY_POINT_LIST, VK_PRIMITIVE_TOPOLOGY_LINE_LIST, VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST, VK_PRIMITIVE_TOPOLOGY_LINE_LIST_WITH_ADJACENCY, or VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST_WITH_ADJACENCY, there is a shader object bound to the VK_SHADER_STAGE_VERTEX_BIT stage or the bound graphics pipeline state was created with the VK_DYNAMIC_STATE_PRIMITIVE_RESTART_ENABLE dynamic state enabled then vkCmdSetPrimitiveRestartEnableEXT **must** be VK_FALSE
- VUID-vkCmdExecuteGeneratedCommandsNV-stage-06481
  The bound graphics pipeline **must** not have been created with the VkPipelineShaderStageCreateInfo::stage member of an element of VkGraphicsPipelineCreateInfo::pStages set to VK_SHADER_STAGE_TASK_BIT_EXT or VK_SHADER_STAGE_MESH_BIT_EXT
- VUID-vkCmdExecuteGeneratedCommandsNV-None-08885
  There **must** be no shader object bound to either of the VK_SHADER_STAGE_TASK_BIT_EXT or VK_SHADER_STAGE_MESH_BIT_EXT stages
- VUID-vkCmdExecuteGeneratedCommandsNV-commandBuffer-02970
  commandBuffer **must** not be a protected command buffer
- VUID-vkCmdExecuteGeneratedCommandsNV-isPreprocessed-02908
  If isPreprocessed is VK_TRUE then vkCmdPreprocessGeneratedCommandsNV **must** have already been executed on the device, using the same pGeneratedCommandsInfo content as well as the content of the input buffers it references (all except VkGeneratedCommandsInfoNV::preprocessBuffer). Furthermore pGeneratedCommandsInfo`s indirectCommandsLayout **must** have been created with the VK_INDIRECT_COMMANDS_LAYOUT_USAGE_EXPLICIT_PREPROCESS_BIT_NV bit set
- VUID-vkCmdExecuteGeneratedCommandsNV-pipeline-02909
  VkGeneratedCommandsInfoNV::pipeline **must** match the current bound pipeline at VkGeneratedCommandsInfoNV::pipelineBindPoint
- VUID-vkCmdExecuteGeneratedCommandsNV-None-02910
  Transform feedback **must** not be active
- VUID-vkCmdExecuteGeneratedCommandsNV-deviceGeneratedCommands-02911

The `VkPhysicalDeviceDeviceGeneratedCommandsFeaturesNV::deviceGeneratedCommands` feature **must** be enabled

## Valid Usage (Implicit)

- VUID-vkCmdExecuteGeneratedCommandsNV-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdExecuteGeneratedCommandsNV-pGeneratedCommandsInfo-parameter
  `pGeneratedCommandsInfo` **must** be a valid pointer to a valid VkGeneratedCommandsInfoNV structure
- VUID-vkCmdExecuteGeneratedCommandsNV-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdExecuteGeneratedCommandsNV-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics, or compute operations
- VUID-vkCmdExecuteGeneratedCommandsNV-renderpass
  This command **must** only be called inside of a render pass instance
- VUID-vkCmdExecuteGeneratedCommandsNV-videocoding
  This command **must** only be called outside of a video coding scope

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Inside | Outside | Graphics<br>Compute | Action<br>Indirection |

The `VkGeneratedCommandsInfoNV` is defined as:

```
// Provided by VK_NV_device_generated_commands
typedef struct VkGeneratedCommandsInfoNV {
    VkStructureType                     sType;
    const void*                         pNext;
    VkPipelineBindPoint                 pipelineBindPoint;
    VkPipeline                          pipeline;
    VkIndirectCommandsLayoutNV          indirectCommandsLayout;
    uint32_t                            streamCount;
    const VkIndirectCommandsStreamNV*   pStreams;
    uint32_t                            sequencesCount;
    VkBuffer                            preprocessBuffer;
    VkDeviceSize                        preprocessOffset;
    VkDeviceSize                        preprocessSize;
    VkBuffer                            sequencesCountBuffer;
    VkDeviceSize                        sequencesCountOffset;
    VkBuffer                            sequencesIndexBuffer;
    VkDeviceSize                        sequencesIndexOffset;
} VkGeneratedCommandsInfoNV;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `pipelineBindPoint` is the VkPipelineBindPoint used for the `pipeline`.
- `pipeline` is the VkPipeline used in the generation and execution process.
- `indirectCommandsLayout` is the VkIndirectCommandsLayoutNV that provides the command sequence to generate.
- `streamCount` defines the number of input streams
- `pStreams` is a pointer to an array of `streamCount` VkIndirectCommandsStreamNV structures providing the input data for the tokens used in `indirectCommandsLayout`.
- `sequencesCount` is the maximum number of sequences to reserve. If `sequencesCountBuffer` is VK_NULL_HANDLE, this is also the actual number of sequences generated.
- `preprocessBuffer` is the VkBuffer that is used for preprocessing the input data for execution. If this structure is used with vkCmdExecuteGeneratedCommandsNV with its `isPreprocessed` set to `VK_TRUE`, then the preprocessing step is skipped and data in this buffer will not be modified. The contents and the layout of this buffer are opaque to applications and **must** not be modified outside functions related to device-generated commands or copied to another buffer for reuse.
- `preprocessOffset` is the byte offset into `preprocessBuffer` where the preprocessed data is stored.
- `preprocessSize` is the maximum byte size within the `preprocessBuffer` after the `preprocessOffset` that is available for preprocessing.
- `sequencesCountBuffer` is a `VkBuffer` in which the actual number of sequences is provided as single `uint32_t` value.
- `sequencesCountOffset` is the byte offset into `sequencesCountBuffer` where the count value is stored.
- `sequencesIndexBuffer` is a `VkBuffer` that encodes the used sequence indices as `uint32_t` array.

- sequencesIndexOffset is the byte offset into sequencesIndexBuffer where the index values start.

## Valid Usage

- VUID-VkGeneratedCommandsInfoNV-pipeline-02912
  The provided pipeline **must** match the pipeline bound at execution time
- VUID-VkGeneratedCommandsInfoNV-indirectCommandsLayout-02913
  If the indirectCommandsLayout uses a token of VK_INDIRECT_COMMANDS_TOKEN_TYPE_SHADER_GROUP_NV, then the pipeline **must** have been created with multiple shader groups
- VUID-VkGeneratedCommandsInfoNV-indirectCommandsLayout-02914
  If the indirectCommandsLayout uses a token of VK_INDIRECT_COMMANDS_TOKEN_TYPE_SHADER_GROUP_NV, then the pipeline **must** have been created with VK_PIPELINE_CREATE_INDIRECT_BINDABLE_BIT_NV set in VkGraphicsPipelineCreateInfo::flags
- VUID-VkGeneratedCommandsInfoNV-indirectCommandsLayout-02915
  If the indirectCommandsLayout uses a token of VK_INDIRECT_COMMANDS_TOKEN_TYPE_PUSH_CONSTANT_NV, then the pipeline's VkPipelineLayout **must** match the VkIndirectCommandsLayoutTokenNV::pushconstantPipelineLayout
- VUID-VkGeneratedCommandsInfoNV-streamCount-02916
  streamCount **must** match the indirectCommandsLayout's streamCount
- VUID-VkGeneratedCommandsInfoNV-pipelineBindPoint-09084
  If pipelineBindPoint is of type VK_PIPELINE_BIND_POINT_COMPUTE, then the pipeline **must** have been created with the flag VK_PIPELINE_CREATE_INDIRECT_BINDABLE_BIT_NV
- VUID-VkGeneratedCommandsInfoNV-pipelineBindPoint-09085
  If pipelineBindPoint is of type VK_PIPELINE_BIND_POINT_COMPUTE, then the pipeline **must** have been created with a VkComputePipelineIndirectBufferInfoNV structure specifying a valid address where its metadata will be saved
- VUID-VkGeneratedCommandsInfoNV-pipelineBindPoint-09086
  If pipelineBindPoint is of type VK_PIPELINE_BIND_POINT_COMPUTE, then vkCmdUpdatePipelineIndirectBufferNV **must** have been called on that pipeline to save its metadata to a device address
- VUID-VkGeneratedCommandsInfoNV-pipelineBindPoint-09087
  If pipelineBindPoint is of type VK_PIPELINE_BIND_POINT_COMPUTE, and if VK_INDIRECT_COMMANDS_TOKEN_TYPE_PIPELINE_NV is used, then pipeline **must** be VK_NULL_HANDLE
- VUID-VkGeneratedCommandsInfoNV-sequencesCount-02917
  sequencesCount **must** be less or equal to VkPhysicalDeviceDeviceGeneratedCommandsPropertiesNV::maxIndirectSequenceCount and VkGeneratedCommandsMemoryRequirementsInfoNV::maxSequencesCount that was used to determine the preprocessSize
- VUID-VkGeneratedCommandsInfoNV-preprocessBuffer-02918
  preprocessBuffer **must** have the VK_BUFFER_USAGE_INDIRECT_BUFFER_BIT bit set in its usage

flag

- VUID-VkGeneratedCommandsInfoNV-preprocessOffset-02919
  preprocessOffset **must** be aligned to VkPhysicalDeviceDeviceGeneratedCommandsPropertiesNV::minIndirectCommandsBufferOffsetAlignment
- VUID-VkGeneratedCommandsInfoNV-preprocessBuffer-02971
  If preprocessBuffer is non-sparse then it **must** be bound completely and contiguously to a single VkDeviceMemory object
- VUID-VkGeneratedCommandsInfoNV-preprocessSize-02920
  preprocessSize **must** be at least equal to the memory requirement`s size returned by vkGetGeneratedCommandsMemoryRequirementsNV using the matching inputs (indirectCommandsLayout, ...) as within this structure
- VUID-VkGeneratedCommandsInfoNV-sequencesCountBuffer-02921
  sequencesCountBuffer **can** be set if the actual used count of sequences is sourced from the provided buffer. In that case the sequencesCount serves as upper bound
- VUID-VkGeneratedCommandsInfoNV-sequencesCountBuffer-02922
  If sequencesCountBuffer is not VK_NULL_HANDLE, its usage flag **must** have the VK_BUFFER_USAGE_INDIRECT_BUFFER_BIT bit set
- VUID-VkGeneratedCommandsInfoNV-sequencesCountBuffer-02923
  If sequencesCountBuffer is not VK_NULL_HANDLE, sequencesCountOffset **must** be aligned to VkPhysicalDeviceDeviceGeneratedCommandsPropertiesNV::minSequencesCountBufferOffsetAlignment
- VUID-VkGeneratedCommandsInfoNV-sequencesCountBuffer-02972
  If sequencesCountBuffer is not VK_NULL_HANDLE and is non-sparse then it **must** be bound completely and contiguously to a single VkDeviceMemory object
- VUID-VkGeneratedCommandsInfoNV-sequencesIndexBuffer-02924
  If indirectCommandsLayout's VK_INDIRECT_COMMANDS_LAYOUT_USAGE_INDEXED_SEQUENCES_BIT_NV is set, sequencesIndexBuffer **must** be set otherwise it **must** be VK_NULL_HANDLE
- VUID-VkGeneratedCommandsInfoNV-sequencesIndexBuffer-02925
  If sequencesIndexBuffer is not VK_NULL_HANDLE, its usage flag **must** have the VK_BUFFER_USAGE_INDIRECT_BUFFER_BIT bit set
- VUID-VkGeneratedCommandsInfoNV-sequencesIndexBuffer-02926
  If sequencesIndexBuffer is not VK_NULL_HANDLE, sequencesIndexOffset **must** be aligned to VkPhysicalDeviceDeviceGeneratedCommandsPropertiesNV::minSequencesIndexBufferOffsetAlignment
- VUID-VkGeneratedCommandsInfoNV-sequencesIndexBuffer-02973
  If sequencesIndexBuffer is not VK_NULL_HANDLE and is non-sparse then it **must** be bound completely and contiguously to a single VkDeviceMemory object
- VUID-VkGeneratedCommandsInfoNV-indirectCommandsLayout-07078
  If the indirectCommandsLayout uses a token of VK_INDIRECT_COMMANDS_TOKEN_TYPE_DRAW_TASKS_NV, then the pipeline **must** contain a shader stage using the MeshNV Execution Model
- VUID-VkGeneratedCommandsInfoNV-indirectCommandsLayout-07079

If the `indirectCommandsLayout` uses a token of `VK_INDIRECT_COMMANDS_TOKEN_TYPE_DRAW_MESH_TASKS_NV`, then the `pipeline` **must** contain a shader stage using the `MeshEXT Execution Model`

## Valid Usage (Implicit)

- VUID-VkGeneratedCommandsInfoNV-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_GENERATED_COMMANDS_INFO_NV`
- VUID-VkGeneratedCommandsInfoNV-pNext-pNext
  `pNext` **must** be `NULL`
- VUID-VkGeneratedCommandsInfoNV-pipelineBindPoint-parameter
  `pipelineBindPoint` **must** be a valid VkPipelineBindPoint value
- VUID-VkGeneratedCommandsInfoNV-pipeline-parameter
  If `pipeline` is not VK_NULL_HANDLE, `pipeline` **must** be a valid VkPipeline handle
- VUID-VkGeneratedCommandsInfoNV-indirectCommandsLayout-parameter
  `indirectCommandsLayout` **must** be a valid VkIndirectCommandsLayoutNV handle
- VUID-VkGeneratedCommandsInfoNV-pStreams-parameter
  `pStreams` **must** be a valid pointer to an array of `streamCount` valid VkIndirectCommandsStreamNV structures
- VUID-VkGeneratedCommandsInfoNV-preprocessBuffer-parameter
  `preprocessBuffer` **must** be a valid VkBuffer handle
- VUID-VkGeneratedCommandsInfoNV-sequencesCountBuffer-parameter
  If `sequencesCountBuffer` is not VK_NULL_HANDLE, `sequencesCountBuffer` **must** be a valid VkBuffer handle
- VUID-VkGeneratedCommandsInfoNV-sequencesIndexBuffer-parameter
  If `sequencesIndexBuffer` is not VK_NULL_HANDLE, `sequencesIndexBuffer` **must** be a valid VkBuffer handle
- VUID-VkGeneratedCommandsInfoNV-streamCount-arraylength
  `streamCount` **must** be greater than `0`
- VUID-VkGeneratedCommandsInfoNV-commonparent
  Each of `indirectCommandsLayout`, `pipeline`, `preprocessBuffer`, `sequencesCountBuffer`, and `sequencesIndexBuffer` that are valid handles of non-ignored parameters **must** have been created, allocated, or retrieved from the same VkDevice

Referencing the functions defined in Indirect Commands Layout, `vkCmdExecuteGeneratedCommandsNV` behaves as:

```
uint32_t sequencesCount = sequencesCountBuffer ?
      min(maxSequencesCount, sequencesCountBuffer.load_uint32(sequencesCountOffset) :
      maxSequencesCount;
```

```
cmdProcessAllSequences(commandBuffer, pipeline,
                       indirectCommandsLayout, pIndirectCommandsStreams,
                       sequencesCount,
                       sequencesIndexBuffer, sequencesIndexOffset);

// The stateful commands within indirectCommandsLayout will not
// affect the state of subsequent commands in the target
// command buffer (cmd)
```

**NOTE** It is important to note that the values of all state related to the `pipelineBindPoint` used are undefined after this command.

Commands **can** be preprocessed prior execution using the following command:

```
// Provided by VK_NV_device_generated_commands
void vkCmdPreprocessGeneratedCommandsNV(
    VkCommandBuffer                             commandBuffer,
    const VkGeneratedCommandsInfoNV*            pGeneratedCommandsInfo);
```

- `commandBuffer` is the command buffer which does the preprocessing.
- `pGeneratedCommandsInfo` is a pointer to a VkGeneratedCommandsInfoNV structure containing parameters affecting the preprocessing step.

### Valid Usage

- VUID-vkCmdPreprocessGeneratedCommandsNV-commandBuffer-02974
  `commandBuffer` **must** not be a protected command buffer
- VUID-vkCmdPreprocessGeneratedCommandsNV-pGeneratedCommandsInfo-02927
  `pGeneratedCommandsInfo`s `indirectCommandsLayout` **must** have been created with the `VK_INDIRECT_COMMANDS_LAYOUT_USAGE_EXPLICIT_PREPROCESS_BIT_NV` bit set
- VUID-vkCmdPreprocessGeneratedCommandsNV-deviceGeneratedCommands-02928
  The `VkPhysicalDeviceDeviceGeneratedCommandsFeaturesNV::deviceGeneratedCommands` feature **must** be enabled

### Valid Usage (Implicit)

- VUID-vkCmdPreprocessGeneratedCommandsNV-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdPreprocessGeneratedCommandsNV-pGeneratedCommandsInfo-parameter
  `pGeneratedCommandsInfo` **must** be a valid pointer to a valid VkGeneratedCommandsInfoNV structure
- VUID-vkCmdPreprocessGeneratedCommandsNV-commandBuffer-recording
  `commandBuffer` **must** be in the recording state

- VUID-vkCmdPreprocessGeneratedCommandsNV-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics, or compute operations
- VUID-vkCmdPreprocessGeneratedCommandsNV-renderpass
  This command **must** only be called outside of a render pass instance
- VUID-vkCmdPreprocessGeneratedCommandsNV-videocoding
  This command **must** only be called outside of a video coding scope

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Outside | Outside | Graphics<br>Compute | Action |

The bound descriptor sets and push constants that will be used with indirect command generation for the compute pipelines **must** already be specified at the time of preprocessing commands with vkCmdPreprocessGeneratedCommandsNV. They **must** not change until the execution of indirect commands is submitted with vkCmdExecuteGeneratedCommandsNV.

If push constants for the compute pipeline are also specified in the VkGeneratedCommandsInfoNV::`indirectCommandsLayout` with `VK_INDIRECT_COMMANDS_TOKEN_TYPE_PUSH_CONSTANT_NV` token, then those values override the push constants that were previously pushed for the compute pipeline.

# Chapter 33. Sparse Resources

As documented in Resource Memory Association, `VkBuffer` and `VkImage` resources in Vulkan **must** be bound completely and contiguously to a single `VkDeviceMemory` object. This binding **must** be done before the resource is used, and the binding is immutable for the lifetime of the resource.

*Sparse resources* relax these restrictions and provide these additional features:

- Sparse resources **can** be bound non-contiguously to one or more `VkDeviceMemory` allocations.
- Sparse resources **can** be re-bound to different memory allocations over the lifetime of the resource.
- Sparse resources **can** have descriptors generated and used orthogonally with memory binding commands.

## 33.1. Sparse Resource Features

Sparse resources have several features that **must** be enabled explicitly at resource creation time. The features are enabled by including bits in the `flags` parameter of VkImageCreateInfo or VkBufferCreateInfo. Each feature also has one or more corresponding feature enables specified in VkPhysicalDeviceFeatures.

- The `sparseBinding` feature is the base, and provides the following capabilities:
  - Resources **can** be bound at some defined (sparse block) granularity.
  - The entire resource **must** be bound to memory before use regardless of regions actually accessed.
  - No specific mapping of image region to memory offset is defined, i.e. the location that each texel corresponds to in memory is implementation-dependent.
  - Sparse buffers have a well-defined mapping of buffer range to memory range, where an offset into a range of the buffer that is bound to a single contiguous range of memory corresponds to an identical offset within that range of memory.
  - Requested via the `VK_IMAGE_CREATE_SPARSE_BINDING_BIT` and `VK_BUFFER_CREATE_SPARSE_BINDING_BIT` bits.
  - A sparse image created using `VK_IMAGE_CREATE_SPARSE_BINDING_BIT` (but not `VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT`) supports all formats that non-sparse usage supports, and supports both `VK_IMAGE_TILING_OPTIMAL` and `VK_IMAGE_TILING_LINEAR` tiling.
- *Sparse Residency* builds on (and requires) the `sparseBinding` feature. It includes the following capabilities:
  - Resources do not have to be completely bound to memory before use on the device.
  - Images have a prescribed sparse image block layout, allowing specific rectangular regions of the image to be bound to specific offsets in memory allocations.
  - Consistency of access to unbound regions of the resource is defined by the absence or presence of `VkPhysicalDeviceSparseProperties::residencyNonResidentStrict`. If this property is present, accesses to unbound regions of the resource are well defined and behave as if the

data bound is populated with all zeros; writes are discarded. When this property is absent, accesses are considered safe, but reads will return undefined values.

- Requested via the VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT and VK_BUFFER_CREATE_SPARSE_RESIDENCY_BIT bits.
- Sparse residency support is advertised on a finer grain via the following features:
  - The sparseResidencyBuffer feature provides support for creating VkBuffer objects with the VK_BUFFER_CREATE_SPARSE_RESIDENCY_BIT.
  - The sparseResidencyImage2D feature provides support for creating 2D single-sampled VkImage objects with VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT.
  - The sparseResidencyImage3D feature provides support for creating 3D VkImage objects with VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT.
  - The sparseResidency2Samples feature provides support for creating 2D VkImage objects with 2 samples and VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT.
  - The sparseResidency4Samples feature provides support for creating 2D VkImage objects with 4 samples and VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT.
  - The sparseResidency8Samples feature provides support for creating 2D VkImage objects with 8 samples and VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT.
  - The sparseResidency16Samples feature provides support for creating 2D VkImage objects with 16 samples and VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT.

    Implementations supporting sparseResidencyImage2D are only **required** to support sparse 2D, single-sampled images. Support for sparse 3D and MSAA images is **optional** and **can** be enabled via sparseResidencyImage3D, sparseResidency2Samples, sparseResidency4Samples, sparseResidency8Samples, and sparseResidency16Samples.
- A sparse image created using VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT supports all non-compressed color formats with power-of-two element size that non-sparse usage supports. Additional formats **may** also be supported and **can** be queried via vkGetPhysicalDeviceSparseImageFormatProperties. VK_IMAGE_TILING_LINEAR tiling is not supported.

- The sparseResidencyAliased feature provides the following capability that **can** be enabled per resource:

  Allows physical memory ranges to be shared between multiple locations in the same sparse resource or between multiple sparse resources, with each binding of a memory location observing a consistent interpretation of the memory contents.

  See Sparse Memory Aliasing for more information.

## 33.2. Sparse Buffers and Fully-Resident Images

Both VkBuffer and VkImage objects created with the VK_IMAGE_CREATE_SPARSE_BINDING_BIT or VK_BUFFER_CREATE_SPARSE_BINDING_BIT bits **can** be thought of as a linear region of address space. In the VkImage case if VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT is not used, this linear region is entirely

opaque, meaning that there is no application-visible mapping between texel location and memory offset.

Unless `VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT` or `VK_BUFFER_CREATE_SPARSE_RESIDENCY_BIT` are also used, the entire resource **must** be bound to one or more `VkDeviceMemory` objects before use.

### 33.2.1. Sparse Buffer and Fully-Resident Image Block Size

The sparse block size in bytes for sparse buffers and fully-resident images is reported as `VkMemoryRequirements::alignment`. `alignment` represents both the memory alignment requirement and the binding granularity (in bytes) for sparse resources.

## 33.3. Sparse Partially-Resident Buffers

`VkBuffer` objects created with the `VK_BUFFER_CREATE_SPARSE_RESIDENCY_BIT` bit allow the buffer to be made only partially resident. Partially resident `VkBuffer` objects are allocated and bound identically to `VkBuffer` objects using only the `VK_BUFFER_CREATE_SPARSE_BINDING_BIT` feature. The only difference is the ability for some regions of the buffer to be unbound during device use.

## 33.4. Sparse Partially-Resident Images

`VkImage` objects created with the `VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT` bit allow specific rectangular regions of the image called sparse image blocks to be bound to specific ranges of memory. This allows the application to manage residency at either image subresource or sparse image block granularity. Each image subresource (outside of the mip tail) starts on a sparse block boundary and has dimensions that are integer multiples of the corresponding dimensions of the sparse image block.

**NOTE**

Applications **can** use these types of images to control LOD based on total memory consumption. If memory pressure becomes an issue the application **can** unbind and disable specific mipmap levels of images without having to recreate resources or modify texel data of unaffected levels.

The application **can** also use this functionality to access subregions of the image in a “megatexture” fashion. The application **can** create a large image and only populate the region of the image that is currently being used in the scene.

### 33.4.1. Accessing Unbound Regions

The following member of `VkPhysicalDeviceSparseProperties` affects how data in unbound regions of sparse resources are handled by the implementation:

- `residencyNonResidentStrict`

If this property is not present, reads of unbound regions of the image will return undefined values. Both reads and writes are still considered *safe* and will not affect other resources or populated regions of the image.

If this property is present, all reads of unbound regions of the image will behave as if the region was bound to memory populated with all zeros; writes will be discarded.

Image operations performed on unbound memory **may** still alter some component values in the natural way for those accesses, e.g. substituting a value of one for alpha in formats that do not have an alpha component.

> Example: Reading the alpha component of an unbacked `VK_FORMAT_R8_UNORM` image will return a value of 1.0f.

See Physical Device Enumeration for instructions for retrieving physical device properties.

> **Implementor's Note**
>
> For implementations that **cannot** natively handle access to unbound regions of a resource, the implementation **may** allocate and bind memory to the unbound regions. Reads and writes to unbound regions will access the implementation-managed memory instead.
>
> Given that the values resulting from reads of unbound regions are undefined in this scenario, implementations **may** use the same physical memory for all unbound regions of multiple resources within the same process.

### 33.4.2. Mip Tail Regions

Sparse images created using `VK_IMAGE_CREATE_SPARSE_BINDING_BIT` (without also using `VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT`) have no specific mapping of image region or image subresource to memory offset defined, so the entire image **can** be thought of as a linear opaque address region. However, images created with `VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT` do have a prescribed sparse image block layout, and hence each image subresource **must** start on a sparse block boundary. Within each array layer, the set of mip levels that have a smaller size than the sparse block size in bytes are grouped together into a *mip tail region*.

If the `VK_SPARSE_IMAGE_FORMAT_ALIGNED_MIP_SIZE_BIT` flag is present in the `flags` member of `VkSparseImageFormatProperties`, for the image's `format`, then any mip level which has dimensions that are not integer multiples of the corresponding dimensions of the sparse image block, and all subsequent mip levels, are also included in the mip tail region.

The following member of `VkPhysicalDeviceSparseProperties` **may** affect how the implementation places mip levels in the mip tail region:

- `residencyAlignedMipSize`

Each mip tail region is bound to memory as an opaque region (i.e. **must** be bound using a VkSparseImageOpaqueMemoryBindInfo structure) and **may** be of a size greater than or equal to the sparse block size in bytes. This size is guaranteed to be an integer multiple of the sparse block size in bytes.

An implementation **may** choose to allow each array-layer's mip tail region to be bound to memory independently or require that all array-layer's mip tail regions be treated as one. This is dictated by `VK_SPARSE_IMAGE_FORMAT_SINGLE_MIPTAIL_BIT` in `VkSparseImageMemoryRequirements::flags`.

The following diagrams depict how `VK_SPARSE_IMAGE_FORMAT_ALIGNED_MIP_SIZE_BIT` and `VK_SPARSE_IMAGE_FORMAT_SINGLE_MIPTAIL_BIT` alter memory usage and requirements.

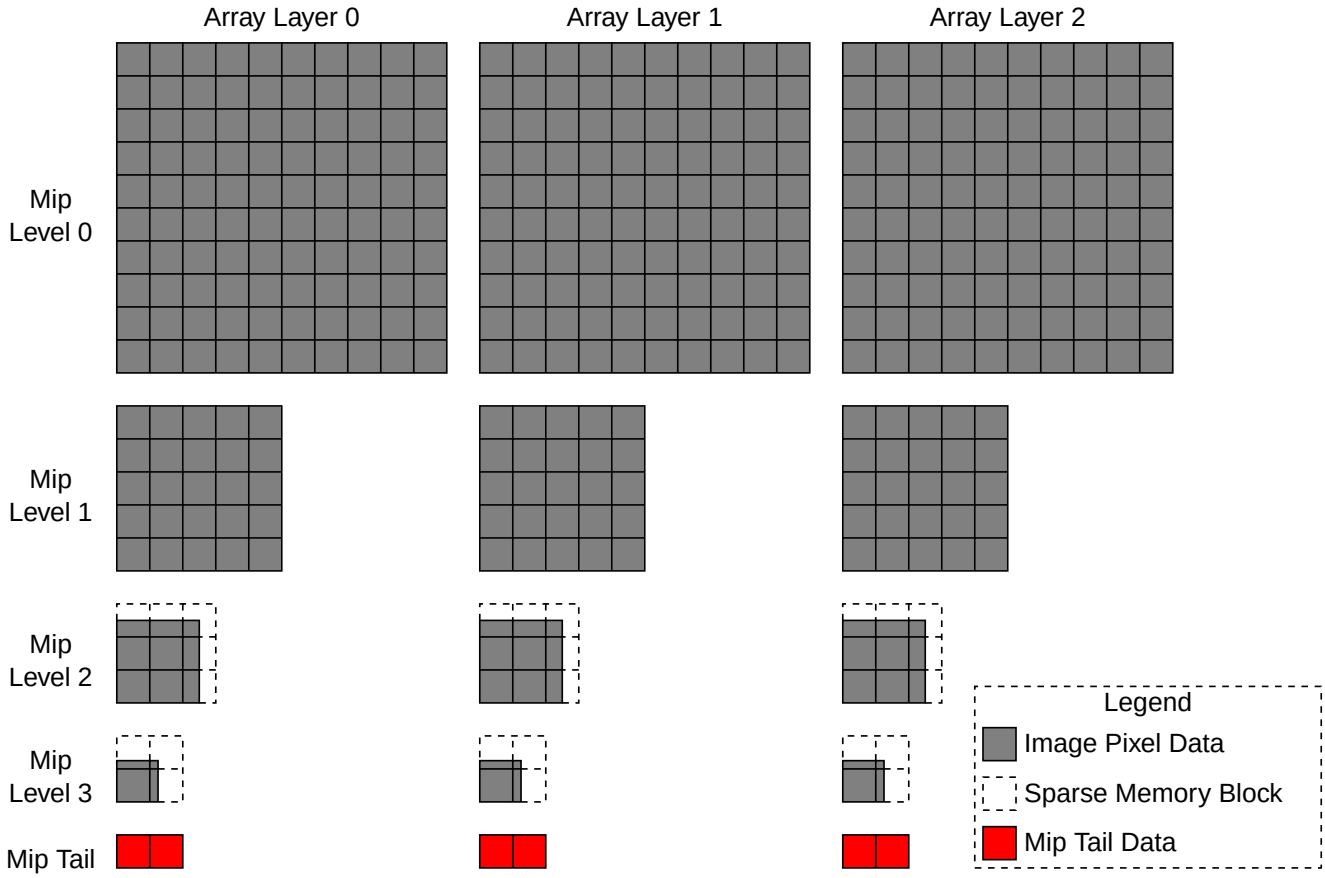



*Figure 22. Sparse Image*

In the absence of `VK_SPARSE_IMAGE_FORMAT_ALIGNED_MIP_SIZE_BIT` and `VK_SPARSE_IMAGE_FORMAT_SINGLE_MIPTAIL_BIT`, each array layer contains a mip tail region containing texel data for all mip levels smaller than the sparse image block in any dimension.

Mip levels that are as large or larger than a sparse image block in all dimensions **can** be bound individually. Right-edges and bottom-edges of each level are allowed to have partially used sparse blocks. Any bound partially-used-sparse-blocks **must** still have their full sparse block size in bytes allocated in memory.

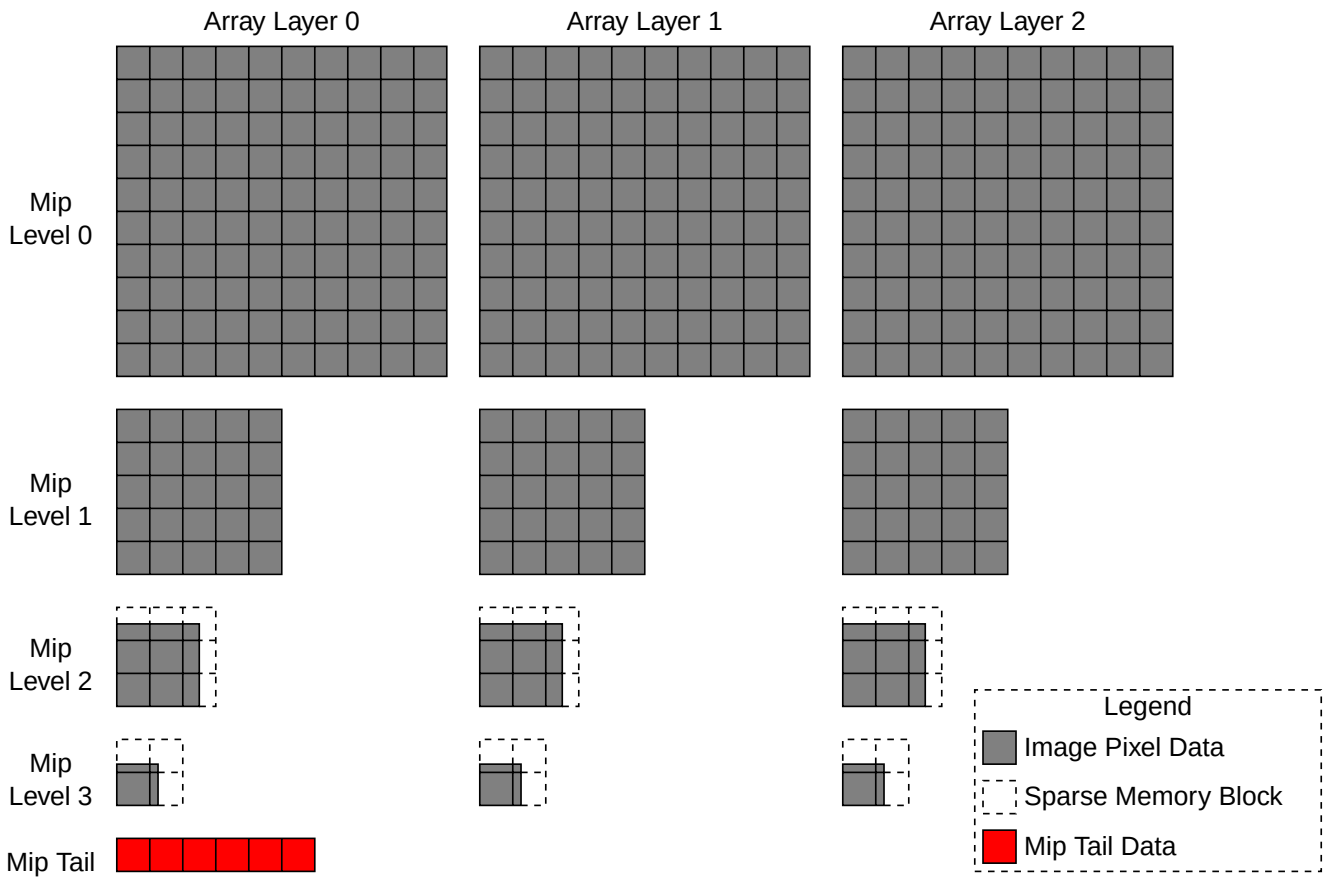



*Figure 23. Sparse Image with Single Mip Tail*

When `VK_SPARSE_IMAGE_FORMAT_SINGLE_MIPTAIL_BIT` is present all array layers will share a single mip tail region.

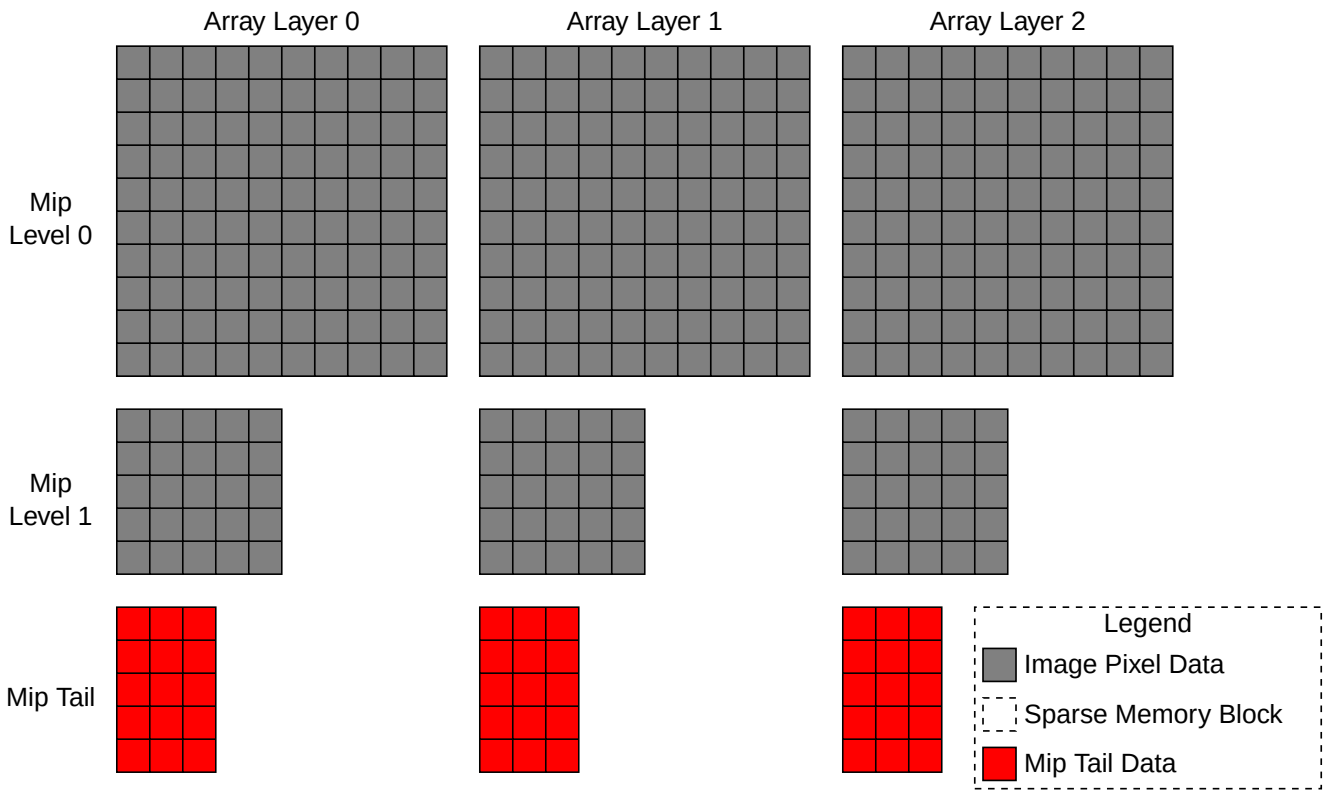



*Figure 24. Sparse Image with Aligned Mip Size*

**NOTE** The mip tail regions are presented here in 2D arrays simply for figure size reasons.

> Each mip tail is logically a single array of sparse blocks with an implementation-dependent mapping of texels or compressed texel blocks to sparse blocks.

When VK_SPARSE_IMAGE_FORMAT_ALIGNED_MIP_SIZE_BIT is present the first mip level that would contain partially used sparse blocks begins the mip tail region. This level and all subsequent levels are placed in the mip tail. Only the first N mip levels whose dimensions are an exact multiple of the sparse image block dimensions **can** be bound and unbound on a sparse block basis.

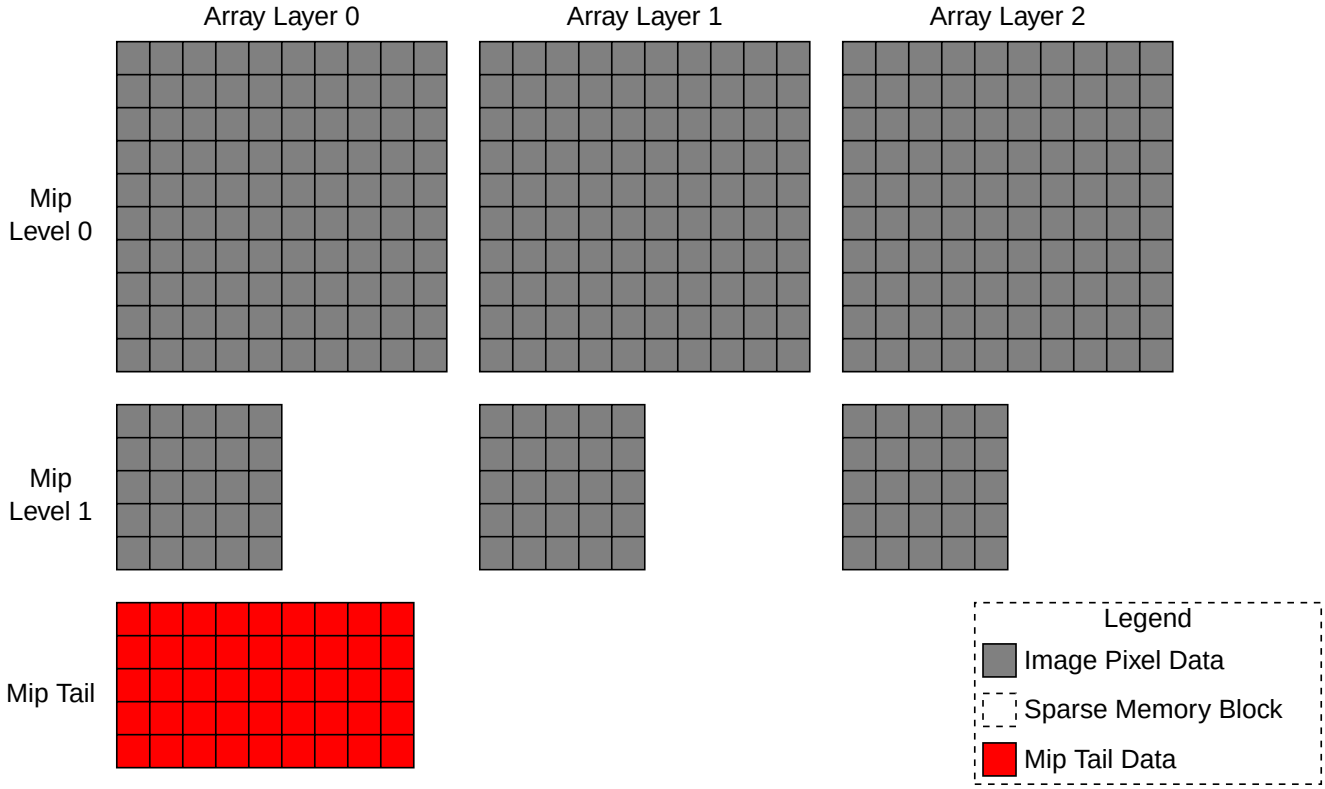



*Figure 25. Sparse Image with Aligned Mip Size and Single Mip Tail*

> **NOTE** The mip tail region is presented here in a 2D array simply for figure size reasons. It is logically a single array of sparse blocks with an implementation-dependent mapping of texels or compressed texel blocks to sparse blocks.

When both VK_SPARSE_IMAGE_FORMAT_ALIGNED_MIP_SIZE_BIT and VK_SPARSE_IMAGE_FORMAT_SINGLE_MIPTAIL_BIT are present the constraints from each of these flags are in effect.

### 33.4.3. Standard Sparse Image Block Shapes

Standard sparse image block shapes define a standard set of dimensions for sparse image blocks that depend on the format of the image. Layout of texels or compressed texel blocks within a sparse image block is implementation-dependent. All currently defined standard sparse image block shapes are 64 KB in size.

For block-compressed formats (e.g. VK_FORMAT_BC5_UNORM_BLOCK), the texel size is the size of the compressed texel block (e.g. 128-bit for BC5) thus the dimensions of the standard sparse image block shapes apply in terms of compressed texel blocks.

| | |
|---|---|
| **NOTE** | For block-compressed formats, the dimensions of a sparse image block in terms of texels **can** be calculated by multiplying the sparse image block dimensions by the compressed texel block dimensions. |

*Table 47. Standard Sparse Image Block Shapes (Single Sample)*

| TEXEL SIZE (bits) | Block Shape (2D) | Block Shape (3D) |
|---|---|---|
| **8-Bit** | 256 × 256 × 1 | 64 × 32 × 32 |
| **16-Bit** | 256 × 128 × 1 | 32 × 32 × 32 |
| **32-Bit** | 128 × 128 × 1 | 32 × 32 × 16 |
| **64-Bit** | 128 × 64 × 1 | 32 × 16 × 16 |
| **128-Bit** | 64 × 64 × 1 | 16 × 16 × 16 |

*Table 48. Standard Sparse Image Block Shapes (MSAA)*

| TEXEL SIZE (bits) | Block Shape (2X) | Block Shape (4X) | Block Shape (8X) | Block Shape (16X) |
|---|---|---|---|---|
| **8-Bit** | 128 × 256 × 1 | 128 × 128 × 1 | 64 × 128 × 1 | 64 × 64 × 1 |
| **16-Bit** | 128 × 128 × 1 | 128 × 64 × 1 | 64 × 64 × 1 | 64 × 32 × 1 |
| **32-Bit** | 64 × 128 × 1 | 64 × 64 × 1 | 32 × 64 × 1 | 32 × 32 × 1 |
| **64-Bit** | 64 × 64 × 1 | 64 × 32 × 1 | 32 × 32 × 1 | 32 × 16 × 1 |
| **128-Bit** | 32 × 64 × 1 | 32 × 32 × 1 | 16 × 32 × 1 | 16 × 16 × 1 |

Implementations that support the standard sparse image block shape for all formats listed in the Standard Sparse Image Block Shapes (Single Sample) and Standard Sparse Image Block Shapes (MSAA) tables **may** advertise the following `VkPhysicalDeviceSparseProperties`:

- `residencyStandard2DBlockShape`
- `residencyStandard2DMultisampleBlockShape`
- `residencyStandard3DBlockShape`

Reporting each of these features does *not* imply that all possible image types are supported as sparse. Instead, this indicates that no supported sparse image of the corresponding type will use custom sparse image block dimensions for any formats that have a corresponding standard sparse image block shape.

### 33.4.4. Custom Sparse Image Block Shapes

An implementation that does not support a standard image block shape for a particular sparse partially-resident image **may** choose to support a custom sparse image block shape for it instead. The dimensions of such a custom sparse image block shape are reported in `VkSparseImageFormatProperties::imageGranularity`. As with standard sparse image block shapes, the size in bytes of the custom sparse image block shape will be reported in `VkMemoryRequirements::alignment`.

Custom sparse image block dimensions are reported through `vkGetPhysicalDeviceSparseImageFormatProperties` and `vkGetImageSparseMemoryRequirements`.

An implementation **must** not support both the standard sparse image block shape and a custom

sparse image block shape for the same image. The standard sparse image block shape **must** be used if it is supported.

### 33.4.5. Multiple Aspects

Partially resident images are allowed to report separate sparse properties for different aspects of the image. One example is for depth/stencil images where the implementation separates the depth and stencil data into separate planes. Another reason for multiple aspects is to allow the application to manage memory allocation for implementation-private *metadata* associated with the image. See the figure below:

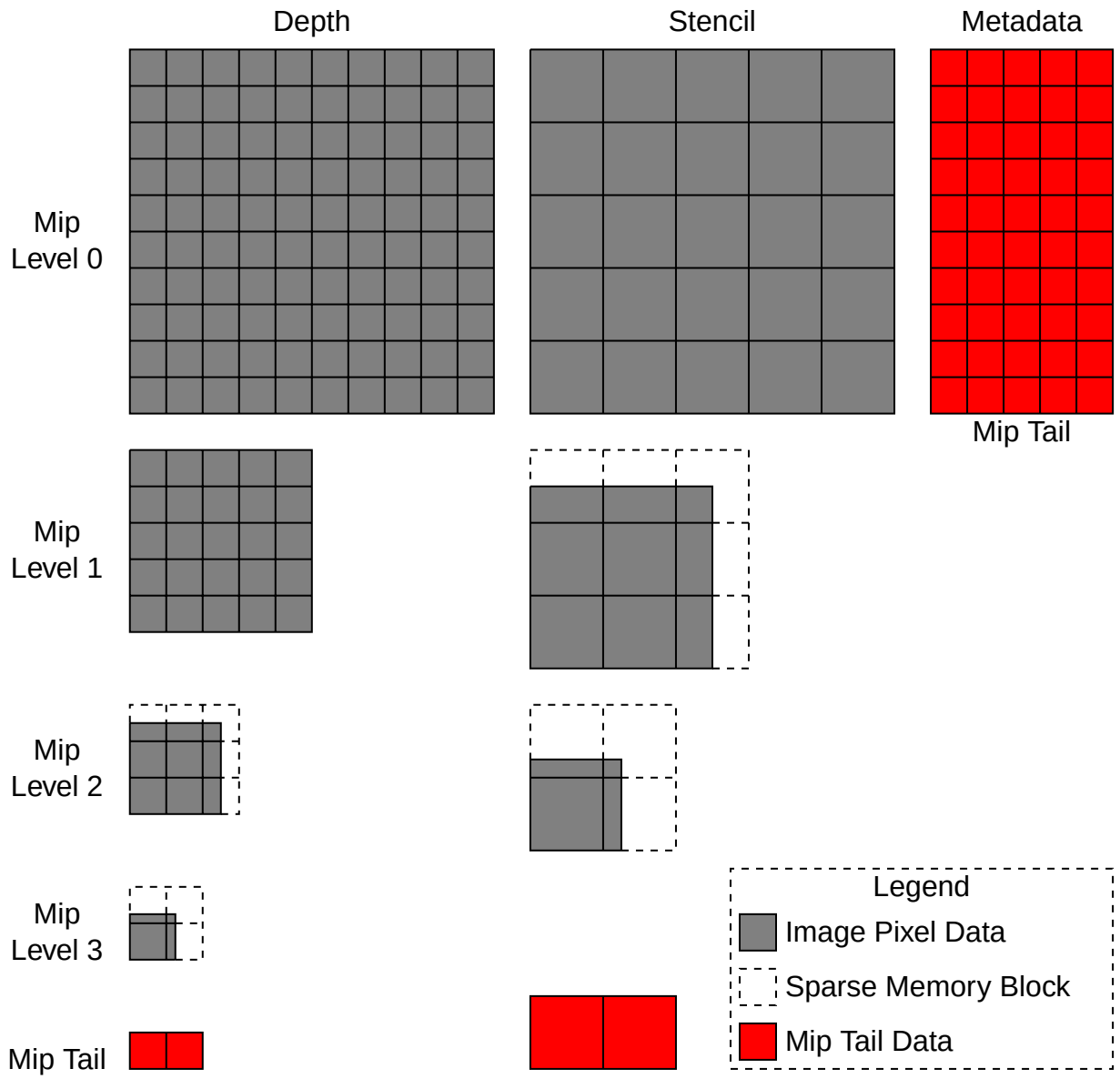



*Figure 26. Multiple Aspect Sparse Image*

| | |
|---|---|
| **NOTE** | The mip tail regions are presented here in 2D arrays simply for figure size reasons. Each mip tail is logically a single array of sparse blocks with an implementation-dependent mapping of texels or compressed texel blocks to sparse blocks. |

In the figure above the depth, stencil, and metadata aspects all have unique sparse properties. The per-texel stencil data is ¼ the size of the depth data, hence the stencil sparse blocks include 4 × the number of texels. The sparse block size in bytes for all of the aspects is identical and defined by

`VkMemoryRequirements::alignment`.

**Metadata**

The metadata aspect of an image has the following constraints:

- All metadata is reported in the mip tail region of the metadata aspect.
- All metadata **must** be bound prior to device use of the sparse image.

## 33.5. Sparse Memory Aliasing

By default sparse resources have the same aliasing rules as non-sparse resources. See Memory Aliasing for more information.

`VkDevice` objects that have the `sparseResidencyAliased` feature enabled are able to use the `VK_BUFFER_CREATE_SPARSE_ALIASED_BIT` and `VK_IMAGE_CREATE_SPARSE_ALIASED_BIT` flags for resource creation. These flags allow resources to access physical memory bound into multiple locations within one or more sparse resources in a *data consistent* fashion. This means that reading physical memory from multiple aliased locations will return the same value.

Care **must** be taken when performing a write operation to aliased physical memory. Memory dependencies **must** be used to separate writes to one alias from reads or writes to another alias. Writes to aliased memory that are not properly guarded against accesses to different aliases will have undefined results for all accesses to the aliased memory.

Applications that wish to make use of data consistent sparse memory aliasing **must** abide by the following guidelines:

- All sparse resources that are bound to aliased physical memory **must** be created with the `VK_BUFFER_CREATE_SPARSE_ALIASED_BIT` / `VK_IMAGE_CREATE_SPARSE_ALIASED_BIT` flag.
- All resources that access aliased physical memory **must** interpret the memory in the same way. This implies the following:
  - Buffers and images **cannot** alias the same physical memory in a data consistent fashion. The physical memory ranges **must** be used exclusively by buffers or used exclusively by images for data consistency to be guaranteed.
  - Memory in sparse image mip tail regions **cannot** access aliased memory in a data consistent fashion.
  - Sparse images that alias the same physical memory **must** have compatible formats and be using the same sparse image block shape in order to access aliased memory in a data consistent fashion.

Failure to follow any of the above guidelines will require the application to abide by the normal, non-sparse resource aliasing rules. In this case memory **cannot** be accessed in a data consistent fashion.

| | |
|---|---|
| **NOTE** | Enabling sparse resource memory aliasing **can** be a way to lower physical memory use, but it **may** reduce performance on some implementations. An application |

developer **can** test on their target HW and balance the memory / performance trade-offs measured.

## 33.6. Sparse Resource Implementation Guidelines (Informative)

This section is Informative. It is included to aid in implementors' understanding of sparse resources.

*Device Virtual Address*

The basic `sparseBinding` feature allows the resource to reserve its own device virtual address range at resource creation time rather than relying on a bind operation to set this. Without any other creation flags, no other constraints are relaxed compared to normal resources. All pages **must** be bound to physical memory before the device accesses the resource.

The `sparseResidency` features allow sparse resources to be used even when not all pages are bound to memory. Implementations that support access to unbound pages without causing a fault **may** support `residencyNonResidentStrict`.

Not faulting on access to unbound pages is not enough to support `residencyNonResidentStrict`. An implementation **must** also guarantee that reads after writes to unbound regions of the resource always return data for the read as if the memory contains zeros. Depending on any caching hierarchy of the implementation this **may** not always be possible.

Any implementation that does not fault, but does not guarantee correct read values **must** not support `residencyNonResidentStrict`.

Any implementation that **cannot** access unbound pages without causing a fault will require the implementation to bind the entire device virtual address range to physical memory. Any pages that the application does not bind to memory **may** be bound to one (or more) "`placeholder" physical page(s) allocated by the implementation. Given the following properties:

- A process **must** not access memory from another process
- Reads return undefined values

It is sufficient for each host process to allocate these placeholder pages and use them for all resources in that process. Implementations **may** allocate more often (per instance, per device, or per resource).

*Binding Memory*

The byte size reported in `VkMemoryRequirements`::`size` **must** be greater than or equal to the amount of physical memory **required** to fully populate the resource. Some implementations require "holes" in the device virtual address range that are never accessed. These holes **may** be included in the `size` reported for the resource.

Including or not including the device virtual address holes in the resource size will alter how

the implementation provides support for `VkSparseImageOpaqueMemoryBindInfo`. This operation **must** be supported for all sparse images, even ones created with `VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT`.

- If the holes are included in the size, this bind function becomes very easy. In most cases the `resourceOffset` is simply a device virtual address offset and the implementation can easily determine what device virtual address to bind. The cost is that the application **may** allocate more physical memory for the resource than it needs.
- If the holes are not included in the size, the application **can** allocate less physical memory than otherwise for the resource. However, in this case the implementation **must** account for the holes when mapping `resourceOffset` to the actual device virtual address intended to be mapped.

**NOTE**

If the application always uses `VkSparseImageMemoryBindInfo` to bind memory for the non-tail mip levels, any holes that are present in the resource size **may** never be bound.

Since `VkSparseImageMemoryBindInfo` uses texel locations to determine which device virtual addresses to bind, it is impossible to bind device virtual address holes with this operation.

*Binding Metadata Memory*

All metadata for sparse images have their own sparse properties and are embedded in the mip tail region for said properties. See the Multiaspect section for details.

Given that metadata is in a mip tail region, and the mip tail region **must** be reported as contiguous (either globally or per-array-layer), some implementations will have to resort to complicated offset → device virtual address mapping for handling `VkSparseImageOpaqueMemoryBindInfo`.

To make this easier on the implementation, the `VK_SPARSE_MEMORY_BIND_METADATA_BIT` explicitly specifies when metadata is bound with `VkSparseImageOpaqueMemoryBindInfo`. When this flag is not present, the `resourceOffset` **may** be treated as a strict device virtual address offset.

When `VK_SPARSE_MEMORY_BIND_METADATA_BIT` is present, the `resourceOffset` **must** have been derived explicitly from the `imageMipTailOffset` in the sparse resource properties returned for the metadata aspect. By manipulating the value returned for `imageMipTailOffset`, the `resourceOffset` does not have to correlate directly to a device virtual address offset, and **may** instead be whatever value makes it easiest for the implementation to derive the correct device virtual address.

## 33.7. Sparse Resource API

The APIs related to sparse resources are grouped into the following categories:

- Physical Device Features
- Physical Device Sparse Properties

- Sparse Image Format Properties
- Sparse Resource Creation
- Sparse Resource Memory Requirements
- Binding Resource Memory

### 33.7.1. Physical Device Features

Some sparse-resource related features are reported and enabled in `VkPhysicalDeviceFeatures`. These features **must** be supported and enabled on the `VkDevice` object before applications **can** use them. See Physical Device Features for information on how to get and set enabled device features, and for more detailed explanations of these features.

#### Sparse Physical Device Features

- `sparseBinding`: Support for creating VkBuffer and `VkImage` objects with the `VK_BUFFER_CREATE_SPARSE_BINDING_BIT` and `VK_IMAGE_CREATE_SPARSE_BINDING_BIT` flags, respectively.
- `sparseResidencyBuffer`: Support for creating VkBuffer objects with the `VK_BUFFER_CREATE_SPARSE_RESIDENCY_BIT` flag.
- `sparseResidencyImage2D`: Support for creating 2D single-sampled `VkImage` objects with `VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT`.
- `sparseResidencyImage3D`: Support for creating 3D VkImage objects with `VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT`.
- `sparseResidency2Samples`: Support for creating 2D VkImage objects with 2 samples and `VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT`.
- `sparseResidency4Samples`: Support for creating 2D VkImage objects with 4 samples and `VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT`.
- `sparseResidency8Samples`: Support for creating 2D VkImage objects with 8 samples and `VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT`.
- `sparseResidency16Samples`: Support for creating 2D VkImage objects with 16 samples and `VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT`.
- `sparseResidencyAliased`: Support for creating VkBuffer and `VkImage` objects with the `VK_BUFFER_CREATE_SPARSE_ALIASED_BIT` and `VK_IMAGE_CREATE_SPARSE_ALIASED_BIT` flags, respectively.

### 33.7.2. Physical Device Sparse Properties

Some features of the implementation are not possible to disable, and are reported to allow applications to alter their sparse resource usage accordingly. These read-only capabilities are reported in the VkPhysicalDeviceProperties::`sparseProperties` member, which is a `VkPhysicalDeviceSparseProperties` structure.

The `VkPhysicalDeviceSparseProperties` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkPhysicalDeviceSparseProperties {
    VkBool32    residencyStandard2DBlockShape;
    VkBool32    residencyStandard2DMultisampleBlockShape;
    VkBool32    residencyStandard3DBlockShape;
    VkBool32    residencyAlignedMipSize;
    VkBool32    residencyNonResidentStrict;
} VkPhysicalDeviceSparseProperties;
```

- residencyStandard2DBlockShape is VK_TRUE if the physical device will access all single-sample 2D sparse resources using the standard sparse image block shapes (based on image format), as described in the Standard Sparse Image Block Shapes (Single Sample) table. If this property is not supported the value returned in the imageGranularity member of the VkSparseImageFormatProperties structure for single-sample 2D images is not **required** to match the standard sparse image block dimensions listed in the table.
- residencyStandard2DMultisampleBlockShape is VK_TRUE if the physical device will access all multisample 2D sparse resources using the standard sparse image block shapes (based on image format), as described in the Standard Sparse Image Block Shapes (MSAA) table. If this property is not supported, the value returned in the imageGranularity member of the VkSparseImageFormatProperties structure for multisample 2D images is not **required** to match the standard sparse image block dimensions listed in the table.
- residencyStandard3DBlockShape is VK_TRUE if the physical device will access all 3D sparse resources using the standard sparse image block shapes (based on image format), as described in the Standard Sparse Image Block Shapes (Single Sample) table. If this property is not supported, the value returned in the imageGranularity member of the VkSparseImageFormatProperties structure for 3D images is not **required** to match the standard sparse image block dimensions listed in the table.
- residencyAlignedMipSize is VK_TRUE if images with mip level dimensions that are not integer multiples of the corresponding dimensions of the sparse image block **may** be placed in the mip tail. If this property is not reported, only mip levels with dimensions smaller than the imageGranularity member of the VkSparseImageFormatProperties structure will be placed in the mip tail. If this property is reported the implementation is allowed to return VK_SPARSE_IMAGE_FORMAT_ALIGNED_MIP_SIZE_BIT in the flags member of VkSparseImageFormatProperties, indicating that mip level dimensions that are not integer multiples of the corresponding dimensions of the sparse image block will be placed in the mip tail.
- residencyNonResidentStrict specifies whether the physical device **can** consistently access non-resident regions of a resource. If this property is VK_TRUE, access to non-resident regions of resources will be guaranteed to return values as if the resource was populated with 0; writes to non-resident regions will be discarded.

### 33.7.3. Sparse Image Format Properties

Given that certain aspects of sparse image support, including the sparse image block dimensions, **may** be implementation-dependent, vkGetPhysicalDeviceSparseImageFormatProperties **can** be used to query for sparse image format properties prior to resource creation. This command is used

to check whether a given set of sparse image parameters is supported and what the sparse image block shape will be.

### Sparse Image Format Properties API

The `VkSparseImageFormatProperties` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkSparseImageFormatProperties {
    VkImageAspectFlags          aspectMask;
    VkExtent3D                  imageGranularity;
    VkSparseImageFormatFlags    flags;
} VkSparseImageFormatProperties;
```

- `aspectMask` is a bitmask VkImageAspectFlagBits specifying which aspects of the image the properties apply to.
- `imageGranularity` is the width, height, and depth of the sparse image block in texels or compressed texel blocks.
- `flags` is a bitmask of VkSparseImageFormatFlagBits specifying additional information about the sparse resource.

Bits which **may** be set in VkSparseImageFormatProperties::`flags`, specifying additional information about the sparse resource, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkSparseImageFormatFlagBits {
    VK_SPARSE_IMAGE_FORMAT_SINGLE_MIPTAIL_BIT = 0x00000001,
    VK_SPARSE_IMAGE_FORMAT_ALIGNED_MIP_SIZE_BIT = 0x00000002,
    VK_SPARSE_IMAGE_FORMAT_NONSTANDARD_BLOCK_SIZE_BIT = 0x00000004,
} VkSparseImageFormatFlagBits;
```

- `VK_SPARSE_IMAGE_FORMAT_SINGLE_MIPTAIL_BIT` specifies that the image uses a single mip tail region for all array layers.
- `VK_SPARSE_IMAGE_FORMAT_ALIGNED_MIP_SIZE_BIT` specifies that the first mip level whose dimensions are not integer multiples of the corresponding dimensions of the sparse image block begins the mip tail region.
- `VK_SPARSE_IMAGE_FORMAT_NONSTANDARD_BLOCK_SIZE_BIT` specifies that the image uses non-standard sparse image block dimensions, and the `imageGranularity` values do not match the standard sparse image block dimensions for the given format.

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkSparseImageFormatFlags;
```

`VkSparseImageFormatFlags` is a bitmask type for setting a mask of zero or more VkSparseImageFormatFlagBits.

vkGetPhysicalDeviceSparseImageFormatProperties returns an array of VkSparseImageFormatProperties. Each element describes properties for one set of image aspects that are bound simultaneously for a VkImage created with the provided image creation parameters. This is usually one element for each aspect in the image, but for interleaved depth/stencil images there is only one element describing the combined aspects.

```
// Provided by VK_VERSION_1_0
void vkGetPhysicalDeviceSparseImageFormatProperties(
    VkPhysicalDevice                            physicalDevice,
    VkFormat                                    format,
    VkImageType                                 type,
    VkSampleCountFlagBits                       samples,
    VkImageUsageFlags                           usage,
    VkImageTiling                               tiling,
    uint32_t*                                   pPropertyCount,
    VkSparseImageFormatProperties*              pProperties);
```

- physicalDevice is the physical device from which to query the sparse image format properties.
- format is the image format.
- type is the dimensionality of the image.
- samples is a VkSampleCountFlagBits value specifying the number of samples per texel.
- usage is a bitmask describing the intended usage of the image.
- tiling is the tiling arrangement of the texel blocks in memory.
- pPropertyCount is a pointer to an integer related to the number of sparse format properties available or queried, as described below.
- pProperties is either NULL or a pointer to an array of VkSparseImageFormatProperties structures.

If pProperties is NULL, then the number of sparse format properties available is returned in pPropertyCount. Otherwise, pPropertyCount **must** point to a variable set by the application to the number of elements in the pProperties array, and on return the variable is overwritten with the number of structures actually written to pProperties. If pPropertyCount is less than the number of sparse format properties available, at most pPropertyCount structures will be written.

If VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT is not supported for the given arguments, pPropertyCount will be set to zero upon return, and no data will be written to pProperties.

Multiple aspects are returned for depth/stencil images that are implemented as separate planes by the implementation. The depth and stencil data planes each have unique VkSparseImageFormatProperties data.

Depth/stencil images with depth and stencil data interleaved into a single plane will return a single VkSparseImageFormatProperties structure with the aspectMask set to VK_IMAGE_ASPECT_DEPTH_BIT | VK_IMAGE_ASPECT_STENCIL_BIT.

## Valid Usage

- VUID-vkGetPhysicalDeviceSparseImageFormatProperties-samples-01094
  `samples` **must** be a valid VkSampleCountFlagBits value that is set in `VkImageFormatProperties::sampleCounts` returned by `vkGetPhysicalDeviceImageFormatProperties` with `format`, `type`, `tiling`, and `usage` equal to those in this command

## Valid Usage (Implicit)

- VUID-vkGetPhysicalDeviceSparseImageFormatProperties-physicalDevice-parameter
  `physicalDevice` **must** be a valid VkPhysicalDevice handle
- VUID-vkGetPhysicalDeviceSparseImageFormatProperties-format-parameter
  `format` **must** be a valid VkFormat value
- VUID-vkGetPhysicalDeviceSparseImageFormatProperties-type-parameter
  `type` **must** be a valid VkImageType value
- VUID-vkGetPhysicalDeviceSparseImageFormatProperties-samples-parameter
  `samples` **must** be a valid VkSampleCountFlagBits value
- VUID-vkGetPhysicalDeviceSparseImageFormatProperties-usage-parameter
  `usage` **must** be a valid combination of VkImageUsageFlagBits values
- VUID-vkGetPhysicalDeviceSparseImageFormatProperties-usage-requiredbitmask
  `usage` **must** not be `0`
- VUID-vkGetPhysicalDeviceSparseImageFormatProperties-tiling-parameter
  `tiling` **must** be a valid VkImageTiling value
- VUID-vkGetPhysicalDeviceSparseImageFormatProperties-pPropertyCount-parameter
  `pPropertyCount` **must** be a valid pointer to a `uint32_t` value
- VUID-vkGetPhysicalDeviceSparseImageFormatProperties-pProperties-parameter
  If the value referenced by `pPropertyCount` is not `0`, and `pProperties` is not `NULL`, `pProperties` **must** be a valid pointer to an array of `pPropertyCount` VkSparseImageFormatProperties structures

`vkGetPhysicalDeviceSparseImageFormatProperties2` returns an array of VkSparseImageFormatProperties2. Each element describes properties for one set of image aspects that are bound simultaneously for a `VkImage` created with the provided image creation parameters. This is usually one element for each aspect in the image, but for interleaved depth/stencil images there is only one element describing the combined aspects.

```
// Provided by VK_VERSION_1_1
void vkGetPhysicalDeviceSparseImageFormatProperties2(
    VkPhysicalDevice                            physicalDevice,
    const VkPhysicalDeviceSparseImageFormatInfo2* pFormatInfo,
    uint32_t*                                   pPropertyCount,
    VkSparseImageFormatProperties2*             pProperties);
```

or the equivalent command

```
// Provided by VK_KHR_get_physical_device_properties2
void vkGetPhysicalDeviceSparseImageFormatProperties2KHR(
    VkPhysicalDevice                            physicalDevice,
    const VkPhysicalDeviceSparseImageFormatInfo2* pFormatInfo,
    uint32_t*                                   pPropertyCount,
    VkSparseImageFormatProperties2*             pProperties);
```

- `physicalDevice` is the physical device from which to query the sparse image format properties.
- `pFormatInfo` is a pointer to a VkPhysicalDeviceSparseImageFormatInfo2 structure containing input parameters to the command.
- `pPropertyCount` is a pointer to an integer related to the number of sparse format properties available or queried, as described below.
- `pProperties` is either `NULL` or a pointer to an array of VkSparseImageFormatProperties2 structures.

`vkGetPhysicalDeviceSparseImageFormatProperties2` behaves identically to vkGetPhysicalDeviceSparseImageFormatProperties, with the ability to return extended information by adding extending structures to the `pNext` chain of its `pProperties` parameter.

### Valid Usage (Implicit)

- VUID-vkGetPhysicalDeviceSparseImageFormatProperties2-physicalDevice-parameter
  `physicalDevice` **must** be a valid VkPhysicalDevice handle
- VUID-vkGetPhysicalDeviceSparseImageFormatProperties2-pFormatInfo-parameter
  `pFormatInfo` **must** be a valid pointer to a valid VkPhysicalDeviceSparseImageFormatInfo2 structure
- VUID-vkGetPhysicalDeviceSparseImageFormatProperties2-pPropertyCount-parameter
  `pPropertyCount` **must** be a valid pointer to a `uint32_t` value
- VUID-vkGetPhysicalDeviceSparseImageFormatProperties2-pProperties-parameter
  If the value referenced by `pPropertyCount` is not `0`, and `pProperties` is not `NULL`, `pProperties` **must** be a valid pointer to an array of `pPropertyCount` VkSparseImageFormatProperties2 structures

The `VkPhysicalDeviceSparseImageFormatInfo2` structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkPhysicalDeviceSparseImageFormatInfo2 {
    VkStructureType          sType;
    const void*              pNext;
    VkFormat                 format;
    VkImageType              type;
    VkSampleCountFlagBits    samples;
    VkImageUsageFlags        usage;
    VkImageTiling            tiling;
} VkPhysicalDeviceSparseImageFormatInfo2;
```

or the equivalent

```
// Provided by VK_KHR_get_physical_device_properties2
typedef VkPhysicalDeviceSparseImageFormatInfo2
VkPhysicalDeviceSparseImageFormatInfo2KHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `format` is the image format.
- `type` is the dimensionality of the image.
- `samples` is a VkSampleCountFlagBits value specifying the number of samples per texel.
- `usage` is a bitmask describing the intended usage of the image.
- `tiling` is the tiling arrangement of the texel blocks in memory.

### Valid Usage

- VUID-VkPhysicalDeviceSparseImageFormatInfo2-samples-01095
  `samples` **must** be a valid VkSampleCountFlagBits value that is set in `VkImageFormatProperties::sampleCounts` returned by `vkGetPhysicalDeviceImageFormatProperties` with `format`, `type`, `tiling`, and `usage` equal to those in this command

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceSparseImageFormatInfo2-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SPARSE_IMAGE_FORMAT_INFO_2`
- VUID-VkPhysicalDeviceSparseImageFormatInfo2-pNext-pNext
  `pNext` **must** be `NULL`
- VUID-VkPhysicalDeviceSparseImageFormatInfo2-format-parameter
  `format` **must** be a valid VkFormat value
- VUID-VkPhysicalDeviceSparseImageFormatInfo2-type-parameter

type **must** be a valid VkImageType value

- VUID-VkPhysicalDeviceSparseImageFormatInfo2-samples-parameter
  samples **must** be a valid VkSampleCountFlagBits value
- VUID-VkPhysicalDeviceSparseImageFormatInfo2-usage-parameter
  usage **must** be a valid combination of VkImageUsageFlagBits values
- VUID-VkPhysicalDeviceSparseImageFormatInfo2-usage-requiredbitmask
  usage **must** not be 0
- VUID-VkPhysicalDeviceSparseImageFormatInfo2-tiling-parameter
  tiling **must** be a valid VkImageTiling value

The VkSparseImageFormatProperties2 structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkSparseImageFormatProperties2 {
    VkStructureType                 sType;
    void*                           pNext;
    VkSparseImageFormatProperties   properties;
} VkSparseImageFormatProperties2;
```

or the equivalent

```
// Provided by VK_KHR_get_physical_device_properties2
typedef VkSparseImageFormatProperties2 VkSparseImageFormatProperties2KHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- properties is a VkSparseImageFormatProperties structure which is populated with the same values as in vkGetPhysicalDeviceSparseImageFormatProperties.

**Valid Usage (Implicit)**

- VUID-VkSparseImageFormatProperties2-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_SPARSE_IMAGE_FORMAT_PROPERTIES_2
- VUID-VkSparseImageFormatProperties2-pNext-pNext
  pNext **must** be NULL

### 33.7.4. Sparse Resource Creation

Sparse resources require that one or more sparse feature flags be specified (as part of the VkPhysicalDeviceFeatures structure described previously in the Physical Device Features section) when calling vkCreateDevice. When the appropriate device features are enabled, the VK_BUFFER_CREATE_SPARSE_* and VK_IMAGE_CREATE_SPARSE_* flags **can** be used. See vkCreateBuffer and

vkCreateImage for details of the resource creation APIs.

| | |
|---|---|
| **NOTE** | Specifying VK_BUFFER_CREATE_SPARSE_RESIDENCY_BIT or VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT requires specifying VK_BUFFER_CREATE_SPARSE_BINDING_BIT or VK_IMAGE_CREATE_SPARSE_BINDING_BIT, respectively, as well. This means that resources **must** be created with the appropriate *_SPARSE_BINDING_BIT to be used with the sparse binding command (vkQueueBindSparse). |

### 33.7.5. Sparse Resource Memory Requirements

Sparse resources have specific memory requirements related to binding sparse memory. These memory requirements are reported differently for VkBuffer objects and VkImage objects.

#### Buffer and Fully-Resident Images

Buffers (both fully and partially resident) and fully-resident images **can** be bound to memory using only the data from VkMemoryRequirements. For all sparse resources the VkMemoryRequirements::alignment member specifies both the binding granularity in bytes and the **required** alignment of VkDeviceMemory.

#### Partially Resident Images

Partially resident images have a different method for binding memory. As with buffers and fully resident images, the VkMemoryRequirements::alignment field specifies the binding granularity in bytes for the image.

Requesting sparse memory requirements for VkImage objects using vkGetImageSparseMemoryRequirements will return an array of one or more VkSparseImageMemoryRequirements structures. Each structure describes the sparse memory requirements for a group of aspects of the image.

The sparse image **must** have been created using the VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT flag to retrieve valid sparse image memory requirements.

#### Sparse Image Memory Requirements

The VkSparseImageMemoryRequirements structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkSparseImageMemoryRequirements {
    VkSparseImageFormatProperties    formatProperties;
    uint32_t                         imageMipTailFirstLod;
    VkDeviceSize                     imageMipTailSize;
    VkDeviceSize                     imageMipTailOffset;
    VkDeviceSize                     imageMipTailStride;
} VkSparseImageMemoryRequirements;
```

- formatProperties is a VkSparseImageFormatProperties structure specifying properties of the

image format.

- imageMipTailFirstLod is the first mip level at which image subresources are included in the mip tail region.
- imageMipTailSize is the memory size (in bytes) of the mip tail region. If formatProperties.flags contains VK_SPARSE_IMAGE_FORMAT_SINGLE_MIPTAIL_BIT, this is the size of the whole mip tail, otherwise this is the size of the mip tail of a single array layer. This value is guaranteed to be a multiple of the sparse block size in bytes.
- imageMipTailOffset is the opaque memory offset used with VkSparseImageOpaqueMemoryBindInfo to bind the mip tail region(s).
- imageMipTailStride is the offset stride between each array-layer's mip tail, if formatProperties.flags does not contain VK_SPARSE_IMAGE_FORMAT_SINGLE_MIPTAIL_BIT (otherwise the value is undefined).

To query sparse memory requirements for an image, call:

```
// Provided by VK_VERSION_1_0
void vkGetImageSparseMemoryRequirements(
    VkDevice                                    device,
    VkImage                                     image,
    uint32_t*                                   pSparseMemoryRequirementCount,
    VkSparseImageMemoryRequirements*            pSparseMemoryRequirements);
```

- device is the logical device that owns the image.
- image is the VkImage object to get the memory requirements for.
- pSparseMemoryRequirementCount is a pointer to an integer related to the number of sparse memory requirements available or queried, as described below.
- pSparseMemoryRequirements is either NULL or a pointer to an array of VkSparseImageMemoryRequirements structures.

If pSparseMemoryRequirements is NULL, then the number of sparse memory requirements available is returned in pSparseMemoryRequirementCount. Otherwise, pSparseMemoryRequirementCount **must** point to a variable set by the application to the number of elements in the pSparseMemoryRequirements array, and on return the variable is overwritten with the number of structures actually written to pSparseMemoryRequirements. If pSparseMemoryRequirementCount is less than the number of sparse memory requirements available, at most pSparseMemoryRequirementCount structures will be written.

If the image was not created with VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT then pSparseMemoryRequirementCount will be set to zero and pSparseMemoryRequirements will not be written to.

**NOTE** It is legal for an implementation to report a larger value in VkMemoryRequirements::size than would be obtained by adding together memory sizes for all VkSparseImageMemoryRequirements returned by vkGetImageSparseMemoryRequirements. This **may** occur when the implementation requires unused padding in the address range describing the resource.

### Valid Usage (Implicit)

- VUID-vkGetImageSparseMemoryRequirements-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkGetImageSparseMemoryRequirements-image-parameter
  `image` **must** be a valid VkImage handle
- VUID-vkGetImageSparseMemoryRequirements-pSparseMemoryRequirementCount-parameter
  `pSparseMemoryRequirementCount` **must** be a valid pointer to a `uint32_t` value
- VUID-vkGetImageSparseMemoryRequirements-pSparseMemoryRequirements-parameter
  If the value referenced by `pSparseMemoryRequirementCount` is not `0`, and `pSparseMemoryRequirements` is not `NULL`, `pSparseMemoryRequirements` **must** be a valid pointer to an array of `pSparseMemoryRequirementCount` VkSparseImageMemoryRequirements structures
- VUID-vkGetImageSparseMemoryRequirements-image-parent
  `image` **must** have been created, allocated, or retrieved from `device`

To query sparse memory requirements for an image, call:

```
// Provided by VK_VERSION_1_1
void vkGetImageSparseMemoryRequirements2(
    VkDevice                                    device,
    const VkImageSparseMemoryRequirementsInfo2* pInfo,
    uint32_t*                                   pSparseMemoryRequirementCount,
    VkSparseImageMemoryRequirements2*           pSparseMemoryRequirements);
```

or the equivalent command

```
// Provided by VK_KHR_get_memory_requirements2
void vkGetImageSparseMemoryRequirements2KHR(
    VkDevice                                    device,
    const VkImageSparseMemoryRequirementsInfo2* pInfo,
    uint32_t*                                   pSparseMemoryRequirementCount,
    VkSparseImageMemoryRequirements2*           pSparseMemoryRequirements);
```

- `device` is the logical device that owns the image.
- `pInfo` is a pointer to a `VkImageSparseMemoryRequirementsInfo2` structure containing parameters required for the memory requirements query.
- `pSparseMemoryRequirementCount` is a pointer to an integer related to the number of sparse memory requirements available or queried, as described below.
- `pSparseMemoryRequirements` is either `NULL` or a pointer to an array of `VkSparseImageMemoryRequirements2` structures.

### Valid Usage (Implicit)

- VUID-vkGetImageSparseMemoryRequirements2-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkGetImageSparseMemoryRequirements2-pInfo-parameter
  `pInfo` **must** be a valid pointer to a valid VkImageSparseMemoryRequirementsInfo2 structure
- VUID-vkGetImageSparseMemoryRequirements2-pSparseMemoryRequirementCount-parameter
  `pSparseMemoryRequirementCount` **must** be a valid pointer to a `uint32_t` value
- VUID-vkGetImageSparseMemoryRequirements2-pSparseMemoryRequirements-parameter
  If the value referenced by `pSparseMemoryRequirementCount` is not `0`, and `pSparseMemoryRequirements` is not `NULL`, `pSparseMemoryRequirements` **must** be a valid pointer to an array of `pSparseMemoryRequirementCount` VkSparseImageMemoryRequirements2 structures

To determine the sparse memory requirements for an image resource without creating an object, call:

```
// Provided by VK_KHR_maintenance4
void vkGetDeviceImageSparseMemoryRequirementsKHR(
    VkDevice                                    device,
    const VkDeviceImageMemoryRequirements*      pInfo,
    uint32_t*                                   pSparseMemoryRequirementCount,
    VkSparseImageMemoryRequirements2*           pSparseMemoryRequirements);
```

- `device` is the logical device intended to own the image.
- `pInfo` is a pointer to a VkDeviceImageMemoryRequirements structure containing parameters required for the memory requirements query.
- `pSparseMemoryRequirementCount` is a pointer to an integer related to the number of sparse memory requirements available or queried, as described below.
- `pSparseMemoryRequirements` is either `NULL` or a pointer to an array of `VkSparseImageMemoryRequirements2` structures.

### Valid Usage (Implicit)

- VUID-vkGetDeviceImageSparseMemoryRequirements-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkGetDeviceImageSparseMemoryRequirements-pInfo-parameter
  `pInfo` **must** be a valid pointer to a valid VkDeviceImageMemoryRequirements structure
- VUID-vkGetDeviceImageSparseMemoryRequirements-pSparseMemoryRequirementCount-parameter

pSparseMemoryRequirementCount **must** be a valid pointer to a uint32_t value

- VUID-vkGetDeviceImageSparseMemoryRequirements-pSparseMemoryRequirements-parameter
  If the value referenced by pSparseMemoryRequirementCount is not 0, and pSparseMemoryRequirements is not NULL, pSparseMemoryRequirements **must** be a valid pointer to an array of pSparseMemoryRequirementCount VkSparseImageMemoryRequirements2 structures

The VkImageSparseMemoryRequirementsInfo2 structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkImageSparseMemoryRequirementsInfo2 {
    VkStructureType    sType;
    const void*        pNext;
    VkImage            image;
} VkImageSparseMemoryRequirementsInfo2;
```

or the equivalent

```
// Provided by VK_KHR_get_memory_requirements2
typedef VkImageSparseMemoryRequirementsInfo2 VkImageSparseMemoryRequirementsInfo2KHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- image is the image to query.

### Valid Usage (Implicit)

- VUID-VkImageSparseMemoryRequirementsInfo2-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_IMAGE_SPARSE_MEMORY_REQUIREMENTS_INFO_2
- VUID-VkImageSparseMemoryRequirementsInfo2-pNext-pNext
  pNext **must** be NULL
- VUID-VkImageSparseMemoryRequirementsInfo2-image-parameter
  image **must** be a valid VkImage handle

The VkSparseImageMemoryRequirements2 structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkSparseImageMemoryRequirements2 {
    VkStructureType                     sType;
    void*                               pNext;
    VkSparseImageMemoryRequirements     memoryRequirements;
} VkSparseImageMemoryRequirements2;
```

or the equivalent

```
// Provided by VK_KHR_get_memory_requirements2
typedef VkSparseImageMemoryRequirements2 VkSparseImageMemoryRequirements2KHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `memoryRequirements` is a VkSparseImageMemoryRequirements structure describing the memory requirements of the sparse image.

**Valid Usage (Implicit)**

- VUID-VkSparseImageMemoryRequirements2-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_SPARSE_IMAGE_MEMORY_REQUIREMENTS_2`
- VUID-VkSparseImageMemoryRequirements2-pNext-pNext
  `pNext` **must** be `NULL`

### 33.7.6. Binding Resource Memory

Non-sparse resources are backed by a single physical allocation prior to device use (via `vkBindImageMemory` or `vkBindBufferMemory`), and their backing **must** not be changed. On the other hand, sparse resources **can** be bound to memory non-contiguously and these bindings **can** be altered during the lifetime of the resource.

**NOTE** It is important to note that freeing a `VkDeviceMemory` object with `vkFreeMemory` will not cause resources (or resource regions) bound to the memory object to become unbound. Applications **must** not access resources bound to memory that has been freed.

Sparse memory bindings execute on a queue that includes the `VK_QUEUE_SPARSE_BINDING_BIT` bit. Applications **must** use synchronization primitives to guarantee that other queues do not access ranges of memory concurrently with a binding change. Applications **can** access other ranges of the same resource while a bind operation is executing.

**NOTE** Implementations **must** provide a guarantee that simultaneously binding sparse blocks while another queue accesses those same sparse blocks via a sparse resource **must** not access memory owned by another process or otherwise corrupt the

system.

While some implementations **may** include VK_QUEUE_SPARSE_BINDING_BIT support in queue families that also include graphics and compute support, other implementations **may** only expose a VK_QUEUE_SPARSE_BINDING_BIT-only queue family. In either case, applications **must** use synchronization primitives to explicitly request any ordering dependencies between sparse memory binding operations and other graphics/compute/transfer operations, as sparse binding operations are not automatically ordered against command buffer execution, even within a single queue.

When binding memory explicitly for the VK_IMAGE_ASPECT_METADATA_BIT the application **must** use the VK_SPARSE_MEMORY_BIND_METADATA_BIT in the VkSparseMemoryBind::flags field when binding memory. Binding memory for metadata is done the same way as binding memory for the mip tail, with the addition of the VK_SPARSE_MEMORY_BIND_METADATA_BIT flag.

Binding the mip tail for any aspect **must** only be performed using VkSparseImageOpaqueMemoryBindInfo. If formatProperties.flags contains VK_SPARSE_IMAGE_FORMAT_SINGLE_MIPTAIL_BIT, then it **can** be bound with a single VkSparseMemoryBind structure, with resourceOffset = imageMipTailOffset and size = imageMipTailSize.

If formatProperties.flags does not contain VK_SPARSE_IMAGE_FORMAT_SINGLE_MIPTAIL_BIT then the offset for the mip tail in each array layer is given as:

```
arrayMipTailOffset = imageMipTailOffset + arrayLayer * imageMipTailStride;
```

and the mip tail **can** be bound with layerCount VkSparseMemoryBind structures, each using size = imageMipTailSize and resourceOffset = arrayMipTailOffset as defined above.

Sparse memory binding is handled by the following APIs and related data structures.

### Sparse Memory Binding Functions

The VkSparseMemoryBind structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkSparseMemoryBind {
    VkDeviceSize               resourceOffset;
    VkDeviceSize               size;
    VkDeviceMemory             memory;
    VkDeviceSize               memoryOffset;
    VkSparseMemoryBindFlags    flags;
} VkSparseMemoryBind;
```

- resourceOffset is the offset into the resource.
- size is the size of the memory region to be bound.
- memory is the VkDeviceMemory object that the range of the resource is bound to. If memory is

VK_NULL_HANDLE, the range is unbound.

- `memoryOffset` is the offset into the VkDeviceMemory object to bind the resource range to. If `memory` is VK_NULL_HANDLE, this value is ignored.
- `flags` is a bitmask of VkSparseMemoryBindFlagBits specifying usage of the binding operation.

The *binding range* [`resourceOffset`, `resourceOffset` + `size`) has different constraints based on `flags`. If `flags` contains `VK_SPARSE_MEMORY_BIND_METADATA_BIT`, the binding range **must** be within the mip tail region of the metadata aspect. This metadata region is defined by:

metadataRegion = [base, base + `imageMipTailSize`)

base = `imageMipTailOffset` + `imageMipTailStride` × n

and `imageMipTailOffset`, `imageMipTailSize`, and `imageMipTailStride` values are from the VkSparseImageMemoryRequirements corresponding to the metadata aspect of the image, and n is a valid array layer index for the image,

`imageMipTailStride` is considered to be zero for aspects where `VkSparseImageMemoryRequirements::formatProperties.flags` contains `VK_SPARSE_IMAGE_FORMAT_SINGLE_MIPTAIL_BIT`.

If `flags` does not contain `VK_SPARSE_MEMORY_BIND_METADATA_BIT`, the binding range **must** be within the range [0,VkMemoryRequirements::`size`).

## Valid Usage

- VUID-VkSparseMemoryBind-memory-01096
  If `memory` is not VK_NULL_HANDLE, `memory` and `memoryOffset` **must** match the memory requirements of the resource, as described in section Resource Memory Association
- VUID-VkSparseMemoryBind-resourceOffset-09491
  If the resource being bound is a `VkBuffer`, `resourceOffset`, `memoryOffset` and `size` **must** be an integer multiple of the `alignment` of the VkMemoryRequirements structure returned from a call to vkGetBufferMemoryRequirements with the buffer resource
- VUID-VkSparseMemoryBind-resourceOffset-09492
  If the resource being bound is a `VkImage`, `resourceOffset` and `memoryOffset` **must** be an integer multiple of the `alignment` of the VkMemoryRequirements structure returned from a call to vkGetImageMemoryRequirements with the image resource
- VUID-VkSparseMemoryBind-memory-01097
  If `memory` is not VK_NULL_HANDLE, `memory` **must** not have been created with a memory type that reports `VK_MEMORY_PROPERTY_LAZILY_ALLOCATED_BIT` bit set
- VUID-VkSparseMemoryBind-size-01098
  `size` **must** be greater than `0`
- VUID-VkSparseMemoryBind-resourceOffset-01099
  `resourceOffset` **must** be less than the size of the resource

- VUID-VkSparseMemoryBind-size-01100
  `size` **must** be less than or equal to the size of the resource minus `resourceOffset`
- VUID-VkSparseMemoryBind-memoryOffset-01101
  `memoryOffset` **must** be less than the size of `memory`
- VUID-VkSparseMemoryBind-size-01102
  `size` **must** be less than or equal to the size of `memory` minus `memoryOffset`
- VUID-VkSparseMemoryBind-memory-02730
  If `memory` was created with VkExportMemoryAllocateInfo::`handleTypes` not equal to `0`, at least one handle type it contained **must** also have been set in VkExternalMemoryBufferCreateInfo::`handleTypes` or VkExternalMemoryImageCreateInfo::`handleTypes` when the resource was created
- VUID-VkSparseMemoryBind-memory-02731
  If `memory` was created by a memory import operation, the external handle type of the imported memory **must** also have been set in VkExternalMemoryBufferCreateInfo::`handleTypes` or VkExternalMemoryImageCreateInfo::`handleTypes` when the resource was created

### Valid Usage (Implicit)

- VUID-VkSparseMemoryBind-memory-parameter
  If `memory` is not VK_NULL_HANDLE, `memory` **must** be a valid VkDeviceMemory handle
- VUID-VkSparseMemoryBind-flags-parameter
  `flags` **must** be a valid combination of VkSparseMemoryBindFlagBits values

Bits which **can** be set in VkSparseMemoryBind::`flags`, specifying usage of a sparse memory binding operation, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkSparseMemoryBindFlagBits {
    VK_SPARSE_MEMORY_BIND_METADATA_BIT = 0x00000001,
} VkSparseMemoryBindFlagBits;
```

- `VK_SPARSE_MEMORY_BIND_METADATA_BIT` specifies that the memory being bound is only for the metadata aspect.

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkSparseMemoryBindFlags;
```

`VkSparseMemoryBindFlags` is a bitmask type for setting a mask of zero or more VkSparseMemoryBindFlagBits.

Memory is bound to `VkBuffer` objects created with the `VK_BUFFER_CREATE_SPARSE_BINDING_BIT` flag using the following structure:

```
// Provided by VK_VERSION_1_0
typedef struct VkSparseBufferMemoryBindInfo {
    VkBuffer                        buffer;
    uint32_t                        bindCount;
    const VkSparseMemoryBind*       pBinds;
} VkSparseBufferMemoryBindInfo;
```

- `buffer` is the VkBuffer object to be bound.
- `bindCount` is the number of VkSparseMemoryBind structures in the `pBinds` array.
- `pBinds` is a pointer to an array of VkSparseMemoryBind structures.

### Valid Usage (Implicit)

- VUID-VkSparseBufferMemoryBindInfo-buffer-parameter
  `buffer` **must** be a valid VkBuffer handle
- VUID-VkSparseBufferMemoryBindInfo-pBinds-parameter
  `pBinds` **must** be a valid pointer to an array of `bindCount` valid VkSparseMemoryBind structures
- VUID-VkSparseBufferMemoryBindInfo-bindCount-arraylength
  `bindCount` **must** be greater than `0`

Memory is bound to opaque regions of `VkImage` objects created with the `VK_IMAGE_CREATE_SPARSE_BINDING_BIT` flag using the following structure:

```
// Provided by VK_VERSION_1_0
typedef struct VkSparseImageOpaqueMemoryBindInfo {
    VkImage                         image;
    uint32_t                        bindCount;
    const VkSparseMemoryBind*       pBinds;
} VkSparseImageOpaqueMemoryBindInfo;
```

- `image` is the VkImage object to be bound.
- `bindCount` is the number of VkSparseMemoryBind structures in the `pBinds` array.
- `pBinds` is a pointer to an array of VkSparseMemoryBind structures.

**NOTE**

This structure is normally used to bind memory to fully-resident sparse images or for mip tail regions of partially resident images. However, it **can** also be used to bind memory for the entire binding range of partially resident images.

If the `pBinds`[i].flags of an element *i* of `pBinds` does not contain `VK_SPARSE_MEMORY_BIND_METADATA_BIT`, the `resourceOffset` is in the range [0, VkMemoryRequirements::`size`), This range includes data from all aspects of the image, including metadata. For most implementations this will probably mean that

the resourceOffset is a simple device address offset within the resource. It is possible for an application to bind a range of memory that includes both resource data and metadata. However, the application would not know what part of the image the memory is used for, or if any range is being used for metadata.

If the pBinds[i].flags of an element *i* of pBinds contains VK_SPARSE_MEMORY_BIND_METADATA_BIT, the binding range specified **must** be within the mip tail region of the metadata aspect. In this case the resourceOffset is not **required** to be a simple device address offset within the resource. However, it *is* defined to be within [imageMipTailOffset, imageMipTailOffset + imageMipTailSize) for the metadata aspect. See VkSparseMemoryBind for the full constraints on binding region with this flag present.

## Valid Usage

- VUID-VkSparseImageOpaqueMemoryBindInfo-pBinds-01103
  If the flags member of any element of pBinds contains VK_SPARSE_MEMORY_BIND_METADATA_BIT, the binding range defined **must** be within the mip tail region of the metadata aspect of image

## Valid Usage (Implicit)

- VUID-VkSparseImageOpaqueMemoryBindInfo-image-parameter
  image **must** be a valid VkImage handle
- VUID-VkSparseImageOpaqueMemoryBindInfo-pBinds-parameter
  pBinds **must** be a valid pointer to an array of bindCount valid VkSparseMemoryBind structures
- VUID-VkSparseImageOpaqueMemoryBindInfo-bindCount-arraylength
  bindCount **must** be greater than 0

Memory **can** be bound to sparse image blocks of VkImage objects created with the VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT flag using the following structure:

```
// Provided by VK_VERSION_1_0
typedef struct VkSparseImageMemoryBindInfo {
    VkImage                         image;
    uint32_t                        bindCount;
    const VkSparseImageMemoryBind*  pBinds;
} VkSparseImageMemoryBindInfo;
```

- image is the VkImage object to be bound
- bindCount is the number of VkSparseImageMemoryBind structures in pBinds array
- pBinds is a pointer to an array of VkSparseImageMemoryBind structures

### Valid Usage

- VUID-VkSparseImageMemoryBindInfo-subresource-01722
  The `subresource.mipLevel` member of each element of `pBinds` **must** be less than the `mipLevels` specified in VkImageCreateInfo when `image` was created
- VUID-VkSparseImageMemoryBindInfo-subresource-01723
  The `subresource.arrayLayer` member of each element of `pBinds` **must** be less than the `arrayLayers` specified in VkImageCreateInfo when `image` was created
- VUID-VkSparseImageMemoryBindInfo-subresource-01106
  The `subresource.aspectMask` member of each element of `pBinds` **must** be valid for the `format` specified in VkImageCreateInfo when `image` was created
- VUID-VkSparseImageMemoryBindInfo-image-02901
  `image` **must** have been created with `VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT` set

### Valid Usage (Implicit)

- VUID-VkSparseImageMemoryBindInfo-image-parameter
  `image` **must** be a valid VkImage handle
- VUID-VkSparseImageMemoryBindInfo-pBinds-parameter
  `pBinds` **must** be a valid pointer to an array of `bindCount` valid VkSparseImageMemoryBind structures
- VUID-VkSparseImageMemoryBindInfo-bindCount-arraylength
  `bindCount` **must** be greater than `0`

The `VkSparseImageMemoryBind` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkSparseImageMemoryBind {
    VkImageSubresource          subresource;
    VkOffset3D                  offset;
    VkExtent3D                  extent;
    VkDeviceMemory              memory;
    VkDeviceSize                memoryOffset;
    VkSparseMemoryBindFlags     flags;
} VkSparseImageMemoryBind;
```

- `subresource` is the image *aspect* and region of interest in the image.
- `offset` are the coordinates of the first texel within the image subresource to bind.
- `extent` is the size in texels of the region within the image subresource to bind. The extent **must** be a multiple of the sparse image block dimensions, except when binding sparse image blocks along the edge of an image subresource it **can** instead be such that any coordinate of `offset` + `extent` equals the corresponding dimensions of the image subresource.

- `memory` is the VkDeviceMemory object that the sparse image blocks of the image are bound to. If `memory` is VK_NULL_HANDLE, the sparse image blocks are unbound.
- `memoryOffset` is an offset into VkDeviceMemory object. If `memory` is VK_NULL_HANDLE, this value is ignored.
- `flags` are sparse memory binding flags.

## Valid Usage

- VUID-VkSparseImageMemoryBind-memory-01104
  If the `sparseResidencyAliased` feature is not enabled, and if any other resources are bound to ranges of `memory`, the range of `memory` being bound **must** not overlap with those bound ranges
- VUID-VkSparseImageMemoryBind-memory-01105
  `memory` and `memoryOffset` **must** match the memory requirements of the calling command's `image`, as described in section Resource Memory Association
- VUID-VkSparseImageMemoryBind-offset-01107
  `offset.x` **must** be a multiple of the sparse image block width (`VkSparseImageFormatProperties::imageGranularity.width`) of the image
- VUID-VkSparseImageMemoryBind-extent-09388
  `extent.width` **must** be greater than `0`
- VUID-VkSparseImageMemoryBind-extent-01108
  `extent.width` **must** either be a multiple of the sparse image block width of the image, or else (`extent.width` + `offset.x`) **must** equal the width of the image subresource
- VUID-VkSparseImageMemoryBind-offset-01109
  `offset.y` **must** be a multiple of the sparse image block height (`VkSparseImageFormatProperties::imageGranularity.height`) of the image
- VUID-VkSparseImageMemoryBind-extent-09389
  `extent.height` **must** be greater than `0`
- VUID-VkSparseImageMemoryBind-extent-01110
  `extent.height` **must** either be a multiple of the sparse image block height of the image, or else (`extent.height` + `offset.y`) **must** equal the height of the image subresource
- VUID-VkSparseImageMemoryBind-offset-01111
  `offset.z` **must** be a multiple of the sparse image block depth (`VkSparseImageFormatProperties::imageGranularity.depth`) of the image
- VUID-VkSparseImageMemoryBind-extent-09390
  `extent.depth` **must** be greater than `0`
- VUID-VkSparseImageMemoryBind-extent-01112
  `extent.depth` **must** either be a multiple of the sparse image block depth of the image, or else (`extent.depth` + `offset.z`) **must** equal the depth of the image subresource
- VUID-VkSparseImageMemoryBind-memory-02732
  If `memory` was created with VkExportMemoryAllocateInfo::`handleTypes` not equal to `0`, at least one handle type it contained **must** also have been set in

VkExternalMemoryImageCreateInfo::handleTypes when the image was created

- VUID-VkSparseImageMemoryBind-memory-02733
  If memory was created by a memory import operation, the external handle type of the imported memory **must** also have been set in VkExternalMemoryImageCreateInfo::handleTypes when image was created

### Valid Usage (Implicit)

- VUID-VkSparseImageMemoryBind-subresource-parameter
  subresource **must** be a valid VkImageSubresource structure
- VUID-VkSparseImageMemoryBind-memory-parameter
  If memory is not VK_NULL_HANDLE, memory **must** be a valid VkDeviceMemory handle
- VUID-VkSparseImageMemoryBind-flags-parameter
  flags **must** be a valid combination of VkSparseMemoryBindFlagBits values

To submit sparse binding operations to a queue, call:

```
// Provided by VK_VERSION_1_0
VkResult vkQueueBindSparse(
    VkQueue                                     queue,
    uint32_t                                    bindInfoCount,
    const VkBindSparseInfo*                     pBindInfo,
    VkFence                                     fence);
```

- queue is the queue that the sparse binding operations will be submitted to.
- bindInfoCount is the number of elements in the pBindInfo array.
- pBindInfo is a pointer to an array of VkBindSparseInfo structures, each specifying a sparse binding submission batch.
- fence is an **optional** handle to a fence to be signaled. If fence is not VK_NULL_HANDLE, it defines a fence signal operation.

vkQueueBindSparse is a queue submission command, with each batch defined by an element of pBindInfo as a VkBindSparseInfo structure. Batches begin execution in the order they appear in pBindInfo, but **may** complete out of order.

Within a batch, a given range of a resource **must** not be bound more than once. Across batches, if a range is to be bound to one allocation and offset and then to another allocation and offset, then the application **must** guarantee (usually using semaphores) that the binding operations are executed in the correct order, as well as to order binding operations against the execution of command buffer submissions.

As no operation to vkQueueBindSparse causes any pipeline stage to access memory, synchronization primitives used in this command effectively only define execution dependencies.

Additional information about fence and semaphore operation is described in the synchronization chapter.

## Valid Usage

- VUID-vkQueueBindSparse-fence-01113
  If `fence` is not VK_NULL_HANDLE, `fence` **must** be unsignaled
- VUID-vkQueueBindSparse-fence-01114
  If `fence` is not VK_NULL_HANDLE, `fence` **must** not be associated with any other queue command that has not yet completed execution on that queue
- VUID-vkQueueBindSparse-pSignalSemaphores-01115
  Each element of the `pSignalSemaphores` member of each element of `pBindInfo` **must** be unsignaled when the semaphore signal operation it defines is executed on the device
- VUID-vkQueueBindSparse-pWaitSemaphores-01116
  When a semaphore wait operation referring to a binary semaphore defined by any element of the `pWaitSemaphores` member of any element of `pBindInfo` executes on `queue`, there **must** be no other queues waiting on the same semaphore
- VUID-vkQueueBindSparse-pWaitSemaphores-03245
  All elements of the `pWaitSemaphores` member of all elements of `pBindInfo` referring to a semaphore created with a VkSemaphoreType of `VK_SEMAPHORE_TYPE_BINARY` **must** reference a semaphore signal operation that has been submitted for execution and any semaphore signal operations on which it depends **must** have also been submitted for execution

## Valid Usage (Implicit)

- VUID-vkQueueBindSparse-queue-parameter
  `queue` **must** be a valid VkQueue handle
- VUID-vkQueueBindSparse-pBindInfo-parameter
  If `bindInfoCount` is not `0`, `pBindInfo` **must** be a valid pointer to an array of `bindInfoCount` valid VkBindSparseInfo structures
- VUID-vkQueueBindSparse-fence-parameter
  If `fence` is not VK_NULL_HANDLE, `fence` **must** be a valid VkFence handle
- VUID-vkQueueBindSparse-queuetype
  The `queue` **must** support sparse binding operations
- VUID-vkQueueBindSparse-commonparent
  Both of `fence`, and `queue` that are valid handles of non-ignored parameters **must** have been created, allocated, or retrieved from the same VkDevice

## Host Synchronization

- Host access to `queue` **must** be externally synchronized
- Host access to `fence` **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| - | - | - | SPARSE_BINDING | - |

## Return Codes

**Success**

- VK_SUCCESS

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY
- VK_ERROR_DEVICE_LOST

The VkBindSparseInfo structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkBindSparseInfo {
    VkStructureType                             sType;
    const void*                                 pNext;
    uint32_t                                    waitSemaphoreCount;
    const VkSemaphore*                          pWaitSemaphores;
    uint32_t                                    bufferBindCount;
    const VkSparseBufferMemoryBindInfo*         pBufferBinds;
    uint32_t                                    imageOpaqueBindCount;
    const VkSparseImageOpaqueMemoryBindInfo*    pImageOpaqueBinds;
    uint32_t                                    imageBindCount;
    const VkSparseImageMemoryBindInfo*          pImageBinds;
    uint32_t                                    signalSemaphoreCount;
    const VkSemaphore*                          pSignalSemaphores;
} VkBindSparseInfo;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- waitSemaphoreCount is the number of semaphores upon which to wait before executing the sparse binding operations for the batch.
- pWaitSemaphores is a pointer to an array of semaphores upon which to wait on before the sparse binding operations for this batch begin execution. If semaphores to wait on are provided, they define a semaphore wait operation.
- bufferBindCount is the number of sparse buffer bindings to perform in the batch.

- pBufferBinds is a pointer to an array of VkSparseBufferMemoryBindInfo structures.
- imageOpaqueBindCount is the number of opaque sparse image bindings to perform.
- pImageOpaqueBinds is a pointer to an array of VkSparseImageOpaqueMemoryBindInfo structures, indicating opaque sparse image bindings to perform.
- imageBindCount is the number of sparse image bindings to perform.
- pImageBinds is a pointer to an array of VkSparseImageMemoryBindInfo structures, indicating sparse image bindings to perform.
- signalSemaphoreCount is the number of semaphores to be signaled once the sparse binding operations specified by the structure have completed execution.
- pSignalSemaphores is a pointer to an array of semaphores which will be signaled when the sparse binding operations for this batch have completed execution. If semaphores to be signaled are provided, they define a semaphore signal operation.

## Valid Usage

- VUID-VkBindSparseInfo-pWaitSemaphores-03246
  If any element of pWaitSemaphores or pSignalSemaphores was created with a VkSemaphoreType of VK_SEMAPHORE_TYPE_TIMELINE then the pNext chain **must** include a VkTimelineSemaphoreSubmitInfo structure
- VUID-VkBindSparseInfo-pNext-03247
  If the pNext chain of this structure includes a VkTimelineSemaphoreSubmitInfo structure and any element of pWaitSemaphores was created with a VkSemaphoreType of VK_SEMAPHORE_TYPE_TIMELINE then its waitSemaphoreValueCount member **must** equal waitSemaphoreCount
- VUID-VkBindSparseInfo-pNext-03248
  If the pNext chain of this structure includes a VkTimelineSemaphoreSubmitInfo structure and any element of pSignalSemaphores was created with a VkSemaphoreType of VK_SEMAPHORE_TYPE_TIMELINE then its signalSemaphoreValueCount member **must** equal signalSemaphoreCount
- VUID-VkBindSparseInfo-pSignalSemaphores-03249
  For each element of pSignalSemaphores created with a VkSemaphoreType of VK_SEMAPHORE_TYPE_TIMELINE the corresponding element of VkTimelineSemaphoreSubmitInfo::pSignalSemaphoreValues **must** have a value greater than the current value of the semaphore when the semaphore signal operation is executed
- VUID-VkBindSparseInfo-pWaitSemaphores-03250
  For each element of pWaitSemaphores created with a VkSemaphoreType of VK_SEMAPHORE_TYPE_TIMELINE the corresponding element of VkTimelineSemaphoreSubmitInfo::pWaitSemaphoreValues **must** have a value which does not differ from the current value of the semaphore or from the value of any outstanding semaphore wait or signal operation on that semaphore by more than maxTimelineSemaphoreValueDifference
- VUID-VkBindSparseInfo-pSignalSemaphores-03251

For each element of pSignalSemaphores created with a VkSemaphoreType of VK_SEMAPHORE_TYPE_TIMELINE the corresponding element of VkTimelineSemaphoreSubmitInfo::pSignalSemaphoreValues **must** have a value which does not differ from the current value of the semaphore or from the value of any outstanding semaphore wait or signal operation on that semaphore by more than maxTimelineSemaphoreValueDifference

### Valid Usage (Implicit)

- VUID-VkBindSparseInfo-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_BIND_SPARSE_INFO
- VUID-VkBindSparseInfo-pNext-pNext
  Each pNext member of any structure (including this one) in the pNext chain **must** be either NULL or a pointer to a valid instance of VkDeviceGroupBindSparseInfo, VkFrameBoundaryEXT, or VkTimelineSemaphoreSubmitInfo
- VUID-VkBindSparseInfo-sType-unique
  The sType value of each struct in the pNext chain **must** be unique
- VUID-VkBindSparseInfo-pWaitSemaphores-parameter
  If waitSemaphoreCount is not 0, pWaitSemaphores **must** be a valid pointer to an array of waitSemaphoreCount valid VkSemaphore handles
- VUID-VkBindSparseInfo-pBufferBinds-parameter
  If bufferBindCount is not 0, pBufferBinds **must** be a valid pointer to an array of bufferBindCount valid VkSparseBufferMemoryBindInfo structures
- VUID-VkBindSparseInfo-pImageOpaqueBinds-parameter
  If imageOpaqueBindCount is not 0, pImageOpaqueBinds **must** be a valid pointer to an array of imageOpaqueBindCount valid VkSparseImageOpaqueMemoryBindInfo structures
- VUID-VkBindSparseInfo-pImageBinds-parameter
  If imageBindCount is not 0, pImageBinds **must** be a valid pointer to an array of imageBindCount valid VkSparseImageMemoryBindInfo structures
- VUID-VkBindSparseInfo-pSignalSemaphores-parameter
  If signalSemaphoreCount is not 0, pSignalSemaphores **must** be a valid pointer to an array of signalSemaphoreCount valid VkSemaphore handles
- VUID-VkBindSparseInfo-commonparent
  Both of the elements of pSignalSemaphores, and the elements of pWaitSemaphores that are valid handles of non-ignored parameters **must** have been created, allocated, or retrieved from the same VkDevice

To specify the values to use when waiting for and signaling semaphores created with a VkSemaphoreType of VK_SEMAPHORE_TYPE_TIMELINE, add a VkTimelineSemaphoreSubmitInfo structure to the pNext chain of the VkBindSparseInfo structure.

If the pNext chain of VkBindSparseInfo includes a VkDeviceGroupBindSparseInfo structure, then that structure includes device indices specifying which instance of the resources and memory are

bound.

The VkDeviceGroupBindSparseInfo structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkDeviceGroupBindSparseInfo {
    VkStructureType    sType;
    const void*        pNext;
    uint32_t           resourceDeviceIndex;
    uint32_t           memoryDeviceIndex;
} VkDeviceGroupBindSparseInfo;
```

or the equivalent

```
// Provided by VK_KHR_device_group
typedef VkDeviceGroupBindSparseInfo VkDeviceGroupBindSparseInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- resourceDeviceIndex is a device index indicating which instance of the resource is bound.
- memoryDeviceIndex is a device index indicating which instance of the memory the resource instance is bound to.

These device indices apply to all buffer and image memory binds included in the batch pointing to this structure. The semaphore waits and signals for the batch are executed only by the physical device specified by the resourceDeviceIndex.

If this structure is not present, resourceDeviceIndex and memoryDeviceIndex are assumed to be zero.

### Valid Usage

- VUID-VkDeviceGroupBindSparseInfo-resourceDeviceIndex-01118
  resourceDeviceIndex and memoryDeviceIndex **must** both be valid device indices
- VUID-VkDeviceGroupBindSparseInfo-memoryDeviceIndex-01119
  Each memory allocation bound in this batch **must** have allocated an instance for memoryDeviceIndex

### Valid Usage (Implicit)

- VUID-VkDeviceGroupBindSparseInfo-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_DEVICE_GROUP_BIND_SPARSE_INFO

# Chapter 34. Window System Integration (WSI)

This chapter discusses the window system integration (WSI) between the Vulkan API and the various forms of displaying the results of rendering to a user. Since the Vulkan API **can** be used without displaying results, WSI is provided through the use of optional Vulkan extensions. This chapter provides an overview of WSI. See the appendix for additional details of each WSI extension, including which extensions **must** be enabled in order to use each of the functions described in this chapter.

## 34.1. WSI Platform

A platform is an abstraction for a window system, OS, etc. Some examples include MS Windows, Android, and Wayland. The Vulkan API **may** be integrated in a unique manner for each platform.

The Vulkan API does not define any type of platform object. Platform-specific WSI extensions are defined, each containing platform-specific functions for using WSI. Use of these extensions is guarded by preprocessor symbols as defined in the Window System-Specific Header Control appendix.

In order for an application to be compiled to use WSI with a given platform, it **must** either:

- `#define` the appropriate preprocessor symbol prior to including the `vulkan.h` header file, or
- include `vulkan_core.h` and any native platform headers, followed by the appropriate platform-specific header.

The preprocessor symbols and platform-specific headers are defined in the Window System Extensions and Headers table.

Each platform-specific extension is an instance extension. The application **must** enable instance extensions with `vkCreateInstance` before using them.

## 34.2. WSI Surface

Native platform surface or window objects are abstracted by surface objects, which are represented by `VkSurfaceKHR` handles:

```
// Provided by VK_KHR_surface
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkSurfaceKHR)
```

The `VK_KHR_surface` extension declares the `VkSurfaceKHR` object, and provides a function for destroying `VkSurfaceKHR` objects. Separate platform-specific extensions each provide a function for creating a `VkSurfaceKHR` object for the respective platform. From the application's perspective this is an opaque handle, just like the handles of other Vulkan objects.

**NOTE** On certain platforms, the Vulkan loader and ICDs **may** have conventions that treat

the handle as a pointer to a structure containing the platform-specific information about the surface. This will be described in the documentation for the loader-ICD interface, and in the vk_icd.h header file of the LoaderAndTools source-code repository. This does not affect the loader-layer interface; layers **may** wrap VkSurfaceKHR objects.

### 34.2.1. Android Platform

To create a VkSurfaceKHR object for an Android native window, call:

```
// Provided by VK_KHR_android_surface
VkResult vkCreateAndroidSurfaceKHR(
    VkInstance                                  instance,
    const VkAndroidSurfaceCreateInfoKHR*        pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkSurfaceKHR*                               pSurface);
```

- instance is the instance to associate the surface with.
- pCreateInfo is a pointer to a VkAndroidSurfaceCreateInfoKHR structure containing parameters affecting the creation of the surface object.
- pAllocator is the allocator used for host memory allocated for the surface object when there is no more specific allocator available (see Memory Allocation).
- pSurface is a pointer to a VkSurfaceKHR handle in which the created surface object is returned.

During the lifetime of a surface created using a particular ANativeWindow handle any attempts to create another surface for the same ANativeWindow and any attempts to connect to the same ANativeWindow through other platform mechanisms will fail.

**NOTE** In particular, only one VkSurfaceKHR **can** exist at a time for a given window. Similarly, a native window **cannot** be used by both a VkSurfaceKHR and EGLSurface simultaneously.

If successful, vkCreateAndroidSurfaceKHR increments the ANativeWindow's reference count, and vkDestroySurfaceKHR will decrement it.

On Android, when a swapchain's imageExtent does not match the surface's currentExtent, the presentable images will be scaled to the surface's dimensions during presentation. minImageExtent is (1,1), and maxImageExtent is the maximum image size supported by the consumer. For the system compositor, currentExtent is the window size (i.e. the consumer's preferred size).

#### Valid Usage (Implicit)

- VUID-vkCreateAndroidSurfaceKHR-instance-parameter
  instance **must** be a valid VkInstance handle
- VUID-vkCreateAndroidSurfaceKHR-pCreateInfo-parameter
  pCreateInfo **must** be a valid pointer to a valid VkAndroidSurfaceCreateInfoKHR structure

- VUID-vkCreateAndroidSurfaceKHR-pAllocator-parameter
  If `pAllocator` is not `NULL`, `pAllocator` **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkCreateAndroidSurfaceKHR-pSurface-parameter
  `pSurface` **must** be a valid pointer to a VkSurfaceKHR handle

## Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_NATIVE_WINDOW_IN_USE_KHR`

The `VkAndroidSurfaceCreateInfoKHR` structure is defined as:

```
// Provided by VK_KHR_android_surface
typedef struct VkAndroidSurfaceCreateInfoKHR {
    VkStructureType                 sType;
    const void*                     pNext;
    VkAndroidSurfaceCreateFlagsKHR  flags;
    struct ANativeWindow*           window;
} VkAndroidSurfaceCreateInfoKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is reserved for future use.
- `window` is a pointer to the ANativeWindow to associate the surface with.

## Valid Usage

- VUID-VkAndroidSurfaceCreateInfoKHR-window-01248
  `window` **must** point to a valid Android ANativeWindow

## Valid Usage (Implicit)

- VUID-VkAndroidSurfaceCreateInfoKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_ANDROID_SURFACE_CREATE_INFO_KHR`
- VUID-VkAndroidSurfaceCreateInfoKHR-pNext-pNext
  `pNext` **must** be `NULL`

- VUID-VkAndroidSurfaceCreateInfoKHR-flags-zerobitmask
  `flags` **must** be `0`

To remove an unnecessary compile time dependency, an incomplete type definition of ANativeWindow is provided in the Vulkan headers:

```
// Provided by VK_KHR_android_surface
struct ANativeWindow;
```

The actual ANativeWindow type is defined in Android NDK headers.

```
// Provided by VK_KHR_android_surface
typedef VkFlags VkAndroidSurfaceCreateFlagsKHR;
```

`VkAndroidSurfaceCreateFlagsKHR` is a bitmask type for setting a mask, but is currently reserved for future use.

### 34.2.2. Wayland Platform

To create a `VkSurfaceKHR` object for a Wayland surface, call:

```
// Provided by VK_KHR_wayland_surface
VkResult vkCreateWaylandSurfaceKHR(
    VkInstance                                  instance,
    const VkWaylandSurfaceCreateInfoKHR*        pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkSurfaceKHR*                               pSurface);
```

- `instance` is the instance to associate the surface with.
- `pCreateInfo` is a pointer to a VkWaylandSurfaceCreateInfoKHR structure containing parameters affecting the creation of the surface object.
- `pAllocator` is the allocator used for host memory allocated for the surface object when there is no more specific allocator available (see Memory Allocation).
- `pSurface` is a pointer to a VkSurfaceKHR handle in which the created surface object is returned.

**Valid Usage (Implicit)**

- VUID-vkCreateWaylandSurfaceKHR-instance-parameter
  `instance` **must** be a valid VkInstance handle
- VUID-vkCreateWaylandSurfaceKHR-pCreateInfo-parameter
  `pCreateInfo` **must** be a valid pointer to a valid VkWaylandSurfaceCreateInfoKHR structure
- VUID-vkCreateWaylandSurfaceKHR-pAllocator-parameter
  If `pAllocator` is not `NULL`, `pAllocator` **must** be a valid pointer to a valid

VkAllocationCallbacks structure

- VUID-vkCreateWaylandSurfaceKHR-pSurface-parameter
  `pSurface` **must** be a valid pointer to a VkSurfaceKHR handle

## Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`

The `VkWaylandSurfaceCreateInfoKHR` structure is defined as:

```
// Provided by VK_KHR_wayland_surface
typedef struct VkWaylandSurfaceCreateInfoKHR {
    VkStructureType                 sType;
    const void*                     pNext;
    VkWaylandSurfaceCreateFlagsKHR  flags;
    struct wl_display*              display;
    struct wl_surface*              surface;
} VkWaylandSurfaceCreateInfoKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is reserved for future use.
- `display` and `surface` are pointers to the Wayland `wl_display` and `wl_surface` to associate the surface with.

## Valid Usage

- VUID-VkWaylandSurfaceCreateInfoKHR-display-01304
  `display` **must** point to a valid Wayland `wl_display`
- VUID-VkWaylandSurfaceCreateInfoKHR-surface-01305
  `surface` **must** point to a valid Wayland `wl_surface`

## Valid Usage (Implicit)

- VUID-VkWaylandSurfaceCreateInfoKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_WAYLAND_SURFACE_CREATE_INFO_KHR`
- VUID-VkWaylandSurfaceCreateInfoKHR-pNext-pNext

pNext **must** be NULL

- VUID-VkWaylandSurfaceCreateInfoKHR-flags-zerobitmask
  flags **must** be 0

On Wayland, currentExtent is the special value (0xFFFFFFFF, 0xFFFFFFFF), indicating that the surface size will be determined by the extent of a swapchain targeting the surface. Whatever the application sets a swapchain's imageExtent to will be the size of the window, after the first image is presented. minImageExtent is (1,1), and maxImageExtent is the maximum supported surface size. Any calls to vkGetPhysicalDeviceSurfacePresentModesKHR on a surface created with vkCreateWaylandSurfaceKHR are **required** to return VK_PRESENT_MODE_MAILBOX_KHR as one of the valid present modes.

Some Vulkan functions **may** send protocol over the specified wl_display connection when using a swapchain or presentable images created from a VkSurfaceKHR referring to a wl_surface. Applications **must** therefore ensure that both the wl_display and the wl_surface remain valid for the lifetime of any VkSwapchainKHR objects created from a particular wl_display and wl_surface. Also, calling vkQueuePresentKHR will result in Vulkan sending wl_surface.commit requests to the underlying wl_surface of each The wl_surface.attach, wl_surface.damage, and wl_surface.commit requests **must** be issued by the implementation during the call to vkQueuePresentKHR and **must** not be issued by the implementation outside of vkQueuePresentKHR. This ensures that any Wayland requests sent by the client after the call to vkQueuePresentKHR returns will be received by the compositor after the wl_surface.commit. Regardless of the mode of swapchain creation, a new wl_event_queue **must** be created for each successful vkCreateWaylandSurfaceKHR call, and every Wayland object created by the implementation **must** be assigned to this event queue. If the platform provides Wayland 1.11 or greater, this **must** be implemented by the use of Wayland proxy object wrappers, to avoid race conditions.

If the application wishes to synchronize any window changes with a particular frame, such requests **must** be sent to the Wayland display server prior to calling vkQueuePresentKHR.

```
// Provided by VK_KHR_wayland_surface
typedef VkFlags VkWaylandSurfaceCreateFlagsKHR;
```

VkWaylandSurfaceCreateFlagsKHR is a bitmask type for setting a mask, but is currently reserved for future use.

### 34.2.3. Win32 Platform

To create a VkSurfaceKHR object for a Win32 window, call:

```
// Provided by VK_KHR_win32_surface
VkResult vkCreateWin32SurfaceKHR(
    VkInstance                                  instance,
    const VkWin32SurfaceCreateInfoKHR*          pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkSurfaceKHR*                               pSurface);
```

- `instance` is the instance to associate the surface with.
- `pCreateInfo` is a pointer to a `VkWin32SurfaceCreateInfoKHR` structure containing parameters affecting the creation of the surface object.
- `pAllocator` is the allocator used for host memory allocated for the surface object when there is no more specific allocator available (see Memory Allocation).
- `pSurface` is a pointer to a VkSurfaceKHR handle in which the created surface object is returned.

### Valid Usage (Implicit)

- VUID-vkCreateWin32SurfaceKHR-instance-parameter
  `instance` **must** be a valid VkInstance handle
- VUID-vkCreateWin32SurfaceKHR-pCreateInfo-parameter
  `pCreateInfo` **must** be a valid pointer to a valid VkWin32SurfaceCreateInfoKHR structure
- VUID-vkCreateWin32SurfaceKHR-pAllocator-parameter
  If `pAllocator` is not `NULL`, `pAllocator` **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkCreateWin32SurfaceKHR-pSurface-parameter
  `pSurface` **must** be a valid pointer to a VkSurfaceKHR handle

### Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`

Some Vulkan functions **may** call the `SendMessage` system API when interacting with a `VkSurfaceKHR` through a `VkSwapchainKHR`. In a multithreaded environment, calling `SendMessage` from a thread that is not the thread associated with `pCreateInfo->hwnd` will block until the application has processed the window message. Thus, applications **should** either call these Vulkan functions on the message pump thread, or make sure their message pump is actively running. Failing to do so **may** result in deadlocks.

The functions subject to this requirement are:

- vkCreateSwapchainKHR
- vkDestroySwapchainKHR
- vkAcquireNextImageKHR and vkAcquireNextImage2KHR
- vkQueuePresentKHR
- vkReleaseSwapchainImagesEXT

- vkAcquireFullScreenExclusiveModeEXT
- vkReleaseFullScreenExclusiveModeEXT
- vkSetHdrMetadataEXT

The `VkWin32SurfaceCreateInfoKHR` structure is defined as:

```
// Provided by VK_KHR_win32_surface
typedef struct VkWin32SurfaceCreateInfoKHR {
    VkStructureType                 sType;
    const void*                     pNext;
    VkWin32SurfaceCreateFlagsKHR    flags;
    HINSTANCE                       hinstance;
    HWND                            hwnd;
} VkWin32SurfaceCreateInfoKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is reserved for future use.
- `hinstance` is the Win32 `HINSTANCE` for the window to associate the surface with.
- `hwnd` is the Win32 `HWND` for the window to associate the surface with.

### Valid Usage

- VUID-VkWin32SurfaceCreateInfoKHR-hinstance-01307
  `hinstance` **must** be a valid Win32 `HINSTANCE`
- VUID-VkWin32SurfaceCreateInfoKHR-hwnd-01308
  `hwnd` **must** be a valid Win32 `HWND`

### Valid Usage (Implicit)

- VUID-VkWin32SurfaceCreateInfoKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_WIN32_SURFACE_CREATE_INFO_KHR`
- VUID-VkWin32SurfaceCreateInfoKHR-pNext-pNext
  `pNext` **must** be `NULL`
- VUID-VkWin32SurfaceCreateInfoKHR-flags-zerobitmask
  `flags` **must** be `0`

With Win32, `minImageExtent`, `maxImageExtent`, and `currentExtent` **must** always equal the window size.

The `currentExtent` of a Win32 surface **must** have both `width` and `height` greater than 0, or both of them 0.

**NOTE**

Due to above restrictions, unless VkSwapchainPresentScalingCreateInfoEXT is used to specify handling of disparities between surface and swapchain dimensions, it is only possible to create a new swapchain on this platform with imageExtent being equal to the current size of the window, as reported in VkSurfaceCapabilitiesKHR ::currentExtent.

The window size **may** become (0, 0) on this platform (e.g. when the window is minimized), and so a swapchain **cannot** be created until the size changes.

```
// Provided by VK_KHR_win32_surface
typedef VkFlags VkWin32SurfaceCreateFlagsKHR;
```

VkWin32SurfaceCreateFlagsKHR is a bitmask type for setting a mask, but is currently reserved for future use.

### 34.2.4. XCB Platform

To create a VkSurfaceKHR object for an X11 window, using the XCB client-side library, call:

```
// Provided by VK_KHR_xcb_surface
VkResult vkCreateXcbSurfaceKHR(
    VkInstance                                  instance,
    const VkXcbSurfaceCreateInfoKHR*            pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkSurfaceKHR*                               pSurface);
```

- instance is the instance to associate the surface with.
- pCreateInfo is a pointer to a VkXcbSurfaceCreateInfoKHR structure containing parameters affecting the creation of the surface object.
- pAllocator is the allocator used for host memory allocated for the surface object when there is no more specific allocator available (see Memory Allocation).
- pSurface is a pointer to a VkSurfaceKHR handle in which the created surface object is returned.

**Valid Usage (Implicit)**

- VUID-vkCreateXcbSurfaceKHR-instance-parameter
  instance **must** be a valid VkInstance handle
- VUID-vkCreateXcbSurfaceKHR-pCreateInfo-parameter
  pCreateInfo **must** be a valid pointer to a valid VkXcbSurfaceCreateInfoKHR structure
- VUID-vkCreateXcbSurfaceKHR-pAllocator-parameter
  If pAllocator is not NULL, pAllocator **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkCreateXcbSurfaceKHR-pSurface-parameter
  pSurface **must** be a valid pointer to a VkSurfaceKHR handle

## Return Codes

**Success**

- VK_SUCCESS

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY

The VkXcbSurfaceCreateInfoKHR structure is defined as:

```
// Provided by VK_KHR_xcb_surface
typedef struct VkXcbSurfaceCreateInfoKHR {
    VkStructureType                 sType;
    const void*                     pNext;
    VkXcbSurfaceCreateFlagsKHR      flags;
    xcb_connection_t*               connection;
    xcb_window_t                    window;
} VkXcbSurfaceCreateInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- flags is reserved for future use.
- connection is a pointer to an xcb_connection_t to the X server.
- window is the xcb_window_t for the X11 window to associate the surface with.

## Valid Usage

- VUID-VkXcbSurfaceCreateInfoKHR-connection-01310
  connection **must** point to a valid X11 xcb_connection_t
- VUID-VkXcbSurfaceCreateInfoKHR-window-01311
  window **must** be a valid X11 xcb_window_t

## Valid Usage (Implicit)

- VUID-VkXcbSurfaceCreateInfoKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_XCB_SURFACE_CREATE_INFO_KHR
- VUID-VkXcbSurfaceCreateInfoKHR-pNext-pNext
  pNext **must** be NULL
- VUID-VkXcbSurfaceCreateInfoKHR-flags-zerobitmask
  flags **must** be 0

With Xcb, minImageExtent, maxImageExtent, and currentExtent **must** always equal the window size.

The currentExtent of an Xcb surface **must** have both width and height greater than 0, or both of them 0.

**NOTE**

Due to above restrictions, unless VkSwapchainPresentScalingCreateInfoEXT is used to specify handling of disparities between surface and swapchain dimensions, it is only possible to create a new swapchain on this platform with imageExtent being equal to the current size of the window, as reported in VkSurfaceCapabilitiesKHR::currentExtent.

The window size **may** become (0, 0) on this platform (e.g. when the window is minimized), and so a swapchain **cannot** be created until the size changes.

Some Vulkan functions **may** send protocol over the specified xcb connection when using a swapchain or presentable images created from a VkSurfaceKHR referring to an xcb window. Applications **must** therefore ensure the xcb connection is available to Vulkan for the duration of any functions that manipulate such swapchains or their presentable images, and any functions that build or queue command buffers that operate on such presentable images. Specifically, applications using Vulkan with xcb-based swapchains **must**

- Avoid holding a server grab on an xcb connection while waiting for Vulkan operations to complete using a swapchain derived from a different xcb connection referring to the same X server instance. Failing to do so **may** result in deadlock.

```
// Provided by VK_KHR_xcb_surface
typedef VkFlags VkXcbSurfaceCreateFlagsKHR;
```

VkXcbSurfaceCreateFlagsKHR is a bitmask type for setting a mask, but is currently reserved for future use.

### 34.2.5. Xlib Platform

To create a VkSurfaceKHR object for an X11 window, using the Xlib client-side library, call:

```
// Provided by VK_KHR_xlib_surface
VkResult vkCreateXlibSurfaceKHR(
    VkInstance                                  instance,
    const VkXlibSurfaceCreateInfoKHR*           pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkSurfaceKHR*                               pSurface);
```

- instance is the instance to associate the surface with.
- pCreateInfo is a pointer to a VkXlibSurfaceCreateInfoKHR structure containing the parameters affecting the creation of the surface object.
- pAllocator is the allocator used for host memory allocated for the surface object when there is

no more specific allocator available (see Memory Allocation).

- pSurface is a pointer to a VkSurfaceKHR handle in which the created surface object is returned.

### Valid Usage (Implicit)

- VUID-vkCreateXlibSurfaceKHR-instance-parameter
  instance **must** be a valid VkInstance handle
- VUID-vkCreateXlibSurfaceKHR-pCreateInfo-parameter
  pCreateInfo **must** be a valid pointer to a valid VkXlibSurfaceCreateInfoKHR structure
- VUID-vkCreateXlibSurfaceKHR-pAllocator-parameter
  If pAllocator is not NULL, pAllocator **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkCreateXlibSurfaceKHR-pSurface-parameter
  pSurface **must** be a valid pointer to a VkSurfaceKHR handle

### Return Codes

**Success**

- VK_SUCCESS

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY

The VkXlibSurfaceCreateInfoKHR structure is defined as:

```
// Provided by VK_KHR_xlib_surface
typedef struct VkXlibSurfaceCreateInfoKHR {
    VkStructureType                 sType;
    const void*                     pNext;
    VkXlibSurfaceCreateFlagsKHR     flags;
    Display*                        dpy;
    Window                          window;
} VkXlibSurfaceCreateInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- flags is reserved for future use.
- dpy is a pointer to an Xlib Display connection to the X server.
- window is an Xlib Window to associate the surface with.

### Valid Usage

- VUID-VkXlibSurfaceCreateInfoKHR-dpy-01313
  `dpy` **must** point to a valid Xlib `Display`
- VUID-VkXlibSurfaceCreateInfoKHR-window-01314
  `window` **must** be a valid Xlib `Window`

### Valid Usage (Implicit)

- VUID-VkXlibSurfaceCreateInfoKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_XLIB_SURFACE_CREATE_INFO_KHR`
- VUID-VkXlibSurfaceCreateInfoKHR-pNext-pNext
  `pNext` **must** be `NULL`
- VUID-VkXlibSurfaceCreateInfoKHR-flags-zerobitmask
  `flags` **must** be `0`

With Xlib, `minImageExtent`, `maxImageExtent`, and `currentExtent` **must** always equal the window size.

The `currentExtent` of an Xlib surface **must** have both `width` and `height` greater than 0, or both of them 0.

**NOTE**

Due to above restrictions, unless VkSwapchainPresentScalingCreateInfoEXT is used to specify handling of disparities between surface and swapchain dimensions, it is only possible to create a new swapchain on this platform with `imageExtent` being equal to the current size of the window, as reported in VkSurfaceCapabilitiesKHR::`currentExtent`.

The window size **may** become (0, 0) on this platform (e.g. when the window is minimized), and so a swapchain **cannot** be created until the size changes.

Some Vulkan functions **may** send protocol over the specified Xlib `Display` connection when using a swapchain or presentable images created from a VkSurfaceKHR referring to an Xlib window. Applications **must** therefore ensure the display connection is available to Vulkan for the duration of any functions that manipulate such swapchains or their presentable images, and any functions that build or queue command buffers that operate on such presentable images. Specifically, applications using Vulkan with Xlib-based swapchains **must**

- Avoid holding a server grab on a display connection while waiting for Vulkan operations to complete using a swapchain derived from a different display connection referring to the same X server instance. Failing to do so **may** result in deadlock.

Some implementations may require threads to implement some presentation modes so applications **must** call `XInitThreads`() before calling any other Xlib functions.

```
// Provided by VK_KHR_xlib_surface
typedef VkFlags VkXlibSurfaceCreateFlagsKHR;
```

VkXlibSurfaceCreateFlagsKHR is a bitmask type for setting a mask, but is currently reserved for future use.

### 34.2.6. DirectFB Platform

To create a VkSurfaceKHR object for a DirectFB surface, call:

```
// Provided by VK_EXT_directfb_surface
VkResult vkCreateDirectFBSurfaceEXT(
    VkInstance                                  instance,
    const VkDirectFBSurfaceCreateInfoEXT*       pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkSurfaceKHR*                               pSurface);
```

- instance is the instance to associate the surface with.
- pCreateInfo is a pointer to a VkDirectFBSurfaceCreateInfoEXT structure containing parameters affecting the creation of the surface object.
- pAllocator is the allocator used for host memory allocated for the surface object when there is no more specific allocator available (see Memory Allocation).
- pSurface is a pointer to a VkSurfaceKHR handle in which the created surface object is returned.

#### Valid Usage (Implicit)

- VUID-vkCreateDirectFBSurfaceEXT-instance-parameter
  instance **must** be a valid VkInstance handle
- VUID-vkCreateDirectFBSurfaceEXT-pCreateInfo-parameter
  pCreateInfo **must** be a valid pointer to a valid VkDirectFBSurfaceCreateInfoEXT structure
- VUID-vkCreateDirectFBSurfaceEXT-pAllocator-parameter
  If pAllocator is not NULL, pAllocator **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkCreateDirectFBSurfaceEXT-pSurface-parameter
  pSurface **must** be a valid pointer to a VkSurfaceKHR handle

#### Return Codes

**Success**

- VK_SUCCESS

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY

- VK_ERROR_OUT_OF_DEVICE_MEMORY

The VkDirectFBSurfaceCreateInfoEXT structure is defined as:

```
// Provided by VK_EXT_directfb_surface
typedef struct VkDirectFBSurfaceCreateInfoEXT {
    VkStructureType                    sType;
    const void*                        pNext;
    VkDirectFBSurfaceCreateFlagsEXT    flags;
    IDirectFB*                         dfb;
    IDirectFBSurface*                  surface;
} VkDirectFBSurfaceCreateInfoEXT;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- flags is reserved for future use.
- dfb is a pointer to the IDirectFB main interface of DirectFB.
- surface is a pointer to a IDirectFBSurface surface interface.

### Valid Usage

- VUID-VkDirectFBSurfaceCreateInfoEXT-dfb-04117
  dfb **must** point to a valid DirectFB IDirectFB
- VUID-VkDirectFBSurfaceCreateInfoEXT-surface-04118
  surface **must** point to a valid DirectFB IDirectFBSurface

### Valid Usage (Implicit)

- VUID-VkDirectFBSurfaceCreateInfoEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_DIRECTFB_SURFACE_CREATE_INFO_EXT
- VUID-VkDirectFBSurfaceCreateInfoEXT-pNext-pNext
  pNext **must** be NULL
- VUID-VkDirectFBSurfaceCreateInfoEXT-flags-zerobitmask
  flags **must** be 0

With DirectFB, minImageExtent, maxImageExtent, and currentExtent **must** always equal the surface size.

```
// Provided by VK_EXT_directfb_surface
typedef VkFlags VkDirectFBSurfaceCreateFlagsEXT;
```

VkDirectFBSurfaceCreateFlagsEXT is a bitmask type for setting a mask, but is currently reserved for future use.

### 34.2.7. Fuchsia Platform

To create a VkSurfaceKHR object for a Fuchsia ImagePipe, call:

```
// Provided by VK_FUCHSIA_imagepipe_surface
VkResult vkCreateImagePipeSurfaceFUCHSIA(
    VkInstance                                  instance,
    const VkImagePipeSurfaceCreateInfoFUCHSIA*  pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkSurfaceKHR*                               pSurface);
```

- instance is the instance to associate with the surface.
- pCreateInfo is a pointer to a VkImagePipeSurfaceCreateInfoFUCHSIA structure containing parameters affecting the creation of the surface object.
- pAllocator is the allocator used for host memory allocated for the surface object when there is no more specific allocator available (see Memory Allocation).
- pSurface is a pointer to a VkSurfaceKHR handle in which the created surface object is returned.

#### Valid Usage (Implicit)

- VUID-vkCreateImagePipeSurfaceFUCHSIA-instance-parameter
  instance **must** be a valid VkInstance handle
- VUID-vkCreateImagePipeSurfaceFUCHSIA-pCreateInfo-parameter
  pCreateInfo **must** be a valid pointer to a valid VkImagePipeSurfaceCreateInfoFUCHSIA structure
- VUID-vkCreateImagePipeSurfaceFUCHSIA-pAllocator-parameter
  If pAllocator is not NULL, pAllocator **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkCreateImagePipeSurfaceFUCHSIA-pSurface-parameter
  pSurface **must** be a valid pointer to a VkSurfaceKHR handle

#### Return Codes

**Success**

- VK_SUCCESS

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY

The VkImagePipeSurfaceCreateInfoFUCHSIA structure is defined as:

```
// Provided by VK_FUCHSIA_imagepipe_surface
typedef struct VkImagePipeSurfaceCreateInfoFUCHSIA {
    VkStructureType                         sType;
    const void*                             pNext;
    VkImagePipeSurfaceCreateFlagsFUCHSIA    flags;
    zx_handle_t                             imagePipeHandle;
} VkImagePipeSurfaceCreateInfoFUCHSIA;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- flags is reserved for future use.
- imagePipeHandle is a zx_handle_t referring to the ImagePipe to associate with the surface.

### Valid Usage

- VUID-VkImagePipeSurfaceCreateInfoFUCHSIA-imagePipeHandle-04863
  imagePipeHandle **must** be a valid zx_handle_t

### Valid Usage (Implicit)

- VUID-VkImagePipeSurfaceCreateInfoFUCHSIA-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_IMAGEPIPE_SURFACE_CREATE_INFO_FUCHSIA
- VUID-VkImagePipeSurfaceCreateInfoFUCHSIA-pNext-pNext
  pNext **must** be NULL
- VUID-VkImagePipeSurfaceCreateInfoFUCHSIA-flags-zerobitmask
  flags **must** be 0

On Fuchsia, the surface currentExtent is the special value (0xFFFFFFFF, 0xFFFFFFFF), indicating that the surface size will be determined by the extent of a swapchain targeting the surface.

```
// Provided by VK_FUCHSIA_imagepipe_surface
typedef VkFlags VkImagePipeSurfaceCreateFlagsFUCHSIA;
```

VkImagePipeSurfaceCreateFlagsFUCHSIA is a bitmask type for setting a mask, but is currently reserved for future use.

### 34.2.8. Google Games Platform

To create a VkSurfaceKHR object for a Google Games Platform stream descriptor, call:

```
// Provided by VK_GGP_stream_descriptor_surface
VkResult vkCreateStreamDescriptorSurfaceGGP(
    VkInstance                                  instance,
    const VkStreamDescriptorSurfaceCreateInfoGGP* pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkSurfaceKHR*                               pSurface);
```

- `instance` is the instance to associate with the surface.
- `pCreateInfo` is a pointer to a `VkStreamDescriptorSurfaceCreateInfoGGP` structure containing parameters that affect the creation of the surface object.
- `pAllocator` is the allocator used for host memory allocated for the surface object when there is no more specific allocator available (see Memory Allocation).
- `pSurface` is a pointer to a VkSurfaceKHR handle in which the created surface object is returned.

### Valid Usage (Implicit)

- VUID-vkCreateStreamDescriptorSurfaceGGP-instance-parameter
  `instance` **must** be a valid VkInstance handle
- VUID-vkCreateStreamDescriptorSurfaceGGP-pCreateInfo-parameter
  `pCreateInfo` **must** be a valid pointer to a valid VkStreamDescriptorSurfaceCreateInfoGGP structure
- VUID-vkCreateStreamDescriptorSurfaceGGP-pAllocator-parameter
  If `pAllocator` is not `NULL`, `pAllocator` **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkCreateStreamDescriptorSurfaceGGP-pSurface-parameter
  `pSurface` **must** be a valid pointer to a VkSurfaceKHR handle

### Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_NATIVE_WINDOW_IN_USE_KHR`

The `VkStreamDescriptorSurfaceCreateInfoGGP` structure is defined as:

```
// Provided by VK_GGP_stream_descriptor_surface
typedef struct VkStreamDescriptorSurfaceCreateInfoGGP {
    VkStructureType                             sType;
    const void*                                 pNext;
    VkStreamDescriptorSurfaceCreateFlagsGGP     flags;
    GgpStreamDescriptor                         streamDescriptor;
} VkStreamDescriptorSurfaceCreateInfoGGP;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is reserved for future use.
- `streamDescriptor` is a `GgpStreamDescriptor` referring to the GGP stream descriptor to associate with the surface.

### Valid Usage

- VUID-VkStreamDescriptorSurfaceCreateInfoGGP-streamDescriptor-02681
  `streamDescriptor` **must** be a valid `GgpStreamDescriptor`

### Valid Usage (Implicit)

- VUID-VkStreamDescriptorSurfaceCreateInfoGGP-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_STREAM_DESCRIPTOR_SURFACE_CREATE_INFO_GGP`
- VUID-VkStreamDescriptorSurfaceCreateInfoGGP-pNext-pNext
  `pNext` **must** be `NULL`
- VUID-VkStreamDescriptorSurfaceCreateInfoGGP-flags-zerobitmask
  `flags` **must** be `0`

On Google Games Platform, the surface extents are dynamic. The `minImageExtent` will never be greater than 1080p and the `maxImageExtent` will never be less than 1080p. The `currentExtent` will reflect the current optimal resolution.

Applications are expected to choose an appropriate size for the swapchain's `imageExtent`, within the bounds of the surface. Using the surface's `currentExtent` will offer the best performance and quality. When a swapchain's `imageExtent` does not match the surface's `currentExtent`, the presentable images are scaled to the surface's dimensions during presentation if possible and `VK_SUBOPTIMAL_KHR` is returned, otherwise presentation fails with `VK_ERROR_OUT_OF_DATE_KHR`.

```
// Provided by VK_GGP_stream_descriptor_surface
typedef VkFlags VkStreamDescriptorSurfaceCreateFlagsGGP;
```

`VkStreamDescriptorSurfaceCreateFlagsGGP` is a bitmask type for setting a mask, but is currently

reserved for future use.

### 34.2.9. iOS Platform

To create a VkSurfaceKHR object for an iOS UIView or CAMetalLayer, call:

```
// Provided by VK_MVK_ios_surface
VkResult vkCreateIOSSurfaceMVK(
    VkInstance                                  instance,
    const VkIOSSurfaceCreateInfoMVK*            pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkSurfaceKHR*                               pSurface);
```

- instance is the instance with which to associate the surface.
- pCreateInfo is a pointer to a VkIOSSurfaceCreateInfoMVK structure containing parameters affecting the creation of the surface object.
- pAllocator is the allocator used for host memory allocated for the surface object when there is no more specific allocator available (see Memory Allocation).
- pSurface is a pointer to a VkSurfaceKHR handle in which the created surface object is returned.

**NOTE** The vkCreateIOSSurfaceMVK function is considered deprecated and has been superseded by vkCreateMetalSurfaceEXT from the VK_EXT_metal_surface extension.

#### Valid Usage (Implicit)

- VUID-vkCreateIOSSurfaceMVK-instance-parameter
  instance **must** be a valid VkInstance handle
- VUID-vkCreateIOSSurfaceMVK-pCreateInfo-parameter
  pCreateInfo **must** be a valid pointer to a valid VkIOSSurfaceCreateInfoMVK structure
- VUID-vkCreateIOSSurfaceMVK-pAllocator-parameter
  If pAllocator is not NULL, pAllocator **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkCreateIOSSurfaceMVK-pSurface-parameter
  pSurface **must** be a valid pointer to a VkSurfaceKHR handle

#### Return Codes

**Success**

- VK_SUCCESS

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY

- VK_ERROR_NATIVE_WINDOW_IN_USE_KHR

The VkIOSSurfaceCreateInfoMVK structure is defined as:

```
// Provided by VK_MVK_ios_surface
typedef struct VkIOSSurfaceCreateInfoMVK {
    VkStructureType                 sType;
    const void*                     pNext;
    VkIOSSurfaceCreateFlagsMVK      flags;
    const void*                     pView;
} VkIOSSurfaceCreateInfoMVK;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- flags is reserved for future use.
- pView is a reference to either a CAMetalLayer object or a UIView object.

### Valid Usage

- VUID-VkIOSSurfaceCreateInfoMVK-pView-04143
  If pView is a CAMetalLayer object, it **must** be a valid CAMetalLayer
- VUID-VkIOSSurfaceCreateInfoMVK-pView-01316
  If pView is a UIView object, it **must** be a valid UIView, **must** be backed by a CALayer object of type CAMetalLayer, and vkCreateIOSSurfaceMVK **must** be called on the main thread

### Valid Usage (Implicit)

- VUID-VkIOSSurfaceCreateInfoMVK-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_IOS_SURFACE_CREATE_INFO_MVK
- VUID-VkIOSSurfaceCreateInfoMVK-pNext-pNext
  pNext **must** be NULL
- VUID-VkIOSSurfaceCreateInfoMVK-flags-zerobitmask
  flags **must** be 0

```
// Provided by VK_MVK_ios_surface
typedef VkFlags VkIOSSurfaceCreateFlagsMVK;
```

VkIOSSurfaceCreateFlagsMVK is a bitmask type for setting a mask, but is currently reserved for future use.

### 34.2.10. macOS Platform

To create a VkSurfaceKHR object for a macOS NSView or CAMetalLayer, call:

```
// Provided by VK_MVK_macos_surface
VkResult vkCreateMacOSSurfaceMVK(
    VkInstance                                  instance,
    const VkMacOSSurfaceCreateInfoMVK*          pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkSurfaceKHR*                               pSurface);
```

- instance is the instance with which to associate the surface.
- pCreateInfo is a pointer to a VkMacOSSurfaceCreateInfoMVK structure containing parameters affecting the creation of the surface object.
- pAllocator is the allocator used for host memory allocated for the surface object when there is no more specific allocator available (see Memory Allocation).
- pSurface is a pointer to a VkSurfaceKHR handle in which the created surface object is returned.

**NOTE** The vkCreateMacOSSurfaceMVK function is considered deprecated and has been superseded by vkCreateMetalSurfaceEXT from the VK_EXT_metal_surface extension.

#### Valid Usage (Implicit)

- VUID-vkCreateMacOSSurfaceMVK-instance-parameter
  instance **must** be a valid VkInstance handle
- VUID-vkCreateMacOSSurfaceMVK-pCreateInfo-parameter
  pCreateInfo **must** be a valid pointer to a valid VkMacOSSurfaceCreateInfoMVK structure
- VUID-vkCreateMacOSSurfaceMVK-pAllocator-parameter
  If pAllocator is not NULL, pAllocator **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkCreateMacOSSurfaceMVK-pSurface-parameter
  pSurface **must** be a valid pointer to a VkSurfaceKHR handle

#### Return Codes

**Success**

- VK_SUCCESS

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY
- VK_ERROR_NATIVE_WINDOW_IN_USE_KHR

The VkMacOSSurfaceCreateInfoMVK structure is defined as:

```
// Provided by VK_MVK_macos_surface
typedef struct VkMacOSSurfaceCreateInfoMVK {
    VkStructureType                 sType;
    const void*                     pNext;
    VkMacOSSurfaceCreateFlagsMVK    flags;
    const void*                     pView;
} VkMacOSSurfaceCreateInfoMVK;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is reserved for future use.
- `pView` is a reference to either a CAMetalLayer object or an `NSView` object.

### Valid Usage

- VUID-VkMacOSSurfaceCreateInfoMVK-pView-04144
  If `pView` is a CAMetalLayer object, it **must** be a valid CAMetalLayer
- VUID-VkMacOSSurfaceCreateInfoMVK-pView-01317
  If `pView` is an `NSView` object, it **must** be a valid `NSView`, **must** be backed by a `CALayer` object of type CAMetalLayer, and vkCreateMacOSSurfaceMVK **must** be called on the main thread

### Valid Usage (Implicit)

- VUID-VkMacOSSurfaceCreateInfoMVK-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_MACOS_SURFACE_CREATE_INFO_MVK`
- VUID-VkMacOSSurfaceCreateInfoMVK-pNext-pNext
  `pNext` **must** be `NULL`
- VUID-VkMacOSSurfaceCreateInfoMVK-flags-zerobitmask
  `flags` **must** be `0`

```
// Provided by VK_MVK_macos_surface
typedef VkFlags VkMacOSSurfaceCreateFlagsMVK;
```

`VkMacOSSurfaceCreateFlagsMVK` is a bitmask type for setting a mask, but is currently reserved for future use.

## 34.2.11. VI Platform

To create a `VkSurfaceKHR` object for an `nn::vi::Layer`, query the layer's native handle using `nn::vi::GetNativeWindow`, and then call:

```
// Provided by VK_NN_vi_surface
VkResult vkCreateViSurfaceNN(
    VkInstance                                  instance,
    const VkViSurfaceCreateInfoNN*              pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkSurfaceKHR*                               pSurface);
```

- `instance` is the instance with which to associate the surface.
- `pCreateInfo` is a pointer to a `VkViSurfaceCreateInfoNN` structure containing parameters affecting the creation of the surface object.
- `pAllocator` is the allocator used for host memory allocated for the surface object when there is no more specific allocator available (see Memory Allocation).
- `pSurface` is a pointer to a VkSurfaceKHR handle in which the created surface object is returned.

During the lifetime of a surface created using a particular `nn::vi::NativeWindowHandle`, applications **must** not attempt to create another surface for the same `nn::vi::Layer` or attempt to connect to the same `nn::vi::Layer` through other platform mechanisms.

If the native window is created with a specified size, `currentExtent` will reflect that size. In this case, applications should use the same size for the swapchain's `imageExtent`. Otherwise, the `currentExtent` will have the special value (0xFFFFFFFF, 0xFFFFFFFF), indicating that applications are expected to choose an appropriate size for the swapchain's `imageExtent` (e.g., by matching the result of a call to `nn::vi::GetDisplayResolution`).

## Valid Usage (Implicit)

- VUID-vkCreateViSurfaceNN-instance-parameter
  `instance` **must** be a valid VkInstance handle
- VUID-vkCreateViSurfaceNN-pCreateInfo-parameter
  `pCreateInfo` **must** be a valid pointer to a valid VkViSurfaceCreateInfoNN structure
- VUID-vkCreateViSurfaceNN-pAllocator-parameter
  If `pAllocator` is not `NULL`, `pAllocator` **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkCreateViSurfaceNN-pSurface-parameter
  `pSurface` **must** be a valid pointer to a VkSurfaceKHR handle

## Return Codes

### Success

- `VK_SUCCESS`

### Failure

- `VK_ERROR_OUT_OF_HOST_MEMORY`

- VK_ERROR_OUT_OF_DEVICE_MEMORY
- VK_ERROR_NATIVE_WINDOW_IN_USE_KHR

The VkViSurfaceCreateInfoNN structure is defined as:

```
// Provided by VK_NN_vi_surface
typedef struct VkViSurfaceCreateInfoNN {
    VkStructureType             sType;
    const void*                 pNext;
    VkViSurfaceCreateFlagsNN    flags;
    void*                       window;
} VkViSurfaceCreateInfoNN;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- flags is reserved for future use.
- window is the nn::vi::NativeWindowHandle for the nn::vi::Layer with which to associate the surface.

### Valid Usage

- VUID-VkViSurfaceCreateInfoNN-window-01318
  window **must** be a valid nn::vi::NativeWindowHandle

### Valid Usage (Implicit)

- VUID-VkViSurfaceCreateInfoNN-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_VI_SURFACE_CREATE_INFO_NN
- VUID-VkViSurfaceCreateInfoNN-pNext-pNext
  pNext **must** be NULL
- VUID-VkViSurfaceCreateInfoNN-flags-zerobitmask
  flags **must** be 0

```
// Provided by VK_NN_vi_surface
typedef VkFlags VkViSurfaceCreateFlagsNN;
```

VkViSurfaceCreateFlagsNN is a bitmask type for setting a mask, but is currently reserved for future use.

## 34.2.12. Metal Platform

To create a VkSurfaceKHR object for a CAMetalLayer, call:

```
// Provided by VK_EXT_metal_surface
VkResult vkCreateMetalSurfaceEXT(
    VkInstance                                  instance,
    const VkMetalSurfaceCreateInfoEXT*          pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkSurfaceKHR*                               pSurface);
```

- `instance` is the instance with which to associate the surface.
- `pCreateInfo` is a pointer to a VkMetalSurfaceCreateInfoEXT structure specifying parameters affecting the creation of the surface object.
- `pAllocator` is the allocator used for host memory allocated for the surface object when there is no more specific allocator available (see Memory Allocation).
- `pSurface` is a pointer to a `VkSurfaceKHR` handle in which the created surface object is returned.

## Valid Usage (Implicit)

- VUID-vkCreateMetalSurfaceEXT-instance-parameter
  `instance` **must** be a valid VkInstance handle
- VUID-vkCreateMetalSurfaceEXT-pCreateInfo-parameter
  `pCreateInfo` **must** be a valid pointer to a valid VkMetalSurfaceCreateInfoEXT structure
- VUID-vkCreateMetalSurfaceEXT-pAllocator-parameter
  If `pAllocator` is not `NULL`, `pAllocator` **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkCreateMetalSurfaceEXT-pSurface-parameter
  `pSurface` **must** be a valid pointer to a VkSurfaceKHR handle

## Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_NATIVE_WINDOW_IN_USE_KHR`

The VkMetalSurfaceCreateInfoEXT structure is defined as:

```
// Provided by VK_EXT_metal_surface
typedef struct VkMetalSurfaceCreateInfoEXT {
    VkStructureType                 sType;
    const void*                     pNext;
    VkMetalSurfaceCreateFlagsEXT    flags;
    const CAMetalLayer*             pLayer;
} VkMetalSurfaceCreateInfoEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is reserved for future use.
- `pLayer` is a reference to a CAMetalLayer object representing a renderable surface.

**Valid Usage (Implicit)**

- VUID-VkMetalSurfaceCreateInfoEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_METAL_SURFACE_CREATE_INFO_EXT`
- VUID-VkMetalSurfaceCreateInfoEXT-pNext-pNext
  `pNext` **must** be `NULL`
- VUID-VkMetalSurfaceCreateInfoEXT-flags-zerobitmask
  `flags` **must** be `0`

To remove an unnecessary compile time dependency, an incomplete type definition of CAMetalLayer is provided in the Vulkan headers:

```
// Provided by VK_EXT_metal_surface
#ifdef __OBJC__
@class CAMetalLayer;
#else
typedef void CAMetalLayer;
#endif
```

The actual CAMetalLayer type is defined in the QuartzCore framework.

```
// Provided by VK_EXT_metal_surface
typedef VkFlags VkMetalSurfaceCreateFlagsEXT;
```

`VkMetalSurfaceCreateFlagsEXT` is a bitmask type for setting a mask, but is currently reserved for future use.

### 34.2.13. QNX Screen Platform

To create a `VkSurfaceKHR` object for a QNX Screen surface, call:

```
// Provided by VK_QNX_screen_surface
VkResult vkCreateScreenSurfaceQNX(
    VkInstance                                  instance,
    const VkScreenSurfaceCreateInfoQNX*         pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkSurfaceKHR*                               pSurface);
```

- `instance` is the instance to associate the surface with.
- `pCreateInfo` is a pointer to a VkScreenSurfaceCreateInfoQNX structure containing parameters affecting the creation of the surface object.
- `pAllocator` is the allocator used for host memory allocated for the surface object when there is no more specific allocator available (see Memory Allocation).
- `pSurface` is a pointer to a VkSurfaceKHR handle in which the created surface object is returned.

## Valid Usage (Implicit)

- VUID-vkCreateScreenSurfaceQNX-instance-parameter
  `instance` **must** be a valid VkInstance handle
- VUID-vkCreateScreenSurfaceQNX-pCreateInfo-parameter
  `pCreateInfo` **must** be a valid pointer to a valid VkScreenSurfaceCreateInfoQNX structure
- VUID-vkCreateScreenSurfaceQNX-pAllocator-parameter
  If `pAllocator` is not `NULL`, `pAllocator` **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkCreateScreenSurfaceQNX-pSurface-parameter
  `pSurface` **must** be a valid pointer to a VkSurfaceKHR handle

## Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`

The `VkScreenSurfaceCreateInfoQNX` structure is defined as:

```
// Provided by VK_QNX_screen_surface
typedef struct VkScreenSurfaceCreateInfoQNX {
    VkStructureType                 sType;
    const void*                     pNext;
    VkScreenSurfaceCreateFlagsQNX   flags;
    struct _screen_context*         context;
    struct _screen_window*          window;
} VkScreenSurfaceCreateInfoQNX;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is reserved for future use.
- `context` and `window` are QNX Screen `context` and `window` to associate the surface with.

### Valid Usage

- VUID-VkScreenSurfaceCreateInfoQNX-context-04741
  `context` **must** point to a valid QNX Screen `struct` _screen_context
- VUID-VkScreenSurfaceCreateInfoQNX-window-04742
  `window` **must** point to a valid QNX Screen `struct` _screen_window

### Valid Usage (Implicit)

- VUID-VkScreenSurfaceCreateInfoQNX-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_SCREEN_SURFACE_CREATE_INFO_QNX`
- VUID-VkScreenSurfaceCreateInfoQNX-pNext-pNext
  `pNext` **must** be `NULL`
- VUID-VkScreenSurfaceCreateInfoQNX-flags-zerobitmask
  `flags` **must** be `0`

```
// Provided by VK_QNX_screen_surface
typedef VkFlags VkScreenSurfaceCreateFlagsQNX;
```

`VkScreenSurfaceCreateFlagsQNX` is a bitmask type for setting a mask, but is currently reserved for future use.

## 34.2.14. Platform-Independent Information

Once created, `VkSurfaceKHR` objects **can** be used in this and other extensions, in particular the `VK_KHR_swapchain` extension.

Several WSI functions return `VK_ERROR_SURFACE_LOST_KHR` if the surface becomes no longer available.

After such an error, the surface (and any child swapchain, if one exists) **should** be destroyed, as there is no way to restore them to a not-lost state. Applications **may** attempt to create a new VkSurfaceKHR using the same native platform window object, but whether such re-creation will succeed is platform-dependent and **may** depend on the reason the surface became unavailable. A lost surface does not otherwise cause devices to be lost.

To destroy a VkSurfaceKHR object, call:

```
// Provided by VK_KHR_surface
void vkDestroySurfaceKHR(
    VkInstance                                  instance,
    VkSurfaceKHR                                surface,
    const VkAllocationCallbacks*                pAllocator);
```

- instance is the instance used to create the surface.
- surface is the surface to destroy.
- pAllocator is the allocator used for host memory allocated for the surface object when there is no more specific allocator available (see Memory Allocation).

Destroying a VkSurfaceKHR merely severs the connection between Vulkan and the native surface, and does not imply destroying the native surface, closing a window, or similar behavior.

### Valid Usage

- VUID-vkDestroySurfaceKHR-surface-01266
  All VkSwapchainKHR objects created for surface **must** have been destroyed prior to destroying surface
- VUID-vkDestroySurfaceKHR-surface-01267
  If VkAllocationCallbacks were provided when surface was created, a compatible set of callbacks **must** be provided here
- VUID-vkDestroySurfaceKHR-surface-01268
  If no VkAllocationCallbacks were provided when surface was created, pAllocator **must** be NULL

### Valid Usage (Implicit)

- VUID-vkDestroySurfaceKHR-instance-parameter
  instance **must** be a valid VkInstance handle
- VUID-vkDestroySurfaceKHR-surface-parameter
  If surface is not VK_NULL_HANDLE, surface **must** be a valid VkSurfaceKHR handle
- VUID-vkDestroySurfaceKHR-pAllocator-parameter
  If pAllocator is not NULL, pAllocator **must** be a valid pointer to a valid VkAllocationCallbacks structure

- VUID-vkDestroySurfaceKHR-surface-parent
  If `surface` is a valid handle, it **must** have been created, allocated, or retrieved from `instance`

**Host Synchronization**

- Host access to `surface` **must** be externally synchronized

# 34.3. Presenting Directly to Display Devices

In some environments applications **can** also present Vulkan rendering directly to display devices without using an intermediate windowing system. This **can** be useful for embedded applications, or implementing the rendering/presentation backend of a windowing system using Vulkan. The `VK_KHR_display` extension provides the functionality necessary to enumerate display devices and create `VkSurfaceKHR` objects that target displays.

## 34.3.1. Display Enumeration

Displays are represented by `VkDisplayKHR` handles:

```
// Provided by VK_KHR_display
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkDisplayKHR)
```

Various functions are provided for enumerating the available display devices present on a Vulkan physical device. To query information about the available displays, call:

```
// Provided by VK_KHR_display
VkResult vkGetPhysicalDeviceDisplayPropertiesKHR(
    VkPhysicalDevice                            physicalDevice,
    uint32_t*                                   pPropertyCount,
    VkDisplayPropertiesKHR*                     pProperties);
```

- `physicalDevice` is a physical device.
- `pPropertyCount` is a pointer to an integer related to the number of display devices available or queried, as described below.
- `pProperties` is either `NULL` or a pointer to an array of VkDisplayPropertiesKHR structures.

If `pProperties` is `NULL`, then the number of display devices available for `physicalDevice` is returned in `pPropertyCount`. Otherwise, `pPropertyCount` **must** point to a variable set by the application to the number of elements in the `pProperties` array, and on return the variable is overwritten with the number of structures actually written to `pProperties`. If the value of `pPropertyCount` is less than the number of display devices for `physicalDevice`, at most `pPropertyCount` structures will be written, and `VK_INCOMPLETE` will be returned instead of `VK_SUCCESS`, to indicate that not all the available properties were returned.

## Valid Usage (Implicit)

- VUID-vkGetPhysicalDeviceDisplayPropertiesKHR-physicalDevice-parameter
  `physicalDevice` **must** be a valid VkPhysicalDevice handle
- VUID-vkGetPhysicalDeviceDisplayPropertiesKHR-pPropertyCount-parameter
  `pPropertyCount` **must** be a valid pointer to a `uint32_t` value
- VUID-vkGetPhysicalDeviceDisplayPropertiesKHR-pProperties-parameter
  If the value referenced by `pPropertyCount` is not `0`, and `pProperties` is not `NULL`, `pProperties` **must** be a valid pointer to an array of `pPropertyCount` VkDisplayPropertiesKHR structures

## Return Codes

**Success**

- `VK_SUCCESS`
- `VK_INCOMPLETE`

**Failure**

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`

The `VkDisplayPropertiesKHR` structure is defined as:

```
// Provided by VK_KHR_display
typedef struct VkDisplayPropertiesKHR {
    VkDisplayKHR                  display;
    const char*                   displayName;
    VkExtent2D                    physicalDimensions;
    VkExtent2D                    physicalResolution;
    VkSurfaceTransformFlagsKHR    supportedTransforms;
    VkBool32                      planeReorderPossible;
    VkBool32                      persistentContent;
} VkDisplayPropertiesKHR;
```

- `display` is a handle that is used to refer to the display described here. This handle will be valid for the lifetime of the Vulkan instance.
- `displayName` is `NULL` or a pointer to a null-terminated UTF-8 string containing the name of the display. Generally, this will be the name provided by the display's EDID. If `NULL`, no suitable name is available. If not `NULL`, the string pointed to **must** remain accessible and unmodified as long as `display` is valid.
- `physicalDimensions` describes the physical width and height of the visible portion of the display, in millimeters.
- `physicalResolution` describes the physical, native, or preferred resolution of the display.

| | |
|---|---|
| **NOTE** | For devices which have no natural value to return here, implementations **should** return the maximum resolution supported. |

- `supportedTransforms` is a bitmask of VkSurfaceTransformFlagBitsKHR describing which transforms are supported by this display.
- `planeReorderPossible` tells whether the planes on this display **can** have their z order changed. If this is `VK_TRUE`, the application **can** re-arrange the planes on this display in any order relative to each other.
- `persistentContent` tells whether the display supports self-refresh/internal buffering. If this is true, the application **can** submit persistent present operations on swapchains created against this display.

| | |
|---|---|
| **NOTE** | Persistent presents **may** have higher latency, and **may** use less power when the screen content is updated infrequently, or when only a portion of the screen needs to be updated in most frames. |

To query information about the available displays, call:

```
// Provided by VK_KHR_get_display_properties2
VkResult vkGetPhysicalDeviceDisplayProperties2KHR(
    VkPhysicalDevice                            physicalDevice,
    uint32_t*                                   pPropertyCount,
    VkDisplayProperties2KHR*                    pProperties);
```

- `physicalDevice` is a physical device.
- `pPropertyCount` is a pointer to an integer related to the number of display devices available or queried, as described below.
- `pProperties` is either `NULL` or a pointer to an array of `VkDisplayProperties2KHR` structures.

`vkGetPhysicalDeviceDisplayProperties2KHR` behaves similarly to vkGetPhysicalDeviceDisplayPropertiesKHR, with the ability to return extended information via chained output structures.

### Valid Usage (Implicit)

- VUID-vkGetPhysicalDeviceDisplayProperties2KHR-physicalDevice-parameter
  `physicalDevice` **must** be a valid VkPhysicalDevice handle
- VUID-vkGetPhysicalDeviceDisplayProperties2KHR-pPropertyCount-parameter
  `pPropertyCount` **must** be a valid pointer to a `uint32_t` value
- VUID-vkGetPhysicalDeviceDisplayProperties2KHR-pProperties-parameter
  If the value referenced by `pPropertyCount` is not `0`, and `pProperties` is not `NULL`, `pProperties` **must** be a valid pointer to an array of `pPropertyCount` VkDisplayProperties2KHR structures

## Return Codes

**Success**

- VK_SUCCESS
- VK_INCOMPLETE

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY

The VkDisplayProperties2KHR structure is defined as:

```
// Provided by VK_KHR_get_display_properties2
typedef struct VkDisplayProperties2KHR {
    VkStructureType           sType;
    void*                     pNext;
    VkDisplayPropertiesKHR    displayProperties;
} VkDisplayProperties2KHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- displayProperties is a VkDisplayPropertiesKHR structure.

## Valid Usage (Implicit)

- VUID-VkDisplayProperties2KHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_DISPLAY_PROPERTIES_2_KHR
- VUID-VkDisplayProperties2KHR-pNext-pNext
  pNext **must** be NULL

### Acquiring and Releasing Displays

On some platforms, access to displays is limited to a single process or native driver instance. On such platforms, some or all of the displays may not be available to Vulkan if they are already in use by a native windowing system or other application.

To acquire permission to directly access a display in Vulkan from an X11 server, call:

```
// Provided by VK_EXT_acquire_xlib_display
VkResult vkAcquireXlibDisplayEXT(
    VkPhysicalDevice                            physicalDevice,
    Display*                                    dpy,
    VkDisplayKHR                                display);
```

- `physicalDevice` The physical device the display is on.
- `dpy` A connection to the X11 server that currently owns `display`.
- `display` The display the caller wishes to control in Vulkan.

All permissions necessary to control the display are granted to the Vulkan instance associated with `physicalDevice` until the display is released or the X11 connection specified by `dpy` is terminated. Permission to access the display **may** be temporarily revoked during periods when the X11 server from which control was acquired itself loses access to `display`. During such periods, operations which require access to the display **must** fail with an appropriate error code. If the X11 server associated with `dpy` does not own `display`, or if permission to access it has already been acquired by another entity, the call **must** return the error code `VK_ERROR_INITIALIZATION_FAILED`.

| | |
|---|---|
| **NOTE** | One example of when an X11 server loses access to a display is when it loses ownership of its virtual terminal. |

### Valid Usage (Implicit)

- VUID-vkAcquireXlibDisplayEXT-physicalDevice-parameter
  `physicalDevice` **must** be a valid VkPhysicalDevice handle
- VUID-vkAcquireXlibDisplayEXT-dpy-parameter
  `dpy` **must** be a valid pointer to a `Display` value
- VUID-vkAcquireXlibDisplayEXT-display-parameter
  `display` **must** be a valid VkDisplayKHR handle
- VUID-vkAcquireXlibDisplayEXT-display-parent
  `display` **must** have been created, allocated, or retrieved from `physicalDevice`

### Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_INITIALIZATION_FAILED`

When acquiring displays from an X11 server, an application may also wish to enumerate and identify them using a native handle rather than a `VkDisplayKHR` handle. To determine the `VkDisplayKHR` handle corresponding to an X11 RandR Output, call:

```
// Provided by VK_EXT_acquire_xlib_display
VkResult vkGetRandROutputDisplayEXT(
    VkPhysicalDevice                            physicalDevice,
    Display*                                    dpy,
    RROutput                                    rrOutput,
    VkDisplayKHR*                               pDisplay);
```

- `physicalDevice` The physical device to query the display handle on.
- `dpy` A connection to the X11 server from which `rrOutput` was queried.
- `rrOutput` An X11 RandR output ID.
- `pDisplay` The corresponding VkDisplayKHR handle will be returned here.

If there is no VkDisplayKHR corresponding to `rrOutput` on `physicalDevice`, VK_NULL_HANDLE **must** be returned in `pDisplay`.

### Valid Usage (Implicit)

- VUID-vkGetRandROutputDisplayEXT-physicalDevice-parameter
  `physicalDevice` **must** be a valid VkPhysicalDevice handle
- VUID-vkGetRandROutputDisplayEXT-dpy-parameter
  `dpy` **must** be a valid pointer to a `Display` value
- VUID-vkGetRandROutputDisplayEXT-pDisplay-parameter
  `pDisplay` **must** be a valid pointer to a VkDisplayKHR handle

### Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- `VK_ERROR_OUT_OF_HOST_MEMORY`

To acquire permission to directly access a display in Vulkan on Windows 10, call:

```
// Provided by VK_NV_acquire_winrt_display
VkResult vkAcquireWinrtDisplayNV(
    VkPhysicalDevice                            physicalDevice,
    VkDisplayKHR                                display);
```

- `physicalDevice` The physical device the display is on.
- `display` The display the caller wishes to control in Vulkan.

All permissions necessary to control the display are granted to the Vulkan instance associated with

physicalDevice until the display is released or the application is terminated. Permission to access the display **may** be revoked by events that cause Windows 10 itself to lose access to display. If this has happened, operations which require access to the display **must** fail with an appropriate error code. If permission to access display has already been acquired by another entity, the call **must** return the error code VK_ERROR_INITIALIZATION_FAILED.

| | |
|---|---|
| **NOTE** | The Vulkan instance acquires control of a "winrt::Windows::Devices::Display::Core::DisplayTarget" by performing an operation equivalent to "winrt::Windows::Devices::Display::Core::DisplayManager.TryAcquireTarget()" on the "DisplayTarget". |
| **NOTE** | One example of when Windows 10 loses access to a display is when the display is hot-unplugged. |
| **NOTE** | One example of when a display has already been acquired by another entity is when the Windows desktop compositor (DWM) is in control of the display. Beginning with Windows 10 version 2004 it is possible to cause DWM to release a display by using the "Advanced display settings" sub-page of the "Display settings" control panel. vkAcquireWinrtDisplayNV does not itself cause DWM to release a display; this action must be performed outside of Vulkan. |

## Valid Usage (Implicit)

- VUID-vkAcquireWinrtDisplayNV-physicalDevice-parameter
  physicalDevice **must** be a valid VkPhysicalDevice handle
- VUID-vkAcquireWinrtDisplayNV-display-parameter
  display **must** be a valid VkDisplayKHR handle
- VUID-vkAcquireWinrtDisplayNV-display-parent
  display **must** have been created, allocated, or retrieved from physicalDevice

## Return Codes

### Success

- VK_SUCCESS

### Failure

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_DEVICE_LOST
- VK_ERROR_INITIALIZATION_FAILED

When acquiring displays on Windows 10, an application may also wish to enumerate and identify them using a native handle rather than a VkDisplayKHR handle.

To determine the VkDisplayKHR handle corresponding to a "winrt::Windows::Devices::Display::Core::DisplayTarget", call:

```
// Provided by VK_NV_acquire_winrt_display
VkResult vkGetWinrtDisplayNV(
    VkPhysicalDevice                            physicalDevice,
    uint32_t                                    deviceRelativeId,
    VkDisplayKHR*                               pDisplay);
```

- physicalDevice The physical device on which to query the display handle.
- deviceRelativeId The value of the "AdapterRelativeId" property of a "DisplayTarget" that is enumerated by a "DisplayAdapter" with an "Id" property matching the deviceLUID property of a VkPhysicalDeviceIDProperties for physicalDevice.
- pDisplay The corresponding VkDisplayKHR handle will be returned here.

If there is no VkDisplayKHR corresponding to deviceRelativeId on physicalDevice, VK_NULL_HANDLE **must** be returned in pDisplay.

### Valid Usage (Implicit)

- VUID-vkGetWinrtDisplayNV-physicalDevice-parameter
  physicalDevice **must** be a valid VkPhysicalDevice handle
- VUID-vkGetWinrtDisplayNV-pDisplay-parameter
  pDisplay **must** be a valid pointer to a VkDisplayKHR handle

### Return Codes

**Success**

- VK_SUCCESS

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_DEVICE_LOST
- VK_ERROR_INITIALIZATION_FAILED

To acquire permission to directly a display in Vulkan from the Direct Rendering Manager (DRM) interface, call:

```
// Provided by VK_EXT_acquire_drm_display
VkResult vkAcquireDrmDisplayEXT(
    VkPhysicalDevice                            physicalDevice,
    int32_t                                     drmFd,
    VkDisplayKHR                                display);
```

- `physicalDevice` The physical device the display is on.
- `drmFd` DRM primary file descriptor.
- `display` The display the caller wishes Vulkan to control.

All permissions necessary to control the display are granted to the Vulkan instance associated with the provided `physicalDevice` until the display is either released or the connector is unplugged. The provided `drmFd` **must** correspond to the one owned by the `physicalDevice`. If not, the error code `VK_ERROR_UNKNOWN` **must** be returned. The DRM FD must have DRM master permissions. If any error is encountered during the acquisition of the display, the call **must** return the error code `VK_ERROR_INITIALIZATION_FAILED`.

The provided DRM fd should not be closed before the display is released, attempting to do it may result in undefined behavior.

### Valid Usage (Implicit)

- VUID-vkAcquireDrmDisplayEXT-physicalDevice-parameter
  `physicalDevice` **must** be a valid VkPhysicalDevice handle
- VUID-vkAcquireDrmDisplayEXT-display-parameter
  `display` **must** be a valid VkDisplayKHR handle
- VUID-vkAcquireDrmDisplayEXT-display-parent
  `display` **must** have been created, allocated, or retrieved from `physicalDevice`

### Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- `VK_ERROR_INITIALIZATION_FAILED`

Before acquiring a display from the DRM interface, the caller may want to select a specific `VkDisplayKHR` handle by identifying it using a `connectorId`. To do so, call:

```
// Provided by VK_EXT_acquire_drm_display
VkResult vkGetDrmDisplayEXT(
    VkPhysicalDevice                            physicalDevice,
    int32_t                                     drmFd,
    uint32_t                                    connectorId,
    VkDisplayKHR*                               display);
```

- `physicalDevice` The physical device to query the display from.
- `drmFd` DRM primary file descriptor.
- `connectorId` Identifier of the specified DRM connector.

- `display` The corresponding VkDisplayKHR handle will be returned here.

If there is no VkDisplayKHR corresponding to the `connectorId` on the `physicalDevice`, the returning `display` **must** be set to VK_NULL_HANDLE. The provided `drmFd` **must** correspond to the one owned by the `physicalDevice`. If not, the error code `VK_ERROR_UNKNOWN` **must** be returned. Master permissions are not required, because the file descriptor is just used for information gathering purposes. The given `connectorId` **must** be a resource owned by the provided `drmFd`. If not, the error code `VK_ERROR_UNKNOWN` **must** be returned. If any error is encountered during the identification of the display, the call **must** return the error code `VK_ERROR_INITIALIZATION_FAILED`.

### Valid Usage (Implicit)

- VUID-vkGetDrmDisplayEXT-physicalDevice-parameter
  `physicalDevice` **must** be a valid VkPhysicalDevice handle
- VUID-vkGetDrmDisplayEXT-display-parameter
  `display` **must** be a valid pointer to a VkDisplayKHR handle

### Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- `VK_ERROR_INITIALIZATION_FAILED`
- `VK_ERROR_OUT_OF_HOST_MEMORY`

To release a previously acquired display, call:

```
// Provided by VK_EXT_direct_mode_display
VkResult vkReleaseDisplayEXT(
    VkPhysicalDevice                            physicalDevice,
    VkDisplayKHR                                display);
```

- `physicalDevice` The physical device the display is on.
- `display` The display to release control of.

### Valid Usage (Implicit)

- VUID-vkReleaseDisplayEXT-physicalDevice-parameter
  `physicalDevice` **must** be a valid VkPhysicalDevice handle
- VUID-vkReleaseDisplayEXT-display-parameter
  `display` **must** be a valid VkDisplayKHR handle
- VUID-vkReleaseDisplayEXT-display-parent

display **must** have been created, allocated, or retrieved from physicalDevice

## Return Codes

**Success**

- VK_SUCCESS

**Failure**

None

### Display Planes

Images are presented to individual planes on a display. Devices **must** support at least one plane on each display. Planes **can** be stacked and blended to composite multiple images on one display. Devices **may** support only a fixed stacking order and fixed mapping between planes and displays, or they **may** allow arbitrary application-specified stacking orders and mappings between planes and displays. To query the properties of device display planes, call:

```
// Provided by VK_KHR_display
VkResult vkGetPhysicalDeviceDisplayPlanePropertiesKHR(
    VkPhysicalDevice                            physicalDevice,
    uint32_t*                                   pPropertyCount,
    VkDisplayPlanePropertiesKHR*                pProperties);
```

- physicalDevice is a physical device.
- pPropertyCount is a pointer to an integer related to the number of display planes available or queried, as described below.
- pProperties is either NULL or a pointer to an array of VkDisplayPlanePropertiesKHR structures.

If pProperties is NULL, then the number of display planes available for physicalDevice is returned in pPropertyCount. Otherwise, pPropertyCount **must** point to a variable set by the application to the number of elements in the pProperties array, and on return the variable is overwritten with the number of structures actually written to pProperties. If the value of pPropertyCount is less than the number of display planes for physicalDevice, at most pPropertyCount structures will be written.

## Valid Usage (Implicit)

- VUID-vkGetPhysicalDeviceDisplayPlanePropertiesKHR-physicalDevice-parameter
  physicalDevice **must** be a valid VkPhysicalDevice handle
- VUID-vkGetPhysicalDeviceDisplayPlanePropertiesKHR-pPropertyCount-parameter
  pPropertyCount **must** be a valid pointer to a uint32_t value
- VUID-vkGetPhysicalDeviceDisplayPlanePropertiesKHR-pProperties-parameter
  If the value referenced by pPropertyCount is not 0, and pProperties is not NULL, pProperties **must** be a valid pointer to an array of pPropertyCount VkDisplayPlanePropertiesKHR

structures

### Return Codes

**Success**

- VK_SUCCESS
- VK_INCOMPLETE

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY

The VkDisplayPlanePropertiesKHR structure is defined as:

```
// Provided by VK_KHR_display
typedef struct VkDisplayPlanePropertiesKHR {
    VkDisplayKHR    currentDisplay;
    uint32_t        currentStackIndex;
} VkDisplayPlanePropertiesKHR;
```

- currentDisplay is the handle of the display the plane is currently associated with. If the plane is not currently attached to any displays, this will be VK_NULL_HANDLE.
- currentStackIndex is the current z-order of the plane. This will be between 0 and the value returned by vkGetPhysicalDeviceDisplayPlanePropertiesKHR in pPropertyCount.

To query the properties of a device's display planes, call:

```
// Provided by VK_KHR_get_display_properties2
VkResult vkGetPhysicalDeviceDisplayPlaneProperties2KHR(
    VkPhysicalDevice                            physicalDevice,
    uint32_t*                                   pPropertyCount,
    VkDisplayPlaneProperties2KHR*               pProperties);
```

- physicalDevice is a physical device.
- pPropertyCount is a pointer to an integer related to the number of display planes available or queried, as described below.
- pProperties is either NULL or a pointer to an array of VkDisplayPlaneProperties2KHR structures.

vkGetPhysicalDeviceDisplayPlaneProperties2KHR behaves similarly to vkGetPhysicalDeviceDisplayPlanePropertiesKHR, with the ability to return extended information via chained output structures.

### Valid Usage (Implicit)

- VUID-vkGetPhysicalDeviceDisplayPlaneProperties2KHR-physicalDevice-parameter
  `physicalDevice` **must** be a valid VkPhysicalDevice handle
- VUID-vkGetPhysicalDeviceDisplayPlaneProperties2KHR-pPropertyCount-parameter
  `pPropertyCount` **must** be a valid pointer to a `uint32_t` value
- VUID-vkGetPhysicalDeviceDisplayPlaneProperties2KHR-pProperties-parameter
  If the value referenced by `pPropertyCount` is not `0`, and `pProperties` is not `NULL`, `pProperties` **must** be a valid pointer to an array of `pPropertyCount` VkDisplayPlaneProperties2KHR structures

### Return Codes

**Success**

- `VK_SUCCESS`
- `VK_INCOMPLETE`

**Failure**

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`

The `VkDisplayPlaneProperties2KHR` structure is defined as:

```
// Provided by VK_KHR_get_display_properties2
typedef struct VkDisplayPlaneProperties2KHR {
    VkStructureType                 sType;
    void*                           pNext;
    VkDisplayPlanePropertiesKHR     displayPlaneProperties;
} VkDisplayPlaneProperties2KHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `displayPlaneProperties` is a VkDisplayPlanePropertiesKHR structure.

### Valid Usage (Implicit)

- VUID-VkDisplayPlaneProperties2KHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_DISPLAY_PLANE_PROPERTIES_2_KHR`
- VUID-VkDisplayPlaneProperties2KHR-pNext-pNext
  `pNext` **must** be `NULL`

To determine which displays a plane is usable with, call

```
// Provided by VK_KHR_display
VkResult vkGetDisplayPlaneSupportedDisplaysKHR(
    VkPhysicalDevice                            physicalDevice,
    uint32_t                                    planeIndex,
    uint32_t*                                   pDisplayCount,
    VkDisplayKHR*                               pDisplays);
```

- `physicalDevice` is a physical device.
- `planeIndex` is the plane which the application wishes to use, and **must** be in the range [0, physical device plane count - 1].
- `pDisplayCount` is a pointer to an integer related to the number of displays available or queried, as described below.
- `pDisplays` is either `NULL` or a pointer to an array of `VkDisplayKHR` handles.

If `pDisplays` is `NULL`, then the number of displays usable with the specified `planeIndex` for `physicalDevice` is returned in `pDisplayCount`. Otherwise, `pDisplayCount` **must** point to a variable set by the application to the number of elements in the `pDisplays` array, and on return the variable is overwritten with the number of handles actually written to `pDisplays`. If the value of `pDisplayCount` is less than the number of usable display-plane pairs for `physicalDevice`, at most `pDisplayCount` handles will be written, and `VK_INCOMPLETE` will be returned instead of `VK_SUCCESS`, to indicate that not all the available pairs were returned.

## Valid Usage

- VUID-vkGetDisplayPlaneSupportedDisplaysKHR-planeIndex-01249
  `planeIndex` **must** be less than the number of display planes supported by the device as determined by calling `vkGetPhysicalDeviceDisplayPlanePropertiesKHR`

## Valid Usage (Implicit)

- VUID-vkGetDisplayPlaneSupportedDisplaysKHR-physicalDevice-parameter
  `physicalDevice` **must** be a valid VkPhysicalDevice handle
- VUID-vkGetDisplayPlaneSupportedDisplaysKHR-pDisplayCount-parameter
  `pDisplayCount` **must** be a valid pointer to a `uint32_t` value
- VUID-vkGetDisplayPlaneSupportedDisplaysKHR-pDisplays-parameter
  If the value referenced by `pDisplayCount` is not `0`, and `pDisplays` is not `NULL`, `pDisplays` **must** be a valid pointer to an array of `pDisplayCount` VkDisplayKHR handles

## Return Codes

**Success**

- `VK_SUCCESS`

- VK_INCOMPLETE

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY

Additional properties of displays are queried using specialized query functions.

### Display Modes

Display modes are represented by VkDisplayModeKHR handles:

```
// Provided by VK_KHR_display
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkDisplayModeKHR)
```

Each display has one or more supported modes associated with it by default. These built-in modes are queried by calling:

```
// Provided by VK_KHR_display
VkResult vkGetDisplayModePropertiesKHR(
    VkPhysicalDevice                            physicalDevice,
    VkDisplayKHR                                display,
    uint32_t*                                   pPropertyCount,
    VkDisplayModePropertiesKHR*                 pProperties);
```

- physicalDevice is the physical device associated with display.
- display is the display to query.
- pPropertyCount is a pointer to an integer related to the number of display modes available or queried, as described below.
- pProperties is either NULL or a pointer to an array of VkDisplayModePropertiesKHR structures.

If pProperties is NULL, then the number of display modes available on the specified display for physicalDevice is returned in pPropertyCount. Otherwise, pPropertyCount **must** point to a variable set by the application to the number of elements in the pProperties array, and on return the variable is overwritten with the number of structures actually written to pProperties. If the value of pPropertyCount is less than the number of display modes for physicalDevice, at most pPropertyCount structures will be written, and VK_INCOMPLETE will be returned instead of VK_SUCCESS, to indicate that not all the available display modes were returned.

#### Valid Usage (Implicit)

- VUID-vkGetDisplayModePropertiesKHR-physicalDevice-parameter
  physicalDevice **must** be a valid VkPhysicalDevice handle
- VUID-vkGetDisplayModePropertiesKHR-display-parameter
  display **must** be a valid VkDisplayKHR handle

- VUID-vkGetDisplayModePropertiesKHR-pPropertyCount-parameter
  `pPropertyCount` **must** be a valid pointer to a `uint32_t` value
- VUID-vkGetDisplayModePropertiesKHR-pProperties-parameter
  If the value referenced by `pPropertyCount` is not `0`, and `pProperties` is not `NULL`, `pProperties` **must** be a valid pointer to an array of `pPropertyCount` VkDisplayModePropertiesKHR structures
- VUID-vkGetDisplayModePropertiesKHR-display-parent
  `display` **must** have been created, allocated, or retrieved from `physicalDevice`

## Return Codes

**Success**

- `VK_SUCCESS`
- `VK_INCOMPLETE`

**Failure**

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`

The `VkDisplayModePropertiesKHR` structure is defined as:

```
// Provided by VK_KHR_display
typedef struct VkDisplayModePropertiesKHR {
    VkDisplayModeKHR              displayMode;
    VkDisplayModeParametersKHR    parameters;
} VkDisplayModePropertiesKHR;
```

- `displayMode` is a handle to the display mode described in this structure. This handle will be valid for the lifetime of the Vulkan instance.
- `parameters` is a VkDisplayModeParametersKHR structure describing the display parameters associated with `displayMode`.

```
// Provided by VK_KHR_display
typedef VkFlags VkDisplayModeCreateFlagsKHR;
```

`VkDisplayModeCreateFlagsKHR` is a bitmask type for setting a mask, but is currently reserved for future use.

To query the properties of a device's built-in display modes, call:

```
// Provided by VK_KHR_get_display_properties2
VkResult vkGetDisplayModeProperties2KHR(
    VkPhysicalDevice                            physicalDevice,
    VkDisplayKHR                                display,
    uint32_t*                                   pPropertyCount,
    VkDisplayModeProperties2KHR*                pProperties);
```

- `physicalDevice` is the physical device associated with `display`.
- `display` is the display to query.
- `pPropertyCount` is a pointer to an integer related to the number of display modes available or queried, as described below.
- `pProperties` is either `NULL` or a pointer to an array of `VkDisplayModeProperties2KHR` structures.

`vkGetDisplayModeProperties2KHR` behaves similarly to vkGetDisplayModePropertiesKHR, with the ability to return extended information via chained output structures.

## Valid Usage (Implicit)

- VUID-vkGetDisplayModeProperties2KHR-physicalDevice-parameter
  `physicalDevice` **must** be a valid VkPhysicalDevice handle
- VUID-vkGetDisplayModeProperties2KHR-display-parameter
  `display` **must** be a valid VkDisplayKHR handle
- VUID-vkGetDisplayModeProperties2KHR-pPropertyCount-parameter
  `pPropertyCount` **must** be a valid pointer to a `uint32_t` value
- VUID-vkGetDisplayModeProperties2KHR-pProperties-parameter
  If the value referenced by `pPropertyCount` is not `0`, and `pProperties` is not `NULL`, `pProperties` **must** be a valid pointer to an array of `pPropertyCount` VkDisplayModeProperties2KHR structures
- VUID-vkGetDisplayModeProperties2KHR-display-parent
  `display` **must** have been created, allocated, or retrieved from `physicalDevice`

## Return Codes

### Success

- `VK_SUCCESS`
- `VK_INCOMPLETE`

### Failure

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`

The `VkDisplayModeProperties2KHR` structure is defined as:

```
// Provided by VK_KHR_get_display_properties2
typedef struct VkDisplayModeProperties2KHR {
    VkStructureType                 sType;
    void*                           pNext;
    VkDisplayModePropertiesKHR      displayModeProperties;
} VkDisplayModeProperties2KHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `displayModeProperties` is a VkDisplayModePropertiesKHR structure.

### Valid Usage (Implicit)

- VUID-VkDisplayModeProperties2KHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_DISPLAY_MODE_PROPERTIES_2_KHR`
- VUID-VkDisplayModeProperties2KHR-pNext-pNext
  `pNext` **must** be `NULL`

The `VkDisplayModeParametersKHR` structure is defined as:

```
// Provided by VK_KHR_display
typedef struct VkDisplayModeParametersKHR {
    VkExtent2D    visibleRegion;
    uint32_t      refreshRate;
} VkDisplayModeParametersKHR;
```

- `visibleRegion` is the 2D extents of the visible region.
- `refreshRate` is a `uint32_t` that is the number of times the display is refreshed each second multiplied by 1000.

**NOTE** For example, a 60Hz display mode would report a `refreshRate` of 60,000.

### Valid Usage

- VUID-VkDisplayModeParametersKHR-width-01990
  The `width` member of `visibleRegion` **must** be greater than `0`
- VUID-VkDisplayModeParametersKHR-height-01991
  The `height` member of `visibleRegion` **must** be greater than `0`
- VUID-VkDisplayModeParametersKHR-refreshRate-01992
  `refreshRate` **must** be greater than `0`

Additional modes **may** also be created by calling:

```
// Provided by VK_KHR_display
VkResult vkCreateDisplayModeKHR(
    VkPhysicalDevice                            physicalDevice,
    VkDisplayKHR                                display,
    const VkDisplayModeCreateInfoKHR*           pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkDisplayModeKHR*                           pMode);
```

- `physicalDevice` is the physical device associated with `display`.
- `display` is the display to create an additional mode for.
- `pCreateInfo` is a pointer to a VkDisplayModeCreateInfoKHR structure describing the new mode to create.
- `pAllocator` is the allocator used for host memory allocated for the display mode object when there is no more specific allocator available (see Memory Allocation).
- `pMode` is a pointer to a VkDisplayModeKHR handle in which the mode created is returned.

## Valid Usage (Implicit)

- VUID-vkCreateDisplayModeKHR-physicalDevice-parameter
  `physicalDevice` **must** be a valid VkPhysicalDevice handle
- VUID-vkCreateDisplayModeKHR-display-parameter
  `display` **must** be a valid VkDisplayKHR handle
- VUID-vkCreateDisplayModeKHR-pCreateInfo-parameter
  `pCreateInfo` **must** be a valid pointer to a valid VkDisplayModeCreateInfoKHR structure
- VUID-vkCreateDisplayModeKHR-pAllocator-parameter
  If `pAllocator` is not `NULL`, `pAllocator` **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkCreateDisplayModeKHR-pMode-parameter
  `pMode` **must** be a valid pointer to a VkDisplayModeKHR handle
- VUID-vkCreateDisplayModeKHR-display-parent
  `display` **must** have been created, allocated, or retrieved from `physicalDevice`

## Host Synchronization

- Host access to `display` **must** be externally synchronized

## Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY
- VK_ERROR_INITIALIZATION_FAILED

The VkDisplayModeCreateInfoKHR structure is defined as:

```
// Provided by VK_KHR_display
typedef struct VkDisplayModeCreateInfoKHR {
    VkStructureType                sType;
    const void*                    pNext;
    VkDisplayModeCreateFlagsKHR    flags;
    VkDisplayModeParametersKHR     parameters;
} VkDisplayModeCreateInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- flags is reserved for future use, and **must** be zero.
- parameters is a VkDisplayModeParametersKHR structure describing the display parameters to use in creating the new mode. If the parameters are not compatible with the specified display, the implementation **must** return VK_ERROR_INITIALIZATION_FAILED.

### Valid Usage (Implicit)

- VUID-VkDisplayModeCreateInfoKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_DISPLAY_MODE_CREATE_INFO_KHR
- VUID-VkDisplayModeCreateInfoKHR-pNext-pNext
  pNext **must** be NULL
- VUID-VkDisplayModeCreateInfoKHR-flags-zerobitmask
  flags **must** be 0
- VUID-VkDisplayModeCreateInfoKHR-parameters-parameter
  parameters **must** be a valid VkDisplayModeParametersKHR structure

Applications that wish to present directly to a display **must** select which layer, or “plane” of the display they wish to target, and a mode to use with the display. Each display supports at least one plane. The capabilities of a given mode and plane combination are determined by calling:

```
// Provided by VK_KHR_display
VkResult vkGetDisplayPlaneCapabilitiesKHR(
    VkPhysicalDevice                            physicalDevice,
    VkDisplayModeKHR                            mode,
    uint32_t                                    planeIndex,
    VkDisplayPlaneCapabilitiesKHR*              pCapabilities);
```

- `physicalDevice` is the physical device associated with the display specified by `mode`
- `mode` is the display mode the application intends to program when using the specified plane. Note this parameter also implicitly specifies a display.
- `planeIndex` is the plane which the application intends to use with the display, and is less than the number of display planes supported by the device.
- `pCapabilities` is a pointer to a VkDisplayPlaneCapabilitiesKHR structure in which the capabilities are returned.

## Valid Usage (Implicit)

- VUID-vkGetDisplayPlaneCapabilitiesKHR-physicalDevice-parameter
  `physicalDevice` **must** be a valid VkPhysicalDevice handle
- VUID-vkGetDisplayPlaneCapabilitiesKHR-mode-parameter
  `mode` **must** be a valid VkDisplayModeKHR handle
- VUID-vkGetDisplayPlaneCapabilitiesKHR-pCapabilities-parameter
  `pCapabilities` **must** be a valid pointer to a VkDisplayPlaneCapabilitiesKHR structure
- VUID-vkGetDisplayPlaneCapabilitiesKHR-mode-parent
  `mode` **must** have been created, allocated, or retrieved from `physicalDevice`

## Host Synchronization

- Host access to `mode` **must** be externally synchronized

## Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`

The `VkDisplayPlaneCapabilitiesKHR` structure is defined as:

```
// Provided by VK_KHR_display
typedef struct VkDisplayPlaneCapabilitiesKHR {
    VkDisplayPlaneAlphaFlagsKHR    supportedAlpha;
    VkOffset2D                     minSrcPosition;
    VkOffset2D                     maxSrcPosition;
    VkExtent2D                     minSrcExtent;
    VkExtent2D                     maxSrcExtent;
    VkOffset2D                     minDstPosition;
    VkOffset2D                     maxDstPosition;
    VkExtent2D                     minDstExtent;
    VkExtent2D                     maxDstExtent;
} VkDisplayPlaneCapabilitiesKHR;
```

- `supportedAlpha` is a bitmask of VkDisplayPlaneAlphaFlagBitsKHR describing the supported alpha blending modes.
- `minSrcPosition` is the minimum source rectangle offset supported by this plane using the specified mode.
- `maxSrcPosition` is the maximum source rectangle offset supported by this plane using the specified mode. The `x` and `y` components of `maxSrcPosition` **must** each be greater than or equal to the `x` and `y` components of `minSrcPosition`, respectively.
- `minSrcExtent` is the minimum source rectangle size supported by this plane using the specified mode.
- `maxSrcExtent` is the maximum source rectangle size supported by this plane using the specified mode.
- `minDstPosition`, `maxDstPosition`, `minDstExtent`, `maxDstExtent` all have similar semantics to their corresponding `*Src*` equivalents, but apply to the output region within the mode rather than the input region within the source image. Unlike the `*Src*` offsets, `minDstPosition` and `maxDstPosition` **may** contain negative values.

The minimum and maximum position and extent fields describe the implementation limits, if any, as they apply to the specified display mode and plane. Vendors **may** support displaying a subset of a swapchain's presentable images on the specified display plane. This is expressed by returning `minSrcPosition`, `maxSrcPosition`, `minSrcExtent`, and `maxSrcExtent` values that indicate a range of possible positions and sizes which **may** be used to specify the region within the presentable images that source pixels will be read from when creating a swapchain on the specified display mode and plane.

Vendors **may** also support mapping the presentable images' content to a subset or superset of the visible region in the specified display mode. This is expressed by returning `minDstPosition`, `maxDstPosition`, `minDstExtent` and `maxDstExtent` values that indicate a range of possible positions and sizes which **may** be used to describe the region within the display mode that the source pixels will be mapped to.

Other vendors **may** support only a 1-1 mapping between pixels in the presentable images and the display mode. This **may** be indicated by returning (0,0) for `minSrcPosition`, `maxSrcPosition`, `minDstPosition`, and `maxDstPosition`, and (display mode width, display mode height) for

minSrcExtent, maxSrcExtent, minDstExtent, and maxDstExtent.

The value supportedAlpha **must** contain at least one valid VkDisplayPlaneAlphaFlagBitsKHR bit.

These values indicate the limits of the implementation's individual fields. Not all combinations of values within the offset and extent ranges returned in VkDisplayPlaneCapabilitiesKHR are guaranteed to be supported. Presentation requests specifying unsupported combinations **may** fail.

To query the capabilities of a given mode and plane combination, call:

```
// Provided by VK_KHR_get_display_properties2
VkResult vkGetDisplayPlaneCapabilities2KHR(
    VkPhysicalDevice                            physicalDevice,
    const VkDisplayPlaneInfo2KHR*               pDisplayPlaneInfo,
    VkDisplayPlaneCapabilities2KHR*             pCapabilities);
```

- physicalDevice is the physical device associated with pDisplayPlaneInfo.
- pDisplayPlaneInfo is a pointer to a VkDisplayPlaneInfo2KHR structure describing the plane and mode.
- pCapabilities is a pointer to a VkDisplayPlaneCapabilities2KHR structure in which the capabilities are returned.

vkGetDisplayPlaneCapabilities2KHR behaves similarly to vkGetDisplayPlaneCapabilitiesKHR, with the ability to specify extended inputs via chained input structures, and to return extended information via chained output structures.

### Valid Usage (Implicit)

- VUID-vkGetDisplayPlaneCapabilities2KHR-physicalDevice-parameter
  physicalDevice **must** be a valid VkPhysicalDevice handle
- VUID-vkGetDisplayPlaneCapabilities2KHR-pDisplayPlaneInfo-parameter
  pDisplayPlaneInfo **must** be a valid pointer to a valid VkDisplayPlaneInfo2KHR structure
- VUID-vkGetDisplayPlaneCapabilities2KHR-pCapabilities-parameter
  pCapabilities **must** be a valid pointer to a VkDisplayPlaneCapabilities2KHR structure

### Return Codes

**Success**

- VK_SUCCESS

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY

The VkDisplayPlaneInfo2KHR structure is defined as:

```
// Provided by VK_KHR_get_display_properties2
typedef struct VkDisplayPlaneInfo2KHR {
    VkStructureType     sType;
    const void*         pNext;
    VkDisplayModeKHR    mode;
    uint32_t            planeIndex;
} VkDisplayPlaneInfo2KHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- mode is the display mode the application intends to program when using the specified plane.

| NOTE | This parameter also implicitly specifies a display. |
|---|---|

- planeIndex is the plane which the application intends to use with the display.

The members of VkDisplayPlaneInfo2KHR correspond to the arguments to vkGetDisplayPlaneCapabilitiesKHR, with sType and pNext added for extensibility.

### Valid Usage (Implicit)

- VUID-VkDisplayPlaneInfo2KHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_DISPLAY_PLANE_INFO_2_KHR
- VUID-VkDisplayPlaneInfo2KHR-pNext-pNext
  pNext **must** be NULL
- VUID-VkDisplayPlaneInfo2KHR-mode-parameter
  mode **must** be a valid VkDisplayModeKHR handle

### Host Synchronization

- Host access to mode **must** be externally synchronized

The VkDisplayPlaneCapabilities2KHR structure is defined as:

```
// Provided by VK_KHR_get_display_properties2
typedef struct VkDisplayPlaneCapabilities2KHR {
    VkStructureType                 sType;
    void*                           pNext;
    VkDisplayPlaneCapabilitiesKHR   capabilities;
} VkDisplayPlaneCapabilities2KHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `capabilities` is a VkDisplayPlaneCapabilitiesKHR structure.

### Valid Usage (Implicit)

- VUID-VkDisplayPlaneCapabilities2KHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_DISPLAY_PLANE_CAPABILITIES_2_KHR`
- VUID-VkDisplayPlaneCapabilities2KHR-pNext-pNext
  `pNext` **must** be `NULL`

## 34.3.2. Display Control

To set the power state of a display, call:

```
// Provided by VK_EXT_display_control
VkResult vkDisplayPowerControlEXT(
    VkDevice                                    device,
    VkDisplayKHR                                display,
    const VkDisplayPowerInfoEXT*                pDisplayPowerInfo);
```

- `device` is a logical device associated with `display`.
- `display` is the display whose power state is modified.
- `pDisplayPowerInfo` is a pointer to a VkDisplayPowerInfoEXT structure specifying the new power state of `display`.

### Valid Usage (Implicit)

- VUID-vkDisplayPowerControlEXT-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkDisplayPowerControlEXT-display-parameter
  `display` **must** be a valid VkDisplayKHR handle
- VUID-vkDisplayPowerControlEXT-pDisplayPowerInfo-parameter
  `pDisplayPowerInfo` **must** be a valid pointer to a valid VkDisplayPowerInfoEXT structure
- VUID-vkDisplayPowerControlEXT-commonparent
  Both of `device`, and `display` **must** have been created, allocated, or retrieved from the same VkPhysicalDevice

### Return Codes

**Success**

- VK_SUCCESS

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY

The VkDisplayPowerInfoEXT structure is defined as:

```
// Provided by VK_EXT_display_control
typedef struct VkDisplayPowerInfoEXT {
    VkStructureType           sType;
    const void*               pNext;
    VkDisplayPowerStateEXT    powerState;
} VkDisplayPowerInfoEXT;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- powerState is a VkDisplayPowerStateEXT value specifying the new power state of the display.

### Valid Usage (Implicit)

- VUID-VkDisplayPowerInfoEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_DISPLAY_POWER_INFO_EXT
- VUID-VkDisplayPowerInfoEXT-pNext-pNext
  pNext **must** be NULL
- VUID-VkDisplayPowerInfoEXT-powerState-parameter
  powerState **must** be a valid VkDisplayPowerStateEXT value

Possible values of VkDisplayPowerInfoEXT::powerState, specifying the new power state of a display, are:

```
// Provided by VK_EXT_display_control
typedef enum VkDisplayPowerStateEXT {
    VK_DISPLAY_POWER_STATE_OFF_EXT = 0,
    VK_DISPLAY_POWER_STATE_SUSPEND_EXT = 1,
    VK_DISPLAY_POWER_STATE_ON_EXT = 2,
} VkDisplayPowerStateEXT;
```

- VK_DISPLAY_POWER_STATE_OFF_EXT specifies that the display is powered down.
- VK_DISPLAY_POWER_STATE_SUSPEND_EXT specifies that the display is put into a low power mode, from which it **may** be able to transition back to VK_DISPLAY_POWER_STATE_ON_EXT more quickly

than if it were in VK_DISPLAY_POWER_STATE_OFF_EXT. This state **may** be the same as VK_DISPLAY_POWER_STATE_OFF_EXT.

- VK_DISPLAY_POWER_STATE_ON_EXT specifies that the display is powered on.

### 34.3.3. Display Surfaces

A complete display configuration includes a mode, one or more display planes and any parameters describing their behavior, and parameters describing some aspects of the images associated with those planes. Display surfaces describe the configuration of a single plane within a complete display configuration. To create a VkSurfaceKHR object for a display plane, call:

```
// Provided by VK_KHR_display
VkResult vkCreateDisplayPlaneSurfaceKHR(
    VkInstance                                  instance,
    const VkDisplaySurfaceCreateInfoKHR*        pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkSurfaceKHR*                               pSurface);
```

- instance is the instance corresponding to the physical device the targeted display is on.
- pCreateInfo is a pointer to a VkDisplaySurfaceCreateInfoKHR structure specifying which mode, plane, and other parameters to use, as described below.
- pAllocator is the allocator used for host memory allocated for the surface object when there is no more specific allocator available (see Memory Allocation).
- pSurface is a pointer to a VkSurfaceKHR handle in which the created surface is returned.

#### Valid Usage (Implicit)

- VUID-vkCreateDisplayPlaneSurfaceKHR-instance-parameter
  instance **must** be a valid VkInstance handle
- VUID-vkCreateDisplayPlaneSurfaceKHR-pCreateInfo-parameter
  pCreateInfo **must** be a valid pointer to a valid VkDisplaySurfaceCreateInfoKHR structure
- VUID-vkCreateDisplayPlaneSurfaceKHR-pAllocator-parameter
  If pAllocator is not NULL, pAllocator **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkCreateDisplayPlaneSurfaceKHR-pSurface-parameter
  pSurface **must** be a valid pointer to a VkSurfaceKHR handle

#### Return Codes

**Success**

- VK_SUCCESS

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY

The VkDisplaySurfaceCreateInfoKHR structure is defined as:

```
// Provided by VK_KHR_display
typedef struct VkDisplaySurfaceCreateInfoKHR {
    VkStructureType                    sType;
    const void*                        pNext;
    VkDisplaySurfaceCreateFlagsKHR     flags;
    VkDisplayModeKHR                   displayMode;
    uint32_t                           planeIndex;
    uint32_t                           planeStackIndex;
    VkSurfaceTransformFlagBitsKHR      transform;
    float                              globalAlpha;
    VkDisplayPlaneAlphaFlagBitsKHR     alphaMode;
    VkExtent2D                         imageExtent;
} VkDisplaySurfaceCreateInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- flags is reserved for future use, and **must** be zero.
- displayMode is a VkDisplayModeKHR handle specifying the mode to use when displaying this surface.
- planeIndex is the plane on which this surface appears.
- planeStackIndex is the z-order of the plane.
- transform is a VkSurfaceTransformFlagBitsKHR value specifying the transformation to apply to images as part of the scanout operation.
- globalAlpha is the global alpha value. This value is ignored if alphaMode is not VK_DISPLAY_PLANE_ALPHA_GLOBAL_BIT_KHR.
- alphaMode is a VkDisplayPlaneAlphaFlagBitsKHR value specifying the type of alpha blending to use.
- imageExtent is the size of the presentable images to use with the surface.

**NOTE** Creating a display surface **must** not modify the state of the displays, planes, or other resources it names. For example, it **must** not apply the specified mode to be set on the associated display. Application of display configuration occurs as a side effect of presenting to a display surface.

## Valid Usage

- VUID-VkDisplaySurfaceCreateInfoKHR-planeIndex-01252
  `planeIndex` **must** be less than the number of display planes supported by the device as determined by calling `vkGetPhysicalDeviceDisplayPlanePropertiesKHR`
- VUID-VkDisplaySurfaceCreateInfoKHR-planeReorderPossible-01253
  If the `planeReorderPossible` member of the `VkDisplayPropertiesKHR` structure returned by `vkGetPhysicalDeviceDisplayPropertiesKHR` for the display corresponding to `displayMode` is `VK_TRUE` then `planeStackIndex` **must** be less than the number of display planes supported by the device as determined by calling `vkGetPhysicalDeviceDisplayPlanePropertiesKHR`; otherwise `planeStackIndex` **must** equal the `currentStackIndex` member of `VkDisplayPlanePropertiesKHR` returned by `vkGetPhysicalDeviceDisplayPlanePropertiesKHR` for the display plane corresponding to `displayMode`
- VUID-VkDisplaySurfaceCreateInfoKHR-alphaMode-01254
  If `alphaMode` is `VK_DISPLAY_PLANE_ALPHA_GLOBAL_BIT_KHR` then `globalAlpha` **must** be between `0` and `1`, inclusive
- VUID-VkDisplaySurfaceCreateInfoKHR-alphaMode-01255
  `alphaMode` **must** be one of the bits present in the `supportedAlpha` member of `VkDisplayPlaneCapabilitiesKHR` for the display plane corresponding to `displayMode`
- VUID-VkDisplaySurfaceCreateInfoKHR-transform-06740
  `transform` **must** be one of the bits present in the `supportedTransforms` member of `VkDisplayPropertiesKHR` for the display corresponding to `displayMode`
- VUID-VkDisplaySurfaceCreateInfoKHR-width-01256
  The `width` and `height` members of `imageExtent` **must** be less than or equal to VkPhysicalDeviceLimits::`maxImageDimension2D`

## Valid Usage (Implicit)

- VUID-VkDisplaySurfaceCreateInfoKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_DISPLAY_SURFACE_CREATE_INFO_KHR`
- VUID-VkDisplaySurfaceCreateInfoKHR-pNext-pNext
  `pNext` **must** be `NULL`
- VUID-VkDisplaySurfaceCreateInfoKHR-flags-zerobitmask
  `flags` **must** be `0`
- VUID-VkDisplaySurfaceCreateInfoKHR-displayMode-parameter
  `displayMode` **must** be a valid VkDisplayModeKHR handle
- VUID-VkDisplaySurfaceCreateInfoKHR-transform-parameter
  `transform` **must** be a valid VkSurfaceTransformFlagBitsKHR value
- VUID-VkDisplaySurfaceCreateInfoKHR-alphaMode-parameter
  `alphaMode` **must** be a valid VkDisplayPlaneAlphaFlagBitsKHR value

```
// Provided by VK_KHR_display
typedef VkFlags VkDisplaySurfaceCreateFlagsKHR;
```

VkDisplaySurfaceCreateFlagsKHR is a bitmask type for setting a mask, but is currently reserved for future use.

Bits which **can** be set in VkDisplaySurfaceCreateInfoKHR::alphaMode, specifying the type of alpha blending to use on a display, are:

```
// Provided by VK_KHR_display
typedef enum VkDisplayPlaneAlphaFlagBitsKHR {
    VK_DISPLAY_PLANE_ALPHA_OPAQUE_BIT_KHR = 0x00000001,
    VK_DISPLAY_PLANE_ALPHA_GLOBAL_BIT_KHR = 0x00000002,
    VK_DISPLAY_PLANE_ALPHA_PER_PIXEL_BIT_KHR = 0x00000004,
    VK_DISPLAY_PLANE_ALPHA_PER_PIXEL_PREMULTIPLIED_BIT_KHR = 0x00000008,
} VkDisplayPlaneAlphaFlagBitsKHR;
```

- VK_DISPLAY_PLANE_ALPHA_OPAQUE_BIT_KHR specifies that the source image will be treated as opaque.
- VK_DISPLAY_PLANE_ALPHA_GLOBAL_BIT_KHR specifies that a global alpha value **must** be specified that will be applied to all pixels in the source image.
- VK_DISPLAY_PLANE_ALPHA_PER_PIXEL_BIT_KHR specifies that the alpha value will be determined by the alpha component of the source image's pixels. If the source format contains no alpha values, no blending will be applied. The source alpha values are not premultiplied into the source image's other color components.
- VK_DISPLAY_PLANE_ALPHA_PER_PIXEL_PREMULTIPLIED_BIT_KHR is equivalent to VK_DISPLAY_PLANE_ALPHA_PER_PIXEL_BIT_KHR, except the source alpha values are assumed to be premultiplied into the source image's other color components.

```
// Provided by VK_KHR_display
typedef VkFlags VkDisplayPlaneAlphaFlagsKHR;
```

VkDisplayPlaneAlphaFlagsKHR is a bitmask type for setting a mask of zero or more VkDisplayPlaneAlphaFlagBitsKHR.

### 34.3.4. Presenting to Headless Surfaces

Vulkan rendering can be presented to a headless surface, where the presentation operation is a no-op producing no externally-visible result.

| | |
|---|---|
| **NOTE** | Because there is no real presentation target, the headless presentation engine may be extended to impose an arbitrary or customizable set of restrictions and features. This makes it a useful portable test target for applications targeting a wide range of presentation engines where the actual target presentation engines might be scarce, unavailable or otherwise undesirable or inconvenient to use for general Vulkan |

> application development.
>
> The usual surface query mechanisms must be used to determine the actual restrictions and features of the implementation.

To create a headless VkSurfaceKHR object, call:

```
// Provided by VK_EXT_headless_surface
VkResult vkCreateHeadlessSurfaceEXT(
    VkInstance                                  instance,
    const VkHeadlessSurfaceCreateInfoEXT*       pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkSurfaceKHR*                               pSurface);
```

- instance is the instance to associate the surface with.
- pCreateInfo is a pointer to a VkHeadlessSurfaceCreateInfoEXT structure containing parameters affecting the creation of the surface object.
- pAllocator is the allocator used for host memory allocated for the surface object when there is no more specific allocator available (see Memory Allocation).
- pSurface is a pointer to a VkSurfaceKHR handle in which the created surface object is returned.

### Valid Usage (Implicit)

- VUID-vkCreateHeadlessSurfaceEXT-instance-parameter
  instance **must** be a valid VkInstance handle
- VUID-vkCreateHeadlessSurfaceEXT-pCreateInfo-parameter
  pCreateInfo **must** be a valid pointer to a valid VkHeadlessSurfaceCreateInfoEXT structure
- VUID-vkCreateHeadlessSurfaceEXT-pAllocator-parameter
  If pAllocator is not NULL, pAllocator **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkCreateHeadlessSurfaceEXT-pSurface-parameter
  pSurface **must** be a valid pointer to a VkSurfaceKHR handle

### Return Codes

**Success**

- VK_SUCCESS

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY

The VkHeadlessSurfaceCreateInfoEXT structure is defined as:

```
// Provided by VK_EXT_headless_surface
typedef struct VkHeadlessSurfaceCreateInfoEXT {
    VkStructureType                    sType;
    const void*                        pNext;
    VkHeadlessSurfaceCreateFlagsEXT    flags;
} VkHeadlessSurfaceCreateInfoEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is reserved for future use.

**Valid Usage (Implicit)**

- VUID-VkHeadlessSurfaceCreateInfoEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_HEADLESS_SURFACE_CREATE_INFO_EXT`
- VUID-VkHeadlessSurfaceCreateInfoEXT-pNext-pNext
  `pNext` **must** be `NULL`
- VUID-VkHeadlessSurfaceCreateInfoEXT-flags-zerobitmask
  `flags` **must** be `0`

For headless surfaces, `currentExtent` is the reserved value (0xFFFFFFFF, 0xFFFFFFFF). Whatever the application sets a swapchain's `imageExtent` to will be the size of the surface, after the first image is presented.

```
// Provided by VK_EXT_headless_surface
typedef VkFlags VkHeadlessSurfaceCreateFlagsEXT;
```

`VkHeadlessSurfaceCreateFlagsEXT` is a bitmask type for setting a mask, but is currently reserved for future use.

## 34.4. Querying for WSI Support

Not all physical devices will include WSI support. Within a physical device, not all queue families will support presentation. WSI support and compatibility **can** be determined in a platform-neutral manner (which determines support for presentation to a particular surface object) and additionally **may** be determined in platform-specific manners (which determine support for presentation on the specified physical device but do not guarantee support for presentation to a particular surface object).

To determine whether a queue family of a physical device supports presentation to a given surface, call:

```
// Provided by VK_KHR_surface
VkResult vkGetPhysicalDeviceSurfaceSupportKHR(
    VkPhysicalDevice                            physicalDevice,
    uint32_t                                    queueFamilyIndex,
    VkSurfaceKHR                                surface,
    VkBool32*                                   pSupported);
```

- `physicalDevice` is the physical device.
- `queueFamilyIndex` is the queue family.
- `surface` is the surface.
- `pSupported` is a pointer to a VkBool32, which is set to `VK_TRUE` to indicate support, and `VK_FALSE` otherwise.

## Valid Usage

- VUID-vkGetPhysicalDeviceSurfaceSupportKHR-queueFamilyIndex-01269
  `queueFamilyIndex` **must** be less than `pQueueFamilyPropertyCount` returned by `vkGetPhysicalDeviceQueueFamilyProperties` for the given `physicalDevice`

## Valid Usage (Implicit)

- VUID-vkGetPhysicalDeviceSurfaceSupportKHR-physicalDevice-parameter
  `physicalDevice` **must** be a valid VkPhysicalDevice handle
- VUID-vkGetPhysicalDeviceSurfaceSupportKHR-surface-parameter
  `surface` **must** be a valid VkSurfaceKHR handle
- VUID-vkGetPhysicalDeviceSurfaceSupportKHR-pSupported-parameter
  `pSupported` **must** be a valid pointer to a VkBool32 value
- VUID-vkGetPhysicalDeviceSurfaceSupportKHR-commonparent
  Both of `physicalDevice`, and `surface` **must** have been created, allocated, or retrieved from the same VkInstance

## Return Codes

### Success

- `VK_SUCCESS`

### Failure

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_SURFACE_LOST_KHR`

### 34.4.1. Android Platform

On Android, all physical devices and queue families **must** be capable of presentation with any native window. As a result there is no Android-specific query for these capabilities.

### 34.4.2. Wayland Platform

To determine whether a queue family of a physical device supports presentation to a Wayland compositor, call:

```
// Provided by VK_KHR_wayland_surface
VkBool32 vkGetPhysicalDeviceWaylandPresentationSupportKHR(
    VkPhysicalDevice                            physicalDevice,
    uint32_t                                    queueFamilyIndex,
    struct wl_display*                          display);
```

- `physicalDevice` is the physical device.
- `queueFamilyIndex` is the queue family index.
- `display` is a pointer to the `wl_display` associated with a Wayland compositor.

This platform-specific function **can** be called prior to creating a surface.

#### Valid Usage

- VUID-vkGetPhysicalDeviceWaylandPresentationSupportKHR-queueFamilyIndex-01306 `queueFamilyIndex` **must** be less than `pQueueFamilyPropertyCount` returned by `vkGetPhysicalDeviceQueueFamilyProperties` for the given `physicalDevice`

#### Valid Usage (Implicit)

- VUID-vkGetPhysicalDeviceWaylandPresentationSupportKHR-physicalDevice-parameter `physicalDevice` **must** be a valid VkPhysicalDevice handle
- VUID-vkGetPhysicalDeviceWaylandPresentationSupportKHR-display-parameter `display` **must** be a valid pointer to a `wl_display` value

### 34.4.3. Win32 Platform

To determine whether a queue family of a physical device supports presentation to the Microsoft Windows desktop, call:

```
// Provided by VK_KHR_win32_surface
VkBool32 vkGetPhysicalDeviceWin32PresentationSupportKHR(
    VkPhysicalDevice                            physicalDevice,
    uint32_t                                    queueFamilyIndex);
```

- `physicalDevice` is the physical device.
- `queueFamilyIndex` is the queue family index.

This platform-specific function **can** be called prior to creating a surface.

### Valid Usage

- VUID-vkGetPhysicalDeviceWin32PresentationSupportKHR-queueFamilyIndex-01309 `queueFamilyIndex` **must** be less than `pQueueFamilyPropertyCount` returned by `vkGetPhysicalDeviceQueueFamilyProperties` for the given `physicalDevice`

### Valid Usage (Implicit)

- VUID-vkGetPhysicalDeviceWin32PresentationSupportKHR-physicalDevice-parameter `physicalDevice` **must** be a valid VkPhysicalDevice handle

## 34.4.4. XCB Platform

To determine whether a queue family of a physical device supports presentation to an X11 server, using the XCB client-side library, call:

```
// Provided by VK_KHR_xcb_surface
VkBool32 vkGetPhysicalDeviceXcbPresentationSupportKHR(
    VkPhysicalDevice                            physicalDevice,
    uint32_t                                    queueFamilyIndex,
    xcb_connection_t*                           connection,
    xcb_visualid_t                              visual_id);
```

- `physicalDevice` is the physical device.
- `queueFamilyIndex` is the queue family index.
- `connection` is a pointer to an `xcb_connection_t` to the X server.
- `visual_id` is an X11 visual (`xcb_visualid_t`).

This platform-specific function **can** be called prior to creating a surface.

### Valid Usage

- VUID-vkGetPhysicalDeviceXcbPresentationSupportKHR-queueFamilyIndex-01312

queueFamilyIndex **must** be less than pQueueFamilyPropertyCount returned by vkGetPhysicalDeviceQueueFamilyProperties for the given physicalDevice

### Valid Usage (Implicit)

- VUID-vkGetPhysicalDeviceXcbPresentationSupportKHR-physicalDevice-parameter physicalDevice **must** be a valid VkPhysicalDevice handle
- VUID-vkGetPhysicalDeviceXcbPresentationSupportKHR-connection-parameter connection **must** be a valid pointer to an xcb_connection_t value

## 34.4.5. Xlib Platform

To determine whether a queue family of a physical device supports presentation to an X11 server, using the Xlib client-side library, call:

```
// Provided by VK_KHR_xlib_surface
VkBool32 vkGetPhysicalDeviceXlibPresentationSupportKHR(
    VkPhysicalDevice                            physicalDevice,
    uint32_t                                    queueFamilyIndex,
    Display*                                    dpy,
    VisualID                                    visualID);
```

- physicalDevice is the physical device.
- queueFamilyIndex is the queue family index.
- dpy is a pointer to an Xlib Display connection to the server.
- visualId is an X11 visual (VisualID).

This platform-specific function **can** be called prior to creating a surface.

### Valid Usage

- VUID-vkGetPhysicalDeviceXlibPresentationSupportKHR-queueFamilyIndex-01315 queueFamilyIndex **must** be less than pQueueFamilyPropertyCount returned by vkGetPhysicalDeviceQueueFamilyProperties for the given physicalDevice

### Valid Usage (Implicit)

- VUID-vkGetPhysicalDeviceXlibPresentationSupportKHR-physicalDevice-parameter physicalDevice **must** be a valid VkPhysicalDevice handle
- VUID-vkGetPhysicalDeviceXlibPresentationSupportKHR-dpy-parameter dpy **must** be a valid pointer to a Display value

### 34.4.6. DirectFB Platform

To determine whether a queue family of a physical device supports presentation with DirectFB library, call:

```
// Provided by VK_EXT_directfb_surface
VkBool32 vkGetPhysicalDeviceDirectFBPresentationSupportEXT(
    VkPhysicalDevice                            physicalDevice,
    uint32_t                                    queueFamilyIndex,
    IDirectFB*                                  dfb);
```

- `physicalDevice` is the physical device.
- `queueFamilyIndex` is the queue family index.
- `dfb` is a pointer to the `IDirectFB` main interface of DirectFB.

This platform-specific function **can** be called prior to creating a surface.

#### Valid Usage

- VUID-vkGetPhysicalDeviceDirectFBPresentationSupportEXT-queueFamilyIndex-04119 `queueFamilyIndex` **must** be less than `pQueueFamilyPropertyCount` returned by `vkGetPhysicalDeviceQueueFamilyProperties` for the given `physicalDevice`

#### Valid Usage (Implicit)

- VUID-vkGetPhysicalDeviceDirectFBPresentationSupportEXT-physicalDevice-parameter `physicalDevice` **must** be a valid VkPhysicalDevice handle
- VUID-vkGetPhysicalDeviceDirectFBPresentationSupportEXT-dfb-parameter `dfb` **must** be a valid pointer to an `IDirectFB` value

### 34.4.7. Fuchsia Platform

On Fuchsia, all physical devices and queue families **must** be capable of presentation with any ImagePipe. As a result there is no Fuchsia-specific query for these capabilities.

### 34.4.8. Google Games Platform

On Google Games Platform, all physical devices and queue families with the `VK_QUEUE_GRAPHICS_BIT` or `VK_QUEUE_COMPUTE_BIT` capabilities **must** be capable of presentation with any Google Games Platform stream descriptor. As a result, there is no query specific to Google Games Platform for these capabilities.

### 34.4.9. iOS Platform

On iOS, all physical devices and queue families **must** be capable of presentation with any layer. As a result there is no iOS-specific query for these capabilities.

### 34.4.10. macOS Platform

On macOS, all physical devices and queue families **must** be capable of presentation with any layer. As a result there is no macOS-specific query for these capabilities.

### 34.4.11. VI Platform

On VI, all physical devices and queue families **must** be capable of presentation with any layer. As a result there is no VI-specific query for these capabilities.

### 34.4.12. QNX Screen Platform

To determine whether a queue family of a physical device supports presentation to a QNX Screen compositor, call:

```
// Provided by VK_QNX_screen_surface
VkBool32 vkGetPhysicalDeviceScreenPresentationSupportQNX(
    VkPhysicalDevice                            physicalDevice,
    uint32_t                                    queueFamilyIndex,
    struct _screen_window*                      window);
```

- `physicalDevice` is the physical device.
- `queueFamilyIndex` is the queue family index.
- `window` is the QNX Screen `window` object.

This platform-specific function **can** be called prior to creating a surface.

#### Valid Usage

- VUID-vkGetPhysicalDeviceScreenPresentationSupportQNX-queueFamilyIndex-04743 `queueFamilyIndex` **must** be less than `pQueueFamilyPropertyCount` returned by `vkGetPhysicalDeviceQueueFamilyProperties` for the given `physicalDevice`

#### Valid Usage (Implicit)

- VUID-vkGetPhysicalDeviceScreenPresentationSupportQNX-physicalDevice-parameter `physicalDevice` **must** be a valid VkPhysicalDevice handle
- VUID-vkGetPhysicalDeviceScreenPresentationSupportQNX-window-parameter `window` **must** be a valid pointer to a `_screen_window` value

## 34.5. Surface Queries

The capabilities of a swapchain targeting a surface are the intersection of the capabilities of the WSI platform, the native window or display, and the physical device. The resulting capabilities **can** be obtained with the queries listed below in this section.

| | |
|---|---|
| **NOTE** | In addition to the surface capabilities as obtained by surface queries below, swapchain images are also subject to ordinary image creation limits as reported by vkGetPhysicalDeviceImageFormatProperties. As an application is instructed by the appropriate Valid Usage sections, both the surface capabilities and the image creation limits have to be satisfied whenever swapchain images are created. |

### 34.5.1. Surface Capabilities

To query the basic capabilities of a surface, needed in order to create a swapchain, call:

```
// Provided by VK_KHR_surface
VkResult vkGetPhysicalDeviceSurfaceCapabilitiesKHR(
    VkPhysicalDevice                            physicalDevice,
    VkSurfaceKHR                                surface,
    VkSurfaceCapabilitiesKHR*                   pSurfaceCapabilities);
```

- `physicalDevice` is the physical device that will be associated with the swapchain to be created, as described for vkCreateSwapchainKHR.
- `surface` is the surface that will be associated with the swapchain.
- `pSurfaceCapabilities` is a pointer to a VkSurfaceCapabilitiesKHR structure in which the capabilities are returned.

#### Valid Usage

- VUID-vkGetPhysicalDeviceSurfaceCapabilitiesKHR-surface-06523
  `surface` **must** be a valid VkSurfaceKHR handle
- VUID-vkGetPhysicalDeviceSurfaceCapabilitiesKHR-surface-06211
  `surface` **must** be supported by `physicalDevice`, as reported by vkGetPhysicalDeviceSurfaceSupportKHR or an equivalent platform-specific mechanism

#### Valid Usage (Implicit)

- VUID-vkGetPhysicalDeviceSurfaceCapabilitiesKHR-physicalDevice-parameter
  `physicalDevice` **must** be a valid VkPhysicalDevice handle
- VUID-vkGetPhysicalDeviceSurfaceCapabilitiesKHR-surface-parameter
  `surface` **must** be a valid VkSurfaceKHR handle
- VUID-vkGetPhysicalDeviceSurfaceCapabilitiesKHR-pSurfaceCapabilities-parameter

pSurfaceCapabilities **must** be a valid pointer to a VkSurfaceCapabilitiesKHR structure

- VUID-vkGetPhysicalDeviceSurfaceCapabilitiesKHR-commonparent
  Both of physicalDevice, and surface **must** have been created, allocated, or retrieved from the same VkInstance

## Return Codes

**Success**

- VK_SUCCESS

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY
- VK_ERROR_SURFACE_LOST_KHR

The VkSurfaceCapabilitiesKHR structure is defined as:

```
// Provided by VK_KHR_surface
typedef struct VkSurfaceCapabilitiesKHR {
    uint32_t                        minImageCount;
    uint32_t                        maxImageCount;
    VkExtent2D                      currentExtent;
    VkExtent2D                      minImageExtent;
    VkExtent2D                      maxImageExtent;
    uint32_t                        maxImageArrayLayers;
    VkSurfaceTransformFlagsKHR      supportedTransforms;
    VkSurfaceTransformFlagBitsKHR   currentTransform;
    VkCompositeAlphaFlagsKHR        supportedCompositeAlpha;
    VkImageUsageFlags               supportedUsageFlags;
} VkSurfaceCapabilitiesKHR;
```

- minImageCount is the minimum number of images the specified device supports for a swapchain created for the surface, and will be at least one.
- maxImageCount is the maximum number of images the specified device supports for a swapchain created for the surface, and will be either 0, or greater than or equal to minImageCount. A value of 0 means that there is no limit on the number of images, though there **may** be limits related to the total amount of memory used by presentable images.
- currentExtent is the current width and height of the surface, or the special value (0xFFFFFFFF, 0xFFFFFFFF) indicating that the surface size will be determined by the extent of a swapchain targeting the surface.
- minImageExtent contains the smallest valid swapchain extent for the surface on the specified device. The width and height of the extent will each be less than or equal to the corresponding width and height of currentExtent, unless currentExtent has the special value described above.

- maxImageExtent contains the largest valid swapchain extent for the surface on the specified device. The width and height of the extent will each be greater than or equal to the corresponding width and height of minImageExtent. The width and height of the extent will each be greater than or equal to the corresponding width and height of currentExtent, unless currentExtent has the special value described above.
- maxImageArrayLayers is the maximum number of layers presentable images **can** have for a swapchain created for this device and surface, and will be at least one.
- supportedTransforms is a bitmask of VkSurfaceTransformFlagBitsKHR indicating the presentation transforms supported for the surface on the specified device. At least one bit will be set.
- currentTransform is VkSurfaceTransformFlagBitsKHR value indicating the surface's current transform relative to the presentation engine's natural orientation.
- supportedCompositeAlpha is a bitmask of VkCompositeAlphaFlagBitsKHR, representing the alpha compositing modes supported by the presentation engine for the surface on the specified device, and at least one bit will be set. Opaque composition **can** be achieved in any alpha compositing mode by either using an image format that has no alpha component, or by ensuring that all pixels in the presentable images have an alpha value of 1.0.
- supportedUsageFlags is a bitmask of VkImageUsageFlagBits representing the ways the application **can** use the presentable images of a swapchain created with VkPresentModeKHR set to VK_PRESENT_MODE_IMMEDIATE_KHR, VK_PRESENT_MODE_MAILBOX_KHR, VK_PRESENT_MODE_FIFO_KHR or VK_PRESENT_MODE_FIFO_RELAXED_KHR for the surface on the specified device. VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT **must** be included in the set. Implementations **may** support additional usages.

| | |
|---|---|
| **NOTE** | Supported usage flags of a presentable image when using VK_PRESENT_MODE_SHARED_DEMAND_REFRESH_KHR or VK_PRESENT_MODE_SHARED_CONTINUOUS_REFRESH_KHR presentation mode are provided by VkSharedPresentSurfaceCapabilitiesKHR::sharedPresentSupportedUsageFlags. |
| **NOTE** | Formulas such as min(N, maxImageCount) are not correct, since maxImageCount **may** be zero. |

To query the basic capabilities of a surface defined by the core or extensions, call:

```
// Provided by VK_KHR_get_surface_capabilities2
VkResult vkGetPhysicalDeviceSurfaceCapabilities2KHR(
    VkPhysicalDevice                            physicalDevice,
    const VkPhysicalDeviceSurfaceInfo2KHR*      pSurfaceInfo,
    VkSurfaceCapabilities2KHR*                  pSurfaceCapabilities);
```

- physicalDevice is the physical device that will be associated with the swapchain to be created, as described for vkCreateSwapchainKHR.
- pSurfaceInfo is a pointer to a VkPhysicalDeviceSurfaceInfo2KHR structure describing the surface and other fixed parameters that would be consumed by vkCreateSwapchainKHR.

- pSurfaceCapabilities is a pointer to a VkSurfaceCapabilities2KHR structure in which the capabilities are returned.

vkGetPhysicalDeviceSurfaceCapabilities2KHR behaves similarly to vkGetPhysicalDeviceSurfaceCapabilitiesKHR, with the ability to specify extended inputs via chained input structures, and to return extended information via chained output structures.

## Valid Usage

- VUID-vkGetPhysicalDeviceSurfaceCapabilities2KHR-pSurfaceInfo-06521
  If the VK_GOOGLE_surfaceless_query extension is not enabled, pSurfaceInfo->surface **must** be a valid VkSurfaceKHR handle
- VUID-vkGetPhysicalDeviceSurfaceCapabilities2KHR-pSurfaceInfo-06522
  If pSurfaceInfo->surface is not VK_NULL_HANDLE, pSurfaceInfo->surface **must** be supported by physicalDevice, as reported by vkGetPhysicalDeviceSurfaceSupportKHR or an equivalent platform-specific mechanism
- VUID-vkGetPhysicalDeviceSurfaceCapabilities2KHR-pNext-02671
  If a VkSurfaceCapabilitiesFullScreenExclusiveEXT structure is included in the pNext chain of pSurfaceCapabilities, a VkSurfaceFullScreenExclusiveWin32InfoEXT structure **must** be included in the pNext chain of pSurfaceInfo
- VUID-vkGetPhysicalDeviceSurfaceCapabilities2KHR-pNext-07776
  If a VkSurfacePresentModeCompatibilityEXT structure is included in the pNext chain of pSurfaceCapabilities, a VkSurfacePresentModeEXT structure **must** be included in the pNext chain of pSurfaceInfo
- VUID-vkGetPhysicalDeviceSurfaceCapabilities2KHR-pNext-07777
  If a VkSurfacePresentScalingCapabilitiesEXT structure is included in the pNext chain of pSurfaceCapabilities, a VkSurfacePresentModeEXT structure **must** be included in the pNext chain of pSurfaceInfo
- VUID-vkGetPhysicalDeviceSurfaceCapabilities2KHR-pNext-07778
  If a VkSurfacePresentModeCompatibilityEXT structure is included in the pNext chain of pSurfaceCapabilities, pSurfaceInfo->surface **must** be a valid VkSurfaceKHR handle
- VUID-vkGetPhysicalDeviceSurfaceCapabilities2KHR-pNext-07779
  If a VkSurfacePresentScalingCapabilitiesEXT structure is included in the pNext chain of pSurfaceCapabilities, pSurfaceInfo->surface **must** be a valid VkSurfaceKHR handle

## Valid Usage (Implicit)

- VUID-vkGetPhysicalDeviceSurfaceCapabilities2KHR-physicalDevice-parameter
  physicalDevice **must** be a valid VkPhysicalDevice handle
- VUID-vkGetPhysicalDeviceSurfaceCapabilities2KHR-pSurfaceInfo-parameter
  pSurfaceInfo **must** be a valid pointer to a valid VkPhysicalDeviceSurfaceInfo2KHR structure
- VUID-vkGetPhysicalDeviceSurfaceCapabilities2KHR-pSurfaceCapabilities-parameter

pSurfaceCapabilities **must** be a valid pointer to a VkSurfaceCapabilities2KHR structure

## Return Codes

**Success**

- VK_SUCCESS

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY
- VK_ERROR_SURFACE_LOST_KHR

The VkPhysicalDeviceSurfaceInfo2KHR structure is defined as:

```
// Provided by VK_KHR_get_surface_capabilities2
typedef struct VkPhysicalDeviceSurfaceInfo2KHR {
    VkStructureType    sType;
    const void*        pNext;
    VkSurfaceKHR       surface;
} VkPhysicalDeviceSurfaceInfo2KHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- surface is the surface that will be associated with the swapchain.

The members of VkPhysicalDeviceSurfaceInfo2KHR correspond to the arguments to vkGetPhysicalDeviceSurfaceCapabilitiesKHR, with sType and pNext added for extensibility.

Additional capabilities of a surface **may** be available to swapchains created with different full-screen exclusive settings - particularly if exclusive full-screen access is application controlled. These additional capabilities **can** be queried by adding a VkSurfaceFullScreenExclusiveInfoEXT structure to the pNext chain of this structure when used to query surface properties. Additionally, for Win32 surfaces with application controlled exclusive full-screen access, chaining a VkSurfaceFullScreenExclusiveWin32InfoEXT structure **may** also report additional surface capabilities. These additional capabilities only apply to swapchains created with the same parameters included in the pNext chain of VkSwapchainCreateInfoKHR.

## Valid Usage

- VUID-VkPhysicalDeviceSurfaceInfo2KHR-pNext-02672
  If the pNext chain includes a VkSurfaceFullScreenExclusiveInfoEXT structure with its fullScreenExclusive member set to VK_FULL_SCREEN_EXCLUSIVE_APPLICATION_CONTROLLED_EXT, and surface was created using vkCreateWin32SurfaceKHR, a VkSurfaceFullScreenExclusiveWin32InfoEXT structure **must** be included in the pNext

chain

- VUID-VkPhysicalDeviceSurfaceInfo2KHR-surface-07919
  If surface is not VK_NULL_HANDLE, and the VK_GOOGLE_surfaceless_query extension is not enabled, surface **must** be a valid VkSurfaceKHR handle

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceSurfaceInfo2KHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SURFACE_INFO_2_KHR
- VUID-VkPhysicalDeviceSurfaceInfo2KHR-pNext-pNext
  Each pNext member of any structure (including this one) in the pNext chain **must** be either NULL or a pointer to a valid instance of VkSurfaceFullScreenExclusiveInfoEXT, VkSurfaceFullScreenExclusiveWin32InfoEXT, or VkSurfacePresentModeEXT
- VUID-VkPhysicalDeviceSurfaceInfo2KHR-sType-unique
  The sType value of each struct in the pNext chain **must** be unique

If the pNext chain of VkSwapchainCreateInfoKHR includes a VkSurfaceFullScreenExclusiveInfoEXT structure, then that structure specifies the application's preferred full-screen transition behavior.

The VkSurfaceFullScreenExclusiveInfoEXT structure is defined as:

```
// Provided by VK_EXT_full_screen_exclusive
typedef struct VkSurfaceFullScreenExclusiveInfoEXT {
    VkStructureType             sType;
    void*                       pNext;
    VkFullScreenExclusiveEXT    fullScreenExclusive;
} VkSurfaceFullScreenExclusiveInfoEXT;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- fullScreenExclusive is a VkFullScreenExclusiveEXT value specifying the preferred full-screen transition behavior.

If this structure is not present, fullScreenExclusive is considered to be VK_FULL_SCREEN_EXCLUSIVE_DEFAULT_EXT.

### Valid Usage (Implicit)

- VUID-VkSurfaceFullScreenExclusiveInfoEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_SURFACE_FULL_SCREEN_EXCLUSIVE_INFO_EXT
- VUID-VkSurfaceFullScreenExclusiveInfoEXT-fullScreenExclusive-parameter
  fullScreenExclusive **must** be a valid VkFullScreenExclusiveEXT value

Possible values of VkSurfaceFullScreenExclusiveInfoEXT::fullScreenExclusive are:

```
// Provided by VK_EXT_full_screen_exclusive
typedef enum VkFullScreenExclusiveEXT {
    VK_FULL_SCREEN_EXCLUSIVE_DEFAULT_EXT = 0,
    VK_FULL_SCREEN_EXCLUSIVE_ALLOWED_EXT = 1,
    VK_FULL_SCREEN_EXCLUSIVE_DISALLOWED_EXT = 2,
    VK_FULL_SCREEN_EXCLUSIVE_APPLICATION_CONTROLLED_EXT = 3,
} VkFullScreenExclusiveEXT;
```

- VK_FULL_SCREEN_EXCLUSIVE_DEFAULT_EXT indicates the implementation **should** determine the appropriate full-screen method by whatever means it deems appropriate.
- VK_FULL_SCREEN_EXCLUSIVE_ALLOWED_EXT indicates the implementation **may** use full-screen exclusive mechanisms when available. Such mechanisms **may** result in better performance and/or the availability of different presentation capabilities, but **may** require a more disruptive transition during swapchain initialization, first presentation and/or destruction.
- VK_FULL_SCREEN_EXCLUSIVE_DISALLOWED_EXT indicates the implementation **should** avoid using full-screen mechanisms which rely on disruptive transitions.
- VK_FULL_SCREEN_EXCLUSIVE_APPLICATION_CONTROLLED_EXT indicates the application will manage full-screen exclusive mode by using the vkAcquireFullScreenExclusiveModeEXT and vkReleaseFullScreenExclusiveModeEXT commands.

The VkSurfaceFullScreenExclusiveWin32InfoEXT structure is defined as:

```
// Provided by VK_KHR_win32_surface with VK_EXT_full_screen_exclusive
typedef struct VkSurfaceFullScreenExclusiveWin32InfoEXT {
    VkStructureType    sType;
    const void*        pNext;
    HMONITOR           hmonitor;
} VkSurfaceFullScreenExclusiveWin32InfoEXT;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- hmonitor is the Win32 HMONITOR handle identifying the display to create the surface with.

**NOTE** If hmonitor is invalidated (e.g. the monitor is unplugged) during the lifetime of a swapchain created with this structure, operations on that swapchain will return VK_ERROR_OUT_OF_DATE_KHR.

**NOTE** It is the responsibility of the application to change the display settings of the targeted Win32 display using the appropriate platform APIs. Such changes **may** alter the surface capabilities reported for the created surface.

### Valid Usage

- VUID-VkSurfaceFullScreenExclusiveWin32InfoEXT-hmonitor-02673
  `hmonitor` **must** be a valid `HMONITOR`

### Valid Usage (Implicit)

- VUID-VkSurfaceFullScreenExclusiveWin32InfoEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_SURFACE_FULL_SCREEN_EXCLUSIVE_WIN32_INFO_EXT`

The `VkSurfaceCapabilities2KHR` structure is defined as:

```
// Provided by VK_KHR_get_surface_capabilities2
typedef struct VkSurfaceCapabilities2KHR {
    VkStructureType             sType;
    void*                       pNext;
    VkSurfaceCapabilitiesKHR    surfaceCapabilities;
} VkSurfaceCapabilities2KHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `surfaceCapabilities` is a VkSurfaceCapabilitiesKHR structure describing the capabilities of the specified surface.

If the `VK_GOOGLE_surfaceless_query` extension is enabled and VkPhysicalDeviceSurfaceInfo2KHR::`surface` in the vkGetPhysicalDeviceSurfaceCapabilities2KHR call is VK_NULL_HANDLE, the values returned in `minImageCount`, `maxImageCount`, `currentExtent`, and `currentTransform` will not reflect that of any surface and will instead be as such:

- `minImageCount` and `maxImageCount` will be 0xFFFFFFFF
- `currentExtent` will be (0xFFFFFFFF, 0xFFFFFFFF)
- `currentTransform` will be `VK_SURFACE_TRANSFORM_INHERIT_BIT_KHR`

### Valid Usage (Implicit)

- VUID-VkSurfaceCapabilities2KHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_SURFACE_CAPABILITIES_2_KHR`
- VUID-VkSurfaceCapabilities2KHR-pNext-pNext
  Each `pNext` member of any structure (including this one) in the `pNext` chain **must** be either `NULL` or a pointer to a valid instance of VkDisplayNativeHdrSurfaceCapabilitiesAMD, VkLatencySurfaceCapabilitiesNV, VkSharedPresentSurfaceCapabilitiesKHR, VkSurfaceCapabilitiesFullScreenExclusiveEXT, VkSurfaceCapabilitiesPresentBarrierNV, VkSurfacePresentModeCompatibilityEXT, VkSurfacePresentScalingCapabilitiesEXT, or

VkSurfaceProtectedCapabilitiesKHR

- VUID-VkSurfaceCapabilities2KHR-sType-unique
  The `sType` value of each struct in the `pNext` chain **must** be unique

An application queries if a protected VkSurfaceKHR is displayable on a specific windowing system using `VkSurfaceProtectedCapabilitiesKHR`, which **can** be passed in `pNext` parameter of `VkSurfaceCapabilities2KHR`.

The `VkSurfaceProtectedCapabilitiesKHR` structure is defined as:

```
// Provided by VK_KHR_surface_protected_capabilities
typedef struct VkSurfaceProtectedCapabilitiesKHR {
    VkStructureType    sType;
    const void*        pNext;
    VkBool32           supportsProtected;
} VkSurfaceProtectedCapabilitiesKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `supportsProtected` specifies whether a protected swapchain created from VkPhysicalDeviceSurfaceInfo2KHR::`surface` for a particular windowing system **can** be displayed on screen or not. If `supportsProtected` is `VK_TRUE`, then creation of swapchains with the `VK_SWAPCHAIN_CREATE_PROTECTED_BIT_KHR` flag set **must** be supported for `surface`.

If the `VK_GOOGLE_surfaceless_query` extension is enabled, the value returned in `supportsProtected` will be identical for every valid surface created on this physical device, and so in the vkGetPhysicalDeviceSurfaceCapabilities2KHR call, VkPhysicalDeviceSurfaceInfo2KHR::`surface` **can** be VK_NULL_HANDLE. In that case, the contents of VkSurfaceCapabilities2KHR::`surfaceCapabilities` as well as any other struct chained to it will be undefined.

### Valid Usage (Implicit)

- VUID-VkSurfaceProtectedCapabilitiesKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_SURFACE_PROTECTED_CAPABILITIES_KHR`

The `VkSurfacePresentScalingCapabilitiesEXT` structure is defined as:

```
// Provided by VK_EXT_surface_maintenance1
typedef struct VkSurfacePresentScalingCapabilitiesEXT {
    VkStructureType             sType;
    void*                       pNext;
    VkPresentScalingFlagsEXT    supportedPresentScaling;
    VkPresentGravityFlagsEXT    supportedPresentGravityX;
    VkPresentGravityFlagsEXT    supportedPresentGravityY;
    VkExtent2D                  minScaledImageExtent;
    VkExtent2D                  maxScaledImageExtent;
} VkSurfacePresentScalingCapabilitiesEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `supportedPresentScaling` is a bitmask of VkPresentScalingFlagBitsEXT representing the scaling methods supported by the surface, or `0` if application-defined scaling is not supported.
- `supportedPresentGravityX` is a bitmask of VkPresentGravityFlagBitsEXT representing the X-axis pixel gravity supported by the surface, or `0` if Vulkan-defined pixel gravity is not supported for the X axis.
- `supportedPresentGravityY` is a bitmask of VkPresentGravityFlagBitsEXT representing the Y-axis pixel gravity supported by the surface, or `0` if Vulkan-defined pixel gravity is not supported for the Y axis.
- `minScaledImageExtent` contains the smallest valid swapchain extent for the surface on the specified device when one of the scaling methods specified in `supportedPresentScaling` is used, or the special value (0xFFFFFFFF, 0xFFFFFFFF) indicating that the surface size will be determined by the extent of a swapchain targeting the surface. The `width` and `height` of the extent will each be smaller than or equal to the corresponding `width` and `height` of VkSurfaceCapabilitiesKHR::`minImageExtent`.
- `maxScaledImageExtent` contains the largest valid swapchain extent for the surface on the specified device when one of the scaling methods specified in `supportedPresentScaling` is used, or the special value described above for `minScaledImageExtent`. The `width` and `height` of the extent will each be greater than or equal to the corresponding `width` and `height` of VkSurfaceCapabilitiesKHR::`maxImageExtent`.

Before creating a swapchain whose scaling mode **can** be specified through the use of VkSwapchainPresentScalingCreateInfoEXT, obtain the set of supported scaling modes by including a VkSurfacePresentModeEXT structure in the `pNext` chain of VkPhysicalDeviceSurfaceInfo2KHR when calling vkGetPhysicalDeviceSurfaceCapabilities2KHR. The implementation **must** return the same values in `VkSurfacePresentScalingCapabilitiesEXT` for any of the compatible present modes as obtained through VkSurfacePresentModeCompatibilityEXT.

### Valid Usage (Implicit)

- VUID-VkSurfacePresentScalingCapabilitiesEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_SURFACE_PRESENT_SCALING_CAPABILITIES_EXT`

Bits which **may** be set in VkSurfacePresentScalingCapabilitiesEXT::supportedPresentScaling, specifying scaling modes supported by the surface, are:

```
// Provided by VK_EXT_surface_maintenance1
typedef enum VkPresentScalingFlagBitsEXT {
    VK_PRESENT_SCALING_ONE_TO_ONE_BIT_EXT = 0x00000001,
    VK_PRESENT_SCALING_ASPECT_RATIO_STRETCH_BIT_EXT = 0x00000002,
    VK_PRESENT_SCALING_STRETCH_BIT_EXT = 0x00000004,
} VkPresentScalingFlagBitsEXT;
```

- VK_PRESENT_SCALING_ONE_TO_ONE_BIT_EXT specifies that no scaling occurs, and pixels in the swapchain image are mapped to one and only one pixel in the surface. The mapping between pixels is defined by the chosen presentation gravity.
- VK_PRESENT_SCALING_ASPECT_RATIO_STRETCH_BIT_EXT specifies that the swapchain image will be minified or magnified such that at least one of the resulting width or height is equal to the corresponding surface dimension, and the other resulting dimension is less than or equal to the corresponding surface dimension, with the aspect ratio of the resulting image being identical to that of the original swapchain image.
- VK_PRESENT_SCALING_STRETCH_BIT_EXT specifies that the swapchain image will be minified or magnified such that the resulting image dimensions are equal to those of the surface.

```
// Provided by VK_EXT_surface_maintenance1
typedef VkFlags VkPresentScalingFlagsEXT;
```

VkPresentScalingFlagsEXT is a bitmask type for setting a mask of zero or more VkPresentScalingFlagBitsEXT.

Bits which **may** be set in the VkSurfacePresentScalingCapabilitiesEXT::supportedPresentGravityX or supportedPresentGravityY fields, specifying the gravity of presented pixels supported by the surface, are:

```
// Provided by VK_EXT_surface_maintenance1
typedef enum VkPresentGravityFlagBitsEXT {
    VK_PRESENT_GRAVITY_MIN_BIT_EXT = 0x00000001,
    VK_PRESENT_GRAVITY_MAX_BIT_EXT = 0x00000002,
    VK_PRESENT_GRAVITY_CENTERED_BIT_EXT = 0x00000004,
} VkPresentGravityFlagBitsEXT;
```

- VK_PRESENT_GRAVITY_MIN_BIT_EXT means that the pixels will gravitate towards the top or left side of the surface.
- VK_PRESENT_GRAVITY_MAX_BIT_EXT means that the pixels will gravitate towards the bottom or right side of the surface.
- VK_PRESENT_GRAVITY_CENTERED_BIT_EXT means that the pixels will be centered in the surface.

If the value in VkSurfaceCapabilitiesKHR::currentTransform is not

VK_SURFACE_TRANSFORM_IDENTITY_BIT_KHR, it is implementation-defined whether the gravity configuration applies to the presented image before or after transformation.

```
// Provided by VK_EXT_surface_maintenance1
typedef VkFlags VkPresentGravityFlagsEXT;
```

VkPresentGravityFlagsEXT is a bitmask type for setting a mask of zero or more VkPresentGravityFlagBitsEXT.

The VkSurfacePresentModeEXT structure is defined as:

```
// Provided by VK_EXT_surface_maintenance1
typedef struct VkSurfacePresentModeEXT {
    VkStructureType     sType;
    void*               pNext;
    VkPresentModeKHR    presentMode;
} VkSurfacePresentModeEXT;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- presentMode is the presentation mode the swapchain will use.

If the VkSurfacePresentModeEXT structure is included in the pNext chain of VkPhysicalDeviceSurfaceInfo2KHR, the values returned in VkSurfaceCapabilitiesKHR::minImageCount, VkSurfaceCapabilitiesKHR::maxImageCount, VkSurfacePresentScalingCapabilitiesEXT::minScaledImageExtent, and VkSurfacePresentScalingCapabilitiesEXT::maxScaledImageExtent are valid only for the specified presentMode. If presentMode is VK_PRESENT_MODE_SHARED_DEMAND_REFRESH_KHR or VK_PRESENT_MODE_SHARED_CONTINUOUS_REFRESH_KHR, the per-present mode image counts **must** both be one. The per-present mode image counts **may** be less-than or greater-than the image counts returned when VkSurfacePresentModeEXT is not provided.

**NOTE**

If VkSwapchainPresentModesCreateInfoEXT is provided to swapchain creation, the requirements for forward progress may be less strict. For example, a FIFO swapchain might only require 2 images to guarantee forward progress, but a MAILBOX one might require 4. Without the per-present image counts, such an implementation would have to return 4 in VkSurfaceCapabilitiesKHR::minImageCount, which pessimizes FIFO. Conversely, an implementation may return a low number for minImageCount, but internally bump the image count when application queries vkGetSwapchainImagesKHR, which can surprise applications, and is not discoverable until swapchain creation. Using VkSurfacePresentModeEXT and VkSwapchainPresentModesCreateInfoEXT together effectively removes this problem.

VkSwapchainPresentModesCreateInfoEXT is required for the specification to be backwards compatible with applications that do not know about, or make use of

this feature.

### Valid Usage

- VUID-VkSurfacePresentModeEXT-presentMode-07780
  `presentMode` **must** be a value reported by vkGetPhysicalDeviceSurfacePresentModesKHR for the specified surface

### Valid Usage (Implicit)

- VUID-VkSurfacePresentModeEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_SURFACE_PRESENT_MODE_EXT`
- VUID-VkSurfacePresentModeEXT-presentMode-parameter
  `presentMode` **must** be a valid VkPresentModeKHR value

The `VkSurfacePresentModeCompatibilityEXT` structure is defined as:

```
// Provided by VK_EXT_surface_maintenance1
typedef struct VkSurfacePresentModeCompatibilityEXT {
    VkStructureType      sType;
    void*                pNext;
    uint32_t             presentModeCount;
    VkPresentModeKHR*    pPresentModes;
} VkSurfacePresentModeCompatibilityEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `presentModeCount` is an integer related to the number of present modes available or queried, as described below.
- `pPresentModes` is a pointer to an array of VkPresentModeKHR in which present modes compatible with a given present mode are returned.

If `pPresentModes` is `NULL`, then the number of present modes that are compatible with the one specified in VkSurfacePresentModeEXT is returned in `presentModeCount`. Otherwise, `presentModeCount` **must** point to a variable set by the application to the number of elements in the `pPresentModes` array, and on return the variable is overwritten with the number of values actually written to `pPresentModes`. If the value of `presentModeCount` is less than the number of compatible present modes that are supported, at most `presentModeCount` values will be written to `pPresentModes`. The implementation **must** include the present mode passed to VkSurfacePresentModeEXT in `pPresentModes`, unless `presentModeCount` is zero.

Before creating a swapchain whose present modes **can** be modified through the use of VkSwapchainPresentModesCreateInfoEXT, obtain the set of present modes compatible with a given initial present mode by including a VkSurfacePresentModeEXT structure in the `pNext` chain of

VkPhysicalDeviceSurfaceInfo2KHR when calling vkGetPhysicalDeviceSurfaceCapabilities2KHR.

### Valid Usage (Implicit)

- VUID-VkSurfacePresentModeCompatibilityEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_SURFACE_PRESENT_MODE_COMPATIBILITY_EXT`
- VUID-VkSurfacePresentModeCompatibilityEXT-pPresentModes-parameter
  If `presentModeCount` is not `0`, and `pPresentModes` is not `NULL`, `pPresentModes` **must** be a valid pointer to an array of `presentModeCount` VkPresentModeKHR values

The `VkSharedPresentSurfaceCapabilitiesKHR` structure is defined as:

```
// Provided by VK_KHR_shared_presentable_image
typedef struct VkSharedPresentSurfaceCapabilitiesKHR {
    VkStructureType      sType;
    void*                pNext;
    VkImageUsageFlags    sharedPresentSupportedUsageFlags;
} VkSharedPresentSurfaceCapabilitiesKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `sharedPresentSupportedUsageFlags` is a bitmask of VkImageUsageFlagBits representing the ways the application **can** use the shared presentable image from a swapchain created with VkPresentModeKHR set to `VK_PRESENT_MODE_SHARED_DEMAND_REFRESH_KHR` or `VK_PRESENT_MODE_SHARED_CONTINUOUS_REFRESH_KHR` for the surface on the specified device. `VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT` **must** be included in the set but implementations **may** support additional usages.

### Valid Usage (Implicit)

- VUID-VkSharedPresentSurfaceCapabilitiesKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_SHARED_PRESENT_SURFACE_CAPABILITIES_KHR`

The `VkDisplayNativeHdrSurfaceCapabilitiesAMD` structure is defined as:

```
// Provided by VK_AMD_display_native_hdr
typedef struct VkDisplayNativeHdrSurfaceCapabilitiesAMD {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           localDimmingSupport;
} VkDisplayNativeHdrSurfaceCapabilitiesAMD;
```

- `sType` is a VkStructureType value identifying this structure.

- pNext is NULL or a pointer to a structure extending this structure.
- localDimmingSupport specifies whether the surface supports local dimming. If this is VK_TRUE, VkSwapchainDisplayNativeHdrCreateInfoAMD **can** be used to explicitly enable or disable local dimming for the surface. Local dimming may also be overridden by vkSetLocalDimmingAMD during the lifetime of the swapchain.

### Valid Usage (Implicit)

- VUID-VkDisplayNativeHdrSurfaceCapabilitiesAMD-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_DISPLAY_NATIVE_HDR_SURFACE_CAPABILITIES_AMD

The VkSurfaceCapabilitiesFullScreenExclusiveEXT structure is defined as:

```
// Provided by VK_EXT_full_screen_exclusive
typedef struct VkSurfaceCapabilitiesFullScreenExclusiveEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           fullScreenExclusiveSupported;
} VkSurfaceCapabilitiesFullScreenExclusiveEXT;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- fullScreenExclusiveControlSupported is a boolean describing whether the surface is able to make use of exclusive full-screen access.

This structure **can** be included in the pNext chain of VkSurfaceCapabilities2KHR to determine support for exclusive full-screen access. If fullScreenExclusiveSupported is VK_FALSE, it indicates that exclusive full-screen access is not obtainable for this surface.

Applications **must** not attempt to create swapchains with VK_FULL_SCREEN_EXCLUSIVE_APPLICATION_CONTROLLED_EXT set if fullScreenExclusiveSupported is VK_FALSE.

### Valid Usage (Implicit)

- VUID-VkSurfaceCapabilitiesFullScreenExclusiveEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_SURFACE_CAPABILITIES_FULL_SCREEN_EXCLUSIVE_EXT

The VkSurfaceCapabilitiesPresentBarrierNV structure is defined as:

```
// Provided by VK_NV_present_barrier
typedef struct VkSurfaceCapabilitiesPresentBarrierNV {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           presentBarrierSupported;
} VkSurfaceCapabilitiesPresentBarrierNV;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `presentBarrierSupported` is a boolean describing whether the surface is able to make use of the present barrier feature.

This structure **can** be included in the `pNext` chain of VkSurfaceCapabilities2KHR to determine support for present barrier access. If `presentBarrierSupported` is `VK_FALSE`, it indicates that the present barrier feature is not obtainable for this surface.

### Valid Usage (Implicit)

- VUID-VkSurfaceCapabilitiesPresentBarrierNV-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_SURFACE_CAPABILITIES_PRESENT_BARRIER_NV`

To query the basic capabilities of a surface, needed in order to create a swapchain, call:

```
// Provided by VK_EXT_display_surface_counter
VkResult vkGetPhysicalDeviceSurfaceCapabilities2EXT(
    VkPhysicalDevice                            physicalDevice,
    VkSurfaceKHR                                surface,
    VkSurfaceCapabilities2EXT*                  pSurfaceCapabilities);
```

- `physicalDevice` is the physical device that will be associated with the swapchain to be created, as described for vkCreateSwapchainKHR.
- `surface` is the surface that will be associated with the swapchain.
- `pSurfaceCapabilities` is a pointer to a VkSurfaceCapabilities2EXT structure in which the capabilities are returned.

`vkGetPhysicalDeviceSurfaceCapabilities2EXT` behaves similarly to vkGetPhysicalDeviceSurfaceCapabilitiesKHR, with the ability to return extended information by adding extending structures to the `pNext` chain of its `pSurfaceCapabilities` parameter.

### Valid Usage

- VUID-vkGetPhysicalDeviceSurfaceCapabilities2EXT-surface-06523
  `surface` **must** be a valid VkSurfaceKHR handle
- VUID-vkGetPhysicalDeviceSurfaceCapabilities2EXT-surface-06211

surface **must** be supported by physicalDevice, as reported by vkGetPhysicalDeviceSurfaceSupportKHR or an equivalent platform-specific mechanism

## Valid Usage (Implicit)

- VUID-vkGetPhysicalDeviceSurfaceCapabilities2EXT-physicalDevice-parameter
  physicalDevice **must** be a valid VkPhysicalDevice handle
- VUID-vkGetPhysicalDeviceSurfaceCapabilities2EXT-surface-parameter
  surface **must** be a valid VkSurfaceKHR handle
- VUID-vkGetPhysicalDeviceSurfaceCapabilities2EXT-pSurfaceCapabilities-parameter
  pSurfaceCapabilities **must** be a valid pointer to a VkSurfaceCapabilities2EXT structure
- VUID-vkGetPhysicalDeviceSurfaceCapabilities2EXT-commonparent
  Both of physicalDevice, and surface **must** have been created, allocated, or retrieved from the same VkInstance

## Return Codes

**Success**

- VK_SUCCESS

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY
- VK_ERROR_SURFACE_LOST_KHR

The VkSurfaceCapabilities2EXT structure is defined as:

```
// Provided by VK_EXT_display_surface_counter
typedef struct VkSurfaceCapabilities2EXT {
    VkStructureType                 sType;
    void*                           pNext;
    uint32_t                        minImageCount;
    uint32_t                        maxImageCount;
    VkExtent2D                      currentExtent;
    VkExtent2D                      minImageExtent;
    VkExtent2D                      maxImageExtent;
    uint32_t                        maxImageArrayLayers;
    VkSurfaceTransformFlagsKHR      supportedTransforms;
    VkSurfaceTransformFlagBitsKHR   currentTransform;
    VkCompositeAlphaFlagsKHR        supportedCompositeAlpha;
    VkImageUsageFlags               supportedUsageFlags;
    VkSurfaceCounterFlagsEXT        supportedSurfaceCounters;
} VkSurfaceCapabilities2EXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `minImageCount` is the minimum number of images the specified device supports for a swapchain created for the surface, and will be at least one.
- `maxImageCount` is the maximum number of images the specified device supports for a swapchain created for the surface, and will be either 0, or greater than or equal to `minImageCount`. A value of 0 means that there is no limit on the number of images, though there **may** be limits related to the total amount of memory used by presentable images.
- `currentExtent` is the current width and height of the surface, or the special value (0xFFFFFFFF, 0xFFFFFFFF) indicating that the surface size will be determined by the extent of a swapchain targeting the surface.
- `minImageExtent` contains the smallest valid swapchain extent for the surface on the specified device. The `width` and `height` of the extent will each be less than or equal to the corresponding `width` and `height` of `currentExtent`, unless `currentExtent` has the special value described above.
- `maxImageExtent` contains the largest valid swapchain extent for the surface on the specified device. The `width` and `height` of the extent will each be greater than or equal to the corresponding `width` and `height` of `minImageExtent`. The `width` and `height` of the extent will each be greater than or equal to the corresponding `width` and `height` of `currentExtent`, unless `currentExtent` has the special value described above.
- `maxImageArrayLayers` is the maximum number of layers presentable images **can** have for a swapchain created for this device and surface, and will be at least one.
- `supportedTransforms` is a bitmask of VkSurfaceTransformFlagBitsKHR indicating the presentation transforms supported for the surface on the specified device. At least one bit will be set.
- `currentTransform` is VkSurfaceTransformFlagBitsKHR value indicating the surface's current transform relative to the presentation engine's natural orientation.
- `supportedCompositeAlpha` is a bitmask of VkCompositeAlphaFlagBitsKHR, representing the alpha compositing modes supported by the presentation engine for the surface on the specified device, and at least one bit will be set. Opaque composition **can** be achieved in any alpha compositing mode by either using an image format that has no alpha component, or by ensuring that all pixels in the presentable images have an alpha value of 1.0.
- `supportedUsageFlags` is a bitmask of VkImageUsageFlagBits representing the ways the application **can** use the presentable images of a swapchain created with VkPresentModeKHR set to `VK_PRESENT_MODE_IMMEDIATE_KHR`, `VK_PRESENT_MODE_MAILBOX_KHR`, `VK_PRESENT_MODE_FIFO_KHR` or `VK_PRESENT_MODE_FIFO_RELAXED_KHR` for the surface on the specified device. `VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT` **must** be included in the set. Implementations **may** support additional usages.
- `supportedSurfaceCounters` is a bitmask of VkSurfaceCounterFlagBitsEXT indicating the supported surface counter types.

## Valid Usage

- VUID-VkSurfaceCapabilities2EXT-supportedSurfaceCounters-01246

supportedSurfaceCounters **must** not include VK_SURFACE_COUNTER_VBLANK_BIT_EXT unless the surface queried is a display surface

### Valid Usage (Implicit)

- VUID-VkSurfaceCapabilities2EXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_SURFACE_CAPABILITIES_2_EXT
- VUID-VkSurfaceCapabilities2EXT-pNext-pNext
  pNext **must** be NULL

Bits which **can** be set in VkSurfaceCapabilities2EXT::supportedSurfaceCounters, indicating supported surface counter types, are:

```
// Provided by VK_EXT_display_surface_counter
typedef enum VkSurfaceCounterFlagBitsEXT {
    VK_SURFACE_COUNTER_VBLANK_BIT_EXT = 0x00000001,
  // VK_SURFACE_COUNTER_VBLANK_EXT is a deprecated alias
    VK_SURFACE_COUNTER_VBLANK_EXT = VK_SURFACE_COUNTER_VBLANK_BIT_EXT,
} VkSurfaceCounterFlagBitsEXT;
```

- VK_SURFACE_COUNTER_VBLANK_BIT_EXT specifies a counter incrementing once every time a vertical blanking period occurs on the display associated with the surface.

```
// Provided by VK_EXT_display_surface_counter
typedef VkFlags VkSurfaceCounterFlagsEXT;
```

VkSurfaceCounterFlagsEXT is a bitmask type for setting a mask of zero or more VkSurfaceCounterFlagBitsEXT.

Bits which **may** be set in VkSurfaceCapabilitiesKHR::supportedTransforms indicating the presentation transforms supported for the surface on the specified device, and possible values of VkSurfaceCapabilitiesKHR::currentTransform indicating the surface's current transform relative to the presentation engine's natural orientation, are:

```
// Provided by VK_KHR_surface
typedef enum VkSurfaceTransformFlagBitsKHR {
    VK_SURFACE_TRANSFORM_IDENTITY_BIT_KHR = 0x00000001,
    VK_SURFACE_TRANSFORM_ROTATE_90_BIT_KHR = 0x00000002,
    VK_SURFACE_TRANSFORM_ROTATE_180_BIT_KHR = 0x00000004,
    VK_SURFACE_TRANSFORM_ROTATE_270_BIT_KHR = 0x00000008,
    VK_SURFACE_TRANSFORM_HORIZONTAL_MIRROR_BIT_KHR = 0x00000010,
    VK_SURFACE_TRANSFORM_HORIZONTAL_MIRROR_ROTATE_90_BIT_KHR = 0x00000020,
    VK_SURFACE_TRANSFORM_HORIZONTAL_MIRROR_ROTATE_180_BIT_KHR = 0x00000040,
    VK_SURFACE_TRANSFORM_HORIZONTAL_MIRROR_ROTATE_270_BIT_KHR = 0x00000080,
    VK_SURFACE_TRANSFORM_INHERIT_BIT_KHR = 0x00000100,
} VkSurfaceTransformFlagBitsKHR;
```

- `VK_SURFACE_TRANSFORM_IDENTITY_BIT_KHR` specifies that image content is presented without being transformed.
- `VK_SURFACE_TRANSFORM_ROTATE_90_BIT_KHR` specifies that image content is rotated 90 degrees clockwise.
- `VK_SURFACE_TRANSFORM_ROTATE_180_BIT_KHR` specifies that image content is rotated 180 degrees clockwise.
- `VK_SURFACE_TRANSFORM_ROTATE_270_BIT_KHR` specifies that image content is rotated 270 degrees clockwise.
- `VK_SURFACE_TRANSFORM_HORIZONTAL_MIRROR_BIT_KHR` specifies that image content is mirrored horizontally.
- `VK_SURFACE_TRANSFORM_HORIZONTAL_MIRROR_ROTATE_90_BIT_KHR` specifies that image content is mirrored horizontally, then rotated 90 degrees clockwise.
- `VK_SURFACE_TRANSFORM_HORIZONTAL_MIRROR_ROTATE_180_BIT_KHR` specifies that image content is mirrored horizontally, then rotated 180 degrees clockwise.
- `VK_SURFACE_TRANSFORM_HORIZONTAL_MIRROR_ROTATE_270_BIT_KHR` specifies that image content is mirrored horizontally, then rotated 270 degrees clockwise.
- `VK_SURFACE_TRANSFORM_INHERIT_BIT_KHR` specifies that the presentation transform is not specified, and is instead determined by platform-specific considerations and mechanisms outside Vulkan.

```
// Provided by VK_KHR_display
typedef VkFlags VkSurfaceTransformFlagsKHR;
```

`VkSurfaceTransformFlagsKHR` is a bitmask type for setting a mask of zero or more VkSurfaceTransformFlagBitsKHR.

The `supportedCompositeAlpha` member is of type VkCompositeAlphaFlagBitsKHR, containing the following values:

```
// Provided by VK_KHR_surface
typedef enum VkCompositeAlphaFlagBitsKHR {
    VK_COMPOSITE_ALPHA_OPAQUE_BIT_KHR = 0x00000001,
    VK_COMPOSITE_ALPHA_PRE_MULTIPLIED_BIT_KHR = 0x00000002,
    VK_COMPOSITE_ALPHA_POST_MULTIPLIED_BIT_KHR = 0x00000004,
    VK_COMPOSITE_ALPHA_INHERIT_BIT_KHR = 0x00000008,
} VkCompositeAlphaFlagBitsKHR;
```

These values are described as follows:

- VK_COMPOSITE_ALPHA_OPAQUE_BIT_KHR: The alpha component, if it exists, of the images is ignored in the compositing process. Instead, the image is treated as if it has a constant alpha of 1.0.
- VK_COMPOSITE_ALPHA_PRE_MULTIPLIED_BIT_KHR: The alpha component, if it exists, of the images is respected in the compositing process. The non-alpha components of the image are expected to already be multiplied by the alpha component by the application.
- VK_COMPOSITE_ALPHA_POST_MULTIPLIED_BIT_KHR: The alpha component, if it exists, of the images is respected in the compositing process. The non-alpha components of the image are not expected to already be multiplied by the alpha component by the application; instead, the compositor will multiply the non-alpha components of the image by the alpha component during compositing.
- VK_COMPOSITE_ALPHA_INHERIT_BIT_KHR: The way in which the presentation engine treats the alpha component in the images is unknown to the Vulkan API. Instead, the application is responsible for setting the composite alpha blending mode using native window system commands. If the application does not set the blending mode using native window system commands, then a platform-specific default will be used.

```
// Provided by VK_KHR_surface
typedef VkFlags VkCompositeAlphaFlagsKHR;
```

VkCompositeAlphaFlagsKHR is a bitmask type for setting a mask of zero or more VkCompositeAlphaFlagBitsKHR.

### 34.5.2. Surface Format Support

To query the supported swapchain format-color space pairs for a surface, call:

```
// Provided by VK_KHR_surface
VkResult vkGetPhysicalDeviceSurfaceFormatsKHR(
    VkPhysicalDevice                            physicalDevice,
    VkSurfaceKHR                                surface,
    uint32_t*                                   pSurfaceFormatCount,
    VkSurfaceFormatKHR*                         pSurfaceFormats);
```

- physicalDevice is the physical device that will be associated with the swapchain to be created, as described for vkCreateSwapchainKHR.

- surface is the surface that will be associated with the swapchain.
- pSurfaceFormatCount is a pointer to an integer related to the number of format pairs available or queried, as described below.
- pSurfaceFormats is either NULL or a pointer to an array of VkSurfaceFormatKHR structures.

If pSurfaceFormats is NULL, then the number of format pairs supported for the given surface is returned in pSurfaceFormatCount. Otherwise, pSurfaceFormatCount **must** point to a variable set by the application to the number of elements in the pSurfaceFormats array, and on return the variable is overwritten with the number of structures actually written to pSurfaceFormats. If the value of pSurfaceFormatCount is less than the number of format pairs supported, at most pSurfaceFormatCount structures will be written, and VK_INCOMPLETE will be returned instead of VK_SUCCESS, to indicate that not all the available format pairs were returned.

The number of format pairs supported **must** be greater than or equal to 1. pSurfaceFormats **must** not contain an entry whose value for format is VK_FORMAT_UNDEFINED.

If pSurfaceFormats includes an entry whose value for colorSpace is VK_COLOR_SPACE_SRGB_NONLINEAR_KHR and whose value for format is a UNORM (or SRGB) format and the corresponding SRGB (or UNORM) format is a color renderable format for VK_IMAGE_TILING_OPTIMAL, then pSurfaceFormats **must** also contain an entry with the same value for colorSpace and format equal to the corresponding SRGB (or UNORM) format.

If the VK_GOOGLE_surfaceless_query extension is enabled, the values returned in pSurfaceFormats will be identical for every valid surface created on this physical device, and so surface **can** be VK_NULL_HANDLE.

### Valid Usage

- VUID-vkGetPhysicalDeviceSurfaceFormatsKHR-surface-06524
  If the VK_GOOGLE_surfaceless_query extension is not enabled, surface **must** be a valid VkSurfaceKHR handle
- VUID-vkGetPhysicalDeviceSurfaceFormatsKHR-surface-06525
  If surface is not VK_NULL_HANDLE, surface **must** be supported by physicalDevice, as reported by vkGetPhysicalDeviceSurfaceSupportKHR or an equivalent platform-specific mechanism

### Valid Usage (Implicit)

- VUID-vkGetPhysicalDeviceSurfaceFormatsKHR-physicalDevice-parameter
  physicalDevice **must** be a valid VkPhysicalDevice handle
- VUID-vkGetPhysicalDeviceSurfaceFormatsKHR-surface-parameter
  If surface is not VK_NULL_HANDLE, surface **must** be a valid VkSurfaceKHR handle
- VUID-vkGetPhysicalDeviceSurfaceFormatsKHR-pSurfaceFormatCount-parameter
  pSurfaceFormatCount **must** be a valid pointer to a uint32_t value
- VUID-vkGetPhysicalDeviceSurfaceFormatsKHR-pSurfaceFormats-parameter

If the value referenced by pSurfaceFormatCount is not 0, and pSurfaceFormats is not NULL, pSurfaceFormats **must** be a valid pointer to an array of pSurfaceFormatCount VkSurfaceFormatKHR structures

- VUID-vkGetPhysicalDeviceSurfaceFormatsKHR-commonparent
  Both of physicalDevice, and surface that are valid handles of non-ignored parameters **must** have been created, allocated, or retrieved from the same VkInstance

## Return Codes

**Success**

- VK_SUCCESS
- VK_INCOMPLETE

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY
- VK_ERROR_SURFACE_LOST_KHR

The VkSurfaceFormatKHR structure is defined as:

```
// Provided by VK_KHR_surface
typedef struct VkSurfaceFormatKHR {
    VkFormat           format;
    VkColorSpaceKHR    colorSpace;
} VkSurfaceFormatKHR;
```

- format is a VkFormat that is compatible with the specified surface.
- colorSpace is a presentation VkColorSpaceKHR that is compatible with the surface.

To query the supported swapchain format tuples for a surface, call:

```
// Provided by VK_KHR_get_surface_capabilities2
VkResult vkGetPhysicalDeviceSurfaceFormats2KHR(
    VkPhysicalDevice                            physicalDevice,
    const VkPhysicalDeviceSurfaceInfo2KHR*      pSurfaceInfo,
    uint32_t*                                   pSurfaceFormatCount,
    VkSurfaceFormat2KHR*                        pSurfaceFormats);
```

- physicalDevice is the physical device that will be associated with the swapchain to be created, as described for vkCreateSwapchainKHR.
- pSurfaceInfo is a pointer to a VkPhysicalDeviceSurfaceInfo2KHR structure describing the surface and other fixed parameters that would be consumed by vkCreateSwapchainKHR.
- pSurfaceFormatCount is a pointer to an integer related to the number of format tuples available

or queried, as described below.

- pSurfaceFormats is either NULL or a pointer to an array of VkSurfaceFormat2KHR structures.

vkGetPhysicalDeviceSurfaceFormats2KHR behaves similarly to vkGetPhysicalDeviceSurfaceFormatsKHR, with the ability to be extended via pNext chains.

If pSurfaceFormats is NULL, then the number of format tuples supported for the given surface is returned in pSurfaceFormatCount. Otherwise, pSurfaceFormatCount **must** point to a variable set by the application to the number of elements in the pSurfaceFormats array, and on return the variable is overwritten with the number of structures actually written to pSurfaceFormats. If the value of pSurfaceFormatCount is less than the number of format tuples supported, at most pSurfaceFormatCount structures will be written, and VK_INCOMPLETE will be returned instead of VK_SUCCESS, to indicate that not all the available values were returned.

## Valid Usage

- VUID-vkGetPhysicalDeviceSurfaceFormats2KHR-pSurfaceInfo-06521
  If the VK_GOOGLE_surfaceless_query extension is not enabled, pSurfaceInfo->surface **must** be a valid VkSurfaceKHR handle
- VUID-vkGetPhysicalDeviceSurfaceFormats2KHR-pSurfaceInfo-06522
  If pSurfaceInfo->surface is not VK_NULL_HANDLE, pSurfaceInfo->surface **must** be supported by physicalDevice, as reported by vkGetPhysicalDeviceSurfaceSupportKHR or an equivalent platform-specific mechanism

## Valid Usage (Implicit)

- VUID-vkGetPhysicalDeviceSurfaceFormats2KHR-physicalDevice-parameter
  physicalDevice **must** be a valid VkPhysicalDevice handle
- VUID-vkGetPhysicalDeviceSurfaceFormats2KHR-pSurfaceInfo-parameter
  pSurfaceInfo **must** be a valid pointer to a valid VkPhysicalDeviceSurfaceInfo2KHR structure
- VUID-vkGetPhysicalDeviceSurfaceFormats2KHR-pSurfaceFormatCount-parameter
  pSurfaceFormatCount **must** be a valid pointer to a uint32_t value
- VUID-vkGetPhysicalDeviceSurfaceFormats2KHR-pSurfaceFormats-parameter
  If the value referenced by pSurfaceFormatCount is not 0, and pSurfaceFormats is not NULL, pSurfaceFormats **must** be a valid pointer to an array of pSurfaceFormatCount VkSurfaceFormat2KHR structures

## Return Codes

**Success**

- VK_SUCCESS
- VK_INCOMPLETE

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY
- VK_ERROR_SURFACE_LOST_KHR

The VkSurfaceFormat2KHR structure is defined as:

```
// Provided by VK_KHR_get_surface_capabilities2
typedef struct VkSurfaceFormat2KHR {
    VkStructureType       sType;
    void*                 pNext;
    VkSurfaceFormatKHR    surfaceFormat;
} VkSurfaceFormat2KHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- surfaceFormat is a VkSurfaceFormatKHR structure describing a format-color space pair that is compatible with the specified surface.

If the imageCompressionControlSwapchain feature is supported and a VkImageCompressionPropertiesEXT structure is included in the pNext chain of this structure, then it will be filled with the compression properties that are supported for the surfaceFormat.

### Valid Usage

- VUID-VkSurfaceFormat2KHR-pNext-06750
  If the imageCompressionControlSwapchain feature is not enabled, the pNext chain **must** not include an VkImageCompressionPropertiesEXT structure

### Valid Usage (Implicit)

- VUID-VkSurfaceFormat2KHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_SURFACE_FORMAT_2_KHR
- VUID-VkSurfaceFormat2KHR-pNext-pNext
  pNext **must** be NULL or a pointer to a valid instance of VkImageCompressionPropertiesEXT
- VUID-VkSurfaceFormat2KHR-sType-unique
  The sType value of each struct in the pNext chain **must** be unique

While the format of a presentable image refers to the encoding of each pixel, the colorSpace determines how the presentation engine interprets the pixel values. A color space in this document refers to a specific color space (defined by the chromaticities of its primaries and a white point in CIE Lab), and transfer functions indicating the mapping between the image data and the

colorimetry with respect to the given color space.

Possible values of VkSurfaceFormatKHR::colorSpace, specifying the color spaces that a presentation engine can accept, are:

```
// Provided by VK_KHR_surface
typedef enum VkColorSpaceKHR {
    VK_COLOR_SPACE_SRGB_NONLINEAR_KHR = 0,
  // Provided by VK_EXT_swapchain_colorspace
    VK_COLOR_SPACE_DISPLAY_P3_NONLINEAR_EXT = 1000104001,
  // Provided by VK_EXT_swapchain_colorspace
    VK_COLOR_SPACE_EXTENDED_SRGB_LINEAR_EXT = 1000104002,
  // Provided by VK_EXT_swapchain_colorspace
    VK_COLOR_SPACE_DISPLAY_P3_LINEAR_EXT = 1000104003,
  // Provided by VK_EXT_swapchain_colorspace
    VK_COLOR_SPACE_DCI_P3_NONLINEAR_EXT = 1000104004,
  // Provided by VK_EXT_swapchain_colorspace
    VK_COLOR_SPACE_BT709_LINEAR_EXT = 1000104005,
  // Provided by VK_EXT_swapchain_colorspace
    VK_COLOR_SPACE_BT709_NONLINEAR_EXT = 1000104006,
  // Provided by VK_EXT_swapchain_colorspace
    VK_COLOR_SPACE_BT2020_LINEAR_EXT = 1000104007,
  // Provided by VK_EXT_swapchain_colorspace
    VK_COLOR_SPACE_HDR10_ST2084_EXT = 1000104008,
  // Provided by VK_EXT_swapchain_colorspace
  // VK_COLOR_SPACE_DOLBYVISION_EXT is deprecated, but no reason was given in the API
XML
    VK_COLOR_SPACE_DOLBYVISION_EXT = 1000104009,
  // Provided by VK_EXT_swapchain_colorspace
    VK_COLOR_SPACE_HDR10_HLG_EXT = 1000104010,
  // Provided by VK_EXT_swapchain_colorspace
    VK_COLOR_SPACE_ADOBERGB_LINEAR_EXT = 1000104011,
  // Provided by VK_EXT_swapchain_colorspace
    VK_COLOR_SPACE_ADOBERGB_NONLINEAR_EXT = 1000104012,
  // Provided by VK_EXT_swapchain_colorspace
    VK_COLOR_SPACE_PASS_THROUGH_EXT = 1000104013,
  // Provided by VK_EXT_swapchain_colorspace
    VK_COLOR_SPACE_EXTENDED_SRGB_NONLINEAR_EXT = 1000104014,
  // Provided by VK_AMD_display_native_hdr
    VK_COLOR_SPACE_DISPLAY_NATIVE_AMD = 1000213000,
  // VK_COLORSPACE_SRGB_NONLINEAR_KHR is a deprecated alias
    VK_COLORSPACE_SRGB_NONLINEAR_KHR = VK_COLOR_SPACE_SRGB_NONLINEAR_KHR,
  // Provided by VK_EXT_swapchain_colorspace
  // VK_COLOR_SPACE_DCI_P3_LINEAR_EXT is a deprecated alias
    VK_COLOR_SPACE_DCI_P3_LINEAR_EXT = VK_COLOR_SPACE_DISPLAY_P3_LINEAR_EXT,
} VkColorSpaceKHR;
```

- VK_COLOR_SPACE_SRGB_NONLINEAR_KHR specifies support for the images in sRGB color space, encoded according to the sRGB specification.

- VK_COLOR_SPACE_DISPLAY_P3_NONLINEAR_EXT specifies support for the images in Display-P3 color space, encoded using a Display-P3 transfer function.
- VK_COLOR_SPACE_EXTENDED_SRGB_LINEAR_EXT specifies support for the images in extended sRGB color space, encoded using a linear transfer function.
- VK_COLOR_SPACE_EXTENDED_SRGB_NONLINEAR_EXT specifies support for the images in extended sRGB color space, encoded according to the scRGB specification.
- VK_COLOR_SPACE_DISPLAY_P3_LINEAR_EXT specifies support for the images in Display-P3 color space, encoded using a linear transfer function.
- VK_COLOR_SPACE_DCI_P3_NONLINEAR_EXT specifies support for the images in DCI-P3 color space, encoded according to the DCI-P3 specification. Note that values in such an image are interpreted as XYZ encoded color data by the presentation engine.
- VK_COLOR_SPACE_BT709_LINEAR_EXT specifies support for the images in BT709 color space, encoded using a linear transfer function.
- VK_COLOR_SPACE_BT709_NONLINEAR_EXT specifies support for the images in BT709 color space, encoded according to the BT709 specification.
- VK_COLOR_SPACE_BT2020_LINEAR_EXT specifies support for the images in BT2020 color space, encoded using a linear transfer function.
- VK_COLOR_SPACE_HDR10_ST2084_EXT specifies support for the images in HDR10 (BT2020) color space, encoded according to SMPTE ST2084 Perceptual Quantizer (PQ) specification.
- VK_COLOR_SPACE_HDR10_HLG_EXT specifies support for the images in HDR10 (BT2020) color space, encoded according to the Hybrid Log Gamma (HLG) specification.
- VK_COLOR_SPACE_ADOBERGB_LINEAR_EXT specifies support for images in Adobe RGB color space, encoded using a linear transfer function.
- VK_COLOR_SPACE_ADOBERGB_NONLINEAR_EXT specifies support for the images in Adobe RGB color space, encoded according to the Adobe RGB specification (approximately Gamma 2.2).
- VK_COLOR_SPACE_PASS_THROUGH_EXT specifies that color components are used “as is”. This is intended to allow applications to supply data for color spaces not described here.
- VK_COLOR_SPACE_DISPLAY_NATIVE_AMD specifies support for the display’s native color space. This matches the color space expectations of AMD’s FreeSync2 standard, for displays supporting it.

**NOTE** In the initial release of the VK_KHR_surface and VK_KHR_swapchain extensions, the token VK_COLORSPACE_SRGB_NONLINEAR_KHR was used. Starting in the 2016-05-13 updates to the extension branches, matching release 1.0.13 of the core API specification, VK_COLOR_SPACE_SRGB_NONLINEAR_KHR is used instead for consistency with Vulkan naming rules. The older enum is still available for backwards compatibility.

**NOTE** In older versions of this extension VK_COLOR_SPACE_DISPLAY_P3_LINEAR_EXT was misnamed VK_COLOR_SPACE_DCI_P3_LINEAR_EXT. This has been updated to indicate that it uses RGB color encoding, not XYZ. The old name is deprecated but is maintained for backwards compatibility.

**NOTE** In older versions of the VK_EXT_swapchain_colorspace extension, VK_COLOR_SPACE_DOLBYVISION_EXT was exposed. The intent was to indicate the presentation engine shall decode an image using the SMPTE ST 2084 Perceptual Quantizer (PQ) EOTF, and then apply a proprietary OOTF to process the image. However, Dolby Vision profile 8.4 describes an encoding using the Hybrid Log Gamma (HLG) OETF, and there is no swapchain extension for signaling Dolby Vision metadata to be used by a proprietary OOTF. This enum is deprecated but is maintained for backwards compatibility.

**NOTE** *Note*

For a traditional "Linear" or non-gamma transfer function color space use VK_COLOR_SPACE_PASS_THROUGH_EXT.

The presentation engine interprets the pixel values of the R, G, and B components as having been encoded using an appropriate transfer function. Applications **should** ensure that the appropriate transfer function has been applied. Textures Output Format Conversion requires that all implementations implicitly apply the sRGB $EOTF^{-1}$ on R, G, and B components when shaders write to an sRGB pixel format image, which is useful for sRGB color spaces. For sRGB color spaces with other pixel formats, or other non-linear color spaces, applications **can** apply the transfer function explicitly in a shader. The A channel is always interpreted as linearly encoded.

This extension defines enums for VkColorSpaceKHR that correspond to the following color spaces:

*Table 49. Color Spaces and Attributes*

| Name | Red Primary | Green Primary | Blue Primary | White-point | Transfer function |
|---|---|---|---|---|---|
| DCI-P3 | 1.000, 0.000 | 0.000, 1.000 | 0.000, 0.000 | 0.3333, 0.3333 | DCI P3 |
| Display-P3 | 0.680, 0.320 | 0.265, 0.690 | 0.150, 0.060 | 0.3127, 0.3290 (D65) | Display-P3 |
| BT709 | 0.640, 0.330 | 0.300, 0.600 | 0.150, 0.060 | 0.3127, 0.3290 (D65) | BT709 |
| sRGB | 0.640, 0.330 | 0.300, 0.600 | 0.150, 0.060 | 0.3127, 0.3290 (D65) | sRGB |
| extended sRGB | 0.640, 0.330 | 0.300, 0.600 | 0.150, 0.060 | 0.3127, 0.3290 (D65) | scRGB |
| HDR10_ST2084 | 0.708, 0.292 | 0.170, 0.797 | 0.131, 0.046 | 0.3127, 0.3290 (D65) | ST2084 PQ |
| HDR10_HLG | 0.708, 0.292 | 0.170, 0.797 | 0.131, 0.046 | 0.3127, 0.3290 (D65) | HLG |
| Adobe RGB | 0.640, 0.330 | 0.210, 0.710 | 0.150, 0.060 | 0.3127, 0.3290 (D65) | Adobe RGB |

The transfer functions are described in the "Transfer Functions" chapter of the Khronos Data Format Specification.

Except Display-P3 OETF, which is:

$$E = \begin{cases} 1.055 \times L^{\frac{1}{2.4}} - 0.055 & \text{for } 0.0030186 \leq L \leq 1 \\ 12.92 \times L & \text{for } 0 \leq L < 0.0030186 \end{cases}$$

where L is the linear value of a color component and E is the encoded value (as stored in the image in memory).

**NOTE** For most uses, the sRGB OETF is equivalent.

### 34.5.3. Surface Presentation Mode Support

To query the supported presentation modes for a surface, call:

```
// Provided by VK_KHR_surface
VkResult vkGetPhysicalDeviceSurfacePresentModesKHR(
    VkPhysicalDevice                            physicalDevice,
    VkSurfaceKHR                                surface,
    uint32_t*                                   pPresentModeCount,
    VkPresentModeKHR*                           pPresentModes);
```

- `physicalDevice` is the physical device that will be associated with the swapchain to be created, as described for vkCreateSwapchainKHR.
- `surface` is the surface that will be associated with the swapchain.
- `pPresentModeCount` is a pointer to an integer related to the number of presentation modes available or queried, as described below.
- `pPresentModes` is either `NULL` or a pointer to an array of VkPresentModeKHR values, indicating the supported presentation modes.

If `pPresentModes` is `NULL`, then the number of presentation modes supported for the given `surface` is returned in `pPresentModeCount`. Otherwise, `pPresentModeCount` **must** point to a variable set by the application to the number of elements in the `pPresentModes` array, and on return the variable is overwritten with the number of values actually written to `pPresentModes`. If the value of `pPresentModeCount` is less than the number of presentation modes supported, at most `pPresentModeCount` values will be written, and `VK_INCOMPLETE` will be returned instead of `VK_SUCCESS`, to indicate that not all the available modes were returned.

If the `VK_GOOGLE_surfaceless_query` extension is enabled and `surface` is VK_NULL_HANDLE, the values returned in `pPresentModes` will only indicate support for `VK_PRESENT_MODE_FIFO_KHR`, `VK_PRESENT_MODE_SHARED_DEMAND_REFRESH_KHR`, and `VK_PRESENT_MODE_SHARED_CONTINUOUS_REFRESH_KHR`. To query support for any other present mode, a valid handle **must** be provided in `surface`.

#### Valid Usage

- VUID-vkGetPhysicalDeviceSurfacePresentModesKHR-surface-06524

If the VK_GOOGLE_surfaceless_query extension is not enabled, surface **must** be a valid VkSurfaceKHR handle

- VUID-vkGetPhysicalDeviceSurfacePresentModesKHR-surface-06525
  If surface is not VK_NULL_HANDLE, surface **must** be supported by physicalDevice, as reported by vkGetPhysicalDeviceSurfaceSupportKHR or an equivalent platform-specific mechanism

## Valid Usage (Implicit)

- VUID-vkGetPhysicalDeviceSurfacePresentModesKHR-physicalDevice-parameter
  physicalDevice **must** be a valid VkPhysicalDevice handle
- VUID-vkGetPhysicalDeviceSurfacePresentModesKHR-surface-parameter
  If surface is not VK_NULL_HANDLE, surface **must** be a valid VkSurfaceKHR handle
- VUID-vkGetPhysicalDeviceSurfacePresentModesKHR-pPresentModeCount-parameter
  pPresentModeCount **must** be a valid pointer to a uint32_t value
- VUID-vkGetPhysicalDeviceSurfacePresentModesKHR-pPresentModes-parameter
  If the value referenced by pPresentModeCount is not 0, and pPresentModes is not NULL, pPresentModes **must** be a valid pointer to an array of pPresentModeCount VkPresentModeKHR values
- VUID-vkGetPhysicalDeviceSurfacePresentModesKHR-commonparent
  Both of physicalDevice, and surface that are valid handles of non-ignored parameters **must** have been created, allocated, or retrieved from the same VkInstance

## Return Codes

### Success

- VK_SUCCESS
- VK_INCOMPLETE

### Failure

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY
- VK_ERROR_SURFACE_LOST_KHR

Alternatively, to query the supported presentation modes for a surface combined with select other fixed swapchain creation parameters, call:

```
// Provided by VK_EXT_full_screen_exclusive
VkResult vkGetPhysicalDeviceSurfacePresentModes2EXT(
    VkPhysicalDevice                            physicalDevice,
    const VkPhysicalDeviceSurfaceInfo2KHR*      pSurfaceInfo,
    uint32_t*                                   pPresentModeCount,
    VkPresentModeKHR*                           pPresentModes);
```

- `physicalDevice` is the physical device that will be associated with the swapchain to be created, as described for vkCreateSwapchainKHR.
- `pSurfaceInfo` is a pointer to a VkPhysicalDeviceSurfaceInfo2KHR structure describing the surface and other fixed parameters that would be consumed by vkCreateSwapchainKHR.
- `pPresentModeCount` is a pointer to an integer related to the number of presentation modes available or queried, as described below.
- `pPresentModes` is either `NULL` or a pointer to an array of VkPresentModeKHR values, indicating the supported presentation modes.

`vkGetPhysicalDeviceSurfacePresentModes2EXT` behaves similarly to vkGetPhysicalDeviceSurfacePresentModesKHR, with the ability to specify extended inputs via chained input structures.

## Valid Usage

- VUID-vkGetPhysicalDeviceSurfacePresentModes2EXT-pSurfaceInfo-06521
  If the `VK_GOOGLE_surfaceless_query` extension is not enabled, `pSurfaceInfo->surface` **must** be a valid VkSurfaceKHR handle
- VUID-vkGetPhysicalDeviceSurfacePresentModes2EXT-pSurfaceInfo-06522
  If `pSurfaceInfo->surface` is not VK_NULL_HANDLE, `pSurfaceInfo->surface` **must** be supported by `physicalDevice`, as reported by vkGetPhysicalDeviceSurfaceSupportKHR or an equivalent platform-specific mechanism

## Valid Usage (Implicit)

- VUID-vkGetPhysicalDeviceSurfacePresentModes2EXT-physicalDevice-parameter
  `physicalDevice` **must** be a valid VkPhysicalDevice handle
- VUID-vkGetPhysicalDeviceSurfacePresentModes2EXT-pSurfaceInfo-parameter
  `pSurfaceInfo` **must** be a valid pointer to a valid VkPhysicalDeviceSurfaceInfo2KHR structure
- VUID-vkGetPhysicalDeviceSurfacePresentModes2EXT-pPresentModeCount-parameter
  `pPresentModeCount` **must** be a valid pointer to a `uint32_t` value
- VUID-vkGetPhysicalDeviceSurfacePresentModes2EXT-pPresentModes-parameter
  If the value referenced by `pPresentModeCount` is not `0`, and `pPresentModes` is not `NULL`, `pPresentModes` **must** be a valid pointer to an array of `pPresentModeCount` VkPresentModeKHR values

### Return Codes

**Success**

- VK_SUCCESS
- VK_INCOMPLETE

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY
- VK_ERROR_SURFACE_LOST_KHR

Possible values of elements of the vkGetPhysicalDeviceSurfacePresentModesKHR::pPresentModes array, indicating the supported presentation modes for a surface, are:

```
// Provided by VK_KHR_surface
typedef enum VkPresentModeKHR {
    VK_PRESENT_MODE_IMMEDIATE_KHR = 0,
    VK_PRESENT_MODE_MAILBOX_KHR = 1,
    VK_PRESENT_MODE_FIFO_KHR = 2,
    VK_PRESENT_MODE_FIFO_RELAXED_KHR = 3,
  // Provided by VK_KHR_shared_presentable_image
    VK_PRESENT_MODE_SHARED_DEMAND_REFRESH_KHR = 1000111000,
  // Provided by VK_KHR_shared_presentable_image
    VK_PRESENT_MODE_SHARED_CONTINUOUS_REFRESH_KHR = 1000111001,
} VkPresentModeKHR;
```

- VK_PRESENT_MODE_IMMEDIATE_KHR specifies that the presentation engine does not wait for a vertical blanking period to update the current image, meaning this mode **may** result in visible tearing. No internal queuing of presentation requests is needed, as the requests are applied immediately.
- VK_PRESENT_MODE_MAILBOX_KHR specifies that the presentation engine waits for the next vertical blanking period to update the current image. Tearing **cannot** be observed. An internal single-entry queue is used to hold pending presentation requests. If the queue is full when a new presentation request is received, the new request replaces the existing entry, and any images associated with the prior entry become available for reuse by the application. One request is removed from the queue and processed during each vertical blanking period in which the queue is non-empty.
- VK_PRESENT_MODE_FIFO_KHR specifies that the presentation engine waits for the next vertical blanking period to update the current image. Tearing **cannot** be observed. An internal queue is used to hold pending presentation requests. New requests are appended to the end of the queue, and one request is removed from the beginning of the queue and processed during each vertical blanking period in which the queue is non-empty. This is the only value of presentMode that is **required** to be supported.
- VK_PRESENT_MODE_FIFO_RELAXED_KHR specifies that the presentation engine generally waits for the

next vertical blanking period to update the current image. If a vertical blanking period has already passed since the last update of the current image then the presentation engine does not wait for another vertical blanking period for the update, meaning this mode **may** result in visible tearing in this case. This mode is useful for reducing visual stutter with an application that will mostly present a new image before the next vertical blanking period, but may occasionally be late, and present a new image just after the next vertical blanking period. An internal queue is used to hold pending presentation requests. New requests are appended to the end of the queue, and one request is removed from the beginning of the queue and processed during or after each vertical blanking period in which the queue is non-empty.

- `VK_PRESENT_MODE_SHARED_DEMAND_REFRESH_KHR` specifies that the presentation engine and application have concurrent access to a single image, which is referred to as a *shared presentable image*. The presentation engine is only required to update the current image after a new presentation request is received. Therefore the application **must** make a presentation request whenever an update is required. However, the presentation engine **may** update the current image at any point, meaning this mode **may** result in visible tearing.
- `VK_PRESENT_MODE_SHARED_CONTINUOUS_REFRESH_KHR` specifies that the presentation engine and application have concurrent access to a single image, which is referred to as a *shared presentable image*. The presentation engine periodically updates the current image on its regular refresh cycle. The application is only required to make one initial presentation request, after which the presentation engine **must** update the current image without any need for further presentation requests. The application **can** indicate the image contents have been updated by making a presentation request, but this does not guarantee the timing of when it will be updated. This mode **may** result in visible tearing if rendering to the image is not timed correctly.

The supported VkImageUsageFlagBits of the presentable images of a swapchain created for a surface **may** differ depending on the presentation mode, and can be determined as per the table below:

*Table 50. Presentable image usage queries*

| Presentation mode | Image usage flags |
|---|---|
| `VK_PRESENT_MODE_IMMEDIATE_KHR` | VkSurfaceCapabilitiesKHR::`supportedUsageFlags` |
| `VK_PRESENT_MODE_MAILBOX_KHR` | VkSurfaceCapabilitiesKHR::`supportedUsageFlags` |
| `VK_PRESENT_MODE_FIFO_KHR` | VkSurfaceCapabilitiesKHR::`supportedUsageFlags` |
| `VK_PRESENT_MODE_FIFO_RELAXED_KHR` | VkSurfaceCapabilitiesKHR::`supportedUsageFlags` |
| `VK_PRESENT_MODE_SHARED_DEMAND_REFRESH_KHR` | VkSharedPresentSurfaceCapabilitiesKHR::`sharedPresentSupportedUsageFlags` |
| `VK_PRESENT_MODE_SHARED_CONTINUOUS_REFRESH_KHR` | VkSharedPresentSurfaceCapabilitiesKHR::`sharedPresentSupportedUsageFlags` |

**NOTE** For reference, the mode indicated by `VK_PRESENT_MODE_FIFO_KHR` is equivalent to the behavior of {wgl|glX|egl}SwapBuffers with a swap interval of 1, while the mode indicated by `VK_PRESENT_MODE_FIFO_RELAXED_KHR` is equivalent to the behavior of {wgl|glX}SwapBuffers with a swap interval of -1 (from the {WGL|GLX}_EXT_swap_control_tear extensions).

## 34.6. Full Screen Exclusive Control

Swapchains created with `fullScreenExclusive` set to `VK_FULL_SCREEN_EXCLUSIVE_APPLICATION_CONTROLLED_EXT` **must** acquire and release exclusive full-screen access explicitly, using the following commands.

To acquire exclusive full-screen access for a swapchain, call:

```
// Provided by VK_EXT_full_screen_exclusive
VkResult vkAcquireFullScreenExclusiveModeEXT(
    VkDevice                                    device,
    VkSwapchainKHR                              swapchain);
```

- `device` is the device associated with `swapchain`.
- `swapchain` is the swapchain to acquire exclusive full-screen access for.

### Valid Usage

- VUID-vkAcquireFullScreenExclusiveModeEXT-swapchain-02674
  `swapchain` **must** not be in the retired state
- VUID-vkAcquireFullScreenExclusiveModeEXT-swapchain-02675
  `swapchain` **must** be a swapchain created with a VkSurfaceFullScreenExclusiveInfoEXT structure, with `fullScreenExclusive` set to `VK_FULL_SCREEN_EXCLUSIVE_APPLICATION_CONTROLLED_EXT`
- VUID-vkAcquireFullScreenExclusiveModeEXT-swapchain-02676
  `swapchain` **must** not currently have exclusive full-screen access

A return value of `VK_SUCCESS` indicates that the `swapchain` successfully acquired exclusive full-screen access. The swapchain will retain this exclusivity until either the application releases exclusive full-screen access with vkReleaseFullScreenExclusiveModeEXT, destroys the swapchain, or if any of the swapchain commands return `VK_ERROR_FULL_SCREEN_EXCLUSIVE_MODE_LOST_EXT` indicating that the mode was lost because of platform-specific changes.

If the swapchain was unable to acquire exclusive full-screen access to the display then `VK_ERROR_INITIALIZATION_FAILED` is returned. An application **can** attempt to acquire exclusive full-screen access again for the same swapchain even if this command fails, or if `VK_ERROR_FULL_SCREEN_EXCLUSIVE_MODE_LOST_EXT` has been returned by a swapchain command.

### Valid Usage (Implicit)

- VUID-vkAcquireFullScreenExclusiveModeEXT-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkAcquireFullScreenExclusiveModeEXT-swapchain-parameter
  `swapchain` **must** be a valid VkSwapchainKHR handle

- VUID-vkAcquireFullScreenExclusiveModeEXT-swapchain-parent
  `swapchain` **must** have been created, allocated, or retrieved from `device`

### Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_INITIALIZATION_FAILED`
- `VK_ERROR_SURFACE_LOST_KHR`

To release exclusive full-screen access from a swapchain, call:

```
// Provided by VK_EXT_full_screen_exclusive
VkResult vkReleaseFullScreenExclusiveModeEXT(
    VkDevice                                    device,
    VkSwapchainKHR                              swapchain);
```

- `device` is the device associated with `swapchain`.
- `swapchain` is the swapchain to release exclusive full-screen access from.

**NOTE** Applications will not be able to present to `swapchain` after this call until exclusive full-screen access is reacquired. This is usually useful to handle when an application is minimized or otherwise intends to stop presenting for a time.

### Valid Usage

- VUID-vkReleaseFullScreenExclusiveModeEXT-swapchain-02677
  `swapchain` **must** not be in the retired state
- VUID-vkReleaseFullScreenExclusiveModeEXT-swapchain-02678
  `swapchain` **must** be a swapchain created with a VkSurfaceFullScreenExclusiveInfoEXT structure, with `fullScreenExclusive` set to `VK_FULL_SCREEN_EXCLUSIVE_APPLICATION_CONTROLLED_EXT`

### Valid Usage (Implicit)

- VUID-vkReleaseFullScreenExclusiveModeEXT-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkReleaseFullScreenExclusiveModeEXT-swapchain-parameter

swapchain **must** be a valid VkSwapchainKHR handle

- VUID-vkReleaseFullScreenExclusiveModeEXT-swapchain-parent
  swapchain **must** have been created, allocated, or retrieved from device

### Return Codes

**Success**

- VK_SUCCESS

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY
- VK_ERROR_SURFACE_LOST_KHR

## 34.7. Device Group Queries

A logical device that represents multiple physical devices **may** support presenting from images on more than one physical device, or combining images from multiple physical devices.

To query these capabilities, call:

```
// Provided by VK_VERSION_1_1 with VK_KHR_swapchain, VK_KHR_device_group with
VK_KHR_surface
VkResult vkGetDeviceGroupPresentCapabilitiesKHR(
    VkDevice                                    device,
    VkDeviceGroupPresentCapabilitiesKHR*        pDeviceGroupPresentCapabilities);
```

- device is the logical device.
- pDeviceGroupPresentCapabilities is a pointer to a VkDeviceGroupPresentCapabilitiesKHR structure in which the device's capabilities are returned.

### Valid Usage (Implicit)

- VUID-vkGetDeviceGroupPresentCapabilitiesKHR-device-parameter
  device **must** be a valid VkDevice handle
- VUID-vkGetDeviceGroupPresentCapabilitiesKHR-pDeviceGroupPresentCapabilities-parameter
  pDeviceGroupPresentCapabilities **must** be a valid pointer to a VkDeviceGroupPresentCapabilitiesKHR structure

## Return Codes

**Success**

- VK_SUCCESS

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY

The VkDeviceGroupPresentCapabilitiesKHR structure is defined as:

```
// Provided by VK_VERSION_1_1 with VK_KHR_swapchain, VK_KHR_device_group with
VK_KHR_surface
typedef struct VkDeviceGroupPresentCapabilitiesKHR {
    VkStructureType                     sType;
    void*                               pNext;
    uint32_t                            presentMask[VK_MAX_DEVICE_GROUP_SIZE];
    VkDeviceGroupPresentModeFlagsKHR    modes;
} VkDeviceGroupPresentCapabilitiesKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- presentMask is an array of VK_MAX_DEVICE_GROUP_SIZE uint32_t masks, where the mask at element i is non-zero if physical device i has a presentation engine, and where bit j is set in element i if physical device i **can** present swapchain images from physical device j. If element i is non-zero, then bit i **must** be set.
- modes is a bitmask of VkDeviceGroupPresentModeFlagBitsKHR indicating which device group presentation modes are supported.

modes always has VK_DEVICE_GROUP_PRESENT_MODE_LOCAL_BIT_KHR set.

The present mode flags are also used when presenting an image, in VkDeviceGroupPresentInfoKHR::mode.

If a device group only includes a single physical device, then modes **must** equal VK_DEVICE_GROUP_PRESENT_MODE_LOCAL_BIT_KHR.

## Valid Usage (Implicit)

- VUID-VkDeviceGroupPresentCapabilitiesKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_DEVICE_GROUP_PRESENT_CAPABILITIES_KHR
- VUID-VkDeviceGroupPresentCapabilitiesKHR-pNext-pNext
  pNext **must** be NULL

Bits which **may** be set in VkDeviceGroupPresentCapabilitiesKHR::modes, indicating which device group presentation modes are supported, are:

```
// Provided by VK_VERSION_1_1 with VK_KHR_swapchain, VK_KHR_device_group with
VK_KHR_surface
typedef enum VkDeviceGroupPresentModeFlagBitsKHR {
    VK_DEVICE_GROUP_PRESENT_MODE_LOCAL_BIT_KHR = 0x00000001,
    VK_DEVICE_GROUP_PRESENT_MODE_REMOTE_BIT_KHR = 0x00000002,
    VK_DEVICE_GROUP_PRESENT_MODE_SUM_BIT_KHR = 0x00000004,
    VK_DEVICE_GROUP_PRESENT_MODE_LOCAL_MULTI_DEVICE_BIT_KHR = 0x00000008,
} VkDeviceGroupPresentModeFlagBitsKHR;
```

- VK_DEVICE_GROUP_PRESENT_MODE_LOCAL_BIT_KHR specifies that any physical device with a presentation engine **can** present its own swapchain images.
- VK_DEVICE_GROUP_PRESENT_MODE_REMOTE_BIT_KHR specifies that any physical device with a presentation engine **can** present swapchain images from any physical device in its presentMask.
- VK_DEVICE_GROUP_PRESENT_MODE_SUM_BIT_KHR specifies that any physical device with a presentation engine **can** present the sum of swapchain images from any physical devices in its presentMask.
- VK_DEVICE_GROUP_PRESENT_MODE_LOCAL_MULTI_DEVICE_BIT_KHR specifies that multiple physical devices with a presentation engine **can** each present their own swapchain images.

```
// Provided by VK_VERSION_1_1 with VK_KHR_swapchain, VK_KHR_device_group with
VK_KHR_surface
typedef VkFlags VkDeviceGroupPresentModeFlagsKHR;
```

VkDeviceGroupPresentModeFlagsKHR is a bitmask type for setting a mask of zero or more VkDeviceGroupPresentModeFlagBitsKHR.

Some surfaces **may** not be capable of using all the device group present modes.

To query the supported device group present modes for a particular surface, call:

```
// Provided by VK_VERSION_1_1 with VK_KHR_swapchain, VK_KHR_device_group with
VK_KHR_surface
VkResult vkGetDeviceGroupSurfacePresentModesKHR(
    VkDevice                                    device,
    VkSurfaceKHR                                surface,
    VkDeviceGroupPresentModeFlagsKHR*           pModes);
```

- device is the logical device.
- surface is the surface.
- pModes is a pointer to a VkDeviceGroupPresentModeFlagsKHR in which the supported device group present modes for the surface are returned.

The modes returned by this command are not invariant, and **may** change in response to the surface

being moved, resized, or occluded. These modes **must** be a subset of the modes returned by vkGetDeviceGroupPresentCapabilitiesKHR.

### Valid Usage

- VUID-vkGetDeviceGroupSurfacePresentModesKHR-surface-06212
  `surface` **must** be supported by all physical devices associated with `device`, as reported by vkGetPhysicalDeviceSurfaceSupportKHR or an equivalent platform-specific mechanism

### Valid Usage (Implicit)

- VUID-vkGetDeviceGroupSurfacePresentModesKHR-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkGetDeviceGroupSurfacePresentModesKHR-surface-parameter
  `surface` **must** be a valid VkSurfaceKHR handle
- VUID-vkGetDeviceGroupSurfacePresentModesKHR-pModes-parameter
  `pModes` **must** be a valid pointer to a VkDeviceGroupPresentModeFlagsKHR value
- VUID-vkGetDeviceGroupSurfacePresentModesKHR-commonparent
  Both of `device`, and `surface` **must** have been created, allocated, or retrieved from the same VkInstance

### Host Synchronization

- Host access to `surface` **must** be externally synchronized

### Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_SURFACE_LOST_KHR`

Alternatively, to query the supported device group presentation modes for a surface combined with select other fixed swapchain creation parameters, call:

```
// Provided by VK_EXT_full_screen_exclusive with VK_KHR_device_group or VK_VERSION_1_1
VkResult vkGetDeviceGroupSurfacePresentModes2EXT(
    VkDevice                                    device,
    const VkPhysicalDeviceSurfaceInfo2KHR*      pSurfaceInfo,
    VkDeviceGroupPresentModeFlagsKHR*           pModes);
```

- `device` is the logical device.
- `pSurfaceInfo` is a pointer to a VkPhysicalDeviceSurfaceInfo2KHR structure describing the surface and other fixed parameters that would be consumed by vkCreateSwapchainKHR.
- `pModes` is a pointer to a VkDeviceGroupPresentModeFlagsKHR in which the supported device group present modes for the surface are returned.

`vkGetDeviceGroupSurfacePresentModes2EXT` behaves similarly to vkGetDeviceGroupSurfacePresentModesKHR, with the ability to specify extended inputs via chained input structures.

## Valid Usage

- VUID-vkGetDeviceGroupSurfacePresentModes2EXT-pSurfaceInfo-06213
  `pSurfaceInfo->surface` **must** be supported by all physical devices associated with `device`, as reported by vkGetPhysicalDeviceSurfaceSupportKHR or an equivalent platform-specific mechanism

## Valid Usage (Implicit)

- VUID-vkGetDeviceGroupSurfacePresentModes2EXT-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkGetDeviceGroupSurfacePresentModes2EXT-pSurfaceInfo-parameter
  `pSurfaceInfo` **must** be a valid pointer to a valid VkPhysicalDeviceSurfaceInfo2KHR structure
- VUID-vkGetDeviceGroupSurfacePresentModes2EXT-pModes-parameter
  `pModes` **must** be a valid pointer to a VkDeviceGroupPresentModeFlagsKHR value

## Return Codes

### Success

- `VK_SUCCESS`

### Failure

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_SURFACE_LOST_KHR`

When using VK_DEVICE_GROUP_PRESENT_MODE_LOCAL_MULTI_DEVICE_BIT_KHR, the application **may** need to know which regions of the surface are used when presenting locally on each physical device. Presentation of swapchain images to this surface need only have valid contents in the regions returned by this command.

To query a set of rectangles used in presentation on the physical device, call:

```
// Provided by VK_VERSION_1_1 with VK_KHR_swapchain, VK_KHR_device_group with
VK_KHR_surface
VkResult vkGetPhysicalDevicePresentRectanglesKHR(
    VkPhysicalDevice                            physicalDevice,
    VkSurfaceKHR                                surface,
    uint32_t*                                   pRectCount,
    VkRect2D*                                   pRects);
```

- physicalDevice is the physical device.
- surface is the surface.
- pRectCount is a pointer to an integer related to the number of rectangles available or queried, as described below.
- pRects is either NULL or a pointer to an array of VkRect2D structures.

If pRects is NULL, then the number of rectangles used when presenting the given surface is returned in pRectCount. Otherwise, pRectCount **must** point to a variable set by the application to the number of elements in the pRects array, and on return the variable is overwritten with the number of structures actually written to pRects. If the value of pRectCount is less than the number of rectangles, at most pRectCount structures will be written, and VK_INCOMPLETE will be returned instead of VK_SUCCESS, to indicate that not all the available rectangles were returned.

The values returned by this command are not invariant, and **may** change in response to the surface being moved, resized, or occluded.

The rectangles returned by this command **must** not overlap.

### Valid Usage

- VUID-vkGetPhysicalDevicePresentRectanglesKHR-surface-06523
  surface **must** be a valid VkSurfaceKHR handle
- VUID-vkGetPhysicalDevicePresentRectanglesKHR-surface-06211
  surface **must** be supported by physicalDevice, as reported by vkGetPhysicalDeviceSurfaceSupportKHR or an equivalent platform-specific mechanism

### Valid Usage (Implicit)

- VUID-vkGetPhysicalDevicePresentRectanglesKHR-physicalDevice-parameter
  physicalDevice **must** be a valid VkPhysicalDevice handle

- VUID-vkGetPhysicalDevicePresentRectanglesKHR-surface-parameter
  `surface` **must** be a valid VkSurfaceKHR handle
- VUID-vkGetPhysicalDevicePresentRectanglesKHR-pRectCount-parameter
  `pRectCount` **must** be a valid pointer to a `uint32_t` value
- VUID-vkGetPhysicalDevicePresentRectanglesKHR-pRects-parameter
  If the value referenced by `pRectCount` is not `0`, and `pRects` is not `NULL`, `pRects` **must** be a valid pointer to an array of `pRectCount` VkRect2D structures
- VUID-vkGetPhysicalDevicePresentRectanglesKHR-commonparent
  Both of `physicalDevice`, and `surface` **must** have been created, allocated, or retrieved from the same VkInstance

### Host Synchronization

- Host access to `surface` **must** be externally synchronized

### Return Codes

**Success**

- `VK_SUCCESS`
- `VK_INCOMPLETE`

**Failure**

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`

## 34.8. Display Timing Queries

Traditional game and real-time-animation applications frequently use `VK_PRESENT_MODE_FIFO_KHR` so that presentable images are updated during the vertical blanking period of a given refresh cycle (RC) of the presentation engine's display. This avoids the visual anomaly known as tearing.

However, synchronizing the presentation of images with the RC does not prevent all forms of visual anomalies. Stuttering occurs when the geometry for each presentable image is not accurately positioned for when that image will be displayed. The geometry may appear to move too little some RCs, and too much for others. Sometimes the animation appears to freeze, when the same image is used for more than one RC.

In order to minimize stuttering, an application needs to correctly position their geometry for when the presentable image will be displayed to the user. To accomplish this, applications need various timing information about the presentation engine's display. They need to know when presentable images were actually presented, and when they could have been presented. Applications also need to tell the presentation engine to display an image no sooner than a given time. This can allow the application's animation to look smooth to the user, with no stuttering. The `VK_GOOGLE_display_timing`

extension allows an application to satisfy these needs.

The presentation engine's display typically refreshes the pixels that are displayed to the user on a periodic basis. The period may be fixed or variable. In many cases, the presentation engine is associated with fixed refresh rate (FRR) display technology, with a fixed refresh rate (RR, e.g. 60Hz). In some cases, the presentation engine is associated with variable refresh rate (VRR) display technology, where each refresh cycle (RC) can vary in length. This extension treats VRR displays as if they are FRR.

To query the duration of a refresh cycle (RC) for the presentation engine's display, call:

```
// Provided by VK_GOOGLE_display_timing
VkResult vkGetRefreshCycleDurationGOOGLE(
    VkDevice                                    device,
    VkSwapchainKHR                              swapchain,
    VkRefreshCycleDurationGOOGLE*               pDisplayTimingProperties);
```

- `device` is the device associated with `swapchain`.
- `swapchain` is the swapchain to obtain the refresh duration for.
- `pDisplayTimingProperties` is a pointer to a `VkRefreshCycleDurationGOOGLE` structure.

## Valid Usage (Implicit)

- VUID-vkGetRefreshCycleDurationGOOGLE-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkGetRefreshCycleDurationGOOGLE-swapchain-parameter
  `swapchain` **must** be a valid VkSwapchainKHR handle
- VUID-vkGetRefreshCycleDurationGOOGLE-pDisplayTimingProperties-parameter
  `pDisplayTimingProperties` **must** be a valid pointer to a VkRefreshCycleDurationGOOGLE structure
- VUID-vkGetRefreshCycleDurationGOOGLE-swapchain-parent
  `swapchain` **must** have been created, allocated, or retrieved from `device`

## Host Synchronization

- Host access to `swapchain` **must** be externally synchronized

## Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_DEVICE_LOST
- VK_ERROR_SURFACE_LOST_KHR

The VkRefreshCycleDurationGOOGLE structure is defined as:

```
// Provided by VK_GOOGLE_display_timing
typedef struct VkRefreshCycleDurationGOOGLE {
    uint64_t    refreshDuration;
} VkRefreshCycleDurationGOOGLE;
```

- refreshDuration is the number of nanoseconds from the start of one refresh cycle to the next.

**NOTE**

The rate at which an application renders and presents new images is known as the image present rate (IPR, aka frame rate). The inverse of IPR, or the duration between each image present, is the image present duration (IPD). In order to provide a smooth, stutter-free animation, an application will want its IPD to be a multiple of refreshDuration. For example, if a display has a 60Hz refresh rate, refreshDuration will be a value in nanoseconds that is approximately equal to 16.67ms. In such a case, an application will want an IPD of 16.67ms (1X multiplier of refreshDuration), or 33.33ms (2X multiplier of refreshDuration), or 50.0ms (3X multiplier of refreshDuration), etc.

In order to determine a target IPD for a display (i.e. a multiple of refreshDuration), an application needs to determine when its images are actually displayed. Suppose an application has an initial target IPD of 16.67ms (1X multiplier of refreshDuration). It will therefore position the geometry of a new image 16.67ms later than the previous image. But suppose this application is running on slower hardware, so that it actually takes 20ms to render each new image. This will create visual anomalies, because the images will not be displayed to the user every 16.67ms, nor every 20ms. In this case, it is better for the application to adjust its target IPD to 33.33ms (i.e. a 2X multiplier of refreshDuration), and tell the presentation engine to not present images any sooner than every 33.33ms. This will allow the geometry to be correctly positioned for each presentable image.

Adjustments to an application's IPD may be needed because different views of an application's geometry can take different amounts of time to render. For example, looking at the sky may take less time to render than looking at multiple, complex items in a room. In general, it is good to not frequently change IPD, as that can cause visual anomalies. Adjustments to a larger IPD because of late images should happen quickly, but adjustments to a smaller IPD should only happen if the actualPresentTime and earliestPresentTime members of the VkPastPresentationTimingGOOGLE structure are consistently different, and if presentMargin is consistently large, over multiple images.

The implementation will maintain a limited amount of history of timing information about previous presents. Because of the asynchronous nature of the presentation engine, the timing information for a given vkQueuePresentKHR command will become available some time later. These time values can be asynchronously queried, and will be returned if available. All time values are in nanoseconds, relative to a monotonically-increasing clock (e.g. CLOCK_MONOTONIC (see clock_gettime(2)) on Android and Linux).

To asynchronously query the presentation engine, for newly-available timing information about one or more previous presents to a given swapchain, call:

```
// Provided by VK_GOOGLE_display_timing
VkResult vkGetPastPresentationTimingGOOGLE(
    VkDevice                                    device,
    VkSwapchainKHR                              swapchain,
    uint32_t*                                   pPresentationTimingCount,
    VkPastPresentationTimingGOOGLE*             pPresentationTimings);
```

- device is the device associated with swapchain.
- swapchain is the swapchain to obtain presentation timing information duration for.
- pPresentationTimingCount is a pointer to an integer related to the number of VkPastPresentationTimingGOOGLE structures to query, as described below.
- pPresentationTimings is either NULL or a pointer to an array of VkPastPresentationTimingGOOGLE structures.

If pPresentationTimings is NULL, then the number of newly-available timing records for the given swapchain is returned in pPresentationTimingCount. Otherwise, pPresentationTimingCount **must** point to a variable set by the user to the number of elements in the pPresentationTimings array, and on return the variable is overwritten with the number of structures actually written to pPresentationTimings. If the value of pPresentationTimingCount is less than the number of newly-available timing records, at most pPresentationTimingCount structures will be written, and VK_INCOMPLETE will be returned instead of VK_SUCCESS, to indicate that not all the available timing records were returned.

### Valid Usage (Implicit)

- VUID-vkGetPastPresentationTimingGOOGLE-device-parameter
  device **must** be a valid VkDevice handle
- VUID-vkGetPastPresentationTimingGOOGLE-swapchain-parameter
  swapchain **must** be a valid VkSwapchainKHR handle
- VUID-vkGetPastPresentationTimingGOOGLE-pPresentationTimingCount-parameter
  pPresentationTimingCount **must** be a valid pointer to a uint32_t value
- VUID-vkGetPastPresentationTimingGOOGLE-pPresentationTimings-parameter
  If the value referenced by pPresentationTimingCount is not 0, and pPresentationTimings is not NULL, pPresentationTimings **must** be a valid pointer to an array of pPresentationTimingCount VkPastPresentationTimingGOOGLE structures

- VUID-vkGetPastPresentationTimingGOOGLE-swapchain-parent
  `swapchain` **must** have been created, allocated, or retrieved from `device`

## Host Synchronization

- Host access to `swapchain` **must** be externally synchronized

## Return Codes

**Success**

- `VK_SUCCESS`
- `VK_INCOMPLETE`

**Failure**

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_DEVICE_LOST`
- `VK_ERROR_OUT_OF_DATE_KHR`
- `VK_ERROR_SURFACE_LOST_KHR`

The `VkPastPresentationTimingGOOGLE` structure is defined as:

```
// Provided by VK_GOOGLE_display_timing
typedef struct VkPastPresentationTimingGOOGLE {
    uint32_t    presentID;
    uint64_t    desiredPresentTime;
    uint64_t    actualPresentTime;
    uint64_t    earliestPresentTime;
    uint64_t    presentMargin;
} VkPastPresentationTimingGOOGLE;
```

- `presentID` is an application-provided value that was given to a previous `vkQueuePresentKHR` command via VkPresentTimeGOOGLE::`presentID` (see below). It **can** be used to uniquely identify a previous present with the vkQueuePresentKHR command.
- `desiredPresentTime` is an application-provided value that was given to a previous vkQueuePresentKHR command via VkPresentTimeGOOGLE::`desiredPresentTime`. If non-zero, it was used by the application to indicate that an image not be presented any sooner than `desiredPresentTime`.
- `actualPresentTime` is the time when the image of the `swapchain` was actually displayed.
- `earliestPresentTime` is the time when the image of the `swapchain` could have been displayed. This **may** differ from `actualPresentTime` if the application requested that the image be presented no sooner than VkPresentTimeGOOGLE::`desiredPresentTime`.
- `presentMargin` is an indication of how early the `vkQueuePresentKHR` command was processed

compared to how soon it needed to be processed, and still be presented at earliestPresentTime.

The results for a given swapchain and presentID are only returned once from vkGetPastPresentationTimingGOOGLE.

The application **can** use the VkPastPresentationTimingGOOGLE values to occasionally adjust its timing. For example, if actualPresentTime is later than expected (e.g. one refreshDuration late), the application may increase its target IPD to a higher multiple of refreshDuration (e.g. decrease its frame rate from 60Hz to 30Hz). If actualPresentTime and earliestPresentTime are consistently different, and if presentMargin is consistently large enough, the application may decrease its target IPD to a smaller multiple of refreshDuration (e.g. increase its frame rate from 30Hz to 60Hz). If actualPresentTime and earliestPresentTime are same, and if presentMargin is consistently high, the application may delay the start of its input-render-present loop in order to decrease the latency between user input and the corresponding present (always leaving some margin in case a new image takes longer to render than the previous image). An application that desires its target IPD to always be the same as refreshDuration, can also adjust features until actualPresentTime is never late and presentMargin is satisfactory.

The full VK_GOOGLE_display_timing extension semantics are described for swapchains created with VK_PRESENT_MODE_FIFO_KHR. For example, non-zero values of VkPresentTimeGOOGLE::desiredPresentTime **must** be honored, and vkGetPastPresentationTimingGOOGLE **should** return a VkPastPresentationTimingGOOGLE structure with valid values for all images presented with vkQueuePresentKHR. The semantics for other present modes are as follows:

- VK_PRESENT_MODE_IMMEDIATE_KHR. The presentation engine **may** ignore non-zero values of VkPresentTimeGOOGLE::desiredPresentTime in favor of presenting immediately. The value of VkPastPresentationTimingGOOGLE::earliestPresentTime **must** be the same as VkPastPresentationTimingGOOGLE::actualPresentTime, which **should** be when the presentation engine displayed the image.
- VK_PRESENT_MODE_MAILBOX_KHR. The intention of using this present mode with this extension is to handle cases where an image is presented late, and the next image is presented soon enough to replace it at the next vertical blanking period. For images that are displayed to the user, the value of VkPastPresentationTimingGOOGLE::actualPresentTime **must** be when the image was displayed. For images that are not displayed to the user, vkGetPastPresentationTimingGOOGLE **may** not return a VkPastPresentationTimingGOOGLE structure, or it **may** return a VkPastPresentationTimingGOOGLE structure with the value of zero for both VkPastPresentationTimingGOOGLE::actualPresentTime and VkPastPresentationTimingGOOGLE::earliestPresentTime. It is possible that an application **can** submit images with VkPresentTimeGOOGLE::desiredPresentTime values such that new images **may** not be displayed. For example, if VkPresentTimeGOOGLE::desiredPresentTime is far enough in the future that an image is not presented before vkQueuePresentKHR is called to present another image, the first image will not be displayed to the user. If the application continues to do that, the presentation **may** not display new images.
- VK_PRESENT_MODE_FIFO_RELAXED_KHR. For images that are presented in time to be displayed at the next vertical blanking period, the semantics are identical as for VK_PRESENT_MODE_FIFO_KHR. For images that are presented late, and are displayed after the start of the vertical blanking period (i.e. with tearing), the values of VkPastPresentationTimingGOOGLE **may** be treated as if the image was displayed at the start of the vertical blanking period, or **may** be treated the same as for

VK_PRESENT_MODE_IMMEDIATE_KHR.

## 34.9. Present Wait

Applications wanting to control the pacing of the application by monitoring when presentation processes have completed to limit the number of outstanding images queued for presentation, need to have a method of being signaled during the presentation process.

Using the VK_GOOGLE_display_timing extension applications can discover when images were presented, but only asynchronously.

Providing a mechanism which allows applications to block, waiting for a specific step of the presentation process to complete allows them to control the amount of outstanding work (and hence the potential lag in responding to user input or changes in the rendering environment).

The VK_KHR_present_wait extension allows applications to tell the presentation engine at the vkQueuePresentKHR call that it plans on waiting for presentation by passing a VkPresentIdKHR structure. The presentId passed in that structure may then be passed to a future vkWaitForPresentKHR call to cause the application to block until that presentation is finished.

## 34.10. WSI Swapchain

A swapchain object (a.k.a. swapchain) provides the ability to present rendering results to a surface. Swapchain objects are represented by VkSwapchainKHR handles:

```
// Provided by VK_KHR_swapchain
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkSwapchainKHR)
```

A swapchain is an abstraction for an array of presentable images that are associated with a surface. The presentable images are represented by VkImage objects created by the platform. One image (which **can** be an array image for multiview/stereoscopic-3D surfaces) is displayed at a time, but multiple images **can** be queued for presentation. An application renders to the image, and then queues the image for presentation to the surface.

A native window **cannot** be associated with more than one non-retired swapchain at a time. Further, swapchains **cannot** be created for native windows that have a non-Vulkan graphics API surface associated with them.

**NOTE**

The presentation engine is an abstraction for the platform's compositor or display engine.

The presentation engine **may** be synchronous or asynchronous with respect to the application and/or logical device.

Some implementations **may** use the device's graphics queue or dedicated presentation hardware to perform presentation.

The presentable images of a swapchain are owned by the presentation engine. An application **can**

acquire use of a presentable image from the presentation engine. Use of a presentable image **must** occur only after the image is returned by vkAcquireNextImageKHR, and before it is released by vkQueuePresentKHR. This includes transitioning the image layout and rendering commands.

An application **can** acquire use of a presentable image with vkAcquireNextImageKHR. After acquiring a presentable image and before modifying it, the application **must** use a synchronization primitive to ensure that the presentation engine has finished reading from the image. The application **can** then transition the image's layout, queue rendering commands to it, etc. Finally, the application presents the image with vkQueuePresentKHR, which releases the acquisition of the image. The application **can** also release the acquisition of the image through vkReleaseSwapchainImagesEXT, if the image is not in use by the device, and skip the present operation.

The presentation engine controls the order in which presentable images are acquired for use by the application.

**NOTE** This allows the platform to handle situations which require out-of-order return of images after presentation. At the same time, it allows the application to generate command buffers referencing all of the images in the swapchain at initialization time, rather than in its main loop.

How this all works is described below.

If a swapchain is created with `presentMode` set to either `VK_PRESENT_MODE_SHARED_DEMAND_REFRESH_KHR` or `VK_PRESENT_MODE_SHARED_CONTINUOUS_REFRESH_KHR`, a single presentable image **can** be acquired, referred to as a shared presentable image. A shared presentable image **may** be concurrently accessed by the application and the presentation engine, without transitioning the image's layout after it is initially presented.

- With `VK_PRESENT_MODE_SHARED_DEMAND_REFRESH_KHR`, the presentation engine is only required to update to the latest contents of a shared presentable image after a present. The application **must** call `vkQueuePresentKHR` to guarantee an update. However, the presentation engine **may** update from it at any time.
- With `VK_PRESENT_MODE_SHARED_CONTINUOUS_REFRESH_KHR`, the presentation engine will automatically present the latest contents of a shared presentable image during every refresh cycle. The application is only required to make one initial call to `vkQueuePresentKHR`, after which the presentation engine will update from it without any need for further present calls. The application **can** indicate the image contents have been updated by calling `vkQueuePresentKHR`, but this does not guarantee the timing of when updates will occur.

The presentation engine **may** access a shared presentable image at any time after it is first presented. To avoid tearing, an application **should** coordinate access with the presentation engine. This requires presentation engine timing information through platform-specific mechanisms and ensuring that color attachment writes are made available during the portion of the presentation engine's refresh cycle they are intended for.

**NOTE** The `VK_KHR_shared_presentable_image` extension does not provide functionality for determining the timing of the presentation engine's refresh cycles.

In order to query a swapchain's status when rendering to a shared presentable image, call:

```
// Provided by VK_KHR_shared_presentable_image
VkResult vkGetSwapchainStatusKHR(
    VkDevice                                    device,
    VkSwapchainKHR                              swapchain);
```

- `device` is the device associated with `swapchain`.
- `swapchain` is the swapchain to query.

### Valid Usage (Implicit)

- VUID-vkGetSwapchainStatusKHR-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkGetSwapchainStatusKHR-swapchain-parameter
  `swapchain` **must** be a valid VkSwapchainKHR handle
- VUID-vkGetSwapchainStatusKHR-swapchain-parent
  `swapchain` **must** have been created, allocated, or retrieved from `device`

### Host Synchronization

- Host access to `swapchain` **must** be externally synchronized

### Return Codes

**Success**

- `VK_SUCCESS`
- `VK_SUBOPTIMAL_KHR`

**Failure**

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_DEVICE_LOST`
- `VK_ERROR_OUT_OF_DATE_KHR`
- `VK_ERROR_SURFACE_LOST_KHR`
- `VK_ERROR_FULL_SCREEN_EXCLUSIVE_MODE_LOST_EXT`

The possible return values for `vkGetSwapchainStatusKHR` **should** be interpreted as follows:

- `VK_SUCCESS` specifies the presentation engine is presenting the contents of the shared presentable image, as per the swapchain's VkPresentModeKHR.

- VK_SUBOPTIMAL_KHR the swapchain no longer matches the surface properties exactly, but the presentation engine is presenting the contents of the shared presentable image, as per the swapchain's VkPresentModeKHR.
- VK_ERROR_OUT_OF_DATE_KHR the surface has changed in such a way that it is no longer compatible with the swapchain.
- VK_ERROR_SURFACE_LOST_KHR the surface is no longer available.

| | |
|---|---|
| **NOTE** | The swapchain state **may** be cached by implementations, so applications **should** regularly call vkGetSwapchainStatusKHR when using a swapchain with VkPresentModeKHR set to VK_PRESENT_MODE_SHARED_CONTINUOUS_REFRESH_KHR. |

To create a swapchain, call:

```
// Provided by VK_KHR_swapchain
VkResult vkCreateSwapchainKHR(
    VkDevice                                    device,
    const VkSwapchainCreateInfoKHR*             pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkSwapchainKHR*                             pSwapchain);
```

- device is the device to create the swapchain for.
- pCreateInfo is a pointer to a VkSwapchainCreateInfoKHR structure specifying the parameters of the created swapchain.
- pAllocator is the allocator used for host memory allocated for the swapchain object when there is no more specific allocator available (see Memory Allocation).
- pSwapchain is a pointer to a VkSwapchainKHR handle in which the created swapchain object will be returned.

As mentioned above, if vkCreateSwapchainKHR succeeds, it will return a handle to a swapchain containing an array of at least pCreateInfo->minImageCount presentable images.

While acquired by the application, presentable images **can** be used in any way that equivalent non-presentable images **can** be used. A presentable image is equivalent to a non-presentable image created with the following VkImageCreateInfo parameters:

| VkImageCreateInfo Field | Value |
| --- | --- |
| flags | VK_IMAGE_CREATE_SPLIT_INSTANCE_BIND_REGIONS_BIT is set if VK_SWAPCHAIN_CREATE_SPLIT_INSTANCE_BIND_REGIONS_BIT_KHR is set<br><br>VK_IMAGE_CREATE_PROTECTED_BIT is set if VK_SWAPCHAIN_CREATE_PROTECTED_BIT_KHR is set<br><br>VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT and VK_IMAGE_CREATE_EXTENDED_USAGE_BIT_KHR are both set if VK_SWAPCHAIN_CREATE_MUTABLE_FORMAT_BIT_KHR is set<br><br>all other bits are unset |
| imageType | VK_IMAGE_TYPE_2D |
| format | pCreateInfo->imageFormat |
| extent | {pCreateInfo->imageExtent.width, pCreateInfo->imageExtent.height, 1} |
| mipLevels | 1 |
| arrayLayers | pCreateInfo->imageArrayLayers |
| samples | VK_SAMPLE_COUNT_1_BIT |
| tiling | VK_IMAGE_TILING_OPTIMAL |
| usage | pCreateInfo->imageUsage |
| sharingMode | pCreateInfo->imageSharingMode |
| queueFamilyIndexCount | pCreateInfo->queueFamilyIndexCount |
| pQueueFamilyIndices | pCreateInfo->pQueueFamilyIndices |
| initialLayout | VK_IMAGE_LAYOUT_UNDEFINED |

The pCreateInfo->surface **must** not be destroyed until after the swapchain is destroyed.

If oldSwapchain is VK_NULL_HANDLE, and the native window referred to by pCreateInfo->surface is already associated with a Vulkan swapchain, VK_ERROR_NATIVE_WINDOW_IN_USE_KHR **must** be returned.

If the native window referred to by pCreateInfo->surface is already associated with a non-Vulkan graphics API surface, VK_ERROR_NATIVE_WINDOW_IN_USE_KHR **must** be returned.

The native window referred to by pCreateInfo->surface **must** not become associated with a non-Vulkan graphics API surface before all associated Vulkan swapchains have been destroyed.

vkCreateSwapchainKHR will return VK_ERROR_DEVICE_LOST if the logical device was lost. The VkSwapchainKHR is a child of the device, and **must** be destroyed before the device. However, VkSurfaceKHR is not a child of any VkDevice and is not affected by the lost device. After successfully recreating a VkDevice, the same VkSurfaceKHR **can** be used to create a new VkSwapchainKHR, provided the previous one was destroyed.

If the `oldSwapchain` parameter of `pCreateInfo` is a valid swapchain, which has exclusive full-screen access, that access is released from `pCreateInfo->oldSwapchain`. If the command succeeds in this case, the newly created swapchain will automatically acquire exclusive full-screen access from `pCreateInfo->oldSwapchain`.

| | |
|---|---|
| **NOTE** | This implicit transfer is intended to avoid exiting and entering full-screen exclusive mode, which may otherwise cause unwanted visual updates to the display. |

In some cases, swapchain creation **may** fail if exclusive full-screen mode is requested for application control, but for some implementation-specific reason exclusive full-screen access is unavailable for the particular combination of parameters provided. If this occurs, `VK_ERROR_INITIALIZATION_FAILED` will be returned.

| | |
|---|---|
| **NOTE** | In particular, it will fail if the `imageExtent` member of `pCreateInfo` does not match the extents of the monitor. Other reasons for failure may include the application not being set as high-dpi aware, or if the physical device and monitor are not compatible in this mode. |

If the `pNext` chain of VkSwapchainCreateInfoKHR includes a VkSwapchainPresentBarrierCreateInfoNV structure, then that structure includes additional swapchain creation parameters specific to the present barrier. Swapchain creation **may** fail if the state of the current system restricts the usage of the present barrier feature VkSurfaceCapabilitiesPresentBarrierNV, or a swapchain itself does not satisfy all the required conditions. In this scenario `VK_ERROR_INITIALIZATION_FAILED` is returned.

When the VkSurfaceKHR in VkSwapchainCreateInfoKHR is a display surface, then the VkDisplayModeKHR in display surface's VkDisplaySurfaceCreateInfoKHR is associated with a particular VkDisplayKHR. Swapchain creation **may** fail if that VkDisplayKHR is not acquired by the application. In this scenario `VK_ERROR_INITIALIZATION_FAILED` is returned.

## Valid Usage (Implicit)

- VUID-vkCreateSwapchainKHR-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkCreateSwapchainKHR-pCreateInfo-parameter
  `pCreateInfo` **must** be a valid pointer to a valid VkSwapchainCreateInfoKHR structure
- VUID-vkCreateSwapchainKHR-pAllocator-parameter
  If `pAllocator` is not `NULL`, `pAllocator` **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkCreateSwapchainKHR-pSwapchain-parameter
  `pSwapchain` **must** be a valid pointer to a VkSwapchainKHR handle

## Host Synchronization

- Host access to `pCreateInfo->surface` **must** be externally synchronized

- Host access to `pCreateInfo->oldSwapchain` **must** be externally synchronized

### Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_DEVICE_LOST`
- `VK_ERROR_SURFACE_LOST_KHR`
- `VK_ERROR_NATIVE_WINDOW_IN_USE_KHR`
- `VK_ERROR_INITIALIZATION_FAILED`
- `VK_ERROR_COMPRESSION_EXHAUSTED_EXT`

The `VkSwapchainCreateInfoKHR` structure is defined as:

```
// Provided by VK_KHR_swapchain
typedef struct VkSwapchainCreateInfoKHR {
    VkStructureType                  sType;
    const void*                      pNext;
    VkSwapchainCreateFlagsKHR        flags;
    VkSurfaceKHR                     surface;
    uint32_t                         minImageCount;
    VkFormat                         imageFormat;
    VkColorSpaceKHR                  imageColorSpace;
    VkExtent2D                       imageExtent;
    uint32_t                         imageArrayLayers;
    VkImageUsageFlags                imageUsage;
    VkSharingMode                    imageSharingMode;
    uint32_t                         queueFamilyIndexCount;
    const uint32_t*                  pQueueFamilyIndices;
    VkSurfaceTransformFlagBitsKHR    preTransform;
    VkCompositeAlphaFlagBitsKHR      compositeAlpha;
    VkPresentModeKHR                 presentMode;
    VkBool32                         clipped;
    VkSwapchainKHR                   oldSwapchain;
} VkSwapchainCreateInfoKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is a bitmask of VkSwapchainCreateFlagBitsKHR indicating parameters of the swapchain creation.

- `surface` is the surface onto which the swapchain will present images. If the creation succeeds, the swapchain becomes associated with `surface`.
- `minImageCount` is the minimum number of presentable images that the application needs. The implementation will either create the swapchain with at least that many images, or it will fail to create the swapchain.
- `imageFormat` is a VkFormat value specifying the format the swapchain image(s) will be created with.
- `imageColorSpace` is a VkColorSpaceKHR value specifying the way the swapchain interprets image data.
- `imageExtent` is the size (in pixels) of the swapchain image(s). The behavior is platform-dependent if the image extent does not match the surface's `currentExtent` as returned by `vkGetPhysicalDeviceSurfaceCapabilitiesKHR`.

| | |
|---|---|
| **NOTE** | On some platforms, it is normal that `maxImageExtent` **may** become `(0, 0)`, for example when the window is minimized. In such a case, it is not possible to create a swapchain due to the Valid Usage requirements , unless scaling is selected through VkSwapchainPresentScalingCreateInfoEXT, if supported . |

- `imageArrayLayers` is the number of views in a multiview/stereo surface. For non-stereoscopic-3D applications, this value is 1.
- `imageUsage` is a bitmask of VkImageUsageFlagBits describing the intended usage of the (acquired) swapchain images.
- `imageSharingMode` is the sharing mode used for the image(s) of the swapchain.
- `queueFamilyIndexCount` is the number of queue families having access to the image(s) of the swapchain when `imageSharingMode` is `VK_SHARING_MODE_CONCURRENT`.
- `pQueueFamilyIndices` is a pointer to an array of queue family indices having access to the images(s) of the swapchain when `imageSharingMode` is `VK_SHARING_MODE_CONCURRENT`.
- `preTransform` is a VkSurfaceTransformFlagBitsKHR value describing the transform, relative to the presentation engine's natural orientation, applied to the image content prior to presentation. If it does not match the `currentTransform` value returned by `vkGetPhysicalDeviceSurfaceCapabilitiesKHR`, the presentation engine will transform the image content as part of the presentation operation.
- `compositeAlpha` is a VkCompositeAlphaFlagBitsKHR value indicating the alpha compositing mode to use when this surface is composited together with other surfaces on certain window systems.
- `presentMode` is the presentation mode the swapchain will use. A swapchain's present mode determines how incoming present requests will be processed and queued internally.
- `clipped` specifies whether the Vulkan implementation is allowed to discard rendering operations that affect regions of the surface that are not visible.
  - If set to `VK_TRUE`, the presentable images associated with the swapchain **may** not own all of their pixels. Pixels in the presentable images that correspond to regions of the target surface obscured by another window on the desktop, or subject to some other clipping mechanism will have undefined content when read back. Fragment shaders **may** not execute for these pixels, and thus any side effects they would have had will not occur. Setting `VK_TRUE` does not

guarantee any clipping will occur, but allows more efficient presentation methods to be used on some platforms.

- If set to VK_FALSE, presentable images associated with the swapchain will own all of the pixels they contain.

**NOTE** Applications **should** set this value to VK_TRUE if they do not expect to read back the content of presentable images before presenting them or after reacquiring them, and if their fragment shaders do not have any side effects that require them to run for all pixels in the presentable image.

- oldSwapchain is VK_NULL_HANDLE, or the existing non-retired swapchain currently associated with surface. Providing a valid oldSwapchain **may** aid in the resource reuse, and also allows the application to still present any images that are already acquired from it.

Upon calling vkCreateSwapchainKHR with an oldSwapchain that is not VK_NULL_HANDLE, oldSwapchain is retired — even if creation of the new swapchain fails. The new swapchain is created in the non-retired state whether or not oldSwapchain is VK_NULL_HANDLE.

Upon calling vkCreateSwapchainKHR with an oldSwapchain that is not VK_NULL_HANDLE, any images from oldSwapchain that are not acquired by the application **may** be freed by the implementation, which **may** occur even if creation of the new swapchain fails. The application **can** destroy oldSwapchain to free all memory associated with oldSwapchain.

**NOTE** Multiple retired swapchains **can** be associated with the same VkSurfaceKHR through multiple uses of oldSwapchain that outnumber calls to vkDestroySwapchainKHR.

After oldSwapchain is retired, the application **can** pass to vkQueuePresentKHR any images it had already acquired from oldSwapchain. E.g., an application may present an image from the old swapchain before an image from the new swapchain is ready to be presented. As usual, vkQueuePresentKHR **may** fail if oldSwapchain has entered a state that causes VK_ERROR_OUT_OF_DATE_KHR to be returned.

The application **can** continue to use a shared presentable image obtained from oldSwapchain until a presentable image is acquired from the new swapchain, as long as it has not entered a state that causes it to return VK_ERROR_OUT_OF_DATE_KHR.

## Valid Usage

- VUID-VkSwapchainCreateInfoKHR-surface-01270
  surface **must** be a surface that is supported by the device as determined using vkGetPhysicalDeviceSurfaceSupportKHR
- VUID-VkSwapchainCreateInfoKHR-minImageCount-01272
  minImageCount **must** be less than or equal to the value returned in the maxImageCount member of the VkSurfaceCapabilitiesKHR structure returned by vkGetPhysicalDeviceSurfaceCapabilitiesKHR for the surface if the returned maxImageCount is not zero
- VUID-VkSwapchainCreateInfoKHR-presentMode-02839

If presentMode is not VK_PRESENT_MODE_SHARED_DEMAND_REFRESH_KHR nor VK_PRESENT_MODE_SHARED_CONTINUOUS_REFRESH_KHR, then minImageCount **must** be greater than or equal to the value returned in the minImageCount member of the VkSurfaceCapabilitiesKHR structure returned by vkGetPhysicalDeviceSurfaceCapabilitiesKHR for the surface

- VUID-VkSwapchainCreateInfoKHR-minImageCount-01383
  minImageCount **must** be 1 if presentMode is either VK_PRESENT_MODE_SHARED_DEMAND_REFRESH_KHR or VK_PRESENT_MODE_SHARED_CONTINUOUS_REFRESH_KHR

- VUID-VkSwapchainCreateInfoKHR-imageFormat-01273
  imageFormat and imageColorSpace **must** match the format and colorSpace members, respectively, of one of the VkSurfaceFormatKHR structures returned by vkGetPhysicalDeviceSurfaceFormatsKHR for the surface

- VUID-VkSwapchainCreateInfoKHR-pNext-07781
  If a VkSwapchainPresentScalingCreateInfoEXT structure was not included in the pNext chain, or it is included and VkSwapchainPresentScalingCreateInfoEXT::scalingBehavior is zero then imageExtent **must** be between minImageExtent and maxImageExtent, inclusive, where minImageExtent and maxImageExtent are members of the VkSurfaceCapabilitiesKHR structure returned by vkGetPhysicalDeviceSurfaceCapabilitiesKHR for the surface

- VUID-VkSwapchainCreateInfoKHR-pNext-07782
  If a VkSwapchainPresentScalingCreateInfoEXT structure was included in the pNext chain and VkSwapchainPresentScalingCreateInfoEXT::scalingBehavior is not zero then imageExtent **must** be between minScaledImageExtent and maxScaledImageExtent, inclusive, where minScaledImageExtent and maxScaledImageExtent are members of the VkSurfacePresentScalingCapabilitiesEXT structure returned by vkGetPhysicalDeviceSurfaceCapabilities2KHR for the surface and presentMode

- VUID-VkSwapchainCreateInfoKHR-imageExtent-01689
  imageExtent members width and height **must** both be non-zero

- VUID-VkSwapchainCreateInfoKHR-imageArrayLayers-01275
  imageArrayLayers **must** be greater than 0 and less than or equal to the maxImageArrayLayers member of the VkSurfaceCapabilitiesKHR structure returned by vkGetPhysicalDeviceSurfaceCapabilitiesKHR for the surface

- VUID-VkSwapchainCreateInfoKHR-presentMode-01427
  If presentMode is VK_PRESENT_MODE_IMMEDIATE_KHR, VK_PRESENT_MODE_MAILBOX_KHR, VK_PRESENT_MODE_FIFO_KHR or VK_PRESENT_MODE_FIFO_RELAXED_KHR, imageUsage **must** be a subset of the supported usage flags present in the supportedUsageFlags member of the VkSurfaceCapabilitiesKHR structure returned by vkGetPhysicalDeviceSurfaceCapabilitiesKHR for surface

- VUID-VkSwapchainCreateInfoKHR-imageUsage-01384
  If presentMode is VK_PRESENT_MODE_SHARED_DEMAND_REFRESH_KHR or VK_PRESENT_MODE_SHARED_CONTINUOUS_REFRESH_KHR, imageUsage **must** be a subset of the supported usage flags present in the sharedPresentSupportedUsageFlags member of the VkSharedPresentSurfaceCapabilitiesKHR structure returned by vkGetPhysicalDeviceSurfaceCapabilities2KHR for surface

- VUID-VkSwapchainCreateInfoKHR-imageSharingMode-01277
  If `imageSharingMode` is `VK_SHARING_MODE_CONCURRENT`, `pQueueFamilyIndices` **must** be a valid pointer to an array of `queueFamilyIndexCount` `uint32_t` values
- VUID-VkSwapchainCreateInfoKHR-imageSharingMode-01278
  If `imageSharingMode` is `VK_SHARING_MODE_CONCURRENT`, `queueFamilyIndexCount` **must** be greater than `1`
- VUID-VkSwapchainCreateInfoKHR-imageSharingMode-01428
  If `imageSharingMode` is `VK_SHARING_MODE_CONCURRENT`, each element of `pQueueFamilyIndices` **must** be unique and **must** be less than `pQueueFamilyPropertyCount` returned by either vkGetPhysicalDeviceQueueFamilyProperties or vkGetPhysicalDeviceQueueFamilyProperties2 for the `physicalDevice` that was used to create `device`
- VUID-VkSwapchainCreateInfoKHR-preTransform-01279
  `preTransform` **must** be one of the bits present in the `supportedTransforms` member of the `VkSurfaceCapabilitiesKHR` structure returned by `vkGetPhysicalDeviceSurfaceCapabilitiesKHR` for the surface
- VUID-VkSwapchainCreateInfoKHR-compositeAlpha-01280
  `compositeAlpha` **must** be one of the bits present in the `supportedCompositeAlpha` member of the `VkSurfaceCapabilitiesKHR` structure returned by `vkGetPhysicalDeviceSurfaceCapabilitiesKHR` for the surface
- VUID-VkSwapchainCreateInfoKHR-presentMode-01281
  `presentMode` **must** be one of the VkPresentModeKHR values returned by `vkGetPhysicalDeviceSurfacePresentModesKHR` for the surface
- VUID-VkSwapchainCreateInfoKHR-physicalDeviceCount-01429
  If the logical device was created with VkDeviceGroupDeviceCreateInfo::`physicalDeviceCount` equal to 1, `flags` **must** not contain `VK_SWAPCHAIN_CREATE_SPLIT_INSTANCE_BIND_REGIONS_BIT_KHR`
- VUID-VkSwapchainCreateInfoKHR-oldSwapchain-01933
  If `oldSwapchain` is not VK_NULL_HANDLE, `oldSwapchain` **must** be a non-retired swapchain associated with native window referred to by `surface`
- VUID-VkSwapchainCreateInfoKHR-imageFormat-01778
  The implied image creation parameters of the swapchain **must** be supported as reported by vkGetPhysicalDeviceImageFormatProperties
- VUID-VkSwapchainCreateInfoKHR-flags-03168
  If `flags` contains `VK_SWAPCHAIN_CREATE_MUTABLE_FORMAT_BIT_KHR` then the `pNext` chain **must** include a VkImageFormatListCreateInfo structure with a `viewFormatCount` greater than zero and `pViewFormats` **must** have an element equal to `imageFormat`
- VUID-VkSwapchainCreateInfoKHR-pNext-04099
  If a VkImageFormatListCreateInfo structure was included in the `pNext` chain and VkImageFormatListCreateInfo::`viewFormatCount` is not zero then all of the formats in VkImageFormatListCreateInfo::`pViewFormats` **must** be compatible with the `format` as described in the compatibility table
- VUID-VkSwapchainCreateInfoKHR-flags-04100

If flags does not contain VK_SWAPCHAIN_CREATE_MUTABLE_FORMAT_BIT_KHR and the pNext chain include a VkImageFormatListCreateInfo structure then VkImageFormatListCreateInfo::viewFormatCount **must** be 0 or 1

- VUID-VkSwapchainCreateInfoKHR-flags-03187
  If flags contains VK_SWAPCHAIN_CREATE_PROTECTED_BIT_KHR, then VkSurfaceProtectedCapabilitiesKHR::supportsProtected **must** be VK_TRUE in the VkSurfaceProtectedCapabilitiesKHR structure returned by vkGetPhysicalDeviceSurfaceCapabilities2KHR for surface
- VUID-VkSwapchainCreateInfoKHR-pNext-02679
  If the pNext chain includes a VkSurfaceFullScreenExclusiveInfoEXT structure with its fullScreenExclusive member set to VK_FULL_SCREEN_EXCLUSIVE_APPLICATION_CONTROLLED_EXT, and surface was created using vkCreateWin32SurfaceKHR, a VkSurfaceFullScreenExclusiveWin32InfoEXT structure **must** be included in the pNext chain
- VUID-VkSwapchainCreateInfoKHR-pNext-06752
  If the imageCompressionControlSwapchain feature is not enabled, the pNext chain **must** not include an VkImageCompressionControlEXT structure

## Valid Usage (Implicit)

- VUID-VkSwapchainCreateInfoKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_SWAPCHAIN_CREATE_INFO_KHR
- VUID-VkSwapchainCreateInfoKHR-pNext-pNext
  Each pNext member of any structure (including this one) in the pNext chain **must** be either NULL or a pointer to a valid instance of VkDeviceGroupSwapchainCreateInfoKHR, VkImageCompressionControlEXT, VkImageFormatListCreateInfo, VkSurfaceFullScreenExclusiveInfoEXT, VkSurfaceFullScreenExclusiveWin32InfoEXT, VkSwapchainCounterCreateInfoEXT, VkSwapchainDisplayNativeHdrCreateInfoAMD, VkSwapchainLatencyCreateInfoNV, VkSwapchainPresentBarrierCreateInfoNV, VkSwapchainPresentModesCreateInfoEXT, or VkSwapchainPresentScalingCreateInfoEXT
- VUID-VkSwapchainCreateInfoKHR-sType-unique
  The sType value of each struct in the pNext chain **must** be unique
- VUID-VkSwapchainCreateInfoKHR-flags-parameter
  flags **must** be a valid combination of VkSwapchainCreateFlagBitsKHR values
- VUID-VkSwapchainCreateInfoKHR-surface-parameter
  surface **must** be a valid VkSurfaceKHR handle
- VUID-VkSwapchainCreateInfoKHR-imageFormat-parameter
  imageFormat **must** be a valid VkFormat value
- VUID-VkSwapchainCreateInfoKHR-imageColorSpace-parameter
  imageColorSpace **must** be a valid VkColorSpaceKHR value
- VUID-VkSwapchainCreateInfoKHR-imageUsage-parameter
  imageUsage **must** be a valid combination of VkImageUsageFlagBits values

- VUID-VkSwapchainCreateInfoKHR-imageUsage-requiredbitmask
  `imageUsage` **must** not be `0`
- VUID-VkSwapchainCreateInfoKHR-imageSharingMode-parameter
  `imageSharingMode` **must** be a valid VkSharingMode value
- VUID-VkSwapchainCreateInfoKHR-preTransform-parameter
  `preTransform` **must** be a valid VkSurfaceTransformFlagBitsKHR value
- VUID-VkSwapchainCreateInfoKHR-compositeAlpha-parameter
  `compositeAlpha` **must** be a valid VkCompositeAlphaFlagBitsKHR value
- VUID-VkSwapchainCreateInfoKHR-presentMode-parameter
  `presentMode` **must** be a valid VkPresentModeKHR value
- VUID-VkSwapchainCreateInfoKHR-oldSwapchain-parameter
  If `oldSwapchain` is not VK_NULL_HANDLE, `oldSwapchain` **must** be a valid VkSwapchainKHR handle
- VUID-VkSwapchainCreateInfoKHR-commonparent
  Both of `oldSwapchain`, and `surface` that are valid handles of non-ignored parameters **must** have been created, allocated, or retrieved from the same VkInstance

Bits which **can** be set in VkSwapchainCreateInfoKHR::`flags`, specifying parameters of swapchain creation, are:

```
// Provided by VK_KHR_swapchain
typedef enum VkSwapchainCreateFlagBitsKHR {
  // Provided by VK_VERSION_1_1 with VK_KHR_swapchain, VK_KHR_device_group with
VK_KHR_swapchain
    VK_SWAPCHAIN_CREATE_SPLIT_INSTANCE_BIND_REGIONS_BIT_KHR = 0x00000001,
  // Provided by VK_VERSION_1_1 with VK_KHR_swapchain
    VK_SWAPCHAIN_CREATE_PROTECTED_BIT_KHR = 0x00000002,
  // Provided by VK_KHR_swapchain_mutable_format
    VK_SWAPCHAIN_CREATE_MUTABLE_FORMAT_BIT_KHR = 0x00000004,
  // Provided by VK_EXT_swapchain_maintenance1
    VK_SWAPCHAIN_CREATE_DEFERRED_MEMORY_ALLOCATION_BIT_EXT = 0x00000008,
} VkSwapchainCreateFlagBitsKHR;
```

- `VK_SWAPCHAIN_CREATE_SPLIT_INSTANCE_BIND_REGIONS_BIT_KHR` specifies that images created from the swapchain (i.e. with the `swapchain` member of VkImageSwapchainCreateInfoKHR set to this swapchain's handle) **must** use `VK_IMAGE_CREATE_SPLIT_INSTANCE_BIND_REGIONS_BIT`.
- `VK_SWAPCHAIN_CREATE_PROTECTED_BIT_KHR` specifies that images created from the swapchain are protected images.
- `VK_SWAPCHAIN_CREATE_MUTABLE_FORMAT_BIT_KHR` specifies that the images of the swapchain **can** be used to create a `VkImageView` with a different format than what the swapchain was created with. The list of allowed image view formats is specified by adding a VkImageFormatListCreateInfo structure to the `pNext` chain of VkSwapchainCreateInfoKHR. In addition, this flag also specifies that the swapchain **can** be created with usage flags that are not supported for the format the swapchain is created with but are supported for at least one of the allowed image view formats.

- VK_SWAPCHAIN_CREATE_DEFERRED_MEMORY_ALLOCATION_BIT_EXT specifies that the implementation **may** defer allocation of memory associated with each swapchain image until its index is to be returned from vkAcquireNextImageKHR or vkAcquireNextImage2KHR for the first time.

```
// Provided by VK_KHR_swapchain
typedef VkFlags VkSwapchainCreateFlagsKHR;
```

VkSwapchainCreateFlagsKHR is a bitmask type for setting a mask of zero or more VkSwapchainCreateFlagBitsKHR.

If the pNext chain of VkSwapchainCreateInfoKHR includes a VkDeviceGroupSwapchainCreateInfoKHR structure, then that structure includes a set of device group present modes that the swapchain **can** be used with.

The VkDeviceGroupSwapchainCreateInfoKHR structure is defined as:

```
// Provided by VK_VERSION_1_1 with VK_KHR_swapchain, VK_KHR_device_group with
VK_KHR_swapchain
typedef struct VkDeviceGroupSwapchainCreateInfoKHR {
    VkStructureType                     sType;
    const void*                         pNext;
    VkDeviceGroupPresentModeFlagsKHR    modes;
} VkDeviceGroupSwapchainCreateInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- modes is a bitfield of modes that the swapchain **can** be used with.

If this structure is not present, modes is considered to be VK_DEVICE_GROUP_PRESENT_MODE_LOCAL_BIT_KHR.

### Valid Usage (Implicit)

- VUID-VkDeviceGroupSwapchainCreateInfoKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_DEVICE_GROUP_SWAPCHAIN_CREATE_INFO_KHR
- VUID-VkDeviceGroupSwapchainCreateInfoKHR-modes-parameter
  modes **must** be a valid combination of VkDeviceGroupPresentModeFlagBitsKHR values
- VUID-VkDeviceGroupSwapchainCreateInfoKHR-modes-requiredbitmask
  modes **must** not be 0

If the pNext chain of VkSwapchainCreateInfoKHR includes a VkSwapchainDisplayNativeHdrCreateInfoAMD structure, then that structure includes additional swapchain creation parameters specific to display native HDR support.

The VkSwapchainDisplayNativeHdrCreateInfoAMD structure is defined as:

```
// Provided by VK_AMD_display_native_hdr
typedef struct VkSwapchainDisplayNativeHdrCreateInfoAMD {
    VkStructureType    sType;
    const void*        pNext;
    VkBool32           localDimmingEnable;
} VkSwapchainDisplayNativeHdrCreateInfoAMD;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `localDimmingEnable` specifies whether local dimming is enabled for the swapchain.

If the `pNext` chain of VkSwapchainCreateInfoKHR does not include this structure, the default value for `localDimmingEnable` is `VK_TRUE`, meaning local dimming is initially enabled for the swapchain.

### Valid Usage (Implicit)

- VUID-VkSwapchainDisplayNativeHdrCreateInfoAMD-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_SWAPCHAIN_DISPLAY_NATIVE_HDR_CREATE_INFO_AMD`

### Valid Usage

- VUID-VkSwapchainDisplayNativeHdrCreateInfoAMD-localDimmingEnable-04449
  It is only valid to set `localDimmingEnable` to `VK_TRUE` if VkDisplayNativeHdrSurfaceCapabilitiesAMD::`localDimmingSupport` is supported

The local dimming HDR setting may also be changed over the life of a swapchain by calling:

```
// Provided by VK_AMD_display_native_hdr
void vkSetLocalDimmingAMD(
    VkDevice                                    device,
    VkSwapchainKHR                              swapChain,
    VkBool32                                    localDimmingEnable);
```

- `device` is the device associated with `swapChain`.
- `swapChain` handle to enable local dimming.
- `localDimmingEnable` specifies whether local dimming is enabled for the swapchain.

### Valid Usage (Implicit)

- VUID-vkSetLocalDimmingAMD-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkSetLocalDimmingAMD-swapChain-parameter

swapChain **must** be a valid VkSwapchainKHR handle

- VUID-vkSetLocalDimmingAMD-swapChain-parent
  swapChain **must** have been created, allocated, or retrieved from device

### Valid Usage

- VUID-vkSetLocalDimmingAMD-localDimmingSupport-04618
  VkDisplayNativeHdrSurfaceCapabilitiesAMD::localDimmingSupport **must** be supported

If the pNext chain of VkSwapchainCreateInfoKHR includes a VkSurfaceFullScreenExclusiveInfoEXT structure, then that structure specifies the application's preferred full-screen presentation behavior. If this structure is not present, fullScreenExclusive is considered to be VK_FULL_SCREEN_EXCLUSIVE_DEFAULT_EXT.

To enable surface counters when creating a swapchain, add a VkSwapchainCounterCreateInfoEXT structure to the pNext chain of VkSwapchainCreateInfoKHR. VkSwapchainCounterCreateInfoEXT is defined as:

```
// Provided by VK_EXT_display_control
typedef struct VkSwapchainCounterCreateInfoEXT {
    VkStructureType             sType;
    const void*                 pNext;
    VkSurfaceCounterFlagsEXT    surfaceCounters;
} VkSwapchainCounterCreateInfoEXT;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- surfaceCounters is a bitmask of VkSurfaceCounterFlagBitsEXT specifying surface counters to enable for the swapchain.

### Valid Usage

- VUID-VkSwapchainCounterCreateInfoEXT-surfaceCounters-01244
  The bits in surfaceCounters **must** be supported by VkSwapchainCreateInfoKHR::surface, as reported by vkGetPhysicalDeviceSurfaceCapabilities2EXT

### Valid Usage (Implicit)

- VUID-VkSwapchainCounterCreateInfoEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_SWAPCHAIN_COUNTER_CREATE_INFO_EXT
- VUID-VkSwapchainCounterCreateInfoEXT-surfaceCounters-parameter
  surfaceCounters **must** be a valid combination of VkSurfaceCounterFlagBitsEXT values

The requested counters become active when the first presentation command for the associated swapchain is processed by the presentation engine. To query the value of an active counter, use:

```
// Provided by VK_EXT_display_control
VkResult vkGetSwapchainCounterEXT(
    VkDevice                                    device,
    VkSwapchainKHR                              swapchain,
    VkSurfaceCounterFlagBitsEXT                 counter,
    uint64_t*                                   pCounterValue);
```

- `device` is the VkDevice associated with `swapchain`.
- `swapchain` is the swapchain from which to query the counter value.
- `counter` is a VkSurfaceCounterFlagBitsEXT value specifying the counter to query.
- `pCounterValue` will return the current value of the counter.

If a counter is not available because the swapchain is out of date, the implementation **may** return `VK_ERROR_OUT_OF_DATE_KHR`.

## Valid Usage

- VUID-vkGetSwapchainCounterEXT-swapchain-01245
  One or more present commands on `swapchain` **must** have been processed by the presentation engine

## Valid Usage (Implicit)

- VUID-vkGetSwapchainCounterEXT-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkGetSwapchainCounterEXT-swapchain-parameter
  `swapchain` **must** be a valid VkSwapchainKHR handle
- VUID-vkGetSwapchainCounterEXT-counter-parameter
  `counter` **must** be a valid VkSurfaceCounterFlagBitsEXT value
- VUID-vkGetSwapchainCounterEXT-pCounterValue-parameter
  `pCounterValue` **must** be a valid pointer to a `uint64_t` value
- VUID-vkGetSwapchainCounterEXT-swapchain-parent
  `swapchain` **must** have been created, allocated, or retrieved from `device`

## Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_DEVICE_LOST
- VK_ERROR_OUT_OF_DATE_KHR

To specify compression properties for the swapchain images in this swapchain, add a VkImageCompressionControlEXT structure to the pNext chain of the VkSwapchainCreateInfoKHR structure.

Applications **can** modify the presentation mode used by the swapchain on a per-presentation basis. However, all presentation modes the application intends to use with the swapchain **must** be specified at swapchain creation time. To specify more than one presentation mode when creating a swapchain, include the VkSwapchainPresentModesCreateInfoEXT structure in the pNext chain of the VkSwapchainCreateInfoKHR structure.

The VkSwapchainPresentModesCreateInfoEXT structure is defined as:

```
// Provided by VK_EXT_swapchain_maintenance1
typedef struct VkSwapchainPresentModesCreateInfoEXT {
    VkStructureType            sType;
    const void*                pNext;
    uint32_t                   presentModeCount;
    const VkPresentModeKHR*    pPresentModes;
} VkSwapchainPresentModesCreateInfoEXT;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- presentModeCount is the number of presentation modes provided.
- pPresentModes is a list of presentation modes with presentModeCount entries

## Valid Usage

- VUID-VkSwapchainPresentModesCreateInfoEXT-None-07762
  Each entry in pPresentModes **must** be one of the VkPresentModeKHR values returned by vkGetPhysicalDeviceSurfacePresentModesKHR for the surface
- VUID-VkSwapchainPresentModesCreateInfoEXT-pPresentModes-07763
  The entries in pPresentModes **must** be a subset of the present modes returned in VkSurfacePresentModeCompatibilityEXT::pPresentModes, given VkSwapchainCreateInfoKHR::presentMode in VkSurfacePresentModeEXT
- VUID-VkSwapchainPresentModesCreateInfoEXT-presentMode-07764
  VkSwapchainCreateInfoKHR::presentMode **must** be included in pPresentModes

### Valid Usage (Implicit)

- VUID-VkSwapchainPresentModesCreateInfoEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_SWAPCHAIN_PRESENT_MODES_CREATE_INFO_EXT`
- VUID-VkSwapchainPresentModesCreateInfoEXT-pPresentModes-parameter
  `pPresentModes` **must** be a valid pointer to an array of `presentModeCount` valid VkPresentModeKHR values
- VUID-VkSwapchainPresentModesCreateInfoEXT-presentModeCount-arraylength
  `presentModeCount` **must** be greater than `0`

When an application presents a swapchain image with dimensions different than those of the target surface, different behavior is possible on different platforms per their respective specifications:

- Presentation fails and `VK_ERROR_OUT_OF_DATE_KHR` is returned
- Scaling is done and `VK_SUCCESS` or `VK_SUBOPTIMAL_KHR` is returned
- Unspecified scaling using an arbitrary combination of stretching, centering and/or clipping.

Applications **can** define specific behavior when creating a swapchain by including the `VkSwapchainPresentScalingCreateInfoEXT` structure in the `pNext` chain of the VkSwapchainCreateInfoKHR structure.

The `VkSwapchainPresentScalingCreateInfoEXT` structure is defined as:

```
// Provided by VK_EXT_swapchain_maintenance1
typedef struct VkSwapchainPresentScalingCreateInfoEXT {
    VkStructureType             sType;
    const void*                 pNext;
    VkPresentScalingFlagsEXT    scalingBehavior;
    VkPresentGravityFlagsEXT    presentGravityX;
    VkPresentGravityFlagsEXT    presentGravityY;
} VkSwapchainPresentScalingCreateInfoEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `scalingBehavior` is `0` or the scaling method to use when the dimensions of the surface and swapchain images differ.
- `presentGravityX` is `0` or the x-axis direction in which swapchain image pixels gravitate relative to the surface when `scalingBehavior` does not result in a one-to-one pixel mapping between the scaled swapchain image and the surface.
- `presentGravityY` is `0` or the y-axis direction in which swapchain image pixels gravitate relative to the surface when `scalingBehavior` does not result in a one-to-one pixel mapping between the scaled swapchain image and the surface.

If `scalingBehavior` is `0`, the result of presenting a swapchain image with dimensions that do not match the surface dimensions is implementation and platform-dependent. If `presentGravityX` or `presentGravityY` are `0`, the presentation gravity **must** match that defined by the native platform surface on platforms which define surface gravity.

### Valid Usage

- VUID-VkSwapchainPresentScalingCreateInfoEXT-presentGravityX-07765
  If `presentGravityX` is `0`, `presentGravityY` **must** be `0`
- VUID-VkSwapchainPresentScalingCreateInfoEXT-presentGravityX-07766
  If `presentGravityX` is not `0`, `presentGravityY` **must** not be `0`
- VUID-VkSwapchainPresentScalingCreateInfoEXT-scalingBehavior-07767
  `scalingBehavior` **must** not have more than one bit set
- VUID-VkSwapchainPresentScalingCreateInfoEXT-presentGravityX-07768
  `presentGravityX` **must** not have more than one bit set
- VUID-VkSwapchainPresentScalingCreateInfoEXT-presentGravityY-07769
  `presentGravityY` **must** not have more than one bit set
- VUID-VkSwapchainPresentScalingCreateInfoEXT-scalingBehavior-07770
  `scalingBehavior` **must** be a valid scaling method for the surface as returned in VkSurfacePresentScalingCapabilitiesEXT::`supportedPresentScaling`, given VkSwapchainCreateInfoKHR::`presentMode` in VkSurfacePresentModeEXT
- VUID-VkSwapchainPresentScalingCreateInfoEXT-scalingBehavior-07771
  If the swapchain is created with VkSwapchainPresentModesCreateInfoEXT, `scalingBehavior` **must** be a valid scaling method for the surface as returned in VkSurfacePresentScalingCapabilitiesEXT::`supportedPresentScaling`, given each present mode in VkSwapchainPresentModesCreateInfoEXT::`pPresentModes` in VkSurfacePresentModeEXT
- VUID-VkSwapchainPresentScalingCreateInfoEXT-presentGravityX-07772
  `presentGravityX` **must** be a valid x-axis present gravity for the surface as returned in VkSurfacePresentScalingCapabilitiesEXT::`supportedPresentGravityX`, given VkSwapchainCreateInfoKHR::`presentMode` in VkSurfacePresentModeEXT
- VUID-VkSwapchainPresentScalingCreateInfoEXT-presentGravityX-07773
  If the swapchain is created with VkSwapchainPresentModesCreateInfoEXT, `presentGravityX` **must** be a valid x-axis present gravity for the surface as returned in VkSurfacePresentScalingCapabilitiesEXT::`supportedPresentGravityX`, given each present mode in VkSwapchainPresentModesCreateInfoEXT::`pPresentModes` in VkSurfacePresentModeEXT
- VUID-VkSwapchainPresentScalingCreateInfoEXT-presentGravityY-07774
  `presentGravityY` **must** be a valid y-axis present gravity for the surface as returned in VkSurfacePresentScalingCapabilitiesEXT::`supportedPresentGravityY`, given VkSwapchainCreateInfoKHR::`presentMode` in VkSurfacePresentModeEXT
- VUID-VkSwapchainPresentScalingCreateInfoEXT-presentGravityY-07775
  If the swapchain is created with VkSwapchainPresentModesCreateInfoEXT,

presentGravityY **must** be a valid y-axis present gravity for the surface as returned in VkSurfacePresentScalingCapabilitiesEXT::supportedPresentGravityY, given each present mode in VkSwapchainPresentModesCreateInfoEXT::pPresentModes in VkSurfacePresentModeEXT

### Valid Usage (Implicit)

- VUID-VkSwapchainPresentScalingCreateInfoEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_SWAPCHAIN_PRESENT_SCALING_CREATE_INFO_EXT
- VUID-VkSwapchainPresentScalingCreateInfoEXT-scalingBehavior-parameter
  scalingBehavior **must** be a valid combination of VkPresentScalingFlagBitsEXT values
- VUID-VkSwapchainPresentScalingCreateInfoEXT-presentGravityX-parameter
  presentGravityX **must** be a valid combination of VkPresentGravityFlagBitsEXT values
- VUID-VkSwapchainPresentScalingCreateInfoEXT-presentGravityY-parameter
  presentGravityY **must** be a valid combination of VkPresentGravityFlagBitsEXT values

To destroy a swapchain object call:

```
// Provided by VK_KHR_swapchain
void vkDestroySwapchainKHR(
    VkDevice                                    device,
    VkSwapchainKHR                              swapchain,
    const VkAllocationCallbacks*                pAllocator);
```

- device is the VkDevice associated with swapchain.
- swapchain is the swapchain to destroy.
- pAllocator is the allocator used for host memory allocated for the swapchain object when there is no more specific allocator available (see Memory Allocation).

The application **must** not destroy a swapchain until after completion of all outstanding operations on images that were acquired from the swapchain. swapchain and all associated VkImage handles are destroyed, and **must** not be acquired or used any more by the application. The memory of each VkImage will only be freed after that image is no longer used by the presentation engine. For example, if one image of the swapchain is being displayed in a window, the memory for that image **may** not be freed until the window is destroyed, or another swapchain is created for the window. Destroying the swapchain does not invalidate the parent VkSurfaceKHR, and a new swapchain **can** be created with it.

When a swapchain associated with a display surface is destroyed, if the image most recently presented to the display surface is from the swapchain being destroyed, then either any display resources modified by presenting images from any swapchain associated with the display surface **must** be reverted by the implementation to their state prior to the first present performed on one of these swapchains, or such resources **must** be left in their current state.

If swapchain has exclusive full-screen access, it is released before the swapchain is destroyed.

### Valid Usage

- VUID-vkDestroySwapchainKHR-swapchain-01282
  All uses of presentable images acquired from swapchain **must** have completed execution
- VUID-vkDestroySwapchainKHR-swapchain-01283
  If VkAllocationCallbacks were provided when swapchain was created, a compatible set of callbacks **must** be provided here
- VUID-vkDestroySwapchainKHR-swapchain-01284
  If no VkAllocationCallbacks were provided when swapchain was created, pAllocator **must** be NULL

### Valid Usage (Implicit)

- VUID-vkDestroySwapchainKHR-device-parameter
  device **must** be a valid VkDevice handle
- VUID-vkDestroySwapchainKHR-swapchain-parameter
  If swapchain is not VK_NULL_HANDLE, swapchain **must** be a valid VkSwapchainKHR handle
- VUID-vkDestroySwapchainKHR-pAllocator-parameter
  If pAllocator is not NULL, pAllocator **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkDestroySwapchainKHR-swapchain-parent
  If swapchain is a valid handle, it **must** have been created, allocated, or retrieved from device

### Host Synchronization

- Host access to swapchain **must** be externally synchronized

When the VK_KHR_display_swapchain extension is enabled, multiple swapchains that share presentable images are created by calling:

```
// Provided by VK_KHR_display_swapchain
VkResult vkCreateSharedSwapchainsKHR(
    VkDevice                                    device,
    uint32_t                                    swapchainCount,
    const VkSwapchainCreateInfoKHR*             pCreateInfos,
    const VkAllocationCallbacks*                pAllocator,
    VkSwapchainKHR*                             pSwapchains);
```

- `device` is the device to create the swapchains for.
- `swapchainCount` is the number of swapchains to create.
- `pCreateInfos` is a pointer to an array of VkSwapchainCreateInfoKHR structures specifying the parameters of the created swapchains.
- `pAllocator` is the allocator used for host memory allocated for the swapchain objects when there is no more specific allocator available (see Memory Allocation).
- `pSwapchains` is a pointer to an array of VkSwapchainKHR handles in which the created swapchain objects will be returned.

`vkCreateSharedSwapchainsKHR` is similar to vkCreateSwapchainKHR, except that it takes an array of VkSwapchainCreateInfoKHR structures, and returns an array of swapchain objects.

The swapchain creation parameters that affect the properties and number of presentable images **must** match between all the swapchains. If the displays used by any of the swapchains do not use the same presentable image layout or are incompatible in a way that prevents sharing images, swapchain creation will fail with the result code `VK_ERROR_INCOMPATIBLE_DISPLAY_KHR`. If any error occurs, no swapchains will be created. Images presented to multiple swapchains **must** be re-acquired from all of them before being modified. After destroying one or more of the swapchains, the remaining swapchains and the presentable images **can** continue to be used.

## Valid Usage (Implicit)

- VUID-vkCreateSharedSwapchainsKHR-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkCreateSharedSwapchainsKHR-pCreateInfos-parameter
  `pCreateInfos` **must** be a valid pointer to an array of `swapchainCount` valid VkSwapchainCreateInfoKHR structures
- VUID-vkCreateSharedSwapchainsKHR-pAllocator-parameter
  If `pAllocator` is not `NULL`, `pAllocator` **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkCreateSharedSwapchainsKHR-pSwapchains-parameter
  `pSwapchains` **must** be a valid pointer to an array of `swapchainCount` VkSwapchainKHR handles
- VUID-vkCreateSharedSwapchainsKHR-swapchainCount-arraylength
  `swapchainCount` **must** be greater than `0`

## Host Synchronization

- Host access to `pCreateInfos`[].surface **must** be externally synchronized
- Host access to `pCreateInfos`[].oldSwapchain **must** be externally synchronized

### Return Codes

**Success**

- VK_SUCCESS

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY
- VK_ERROR_INCOMPATIBLE_DISPLAY_KHR
- VK_ERROR_DEVICE_LOST
- VK_ERROR_SURFACE_LOST_KHR

To obtain the array of presentable images associated with a swapchain, call:

```
// Provided by VK_KHR_swapchain
VkResult vkGetSwapchainImagesKHR(
    VkDevice                                    device,
    VkSwapchainKHR                              swapchain,
    uint32_t*                                   pSwapchainImageCount,
    VkImage*                                    pSwapchainImages);
```

- device is the device associated with swapchain.
- swapchain is the swapchain to query.
- pSwapchainImageCount is a pointer to an integer related to the number of presentable images available or queried, as described below.
- pSwapchainImages is either NULL or a pointer to an array of VkImage handles.

If pSwapchainImages is NULL, then the number of presentable images for swapchain is returned in pSwapchainImageCount. Otherwise, pSwapchainImageCount **must** point to a variable set by the application to the number of elements in the pSwapchainImages array, and on return the variable is overwritten with the number of structures actually written to pSwapchainImages. If the value of pSwapchainImageCount is less than the number of presentable images for swapchain, at most pSwapchainImageCount structures will be written, and VK_INCOMPLETE will be returned instead of VK_SUCCESS, to indicate that not all the available presentable images were returned.

### Valid Usage (Implicit)

- VUID-vkGetSwapchainImagesKHR-device-parameter
  device **must** be a valid VkDevice handle
- VUID-vkGetSwapchainImagesKHR-swapchain-parameter
  swapchain **must** be a valid VkSwapchainKHR handle
- VUID-vkGetSwapchainImagesKHR-pSwapchainImageCount-parameter
  pSwapchainImageCount **must** be a valid pointer to a uint32_t value

- VUID-vkGetSwapchainImagesKHR-pSwapchainImages-parameter
  If the value referenced by `pSwapchainImageCount` is not `0`, and `pSwapchainImages` is not `NULL`, `pSwapchainImages` **must** be a valid pointer to an array of `pSwapchainImageCount` VkImage handles
- VUID-vkGetSwapchainImagesKHR-swapchain-parent
  `swapchain` **must** have been created, allocated, or retrieved from `device`

### Return Codes

**Success**

- `VK_SUCCESS`
- `VK_INCOMPLETE`

**Failure**

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`

**NOTE** By knowing all presentable images used in the swapchain, the application can create command buffers that reference these images prior to entering its main rendering loop. However, command buffers are not allowed to reference presentable images created with `VK_SWAPCHAIN_CREATE_DEFERRED_MEMORY_ALLOCATION_BIT_EXT` until their indices have been returned from vkAcquireNextImageKHR at least once.

Images returned by vkGetSwapchainImagesKHR are fully backed by memory before they are passed to the application, as if they are each bound completely and contiguously to a single `VkDeviceMemory` object , unless the swapchain is created with the `VK_SWAPCHAIN_CREATE_DEFERRED_MEMORY_ALLOCATION_BIT_EXT` flag . All presentable images are initially in the `VK_IMAGE_LAYOUT_UNDEFINED` layout, thus before using presentable images, the application **must** transition them to a valid layout for the intended use.

Further, the lifetime of presentable images is controlled by the implementation, so applications **must** not destroy a presentable image. See vkDestroySwapchainKHR for further details on the lifetime of presentable images.

Images **can** also be created by using vkCreateImage with VkImageSwapchainCreateInfoKHR and bound to swapchain memory using vkBindImageMemory2 with VkBindImageMemorySwapchainInfoKHR. These images **can** be used anywhere swapchain images are used, and are useful in logical devices with multiple physical devices to create peer memory bindings of swapchain memory. These images and bindings have no effect on what memory is presented. Unlike images retrieved from `vkGetSwapchainImagesKHR`, these images **must** be destroyed with vkDestroyImage.

To acquire an available presentable image to use, and retrieve the index of that image, call:

```
// Provided by VK_KHR_swapchain
VkResult vkAcquireNextImageKHR(
    VkDevice                                    device,
    VkSwapchainKHR                              swapchain,
    uint64_t                                    timeout,
    VkSemaphore                                 semaphore,
    VkFence                                     fence,
    uint32_t*                                   pImageIndex);
```

- `device` is the device associated with `swapchain`.
- `swapchain` is the non-retired swapchain from which an image is being acquired.
- `timeout` specifies how long the function waits, in nanoseconds, if no image is available.
- `semaphore` is VK_NULL_HANDLE or a semaphore to signal.
- `fence` is VK_NULL_HANDLE or a fence to signal.
- `pImageIndex` is a pointer to a `uint32_t` in which the index of the next image to use (i.e. an index into the array of images returned by `vkGetSwapchainImagesKHR`) is returned.

If the `swapchain` has been created with the `VK_SWAPCHAIN_CREATE_DEFERRED_MEMORY_ALLOCATION_BIT_EXT` flag, the image whose index is returned in `pImageIndex` will be fully backed by memory before this call returns to the application, as if it is bound completely and contiguously to a single `VkDeviceMemory` object.

## Valid Usage

- VUID-vkAcquireNextImageKHR-swapchain-01285
  `swapchain` **must** not be in the retired state
- VUID-vkAcquireNextImageKHR-semaphore-01286
  If `semaphore` is not VK_NULL_HANDLE, it **must** be unsignaled
- VUID-vkAcquireNextImageKHR-semaphore-01779
  If `semaphore` is not VK_NULL_HANDLE, it **must** not have any uncompleted signal or wait operations pending
- VUID-vkAcquireNextImageKHR-fence-01287
  If `fence` is not VK_NULL_HANDLE, `fence` **must** be unsignaled
- VUID-vkAcquireNextImageKHR-fence-10066
  If `fence` is not VK_NULL_HANDLE, `fence` **must** not be associated with any other queue command that has not yet completed execution on that queue
- VUID-vkAcquireNextImageKHR-semaphore-01780
  `semaphore` and `fence` **must** not both be equal to VK_NULL_HANDLE
- VUID-vkAcquireNextImageKHR-surface-07783
  If forward progress cannot be guaranteed for the `surface` used to create the `swapchain` member of `pAcquireInfo`, the `timeout` member of `pAcquireInfo` **must** not be `UINT64_MAX`
- VUID-vkAcquireNextImageKHR-semaphore-03265

semaphore **must** have a VkSemaphoreType of VK_SEMAPHORE_TYPE_BINARY

## Valid Usage (Implicit)

- VUID-vkAcquireNextImageKHR-device-parameter
  device **must** be a valid VkDevice handle
- VUID-vkAcquireNextImageKHR-swapchain-parameter
  swapchain **must** be a valid VkSwapchainKHR handle
- VUID-vkAcquireNextImageKHR-semaphore-parameter
  If semaphore is not VK_NULL_HANDLE, semaphore **must** be a valid VkSemaphore handle
- VUID-vkAcquireNextImageKHR-fence-parameter
  If fence is not VK_NULL_HANDLE, fence **must** be a valid VkFence handle
- VUID-vkAcquireNextImageKHR-pImageIndex-parameter
  pImageIndex **must** be a valid pointer to a uint32_t value
- VUID-vkAcquireNextImageKHR-swapchain-parent
  swapchain **must** have been created, allocated, or retrieved from device
- VUID-vkAcquireNextImageKHR-semaphore-parent
  If semaphore is a valid handle, it **must** have been created, allocated, or retrieved from device
- VUID-vkAcquireNextImageKHR-fence-parent
  If fence is a valid handle, it **must** have been created, allocated, or retrieved from device

## Host Synchronization

- Host access to swapchain **must** be externally synchronized
- Host access to semaphore **must** be externally synchronized
- Host access to fence **must** be externally synchronized

## Return Codes

### Success

- VK_SUCCESS
- VK_TIMEOUT
- VK_NOT_READY
- VK_SUBOPTIMAL_KHR

### Failure

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY

- VK_ERROR_DEVICE_LOST
- VK_ERROR_OUT_OF_DATE_KHR
- VK_ERROR_SURFACE_LOST_KHR
- VK_ERROR_FULL_SCREEN_EXCLUSIVE_MODE_LOST_EXT

If an image is acquired successfully, vkAcquireNextImageKHR **must** either return VK_SUCCESS or VK_SUBOPTIMAL_KHR. The implementation **may** return VK_SUBOPTIMAL_KHR if the swapchain no longer matches the surface properties exactly, but **can** still be used for presentation.

When successful, vkAcquireNextImageKHR acquires a presentable image from swapchain that an application **can** use, and sets pImageIndex to the index of that image within the swapchain. The presentation engine **may** not have finished reading from the image at the time it is acquired, so the application **must** use semaphore and/or fence to ensure that the image layout and contents are not modified until the presentation engine reads have completed. Once vkAcquireNextImageKHR successfully acquires an image, the semaphore signal operation referenced by semaphore, if not VK_NULL_HANDLE, and the fence signal operation referenced by fence, if not VK_NULL_HANDLE, are submitted for execution. If vkAcquireNextImageKHR does not successfully acquire an image, semaphore and fence are unaffected. The order in which images are acquired is implementation-dependent, and **may** be different than the order the images were presented.

If timeout is zero, then vkAcquireNextImageKHR does not wait, and will either successfully acquire an image, or fail and return VK_NOT_READY if no image is available.

If the specified timeout period expires before an image is acquired, vkAcquireNextImageKHR returns VK_TIMEOUT. If timeout is UINT64_MAX, the timeout period is treated as infinite, and vkAcquireNextImageKHR will block until an image is acquired or an error occurs.

Let S be the number of images in swapchain. If swapchain is created with VkSwapchainPresentModesCreateInfoEXT, let M be the maximum of the values in VkSurfaceCapabilitiesKHR::minImageCount when queried with each present mode in VkSwapchainPresentModesCreateInfoEXT::pPresentModes in VkSurfacePresentModeEXT. Otherwise, let M be the value of VkSurfaceCapabilitiesKHR::minImageCount without a VkSurfacePresentModeEXT as part of the query input.

vkAcquireNextImageKHR **should** not be called if the number of images that the application has currently acquired is greater than S-M. If vkAcquireNextImageKHR is called when the number of images that the application has currently acquired is less than or equal to S-M, vkAcquireNextImageKHR **must** return in finite time with an allowed VkResult code.

**NOTE** Returning a result in finite time guarantees that the implementation cannot deadlock an application, or suspend its execution indefinitely with correct API usage. Acquiring too many images at once may block indefinitely, which is covered by valid usage when attempting to use UINT64_MAX. For example, a scenario here is when a compositor holds on to images which are currently being presented, and there are not any vacant images left to be acquired.

If the swapchain images no longer match native surface properties, either VK_SUBOPTIMAL_KHR or

VK_ERROR_OUT_OF_DATE_KHR **must** be returned. If VK_ERROR_OUT_OF_DATE_KHR is returned, no image is acquired and attempts to present previously acquired images to the swapchain will also fail with VK_ERROR_OUT_OF_DATE_KHR. Applications need to create a new swapchain for the surface to continue presenting if VK_ERROR_OUT_OF_DATE_KHR is returned.

**NOTE**

VK_SUBOPTIMAL_KHR **may** happen, for example, if the platform surface has been resized but the platform is able to scale the presented images to the new size to produce valid surface updates. It is up to the application to decide whether it prefers to continue using the current swapchain in this state, or to re-create the swapchain to better match the platform surface properties.

If device loss occurs (see Lost Device) before the timeout has expired, vkAcquireNextImageKHR **must** return in finite time with either one of the allowed success codes, or VK_ERROR_DEVICE_LOST.

If semaphore is not VK_NULL_HANDLE, the semaphore **must** be unsignaled, with no signal or wait operations pending. It will become signaled when the application **can** use the image.

**NOTE**

Use of semaphore allows rendering operations to be recorded and submitted before the presentation engine has completed its use of the image.

If fence is not equal to VK_NULL_HANDLE, the fence **must** be unsignaled, with no signal operations pending. It will become signaled when the application **can** use the image.

**NOTE**

Applications **should** not rely on vkAcquireNextImageKHR blocking in order to meter their rendering speed. The implementation **may** return from this function immediately regardless of how many presentation requests are queued, and regardless of when queued presentation requests will complete relative to the call. Instead, applications **can** use fence to meter their frame generation work to match the presentation rate.

An application **must** wait until either the semaphore or fence is signaled before accessing the image's data.

**NOTE**

When the presentable image will be accessed by some stage S, the recommended idiom for ensuring correct synchronization is:

- The VkSubmitInfo used to submit the image layout transition for execution includes vkAcquireNextImageKHR::semaphore in its pWaitSemaphores member, with the corresponding element of pWaitDstStageMask including S.
- The synchronization command that performs any necessary image layout transition includes S in both the srcStageMask and dstStageMask.

After a successful return, the image indicated by pImageIndex and its data will be unmodified compared to when it was presented.

**NOTE**

Exclusive ownership of presentable images corresponding to a swapchain created with VK_SHARING_MODE_EXCLUSIVE as defined in Resource Sharing is not altered by a call to vkAcquireNextImageKHR. That means upon the first acquisition from such a

> swapchain presentable images are not owned by any queue family, while at subsequent acquisitions the presentable images remain owned by the queue family the image was previously presented on.

The possible return values for `vkAcquireNextImageKHR` depend on the `timeout` provided:

- `VK_SUCCESS` is returned if an image became available.
- `VK_ERROR_SURFACE_LOST_KHR` is returned if the surface becomes no longer available.
- `VK_NOT_READY` is returned if `timeout` is zero and no image was available.
- `VK_TIMEOUT` is returned if `timeout` is greater than zero and less than `UINT64_MAX`, and no image became available within the time allowed.
- `VK_SUBOPTIMAL_KHR` is returned if an image became available, and the swapchain no longer matches the surface properties exactly, but **can** still be used to present to the surface successfully.

> **NOTE** This **may** happen, for example, if the platform surface has been resized but the platform is able to scale the presented images to the new size to produce valid surface updates. It is up to the application to decide whether it prefers to continue using the current swapchain indefinitely or temporarily in this state, or to re-create the swapchain to better match the platform surface properties.

- `VK_ERROR_OUT_OF_DATE_KHR` is returned if the surface has changed in such a way that it is no longer compatible with the swapchain, and further presentation requests using the swapchain will fail. Applications **must** query the new surface properties and recreate their swapchain if they wish to continue presenting to the surface.

If the native surface and presented image sizes no longer match, presentation **may** fail unless the swapchain is created with a non-zero value in VkSwapchainPresentScalingCreateInfoEXT ::`scalingBehavior` . If presentation does succeed, the mapping from the presented image to the native surface is defined by the VkSwapchainPresentScalingCreateInfoEXT structure if provided. Otherwise it is implementation-defined. It is the application's responsibility to detect surface size changes and react appropriately. If presentation fails because of a mismatch in the surface and presented image sizes, a `VK_ERROR_OUT_OF_DATE_KHR` error will be returned.

> **NOTE** For example, consider a 4x3 window/surface that gets resized to be 3x4 (taller than wider). On some window systems, the portion of the window/surface that was previously and still is visible (the 3x3 part) will contain the same contents as before, while the remaining parts of the window will have undefined contents. Other window systems **may** squash/stretch the image to fill the new window size without any undefined contents, or apply some other mapping.

To acquire an available presentable image to use, and retrieve the index of that image, call:

```
// Provided by VK_VERSION_1_1 with VK_KHR_swapchain, VK_KHR_device_group with
VK_KHR_swapchain
VkResult vkAcquireNextImage2KHR(
    VkDevice                                    device,
    const VkAcquireNextImageInfoKHR*            pAcquireInfo,
    uint32_t*                                   pImageIndex);
```

- `device` is the device associated with `swapchain`.
- `pAcquireInfo` is a pointer to a VkAcquireNextImageInfoKHR structure containing parameters of the acquire.
- `pImageIndex` is a pointer to a `uint32_t` that is set to the index of the next image to use.

If the `swapchain` has been created with the `VK_SWAPCHAIN_CREATE_DEFERRED_MEMORY_ALLOCATION_BIT_EXT` flag, the image whose index is returned in `pImageIndex` will be fully backed by memory before this call returns to the application.

## Valid Usage

- VUID-vkAcquireNextImage2KHR-surface-07784
  If forward progress cannot be guaranteed for the `surface` used to create `swapchain`, the `timeout` member of `pAcquireInfo` **must** not be `UINT64_MAX`

## Valid Usage (Implicit)

- VUID-vkAcquireNextImage2KHR-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkAcquireNextImage2KHR-pAcquireInfo-parameter
  `pAcquireInfo` **must** be a valid pointer to a valid VkAcquireNextImageInfoKHR structure
- VUID-vkAcquireNextImage2KHR-pImageIndex-parameter
  `pImageIndex` **must** be a valid pointer to a `uint32_t` value

## Return Codes

**Success**

- `VK_SUCCESS`
- `VK_TIMEOUT`
- `VK_NOT_READY`
- `VK_SUBOPTIMAL_KHR`

**Failure**

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`

- VK_ERROR_DEVICE_LOST
- VK_ERROR_OUT_OF_DATE_KHR
- VK_ERROR_SURFACE_LOST_KHR
- VK_ERROR_FULL_SCREEN_EXCLUSIVE_MODE_LOST_EXT

The VkAcquireNextImageInfoKHR structure is defined as:

```
// Provided by VK_VERSION_1_1 with VK_KHR_swapchain, VK_KHR_device_group with
VK_KHR_swapchain
typedef struct VkAcquireNextImageInfoKHR {
    VkStructureType    sType;
    const void*        pNext;
    VkSwapchainKHR     swapchain;
    uint64_t           timeout;
    VkSemaphore        semaphore;
    VkFence            fence;
    uint32_t           deviceMask;
} VkAcquireNextImageInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- swapchain is a non-retired swapchain from which an image is acquired.
- timeout specifies how long the function waits, in nanoseconds, if no image is available.
- semaphore is VK_NULL_HANDLE or a semaphore to signal.
- fence is VK_NULL_HANDLE or a fence to signal.
- deviceMask is a mask of physical devices for which the swapchain image will be ready to use when the semaphore or fence is signaled.

If vkAcquireNextImageKHR is used, the device mask is considered to include all physical devices in the logical device.

**NOTE**

vkAcquireNextImage2KHR signals at most one semaphore, even if the application requests waiting for multiple physical devices to be ready via the deviceMask. However, only a single physical device **can** wait on that semaphore, since the semaphore becomes unsignaled when the wait succeeds. For other physical devices to wait for the image to be ready, it is necessary for the application to submit semaphore signal operation(s) to that first physical device to signal additional semaphore(s) after the wait succeeds, which the other physical device(s) **can** wait upon.

### Valid Usage

- VUID-VkAcquireNextImageInfoKHR-swapchain-01675

swapchain **must** not be in the retired state

- VUID-VkAcquireNextImageInfoKHR-semaphore-01288
  If semaphore is not VK_NULL_HANDLE, it **must** be unsignaled
- VUID-VkAcquireNextImageInfoKHR-semaphore-01781
  If semaphore is not VK_NULL_HANDLE, it **must** not have any uncompleted signal or wait operations pending
- VUID-VkAcquireNextImageInfoKHR-fence-01289
  If fence is not VK_NULL_HANDLE, fence **must** be unsignaled
- VUID-VkAcquireNextImageInfoKHR-fence-10067
  If fence is not VK_NULL_HANDLE, fence **must** not be associated with any other queue command that has not yet completed execution on that queue
- VUID-VkAcquireNextImageInfoKHR-semaphore-01782
  semaphore and fence **must** not both be equal to VK_NULL_HANDLE
- VUID-VkAcquireNextImageInfoKHR-deviceMask-01290
  deviceMask **must** be a valid device mask
- VUID-VkAcquireNextImageInfoKHR-deviceMask-01291
  deviceMask **must** not be zero
- VUID-VkAcquireNextImageInfoKHR-semaphore-03266
  semaphore **must** have a VkSemaphoreType of VK_SEMAPHORE_TYPE_BINARY

## Valid Usage (Implicit)

- VUID-VkAcquireNextImageInfoKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_ACQUIRE_NEXT_IMAGE_INFO_KHR
- VUID-VkAcquireNextImageInfoKHR-pNext-pNext
  pNext **must** be NULL
- VUID-VkAcquireNextImageInfoKHR-swapchain-parameter
  swapchain **must** be a valid VkSwapchainKHR handle
- VUID-VkAcquireNextImageInfoKHR-semaphore-parameter
  If semaphore is not VK_NULL_HANDLE, semaphore **must** be a valid VkSemaphore handle
- VUID-VkAcquireNextImageInfoKHR-fence-parameter
  If fence is not VK_NULL_HANDLE, fence **must** be a valid VkFence handle
- VUID-VkAcquireNextImageInfoKHR-commonparent
  Each of fence, semaphore, and swapchain that are valid handles of non-ignored parameters **must** have been created, allocated, or retrieved from the same VkDevice

## Host Synchronization

- Host access to swapchain **must** be externally synchronized
- Host access to semaphore **must** be externally synchronized

- Host access to fence **must** be externally synchronized

After queueing all rendering commands and transitioning the image to the correct layout, to queue an image for presentation, call:

```
// Provided by VK_KHR_swapchain
VkResult vkQueuePresentKHR(
    VkQueue                                     queue,
    const VkPresentInfoKHR*                     pPresentInfo);
```

- queue is a queue that is capable of presentation to the target surface's platform on the same device as the image's swapchain.
- pPresentInfo is a pointer to a VkPresentInfoKHR structure specifying parameters of the presentation.

**NOTE** There is no requirement for an application to present images in the same order that they were acquired - applications can arbitrarily present any image that is currently acquired.

**NOTE** The origin of the native orientation of the surface coordinate system is not specified in the Vulkan specification; it depends on the platform. For most platforms the origin is by default upper-left, meaning the pixel of the presented VkImage at coordinates (0,0) would appear at the upper left pixel of the platform surface (assuming VK_SURFACE_TRANSFORM_IDENTITY_BIT_KHR, and the display standing the right way up).

The result codes VK_ERROR_OUT_OF_DATE_KHR and VK_SUBOPTIMAL_KHR have the same meaning when returned by vkQueuePresentKHR as they do when returned by vkAcquireNextImageKHR. If any swapchain member of pPresentInfo was created with VK_FULL_SCREEN_EXCLUSIVE_APPLICATION_CONTROLLED_EXT, VK_ERROR_FULL_SCREEN_EXCLUSIVE_MODE_LOST_EXT will be returned if that swapchain does not have exclusive full-screen access, possibly for implementation-specific reasons outside of the application's control. If multiple swapchains are presented, the result code is determined by applying the following rules in order:

- If the device is lost, VK_ERROR_DEVICE_LOST is returned.
- If any of the target surfaces are no longer available the error VK_ERROR_SURFACE_LOST_KHR is returned.
- If any of the presents would have a result of VK_ERROR_OUT_OF_DATE_KHR if issued separately then VK_ERROR_OUT_OF_DATE_KHR is returned.
- If any of the presents would have a result of VK_ERROR_FULL_SCREEN_EXCLUSIVE_MODE_LOST_EXT if issued separately then VK_ERROR_FULL_SCREEN_EXCLUSIVE_MODE_LOST_EXT is returned.
- If any of the presents would have a result of VK_SUBOPTIMAL_KHR if issued separately then VK_SUBOPTIMAL_KHR is returned.
- Otherwise VK_SUCCESS is returned.

Any writes to memory backing the images referenced by the pImageIndices and pSwapchains members of pPresentInfo, that are available before vkQueuePresentKHR is executed, are automatically made visible to the read access performed by the presentation engine. This automatic visibility operation for an image happens-after the semaphore signal operation, and happens-before the presentation engine accesses the image.

Presentation is a read-only operation that will not affect the content of the presentable images. Upon reacquiring the image and transitioning it away from the VK_IMAGE_LAYOUT_PRESENT_SRC_KHR layout, the contents will be the same as they were prior to transitioning the image to the present source layout and presenting it. However, if a mechanism other than Vulkan is used to modify the platform window associated with the swapchain, the content of all presentable images in the swapchain becomes undefined.

Calls to vkQueuePresentKHR **may** block, but **must** return in finite time. The processing of the presentation happens in issue order with other queue operations, but semaphores **must** be used to ensure that prior rendering and other commands in the specified queue complete before the presentation begins. The presentation command itself does not delay processing of subsequent commands on the queue. However, presentation requests sent to a particular queue are always performed in order. Exact presentation timing is controlled by the semantics of the presentation engine and native platform in use.

If an image is presented to a swapchain created from a display surface, the mode of the associated display will be updated, if necessary, to match the mode specified when creating the display surface. The mode switch and presentation of the specified image will be performed as one atomic operation.

Queueing an image for presentation defines a set of *queue operations*, including waiting on the semaphores and submitting a presentation request to the presentation engine. However, the scope of this set of queue operations does not include the actual processing of the image by the presentation engine.

If vkQueuePresentKHR fails to enqueue the corresponding set of queue operations, it **may** return VK_ERROR_OUT_OF_HOST_MEMORY or VK_ERROR_OUT_OF_DEVICE_MEMORY. If it does, the implementation **must** ensure that the state and contents of any resources or synchronization primitives referenced is unaffected by the call or its failure.

If vkQueuePresentKHR fails in such a way that the implementation is unable to make that guarantee, the implementation **must** return VK_ERROR_DEVICE_LOST.

However, if the presentation request is rejected by the presentation engine with an error VK_ERROR_OUT_OF_DATE_KHR, VK_ERROR_FULL_SCREEN_EXCLUSIVE_MODE_LOST_EXT, or VK_ERROR_SURFACE_LOST_KHR, the set of queue operations are still considered to be enqueued and thus any semaphore wait operation specified in VkPresentInfoKHR will execute when the corresponding queue operation is complete.

vkQueuePresentKHR releases the acquisition of the images referenced by imageIndices. The queue family corresponding to the queue vkQueuePresentKHR is executed on **must** have ownership of the presented images as defined in Resource Sharing. vkQueuePresentKHR does not alter the queue family ownership, but the presented images **must** not be used again before they have been reacquired

using vkAcquireNextImageKHR.

**NOTE** The application **can** continue to present any acquired images from a retired swapchain as long as the swapchain has not entered a state that causes vkQueuePresentKHR to return VK_ERROR_OUT_OF_DATE_KHR.

## Valid Usage

- VUID-vkQueuePresentKHR-pSwapchains-01292
  Each element of pSwapchains member of pPresentInfo **must** be a swapchain that is created for a surface for which presentation is supported from queue as determined using a call to vkGetPhysicalDeviceSurfaceSupportKHR
- VUID-vkQueuePresentKHR-pSwapchains-01293
  If more than one member of pSwapchains was created from a display surface, all display surfaces referenced that refer to the same display **must** use the same display mode
- VUID-vkQueuePresentKHR-pWaitSemaphores-01294
  When a semaphore wait operation referring to a binary semaphore defined by the elements of the pWaitSemaphores member of pPresentInfo executes on queue, there **must** be no other queues waiting on the same semaphore
- VUID-vkQueuePresentKHR-pWaitSemaphores-03267
  All elements of the pWaitSemaphores member of pPresentInfo **must** be created with a VkSemaphoreType of VK_SEMAPHORE_TYPE_BINARY
- VUID-vkQueuePresentKHR-pWaitSemaphores-03268
  All elements of the pWaitSemaphores member of pPresentInfo **must** reference a semaphore signal operation that has been submitted for execution and any semaphore signal operations on which it depends **must** have also been submitted for execution

## Valid Usage (Implicit)

- VUID-vkQueuePresentKHR-queue-parameter
  queue **must** be a valid VkQueue handle
- VUID-vkQueuePresentKHR-pPresentInfo-parameter
  pPresentInfo **must** be a valid pointer to a valid VkPresentInfoKHR structure

## Host Synchronization

- Host access to queue **must** be externally synchronized
- Host access to pPresentInfo->pWaitSemaphores[] **must** be externally synchronized
- Host access to pPresentInfo->pSwapchains[] **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| - | - | - | Any | - |

## Return Codes

**Success**

- VK_SUCCESS
- VK_SUBOPTIMAL_KHR

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY
- VK_ERROR_DEVICE_LOST
- VK_ERROR_OUT_OF_DATE_KHR
- VK_ERROR_SURFACE_LOST_KHR
- VK_ERROR_FULL_SCREEN_EXCLUSIVE_MODE_LOST_EXT

The VkPresentInfoKHR structure is defined as:

```
// Provided by VK_KHR_swapchain
typedef struct VkPresentInfoKHR {
    VkStructureType          sType;
    const void*              pNext;
    uint32_t                 waitSemaphoreCount;
    const VkSemaphore*       pWaitSemaphores;
    uint32_t                 swapchainCount;
    const VkSwapchainKHR*    pSwapchains;
    const uint32_t*          pImageIndices;
    VkResult*                pResults;
} VkPresentInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- waitSemaphoreCount is the number of semaphores to wait for before issuing the present request. The number **may** be zero.
- pWaitSemaphores is NULL or a pointer to an array of VkSemaphore objects with waitSemaphoreCount entries, and specifies the semaphores to wait for before issuing the present request.
- swapchainCount is the number of swapchains being presented to by this command.

- pSwapchains is a pointer to an array of VkSwapchainKHR objects with swapchainCount entries.
- pImageIndices is a pointer to an array of indices into the array of each swapchain's presentable images, with swapchainCount entries. Each entry in this array identifies the image to present on the corresponding entry in the pSwapchains array.
- pResults is a pointer to an array of VkResult typed elements with swapchainCount entries. Applications that do not need per-swapchain results **can** use NULL for pResults. If non-NULL, each entry in pResults will be set to the VkResult for presenting the swapchain corresponding to the same index in pSwapchains.

Before an application **can** present an image, the image's layout **must** be transitioned to the VK_IMAGE_LAYOUT_PRESENT_SRC_KHR layout, or for a shared presentable image the VK_IMAGE_LAYOUT_SHARED_PRESENT_KHR layout.

**NOTE** When transitioning the image to VK_IMAGE_LAYOUT_SHARED_PRESENT_KHR or VK_IMAGE_LAYOUT_PRESENT_SRC_KHR, there is no need to delay subsequent processing, or perform any visibility operations (as vkQueuePresentKHR performs automatic visibility operations). To achieve this, the dstAccessMask member of the VkImageMemoryBarrier **should** be set to 0, and the dstStageMask parameter **should** be set to VK_PIPELINE_STAGE_BOTTOM_OF_PIPE_BIT.

## Valid Usage

- VUID-VkPresentInfoKHR-pSwapchain-09231
  Elements of pSwapchain **must** be unique
- VUID-VkPresentInfoKHR-pImageIndices-01430
  Each element of pImageIndices **must** be the index of a presentable image acquired from the swapchain specified by the corresponding element of the pSwapchains array, and the presented image subresource **must** be in the VK_IMAGE_LAYOUT_PRESENT_SRC_KHR or VK_IMAGE_LAYOUT_SHARED_PRESENT_KHR layout at the time the operation is executed on a VkDevice
- VUID-VkPresentInfoKHR-pNext-06235
  If a VkPresentIdKHR structure is included in the pNext chain, and the presentId feature is not enabled, each presentIds entry in that structure **must** be NULL
- VUID-VkPresentInfoKHR-pSwapchains-09199
  If any element of the pSwapchains array has been created with VkSwapchainPresentModesCreateInfoEXT, all of the elements of this array **must** be created with VkSwapchainPresentModesCreateInfoEXT

## Valid Usage (Implicit)

- VUID-VkPresentInfoKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PRESENT_INFO_KHR
- VUID-VkPresentInfoKHR-pNext-pNext
  Each pNext member of any structure (including this one) in the pNext chain **must** be either

NULL or a pointer to a valid instance of VkDeviceGroupPresentInfoKHR, VkDisplayPresentInfoKHR, VkFrameBoundaryEXT, VkPresentFrameTokenGGP, VkPresentIdKHR, VkPresentRegionsKHR, VkPresentTimesInfoGOOGLE, VkSwapchainPresentFenceInfoEXT, or VkSwapchainPresentModeInfoEXT

- VUID-VkPresentInfoKHR-sType-unique
  The sType value of each struct in the pNext chain **must** be unique
- VUID-VkPresentInfoKHR-pWaitSemaphores-parameter
  If waitSemaphoreCount is not 0, pWaitSemaphores **must** be a valid pointer to an array of waitSemaphoreCount valid VkSemaphore handles
- VUID-VkPresentInfoKHR-pSwapchains-parameter
  pSwapchains **must** be a valid pointer to an array of swapchainCount valid VkSwapchainKHR handles
- VUID-VkPresentInfoKHR-pImageIndices-parameter
  pImageIndices **must** be a valid pointer to an array of swapchainCount uint32_t values
- VUID-VkPresentInfoKHR-pResults-parameter
  If pResults is not NULL, pResults **must** be a valid pointer to an array of swapchainCount VkResult values
- VUID-VkPresentInfoKHR-swapchainCount-arraylength
  swapchainCount **must** be greater than 0
- VUID-VkPresentInfoKHR-commonparent
  Both of the elements of pSwapchains, and the elements of pWaitSemaphores that are valid handles of non-ignored parameters **must** have been created, allocated, or retrieved from the same VkDevice

When the VK_KHR_incremental_present extension is enabled, additional fields **can** be specified that allow an application to specify that only certain rectangular regions of the presentable images of a swapchain are changed. This is an optimization hint that a presentation engine **may** use to only update the region of a surface that is actually changing. The application still **must** ensure that all pixels of a presented image contain the desired values, in case the presentation engine ignores this hint. An application **can** provide this hint by adding a VkPresentRegionsKHR structure to the pNext chain of the VkPresentInfoKHR structure.

The VkPresentRegionsKHR structure is defined as:

```
// Provided by VK_KHR_incremental_present
typedef struct VkPresentRegionsKHR {
    VkStructureType              sType;
    const void*                  pNext;
    uint32_t                     swapchainCount;
    const VkPresentRegionKHR*    pRegions;
} VkPresentRegionsKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.

- swapchainCount is the number of swapchains being presented to by this command.
- pRegions is NULL or a pointer to an array of VkPresentRegionKHR elements with swapchainCount entries. If not NULL, each element of pRegions contains the region that has changed since the last present to the swapchain in the corresponding entry in the VkPresentInfoKHR::pSwapchains array.

### Valid Usage

- VUID-VkPresentRegionsKHR-swapchainCount-01260
  swapchainCount **must** be the same value as VkPresentInfoKHR::swapchainCount, where VkPresentInfoKHR is included in the pNext chain of this VkPresentRegionsKHR structure

### Valid Usage (Implicit)

- VUID-VkPresentRegionsKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PRESENT_REGIONS_KHR
- VUID-VkPresentRegionsKHR-pRegions-parameter
  If pRegions is not NULL, pRegions **must** be a valid pointer to an array of swapchainCount valid VkPresentRegionKHR structures
- VUID-VkPresentRegionsKHR-swapchainCount-arraylength
  swapchainCount **must** be greater than 0

For a given image and swapchain, the region to present is specified by the VkPresentRegionKHR structure, which is defined as:

```
// Provided by VK_KHR_incremental_present
typedef struct VkPresentRegionKHR {
    uint32_t                    rectangleCount;
    const VkRectLayerKHR*       pRectangles;
} VkPresentRegionKHR;
```

- rectangleCount is the number of rectangles in pRectangles, or zero if the entire image has changed and should be presented.
- pRectangles is either NULL or a pointer to an array of VkRectLayerKHR structures. The VkRectLayerKHR structure is the framebuffer coordinates, plus layer, of a portion of a presentable image that has changed and **must** be presented. If non-NULL, each entry in pRectangles is a rectangle of the given image that has changed since the last image was presented to the given swapchain. The rectangles **must** be specified relative to VkSurfaceCapabilitiesKHR::currentTransform, regardless of the swapchain's preTransform. The presentation engine will apply the preTransform transformation to the rectangles, along with any further transformation it applies to the image content.

### Valid Usage (Implicit)

- VUID-VkPresentRegionKHR-pRectangles-parameter
  If `rectangleCount` is not `0`, and `pRectangles` is not `NULL`, `pRectangles` **must** be a valid pointer to an array of `rectangleCount` valid VkRectLayerKHR structures

The `VkRectLayerKHR` structure is defined as:

```
// Provided by VK_KHR_incremental_present
typedef struct VkRectLayerKHR {
    VkOffset2D    offset;
    VkExtent2D    extent;
    uint32_t      layer;
} VkRectLayerKHR;
```

- `offset` is the origin of the rectangle, in pixels.
- `extent` is the size of the rectangle, in pixels.
- `layer` is the layer of the image. For images with only one layer, the value of `layer` **must** be 0.

Some platforms allow the size of a surface to change, and then scale the pixels of the image to fit the surface. `VkRectLayerKHR` specifies pixels of the swapchain's image(s), which will be constant for the life of the swapchain.

### Valid Usage

- VUID-VkRectLayerKHR-offset-04864
  The sum of `offset` and `extent`, after being transformed according to the `preTransform` member of the VkSwapchainCreateInfoKHR structure, **must** be no greater than the `imageExtent` member of the VkSwapchainCreateInfoKHR structure passed to vkCreateSwapchainKHR
- VUID-VkRectLayerKHR-layer-01262
  `layer` **must** be less than the `imageArrayLayers` member of the VkSwapchainCreateInfoKHR structure passed to vkCreateSwapchainKHR

When the `VK_KHR_display_swapchain` extension is enabled, additional fields **can** be specified when presenting an image to a swapchain by setting VkPresentInfoKHR::`pNext` to point to a VkDisplayPresentInfoKHR structure.

The `VkDisplayPresentInfoKHR` structure is defined as:

```
// Provided by VK_KHR_display_swapchain
typedef struct VkDisplayPresentInfoKHR {
    VkStructureType    sType;
    const void*        pNext;
    VkRect2D           srcRect;
    VkRect2D           dstRect;
    VkBool32           persistent;
} VkDisplayPresentInfoKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `srcRect` is a rectangular region of pixels to present. It **must** be a subset of the image being presented. If `VkDisplayPresentInfoKHR` is not specified, this region will be assumed to be the entire presentable image.
- `dstRect` is a rectangular region within the visible region of the swapchain's display mode. If `VkDisplayPresentInfoKHR` is not specified, this region will be assumed to be the entire visible region of the swapchain's mode. If the specified rectangle is a subset of the display mode's visible region, content from display planes below the swapchain's plane will be visible outside the rectangle. If there are no planes below the swapchain's, the area outside the specified rectangle will be black. If portions of the specified rectangle are outside of the display's visible region, pixels mapping only to those portions of the rectangle will be discarded.
- `persistent`: If this is `VK_TRUE`, the display engine will enable buffered mode on displays that support it. This allows the display engine to stop sending content to the display until a new image is presented. The display will instead maintain a copy of the last presented image. This allows less power to be used, but **may** increase presentation latency. If `VkDisplayPresentInfoKHR` is not specified, persistent mode will not be used.

If the extent of the `srcRect` and `dstRect` are not equal, the presented pixels will be scaled accordingly.

### Valid Usage

- VUID-VkDisplayPresentInfoKHR-srcRect-01257
  `srcRect` **must** specify a rectangular region that is a subset of the image being presented
- VUID-VkDisplayPresentInfoKHR-dstRect-01258
  `dstRect` **must** specify a rectangular region that is a subset of the `visibleRegion` parameter of the display mode the swapchain being presented uses
- VUID-VkDisplayPresentInfoKHR-persistentContent-01259
  If the `persistentContent` member of the `VkDisplayPropertiesKHR` structure returned by `vkGetPhysicalDeviceDisplayPropertiesKHR` for the display the present operation targets is `VK_FALSE`, then `persistent` **must** be `VK_FALSE`

### Valid Usage (Implicit)

- VUID-VkDisplayPresentInfoKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_DISPLAY_PRESENT_INFO_KHR`

If the `pNext` chain of VkPresentInfoKHR includes a `VkDeviceGroupPresentInfoKHR` structure, then that structure includes an array of device masks and a device group present mode.

The `VkDeviceGroupPresentInfoKHR` structure is defined as:

```
// Provided by VK_VERSION_1_1 with VK_KHR_swapchain, VK_KHR_device_group with
VK_KHR_swapchain
typedef struct VkDeviceGroupPresentInfoKHR {
    VkStructureType                         sType;
    const void*                             pNext;
    uint32_t                                swapchainCount;
    const uint32_t*                         pDeviceMasks;
    VkDeviceGroupPresentModeFlagBitsKHR     mode;
} VkDeviceGroupPresentInfoKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `swapchainCount` is zero or the number of elements in `pDeviceMasks`.
- `pDeviceMasks` is a pointer to an array of device masks, one for each element of VkPresentInfoKHR::`pSwapchains`.
- `mode` is a VkDeviceGroupPresentModeFlagBitsKHR value specifying the device group present mode that will be used for this present.

If `mode` is `VK_DEVICE_GROUP_PRESENT_MODE_LOCAL_BIT_KHR`, then each element of `pDeviceMasks` selects which instance of the swapchain image is presented. Each element of `pDeviceMasks` **must** have exactly one bit set, and the corresponding physical device **must** have a presentation engine as reported by VkDeviceGroupPresentCapabilitiesKHR.

If `mode` is `VK_DEVICE_GROUP_PRESENT_MODE_REMOTE_BIT_KHR`, then each element of `pDeviceMasks` selects which instance of the swapchain image is presented. Each element of `pDeviceMasks` **must** have exactly one bit set, and some physical device in the logical device **must** include that bit in its VkDeviceGroupPresentCapabilitiesKHR::`presentMask`.

If `mode` is `VK_DEVICE_GROUP_PRESENT_MODE_SUM_BIT_KHR`, then each element of `pDeviceMasks` selects which instances of the swapchain image are component-wise summed and the sum of those images is presented. If the sum in any component is outside the representable range, the value of that component is undefined. Each element of `pDeviceMasks` **must** have a value for which all set bits are set in one of the elements of VkDeviceGroupPresentCapabilitiesKHR::`presentMask`.

If `mode` is `VK_DEVICE_GROUP_PRESENT_MODE_LOCAL_MULTI_DEVICE_BIT_KHR`, then each element of `pDeviceMasks` selects which instance(s) of the swapchain images are presented. For each bit set in

each element of pDeviceMasks, the corresponding physical device **must** have a presentation engine as reported by VkDeviceGroupPresentCapabilitiesKHR.

If VkDeviceGroupPresentInfoKHR is not provided or swapchainCount is zero then the masks are considered to be 1. If VkDeviceGroupPresentInfoKHR is not provided, mode is considered to be VK_DEVICE_GROUP_PRESENT_MODE_LOCAL_BIT_KHR.

## Valid Usage

- VUID-VkDeviceGroupPresentInfoKHR-swapchainCount-01297
  swapchainCount **must** equal 0 or VkPresentInfoKHR::swapchainCount
- VUID-VkDeviceGroupPresentInfoKHR-mode-01298
  If mode is VK_DEVICE_GROUP_PRESENT_MODE_LOCAL_BIT_KHR, then each element of pDeviceMasks **must** have exactly one bit set, and the corresponding element of VkDeviceGroupPresentCapabilitiesKHR::presentMask **must** be non-zero
- VUID-VkDeviceGroupPresentInfoKHR-mode-01299
  If mode is VK_DEVICE_GROUP_PRESENT_MODE_REMOTE_BIT_KHR, then each element of pDeviceMasks **must** have exactly one bit set, and some physical device in the logical device **must** include that bit in its VkDeviceGroupPresentCapabilitiesKHR::presentMask
- VUID-VkDeviceGroupPresentInfoKHR-mode-01300
  If mode is VK_DEVICE_GROUP_PRESENT_MODE_SUM_BIT_KHR, then each element of pDeviceMasks **must** have a value for which all set bits are set in one of the elements of VkDeviceGroupPresentCapabilitiesKHR::presentMask
- VUID-VkDeviceGroupPresentInfoKHR-mode-01301
  If mode is VK_DEVICE_GROUP_PRESENT_MODE_LOCAL_MULTI_DEVICE_BIT_KHR, then for each bit set in each element of pDeviceMasks, the corresponding element of VkDeviceGroupPresentCapabilitiesKHR::presentMask **must** be non-zero
- VUID-VkDeviceGroupPresentInfoKHR-pDeviceMasks-01302
  The value of each element of pDeviceMasks **must** be equal to the device mask passed in VkAcquireNextImageInfoKHR::deviceMask when the image index was last acquired
- VUID-VkDeviceGroupPresentInfoKHR-mode-01303
  mode **must** have exactly one bit set, and that bit **must** have been included in VkDeviceGroupSwapchainCreateInfoKHR::modes

## Valid Usage (Implicit)

- VUID-VkDeviceGroupPresentInfoKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_DEVICE_GROUP_PRESENT_INFO_KHR
- VUID-VkDeviceGroupPresentInfoKHR-pDeviceMasks-parameter
  If swapchainCount is not 0, pDeviceMasks **must** be a valid pointer to an array of swapchainCount uint32_t values
- VUID-VkDeviceGroupPresentInfoKHR-mode-parameter
  mode **must** be a valid VkDeviceGroupPresentModeFlagBitsKHR value

When the VK_GOOGLE_display_timing extension is enabled, additional fields **can** be specified that allow an application to specify the earliest time that an image should be displayed. This allows an application to avoid stutter that is caused by an image being displayed earlier than planned. Such stuttering can occur with both fixed and variable-refresh-rate displays, because stuttering occurs when the geometry is not correctly positioned for when the image is displayed. An application **can** instruct the presentation engine that an image should not be displayed earlier than a specified time by adding a VkPresentTimesInfoGOOGLE structure to the pNext chain of the VkPresentInfoKHR structure.

The VkPresentTimesInfoGOOGLE structure is defined as:

```
// Provided by VK_GOOGLE_display_timing
typedef struct VkPresentTimesInfoGOOGLE {
    VkStructureType                 sType;
    const void*                     pNext;
    uint32_t                        swapchainCount;
    const VkPresentTimeGOOGLE*      pTimes;
} VkPresentTimesInfoGOOGLE;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- swapchainCount is the number of swapchains being presented to by this command.
- pTimes is NULL or a pointer to an array of VkPresentTimeGOOGLE elements with swapchainCount entries. If not NULL, each element of pTimes contains the earliest time to present the image corresponding to the entry in the VkPresentInfoKHR::pImageIndices array.

### Valid Usage

- VUID-VkPresentTimesInfoGOOGLE-swapchainCount-01247
  swapchainCount **must** be the same value as VkPresentInfoKHR::swapchainCount, where VkPresentInfoKHR is included in the pNext chain of this VkPresentTimesInfoGOOGLE structure

### Valid Usage (Implicit)

- VUID-VkPresentTimesInfoGOOGLE-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PRESENT_TIMES_INFO_GOOGLE
- VUID-VkPresentTimesInfoGOOGLE-pTimes-parameter
  If pTimes is not NULL, pTimes **must** be a valid pointer to an array of swapchainCount VkPresentTimeGOOGLE structures
- VUID-VkPresentTimesInfoGOOGLE-swapchainCount-arraylength
  swapchainCount **must** be greater than 0

The VkPresentTimeGOOGLE structure is defined as:

```
// Provided by VK_GOOGLE_display_timing
typedef struct VkPresentTimeGOOGLE {
    uint32_t    presentID;
    uint64_t    desiredPresentTime;
} VkPresentTimeGOOGLE;
```

- presentID is an application-provided identification value, that **can** be used with the results of vkGetPastPresentationTimingGOOGLE, in order to uniquely identify this present. In order to be useful to the application, it **should** be unique within some period of time that is meaningful to the application.
- desiredPresentTime specifies that the image given **should** not be displayed to the user any earlier than this time. desiredPresentTime is a time in nanoseconds, relative to a monotonically-increasing clock (e.g. CLOCK_MONOTONIC (see clock_gettime(2)) on Android and Linux). A value of zero specifies that the presentation engine **may** display the image at any time. This is useful when the application desires to provide presentID, but does not need a specific desiredPresentTime.

The VkPresentIdKHR structure is defined as:

```
// Provided by VK_KHR_present_id
typedef struct VkPresentIdKHR {
    VkStructureType    sType;
    const void*        pNext;
    uint32_t           swapchainCount;
    const uint64_t*    pPresentIds;
} VkPresentIdKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- swapchainCount is the number of swapchains being presented to the vkQueuePresentKHR command.
- pPresentIds is NULL or a pointer to an array of uint64_t with swapchainCount entries. If not NULL, each non-zero value in pPresentIds specifies the present id to be associated with the presentation of the swapchain with the same index in the vkQueuePresentKHR call.

For applications to be able to reference specific presentation events queued by a call to vkQueuePresentKHR, an identifier needs to be associated with them. When the presentId feature is enabled, applications **can** include the VkPresentIdKHR structure in the pNext chain of the VkPresentInfoKHR structure to supply identifiers.

Each VkSwapchainKHR has a presentId associated with it. This value is initially set to zero when the VkSwapchainKHR is created.

When a VkPresentIdKHR structure with a non-NULL pPresentIds is included in the pNext chain of a VkPresentInfoKHR structure, each pSwapchains entry has a presentId associated in the pPresentIds array at the same index as the swapchain in the pSwapchains array. If this presentId is non-zero,

then the application **can** later use this value to refer to that image presentation. A value of zero indicates that this presentation has no associated presentId. A non-zero presentId **must** be greater than any non-zero presentId passed previously by the application for the same swapchain.

There is no requirement for any precise timing relationship between the presentation of the image to the user and the update of the presentId value, but implementations **should** make this as close as possible to the presentation of the first pixel in the new image to the user.

### Valid Usage

- VUID-VkPresentIdKHR-swapchainCount-04998
  `swapchainCount` **must** be the same value as `VkPresentInfoKHR::swapchainCount`, where this `VkPresentIdKHR` is in the `pNext` chain of the `VkPresentInfoKHR` structure
- VUID-VkPresentIdKHR-presentIds-04999
  Each `presentIds` entry **must** be greater than any previous `presentIds` entry passed for the associated `pSwapchains` entry

### Valid Usage (Implicit)

- VUID-VkPresentIdKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PRESENT_ID_KHR`
- VUID-VkPresentIdKHR-pPresentIds-parameter
  If `pPresentIds` is not `NULL`, `pPresentIds` **must** be a valid pointer to an array of `swapchainCount` `uint64_t` values
- VUID-VkPresentIdKHR-swapchainCount-arraylength
  `swapchainCount` **must** be greater than `0`

When the `presentWait` feature is enabled, an application **can** wait for an image to be presented to the user by first specifying a presentId for the target presentation by adding a `VkPresentIdKHR` structure to the `pNext` chain of the VkPresentInfoKHR structure and then waiting for that presentation to complete by calling:

```
// Provided by VK_KHR_present_wait
VkResult vkWaitForPresentKHR(
    VkDevice                                    device,
    VkSwapchainKHR                              swapchain,
    uint64_t                                    presentId,
    uint64_t                                    timeout);
```

- `device` is the device associated with `swapchain`.
- `swapchain` is the non-retired swapchain on which an image was queued for presentation.
- `presentId` is the presentation presentId to wait for.
- `timeout` is the timeout period in units of nanoseconds. `timeout` is adjusted to the closest value

allowed by the implementation-dependent timeout accuracy, which **may** be substantially longer than one nanosecond, and **may** be longer than the requested period.

`vkWaitForPresentKHR` waits for the presentId associated with `swapchain` to be increased in value so that it is at least equal to `presentId`.

For `VK_PRESENT_MODE_MAILBOX_KHR` (or other present mode where images may be replaced in the presentation queue) any wait of this type associated with such an image **must** be signaled no later than a wait associated with the replacing image would be signaled.

When the presentation has completed, the presentId associated with the related `pSwapchains` entry will be increased in value so that it is at least equal to the value provided in the `VkPresentIdKHR` structure.

There is no requirement for any precise timing relationship between the presentation of the image to the user and the update of the presentId value, but implementations **should** make this as close as possible to the presentation of the first pixel in the next image being presented to the user.

The call to `vkWaitForPresentKHR` will block until either the presentId associated with `swapchain` is greater than or equal to `presentId`, or `timeout` nanoseconds passes. When the swapchain becomes OUT_OF_DATE, the call will either return `VK_SUCCESS` (if the image was delivered to the presentation engine and may have been presented to the user) or will return early with status `VK_ERROR_OUT_OF_DATE_KHR` (if the image was not presented to the user).

As an exception to the normal rules for objects which are externally synchronized, the `swapchain` passed to `vkWaitForPresentKHR` **may** be simultaneously used by other threads in calls to functions other than vkDestroySwapchainKHR. Access to the swapchain data associated with this extension **must** be atomic within the implementation.

## Valid Usage

- VUID-vkWaitForPresentKHR-swapchain-04997
  `swapchain` **must** not be in the retired state
- VUID-vkWaitForPresentKHR-presentWait-06234
  The `presentWait` feature **must** be enabled

## Valid Usage (Implicit)

- VUID-vkWaitForPresentKHR-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkWaitForPresentKHR-swapchain-parameter
  `swapchain` **must** be a valid VkSwapchainKHR handle
- VUID-vkWaitForPresentKHR-swapchain-parent
  `swapchain` **must** have been created, allocated, or retrieved from `device`

## Host Synchronization

- Host access to swapchain **must** be externally synchronized

## Return Codes

**Success**

- VK_SUCCESS
- VK_TIMEOUT
- VK_SUBOPTIMAL_KHR

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY
- VK_ERROR_DEVICE_LOST
- VK_ERROR_OUT_OF_DATE_KHR
- VK_ERROR_SURFACE_LOST_KHR
- VK_ERROR_FULL_SCREEN_EXCLUSIVE_MODE_LOST_EXT

When the VK_GGP_frame_token extension is enabled, a Google Games Platform frame token **can** be specified when presenting an image to a swapchain by adding a VkPresentFrameTokenGGP structure to the pNext chain of the VkPresentInfoKHR structure.

The VkPresentFrameTokenGGP structure is defined as:

```
// Provided by VK_GGP_frame_token
typedef struct VkPresentFrameTokenGGP {
    VkStructureType    sType;
    const void*        pNext;
    GgpFrameToken      frameToken;
} VkPresentFrameTokenGGP;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- frameToken is the Google Games Platform frame token.

## Valid Usage

- VUID-VkPresentFrameTokenGGP-frameToken-02680
  frameToken **must** be a valid GgpFrameToken

### Valid Usage (Implicit)

- VUID-VkPresentFrameTokenGGP-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PRESENT_FRAME_TOKEN_GGP`

The `VkSwapchainPresentModeInfoEXT` structure is defined as:

```
// Provided by VK_EXT_swapchain_maintenance1
typedef struct VkSwapchainPresentModeInfoEXT {
    VkStructureType            sType;
    const void*                pNext;
    uint32_t                   swapchainCount;
    const VkPresentModeKHR*    pPresentModes;
} VkSwapchainPresentModeInfoEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `swapchainCount` is the number of swapchains being presented to by this command.
- `pPresentModes` is a list of presentation modes with `swapchainCount` entries.

If the `pNext` chain of VkPresentInfoKHR includes a `VkSwapchainPresentModeInfoEXT` structure, then that structure defines the presentation modes used for the current and subsequent presentation operations.

When the application changes present modes with VkSwapchainPresentModeInfoEXT, images that have already been queued for presentation will continue to be presented according to the previous present mode. The current image being queued for presentation and subsequent images will be presented according to the new present mode. The behavior during the transition between the two modes is defined as follows.

- Transition from `VK_PRESENT_MODE_SHARED_CONTINUOUS_REFRESH_KHR` to `VK_PRESENT_MODE_SHARED_DEMAND_REFRESH_KHR`: the presentation engine updates the shared presentable image according to the behavior of `VK_PRESENT_MODE_SHARED_DEMAND_REFRESH_KHR`.
- Transition from `VK_PRESENT_MODE_SHARED_DEMAND_REFRESH_KHR` to `VK_PRESENT_MODE_SHARED_CONTINUOUS_REFRESH_KHR`: the presentation engine **may** update the shared presentable image or defer that to its regular refresh cycle, according to the behavior of `VK_PRESENT_MODE_SHARED_CONTINUOUS_REFRESH_KHR`.
- Transition between `VK_PRESENT_MODE_FIFO_KHR` and `VK_PRESENT_MODE_FIFO_RELAXED_KHR`: Images continue to be appended to the same FIFO queue, and the behavior with respect to waiting for vertical blanking period will follow the new mode for current and subsequent images.
- Transition from `VK_PRESENT_MODE_IMMEDIATE_KHR` to `VK_PRESENT_MODE_FIFO_KHR` or `VK_PRESENT_MODE_FIFO_RELAXED_KHR`: As all prior present requests in the `VK_PRESENT_MODE_IMMEDIATE_KHR` mode are applied immediately, there are no outstanding present operations in this mode, and current and subsequent images are appended to the FIFO queue

and presented according to the new mode.

- Transition from VK_PRESENT_MODE_MAILBOX_KHR to VK_PRESENT_MODE_FIFO_KHR or VK_PRESENT_MODE_FIFO_RELAXED_KHR: Presentation in both modes require waiting for the next vertical blanking period, with VK_PRESENT_MODE_MAILBOX_KHR allowing the pending present operation to be replaced by a new one. In this case, the current present operation will replace the pending present operation and is applied according to the new mode.
- Transition from VK_PRESENT_MODE_FIFO_KHR or VK_PRESENT_MODE_FIFO_RELAXED_KHR to VK_PRESENT_MODE_IMMEDIATE_KHR or VK_PRESENT_MODE_MAILBOX_KHR: If the FIFO queue is empty, presentation is done according to the behavior of the new mode. If there are present operations in the FIFO queue, once the last present operation is performed based on the respective vertical blanking period, the current and subsequent updates are applied according to the new mode.
- The behavior during transition between any other present modes, if possible, is implementation defined.

### Valid Usage

- VUID-VkSwapchainPresentModeInfoEXT-swapchainCount-07760
  swapchainCount **must** be equal to VkPresentInfoKHR::swapchainCount
- VUID-VkSwapchainPresentModeInfoEXT-pPresentModes-07761
  Each entry in pPresentModes **must** be a presentation mode specified in VkSwapchainPresentModesCreateInfoEXT::pPresentModes when creating the entry's corresponding swapchain

### Valid Usage (Implicit)

- VUID-VkSwapchainPresentModeInfoEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_SWAPCHAIN_PRESENT_MODE_INFO_EXT
- VUID-VkSwapchainPresentModeInfoEXT-pPresentModes-parameter
  pPresentModes **must** be a valid pointer to an array of swapchainCount valid VkPresentModeKHR values
- VUID-VkSwapchainPresentModeInfoEXT-swapchainCount-arraylength
  swapchainCount **must** be greater than 0

The VkSwapchainPresentFenceInfoEXT structure is defined as:

```
// Provided by VK_EXT_swapchain_maintenance1
typedef struct VkSwapchainPresentFenceInfoEXT {
    VkStructureType    sType;
    const void*        pNext;
    uint32_t           swapchainCount;
    const VkFence*     pFences;
} VkSwapchainPresentFenceInfoEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `swapchainCount` is the number of swapchains being presented to by this command.
- `pFences` is a list of fences with `swapchainCount` entries. Each entry **must** be VK_NULL_HANDLE or the handle of a fence to signal when the relevant operations on the associated swapchain have completed.

The set of *queue operations* defined by queuing an image for presentation, as well as operations performed by the presentation engine access the payloads of objects associated with the presentation operation. The associated objects include:

- The swapchain image, its implicitly bound memory, and any other resources bound to that memory.
- The wait semaphores specified when queuing the image for presentation.

The application **can** provide a fence that the implementation will signal when all such queue operations have completed and the presentation engine has taken a reference to the payload of any objects it accesses as part of the present operation. For all binary wait semaphores imported by the presentation engine using the equivalent of reference transference, as described in Importing Semaphore Payloads, this fence **must** not signal until all such semaphore payloads have been reset by the presentation engine.

The application **can** destroy the wait semaphores associated with a given presentation operation when at least one of the associated fences is signaled, and **can** destroy the swapchain when the fences associated with all past presentation requests referring to that swapchain have signaled.

Fences associated with presentations to the same swapchain on the same VkQueue **must** be signaled in the same order as the present operations.

To specify a fence for each swapchain in a present operation, include the `VkSwapchainPresentFenceInfoEXT` structure in the `pNext` chain of the VkPresentInfoKHR structure.

### Valid Usage

- VUID-VkSwapchainPresentFenceInfoEXT-swapchainCount-07757
  `swapchainCount` **must** be equal to VkPresentInfoKHR::`swapchainCount`
- VUID-VkSwapchainPresentFenceInfoEXT-pFences-07758
  Each element of `pFences` **must** be unsignaled
- VUID-VkSwapchainPresentFenceInfoEXT-pFences-07759
  Each element of `pFences` **must** not be associated with any other queue command that has not yet completed execution on that queue

### Valid Usage (Implicit)

- VUID-VkSwapchainPresentFenceInfoEXT-sType-sType

sType **must** be VK_STRUCTURE_TYPE_SWAPCHAIN_PRESENT_FENCE_INFO_EXT

- VUID-VkSwapchainPresentFenceInfoEXT-pFences-parameter
  pFences **must** be a valid pointer to an array of swapchainCount valid VkFence handles
- VUID-VkSwapchainPresentFenceInfoEXT-swapchainCount-arraylength
  swapchainCount **must** be greater than 0

To release images previously acquired through vkAcquireNextImage2KHR or vkAcquireNextImageKHR, call:

```
// Provided by VK_EXT_swapchain_maintenance1
VkResult vkReleaseSwapchainImagesEXT(
    VkDevice                                    device,
    const VkReleaseSwapchainImagesInfoEXT*      pReleaseInfo);
```

- device is the device associated with VkReleaseSwapchainImagesInfoEXT::swapchain.
- pReleaseInfo is a pointer to a VkReleaseSwapchainImagesInfoEXT structure containing parameters of the release.

Only images that are not in use by the device **can** be released.

Releasing images is a read-only operation that will not affect the content of the released images. Upon reacquiring the image, the image contents and its layout will be the same as they were prior to releasing it. However, if a mechanism other than Vulkan is used to modify the platform window associated with the swapchain, the content of all presentable images in the swapchain becomes undefined.

| | |
|---|---|
| **NOTE** | This functionality is useful during swapchain recreation, where acquired images from the old swapchain can be released instead of presented. |

### Valid Usage (Implicit)

- VUID-vkReleaseSwapchainImagesEXT-device-parameter
  device **must** be a valid VkDevice handle
- VUID-vkReleaseSwapchainImagesEXT-pReleaseInfo-parameter
  pReleaseInfo **must** be a valid pointer to a valid VkReleaseSwapchainImagesInfoEXT structure

### Return Codes

**Success**

- VK_SUCCESS

**Failure**

- VK_ERROR_SURFACE_LOST_KHR

The VkReleaseSwapchainImagesInfoEXT structure is defined as:

```
// Provided by VK_EXT_swapchain_maintenance1
typedef struct VkReleaseSwapchainImagesInfoEXT {
    VkStructureType    sType;
    const void*        pNext;
    VkSwapchainKHR     swapchain;
    uint32_t           imageIndexCount;
    const uint32_t*    pImageIndices;
} VkReleaseSwapchainImagesInfoEXT;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- swapchain is a swapchain to which images are being released.
- imageIndexCount is the number of image indices to be released.
- pImageIndices is a pointer to an array of indices into the array of swapchain's presentable images, with imageIndexCount entries.

## Valid Usage

- VUID-VkReleaseSwapchainImagesInfoEXT-pImageIndices-07785
  Each element of pImageIndices **must** be the index of a presentable image acquired from the swapchain specified by swapchain
- VUID-VkReleaseSwapchainImagesInfoEXT-pImageIndices-07786
  All uses of presentable images identified by elements of pImageIndices **must** have completed execution

## Valid Usage (Implicit)

- VUID-VkReleaseSwapchainImagesInfoEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_RELEASE_SWAPCHAIN_IMAGES_INFO_EXT
- VUID-VkReleaseSwapchainImagesInfoEXT-pNext-pNext
  pNext **must** be NULL
- VUID-VkReleaseSwapchainImagesInfoEXT-swapchain-parameter
  swapchain **must** be a valid VkSwapchainKHR handle
- VUID-VkReleaseSwapchainImagesInfoEXT-pImageIndices-parameter
  pImageIndices **must** be a valid pointer to an array of imageIndexCount uint32_t values
- VUID-VkReleaseSwapchainImagesInfoEXT-imageIndexCount-arraylength
  imageIndexCount **must** be greater than 0

### Host Synchronization

- Host access to `swapchain` **must** be externally synchronized

## 34.11. HDR Metadata

This section describes how to improve color reproduction of content to better reproduce colors as seen on the display used to originally optimize the content.

To provide HDR metadata to an implementation, call:

```
// Provided by VK_EXT_hdr_metadata
void vkSetHdrMetadataEXT(
    VkDevice                                    device,
    uint32_t                                    swapchainCount,
    const VkSwapchainKHR*                       pSwapchains,
    const VkHdrMetadataEXT*                     pMetadata);
```

- `device` is the logical device where the swapchain(s) were created.
- `swapchainCount` is the number of swapchains included in `pSwapchains`.
- `pSwapchains` is a pointer to an array of `swapchainCount` VkSwapchainKHR handles.
- `pMetadata` is a pointer to an array of `swapchainCount` VkHdrMetadataEXT structures.

The metadata will be applied to the specified VkSwapchainKHR objects at the next vkQueuePresentKHR call using that VkSwapchainKHR object. The metadata will persist until a subsequent `vkSetHdrMetadataEXT` changes it.

### Valid Usage (Implicit)

- VUID-vkSetHdrMetadataEXT-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkSetHdrMetadataEXT-pSwapchains-parameter
  `pSwapchains` **must** be a valid pointer to an array of `swapchainCount` valid VkSwapchainKHR handles
- VUID-vkSetHdrMetadataEXT-pMetadata-parameter
  `pMetadata` **must** be a valid pointer to an array of `swapchainCount` valid VkHdrMetadataEXT structures
- VUID-vkSetHdrMetadataEXT-swapchainCount-arraylength
  `swapchainCount` **must** be greater than `0`
- VUID-vkSetHdrMetadataEXT-pSwapchains-parent
  Each element of `pSwapchains` **must** have been created, allocated, or retrieved from `device`

The VkHdrMetadataEXT structure is defined as:

```
// Provided by VK_EXT_hdr_metadata
typedef struct VkHdrMetadataEXT {
    VkStructureType    sType;
    const void*        pNext;
    VkXYColorEXT       displayPrimaryRed;
    VkXYColorEXT       displayPrimaryGreen;
    VkXYColorEXT       displayPrimaryBlue;
    VkXYColorEXT       whitePoint;
    float              maxLuminance;
    float              minLuminance;
    float              maxContentLightLevel;
    float              maxFrameAverageLightLevel;
} VkHdrMetadataEXT;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- displayPrimaryRed is a VkXYColorEXT structure specifying the red primary of the display used to optimize the content
- displayPrimaryGreen is a VkXYColorEXT structure specifying the green primary of the display used to optimize the content
- displayPrimaryBlue is a VkXYColorEXT structure specifying the blue primary of the display used to optimize the content
- whitePoint is a VkXYColorEXT structure specifying the white-point of the display used to optimize the content
- maxLuminance is the maximum luminance of the display used to optimize the content in nits
- minLuminance is the minimum luminance of the display used to optimize the content in nits
- maxContentLightLevel is the value in nits of the desired luminance for the brightest pixels in the displayed image.
- maxFrameAverageLightLevel is the value in nits of the average luminance of the frame which has the brightest average luminance anywhere in the content.

If any of the above values are unknown, they **can** be set to 0.

| | |
|---|---|
| **NOTE** | The meta-data provided here is intended to be used as defined in the SMPTE 2086, CTA 861.3 and CIE 15:2004 specifications. The validity and use of this data is outside the scope of Vulkan. |

### Valid Usage (Implicit)

- VUID-VkHdrMetadataEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_HDR_METADATA_EXT

- VUID-VkHdrMetadataEXT-pNext-pNext
  `pNext` **must** be `NULL`

The `VkXYColorEXT` structure is defined as:

```
// Provided by VK_EXT_hdr_metadata
typedef struct VkXYColorEXT {
    float    x;
    float    y;
} VkXYColorEXT;
```

- `x` is the x chromaticity coordinate.
- `y` is the y chromaticity coordinate.

Chromaticity coordinates are as specified in CIE 15:2004 “Calculation of chromaticity coordinates” (Section 7.3) and are limited to between 0 and 1 for real colors.

# 34.12. Lag Control

Some implementations support extensions to reduce display latency and control the swapchain present interval, as described in the remainder of this section.

## 34.12.1. Anti-Lag

The VK_AMD_anti_lag extension lowers latency between receiving input and displaying on the screen. It adds a command to indicate when input is being processed for a frame, and when that frame’s images are presented.

To lower latency, call:

```
// Provided by VK_AMD_anti_lag
void vkAntiLagUpdateAMD(
    VkDevice                                    device,
    const VkAntiLagDataAMD*                     pData);
```

- `device` is the logical device
- `pData` is a pointer to a VkAntiLagDataAMD structure containing latency reduction parameters.

This command should be executed immediately before the application processes user input. If `pData` is not `NULL` and VkAntiLagDataAMD::`presentationInfo` is not `NULL`, this command **should** be executed again before vkQueuePresentKHR, with `pPresentationInfo` set to matching values.

### Valid Usage

- VUID-vkAntiLagUpdateAMD-antiLag-10061

The antiLag feature **must** be enabled

### Valid Usage (Implicit)

- VUID-vkAntiLagUpdateAMD-device-parameter
  device **must** be a valid VkDevice handle
- VUID-vkAntiLagUpdateAMD-pData-parameter
  pData **must** be a valid pointer to a valid VkAntiLagDataAMD structure

The VkAntiLagDataAMD structure is defined as:

```
// Provided by VK_AMD_anti_lag
typedef struct VkAntiLagDataAMD {
    VkStructureType                          sType;
    const void*                              pNext;
    VkAntiLagModeAMD                         mode;
    uint32_t                                 maxFPS;
    const VkAntiLagPresentationInfoAMD*      pPresentationInfo;
} VkAntiLagDataAMD;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- mode is a VkAntiLagModeAMD value specifying the anti-lag status.
- maxFPS is the framerate limit, in frames per second, used by the application. This limit will be imposed if anti-lag is enabled. If the application tries to render faster, the framerate will be reduced to match this limit. A value of 0 will disable the limit.
- pPresentationInfo is a pointer to a VkAntiLagPresentationInfoAMD structure containing information about the application stage.

This structure specifies anti-lag parameters.

### Valid Usage (Implicit)

- VUID-VkAntiLagDataAMD-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_ANTI_LAG_DATA_AMD
- VUID-VkAntiLagDataAMD-mode-parameter
  mode **must** be a valid VkAntiLagModeAMD value
- VUID-VkAntiLagDataAMD-pPresentationInfo-parameter
  If pPresentationInfo is not NULL, pPresentationInfo **must** be a valid pointer to a valid VkAntiLagPresentationInfoAMD structure

Possible values of VkAntiLagDataAMD::mode, specifying the anti-lag status, are:

```
// Provided by VK_AMD_anti_lag
typedef enum VkAntiLagModeAMD {
    VK_ANTI_LAG_MODE_DRIVER_CONTROL_AMD = 0,
    VK_ANTI_LAG_MODE_ON_AMD = 1,
    VK_ANTI_LAG_MODE_OFF_AMD = 2,
} VkAntiLagModeAMD;
```

- `VK_ANTI_LAG_MODE_DRIVER_CONTROL_AMD` specifies that anti-lag will be enabled or disabled depending on driver settings.
- `VK_ANTI_LAG_MODE_ON_AMD` specifies that anti-lag will be enabled.
- `VK_ANTI_LAG_MODE_OFF_AMD` specifies that anti-lag will be disabled.

The `VkAntiLagPresentationInfoAMD` structure is defined as:

```
// Provided by VK_AMD_anti_lag
typedef struct VkAntiLagPresentationInfoAMD {
    VkStructureType      sType;
    void*                pNext;
    VkAntiLagStageAMD    stage;
    uint64_t             frameIndex;
} VkAntiLagPresentationInfoAMD;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `stage` is a VkAntiLagStageAMD value specifying the current application stage.
- `frameIndex` is set just before the application processes input data (`VK_ANTI_LAG_STAGE_INPUT_AMD`). The same `frameIndex` value **should** be set before the frame with current input data will be presented by vkQueuePresentKHR (`VK_ANTI_LAG_STAGE_PRESENT_AMD`). This **should** be done for each frame.

This structure specifies information about the presentation stage for which anti-lag parameters are being set.

### Valid Usage (Implicit)

- VUID-VkAntiLagPresentationInfoAMD-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_ANTI_LAG_PRESENTATION_INFO_AMD`
- VUID-VkAntiLagPresentationInfoAMD-stage-parameter
  `stage` **must** be a valid VkAntiLagStageAMD value

Possible values of VkAntiLagPresentationInfoAMD::`stage`, specifying the current application stage, are:

```
// Provided by VK_AMD_anti_lag
typedef enum VkAntiLagStageAMD {
    VK_ANTI_LAG_STAGE_INPUT_AMD = 0,
    VK_ANTI_LAG_STAGE_PRESENT_AMD = 1,
} VkAntiLagStageAMD;
```

- VK_ANTI_LAG_STAGE_INPUT_AMD specifies the stage before processing input.
- VK_ANTI_LAG_STAGE_PRESENT_AMD specifies the stage before vkQueuePresentKHR.

### 34.12.2. Latency Reduction

To enable or disable low latency mode on a swapchain, call:

```
// Provided by VK_NV_low_latency2
VkResult vkSetLatencySleepModeNV(
    VkDevice                                    device,
    VkSwapchainKHR                              swapchain,
    const VkLatencySleepModeInfoNV*             pSleepModeInfo);
```

- device is the device associated with swapchain.
- swapchain is the swapchain to enable or disable low latency mode on.
- pSleepModeInfo is NULL or a pointer to a VkLatencySleepModeInfoNV structure specifying the parameters of the latency sleep mode.

If pSleepModeInfo is NULL, vkSetLatencySleepModeNV will disable low latency mode, low latency boost, and set the minimum present interval previously specified by VkLatencySleepModeInfoNV to zero on swapchain. As an exception to the normal rules for objects which are externally synchronized, the swapchain passed to vkSetLatencySleepModeNV **may** be simultaneously used by other threads in calls to functions other than vkDestroySwapchainKHR. Access to the swapchain data associated with this extension **must** be atomic within the implementation.

#### Valid Usage (Implicit)

- VUID-vkSetLatencySleepModeNV-device-parameter
  device **must** be a valid VkDevice handle
- VUID-vkSetLatencySleepModeNV-swapchain-parameter
  swapchain **must** be a valid VkSwapchainKHR handle
- VUID-vkSetLatencySleepModeNV-pSleepModeInfo-parameter
  pSleepModeInfo **must** be a valid pointer to a valid VkLatencySleepModeInfoNV structure
- VUID-vkSetLatencySleepModeNV-swapchain-parent
  swapchain **must** have been created, allocated, or retrieved from device

### Return Codes

**Success**

- VK_SUCCESS

**Failure**

- VK_ERROR_INITIALIZATION_FAILED

The VkLatencySleepModeInfoNV structure is defined as:

```
// Provided by VK_NV_low_latency2
typedef struct VkLatencySleepModeInfoNV {
    VkStructureType    sType;
    const void*        pNext;
    VkBool32           lowLatencyMode;
    VkBool32           lowLatencyBoost;
    uint32_t           minimumIntervalUs;
} VkLatencySleepModeInfoNV;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- lowLatencyMode is the toggle to enable or disable low latency mode.
- lowLatencyBoost allows an application to hint to the GPU to increase performance to provide additional latency savings at a cost of increased power consumption.
- minimumIntervalUs is the microseconds between vkQueuePresentKHR calls for a given swapchain that vkLatencySleepNV will enforce.

If lowLatencyMode is set to VK_FALSE, lowLatencyBoost will still hint to the GPU to increase its power state and vkLatencySleepNV will still enforce minimumIntervalUs between vkQueuePresentKHR calls.

### Valid Usage (Implicit)

- VUID-VkLatencySleepModeInfoNV-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_LATENCY_SLEEP_MODE_INFO_NV

To provide the synchronization primitive used to delay host CPU work for lower latency rendering, call:

```
// Provided by VK_NV_low_latency2
VkResult vkLatencySleepNV(
    VkDevice                                    device,
    VkSwapchainKHR                              swapchain,
    const VkLatencySleepInfoNV*                 pSleepInfo);
```

- device is the device associated with swapchain.
- swapchain is the swapchain to delay associated CPU work based on VkLatencySubmissionPresentIdNV submissions.
- pSleepInfo is a pointer to a VkLatencySleepInfoNV structure specifying the parameters of the latency sleep.

vkLatencySleepNV returns immediately. Applications **should** use vkWaitSemaphoresKHR with pSleepInfo->signalSemaphore to delay host CPU work. CPU work refers to application work done before presenting which includes but is not limited to: input sampling, simulation, command buffer recording, command buffer submission, and present submission. It is recommended to call this function before input sampling. When using this function, it **should** be called exactly once between presents.

### Valid Usage (Implicit)

- VUID-vkLatencySleepNV-device-parameter
  device **must** be a valid VkDevice handle
- VUID-vkLatencySleepNV-swapchain-parameter
  swapchain **must** be a valid VkSwapchainKHR handle
- VUID-vkLatencySleepNV-pSleepInfo-parameter
  pSleepInfo **must** be a valid pointer to a valid VkLatencySleepInfoNV structure
- VUID-vkLatencySleepNV-swapchain-parent
  swapchain **must** have been created, allocated, or retrieved from device

### Return Codes

**Success**

- VK_SUCCESS

**Failure**

None

The VkLatencySleepInfoNV structure is defined as:

```
// Provided by VK_NV_low_latency2
typedef struct VkLatencySleepInfoNV {
    VkStructureType    sType;
    const void*        pNext;
    VkSemaphore        signalSemaphore;
    uint64_t           value;
} VkLatencySleepInfoNV;
```

- sType is a VkStructureType value identifying this structure.

- pNext is NULL or a pointer to a structure extending this structure.
- signalSemaphore is a semaphore that is signaled to indicate that the application **should** resume input sampling work.
- value is the value that signalSemaphore is set to for resuming sampling work.

### Valid Usage

- VUID-VkLatencySleepInfoNV-signalSemaphore-09361
  signalSemaphore **must** be a timeline semaphore

### Valid Usage (Implicit)

- VUID-VkLatencySleepInfoNV-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_LATENCY_SLEEP_INFO_NV
- VUID-VkLatencySleepInfoNV-signalSemaphore-parameter
  signalSemaphore **must** be a valid VkSemaphore handle

An application **can** provide timestamps at various stages of its frame generation work by calling:

```
// Provided by VK_NV_low_latency2
void vkSetLatencyMarkerNV(
    VkDevice                                    device,
    VkSwapchainKHR                              swapchain,
    const VkSetLatencyMarkerInfoNV*             pLatencyMarkerInfo);
```

- device is the device associated with swapchain.
- swapchain is the swapchain to capture timestamps on.
- pSetLatencyMarkerInfo is a pointer to a VkSetLatencyMarkerInfoNV structure specifying the parameters of the marker to set.

At the beginning and end of simulation and render threads and beginning and end of vkQueuePresentKHR calls, vkSetLatencyMarkerNV **can** be called to provide timestamps for the application's reference. These timestamps are returned with a call to vkGetLatencyTimingsNV alongside driver provided timestamps at various points of interest with regards to latency within the application. As an exception to the normal rules for objects which are externally synchronized, the swapchain passed to vkSetLatencyMarkerNV **may** be simultaneously used by other threads in calls to functions other than vkDestroySwapchainKHR. Access to the swapchain data associated with this extension **must** be atomic within the implementation.

### Valid Usage (Implicit)

- VUID-vkSetLatencyMarkerNV-device-parameter
  device **must** be a valid VkDevice handle

- VUID-vkSetLatencyMarkerNV-swapchain-parameter
  `swapchain` **must** be a valid VkSwapchainKHR handle
- VUID-vkSetLatencyMarkerNV-pLatencyMarkerInfo-parameter
  `pLatencyMarkerInfo` **must** be a valid pointer to a valid VkSetLatencyMarkerInfoNV structure
- VUID-vkSetLatencyMarkerNV-swapchain-parent
  `swapchain` **must** have been created, allocated, or retrieved from `device`

The `VkSetLatencyMarkerInfoNV` structure is defined as:

```
// Provided by VK_NV_low_latency2
typedef struct VkSetLatencyMarkerInfoNV {
    VkStructureType      sType;
    const void*          pNext;
    uint64_t             presentID;
    VkLatencyMarkerNV    marker;
} VkSetLatencyMarkerInfoNV;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `presentID` is an application provided value that is used to associate the timestamp with a `vkQueuePresentKHR` command using VkPresentIdKHR::`pPresentIds` for a given present.
- `marker` is the type of timestamp to be recorded.

### Valid Usage (Implicit)

- VUID-VkSetLatencyMarkerInfoNV-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_SET_LATENCY_MARKER_INFO_NV`
- VUID-VkSetLatencyMarkerInfoNV-marker-parameter
  `marker` **must** be a valid VkLatencyMarkerNV value

The VkLatencyMarkerNV enum is defined as:

```
// Provided by VK_NV_low_latency2
typedef enum VkLatencyMarkerNV {
    VK_LATENCY_MARKER_SIMULATION_START_NV = 0,
    VK_LATENCY_MARKER_SIMULATION_END_NV = 1,
    VK_LATENCY_MARKER_RENDERSUBMIT_START_NV = 2,
    VK_LATENCY_MARKER_RENDERSUBMIT_END_NV = 3,
    VK_LATENCY_MARKER_PRESENT_START_NV = 4,
    VK_LATENCY_MARKER_PRESENT_END_NV = 5,
    VK_LATENCY_MARKER_INPUT_SAMPLE_NV = 6,
    VK_LATENCY_MARKER_TRIGGER_FLASH_NV = 7,
    VK_LATENCY_MARKER_OUT_OF_BAND_RENDERSUBMIT_START_NV = 8,
    VK_LATENCY_MARKER_OUT_OF_BAND_RENDERSUBMIT_END_NV = 9,
    VK_LATENCY_MARKER_OUT_OF_BAND_PRESENT_START_NV = 10,
    VK_LATENCY_MARKER_OUT_OF_BAND_PRESENT_END_NV = 11,
} VkLatencyMarkerNV;
```

The members of the VkLatencyMarkerNV are used as arguments for vkSetLatencyMarkerNV in the use cases described below:

- VK_LATENCY_MARKER_SIMULATION_START_NV **should** be called at the start of the simulation execution each frame, but after the call to vkLatencySleepNV.
- VK_LATENCY_MARKER_SIMULATION_END_NV **should** be called at the end of the simulation execution each frame.
- VK_LATENCY_MARKER_RENDERSUBMIT_START_NV **should** be called at the beginning of the render submission execution each frame. This **should** be wherever Vulkan API calls are made and **must** not span into asynchronous rendering.
- VK_LATENCY_MARKER_RENDERSUBMIT_END_NV **should** be called at the end of the render submission execution each frame.
- VK_LATENCY_MARKER_PRESENT_START_NV **should** be called just before vkQueuePresentKHR.
- VK_LATENCY_MARKER_PRESENT_END_NV **should** be called when vkQueuePresentKHR returns.
- VK_LATENCY_MARKER_INPUT_SAMPLE_NV **should** be called just before the application gathers input data.
- VK_LATENCY_MARKER_TRIGGER_FLASH_NV **should** be called anywhere between VK_LATENCY_MARKER_SIMULATION_START_NV and VK_LATENCY_MARKER_SIMULATION_END_NV whenever a left mouse click occurs.

To get an array containing the newest collected latency data, call:

```
// Provided by VK_NV_low_latency2
void vkGetLatencyTimingsNV(
    VkDevice                                    device,
    VkSwapchainKHR                              swapchain,
    VkGetLatencyMarkerInfoNV*                   pLatencyMarkerInfo);
```

- device is the device associated with swapchain.

- swapchain is the swapchain to return data from.
- pGetLatencyMarkerInfo is a pointer to a VkGetLatencyMarkerInfoNV structure specifying the parameters for returning latency information.

The timings returned by vkGetLatencyTimingsNV contain the timestamps requested from vkSetLatencyMarkerNV and additional implementation-specific markers defined in VkLatencyTimingsFrameReportNV.

### Valid Usage (Implicit)

- VUID-vkGetLatencyTimingsNV-device-parameter
  device **must** be a valid VkDevice handle
- VUID-vkGetLatencyTimingsNV-swapchain-parameter
  swapchain **must** be a valid VkSwapchainKHR handle
- VUID-vkGetLatencyTimingsNV-pLatencyMarkerInfo-parameter
  pLatencyMarkerInfo **must** be a valid pointer to a VkGetLatencyMarkerInfoNV structure
- VUID-vkGetLatencyTimingsNV-swapchain-parent
  swapchain **must** have been created, allocated, or retrieved from device

The VkGetLatencyMarkerInfoNV structure is defined as:

```
// Provided by VK_NV_low_latency2
typedef struct VkGetLatencyMarkerInfoNV {
    VkStructureType                     sType;
    const void*                         pNext;
    uint32_t                            timingCount;
    VkLatencyTimingsFrameReportNV*      pTimings;
} VkGetLatencyMarkerInfoNV;
```

- sType is a VkStructureType value identifying this structure.
- pNext is either NULL or a pointer to a structure extending this structure.
- timingCount is an integer related to the number of previous frames of latency data available or queried, as described below.
- pTimings is either NULL or a pointer to an array of VkLatencyTimingsFrameReportNV structures.

If pTimings is NULL then the maximum number of queryable frame data is returned in timingCount. Otherwise, timingCount **must** be set by the application to the number of elements in the pTimings array, and on return the variable is overwritten with the number of values actually written to pTimings. The elements of pTimings are arranged in the order they were requested in, with the oldest data in the first entry.

### Valid Usage (Implicit)

- VUID-VkGetLatencyMarkerInfoNV-sType-sType

sType **must** be VK_STRUCTURE_TYPE_GET_LATENCY_MARKER_INFO_NV

- VUID-VkGetLatencyMarkerInfoNV-pTimings-parameter
  If timingCount is not 0, and pTimings is not NULL, pTimings **must** be a valid pointer to an array of timingCount VkLatencyTimingsFrameReportNV structures

The VkLatencyTimingsFrameReportNV structure describes latency data returned by vkGetLatencyTimingsNV

```
// Provided by VK_NV_low_latency2
typedef struct VkLatencyTimingsFrameReportNV {
    VkStructureType    sType;
    const void*        pNext;
    uint64_t           presentID;
    uint64_t           inputSampleTimeUs;
    uint64_t           simStartTimeUs;
    uint64_t           simEndTimeUs;
    uint64_t           renderSubmitStartTimeUs;
    uint64_t           renderSubmitEndTimeUs;
    uint64_t           presentStartTimeUs;
    uint64_t           presentEndTimeUs;
    uint64_t           driverStartTimeUs;
    uint64_t           driverEndTimeUs;
    uint64_t           osRenderQueueStartTimeUs;
    uint64_t           osRenderQueueEndTimeUs;
    uint64_t           gpuRenderStartTimeUs;
    uint64_t           gpuRenderEndTimeUs;
} VkLatencyTimingsFrameReportNV;
```

The members of the VkLatencyTimingsFrameReportNV structure describe the following:

- presentID is the application provided value that is used to associate the timestamp with a vkQueuePresentKHR command using VkPresentIdKHR::pPresentIds for a given present.
- simStartTimeUs is the timestamp written when vkSetLatencyMarkerNV is called with the VkLatencyMarkerNV enum VK_LATENCY_MARKER_SIMULATION_START_NV.
- simEndTimeUs is the timestamp written when vkSetLatencyMarkerNV is called with the VkLatencyMarkerNV enum VK_LATENCY_MARKER_SIMULATION_END_NV
- renderStartTimeUs is the timestamp written when vkSetLatencyMarkerNV is called with the VkLatencyMarkerNV enum VK_LATENCY_MARKER_RENDERSUBMIT_START_NV.
- renderEndTimeUs is the timestamp written when vkSetLatencyMarkerNV is called with the VkLatencyMarkerNV enum VK_LATENCY_MARKER_RENDERSUBMIT_END_NV.
- presentStartTimeUs is the timestamp written when vkSetLatencyMarkerNV is called with the VkLatencyMarkerNV enum VK_LATENCY_MARKER_PRESENT_START_NV.
- presentEndTimeUs is the timestamp written when vkSetLatencyMarkerNV is called with the VkLatencyMarkerNV enum VK_LATENCY_MARKER_PRESENT_END_NV.
- driverStartTimeUs is the timestamp written when the first vkQueueSubmit for the frame is called.

- driverEndTimeUs is the timestamp written when the final vkQueueSubmit hands off from the Vulkan Driver.
- osRenderQueueStartTimeUs is the timestamp written when the final vkQueueSubmit hands off from the Vulkan Driver.
- osRenderQueueEndTimeUs is the timestamp written when the first submission reaches the GPU.
- gpuRenderStartTimeUs is the timestamp written when the first submission reaches the GPU.
- gpuRenderEndTimeUs is the timestamp written when the final submission finishes on the GPU for the frame.

### Valid Usage (Implicit)

- VUID-VkLatencyTimingsFrameReportNV-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_LATENCY_TIMINGS_FRAME_REPORT_NV

The VkLatencySubmissionPresentIdNV structure is defined as:

```
// Provided by VK_NV_low_latency2
typedef struct VkLatencySubmissionPresentIdNV {
    VkStructureType    sType;
    const void*        pNext;
    uint64_t           presentID;
} VkLatencySubmissionPresentIdNV;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- presentID is used to associate the vkQueueSubmit with the presentId used for a given vkQueuePresentKHR via VkPresentIdKHR::pPresentIds.

For any submission to be tracked with low latency mode pacing, it needs to be associated with other submissions in a given present. To associate a submission with presentID for low latency mode, the pNext chain of vkQueueSubmit **must** include a VkLatencySubmissionPresentIdNV structure.

### Valid Usage (Implicit)

- VUID-VkLatencySubmissionPresentIdNV-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_LATENCY_SUBMISSION_PRESENT_ID_NV

An application can mark a queue as Out of Band to indicate that all vkQueueSubmit calls on this queue are ignored for latency evaluation by calling:

```
// Provided by VK_NV_low_latency2
void vkQueueNotifyOutOfBandNV(
    VkQueue                                     queue,
    const VkOutOfBandQueueTypeInfoNV*           pQueueTypeInfo);
```

- queue is the VkQueue to be marked as out of band.
- pQueueTypeInfo is a pointer to a VkOutOfBandQueueTypeInfoNV structure specifying the queue type.

### Valid Usage (Implicit)

- VUID-vkQueueNotifyOutOfBandNV-queue-parameter
  queue **must** be a valid VkQueue handle
- VUID-vkQueueNotifyOutOfBandNV-pQueueTypeInfo-parameter
  pQueueTypeInfo **must** be a valid pointer to a valid VkOutOfBandQueueTypeInfoNV structure

### Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| - | - | - | Any | - |

The VkOutOfBandQueueTypeInfoNV structure is defined as:

```
// Provided by VK_NV_low_latency2
typedef struct VkOutOfBandQueueTypeInfoNV {
    VkStructureType           sType;
    const void*               pNext;
    VkOutOfBandQueueTypeNV    queueType;
} VkOutOfBandQueueTypeInfoNV;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- queueType describes the usage of the queue to be marked as out of band.

### Valid Usage (Implicit)

- VUID-VkOutOfBandQueueTypeInfoNV-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_OUT_OF_BAND_QUEUE_TYPE_INFO_NV
- VUID-VkOutOfBandQueueTypeInfoNV-queueType-parameter

queueType **must** be a valid VkOutOfBandQueueTypeNV value

The VkOutOfBandQueueTypeNV enum is defined as:

```
// Provided by VK_NV_low_latency2
typedef enum VkOutOfBandQueueTypeNV {
    VK_OUT_OF_BAND_QUEUE_TYPE_RENDER_NV = 0,
    VK_OUT_OF_BAND_QUEUE_TYPE_PRESENT_NV = 1,
} VkOutOfBandQueueTypeNV;
```

The members of the VkOutOfBandQueueTypeNV are used to describe the queue type in VkOutOfBandQueueTypeInfoNV as described below:

- VK_OUT_OF_BAND_QUEUE_TYPE_RENDER_NV indicates that work will be submitted to this queue.
- VK_OUT_OF_BAND_QUEUE_TYPE_PRESENT_NV indicates that this queue will be presented from.

To allow low latency mode to be used by a swapchain, add a VkSwapchainLatencyCreateInfoNV structure to the pNext chain of VkSwapchainCreateInfoKHR.

The VkSwapchainLatencyCreateInfoNV structure is defined as:

```
// Provided by VK_NV_low_latency2
typedef struct VkSwapchainLatencyCreateInfoNV {
    VkStructureType    sType;
    const void*        pNext;
    VkBool32           latencyModeEnable;
} VkSwapchainLatencyCreateInfoNV;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- lowLatencyModeEnable indicates if the swapchain created will utilize low latency mode.

### Valid Usage (Implicit)

- VUID-VkSwapchainLatencyCreateInfoNV-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_SWAPCHAIN_LATENCY_CREATE_INFO_NV

The VkLatencySurfaceCapabilitiesNV structure is defined as:

```
// Provided by VK_NV_low_latency2
typedef struct VkLatencySurfaceCapabilitiesNV {
    VkStructureType      sType;
    const void*          pNext;
    uint32_t             presentModeCount;
    VkPresentModeKHR*    pPresentModes;
} VkLatencySurfaceCapabilitiesNV;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `presentModeCount` is the number of presentation modes provided.
- `pPresentModes` is list of presentation modes optimized for use with low latency mode with `presentModeCount` entries.

If `pPresentModes` is `NULL`, then the number of present modes that are optimized for use with low latency mode returned in `presentModeCount`. Otherwise, `presentModeCount` **must** be set by the application to the number of elements in the `pPresentModes` array, and on return the variable is overwritten with the number of values actually written to `pPresentModes`. If the value of `presentModeCount` is less than the number of optimized present modes, at most `presentModeCount` values will be written to `pPresentModes`.

**Valid Usage (Implicit)**

- VUID-VkLatencySurfaceCapabilitiesNV-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_LATENCY_SURFACE_CAPABILITIES_NV`
- VUID-VkLatencySurfaceCapabilitiesNV-pPresentModes-parameter
  If `presentModeCount` is not `0`, and `pPresentModes` is not `NULL`, `pPresentModes` **must** be a valid pointer to an array of `presentModeCount` VkPresentModeKHR values

## 34.13. Present Barrier

The `VK_NV_present_barrier` extension allows applications to synchronize corresponding presentation requests across multiple swapchains using the *present barrier*. A swapchain is said to be using the *present barrier* if the swapchain is created by adding a VkSwapchainPresentBarrierCreateInfoNV structure to the `pNext` chain of the VkSwapchainCreateInfoKHR structure, and setting VkSwapchainPresentBarrierCreateInfoNV::`presentBarrierEnable` to true.

A set of corresponding presentation requests is defined as exactly one queued presentation request associated with each swapchain using the present barrier, whether or not that queued request has executed. A given presentation request is added, when created by calling vkQueuePresentKHR and specifying a swapchain using the present barrier, either to the oldest existing set of corresponding requests for which there is no existing member associated with the request's swapchain, or to a new set of corresponding requests if no such set exists.

A set of corresponding requests is said to be *full* when it contains one request from each swapchain using the present barrier. Queued presentation of an image to a swapchain using the *present barrier* is *deferred* by the implementation until the set of corresponding requests is full, and the visibility operations associated with all requests in that set, as described by vkQueuePresentKHR, have completed.

Additionally, the set of swapchains using the present barrier can be in the same process, or different processes running under the same operating system. And if the required synchronization hardware is connected and correctly configured, this extension also supports applications to synchronize corresponding presentation requests using the *present barrier* across distributed systems. However, the configuration mechanism of the required hardware is outside the scope of the Vulkan specification and this extension.

The VkSwapchainPresentBarrierCreateInfoNV structure is defined as:

```
// Provided by VK_NV_present_barrier
typedef struct VkSwapchainPresentBarrierCreateInfoNV {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           presentBarrierEnable;
} VkSwapchainPresentBarrierCreateInfoNV;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `presentBarrierEnable` is a boolean value indicating a request for using the *present barrier*.

If the `pNext` chain of VkSwapchainCreateInfoKHR does not include this structure, the default value for `presentBarrierEnable` is `VK_FALSE`, meaning the swapchain does not request to use the present barrier. Additionally, when recreating a swapchain that was using the present barrier, and the `pNext` chain of VkSwapchainCreateInfoKHR does not include this structure, it means the swapchain will stop using the present barrier.

### Valid Usage (Implicit)

- VUID-VkSwapchainPresentBarrierCreateInfoNV-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_SWAPCHAIN_PRESENT_BARRIER_CREATE_INFO_NV`

# Chapter 35. Deferred Host Operations

Certain Vulkan commands are inherently expensive for the host CPU to execute. It is often desirable to offload such work onto background threads, and to parallelize the work across multiple CPUs. The concept of *deferred operations* allows applications and drivers to coordinate the execution of expensive host commands using an application-managed thread pool.

The `VK_KHR_deferred_host_operations` extension defines the infrastructure and usage patterns for *deferrable commands*, but does not specify any commands as deferrable. This is left to additional dependent extensions. Commands **must** not be deferred unless the deferral is specifically allowed by another extension which depends on `VK_KHR_deferred_host_operations`. This specification will refer to such extensions as *deferral extensions*.

## 35.1. Requesting Deferral

When an application requests an operation deferral, the implementation **may** defer the operation. When deferral is requested and the implementation defers any operation, the implementation **must** return `VK_OPERATION_DEFERRED_KHR` as the success code if no errors occurred. When deferral is requested, the implementation **should** defer the operation when the workload is significant, however if the implementation chooses not to defer any of the requested operations and instead executes all of them immediately, the implementation **must** return `VK_OPERATION_NOT_DEFERRED_KHR` as the success code if no errors occurred.

A deferred operation is created *complete* with an initial result value of `VK_SUCCESS`. The deferred operation becomes *pending* when an operation has been successfully deferred with that deferred operation object.

A deferred operation is considered pending until the deferred operation completes. A pending deferred operation becomes *complete* when it has been fully executed by one or more threads. Pending deferred operations will never complete until they are *joined* by an application thread, using vkDeferredOperationJoinKHR. Applications **can** join multiple threads to the same deferred operation, enabling concurrent execution of subtasks within that operation.

The application **can** query the status of a VkDeferredOperationKHR using the vkGetDeferredOperationMaxConcurrencyKHR or vkGetDeferredOperationResultKHR commands.

Parameters to the command requesting a deferred operation **may** be accessed by the implementation at any time until the deferred operation enters the complete state. The application **must** obey the following rules while a deferred operation is pending:

- Externally synchronized parameters **must** not be accessed.
- Pointer parameters **must** not be modified (e.g. reallocated/freed).
- The contents of pointer parameters which **may** be read by the command **must** not be modified.
- The contents of pointer parameters which **may** be written by the command **must** not be read.
- Vulkan object parameters **must** not be passed as externally synchronized parameters to any other command.

When the deferred operation is complete, the application **should** call vkGetDeferredOperationResultKHR to obtain the VkResult indicating success or failure of the operation. The VkResult value returned will be one of the values that the command requesting the deferred operation is able to return. Writes to output parameters of the requesting command will happen-before the deferred operation is complete.

When a deferral is requested for a command, the implementation **may** perform memory management operations on the allocator supplied to vkCreateDeferredOperationKHR for the deferred operation object, as described in the Memory Allocation chapter. Such allocations **must** occur on the thread which requests deferral.

If an allocator was supplied for the deferred command at the time of the deferral request, then the implementation **may** perform memory management operations on this allocator during the execution of vkDeferredOperationJoinKHR. These operations **may** occur concurrently and **may** be performed by any joined thread. The application **must** ensure that the supplied allocator is able to operate correctly under these conditions.

## 35.2. Deferred Host Operations API

The `VkDeferredOperationKHR` handle is defined as:

```
// Provided by VK_KHR_deferred_host_operations
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkDeferredOperationKHR)
```

This handle refers to a tracking structure which manages the execution state for a deferred command.

To construct the tracking object for a deferred command, call:

```
// Provided by VK_KHR_deferred_host_operations
VkResult vkCreateDeferredOperationKHR(
    VkDevice                                    device,
    const VkAllocationCallbacks*                pAllocator,
    VkDeferredOperationKHR*                     pDeferredOperation);
```

- `device` is the device which owns `operation`.
- `pAllocator` controls host memory allocation as described in the Memory Allocation chapter.
- `pDeferredOperation` is a pointer to a handle in which the created VkDeferredOperationKHR is returned.

### Valid Usage (Implicit)

- VUID-vkCreateDeferredOperationKHR-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkCreateDeferredOperationKHR-pAllocator-parameter

If pAllocator is not NULL, pAllocator **must** be a valid pointer to a valid VkAllocationCallbacks structure

- VUID-vkCreateDeferredOperationKHR-pDeferredOperation-parameter pDeferredOperation **must** be a valid pointer to a VkDeferredOperationKHR handle

### Return Codes

**Success**

- VK_SUCCESS

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY

To assign a thread to a deferred operation, call:

```
// Provided by VK_KHR_deferred_host_operations
VkResult vkDeferredOperationJoinKHR(
    VkDevice                                    device,
    VkDeferredOperationKHR                      operation);
```

- device is the device which owns operation.
- operation is the deferred operation that the calling thread should work on.

The vkDeferredOperationJoinKHR command will execute a portion of the deferred operation on the calling thread.

The return value will be one of the following:

- A return value of VK_SUCCESS indicates that operation is complete. The application **should** use vkGetDeferredOperationResultKHR to retrieve the result of operation.
- A return value of VK_THREAD_DONE_KHR indicates that the deferred operation is not complete, but there is no work remaining to assign to threads. Future calls to vkDeferredOperationJoinKHR are not necessary and will simply harm performance. This situation **may** occur when other threads executing vkDeferredOperationJoinKHR are about to complete operation, and the implementation is unable to partition the workload any further.
- A return value of VK_THREAD_IDLE_KHR indicates that the deferred operation is not complete, and there is no work for the thread to do at the time of the call. This situation **may** occur if the operation encounters a temporary reduction in parallelism. By returning VK_THREAD_IDLE_KHR, the implementation is signaling that it expects that more opportunities for parallelism will emerge as execution progresses, and that future calls to vkDeferredOperationJoinKHR **can** be beneficial. In the meantime, the application **can** perform other work on the calling thread.

Implementations **must** guarantee forward progress by enforcing the following invariants:

1. If only one thread has invoked vkDeferredOperationJoinKHR on a given operation, that thread

**must** execute the operation to completion and return VK_SUCCESS.

2. If multiple threads have concurrently invoked vkDeferredOperationJoinKHR on the same operation, then at least one of them **must** complete the operation and return VK_SUCCESS.

### Valid Usage (Implicit)

- VUID-vkDeferredOperationJoinKHR-device-parameter
  device **must** be a valid VkDevice handle
- VUID-vkDeferredOperationJoinKHR-operation-parameter
  operation **must** be a valid VkDeferredOperationKHR handle
- VUID-vkDeferredOperationJoinKHR-operation-parent
  operation **must** have been created, allocated, or retrieved from device

### Return Codes

**Success**

- VK_SUCCESS
- VK_THREAD_DONE_KHR
- VK_THREAD_IDLE_KHR

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY

When a deferred operation is completed, the application **can** destroy the tracking object by calling:

```
// Provided by VK_KHR_deferred_host_operations
void vkDestroyDeferredOperationKHR(
    VkDevice                                    device,
    VkDeferredOperationKHR                      operation,
    const VkAllocationCallbacks*                pAllocator);
```

- device is the device which owns operation.
- operation is the completed operation to be destroyed.
- pAllocator controls host memory allocation as described in the Memory Allocation chapter.

### Valid Usage

- VUID-vkDestroyDeferredOperationKHR-operation-03434
  If VkAllocationCallbacks were provided when operation was created, a compatible set of callbacks **must** be provided here
- VUID-vkDestroyDeferredOperationKHR-operation-03435

If no VkAllocationCallbacks were provided when operation was created, pAllocator **must** be NULL

- VUID-vkDestroyDeferredOperationKHR-operation-03436
  operation **must** be completed

### Valid Usage (Implicit)

- VUID-vkDestroyDeferredOperationKHR-device-parameter
  device **must** be a valid VkDevice handle
- VUID-vkDestroyDeferredOperationKHR-operation-parameter
  If operation is not VK_NULL_HANDLE, operation **must** be a valid VkDeferredOperationKHR handle
- VUID-vkDestroyDeferredOperationKHR-pAllocator-parameter
  If pAllocator is not NULL, pAllocator **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkDestroyDeferredOperationKHR-operation-parent
  If operation is a valid handle, it **must** have been created, allocated, or retrieved from device

### Host Synchronization

- Host access to operation **must** be externally synchronized

To query the number of additional threads that can usefully be joined to a deferred operation, call:

```
// Provided by VK_KHR_deferred_host_operations
uint32_t vkGetDeferredOperationMaxConcurrencyKHR(
    VkDevice                                    device,
    VkDeferredOperationKHR                      operation);
```

- device is the device which owns operation.
- operation is the deferred operation to be queried.

The returned value is the maximum number of threads that can usefully execute a deferred operation concurrently, reported for the state of the deferred operation at the point this command is called. This value is intended to be used to better schedule work onto available threads. Applications **can** join any number of threads to the deferred operation and expect it to eventually complete, though excessive joins **may** return VK_THREAD_DONE_KHR immediately, performing no useful work.

If operation is complete, vkGetDeferredOperationMaxConcurrencyKHR returns zero.

If operation is currently joined to any threads, the value returned by this command **may**

immediately be out of date.

If `operation` is pending, implementations **must** not return zero unless at least one thread is currently executing vkDeferredOperationJoinKHR on `operation`. If there are such threads, the implementation **should** return an estimate of the number of additional threads which it could profitably use.

Implementations **may** return $2^{32}$-1 to indicate that the maximum concurrency is unknown and cannot be easily derived. Implementations **may** return values larger than the maximum concurrency available on the host CPU. In these situations, an application **should** clamp the return value rather than oversubscribing the machine.

**NOTE** The recommended usage pattern for applications is to query this value once, after deferral, and schedule no more than the specified number of threads to join the operation. Each time a joined thread receives `VK_THREAD_IDLE_KHR`, the application should schedule an additional join at some point in the future, but is not required to do so.

### Valid Usage (Implicit)

- VUID-vkGetDeferredOperationMaxConcurrencyKHR-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkGetDeferredOperationMaxConcurrencyKHR-operation-parameter
  `operation` **must** be a valid VkDeferredOperationKHR handle
- VUID-vkGetDeferredOperationMaxConcurrencyKHR-operation-parent
  `operation` **must** have been created, allocated, or retrieved from `device`

The `vkGetDeferredOperationResultKHR` function is defined as:

```
// Provided by VK_KHR_deferred_host_operations
VkResult vkGetDeferredOperationResultKHR(
    VkDevice                                    device,
    VkDeferredOperationKHR                      operation);
```

- `device` is the device which owns `operation`.
- `operation` is the operation whose deferred result is being queried.

If no command has been deferred on `operation`, `vkGetDeferredOperationResultKHR` returns `VK_SUCCESS`.

If the deferred operation is pending, `vkGetDeferredOperationResultKHR` returns `VK_NOT_READY`.

If the deferred operation is complete, it returns the appropriate return value from the original command. This value **must** be one of the VkResult values which could have been returned by the original command if the operation had not been deferred.

## Valid Usage (Implicit)

- VUID-vkGetDeferredOperationResultKHR-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkGetDeferredOperationResultKHR-operation-parameter
  `operation` **must** be a valid VkDeferredOperationKHR handle
- VUID-vkGetDeferredOperationResultKHR-operation-parent
  `operation` **must** have been created, allocated, or retrieved from `device`

## Return Codes

### Success

- `VK_SUCCESS`
- `VK_NOT_READY`

### Failure

None

# Chapter 36. Private Data

The private data extension provides a way for users to associate arbitrary application-defined data with Vulkan objects. This association is accomplished by storing 64-bit unsigned integers of application-defined data in private data slots. A private data slot represents a storage allocation for one data item for each child object of the device.

An application **can** reserve private data slots at device creation. To reserve private data slots, insert a VkDevicePrivateDataCreateInfo in the `pNext` chain in VkDeviceCreateInfo before device creation. Multiple VkDevicePrivateDataCreateInfo structures **can** be chained together, and the sum of the requested slots will be reserved. This is an exception to the specified valid usage for structure pointer chains. Reserving slots in this manner is not strictly necessary but it **may** improve performance.

Private data slots are represented by `VkPrivateDataSlot` handles:

```
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkPrivateDataSlot)
```

or the equivalent

```
// Provided by VK_EXT_private_data
typedef VkPrivateDataSlot VkPrivateDataSlotEXT;
```

To create a private data slot, call:

```
// Provided by VK_EXT_private_data
VkResult vkCreatePrivateDataSlotEXT(
    VkDevice                                    device,
    const VkPrivateDataSlotCreateInfo*          pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkPrivateDataSlot*                          pPrivateDataSlot);
```

- `device` is the logical device associated with the creation of the object(s) holding the private data slot.
- `pCreateInfo` is a pointer to a `VkPrivateDataSlotCreateInfo`
- `pAllocator` controls host memory allocation as described in the Memory Allocation chapter.
- `pPrivateDataSlot` is a pointer to a VkPrivateDataSlot handle in which the resulting private data slot is returned

### Valid Usage

- VUID-vkCreatePrivateDataSlot-privateData-04564
  The `privateData` feature **must** be enabled

### Valid Usage (Implicit)

- VUID-vkCreatePrivateDataSlot-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkCreatePrivateDataSlot-pCreateInfo-parameter
  `pCreateInfo` **must** be a valid pointer to a valid VkPrivateDataSlotCreateInfo structure
- VUID-vkCreatePrivateDataSlot-pAllocator-parameter
  If `pAllocator` is not `NULL`, `pAllocator` **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkCreatePrivateDataSlot-pPrivateDataSlot-parameter
  `pPrivateDataSlot` **must** be a valid pointer to a VkPrivateDataSlot handle

### Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- `VK_ERROR_OUT_OF_HOST_MEMORY`

The `VkPrivateDataSlotCreateInfo` structure is defined as:

```
typedef struct VkPrivateDataSlotCreateInfo {
    VkStructureType                 sType;
    const void*                     pNext;
    VkPrivateDataSlotCreateFlags    flags;
} VkPrivateDataSlotCreateInfo;
```

or the equivalent

```
// Provided by VK_EXT_private_data
typedef VkPrivateDataSlotCreateInfo VkPrivateDataSlotCreateInfoEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is reserved for future use.

### Valid Usage (Implicit)

- VUID-VkPrivateDataSlotCreateInfo-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PRIVATE_DATA_SLOT_CREATE_INFO`
- VUID-VkPrivateDataSlotCreateInfo-pNext-pNext

pNext **must** be NULL

- VUID-VkPrivateDataSlotCreateInfo-flags-zerobitmask
  flags **must** be 0

```
typedef VkFlags VkPrivateDataSlotCreateFlags;
```

or the equivalent

```
// Provided by VK_EXT_private_data
typedef VkPrivateDataSlotCreateFlags VkPrivateDataSlotCreateFlagsEXT;
```

VkPrivateDataSlotCreateFlags is a bitmask type for setting a mask, but is currently reserved for future use.

To destroy a private data slot, call:

```
// Provided by VK_EXT_private_data
void vkDestroyPrivateDataSlotEXT(
    VkDevice                                    device,
    VkPrivateDataSlot                           privateDataSlot,
    const VkAllocationCallbacks*                pAllocator);
```

- device is the logical device associated with the creation of the object(s) holding the private data slot.
- pAllocator controls host memory allocation as described in the Memory Allocation chapter.
- privateDataSlot is the private data slot to destroy.

### Valid Usage

- VUID-vkDestroyPrivateDataSlot-privateDataSlot-04062
  If VkAllocationCallbacks were provided when privateDataSlot was created, a compatible set of callbacks **must** be provided here
- VUID-vkDestroyPrivateDataSlot-privateDataSlot-04063
  If no VkAllocationCallbacks were provided when privateDataSlot was created, pAllocator **must** be NULL

### Valid Usage (Implicit)

- VUID-vkDestroyPrivateDataSlot-device-parameter
  device **must** be a valid VkDevice handle
- VUID-vkDestroyPrivateDataSlot-privateDataSlot-parameter
  If privateDataSlot is not VK_NULL_HANDLE, privateDataSlot **must** be a valid

VkPrivateDataSlot handle

- VUID-vkDestroyPrivateDataSlot-pAllocator-parameter
  If `pAllocator` is not `NULL`, `pAllocator` **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkDestroyPrivateDataSlot-privateDataSlot-parent
  If `privateDataSlot` is a valid handle, it **must** have been created, allocated, or retrieved from `device`

## Host Synchronization

- Host access to `privateDataSlot` **must** be externally synchronized

To store application-defined data in a slot associated with a Vulkan object, call:

```
// Provided by VK_EXT_private_data
VkResult vkSetPrivateDataEXT(
    VkDevice                                    device,
    VkObjectType                                objectType,
    uint64_t                                    objectHandle,
    VkPrivateDataSlot                           privateDataSlot,
    uint64_t                                    data);
```

- `device` is the device that created the object.
- `objectType` is a VkObjectType specifying the type of object to associate data with.
- `objectHandle` is a handle to the object to associate data with.
- `privateDataSlot` is a handle to a VkPrivateDataSlot specifying location of private data storage.
- `data` is application-defined data to associate the object with. This data will be stored at `privateDataSlot`.

## Valid Usage

- VUID-vkSetPrivateData-objectHandle-04016
  `objectHandle` **must** be `device` or a child of `device`
- VUID-vkSetPrivateData-objectHandle-04017
  `objectHandle` **must** be a valid handle to an object of type `objectType`

## Valid Usage (Implicit)

- VUID-vkSetPrivateData-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkSetPrivateData-objectType-parameter

objectType **must** be a valid VkObjectType value

- VUID-vkSetPrivateData-privateDataSlot-parameter
  privateDataSlot **must** be a valid VkPrivateDataSlot handle
- VUID-vkSetPrivateData-privateDataSlot-parent
  privateDataSlot **must** have been created, allocated, or retrieved from device

## Return Codes

**Success**

- VK_SUCCESS

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY

To retrieve application-defined data from a slot associated with a Vulkan object, call:

```
// Provided by VK_EXT_private_data
void vkGetPrivateDataEXT(
    VkDevice                                    device,
    VkObjectType                                objectType,
    uint64_t                                    objectHandle,
    VkPrivateDataSlot                           privateDataSlot,
    uint64_t*                                   pData);
```

- device is the device that created the object
- objectType is a VkObjectType specifying the type of object data is associated with.
- objectHandle is a handle to the object data is associated with.
- privateDataSlot is a handle to a VkPrivateDataSlot specifying location of private data pointer storage.
- pData is a pointer to specify where application-defined data is returned. 0 will be written in the absence of a previous call to vkSetPrivateDataEXT using the object specified by objectHandle.

**NOTE** Due to platform details on Android, implementations might not be able to reliably return 0 from calls to vkGetPrivateDataEXT for VkSwapchainKHR objects on which vkSetPrivateDataEXT has not previously been called. This erratum is exclusive to the Android platform and objects of type VkSwapchainKHR.

## Valid Usage

- VUID-vkGetPrivateData-objectType-04018
  objectHandle **must** be device or a child of device
- VUID-vkGetPrivateData-objectHandle-09498

`objectHandle` **must** be a valid handle to an object of type `objectType`

## Valid Usage (Implicit)

- VUID-vkGetPrivateData-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkGetPrivateData-objectType-parameter
  `objectType` **must** be a valid VkObjectType value
- VUID-vkGetPrivateData-privateDataSlot-parameter
  `privateDataSlot` **must** be a valid VkPrivateDataSlot handle
- VUID-vkGetPrivateData-pData-parameter
  `pData` **must** be a valid pointer to a `uint64_t` value
- VUID-vkGetPrivateData-privateDataSlot-parent
  `privateDataSlot` **must** have been created, allocated, or retrieved from `device`

# Chapter 37. Acceleration Structures

## 37.1. Acceleration Structures

*Acceleration structures* are data structures used by the implementation to efficiently manage scene geometry as it is traversed during a ray tracing query. The application is responsible for managing acceleration structure objects (see Acceleration Structures), including allocation, destruction, executing builds or updates, and synchronizing resources used during ray tracing queries.

There are two types of acceleration structures, *top level acceleration structures* and *bottom level acceleration structures*.

An acceleration structure is considered to be constructed if an acceleration structure build command or copy command has been executed with the given acceleration structure as the destination.

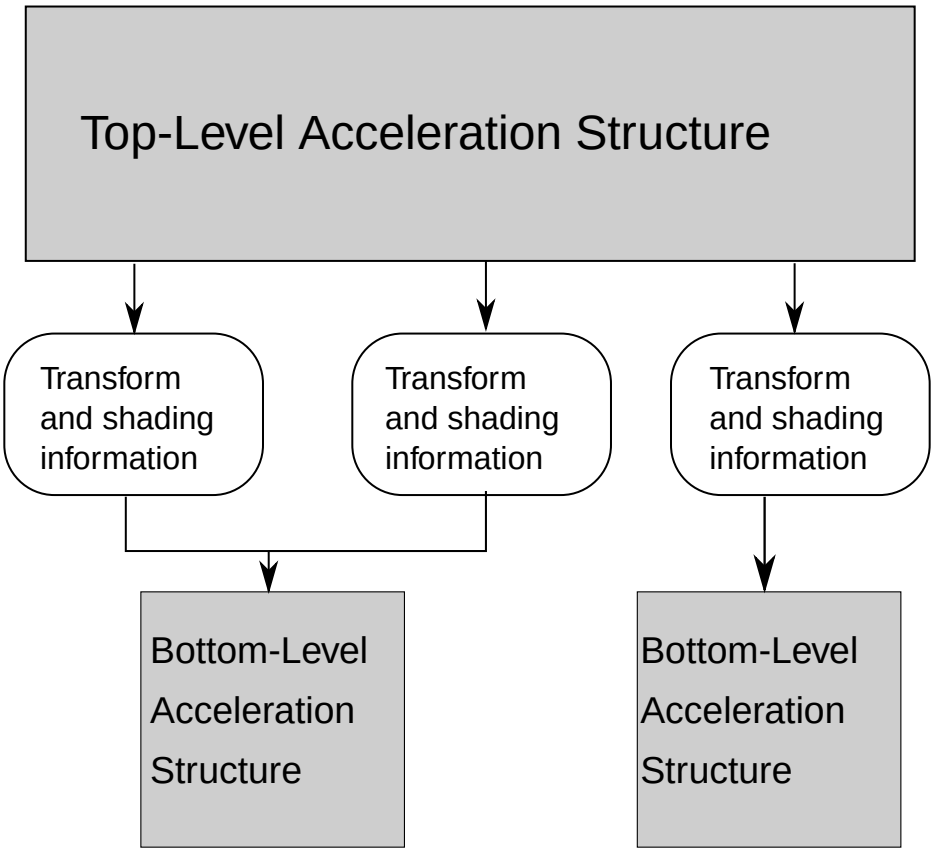



*Figure 27. Acceleration Structure*

**Caption**

The diagram shows the relationship between top and bottom level acceleration structures.

### 37.1.1. Geometry

*Geometries* refer to a triangle or axis-aligned bounding box.

### 37.1.2. Top Level Acceleration Structures

Opaque acceleration structure for an array of instances. The descriptor or device address referencing this is the starting point for traversal.

The top level acceleration structure takes a reference to any bottom level acceleration structure referenced by its instances. Those bottom level acceleration structure objects **must** be valid when the top level acceleration structure is accessed.

### 37.1.3. Bottom Level Acceleration Structures

Opaque acceleration structure for an array of geometries.

### 37.1.4. Acceleration Structure Update Rules

The API defines two types of operations to produce acceleration structures from geometry:

- A *build* operation is used to construct an acceleration structure.
- An *update* operation is used to modify an existing acceleration structure.

An update operation imposes certain constraints on the input, in exchange for considerably faster execution. When performing an update, the application is required to provide a full description of the acceleration structure, but is prohibited from changing anything other than instance definitions, transform matrices, and vertex or AABB positions. All other aspects of the description **must** exactly match the one from the original build.

More precisely, the application **must** not use an update operation to do any of the following:

- Change primitives or instances from *active* to *inactive*, or vice versa (as defined in Inactive Primitives and Instances).
- Change the index or vertex formats of triangle geometry.
- Change triangle geometry transform pointers from null to non-null or vice versa.
- Change the number of geometries or instances in the structure.
- Change the geometry flags for any geometry in the structure.
- Change the number of vertices or primitives for any geometry in the structure.

If the original acceleration structure was built using opacity micromaps and `VK_BUILD_ACCELERATION_STRUCTURE_ALLOW_OPACITY_MICROMAP_DATA_UPDATE_EXT` was set in `flags`, the application **must** provide the corresponding micromap information to the update operation. The application is prohibited from changing anything other than the specific opacity values assigned to the triangles.

More precisely, the application **must** not use an update operation to do any of the following:

- Remove micromaps or VkOpacityMicromapSpecialIndexEXT values from a geometry which previously had them, or vice versa.
- Change between use of VkOpacityMicromapSpecialIndexEXT values and explicit micro-map

triangles.

- Change the subdivision level or format of the micromap triangle associated with any acceleration-structure triangle.

If the original acceleration structure was built using opacity micromaps and `VK_BUILD_ACCELERATION_STRUCTURE_ALLOW_OPACITY_MICROMAP_UPDATE_EXT` was set in `flags`, the application **must** provide a micromap to the update operation.

If the original acceleration structure was built using opacity micromaps and neither opacity micromap update flag is set the application **must** provide the original micromap to the update operation.

If the original acceleration structure was built using displacement micromaps and `VK_BUILD_ACCELERATION_STRUCTURE_ALLOW_DISPLACEMENT_MICROMAP_UPDATE_NV` was set in `flags`, the application **must** provide a displacement micromap to the update operation.

If the original acceleration structure was built using displacement micromaps and the displacement micromap update flag is not set the application **must** provide the original micromap to the update operation.

### 37.1.5. Inactive Primitives and Instances

Acceleration structures allow the use of particular input values to signal *inactive* primitives or instances.

An *inactive* triangle is one for which the first (X) component of any vertex is NaN. If any other vertex component is NaN, and the first is not, the behavior is undefined. If the vertex format does not have a NaN representation, then all triangles are considered active.

An *inactive* instance is one whose acceleration structure reference is `0`.

An *inactive* AABB is one for which the minimum X coordinate is NaN. If any other component is NaN, and the first is not, the behavior is undefined.

In the above definitions, “NaN” refers to any type of NaN. Signaling, non-signaling, quiet, loud, or otherwise.

An inactive object is considered invisible to all rays, and **should** not be represented in the acceleration structure. Implementations **should** ensure that the presence of inactive objects does not seriously degrade traversal performance.

Inactive objects are counted in the auto-generated index sequences which are provided to shaders via `InstanceId` and `PrimitiveId` SPIR-V decorations. This allows objects in the scene to change freely between the active and inactive states, without affecting the layout of any arrays which are being indexed using the ID values.

Any transition between the active and inactive states requires a full acceleration structure rebuild. Applications **must** not perform an acceleration structure update where an object is active in the source acceleration structure but would be inactive in the destination, or vice versa.

### 37.1.6. Building Acceleration Structures

To build an acceleration structure call:

```
// Provided by VK_NV_ray_tracing
void vkCmdBuildAccelerationStructureNV(
    VkCommandBuffer                             commandBuffer,
    const VkAccelerationStructureInfoNV*        pInfo,
    VkBuffer                                    instanceData,
    VkDeviceSize                                instanceOffset,
    VkBool32                                    update,
    VkAccelerationStructureNV                   dst,
    VkAccelerationStructureNV                   src,
    VkBuffer                                    scratch,
    VkDeviceSize                                scratchOffset);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `pInfo` contains the shared information for the acceleration structure's structure.
- `instanceData` is the buffer containing an array of VkAccelerationStructureInstanceKHR structures defining acceleration structures. This parameter **must** be `NULL` for bottom level acceleration structures.
- `instanceOffset` is the offset in bytes (relative to the start of `instanceData`) at which the instance data is located.
- `update` specifies whether to update the `dst` acceleration structure with the data in `src`.
- `dst` is a pointer to the target acceleration structure for the build.
- `src` is a pointer to an existing acceleration structure that is to be used to update the `dst` acceleration structure.
- `scratch` is the VkBuffer that will be used as scratch memory for the build.
- `scratchOffset` is the offset in bytes relative to the start of `scratch` that will be used as a scratch memory.

Accesses to `dst`, `src`, and `scratch` **must** be synchronized with the `VK_PIPELINE_STAGE_ACCELERATION_STRUCTURE_BUILD_BIT_KHR` pipeline stage and an access type of `VK_ACCESS_ACCELERATION_STRUCTURE_READ_BIT_KHR` or `VK_ACCESS_ACCELERATION_STRUCTURE_WRITE_BIT_KHR`.

#### Valid Usage

- VUID-vkCmdBuildAccelerationStructureNV-geometryCount-02241
  `geometryCount` **must** be less than or equal to VkPhysicalDeviceRayTracingPropertiesNV::`maxGeometryCount`
- VUID-vkCmdBuildAccelerationStructureNV-dst-02488
  `dst` **must** have been created with compatible VkAccelerationStructureInfoNV where VkAccelerationStructureInfoNV::`type` and VkAccelerationStructureInfoNV::`flags` are identical, VkAccelerationStructureInfoNV::`instanceCount` and

VkAccelerationStructureInfoNV::geometryCount for dst are greater than or equal to the build size and each geometry in VkAccelerationStructureInfoNV::pGeometries for dst has greater than or equal to the number of vertices, indices, and AABBs

- VUID-vkCmdBuildAccelerationStructureNV-update-02489
  If update is VK_TRUE, src **must** not be VK_NULL_HANDLE
- VUID-vkCmdBuildAccelerationStructureNV-update-02490
  If update is VK_TRUE, src **must** have previously been constructed with VK_BUILD_ACCELERATION_STRUCTURE_ALLOW_UPDATE_BIT_NV set in VkAccelerationStructureInfoNV::flags in the original build
- VUID-vkCmdBuildAccelerationStructureNV-update-02491
  If update is VK_FALSE, the size member of the VkMemoryRequirements structure returned from a call to vkGetAccelerationStructureMemoryRequirementsNV with VkAccelerationStructureMemoryRequirementsInfoNV::accelerationStructure set to dst and VkAccelerationStructureMemoryRequirementsInfoNV::type set to VK_ACCELERATION_STRUCTURE_MEMORY_REQUIREMENTS_TYPE_BUILD_SCRATCH_NV **must** be less than or equal to the size of scratch minus scratchOffset
- VUID-vkCmdBuildAccelerationStructureNV-update-02492
  If update is VK_TRUE, the size member of the VkMemoryRequirements structure returned from a call to vkGetAccelerationStructureMemoryRequirementsNV with VkAccelerationStructureMemoryRequirementsInfoNV::accelerationStructure set to dst and VkAccelerationStructureMemoryRequirementsInfoNV::type set to VK_ACCELERATION_STRUCTURE_MEMORY_REQUIREMENTS_TYPE_UPDATE_SCRATCH_NV **must** be less than or equal to the size of scratch minus scratchOffset
- VUID-vkCmdBuildAccelerationStructureNV-scratch-03522
  scratch **must** have been created with VK_BUFFER_USAGE_RAY_TRACING_BIT_NV usage flag
- VUID-vkCmdBuildAccelerationStructureNV-instanceData-03523
  If instanceData is not VK_NULL_HANDLE, instanceData **must** have been created with VK_BUFFER_USAGE_RAY_TRACING_BIT_NV usage flag
- VUID-vkCmdBuildAccelerationStructureNV-accelerationStructureReference-03786
  Each VkAccelerationStructureInstanceKHR::accelerationStructureReference value in instanceData **must** be a valid device address containing a value obtained from vkGetAccelerationStructureHandleNV
- VUID-vkCmdBuildAccelerationStructureNV-update-03524
  If update is VK_TRUE, then objects that were previously active **must** not be made inactive as per Inactive Primitives and Instances
- VUID-vkCmdBuildAccelerationStructureNV-update-03525
  If update is VK_TRUE, then objects that were previously inactive **must** not be made active as per Inactive Primitives and Instances
- VUID-vkCmdBuildAccelerationStructureNV-update-03526
  If update is VK_TRUE, the src and dst objects **must** either be the same object or not have any memory aliasing
- VUID-vkCmdBuildAccelerationStructureNV-dst-07787
  dst **must** be bound completely and contiguously to a single VkDeviceMemory object via

vkBindAccelerationStructureMemoryNV

## Valid Usage (Implicit)

- VUID-vkCmdBuildAccelerationStructureNV-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdBuildAccelerationStructureNV-pInfo-parameter
  `pInfo` **must** be a valid pointer to a valid VkAccelerationStructureInfoNV structure
- VUID-vkCmdBuildAccelerationStructureNV-instanceData-parameter
  If `instanceData` is not VK_NULL_HANDLE, `instanceData` **must** be a valid VkBuffer handle
- VUID-vkCmdBuildAccelerationStructureNV-dst-parameter
  `dst` **must** be a valid VkAccelerationStructureNV handle
- VUID-vkCmdBuildAccelerationStructureNV-src-parameter
  If `src` is not VK_NULL_HANDLE, `src` **must** be a valid VkAccelerationStructureNV handle
- VUID-vkCmdBuildAccelerationStructureNV-scratch-parameter
  `scratch` **must** be a valid VkBuffer handle
- VUID-vkCmdBuildAccelerationStructureNV-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdBuildAccelerationStructureNV-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support compute operations
- VUID-vkCmdBuildAccelerationStructureNV-renderpass
  This command **must** only be called outside of a render pass instance
- VUID-vkCmdBuildAccelerationStructureNV-videocoding
  This command **must** only be called outside of a video coding scope
- VUID-vkCmdBuildAccelerationStructureNV-commonparent
  Each of `commandBuffer`, `dst`, `instanceData`, `scratch`, and `src` that are valid handles of non-ignored parameters **must** have been created, allocated, or retrieved from the same VkDevice

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

### Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Outside | Outside | Compute | Action |

To build acceleration structures call:

```
// Provided by VK_KHR_acceleration_structure
void vkCmdBuildAccelerationStructuresKHR(
    VkCommandBuffer                                     commandBuffer,
    uint32_t                                            infoCount,
    const VkAccelerationStructureBuildGeometryInfoKHR* pInfos,
    const VkAccelerationStructureBuildRangeInfoKHR* const* ppBuildRangeInfos);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `infoCount` is the number of acceleration structures to build. It specifies the number of the `pInfos` structures and `ppBuildRangeInfos` pointers that **must** be provided.
- `pInfos` is a pointer to an array of `infoCount` VkAccelerationStructureBuildGeometryInfoKHR structures defining the geometry used to build each acceleration structure.
- `ppBuildRangeInfos` is a pointer to an array of `infoCount` pointers to arrays of VkAccelerationStructureBuildRangeInfoKHR structures. Each `ppBuildRangeInfos`[i] is a pointer to an array of `pInfos`[i].`geometryCount` VkAccelerationStructureBuildRangeInfoKHR structures defining dynamic offsets to the addresses where geometry data is stored, as defined by `pInfos`[i].

The `vkCmdBuildAccelerationStructuresKHR` command provides the ability to initiate multiple acceleration structures builds, however there is no ordering or synchronization implied between any of the individual acceleration structure builds.

**NOTE** This means that an application **cannot** build a top-level acceleration structure in the same vkCmdBuildAccelerationStructuresKHR call as the associated bottom-level or instance acceleration structures are being built. There also **cannot** be any memory aliasing between any acceleration structure memories or scratch memories being used by any of the builds.

Accesses to the acceleration structure scratch buffers as identified by the VkAccelerationStructureBuildGeometryInfoKHR::`scratchData` buffer device addresses **must** be synchronized with the `VK_PIPELINE_STAGE_ACCELERATION_STRUCTURE_BUILD_BIT_KHR` pipeline stage and an access type of (`VK_ACCESS_ACCELERATION_STRUCTURE_READ_BIT_KHR` | `VK_ACCESS_ACCELERATION_STRUCTURE_WRITE_BIT_KHR`). Accesses to each VkAccelerationStructureBuildGeometryInfoKHR::`srcAccelerationStructure` and VkAccelerationStructureBuildGeometryInfoKHR::`dstAccelerationStructure` **must** be synchronized with the `VK_PIPELINE_STAGE_ACCELERATION_STRUCTURE_BUILD_BIT_KHR` pipeline stage and an access type

of VK_ACCESS_ACCELERATION_STRUCTURE_READ_BIT_KHR or VK_ACCESS_ACCELERATION_STRUCTURE_WRITE_BIT_KHR, as appropriate.

Accesses to other input buffers as identified by any used values of VkAccelerationStructureGeometryMotionTrianglesDataNV::vertexData, VkAccelerationStructureGeometryTrianglesDataKHR::vertexData, VkAccelerationStructureGeometryTrianglesDataKHR::indexData, VkAccelerationStructureGeometryTrianglesDataKHR::transformData, VkAccelerationStructureGeometryAabbsDataKHR::data, and VkAccelerationStructureGeometryInstancesDataKHR::data **must** be synchronized with the VK_PIPELINE_STAGE_ACCELERATION_STRUCTURE_BUILD_BIT_KHR pipeline stage and an access type of VK_ACCESS_SHADER_READ_BIT.

## Valid Usage

- VUID-vkCmdBuildAccelerationStructuresKHR-accelerationStructure-08923
  The VkPhysicalDeviceAccelerationStructureFeaturesKHR::accelerationStructure feature **must** be enabled
- VUID-vkCmdBuildAccelerationStructuresKHR-mode-04628
  The mode member of each element of pInfos **must** be a valid VkBuildAccelerationStructureModeKHR value
- VUID-vkCmdBuildAccelerationStructuresKHR-srcAccelerationStructure-04629
  If the srcAccelerationStructure member of any element of pInfos is not VK_NULL_HANDLE, the srcAccelerationStructure member **must** be a valid VkAccelerationStructureKHR handle
- VUID-vkCmdBuildAccelerationStructuresKHR-pInfos-04630
  For each element of pInfos, if its mode member is VK_BUILD_ACCELERATION_STRUCTURE_MODE_UPDATE_KHR, its srcAccelerationStructure member **must** not be VK_NULL_HANDLE
- VUID-vkCmdBuildAccelerationStructuresKHR-pInfos-03403
  The srcAccelerationStructure member of any element of pInfos **must** not be the same acceleration structure as the dstAccelerationStructure member of any other element of pInfos
- VUID-vkCmdBuildAccelerationStructuresKHR-dstAccelerationStructure-03698
  The dstAccelerationStructure member of any element of pInfos **must** not be the same acceleration structure as the dstAccelerationStructure member of any other element of pInfos
- VUID-vkCmdBuildAccelerationStructuresKHR-dstAccelerationStructure-03800
  The dstAccelerationStructure member of any element of pInfos **must** be a valid VkAccelerationStructureKHR handle
- VUID-vkCmdBuildAccelerationStructuresKHR-pInfos-03699
  For each element of pInfos, if its type member is VK_ACCELERATION_STRUCTURE_TYPE_TOP_LEVEL_KHR, its dstAccelerationStructure member **must** have been created with a value of VkAccelerationStructureCreateInfoKHR::type

equal to either VK_ACCELERATION_STRUCTURE_TYPE_TOP_LEVEL_KHR or VK_ACCELERATION_STRUCTURE_TYPE_GENERIC_KHR

- VUID-vkCmdBuildAccelerationStructuresKHR-pInfos-03700
  For each element of pInfos, if its type member is VK_ACCELERATION_STRUCTURE_TYPE_BOTTOM_LEVEL_KHR, its dstAccelerationStructure member **must** have been created with a value of VkAccelerationStructureCreateInfoKHR::type equal to either VK_ACCELERATION_STRUCTURE_TYPE_BOTTOM_LEVEL_KHR or VK_ACCELERATION_STRUCTURE_TYPE_GENERIC_KHR
- VUID-vkCmdBuildAccelerationStructuresKHR-pInfos-03663
  For each element of pInfos, if its mode member is VK_BUILD_ACCELERATION_STRUCTURE_MODE_UPDATE_KHR, inactive primitives in its srcAccelerationStructure member **must** not be made active
- VUID-vkCmdBuildAccelerationStructuresKHR-pInfos-03664
  For each element of pInfos, if its mode member is VK_BUILD_ACCELERATION_STRUCTURE_MODE_UPDATE_KHR, active primitives in its srcAccelerationStructure member **must** not be made inactive
- VUID-vkCmdBuildAccelerationStructuresKHR-None-03407
  The dstAccelerationStructure member of any element of pInfos **must** not be referenced by the geometry.instances.data member of any element of pGeometries or ppGeometries with a geometryType of VK_GEOMETRY_TYPE_INSTANCES_KHR in any other element of pInfos
- VUID-vkCmdBuildAccelerationStructuresKHR-dstAccelerationStructure-03701
  The range of memory backing the dstAccelerationStructure member of any element of pInfos that is accessed by this command **must** not overlap the memory backing the srcAccelerationStructure member of any other element of pInfos with a mode equal to VK_BUILD_ACCELERATION_STRUCTURE_MODE_UPDATE_KHR, which is accessed by this command
- VUID-vkCmdBuildAccelerationStructuresKHR-dstAccelerationStructure-03702
  The range of memory backing the dstAccelerationStructure member of any element of pInfos that is accessed by this command **must** not overlap the memory backing the dstAccelerationStructure member of any other element of pInfos, which is accessed by this command
- VUID-vkCmdBuildAccelerationStructuresKHR-dstAccelerationStructure-03703
  The range of memory backing the dstAccelerationStructure member of any element of pInfos that is accessed by this command **must** not overlap the memory backing the scratchData member of any element of pInfos (including the same element), which is accessed by this command
- VUID-vkCmdBuildAccelerationStructuresKHR-scratchData-03704
  The range of memory backing the scratchData member of any element of pInfos that is accessed by this command **must** not overlap the memory backing the scratchData member of any other element of pInfos, which is accessed by this command
- VUID-vkCmdBuildAccelerationStructuresKHR-scratchData-03705
  The range of memory backing the scratchData member of any element of pInfos that is accessed by this command **must** not overlap the memory backing the srcAccelerationStructure member of any element of pInfos with a mode equal to VK_BUILD_ACCELERATION_STRUCTURE_MODE_UPDATE_KHR (including the same element), which is

accessed by this command

- VUID-vkCmdBuildAccelerationStructuresKHR-dstAccelerationStructure-03706
  The range of memory backing the `dstAccelerationStructure` member of any element of `pInfos` that is accessed by this command **must** not overlap the memory backing any acceleration structure referenced by the `geometry.instances.data` member of any element of `pGeometries` or `ppGeometries` with a `geometryType` of `VK_GEOMETRY_TYPE_INSTANCES_KHR` in any other element of `pInfos`, which is accessed by this command

- VUID-vkCmdBuildAccelerationStructuresKHR-pInfos-03667
  For each element of `pInfos`, if its `mode` member is `VK_BUILD_ACCELERATION_STRUCTURE_MODE_UPDATE_KHR`, its `srcAccelerationStructure` member **must** have previously been constructed with `VK_BUILD_ACCELERATION_STRUCTURE_ALLOW_UPDATE_BIT_KHR` set in VkAccelerationStructureBuildGeometryInfoKHR::`flags` in the build

- VUID-vkCmdBuildAccelerationStructuresKHR-pInfos-03668
  For each element of `pInfos`, if its `mode` member is `VK_BUILD_ACCELERATION_STRUCTURE_MODE_UPDATE_KHR`, its `srcAccelerationStructure` and `dstAccelerationStructure` members **must** either be the same VkAccelerationStructureKHR, or not have any memory aliasing

- VUID-vkCmdBuildAccelerationStructuresKHR-pInfos-03758
  For each element of `pInfos`, if its `mode` member is `VK_BUILD_ACCELERATION_STRUCTURE_MODE_UPDATE_KHR`, its `geometryCount` member **must** have the same value which was specified when `srcAccelerationStructure` was last built

- VUID-vkCmdBuildAccelerationStructuresKHR-pInfos-03759
  For each element of `pInfos`, if its `mode` member is `VK_BUILD_ACCELERATION_STRUCTURE_MODE_UPDATE_KHR`, its `flags` member **must** have the same value which was specified when `srcAccelerationStructure` was last built

- VUID-vkCmdBuildAccelerationStructuresKHR-pInfos-03760
  For each element of `pInfos`, if its `mode` member is `VK_BUILD_ACCELERATION_STRUCTURE_MODE_UPDATE_KHR`, its `type` member **must** have the same value which was specified when `srcAccelerationStructure` was last built

- VUID-vkCmdBuildAccelerationStructuresKHR-pInfos-03761
  For each element of `pInfos`, if its `mode` member is `VK_BUILD_ACCELERATION_STRUCTURE_MODE_UPDATE_KHR`, then for each `VkAccelerationStructureGeometryKHR` structure referred to by its `pGeometries` or `ppGeometries` members, its `geometryType` member **must** have the same value which was specified when `srcAccelerationStructure` was last built

- VUID-vkCmdBuildAccelerationStructuresKHR-pInfos-03762
  For each element of `pInfos`, if its `mode` member is `VK_BUILD_ACCELERATION_STRUCTURE_MODE_UPDATE_KHR`, then for each `VkAccelerationStructureGeometryKHR` structure referred to by its `pGeometries` or `ppGeometries` members, its `flags` member **must** have the same value which was specified when `srcAccelerationStructure` was last built

- VUID-vkCmdBuildAccelerationStructuresKHR-pInfos-03763
  For each element of `pInfos`, if its `mode` member is

VK_BUILD_ACCELERATION_STRUCTURE_MODE_UPDATE_KHR, then for each VkAccelerationStructureGeometryKHR structure referred to by its pGeometries or ppGeometries members, if geometryType is VK_GEOMETRY_TYPE_TRIANGLES_KHR, its geometry.triangles.vertexFormat member **must** have the same value which was specified when srcAccelerationStructure was last built

- VUID-vkCmdBuildAccelerationStructuresKHR-pInfos-03764
  For each element of pInfos, if its mode member is VK_BUILD_ACCELERATION_STRUCTURE_MODE_UPDATE_KHR, then for each VkAccelerationStructureGeometryKHR structure referred to by its pGeometries or ppGeometries members, if geometryType is VK_GEOMETRY_TYPE_TRIANGLES_KHR, its geometry.triangles.maxVertex member **must** have the same value which was specified when srcAccelerationStructure was last built
- VUID-vkCmdBuildAccelerationStructuresKHR-pInfos-03765
  For each element of pInfos, if its mode member is VK_BUILD_ACCELERATION_STRUCTURE_MODE_UPDATE_KHR, then for each VkAccelerationStructureGeometryKHR structure referred to by its pGeometries or ppGeometries members, if geometryType is VK_GEOMETRY_TYPE_TRIANGLES_KHR, its geometry.triangles.indexType member **must** have the same value which was specified when srcAccelerationStructure was last built
- VUID-vkCmdBuildAccelerationStructuresKHR-pInfos-03766
  For each element of pInfos, if its mode member is VK_BUILD_ACCELERATION_STRUCTURE_MODE_UPDATE_KHR, then for each VkAccelerationStructureGeometryKHR structure referred to by its pGeometries or ppGeometries members, if geometryType is VK_GEOMETRY_TYPE_TRIANGLES_KHR, if its geometry.triangles.transformData address was NULL when srcAccelerationStructure was last built, then it **must** be NULL
- VUID-vkCmdBuildAccelerationStructuresKHR-pInfos-03767
  For each element of pInfos, if its mode member is VK_BUILD_ACCELERATION_STRUCTURE_MODE_UPDATE_KHR, then for each VkAccelerationStructureGeometryKHR structure referred to by its pGeometries or ppGeometries members, if geometryType is VK_GEOMETRY_TYPE_TRIANGLES_KHR, if its geometry.triangles.transformData address was not NULL when srcAccelerationStructure was last built, then it **must** not be NULL
- VUID-vkCmdBuildAccelerationStructuresKHR-pInfos-03768
  For each element of pInfos, if its mode member is VK_BUILD_ACCELERATION_STRUCTURE_MODE_UPDATE_KHR, then for each VkAccelerationStructureGeometryKHR structure referred to by its pGeometries or ppGeometries members, if geometryType is VK_GEOMETRY_TYPE_TRIANGLES_KHR, and geometry.triangles.indexType is not VK_INDEX_TYPE_NONE_KHR, then the value of each index referenced **must** be the same as the corresponding index value when srcAccelerationStructure was last built
- VUID-vkCmdBuildAccelerationStructuresKHR-primitiveCount-03769
  For each element of pInfos, if its mode member is VK_BUILD_ACCELERATION_STRUCTURE_MODE_UPDATE_KHR, then for each VkAccelerationStructureGeometryKHR structure referred to by its pGeometries or

ppGeometries members, the primitiveCount member of its corresponding VkAccelerationStructureBuildRangeInfoKHR structure **must** have the same value which was specified when srcAccelerationStructure was last built

- VUID-vkCmdBuildAccelerationStructuresKHR-pInfos-03801
  For each element of pInfos[i].pGeometries or pInfos[i].ppGeometries with a geometryType of VK_GEOMETRY_TYPE_INSTANCES_KHR, the corresponding ppBuildRangeInfos[i][j].primitiveCount **must** be less than or equal to VkPhysicalDeviceAccelerationStructurePropertiesKHR::maxInstanceCount

- VUID-vkCmdBuildAccelerationStructuresKHR-pInfos-03707
  For each element of pInfos, the buffer used to create its dstAccelerationStructure member **must** be bound to device memory

- VUID-vkCmdBuildAccelerationStructuresKHR-pInfos-03708
  For each element of pInfos, if its mode member is VK_BUILD_ACCELERATION_STRUCTURE_MODE_UPDATE_KHR the buffer used to create its srcAccelerationStructure member **must** be bound to device memory

- VUID-vkCmdBuildAccelerationStructuresKHR-pInfos-03709
  For each element of pInfos, the buffer used to create each acceleration structure referenced by the geometry.instances.data member of any element of pGeometries or ppGeometries with a geometryType of VK_GEOMETRY_TYPE_INSTANCES_KHR **must** be bound to device memory

- VUID-vkCmdBuildAccelerationStructuresKHR-pInfos-03671
  If pInfos[i].mode is VK_BUILD_ACCELERATION_STRUCTURE_MODE_BUILD_KHR, all addresses between pInfos[i].scratchData.deviceAddress and pInfos[i].scratchData.deviceAddress + N - 1 **must** be in the buffer device address range of the same buffer, where N is given by the buildScratchSize member of the VkAccelerationStructureBuildSizesInfoKHR structure returned from a call to vkGetAccelerationStructureBuildSizesKHR with an identical VkAccelerationStructureBuildGeometryInfoKHR structure and primitive count

- VUID-vkCmdBuildAccelerationStructuresKHR-pInfos-03672
  If pInfos[i].mode is VK_BUILD_ACCELERATION_STRUCTURE_MODE_UPDATE_KHR, all addresses between pInfos[i].scratchData.deviceAddress and pInfos[i].scratchData.deviceAddress + N - 1 **must** be in the buffer device address range of the same buffer, where N is given by the updateScratchSize member of the VkAccelerationStructureBuildSizesInfoKHR structure returned from a call to vkGetAccelerationStructureBuildSizesKHR with an identical VkAccelerationStructureBuildGeometryInfoKHR structure and primitive count

- VUID-vkCmdBuildAccelerationStructuresKHR-geometry-03673
  The buffers from which the buffer device addresses for all of the geometry.triangles.vertexData, geometry.triangles.indexData, geometry.triangles.transformData, geometry.aabbs.data, and geometry.instances.data members of all pInfos[i].pGeometries and pInfos[i].ppGeometries are queried **must** have been created with the VK_BUFFER_USAGE_ACCELERATION_STRUCTURE_BUILD_INPUT_READ_ONLY_BIT_KHR usage flag

- VUID-vkCmdBuildAccelerationStructuresKHR-pInfos-03674
  The buffer from which the buffer device address pInfos[i].scratchData.deviceAddress is queried **must** have been created with VK_BUFFER_USAGE_STORAGE_BUFFER_BIT usage flag

- VUID-vkCmdBuildAccelerationStructuresKHR-pInfos-03802
  For each element of pInfos, its scratchData.deviceAddress member **must** be a valid device address obtained from vkGetBufferDeviceAddressKHR
- VUID-vkCmdBuildAccelerationStructuresKHR-pInfos-03803
  For each element of pInfos, if scratchData.deviceAddress is the address of a non-sparse buffer then it **must** be bound completely and contiguously to a single VkDeviceMemory object
- VUID-vkCmdBuildAccelerationStructuresKHR-pInfos-03710
  For each element of pInfos, its scratchData.deviceAddress member **must** be a multiple of VkPhysicalDeviceAccelerationStructurePropertiesKHR::minAccelerationStructureScratchOffsetAlignment
- VUID-vkCmdBuildAccelerationStructuresKHR-pInfos-03804
  For any element of pInfos[i].pGeometries or pInfos[i].ppGeometries with a geometryType of VK_GEOMETRY_TYPE_TRIANGLES_KHR, geometry.triangles.vertexData.deviceAddress **must** be a valid device address obtained from vkGetBufferDeviceAddressKHR
- VUID-vkCmdBuildAccelerationStructuresKHR-pInfos-03805
  For any element of pInfos[i].pGeometries or pInfos[i].ppGeometries with a geometryType of VK_GEOMETRY_TYPE_TRIANGLES_KHR, if geometry.triangles.vertexData.deviceAddress is the address of a non-sparse buffer then it **must** be bound completely and contiguously to a single VkDeviceMemory object
- VUID-vkCmdBuildAccelerationStructuresKHR-pInfos-03711
  For any element of pInfos[i].pGeometries or pInfos[i].ppGeometries with a geometryType of VK_GEOMETRY_TYPE_TRIANGLES_KHR, geometry.triangles.vertexData.deviceAddress **must** be aligned to the size in bytes of the smallest component of the format in vertexFormat
- VUID-vkCmdBuildAccelerationStructuresKHR-pInfos-03806
  For any element of pInfos[i].pGeometries or pInfos[i].ppGeometries with a geometryType of VK_GEOMETRY_TYPE_TRIANGLES_KHR, if geometry.triangles.indexType is not VK_INDEX_TYPE_NONE_KHR, geometry.triangles.indexData.deviceAddress **must** be a valid device address obtained from vkGetBufferDeviceAddressKHR
- VUID-vkCmdBuildAccelerationStructuresKHR-pInfos-03807
  For any element of pInfos[i].pGeometries or pInfos[i].ppGeometries with a geometryType of VK_GEOMETRY_TYPE_TRIANGLES_KHR, if geometry.triangles.indexType is not VK_INDEX_TYPE_NONE_KHR, if geometry.triangles.indexData.deviceAddress is the address of a non-sparse buffer then it **must** be bound completely and contiguously to a single VkDeviceMemory object
- VUID-vkCmdBuildAccelerationStructuresKHR-pInfos-03712
  For any element of pInfos[i].pGeometries or pInfos[i].ppGeometries with a geometryType of VK_GEOMETRY_TYPE_TRIANGLES_KHR, and with geometry.triangles.indexType not equal to VK_INDEX_TYPE_NONE_KHR, geometry.triangles.indexData.deviceAddress **must** be aligned to the size in bytes of the type in indexType
- VUID-vkCmdBuildAccelerationStructuresKHR-pInfos-03808
  For any element of pInfos[i].pGeometries or pInfos[i].ppGeometries with a geometryType of VK_GEOMETRY_TYPE_TRIANGLES_KHR, if geometry.triangles.transformData.deviceAddress is not 0, it **must** be a valid device address obtained from vkGetBufferDeviceAddressKHR

- VUID-vkCmdBuildAccelerationStructuresKHR-pInfos-03809
  For any element of pInfos[i].pGeometries or pInfos[i].ppGeometries with a geometryType of VK_GEOMETRY_TYPE_TRIANGLES_KHR, if geometry.triangles.transformData.deviceAddress is the address of a non-sparse buffer then it **must** be bound completely and contiguously to a single VkDeviceMemory object
- VUID-vkCmdBuildAccelerationStructuresKHR-pInfos-03810
  For any element of pInfos[i].pGeometries or pInfos[i].ppGeometries with a geometryType of VK_GEOMETRY_TYPE_TRIANGLES_KHR, if geometry.triangles.transformData.deviceAddress is not 0, it **must** be aligned to 16 bytes
- VUID-vkCmdBuildAccelerationStructuresKHR-pInfos-03811
  For any element of pInfos[i].pGeometries or pInfos[i].ppGeometries with a geometryType of VK_GEOMETRY_TYPE_AABBS_KHR, geometry.aabbs.data.deviceAddress **must** be a valid device address obtained from vkGetBufferDeviceAddressKHR
- VUID-vkCmdBuildAccelerationStructuresKHR-pInfos-03812
  For any element of pInfos[i].pGeometries or pInfos[i].ppGeometries with a geometryType of VK_GEOMETRY_TYPE_AABBS_KHR, if geometry.aabbs.data.deviceAddress is the address of a non-sparse buffer then it **must** be bound completely and contiguously to a single VkDeviceMemory object
- VUID-vkCmdBuildAccelerationStructuresKHR-pInfos-03714
  For any element of pInfos[i].pGeometries or pInfos[i].ppGeometries with a geometryType of VK_GEOMETRY_TYPE_AABBS_KHR, geometry.aabbs.data.deviceAddress **must** be aligned to 8 bytes
- VUID-vkCmdBuildAccelerationStructuresKHR-pInfos-03715
  For any element of pInfos[i].pGeometries or pInfos[i].ppGeometries with a geometryType of VK_GEOMETRY_TYPE_INSTANCES_KHR, if geometry.arrayOfPointers is VK_FALSE, geometry.instances.data.deviceAddress **must** be aligned to 16 bytes
- VUID-vkCmdBuildAccelerationStructuresKHR-pInfos-03716
  For any element of pInfos[i].pGeometries or pInfos[i].ppGeometries with a geometryType of VK_GEOMETRY_TYPE_INSTANCES_KHR, if geometry.arrayOfPointers is VK_TRUE, geometry.instances.data.deviceAddress **must** be aligned to 8 bytes
- VUID-vkCmdBuildAccelerationStructuresKHR-pInfos-03717
  For any element of pInfos[i].pGeometries or pInfos[i].ppGeometries with a geometryType of VK_GEOMETRY_TYPE_INSTANCES_KHR, if geometry.arrayOfPointers is VK_TRUE, each element of geometry.instances.data.deviceAddress in device memory **must** be aligned to 16 bytes
- VUID-vkCmdBuildAccelerationStructuresKHR-pInfos-03813
  For any element of pInfos[i].pGeometries or pInfos[i].ppGeometries with a geometryType of VK_GEOMETRY_TYPE_INSTANCES_KHR, geometry.instances.data.deviceAddress **must** be a valid device address obtained from vkGetBufferDeviceAddressKHR
- VUID-vkCmdBuildAccelerationStructuresKHR-pInfos-03814
  For any element of pInfos[i].pGeometries or pInfos[i].ppGeometries with a geometryType of VK_GEOMETRY_TYPE_INSTANCES_KHR, if geometry.instances.data.deviceAddress is the address of a non-sparse buffer then it **must** be bound completely and contiguously to a single VkDeviceMemory object
- VUID-vkCmdBuildAccelerationStructuresKHR-pInfos-06707
  For any element of pInfos[i].pGeometries or pInfos[i].ppGeometries with a geometryType of

VK_GEOMETRY_TYPE_INSTANCES_KHR, each VkAccelerationStructureInstanceKHR::accelerationStructureReference value in geometry.instances.data.deviceAddress **must** be a valid device address containing a value obtained from vkGetAccelerationStructureDeviceAddressKHR or 0

- VUID-vkCmdBuildAccelerationStructuresKHR-commandBuffer-09547
  commandBuffer **must** not be a protected command buffer

- VUID-vkCmdBuildAccelerationStructuresKHR-pInfos-10126
  For each pInfos[i], dstAccelerationStructure **must** have been created with a value of VkAccelerationStructureCreateInfoKHR::size greater than or equal to either:
  - the memory size required by the build operation, as returned by vkGetAccelerationStructureBuildSizesKHR with pBuildInfo = pInfos[i] and with each element of the pMaxPrimitiveCounts array greater than or equal to the equivalent ppBuildRangeInfos[i][j].primitiveCount values for j in [0,pInfos[i].geometryCount) or,
  - the result of querying the corresponding VK_QUERY_TYPE_ACCELERATION_STRUCTURE_COMPACTED_SIZE_KHR, if updating a compacted acceleration structure

- VUID-vkCmdBuildAccelerationStructuresKHR-ppBuildRangeInfos-03676
  Each element of ppBuildRangeInfos[i] **must** be a valid pointer to an array of pInfos[i].geometryCount VkAccelerationStructureBuildRangeInfoKHR structures

## Valid Usage (Implicit)

- VUID-vkCmdBuildAccelerationStructuresKHR-commandBuffer-parameter
  commandBuffer **must** be a valid VkCommandBuffer handle

- VUID-vkCmdBuildAccelerationStructuresKHR-pInfos-parameter
  pInfos **must** be a valid pointer to an array of infoCount valid VkAccelerationStructureBuildGeometryInfoKHR structures

- VUID-vkCmdBuildAccelerationStructuresKHR-ppBuildRangeInfos-parameter
  ppBuildRangeInfos **must** be a valid pointer to an array of infoCount VkAccelerationStructureBuildRangeInfoKHR structures

- VUID-vkCmdBuildAccelerationStructuresKHR-commandBuffer-recording
  commandBuffer **must** be in the recording state

- VUID-vkCmdBuildAccelerationStructuresKHR-commandBuffer-cmdpool
  The VkCommandPool that commandBuffer was allocated from **must** support compute operations

- VUID-vkCmdBuildAccelerationStructuresKHR-renderpass
  This command **must** only be called outside of a render pass instance

- VUID-vkCmdBuildAccelerationStructuresKHR-videocoding
  This command **must** only be called outside of a video coding scope

- VUID-vkCmdBuildAccelerationStructuresKHR-infoCount-arraylength
  infoCount **must** be greater than 0

## Host Synchronization

- Host access to commandBuffer **must** be externally synchronized
- Host access to the VkCommandPool that commandBuffer was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Outside | Outside | Compute | Action |

To build acceleration structures with some parameters sourced on the device call:

```
// Provided by VK_KHR_acceleration_structure
void vkCmdBuildAccelerationStructuresIndirectKHR(
    VkCommandBuffer                                 commandBuffer,
    uint32_t                                        infoCount,
    const VkAccelerationStructureBuildGeometryInfoKHR* pInfos,
    const VkDeviceAddress*                          pIndirectDeviceAddresses,
    const uint32_t*                                 pIndirectStrides,
    const uint32_t* const*                          ppMaxPrimitiveCounts);
```

- commandBuffer is the command buffer into which the command will be recorded.
- infoCount is the number of acceleration structures to build.
- pInfos is a pointer to an array of infoCount VkAccelerationStructureBuildGeometryInfoKHR structures defining the geometry used to build each acceleration structure.
- pIndirectDeviceAddresses is a pointer to an array of infoCount buffer device addresses which point to pInfos[i].geometryCount VkAccelerationStructureBuildRangeInfoKHR structures defining dynamic offsets to the addresses where geometry data is stored, as defined by pInfos[i].
- pIndirectStrides is a pointer to an array of infoCount byte strides between elements of pIndirectDeviceAddresses.
- ppMaxPrimitiveCounts is a pointer to an array of infoCount pointers to arrays of pInfos[i].geometryCount values indicating the maximum number of primitives that will be built by this command for each geometry.

Accesses to acceleration structures, scratch buffers, vertex buffers, index buffers, and instance buffers **must** be synchronized as with vkCmdBuildAccelerationStructuresKHR.

Accesses to any element of pIndirectDeviceAddresses **must** be synchronized with the VK_PIPELINE_STAGE_ACCELERATION_STRUCTURE_BUILD_BIT_KHR pipeline stage and an access type of

VK_ACCESS_INDIRECT_COMMAND_READ_BIT.

## Valid Usage

- VUID-vkCmdBuildAccelerationStructuresIndirectKHR-accelerationStructureIndirectBuild-03650
  The VkPhysicalDeviceAccelerationStructureFeaturesKHR::accelerationStructureIndirectBuild feature **must** be enabled
- VUID-vkCmdBuildAccelerationStructuresIndirectKHR-mode-04628
  The mode member of each element of pInfos **must** be a valid VkBuildAccelerationStructureModeKHR value
- VUID-vkCmdBuildAccelerationStructuresIndirectKHR-srcAccelerationStructure-04629
  If the srcAccelerationStructure member of any element of pInfos is not VK_NULL_HANDLE, the srcAccelerationStructure member **must** be a valid VkAccelerationStructureKHR handle
- VUID-vkCmdBuildAccelerationStructuresIndirectKHR-pInfos-04630
  For each element of pInfos, if its mode member is VK_BUILD_ACCELERATION_STRUCTURE_MODE_UPDATE_KHR, its srcAccelerationStructure member **must** not be VK_NULL_HANDLE
- VUID-vkCmdBuildAccelerationStructuresIndirectKHR-pInfos-03403
  The srcAccelerationStructure member of any element of pInfos **must** not be the same acceleration structure as the dstAccelerationStructure member of any other element of pInfos
- VUID-vkCmdBuildAccelerationStructuresIndirectKHR-dstAccelerationStructure-03698
  The dstAccelerationStructure member of any element of pInfos **must** not be the same acceleration structure as the dstAccelerationStructure member of any other element of pInfos
- VUID-vkCmdBuildAccelerationStructuresIndirectKHR-dstAccelerationStructure-03800
  The dstAccelerationStructure member of any element of pInfos **must** be a valid VkAccelerationStructureKHR handle
- VUID-vkCmdBuildAccelerationStructuresIndirectKHR-pInfos-03699
  For each element of pInfos, if its type member is VK_ACCELERATION_STRUCTURE_TYPE_TOP_LEVEL_KHR, its dstAccelerationStructure member **must** have been created with a value of VkAccelerationStructureCreateInfoKHR::type equal to either VK_ACCELERATION_STRUCTURE_TYPE_TOP_LEVEL_KHR or VK_ACCELERATION_STRUCTURE_TYPE_GENERIC_KHR
- VUID-vkCmdBuildAccelerationStructuresIndirectKHR-pInfos-03700
  For each element of pInfos, if its type member is VK_ACCELERATION_STRUCTURE_TYPE_BOTTOM_LEVEL_KHR, its dstAccelerationStructure member **must** have been created with a value of VkAccelerationStructureCreateInfoKHR::type equal to either VK_ACCELERATION_STRUCTURE_TYPE_BOTTOM_LEVEL_KHR or VK_ACCELERATION_STRUCTURE_TYPE_GENERIC_KHR
- VUID-vkCmdBuildAccelerationStructuresIndirectKHR-pInfos-03663

For each element of pInfos, if its mode member is VK_BUILD_ACCELERATION_STRUCTURE_MODE_UPDATE_KHR, inactive primitives in its srcAccelerationStructure member **must** not be made active

- VUID-vkCmdBuildAccelerationStructuresIndirectKHR-pInfos-03664
  For each element of pInfos, if its mode member is VK_BUILD_ACCELERATION_STRUCTURE_MODE_UPDATE_KHR, active primitives in its srcAccelerationStructure member **must** not be made inactive
- VUID-vkCmdBuildAccelerationStructuresIndirectKHR-None-03407
  The dstAccelerationStructure member of any element of pInfos **must** not be referenced by the geometry.instances.data member of any element of pGeometries or ppGeometries with a geometryType of VK_GEOMETRY_TYPE_INSTANCES_KHR in any other element of pInfos
- VUID-vkCmdBuildAccelerationStructuresIndirectKHR-dstAccelerationStructure-03701
  The range of memory backing the dstAccelerationStructure member of any element of pInfos that is accessed by this command **must** not overlap the memory backing the srcAccelerationStructure member of any other element of pInfos with a mode equal to VK_BUILD_ACCELERATION_STRUCTURE_MODE_UPDATE_KHR, which is accessed by this command
- VUID-vkCmdBuildAccelerationStructuresIndirectKHR-dstAccelerationStructure-03702
  The range of memory backing the dstAccelerationStructure member of any element of pInfos that is accessed by this command **must** not overlap the memory backing the dstAccelerationStructure member of any other element of pInfos, which is accessed by this command
- VUID-vkCmdBuildAccelerationStructuresIndirectKHR-dstAccelerationStructure-03703
  The range of memory backing the dstAccelerationStructure member of any element of pInfos that is accessed by this command **must** not overlap the memory backing the scratchData member of any element of pInfos (including the same element), which is accessed by this command
- VUID-vkCmdBuildAccelerationStructuresIndirectKHR-scratchData-03704
  The range of memory backing the scratchData member of any element of pInfos that is accessed by this command **must** not overlap the memory backing the scratchData member of any other element of pInfos, which is accessed by this command
- VUID-vkCmdBuildAccelerationStructuresIndirectKHR-scratchData-03705
  The range of memory backing the scratchData member of any element of pInfos that is accessed by this command **must** not overlap the memory backing the srcAccelerationStructure member of any element of pInfos with a mode equal to VK_BUILD_ACCELERATION_STRUCTURE_MODE_UPDATE_KHR (including the same element), which is accessed by this command
- VUID-vkCmdBuildAccelerationStructuresIndirectKHR-dstAccelerationStructure-03706
  The range of memory backing the dstAccelerationStructure member of any element of pInfos that is accessed by this command **must** not overlap the memory backing any acceleration structure referenced by the geometry.instances.data member of any element of pGeometries or ppGeometries with a geometryType of VK_GEOMETRY_TYPE_INSTANCES_KHR in any other element of pInfos, which is accessed by this command
- VUID-vkCmdBuildAccelerationStructuresIndirectKHR-pInfos-03667
  For each element of pInfos, if its mode member is

VK_BUILD_ACCELERATION_STRUCTURE_MODE_UPDATE_KHR, its srcAccelerationStructure member **must** have previously been constructed with VK_BUILD_ACCELERATION_STRUCTURE_ALLOW_UPDATE_BIT_KHR set in VkAccelerationStructureBuildGeometryInfoKHR::flags in the build

- VUID-vkCmdBuildAccelerationStructuresIndirectKHR-pInfos-03668
  For each element of pInfos, if its mode member is VK_BUILD_ACCELERATION_STRUCTURE_MODE_UPDATE_KHR, its srcAccelerationStructure and dstAccelerationStructure members **must** either be the same VkAccelerationStructureKHR, or not have any memory aliasing
- VUID-vkCmdBuildAccelerationStructuresIndirectKHR-pInfos-03758
  For each element of pInfos, if its mode member is VK_BUILD_ACCELERATION_STRUCTURE_MODE_UPDATE_KHR, its geometryCount member **must** have the same value which was specified when srcAccelerationStructure was last built
- VUID-vkCmdBuildAccelerationStructuresIndirectKHR-pInfos-03759
  For each element of pInfos, if its mode member is VK_BUILD_ACCELERATION_STRUCTURE_MODE_UPDATE_KHR, its flags member **must** have the same value which was specified when srcAccelerationStructure was last built
- VUID-vkCmdBuildAccelerationStructuresIndirectKHR-pInfos-03760
  For each element of pInfos, if its mode member is VK_BUILD_ACCELERATION_STRUCTURE_MODE_UPDATE_KHR, its type member **must** have the same value which was specified when srcAccelerationStructure was last built
- VUID-vkCmdBuildAccelerationStructuresIndirectKHR-pInfos-03761
  For each element of pInfos, if its mode member is VK_BUILD_ACCELERATION_STRUCTURE_MODE_UPDATE_KHR, then for each VkAccelerationStructureGeometryKHR structure referred to by its pGeometries or ppGeometries members, its geometryType member **must** have the same value which was specified when srcAccelerationStructure was last built
- VUID-vkCmdBuildAccelerationStructuresIndirectKHR-pInfos-03762
  For each element of pInfos, if its mode member is VK_BUILD_ACCELERATION_STRUCTURE_MODE_UPDATE_KHR, then for each VkAccelerationStructureGeometryKHR structure referred to by its pGeometries or ppGeometries members, its flags member **must** have the same value which was specified when srcAccelerationStructure was last built
- VUID-vkCmdBuildAccelerationStructuresIndirectKHR-pInfos-03763
  For each element of pInfos, if its mode member is VK_BUILD_ACCELERATION_STRUCTURE_MODE_UPDATE_KHR, then for each VkAccelerationStructureGeometryKHR structure referred to by its pGeometries or ppGeometries members, if geometryType is VK_GEOMETRY_TYPE_TRIANGLES_KHR, its geometry.triangles.vertexFormat member **must** have the same value which was specified when srcAccelerationStructure was last built
- VUID-vkCmdBuildAccelerationStructuresIndirectKHR-pInfos-03764
  For each element of pInfos, if its mode member is VK_BUILD_ACCELERATION_STRUCTURE_MODE_UPDATE_KHR, then for each VkAccelerationStructureGeometryKHR structure referred to by its pGeometries or ppGeometries members, if geometryType is VK_GEOMETRY_TYPE_TRIANGLES_KHR, its

geometry.triangles.maxVertex member **must** have the same value which was specified when srcAccelerationStructure was last built

- VUID-vkCmdBuildAccelerationStructuresIndirectKHR-pInfos-03765
  For each element of pInfos, if its mode member is VK_BUILD_ACCELERATION_STRUCTURE_MODE_UPDATE_KHR, then for each VkAccelerationStructureGeometryKHR structure referred to by its pGeometries or ppGeometries members, if geometryType is VK_GEOMETRY_TYPE_TRIANGLES_KHR, its geometry.triangles.indexType member **must** have the same value which was specified when srcAccelerationStructure was last built
- VUID-vkCmdBuildAccelerationStructuresIndirectKHR-pInfos-03766
  For each element of pInfos, if its mode member is VK_BUILD_ACCELERATION_STRUCTURE_MODE_UPDATE_KHR, then for each VkAccelerationStructureGeometryKHR structure referred to by its pGeometries or ppGeometries members, if geometryType is VK_GEOMETRY_TYPE_TRIANGLES_KHR, if its geometry.triangles.transformData address was NULL when srcAccelerationStructure was last built, then it **must** be NULL
- VUID-vkCmdBuildAccelerationStructuresIndirectKHR-pInfos-03767
  For each element of pInfos, if its mode member is VK_BUILD_ACCELERATION_STRUCTURE_MODE_UPDATE_KHR, then for each VkAccelerationStructureGeometryKHR structure referred to by its pGeometries or ppGeometries members, if geometryType is VK_GEOMETRY_TYPE_TRIANGLES_KHR, if its geometry.triangles.transformData address was not NULL when srcAccelerationStructure was last built, then it **must** not be NULL
- VUID-vkCmdBuildAccelerationStructuresIndirectKHR-pInfos-03768
  For each element of pInfos, if its mode member is VK_BUILD_ACCELERATION_STRUCTURE_MODE_UPDATE_KHR, then for each VkAccelerationStructureGeometryKHR structure referred to by its pGeometries or ppGeometries members, if geometryType is VK_GEOMETRY_TYPE_TRIANGLES_KHR, and geometry.triangles.indexType is not VK_INDEX_TYPE_NONE_KHR, then the value of each index referenced **must** be the same as the corresponding index value when srcAccelerationStructure was last built
- VUID-vkCmdBuildAccelerationStructuresIndirectKHR-primitiveCount-03769
  For each element of pInfos, if its mode member is VK_BUILD_ACCELERATION_STRUCTURE_MODE_UPDATE_KHR, then for each VkAccelerationStructureGeometryKHR structure referred to by its pGeometries or ppGeometries members, the primitiveCount member of its corresponding VkAccelerationStructureBuildRangeInfoKHR structure **must** have the same value which was specified when srcAccelerationStructure was last built
- VUID-vkCmdBuildAccelerationStructuresIndirectKHR-pInfos-03801
  For each element of pInfos[i].pGeometries or pInfos[i].ppGeometries with a geometryType of VK_GEOMETRY_TYPE_INSTANCES_KHR, the corresponding ppMaxPrimitiveCounts[i][j] **must** be less than or equal to VkPhysicalDeviceAccelerationStructurePropertiesKHR::maxInstanceCount
- VUID-vkCmdBuildAccelerationStructuresIndirectKHR-pInfos-03707
  For each element of pInfos, the buffer used to create its dstAccelerationStructure member **must** be bound to device memory

- VUID-vkCmdBuildAccelerationStructuresIndirectKHR-pInfos-03708
  For each element of pInfos, if its mode member is VK_BUILD_ACCELERATION_STRUCTURE_MODE_UPDATE_KHR the buffer used to create its srcAccelerationStructure member **must** be bound to device memory
- VUID-vkCmdBuildAccelerationStructuresIndirectKHR-pInfos-03709
  For each element of pInfos, the buffer used to create each acceleration structure referenced by the geometry.instances.data member of any element of pGeometries or ppGeometries with a geometryType of VK_GEOMETRY_TYPE_INSTANCES_KHR **must** be bound to device memory
- VUID-vkCmdBuildAccelerationStructuresIndirectKHR-pInfos-03671
  If pInfos[i].mode is VK_BUILD_ACCELERATION_STRUCTURE_MODE_BUILD_KHR, all addresses between pInfos[i].scratchData.deviceAddress and pInfos[i].scratchData.deviceAddress + N - 1 **must** be in the buffer device address range of the same buffer, where N is given by the buildScratchSize member of the VkAccelerationStructureBuildSizesInfoKHR structure returned from a call to vkGetAccelerationStructureBuildSizesKHR with an identical VkAccelerationStructureBuildGeometryInfoKHR structure and primitive count
- VUID-vkCmdBuildAccelerationStructuresIndirectKHR-pInfos-03672
  If pInfos[i].mode is VK_BUILD_ACCELERATION_STRUCTURE_MODE_UPDATE_KHR, all addresses between pInfos[i].scratchData.deviceAddress and pInfos[i].scratchData.deviceAddress + N - 1 **must** be in the buffer device address range of the same buffer, where N is given by the updateScratchSize member of the VkAccelerationStructureBuildSizesInfoKHR structure returned from a call to vkGetAccelerationStructureBuildSizesKHR with an identical VkAccelerationStructureBuildGeometryInfoKHR structure and primitive count
- VUID-vkCmdBuildAccelerationStructuresIndirectKHR-geometry-03673
  The buffers from which the buffer device addresses for all of the geometry.triangles.vertexData, geometry.triangles.indexData, geometry.triangles.transformData, geometry.aabbs.data, and geometry.instances.data members of all pInfos[i].pGeometries and pInfos[i].ppGeometries are queried **must** have been created with the VK_BUFFER_USAGE_ACCELERATION_STRUCTURE_BUILD_INPUT_READ_ONLY_BIT_KHR usage flag
- VUID-vkCmdBuildAccelerationStructuresIndirectKHR-pInfos-03674
  The buffer from which the buffer device address pInfos[i].scratchData.deviceAddress is queried **must** have been created with VK_BUFFER_USAGE_STORAGE_BUFFER_BIT usage flag
- VUID-vkCmdBuildAccelerationStructuresIndirectKHR-pInfos-03802
  For each element of pInfos, its scratchData.deviceAddress member **must** be a valid device address obtained from vkGetBufferDeviceAddressKHR
- VUID-vkCmdBuildAccelerationStructuresIndirectKHR-pInfos-03803
  For each element of pInfos, if scratchData.deviceAddress is the address of a non-sparse buffer then it **must** be bound completely and contiguously to a single VkDeviceMemory object
- VUID-vkCmdBuildAccelerationStructuresIndirectKHR-pInfos-03710
  For each element of pInfos, its scratchData.deviceAddress member **must** be a multiple of VkPhysicalDeviceAccelerationStructurePropertiesKHR::minAccelerationStructureScratchOffsetAlignment

- VUID-vkCmdBuildAccelerationStructuresIndirectKHR-pInfos-03804
  For any element of pInfos[i].pGeometries or pInfos[i].ppGeometries with a geometryType of VK_GEOMETRY_TYPE_TRIANGLES_KHR, geometry.triangles.vertexData.deviceAddress **must** be a valid device address obtained from vkGetBufferDeviceAddressKHR
- VUID-vkCmdBuildAccelerationStructuresIndirectKHR-pInfos-03805
  For any element of pInfos[i].pGeometries or pInfos[i].ppGeometries with a geometryType of VK_GEOMETRY_TYPE_TRIANGLES_KHR, if geometry.triangles.vertexData.deviceAddress is the address of a non-sparse buffer then it **must** be bound completely and contiguously to a single VkDeviceMemory object
- VUID-vkCmdBuildAccelerationStructuresIndirectKHR-pInfos-03711
  For any element of pInfos[i].pGeometries or pInfos[i].ppGeometries with a geometryType of VK_GEOMETRY_TYPE_TRIANGLES_KHR, geometry.triangles.vertexData.deviceAddress **must** be aligned to the size in bytes of the smallest component of the format in vertexFormat
- VUID-vkCmdBuildAccelerationStructuresIndirectKHR-pInfos-03806
  For any element of pInfos[i].pGeometries or pInfos[i].ppGeometries with a geometryType of VK_GEOMETRY_TYPE_TRIANGLES_KHR, if geometry.triangles.indexType is not VK_INDEX_TYPE_NONE_KHR, geometry.triangles.indexData.deviceAddress **must** be a valid device address obtained from vkGetBufferDeviceAddressKHR
- VUID-vkCmdBuildAccelerationStructuresIndirectKHR-pInfos-03807
  For any element of pInfos[i].pGeometries or pInfos[i].ppGeometries with a geometryType of VK_GEOMETRY_TYPE_TRIANGLES_KHR, if geometry.triangles.indexType is not VK_INDEX_TYPE_NONE_KHR, if geometry.triangles.indexData.deviceAddress is the address of a non-sparse buffer then it **must** be bound completely and contiguously to a single VkDeviceMemory object
- VUID-vkCmdBuildAccelerationStructuresIndirectKHR-pInfos-03712
  For any element of pInfos[i].pGeometries or pInfos[i].ppGeometries with a geometryType of VK_GEOMETRY_TYPE_TRIANGLES_KHR, and with geometry.triangles.indexType not equal to VK_INDEX_TYPE_NONE_KHR, geometry.triangles.indexData.deviceAddress **must** be aligned to the size in bytes of the type in indexType
- VUID-vkCmdBuildAccelerationStructuresIndirectKHR-pInfos-03808
  For any element of pInfos[i].pGeometries or pInfos[i].ppGeometries with a geometryType of VK_GEOMETRY_TYPE_TRIANGLES_KHR, if geometry.triangles.transformData.deviceAddress is not 0, it **must** be a valid device address obtained from vkGetBufferDeviceAddressKHR
- VUID-vkCmdBuildAccelerationStructuresIndirectKHR-pInfos-03809
  For any element of pInfos[i].pGeometries or pInfos[i].ppGeometries with a geometryType of VK_GEOMETRY_TYPE_TRIANGLES_KHR, if geometry.triangles.transformData.deviceAddress is the address of a non-sparse buffer then it **must** be bound completely and contiguously to a single VkDeviceMemory object
- VUID-vkCmdBuildAccelerationStructuresIndirectKHR-pInfos-03810
  For any element of pInfos[i].pGeometries or pInfos[i].ppGeometries with a geometryType of VK_GEOMETRY_TYPE_TRIANGLES_KHR, if geometry.triangles.transformData.deviceAddress is not 0, it **must** be aligned to 16 bytes
- VUID-vkCmdBuildAccelerationStructuresIndirectKHR-pInfos-03811
  For any element of pInfos[i].pGeometries or pInfos[i].ppGeometries with a geometryType of

VK_GEOMETRY_TYPE_AABBS_KHR, geometry.aabbs.data.deviceAddress **must** be a valid device address obtained from vkGetBufferDeviceAddressKHR

- VUID-vkCmdBuildAccelerationStructuresIndirectKHR-pInfos-03812
  For any element of pInfos[i].pGeometries or pInfos[i].ppGeometries with a geometryType of VK_GEOMETRY_TYPE_AABBS_KHR, if geometry.aabbs.data.deviceAddress is the address of a non-sparse buffer then it **must** be bound completely and contiguously to a single VkDeviceMemory object
- VUID-vkCmdBuildAccelerationStructuresIndirectKHR-pInfos-03714
  For any element of pInfos[i].pGeometries or pInfos[i].ppGeometries with a geometryType of VK_GEOMETRY_TYPE_AABBS_KHR, geometry.aabbs.data.deviceAddress **must** be aligned to 8 bytes
- VUID-vkCmdBuildAccelerationStructuresIndirectKHR-pInfos-03715
  For any element of pInfos[i].pGeometries or pInfos[i].ppGeometries with a geometryType of VK_GEOMETRY_TYPE_INSTANCES_KHR, if geometry.arrayOfPointers is VK_FALSE, geometry.instances.data.deviceAddress **must** be aligned to 16 bytes
- VUID-vkCmdBuildAccelerationStructuresIndirectKHR-pInfos-03716
  For any element of pInfos[i].pGeometries or pInfos[i].ppGeometries with a geometryType of VK_GEOMETRY_TYPE_INSTANCES_KHR, if geometry.arrayOfPointers is VK_TRUE, geometry.instances.data.deviceAddress **must** be aligned to 8 bytes
- VUID-vkCmdBuildAccelerationStructuresIndirectKHR-pInfos-03717
  For any element of pInfos[i].pGeometries or pInfos[i].ppGeometries with a geometryType of VK_GEOMETRY_TYPE_INSTANCES_KHR, if geometry.arrayOfPointers is VK_TRUE, each element of geometry.instances.data.deviceAddress in device memory **must** be aligned to 16 bytes
- VUID-vkCmdBuildAccelerationStructuresIndirectKHR-pInfos-03813
  For any element of pInfos[i].pGeometries or pInfos[i].ppGeometries with a geometryType of VK_GEOMETRY_TYPE_INSTANCES_KHR, geometry.instances.data.deviceAddress **must** be a valid device address obtained from vkGetBufferDeviceAddressKHR
- VUID-vkCmdBuildAccelerationStructuresIndirectKHR-pInfos-03814
  For any element of pInfos[i].pGeometries or pInfos[i].ppGeometries with a geometryType of VK_GEOMETRY_TYPE_INSTANCES_KHR, if geometry.instances.data.deviceAddress is the address of a non-sparse buffer then it **must** be bound completely and contiguously to a single VkDeviceMemory object
- VUID-vkCmdBuildAccelerationStructuresIndirectKHR-pInfos-06707
  For any element of pInfos[i].pGeometries or pInfos[i].ppGeometries with a geometryType of VK_GEOMETRY_TYPE_INSTANCES_KHR, each VkAccelerationStructureInstanceKHR::accelerationStructureReference value in geometry.instances.data.deviceAddress **must** be a valid device address containing a value obtained from vkGetAccelerationStructureDeviceAddressKHR or 0
- VUID-vkCmdBuildAccelerationStructuresIndirectKHR-commandBuffer-09547
  commandBuffer **must** not be a protected command buffer
- VUID-vkCmdBuildAccelerationStructuresIndirectKHR-pIndirectDeviceAddresses-03645
  For any element of pIndirectDeviceAddresses, if the buffer from which it was queried is non-sparse then it **must** be bound completely and contiguously to a single VkDeviceMemory object

- VUID-vkCmdBuildAccelerationStructuresIndirectKHR-pIndirectDeviceAddresses-03646
  For any element of `pIndirectDeviceAddresses`[i], all device addresses between `pIndirectDeviceAddresses`[i] and `pIndirectDeviceAddresses`[i] + (`pInfos`[i].`geometryCount` × `pIndirectStrides`[i]) - 1 **must** be in the buffer device address range of the same buffer
- VUID-vkCmdBuildAccelerationStructuresIndirectKHR-pIndirectDeviceAddresses-03647
  For any element of `pIndirectDeviceAddresses`, the buffer from which it was queried **must** have been created with the `VK_BUFFER_USAGE_INDIRECT_BUFFER_BIT` bit set
- VUID-vkCmdBuildAccelerationStructuresIndirectKHR-pIndirectDeviceAddresses-03648
  Each element of `pIndirectDeviceAddresses` **must** be a multiple of `4`
- VUID-vkCmdBuildAccelerationStructuresIndirectKHR-pIndirectStrides-03787
  Each element of `pIndirectStrides` **must** be a multiple of `4`
- VUID-vkCmdBuildAccelerationStructuresIndirectKHR-pIndirectDeviceAddresses-03651
  Each VkAccelerationStructureBuildRangeInfoKHR structure referenced by any element of `pIndirectDeviceAddresses` **must** be a valid VkAccelerationStructureBuildRangeInfoKHR structure
- VUID-vkCmdBuildAccelerationStructuresIndirectKHR-pInfos-03652
  `pInfos`[i].`dstAccelerationStructure` **must** have been created with a value of VkAccelerationStructureCreateInfoKHR::`size` greater than or equal to the memory size required by the build operation, as returned by vkGetAccelerationStructureBuildSizesKHR with `pBuildInfo` = `pInfos`[i] and `pMaxPrimitiveCounts` = `ppMaxPrimitiveCounts`[i]
- VUID-vkCmdBuildAccelerationStructuresIndirectKHR-ppMaxPrimitiveCounts-03653
  Each `ppMaxPrimitiveCounts`[i][j] **must** be greater than or equal to the `primitiveCount` value specified by the VkAccelerationStructureBuildRangeInfoKHR structure located at `pIndirectDeviceAddresses`[i] + (`j` × `pIndirectStrides`[i])

## Valid Usage (Implicit)

- VUID-vkCmdBuildAccelerationStructuresIndirectKHR-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdBuildAccelerationStructuresIndirectKHR-pInfos-parameter
  `pInfos` **must** be a valid pointer to an array of `infoCount` valid VkAccelerationStructureBuildGeometryInfoKHR structures
- VUID-vkCmdBuildAccelerationStructuresIndirectKHR-pIndirectDeviceAddresses-parameter
  `pIndirectDeviceAddresses` **must** be a valid pointer to an array of `infoCount` VkDeviceAddress values
- VUID-vkCmdBuildAccelerationStructuresIndirectKHR-pIndirectStrides-parameter
  `pIndirectStrides` **must** be a valid pointer to an array of `infoCount` `uint32_t` values
- VUID-vkCmdBuildAccelerationStructuresIndirectKHR-ppMaxPrimitiveCounts-parameter
  `ppMaxPrimitiveCounts` **must** be a valid pointer to an array of `infoCount` `uint32_t` values
- VUID-vkCmdBuildAccelerationStructuresIndirectKHR-commandBuffer-recording

commandBuffer **must** be in the recording state

- VUID-vkCmdBuildAccelerationStructuresIndirectKHR-commandBuffer-cmdpool
  The VkCommandPool that commandBuffer was allocated from **must** support compute operations
- VUID-vkCmdBuildAccelerationStructuresIndirectKHR-renderpass
  This command **must** only be called outside of a render pass instance
- VUID-vkCmdBuildAccelerationStructuresIndirectKHR-videocoding
  This command **must** only be called outside of a video coding scope
- VUID-vkCmdBuildAccelerationStructuresIndirectKHR-infoCount-arraylength
  infoCount **must** be greater than 0

## Host Synchronization

- Host access to commandBuffer **must** be externally synchronized
- Host access to the VkCommandPool that commandBuffer was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Outside | Outside | Compute | Action |

The VkAccelerationStructureBuildGeometryInfoKHR structure is defined as:

```
// Provided by VK_KHR_acceleration_structure
typedef struct VkAccelerationStructureBuildGeometryInfoKHR {
    VkStructureType                                     sType;
    const void*                                         pNext;
    VkAccelerationStructureTypeKHR                      type;
    VkBuildAccelerationStructureFlagsKHR                flags;
    VkBuildAccelerationStructureModeKHR                 mode;
    VkAccelerationStructureKHR                          srcAccelerationStructure;
    VkAccelerationStructureKHR                          dstAccelerationStructure;
    uint32_t                                            geometryCount;
    const VkAccelerationStructureGeometryKHR*           pGeometries;
    const VkAccelerationStructureGeometryKHR* const*    ppGeometries;
    VkDeviceOrHostAddressKHR                            scratchData;
} VkAccelerationStructureBuildGeometryInfoKHR;
```

- sType is a VkStructureType value identifying this structure.

- pNext is NULL or a pointer to a structure extending this structure.
- type is a VkAccelerationStructureTypeKHR value specifying the type of acceleration structure being built.
- flags is a bitmask of VkBuildAccelerationStructureFlagBitsKHR specifying additional parameters of the acceleration structure.
- mode is a VkBuildAccelerationStructureModeKHR value specifying the type of operation to perform.
- srcAccelerationStructure is a pointer to an existing acceleration structure that is to be used to update the dstAccelerationStructure acceleration structure when mode is VK_BUILD_ACCELERATION_STRUCTURE_MODE_UPDATE_KHR.
- dstAccelerationStructure is a pointer to the target acceleration structure for the build.
- geometryCount specifies the number of geometries that will be built into dstAccelerationStructure.
- pGeometries is a pointer to an array of VkAccelerationStructureGeometryKHR structures.
- ppGeometries is a pointer to an array of pointers to VkAccelerationStructureGeometryKHR structures.
- scratchData is the device or host address to memory that will be used as scratch memory for the build.

Only one of pGeometries or ppGeometries **can** be a valid pointer, the other **must** be NULL. Each element of the non-NULL array describes the data used to build each acceleration structure geometry.

The index of each element of the pGeometries or ppGeometries members of VkAccelerationStructureBuildGeometryInfoKHR is used as the *geometry index* during ray traversal. The geometry index is available in ray shaders via the RayGeometryIndexKHR built-in, and is used to determine hit and intersection shaders executed during traversal. The geometry index is available to ray queries via the OpRayQueryGetIntersectionGeometryIndexKHR instruction.

Setting VK_BUILD_ACCELERATION_STRUCTURE_MOTION_BIT_NV in flags indicates that this build is a motion top level acceleration structure. A motion top level uses instances of format VkAccelerationStructureMotionInstanceNV if VkAccelerationStructureGeometryInstancesDataKHR::arrayOfPointers is VK_FALSE.

If VkAccelerationStructureGeometryInstancesDataKHR::arrayOfPointers is VK_TRUE, the pointer for each element of the array of instance pointers consists of 4 bits of VkAccelerationStructureMotionInstanceTypeNV in the low 4 bits of the pointer identifying the type of structure at the pointer. The device address accessed is the value in the array with the low 4 bits set to zero. The structure at the pointer is one of VkAccelerationStructureInstanceKHR, VkAccelerationStructureMatrixMotionInstanceNV or VkAccelerationStructureSRTMotionInstanceNV, depending on the type value encoded in the low 4 bits.

A top level acceleration structure with either motion instances or vertex motion in its instances **must** set VK_BUILD_ACCELERATION_STRUCTURE_MOTION_BIT_NV in flags.

Members srcAccelerationStructure and dstAccelerationStructure **may** be the same or different for an update operation (when mode is VK_BUILD_ACCELERATION_STRUCTURE_MODE_UPDATE_KHR). If they are the same, the update happens in-place. Otherwise, the target acceleration structure is updated and the source is not modified.

### Valid Usage

- VUID-VkAccelerationStructureBuildGeometryInfoKHR-type-03654
  type **must** not be VK_ACCELERATION_STRUCTURE_TYPE_GENERIC_KHR
- VUID-VkAccelerationStructureBuildGeometryInfoKHR-pGeometries-03788
  If geometryCount is not 0, exactly one of pGeometries or ppGeometries **must** be a valid pointer, the other **must** be NULL
- VUID-VkAccelerationStructureBuildGeometryInfoKHR-type-03789
  If type is VK_ACCELERATION_STRUCTURE_TYPE_TOP_LEVEL_KHR, the geometryType member of elements of either pGeometries or ppGeometries **must** be VK_GEOMETRY_TYPE_INSTANCES_KHR
- VUID-VkAccelerationStructureBuildGeometryInfoKHR-type-03790
  If type is VK_ACCELERATION_STRUCTURE_TYPE_TOP_LEVEL_KHR, geometryCount **must** be 1
- VUID-VkAccelerationStructureBuildGeometryInfoKHR-type-03791
  If type is VK_ACCELERATION_STRUCTURE_TYPE_BOTTOM_LEVEL_KHR the geometryType member of elements of either pGeometries or ppGeometries **must** not be VK_GEOMETRY_TYPE_INSTANCES_KHR
- VUID-VkAccelerationStructureBuildGeometryInfoKHR-type-03792
  If type is VK_ACCELERATION_STRUCTURE_TYPE_BOTTOM_LEVEL_KHR then the geometryType member of each geometry in either pGeometries or ppGeometries **must** be the same
- VUID-VkAccelerationStructureBuildGeometryInfoKHR-type-03793
  If type is VK_ACCELERATION_STRUCTURE_TYPE_BOTTOM_LEVEL_KHR then geometryCount **must** be less than or equal to VkPhysicalDeviceAccelerationStructurePropertiesKHR::maxGeometryCount
- VUID-VkAccelerationStructureBuildGeometryInfoKHR-type-03794
  If type is VK_ACCELERATION_STRUCTURE_TYPE_BOTTOM_LEVEL_KHR and the geometryType member of either pGeometries or ppGeometries is VK_GEOMETRY_TYPE_AABBS_KHR, the total number of AABBs in all geometries **must** be less than or equal to VkPhysicalDeviceAccelerationStructurePropertiesKHR::maxPrimitiveCount
- VUID-VkAccelerationStructureBuildGeometryInfoKHR-type-03795
  If type is VK_ACCELERATION_STRUCTURE_TYPE_BOTTOM_LEVEL_KHR and the geometryType member of either pGeometries or ppGeometries is VK_GEOMETRY_TYPE_TRIANGLES_KHR, the total number of triangles in all geometries **must** be less than or equal to VkPhysicalDeviceAccelerationStructurePropertiesKHR::maxPrimitiveCount
- VUID-VkAccelerationStructureBuildGeometryInfoKHR-flags-03796
  If flags has the VK_BUILD_ACCELERATION_STRUCTURE_PREFER_FAST_TRACE_BIT_KHR bit set, then it **must** not have the VK_BUILD_ACCELERATION_STRUCTURE_PREFER_FAST_BUILD_BIT_KHR bit set
- VUID-VkAccelerationStructureBuildGeometryInfoKHR-dstAccelerationStructure-04927
  If dstAccelerationStructure was created with

VK_ACCELERATION_STRUCTURE_CREATE_MOTION_BIT_NV set in VkAccelerationStructureCreateInfoKHR::flags, VK_BUILD_ACCELERATION_STRUCTURE_MOTION_BIT_NV **must** be set in flags

- VUID-VkAccelerationStructureBuildGeometryInfoKHR-flags-04928
  If VK_BUILD_ACCELERATION_STRUCTURE_MOTION_BIT_NV is set in flags, dstAccelerationStructure **must** have been created with VK_ACCELERATION_STRUCTURE_CREATE_MOTION_BIT_NV set in VkAccelerationStructureCreateInfoKHR::flags
- VUID-VkAccelerationStructureBuildGeometryInfoKHR-flags-04929
  If VK_BUILD_ACCELERATION_STRUCTURE_MOTION_BIT_NV is set in flags, type **must** not be VK_ACCELERATION_STRUCTURE_TYPE_GENERIC_KHR
- VUID-VkAccelerationStructureBuildGeometryInfoKHR-flags-07334
  If flags has the VK_BUILD_ACCELERATION_STRUCTURE_ALLOW_OPACITY_MICROMAP_UPDATE_EXT bit set then it **must** not have the VK_BUILD_ACCELERATION_STRUCTURE_ALLOW_OPACITY_MICROMAP_DATA_UPDATE_EXT bit set

## Valid Usage (Implicit)

- VUID-VkAccelerationStructureBuildGeometryInfoKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_ACCELERATION_STRUCTURE_BUILD_GEOMETRY_INFO_KHR
- VUID-VkAccelerationStructureBuildGeometryInfoKHR-pNext-pNext
  pNext **must** be NULL
- VUID-VkAccelerationStructureBuildGeometryInfoKHR-type-parameter
  type **must** be a valid VkAccelerationStructureTypeKHR value
- VUID-VkAccelerationStructureBuildGeometryInfoKHR-flags-parameter
  flags **must** be a valid combination of VkBuildAccelerationStructureFlagBitsKHR values
- VUID-VkAccelerationStructureBuildGeometryInfoKHR-pGeometries-parameter
  If geometryCount is not 0, and pGeometries is not NULL, pGeometries **must** be a valid pointer to an array of geometryCount valid VkAccelerationStructureGeometryKHR structures
- VUID-VkAccelerationStructureBuildGeometryInfoKHR-ppGeometries-parameter
  If geometryCount is not 0, and ppGeometries is not NULL, ppGeometries **must** be a valid pointer to an array of geometryCount valid pointers to valid VkAccelerationStructureGeometryKHR structures
- VUID-VkAccelerationStructureBuildGeometryInfoKHR-commonparent
  Both of dstAccelerationStructure, and srcAccelerationStructure that are valid handles of non-ignored parameters **must** have been created, allocated, or retrieved from the same VkDevice

The VkBuildAccelerationStructureModeKHR enumeration is defined as:

```
// Provided by VK_KHR_acceleration_structure
typedef enum VkBuildAccelerationStructureModeKHR {
    VK_BUILD_ACCELERATION_STRUCTURE_MODE_BUILD_KHR = 0,
    VK_BUILD_ACCELERATION_STRUCTURE_MODE_UPDATE_KHR = 1,
} VkBuildAccelerationStructureModeKHR;
```

- `VK_BUILD_ACCELERATION_STRUCTURE_MODE_BUILD_KHR` specifies that the destination acceleration structure will be built using the specified geometries.
- `VK_BUILD_ACCELERATION_STRUCTURE_MODE_UPDATE_KHR` specifies that the destination acceleration structure will be built using data in a source acceleration structure, updated by the specified geometries.

The `VkDeviceOrHostAddressKHR` union is defined as:

```
// Provided by VK_KHR_acceleration_structure
typedef union VkDeviceOrHostAddressKHR {
    VkDeviceAddress    deviceAddress;
    void*              hostAddress;
} VkDeviceOrHostAddressKHR;
```

- `deviceAddress` is a buffer device address as returned by the vkGetBufferDeviceAddressKHR command.
- `hostAddress` is a host memory address.

The `VkDeviceOrHostAddressConstKHR` union is defined as:

```
// Provided by VK_KHR_acceleration_structure
typedef union VkDeviceOrHostAddressConstKHR {
    VkDeviceAddress    deviceAddress;
    const void*        hostAddress;
} VkDeviceOrHostAddressConstKHR;
```

- `deviceAddress` is a buffer device address as returned by the vkGetBufferDeviceAddressKHR command.
- `hostAddress` is a const host memory address.

The `VkAccelerationStructureGeometryKHR` structure is defined as:

```
// Provided by VK_KHR_acceleration_structure
typedef struct VkAccelerationStructureGeometryKHR {
    VkStructureType                                 sType;
    const void*                                     pNext;
    VkGeometryTypeKHR                               geometryType;
    VkAccelerationStructureGeometryDataKHR          geometry;
    VkGeometryFlagsKHR                              flags;
} VkAccelerationStructureGeometryKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `geometryType` describes which type of geometry this `VkAccelerationStructureGeometryKHR` refers to.
- `geometry` is a VkAccelerationStructureGeometryDataKHR union describing the geometry data for the relevant geometry type.
- `flags` is a bitmask of VkGeometryFlagBitsKHR values describing additional properties of how the geometry should be built.

### Valid Usage (Implicit)

- VUID-VkAccelerationStructureGeometryKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_ACCELERATION_STRUCTURE_GEOMETRY_KHR`
- VUID-VkAccelerationStructureGeometryKHR-pNext-pNext
  `pNext` **must** be `NULL`
- VUID-VkAccelerationStructureGeometryKHR-geometryType-parameter
  `geometryType` **must** be a valid VkGeometryTypeKHR value
- VUID-VkAccelerationStructureGeometryKHR-triangles-parameter
  If `geometryType` is `VK_GEOMETRY_TYPE_TRIANGLES_KHR`, the `triangles` member of `geometry` **must** be a valid VkAccelerationStructureGeometryTrianglesDataKHR structure
- VUID-VkAccelerationStructureGeometryKHR-aabbs-parameter
  If `geometryType` is `VK_GEOMETRY_TYPE_AABBS_KHR`, the `aabbs` member of `geometry` **must** be a valid VkAccelerationStructureGeometryAabbsDataKHR structure
- VUID-VkAccelerationStructureGeometryKHR-instances-parameter
  If `geometryType` is `VK_GEOMETRY_TYPE_INSTANCES_KHR`, the `instances` member of `geometry` **must** be a valid VkAccelerationStructureGeometryInstancesDataKHR structure
- VUID-VkAccelerationStructureGeometryKHR-flags-parameter
  `flags` **must** be a valid combination of VkGeometryFlagBitsKHR values

The `VkAccelerationStructureGeometryDataKHR` union is defined as:

```
// Provided by VK_KHR_acceleration_structure
typedef union VkAccelerationStructureGeometryDataKHR {
    VkAccelerationStructureGeometryTrianglesDataKHR    triangles;
    VkAccelerationStructureGeometryAabbsDataKHR        aabbs;
    VkAccelerationStructureGeometryInstancesDataKHR    instances;
} VkAccelerationStructureGeometryDataKHR;
```

- triangles is a VkAccelerationStructureGeometryTrianglesDataKHR structure.
- aabbs is a VkAccelerationStructureGeometryAabbsDataKHR structure.
- instances is a VkAccelerationStructureGeometryInstancesDataKHR structure.

The VkAccelerationStructureGeometryTrianglesDataKHR structure is defined as:

```
// Provided by VK_KHR_acceleration_structure
typedef struct VkAccelerationStructureGeometryTrianglesDataKHR {
    VkStructureType                  sType;
    const void*                      pNext;
    VkFormat                         vertexFormat;
    VkDeviceOrHostAddressConstKHR    vertexData;
    VkDeviceSize                     vertexStride;
    uint32_t                         maxVertex;
    VkIndexType                      indexType;
    VkDeviceOrHostAddressConstKHR    indexData;
    VkDeviceOrHostAddressConstKHR    transformData;
} VkAccelerationStructureGeometryTrianglesDataKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- vertexFormat is the VkFormat of each vertex element.
- vertexData is a device or host address to memory containing vertex data for this geometry.
- maxVertex is the number of vertices in vertexData minus one.
- vertexStride is the stride in bytes between each vertex.
- indexType is the VkIndexType of each index element.
- indexData is a device or host address to memory containing index data for this geometry.
- transformData is a device or host address to memory containing an optional reference to a VkTransformMatrixKHR structure describing a transformation from the space in which the vertices in this geometry are described to the space in which the acceleration structure is defined.

**NOTE** Unlike the stride for vertex buffers in VkVertexInputBindingDescription for graphics pipelines which must not exceed maxVertexInputBindingStride, vertexStride for acceleration structure geometry is instead restricted to being a 32-bit value.

## Valid Usage

- VUID-VkAccelerationStructureGeometryTrianglesDataKHR-vertexStride-03735
  `vertexStride` **must** be a multiple of the size in bytes of the smallest component of `vertexFormat`
- VUID-VkAccelerationStructureGeometryTrianglesDataKHR-vertexStride-03819
  `vertexStride` **must** be less than or equal to $2^{32}$-1
- VUID-VkAccelerationStructureGeometryTrianglesDataKHR-vertexFormat-03797
  The format features of `vertexFormat` **must** contain `VK_FORMAT_FEATURE_ACCELERATION_STRUCTURE_VERTEX_BUFFER_BIT_KHR`
- VUID-VkAccelerationStructureGeometryTrianglesDataKHR-indexType-03798
  `indexType` **must** be `VK_INDEX_TYPE_UINT16`, `VK_INDEX_TYPE_UINT32`, or `VK_INDEX_TYPE_NONE_KHR`

## Valid Usage (Implicit)

- VUID-VkAccelerationStructureGeometryTrianglesDataKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_ACCELERATION_STRUCTURE_GEOMETRY_TRIANGLES_DATA_KHR`
- VUID-VkAccelerationStructureGeometryTrianglesDataKHR-pNext-pNext
  Each `pNext` member of any structure (including this one) in the `pNext` chain **must** be either `NULL` or a pointer to a valid instance of VkAccelerationStructureGeometryMotionTrianglesDataNV, VkAccelerationStructureTrianglesDisplacementMicromapNV, or VkAccelerationStructureTrianglesOpacityMicromapEXT
- VUID-VkAccelerationStructureGeometryTrianglesDataKHR-sType-unique
  The `sType` value of each struct in the `pNext` chain **must** be unique
- VUID-VkAccelerationStructureGeometryTrianglesDataKHR-vertexFormat-parameter
  `vertexFormat` **must** be a valid VkFormat value
- VUID-VkAccelerationStructureGeometryTrianglesDataKHR-indexType-parameter
  `indexType` **must** be a valid VkIndexType value

The `VkAccelerationStructureGeometryMotionTrianglesDataNV` structure is defined as:

```
// Provided by VK_NV_ray_tracing_motion_blur
typedef struct VkAccelerationStructureGeometryMotionTrianglesDataNV {
    VkStructureType                 sType;
    const void*                     pNext;
    VkDeviceOrHostAddressConstKHR   vertexData;
} VkAccelerationStructureGeometryMotionTrianglesDataNV;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.

- vertexData is a pointer to vertex data for this geometry at time 1.0

If VkAccelerationStructureGeometryMotionTrianglesDataNV is included in the pNext chain of a VkAccelerationStructureGeometryTrianglesDataKHR structure, the basic vertex positions are used for the position of the triangles in the geometry at time 0.0 and the vertexData in VkAccelerationStructureGeometryMotionTrianglesDataNV is used for the vertex positions at time 1.0, with positions linearly interpolated at intermediate times.

Indexing for VkAccelerationStructureGeometryMotionTrianglesDataNV vertexData is equivalent to the basic vertex position data.

### Valid Usage (Implicit)

- VUID-VkAccelerationStructureGeometryMotionTrianglesDataNV-sType-sType sType **must** be VK_STRUCTURE_TYPE_ACCELERATION_STRUCTURE_GEOMETRY_MOTION_TRIANGLES_DATA_NV

The VkAccelerationStructureTrianglesOpacityMicromapEXT structure is defined as:

```
// Provided by VK_EXT_opacity_micromap
typedef struct VkAccelerationStructureTrianglesOpacityMicromapEXT {
    VkStructureType                    sType;
    void*                              pNext;
    VkIndexType                        indexType;
    VkDeviceOrHostAddressConstKHR      indexBuffer;
    VkDeviceSize                       indexStride;
    uint32_t                           baseTriangle;
    uint32_t                           usageCountsCount;
    const VkMicromapUsageEXT*          pUsageCounts;
    const VkMicromapUsageEXT* const*   ppUsageCounts;
    VkMicromapEXT                      micromap;
} VkAccelerationStructureTrianglesOpacityMicromapEXT;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- indexType is the type of triangle indices used when indexing this micromap
- indexBuffer is the address containing the triangle indices
- indexStride is the byte stride between triangle indices
- baseTriangle is the base value added to the non-negative triangle indices
- usageCountsCount specifies the number of usage counts structures that will be used to determine the size of this micromap.
- pUsageCounts is a pointer to an array of VkMicromapUsageEXT structures.
- ppUsageCounts is a pointer to an array of pointers to VkMicromapUsageEXT structures.
- micromap is the handle to the micromap object to include in this geometry

If VkAccelerationStructureTrianglesOpacityMicromapEXT is included in the pNext chain of a VkAccelerationStructureGeometryTrianglesDataKHR structure, that geometry will reference that micromap.

For each triangle in the geometry, the acceleration structure build fetches an index from indexBuffer using indexType and indexStride. If that value is the unsigned cast of one of the values from VkOpacityMicromapSpecialIndexEXT then that triangle behaves as described for that special value in Ray Opacity Micromap. Otherwise that triangle uses the opacity micromap information from micromap at that index plus baseTriangle.

Only one of pUsageCounts or ppUsageCounts **can** be a valid pointer, the other **must** be NULL. The elements of the non-NULL array describe the total count used to build this geometry. For a given format and subdivisionLevel the number of triangles in this geometry matching those values after indirection and special index handling **must** be equal to the sum of matching count provided.

If micromap is VK_NULL_HANDLE, then every value read from indexBuffer **must** be one of the values in VkOpacityMicromapSpecialIndexEXT.

### Valid Usage

- VUID-VkAccelerationStructureTrianglesOpacityMicromapEXT-pUsageCounts-07335
  Only one of pUsageCounts or ppUsageCounts **can** be a valid pointer, the other **must** be NULL

### Valid Usage (Implicit)

- VUID-VkAccelerationStructureTrianglesOpacityMicromapEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_ACCELERATION_STRUCTURE_TRIANGLES_OPACITY_MICROMAP_EXT
- VUID-VkAccelerationStructureTrianglesOpacityMicromapEXT-indexType-parameter
  indexType **must** be a valid VkIndexType value
- VUID-VkAccelerationStructureTrianglesOpacityMicromapEXT-pUsageCounts-parameter
  If usageCountsCount is not 0, and pUsageCounts is not NULL, pUsageCounts **must** be a valid pointer to an array of usageCountsCount VkMicromapUsageEXT structures
- VUID-VkAccelerationStructureTrianglesOpacityMicromapEXT-ppUsageCounts-parameter
  If usageCountsCount is not 0, and ppUsageCounts is not NULL, ppUsageCounts **must** be a valid pointer to an array of usageCountsCount valid pointers to VkMicromapUsageEXT structures
- VUID-VkAccelerationStructureTrianglesOpacityMicromapEXT-micromap-parameter
  If micromap is not VK_NULL_HANDLE, micromap **must** be a valid VkMicromapEXT handle

The VkOpacityMicromapSpecialIndexEXT enumeration is defined as:

```
// Provided by VK_EXT_opacity_micromap
typedef enum VkOpacityMicromapSpecialIndexEXT {
    VK_OPACITY_MICROMAP_SPECIAL_INDEX_FULLY_TRANSPARENT_EXT = -1,
    VK_OPACITY_MICROMAP_SPECIAL_INDEX_FULLY_OPAQUE_EXT = -2,
    VK_OPACITY_MICROMAP_SPECIAL_INDEX_FULLY_UNKNOWN_TRANSPARENT_EXT = -3,
    VK_OPACITY_MICROMAP_SPECIAL_INDEX_FULLY_UNKNOWN_OPAQUE_EXT = -4,
} VkOpacityMicromapSpecialIndexEXT;
```

- VK_OPACITY_MICROMAP_SPECIAL_INDEX_FULLY_TRANSPARENT_EXT specifies that the entire triangle is fully transparent.
- VK_OPACITY_MICROMAP_SPECIAL_INDEX_FULLY_OPAQUE_EXT specifies that the entire triangle is fully opaque.
- VK_OPACITY_MICROMAP_SPECIAL_INDEX_FULLY_UNKNOWN_TRANSPARENT_EXT specifies that the entire triangle is unknown-transparent.
- VK_OPACITY_MICROMAP_SPECIAL_INDEX_FULLY_UNKNOWN_OPAQUE_EXT specifies that the entire triangle is unknown-opaque.

The VkAccelerationStructureTrianglesDisplacementMicromapNV structure is defined as:

```
// Provided by VK_NV_displacement_micromap
typedef struct VkAccelerationStructureTrianglesDisplacementMicromapNV {
    VkStructureType                     sType;
    void*                               pNext;
    VkFormat                            displacementBiasAndScaleFormat;
    VkFormat                            displacementVectorFormat;
    VkDeviceOrHostAddressConstKHR       displacementBiasAndScaleBuffer;
    VkDeviceSize                        displacementBiasAndScaleStride;
    VkDeviceOrHostAddressConstKHR       displacementVectorBuffer;
    VkDeviceSize                        displacementVectorStride;
    VkDeviceOrHostAddressConstKHR       displacedMicromapPrimitiveFlags;
    VkDeviceSize                        displacedMicromapPrimitiveFlagsStride;
    VkIndexType                         indexType;
    VkDeviceOrHostAddressConstKHR       indexBuffer;
    VkDeviceSize                        indexStride;
    uint32_t                            baseTriangle;
    uint32_t                            usageCountsCount;
    const VkMicromapUsageEXT*           pUsageCounts;
    const VkMicromapUsageEXT* const*    ppUsageCounts;
    VkMicromapEXT                       micromap;
} VkAccelerationStructureTrianglesDisplacementMicromapNV;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- displacementBiasAndScaleFormat is the format of displacement bias and scale used in this displacement micromap reference.

- displacementVectorFormat is the format of displacement vector used in this displacement micromap reference.
- displacementBiasAndScaleBuffer is the address containing the bias and scale.
- displacementBiasAndScaleStride is the byte stride between bias and scale values.
- displacementVectorBuffer is the address containing the displacement vector values.
- displacementVectorStride is the byte stride between displacement vector values.
- displacedMicromapPrimitiveFlags is the address containing the primitive flags.
- displacedMicromapPrimitiveFlagsStride is the byte stride between primitive flag values.
- indexType is the type of triangle indices used when indexing this micromap.
- indexBuffer is the address containing the triangle indices.
- indexStride is the byte stride between triangle indices.
- baseTriangle is the base value added to the non-negative triangle indices.
- usageCountsCount specifies the number of usage counts structures that will be used to determine the size of this micromap.
- pUsageCounts is a pointer to an array of VkMicromapUsageEXT structures.
- ppUsageCounts is a pointer to an array of pointers to VkMicromapUsageEXT structures.
- micromap is the handle to the micromap object to include in this geometry.

If VkAccelerationStructureTrianglesDisplacementMicromapNV is included in the pNext chain of a VkAccelerationStructureGeometryTrianglesDataKHR structure, that geometry will reference that micromap.

For each triangle in the geometry, the acceleration structure build fetches an index from indexBuffer using indexType and indexStride. That triangle uses the displacement micromap information from micromap at that index plus baseTriangle.

Only one of pUsageCounts or ppUsageCounts **can** be a valid pointer, the other **must** be NULL. The elements of the non-NULL array describe the total count used to build this geometry. For a given format and subdivisionLevel the number of triangles in this geometry matching those values after indirection **must** be equal to the sum of matching count provided.

### Valid Usage

- VUID-VkAccelerationStructureTrianglesDisplacementMicromapNV-displacementBiasAndScaleFormat-09501
  displacementBiasAndScaleFormat **must** not be VK_FORMAT_UNDEFINED
- VUID-VkAccelerationStructureTrianglesDisplacementMicromapNV-displacementVectorFormat-09502
  displacementVectorFormat **must** not be VK_FORMAT_UNDEFINED
- VUID-VkAccelerationStructureTrianglesDisplacementMicromapNV-pUsageCounts-07992
  Only one of pUsageCounts or ppUsageCounts **can** be a valid pointer, the other **must** be NULL

### Valid Usage (Implicit)

- VUID-VkAccelerationStructureTrianglesDisplacementMicromapNV-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_ACCELERATION_STRUCTURE_TRIANGLES_DISPLACEMENT_MICROMAP_NV`
- VUID-VkAccelerationStructureTrianglesDisplacementMicromapNV-displacementBiasAndScaleFormat-parameter
  `displacementBiasAndScaleFormat` **must** be a valid VkFormat value
- VUID-VkAccelerationStructureTrianglesDisplacementMicromapNV-displacementVectorFormat-parameter
  `displacementVectorFormat` **must** be a valid VkFormat value
- VUID-VkAccelerationStructureTrianglesDisplacementMicromapNV-indexType-parameter
  `indexType` **must** be a valid VkIndexType value
- VUID-VkAccelerationStructureTrianglesDisplacementMicromapNV-pUsageCounts-parameter
  If `usageCountsCount` is not `0`, and `pUsageCounts` is not `NULL`, `pUsageCounts` **must** be a valid pointer to an array of `usageCountsCount` VkMicromapUsageEXT structures
- VUID-VkAccelerationStructureTrianglesDisplacementMicromapNV-ppUsageCounts-parameter
  If `usageCountsCount` is not `0`, and `ppUsageCounts` is not `NULL`, `ppUsageCounts` **must** be a valid pointer to an array of `usageCountsCount` valid pointers to VkMicromapUsageEXT structures
- VUID-VkAccelerationStructureTrianglesDisplacementMicromapNV-micromap-parameter
  If `micromap` is not VK_NULL_HANDLE, `micromap` **must** be a valid VkMicromapEXT handle

The `VkTransformMatrixKHR` structure is defined as:

```
// Provided by VK_KHR_acceleration_structure
typedef struct VkTransformMatrixKHR {
    float    matrix[3][4];
} VkTransformMatrixKHR;
```

or the equivalent

```
// Provided by VK_NV_ray_tracing
typedef VkTransformMatrixKHR VkTransformMatrixNV;
```

- `matrix` is a 3x4 row-major affine transformation matrix.

### Valid Usage

- VUID-VkTransformMatrixKHR-matrix-03799
  The first three columns of `matrix` **must** define an invertible 3x3 matrix

The VkAccelerationStructureGeometryAabbsDataKHR structure is defined as:

```
// Provided by VK_KHR_acceleration_structure
typedef struct VkAccelerationStructureGeometryAabbsDataKHR {
    VkStructureType                 sType;
    const void*                     pNext;
    VkDeviceOrHostAddressConstKHR   data;
    VkDeviceSize                    stride;
} VkAccelerationStructureGeometryAabbsDataKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- data is a device or host address to memory containing VkAabbPositionsKHR structures containing position data for each axis-aligned bounding box in the geometry.
- stride is the stride in bytes between each entry in data. The stride **must** be a multiple of 8.

### Valid Usage

- VUID-VkAccelerationStructureGeometryAabbsDataKHR-stride-03545
  stride **must** be a multiple of 8
- VUID-VkAccelerationStructureGeometryAabbsDataKHR-stride-03820
  stride **must** be less than or equal to $2^{32}$-1

### Valid Usage (Implicit)

- VUID-VkAccelerationStructureGeometryAabbsDataKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_ACCELERATION_STRUCTURE_GEOMETRY_AABBS_DATA_KHR
- VUID-VkAccelerationStructureGeometryAabbsDataKHR-pNext-pNext
  pNext **must** be NULL

The VkAabbPositionsKHR structure is defined as:

```
// Provided by VK_KHR_acceleration_structure
typedef struct VkAabbPositionsKHR {
    float    minX;
    float    minY;
    float    minZ;
    float    maxX;
    float    maxY;
    float    maxZ;
} VkAabbPositionsKHR;
```

or the equivalent

```
// Provided by VK_NV_ray_tracing
typedef VkAabbPositionsKHR VkAabbPositionsNV;
```

- minX is the x position of one opposing corner of a bounding box.
- minY is the y position of one opposing corner of a bounding box.
- minZ is the z position of one opposing corner of a bounding box.
- maxX is the x position of the other opposing corner of a bounding box.
- maxY is the y position of the other opposing corner of a bounding box.
- maxZ is the z position of the other opposing corner of a bounding box.

### Valid Usage

- VUID-VkAabbPositionsKHR-minX-03546
  minX **must** be less than or equal to maxX
- VUID-VkAabbPositionsKHR-minY-03547
  minY **must** be less than or equal to maxY
- VUID-VkAabbPositionsKHR-minZ-03548
  minZ **must** be less than or equal to maxZ

The VkAccelerationStructureGeometryInstancesDataKHR structure is defined as:

```
// Provided by VK_KHR_acceleration_structure
typedef struct VkAccelerationStructureGeometryInstancesDataKHR {
    VkStructureType                 sType;
    const void*                     pNext;
    VkBool32                        arrayOfPointers;
    VkDeviceOrHostAddressConstKHR   data;
} VkAccelerationStructureGeometryInstancesDataKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- arrayOfPointers specifies whether data is used as an array of addresses or just an array.
- data is either the address of an array of device or host addresses referencing individual VkAccelerationStructureInstanceKHR structures or packed motion instance information as described in motion instances if arrayOfPointers is VK_TRUE, or the address of an array of VkAccelerationStructureInstanceKHR or VkAccelerationStructureMotionInstanceNV structures. Addresses and VkAccelerationStructureInstanceKHR structures are tightly packed. VkAccelerationStructureMotionInstanceNV structures have a stride of 160 bytes.

### Valid Usage (Implicit)

- VUID-VkAccelerationStructureGeometryInstancesDataKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_ACCELERATION_STRUCTURE_GEOMETRY_INSTANCES_DATA_KHR`
- VUID-VkAccelerationStructureGeometryInstancesDataKHR-pNext-pNext
  `pNext` **must** be `NULL`

*Acceleration structure instances* **can** be built into top-level acceleration structures. Each acceleration structure instance is a separate entry in the top-level acceleration structure which includes all the geometry of a bottom-level acceleration structure at a transformed location. Multiple instances **can** point to the same bottom level acceleration structure.

An acceleration structure instance is defined by the structure:

```
// Provided by VK_KHR_acceleration_structure
typedef struct VkAccelerationStructureInstanceKHR {
    VkTransformMatrixKHR          transform;
    uint32_t                      instanceCustomIndex:24;
    uint32_t                      mask:8;
    uint32_t                      instanceShaderBindingTableRecordOffset:24;
    VkGeometryInstanceFlagsKHR    flags:8;
    uint64_t                      accelerationStructureReference;
} VkAccelerationStructureInstanceKHR;
```

or the equivalent

```
// Provided by VK_NV_ray_tracing
typedef VkAccelerationStructureInstanceKHR VkAccelerationStructureInstanceNV;
```

- `transform` is a VkTransformMatrixKHR structure describing a transformation to be applied to the acceleration structure.
- `instanceCustomIndex` is a 24-bit application-specified index value accessible to ray shaders in the `InstanceCustomIndexKHR` built-in.
- `mask` is an 8-bit visibility mask for the geometry. The instance **may** only be hit if `Cull Mask & instance.mask != 0`
- `instanceShaderBindingTableRecordOffset` is a 24-bit offset used in calculating the hit shader binding table index.
- `flags` is an 8-bit mask of VkGeometryInstanceFlagBitsKHR values to apply to this instance.
- `accelerationStructureReference` is either:
  - a device address containing the value obtained from vkGetAccelerationStructureDeviceAddressKHR or vkGetAccelerationStructureHandleNV (used by device operations which reference acceleration structures) or,
  - a VkAccelerationStructureKHR object (used by host operations which reference acceleration

structures).

The C language specification does not define the ordering of bit-fields, but in practice, this struct produces the correct layout with existing compilers. The intended bit pattern is for the following:

- `instanceCustomIndex` and `mask` occupy the same memory as if a single `uint32_t` was specified in their place
  - `instanceCustomIndex` occupies the 24 least significant bits of that memory
  - `mask` occupies the 8 most significant bits of that memory
- `instanceShaderBindingTableRecordOffset` and `flags` occupy the same memory as if a single `uint32_t` was specified in their place
  - `instanceShaderBindingTableRecordOffset` occupies the 24 least significant bits of that memory
  - `flags` occupies the 8 most significant bits of that memory

If a compiler produces code that diverges from that pattern, applications **must** employ another method to set values according to the correct bit pattern.

### Valid Usage (Implicit)

- VUID-VkAccelerationStructureInstanceKHR-flags-parameter
  `flags` **must** be a valid combination of VkGeometryInstanceFlagBitsKHR values

Possible values of `flags` in the instance modifying the behavior of that instance are:

```
// Provided by VK_KHR_acceleration_structure
typedef enum VkGeometryInstanceFlagBitsKHR {
    VK_GEOMETRY_INSTANCE_TRIANGLE_FACING_CULL_DISABLE_BIT_KHR = 0x00000001,
    VK_GEOMETRY_INSTANCE_TRIANGLE_FLIP_FACING_BIT_KHR = 0x00000002,
    VK_GEOMETRY_INSTANCE_FORCE_OPAQUE_BIT_KHR = 0x00000004,
    VK_GEOMETRY_INSTANCE_FORCE_NO_OPAQUE_BIT_KHR = 0x00000008,
  // Provided by VK_EXT_opacity_micromap
    VK_GEOMETRY_INSTANCE_FORCE_OPACITY_MICROMAP_2_STATE_EXT = 0x00000010,
  // Provided by VK_EXT_opacity_micromap
    VK_GEOMETRY_INSTANCE_DISABLE_OPACITY_MICROMAPS_EXT = 0x00000020,
    VK_GEOMETRY_INSTANCE_TRIANGLE_FRONT_COUNTERCLOCKWISE_BIT_KHR =
VK_GEOMETRY_INSTANCE_TRIANGLE_FLIP_FACING_BIT_KHR,
  // Provided by VK_NV_ray_tracing
    VK_GEOMETRY_INSTANCE_TRIANGLE_CULL_DISABLE_BIT_NV =
VK_GEOMETRY_INSTANCE_TRIANGLE_FACING_CULL_DISABLE_BIT_KHR,
  // Provided by VK_NV_ray_tracing
    VK_GEOMETRY_INSTANCE_TRIANGLE_FRONT_COUNTERCLOCKWISE_BIT_NV =
VK_GEOMETRY_INSTANCE_TRIANGLE_FRONT_COUNTERCLOCKWISE_BIT_KHR,
  // Provided by VK_NV_ray_tracing
    VK_GEOMETRY_INSTANCE_FORCE_OPAQUE_BIT_NV =
VK_GEOMETRY_INSTANCE_FORCE_OPAQUE_BIT_KHR,
  // Provided by VK_NV_ray_tracing
    VK_GEOMETRY_INSTANCE_FORCE_NO_OPAQUE_BIT_NV =
VK_GEOMETRY_INSTANCE_FORCE_NO_OPAQUE_BIT_KHR,
} VkGeometryInstanceFlagBitsKHR;
```

or the equivalent

```
// Provided by VK_NV_ray_tracing
typedef VkGeometryInstanceFlagBitsKHR VkGeometryInstanceFlagBitsNV;
```

- `VK_GEOMETRY_INSTANCE_TRIANGLE_FACING_CULL_DISABLE_BIT_KHR` disables face culling for this instance.
- `VK_GEOMETRY_INSTANCE_TRIANGLE_FLIP_FACING_BIT_KHR` indicates that the facing determination for geometry in this instance is inverted. Because the facing is determined in object space, an instance transform does not change the winding, but a geometry transform does.
- `VK_GEOMETRY_INSTANCE_FORCE_OPAQUE_BIT_KHR` causes this instance to act as though `VK_GEOMETRY_OPAQUE_BIT_KHR` were specified on all geometries referenced by this instance. This behavior **can** be overridden by the SPIR-V `NoOpaqueKHR` ray flag.
- `VK_GEOMETRY_INSTANCE_FORCE_NO_OPAQUE_BIT_KHR` causes this instance to act as though `VK_GEOMETRY_OPAQUE_BIT_KHR` were not specified on all geometries referenced by this instance. This behavior **can** be overridden by the SPIR-V `OpaqueKHR` ray flag.

`VK_GEOMETRY_INSTANCE_FORCE_NO_OPAQUE_BIT_KHR` and `VK_GEOMETRY_INSTANCE_FORCE_OPAQUE_BIT_KHR` **must** not be used in the same flag.

```
// Provided by VK_KHR_acceleration_structure
typedef VkFlags VkGeometryInstanceFlagsKHR;
```

or the equivalent

```
// Provided by VK_NV_ray_tracing
typedef VkGeometryInstanceFlagsKHR VkGeometryInstanceFlagsNV;
```

`VkGeometryInstanceFlagsKHR` is a bitmask type for setting a mask of zero or more VkGeometryInstanceFlagBitsKHR.

*Acceleration structure motion instances* **can** be built into top-level acceleration structures. Each acceleration structure instance is a separate entry in the top-level acceleration structure which includes all the geometry of a bottom-level acceleration structure at a transformed location including a type of motion and parameters to determine the motion of the instance over time.

An acceleration structure motion instance is defined by the structure:

```
// Provided by VK_NV_ray_tracing_motion_blur
typedef struct VkAccelerationStructureMotionInstanceNV {
    VkAccelerationStructureMotionInstanceTypeNV     type;
    VkAccelerationStructureMotionInstanceFlagsNV    flags;
    VkAccelerationStructureMotionInstanceDataNV     data;
} VkAccelerationStructureMotionInstanceNV;
```

- `type` is a VkAccelerationStructureMotionInstanceTypeNV enumerant identifying which type of motion instance this is and which type of the union is valid.
- `flags` is currently unused, but is required to keep natural alignment of `data`.
- `data` is a VkAccelerationStructureMotionInstanceDataNV containing motion instance data for this instance.

**NOTE** If writing this other than with a standard C compiler, note that the final structure should be 152 bytes in size.

### Valid Usage (Implicit)

- VUID-VkAccelerationStructureMotionInstanceNV-type-parameter
  `type` **must** be a valid VkAccelerationStructureMotionInstanceTypeNV value
- VUID-VkAccelerationStructureMotionInstanceNV-flags-zerobitmask
  `flags` **must** be `0`
- VUID-VkAccelerationStructureMotionInstanceNV-staticInstance-parameter
  If `type` is `VK_ACCELERATION_STRUCTURE_MOTION_INSTANCE_TYPE_STATIC_NV`, the `staticInstance` member of `data` **must** be a valid VkAccelerationStructureInstanceKHR structure

- VUID-VkAccelerationStructureMotionInstanceNV-matrixMotionInstance-parameter
  If `type` is `VK_ACCELERATION_STRUCTURE_MOTION_INSTANCE_TYPE_MATRIX_MOTION_NV`, the `matrixMotionInstance` member of `data` **must** be a valid VkAccelerationStructureMatrixMotionInstanceNV structure
- VUID-VkAccelerationStructureMotionInstanceNV-srtMotionInstance-parameter
  If `type` is `VK_ACCELERATION_STRUCTURE_MOTION_INSTANCE_TYPE_SRT_MOTION_NV`, the `srtMotionInstance` member of `data` **must** be a valid VkAccelerationStructureSRTMotionInstanceNV structure

Acceleration structure motion instance is defined by the union:

```
// Provided by VK_NV_ray_tracing_motion_blur
typedef union VkAccelerationStructureMotionInstanceDataNV {
    VkAccelerationStructureInstanceKHR               staticInstance;
    VkAccelerationStructureMatrixMotionInstanceNV    matrixMotionInstance;
    VkAccelerationStructureSRTMotionInstanceNV       srtMotionInstance;
} VkAccelerationStructureMotionInstanceDataNV;
```

- `staticInstance` is a VkAccelerationStructureInstanceKHR structure containing data for a static instance.
- `matrixMotionInstance` is a VkAccelerationStructureMatrixMotionInstanceNV structure containing data for a matrix motion instance.
- `srtMotionInstance` is a VkAccelerationStructureSRTMotionInstanceNV structure containing data for an SRT motion instance.

```
// Provided by VK_NV_ray_tracing_motion_blur
typedef VkFlags VkAccelerationStructureMotionInstanceFlagsNV;
```

`VkAccelerationStructureMotionInstanceFlagsNV` is a bitmask type for setting a mask, but is currently reserved for future use.

The `VkAccelerationStructureMotionInstanceTypeNV` enumeration is defined as:

```
// Provided by VK_NV_ray_tracing_motion_blur
typedef enum VkAccelerationStructureMotionInstanceTypeNV {
    VK_ACCELERATION_STRUCTURE_MOTION_INSTANCE_TYPE_STATIC_NV = 0,
    VK_ACCELERATION_STRUCTURE_MOTION_INSTANCE_TYPE_MATRIX_MOTION_NV = 1,
    VK_ACCELERATION_STRUCTURE_MOTION_INSTANCE_TYPE_SRT_MOTION_NV = 2,
} VkAccelerationStructureMotionInstanceTypeNV;
```

- `VK_ACCELERATION_STRUCTURE_MOTION_INSTANCE_TYPE_STATIC_NV` specifies that the instance is a static instance with no instance motion.
- `VK_ACCELERATION_STRUCTURE_MOTION_INSTANCE_TYPE_MATRIX_MOTION_NV` specifies that the instance is a motion instance with motion specified by interpolation between two matrices.

- VK_ACCELERATION_STRUCTURE_MOTION_INSTANCE_TYPE_SRT_MOTION_NV specifies that the instance is a motion instance with motion specified by interpolation in the SRT decomposition.

An acceleration structure matrix motion instance is defined by the structure:

```
// Provided by VK_NV_ray_tracing_motion_blur
typedef struct VkAccelerationStructureMatrixMotionInstanceNV {
    VkTransformMatrixKHR          transformT0;
    VkTransformMatrixKHR          transformT1;
    uint32_t                      instanceCustomIndex:24;
    uint32_t                      mask:8;
    uint32_t                      instanceShaderBindingTableRecordOffset:24;
    VkGeometryInstanceFlagsKHR    flags:8;
    uint64_t                      accelerationStructureReference;
} VkAccelerationStructureMatrixMotionInstanceNV;
```

- transformT0 is a VkTransformMatrixKHR structure describing a transformation to be applied to the acceleration structure at time 0.
- transformT1 is a VkTransformMatrixKHR structure describing a transformation to be applied to the acceleration structure at time 1.
- instanceCustomIndex is a 24-bit application-specified index value accessible to ray shaders in the InstanceCustomIndexKHR built-in.
- mask is an 8-bit visibility mask for the geometry. The instance **may** only be hit if Cull Mask & instance.mask != 0
- instanceShaderBindingTableRecordOffset is a 24-bit offset used in calculating the hit shader binding table index.
- flags is an 8-bit mask of VkGeometryInstanceFlagBitsKHR values to apply to this instance.
- accelerationStructureReference is either:
  - a device address containing the value obtained from vkGetAccelerationStructureDeviceAddressKHR or vkGetAccelerationStructureHandleNV (used by device operations which reference acceleration structures) or,
  - a VkAccelerationStructureKHR object (used by host operations which reference acceleration structures).

The C language specification does not define the ordering of bit-fields, but in practice, this struct produces the correct layout with existing compilers. The intended bit pattern is for the following:

- instanceCustomIndex and mask occupy the same memory as if a single uint32_t was specified in their place
  - instanceCustomIndex occupies the 24 least significant bits of that memory
  - mask occupies the 8 most significant bits of that memory
- instanceShaderBindingTableRecordOffset and flags occupy the same memory as if a single uint32_t was specified in their place

- instanceShaderBindingTableRecordOffset occupies the 24 least significant bits of that memory
- flags occupies the 8 most significant bits of that memory

If a compiler produces code that diverges from that pattern, applications **must** employ another method to set values according to the correct bit pattern.

The transform for a matrix motion instance at a point in time is derived by component-wise linear interpolation of the two transforms. That is, for a time in [0,1] the resulting transform is

transformT0 × (1 - time) + transformT1 × time

### Valid Usage (Implicit)

- VUID-VkAccelerationStructureMatrixMotionInstanceNV-flags-parameter
  flags **must** be a valid combination of VkGeometryInstanceFlagBitsKHR values

An acceleration structure SRT motion instance is defined by the structure:

```
// Provided by VK_NV_ray_tracing_motion_blur
typedef struct VkAccelerationStructureSRTMotionInstanceNV {
    VkSRTDataNV                    transformT0;
    VkSRTDataNV                    transformT1;
    uint32_t                       instanceCustomIndex:24;
    uint32_t                       mask:8;
    uint32_t                       instanceShaderBindingTableRecordOffset:24;
    VkGeometryInstanceFlagsKHR     flags:8;
    uint64_t                       accelerationStructureReference;
} VkAccelerationStructureSRTMotionInstanceNV;
```

- transformT0 is a VkSRTDataNV structure describing a transformation to be applied to the acceleration structure at time 0.
- transformT1 is a VkSRTDataNV structure describing a transformation to be applied to the acceleration structure at time 1.
- instanceCustomIndex is a 24-bit application-specified index value accessible to ray shaders in the InstanceCustomIndexKHR built-in.
- mask is an 8-bit visibility mask for the geometry. The instance **may** only be hit if Cull Mask & instance.mask != 0
- instanceShaderBindingTableRecordOffset is a 24-bit offset used in calculating the hit shader binding table index.
- flags is an 8-bit mask of VkGeometryInstanceFlagBitsKHR values to apply to this instance.
- accelerationStructureReference is either:
  - a device address containing the value obtained from vkGetAccelerationStructureDeviceAddressKHR or vkGetAccelerationStructureHandleNV

(used by device operations which reference acceleration structures) or,

- a VkAccelerationStructureKHR object (used by host operations which reference acceleration structures).

The C language specification does not define the ordering of bit-fields, but in practice, this struct produces the correct layout with existing compilers. The intended bit pattern is for the following:

- `instanceCustomIndex` and `mask` occupy the same memory as if a single `uint32_t` was specified in their place
  - `instanceCustomIndex` occupies the 24 least significant bits of that memory
  - `mask` occupies the 8 most significant bits of that memory
- `instanceShaderBindingTableRecordOffset` and `flags` occupy the same memory as if a single `uint32_t` was specified in their place
  - `instanceShaderBindingTableRecordOffset` occupies the 24 least significant bits of that memory
  - `flags` occupies the 8 most significant bits of that memory

If a compiler produces code that diverges from that pattern, applications **must** employ another method to set values according to the correct bit pattern.

The transform for a SRT motion instance at a point in time is derived from component-wise linear interpolation of the two SRT transforms. That is, for a `time` in [0,1] the resulting transform is

$$\texttt{transformT0} \times (1 - \texttt{time}) + \texttt{transformT1} \times \texttt{time}$$

### Valid Usage (Implicit)

- VUID-VkAccelerationStructureSRTMotionInstanceNV-flags-parameter
  `flags` **must** be a valid combination of VkGeometryInstanceFlagBitsKHR values

An acceleration structure SRT transform is defined by the structure:

```
// Provided by VK_NV_ray_tracing_motion_blur
typedef struct VkSRTDataNV {
    float    sx;
    float    a;
    float    b;
    float    pvx;
    float    sy;
    float    c;
    float    pvy;
    float    sz;
    float    pvz;
    float    qx;
    float    qy;
    float    qz;
    float    qw;
    float    tx;
    float    ty;
    float    tz;
} VkSRTDataNV;
```

- `sx` is the x component of the scale of the transform
- `a` is one component of the shear for the transform
- `b` is one component of the shear for the transform
- `pvx` is the x component of the pivot point of the transform
- `sy` is the y component of the scale of the transform
- `c` is one component of the shear for the transform
- `pvy` is the y component of the pivot point of the transform
- `sz` is the z component of the scale of the transform
- `pvz` is the z component of the pivot point of the transform
- `qx` is the x component of the rotation quaternion
- `qy` is the y component of the rotation quaternion
- `qz` is the z component of the rotation quaternion
- `qw` is the w component of the rotation quaternion
- `tx` is the x component of the post-rotation translation
- `ty` is the y component of the post-rotation translation
- `tz` is the z component of the post-rotation translation

This transform decomposition consists of three elements. The first is a matrix S, consisting of a scale, shear, and translation, usually used to define the pivot point of the following rotation. This matrix is constructed from the parameters above by:

$$S = \begin{pmatrix} sx & a & b & pvx \\ 0 & sy & c & pvy \\ 0 & 0 & sz & pvz \end{pmatrix}$$

The rotation quaternion is defined as:

R = [ qx, qy, qz, qw ]

This is a rotation around a conceptual normalized axis [ ax, ay, az ] of amount theta such that:

[ qx, qy, qz ] = sin(theta/2) × [ ax, ay, az ]

and

qw = cos(theta/2)

Finally, the transform has a translation T constructed from the parameters above by:

$$T = \begin{pmatrix} 1 & 0 & 0 & tx \\ 0 & 1 & 0 & ty \\ 0 & 0 & 1 & tz \end{pmatrix}$$

The effective derived transform is then given by

T × R × S

VkAccelerationStructureBuildRangeInfoKHR is defined as:

```
// Provided by VK_KHR_acceleration_structure
typedef struct VkAccelerationStructureBuildRangeInfoKHR {
    uint32_t    primitiveCount;
    uint32_t    primitiveOffset;
    uint32_t    firstVertex;
    uint32_t    transformOffset;
} VkAccelerationStructureBuildRangeInfoKHR;
```

- primitiveCount defines the number of primitives for a corresponding acceleration structure geometry.
- primitiveOffset defines an offset in bytes into the memory where primitive data is defined.
- firstVertex is the index of the first vertex to build from for triangle geometry.
- transformOffset defines an offset in bytes into the memory where a transform matrix is defined.

The primitive count and primitive offset are interpreted differently depending on the VkGeometryTypeKHR used:

- For geometries of type VK_GEOMETRY_TYPE_TRIANGLES_KHR, primitiveCount is the number of triangles to be built, where each triangle is treated as 3 vertices.
  - If the geometry uses indices, primitiveCount × 3 indices are consumed from VkAccelerationStructureGeometryTrianglesDataKHR::indexData, starting at an offset of primitiveOffset. The value of firstVertex is added to the index values before fetching vertices.
  - If the geometry does not use indices, primitiveCount × 3 vertices are consumed from VkAccelerationStructureGeometryTrianglesDataKHR::vertexData, starting at an offset of primitiveOffset + VkAccelerationStructureGeometryTrianglesDataKHR::vertexStride × firstVertex.
  - If VkAccelerationStructureGeometryTrianglesDataKHR::transformData is not NULL, a single VkTransformMatrixKHR structure is consumed from VkAccelerationStructureGeometryTrianglesDataKHR::transformData, at an offset of transformOffset. This matrix describes a transformation from the space in which the vertices for all triangles in this geometry are described to the space in which the acceleration structure is defined.
- For geometries of type VK_GEOMETRY_TYPE_AABBS_KHR, primitiveCount is the number of axis-aligned bounding boxes. primitiveCount VkAabbPositionsKHR structures are consumed from VkAccelerationStructureGeometryAabbsDataKHR::data, starting at an offset of primitiveOffset.
- For geometries of type VK_GEOMETRY_TYPE_INSTANCES_KHR, primitiveCount is the number of acceleration structures. primitiveCount VkAccelerationStructureInstanceKHR or VkAccelerationStructureMotionInstanceNV structures are consumed from VkAccelerationStructureGeometryInstancesDataKHR::data, starting at an offset of primitiveOffset.

## Valid Usage

- VUID-VkAccelerationStructureBuildRangeInfoKHR-primitiveOffset-03656
  For geometries of type VK_GEOMETRY_TYPE_TRIANGLES_KHR, if the geometry uses indices, the offset primitiveOffset from VkAccelerationStructureGeometryTrianglesDataKHR::indexData **must** be a multiple of the element size of VkAccelerationStructureGeometryTrianglesDataKHR::indexType
- VUID-VkAccelerationStructureBuildRangeInfoKHR-primitiveOffset-03657
  For geometries of type VK_GEOMETRY_TYPE_TRIANGLES_KHR, if the geometry does not use indices, the offset primitiveOffset from VkAccelerationStructureGeometryTrianglesDataKHR::vertexData **must** be a multiple of the component size of VkAccelerationStructureGeometryTrianglesDataKHR::vertexFormat
- VUID-VkAccelerationStructureBuildRangeInfoKHR-transformOffset-03658
  For geometries of type VK_GEOMETRY_TYPE_TRIANGLES_KHR, the offset transformOffset from VkAccelerationStructureGeometryTrianglesDataKHR::transformData **must** be a multiple of 16
- VUID-VkAccelerationStructureBuildRangeInfoKHR-primitiveOffset-03659
  For geometries of type VK_GEOMETRY_TYPE_AABBS_KHR, the offset primitiveOffset from VkAccelerationStructureGeometryAabbsDataKHR::data **must** be a multiple of 8

- VUID-VkAccelerationStructureBuildRangeInfoKHR-primitiveOffset-03660
  For geometries of type VK_GEOMETRY_TYPE_INSTANCES_KHR, the offset primitiveOffset from VkAccelerationStructureGeometryInstancesDataKHR::data **must** be a multiple of 16

### 37.1.7. Copying Acceleration Structures

An additional command exists for copying acceleration structures without updating their contents. The acceleration structure object **can** be compacted in order to improve performance. Before copying, an application **must** query the size of the resulting acceleration structure.

To query acceleration structure size parameters call:

```
// Provided by VK_KHR_acceleration_structure
void vkCmdWriteAccelerationStructuresPropertiesKHR(
    VkCommandBuffer                             commandBuffer,
    uint32_t                                    accelerationStructureCount,
    const VkAccelerationStructureKHR*           pAccelerationStructures,
    VkQueryType                                 queryType,
    VkQueryPool                                 queryPool,
    uint32_t                                    firstQuery);
```

- commandBuffer is the command buffer into which the command will be recorded.
- accelerationStructureCount is the count of acceleration structures for which to query the property.
- pAccelerationStructures is a pointer to an array of existing previously built acceleration structures.
- queryType is a VkQueryType value specifying the type of queries managed by the pool.
- queryPool is the query pool that will manage the results of the query.
- firstQuery is the first query index within the query pool that will contain the accelerationStructureCount number of results.

Accesses to any of the acceleration structures listed in pAccelerationStructures **must** be synchronized with the VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_COPY_BIT_KHR pipeline stage or the VK_PIPELINE_STAGE_ACCELERATION_STRUCTURE_BUILD_BIT_KHR pipeline stage, and an access type of VK_ACCESS_ACCELERATION_STRUCTURE_READ_BIT_KHR.

- If queryType is VK_QUERY_TYPE_ACCELERATION_STRUCTURE_COMPACTED_SIZE_KHR, then the value written out is the number of bytes required by a compacted acceleration structure.
- If queryType is VK_QUERY_TYPE_ACCELERATION_STRUCTURE_SERIALIZATION_SIZE_KHR, then the value written out is the number of bytes required by a serialized acceleration structure.

#### Valid Usage

- VUID-vkCmdWriteAccelerationStructuresPropertiesKHR-accelerationStructure-08924
  The VkPhysicalDeviceAccelerationStructureFeaturesKHR::accelerationStructure feature

**must** be enabled

- VUID-vkCmdWriteAccelerationStructuresPropertiesKHR-queryPool-02493
  `queryPool` **must** have been created with a `queryType` matching `queryType`
- VUID-vkCmdWriteAccelerationStructuresPropertiesKHR-queryPool-02494
  The queries identified by `queryPool` and `firstQuery` **must** be *unavailable*
- VUID-vkCmdWriteAccelerationStructuresPropertiesKHR-buffer-03736
  The `buffer` used to create each acceleration structure in `pAccelerationStructures` **must** be bound to device memory
- VUID-vkCmdWriteAccelerationStructuresPropertiesKHR-query-04880
  The sum of `firstQuery` plus `accelerationStructureCount` **must** be less than or equal to the number of queries in `queryPool`
- VUID-vkCmdWriteAccelerationStructuresPropertiesKHR-pAccelerationStructures-04964
  All acceleration structures in `pAccelerationStructures` **must** have been built prior to the execution of this command
- VUID-vkCmdWriteAccelerationStructuresPropertiesKHR-accelerationStructures-03431
  All acceleration structures in `pAccelerationStructures` **must** have been built with `VK_BUILD_ACCELERATION_STRUCTURE_ALLOW_COMPACTION_BIT_KHR` if `queryType` is `VK_QUERY_TYPE_ACCELERATION_STRUCTURE_COMPACTED_SIZE_KHR`
- VUID-vkCmdWriteAccelerationStructuresPropertiesKHR-queryType-06742
  `queryType` **must** be `VK_QUERY_TYPE_ACCELERATION_STRUCTURE_SIZE_KHR`, `VK_QUERY_TYPE_ACCELERATION_STRUCTURE_SERIALIZATION_BOTTOM_LEVEL_POINTERS_KHR`, `VK_QUERY_TYPE_ACCELERATION_STRUCTURE_COMPACTED_SIZE_KHR`, or `VK_QUERY_TYPE_ACCELERATION_STRUCTURE_SERIALIZATION_SIZE_KHR`

## Valid Usage (Implicit)

- VUID-vkCmdWriteAccelerationStructuresPropertiesKHR-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdWriteAccelerationStructuresPropertiesKHR-pAccelerationStructures-parameter
  `pAccelerationStructures` **must** be a valid pointer to an array of `accelerationStructureCount` valid VkAccelerationStructureKHR handles
- VUID-vkCmdWriteAccelerationStructuresPropertiesKHR-queryType-parameter
  `queryType` **must** be a valid VkQueryType value
- VUID-vkCmdWriteAccelerationStructuresPropertiesKHR-queryPool-parameter
  `queryPool` **must** be a valid VkQueryPool handle
- VUID-vkCmdWriteAccelerationStructuresPropertiesKHR-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdWriteAccelerationStructuresPropertiesKHR-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support compute operations

- VUID-vkCmdWriteAccelerationStructuresPropertiesKHR-renderpass
  This command **must** only be called outside of a render pass instance
- VUID-vkCmdWriteAccelerationStructuresPropertiesKHR-videocoding
  This command **must** only be called outside of a video coding scope
- VUID-vkCmdWriteAccelerationStructuresPropertiesKHR-accelerationStructureCount-arraylength
  `accelerationStructureCount` **must** be greater than `0`
- VUID-vkCmdWriteAccelerationStructuresPropertiesKHR-commonparent
  Each of `commandBuffer`, `queryPool`, and the elements of `pAccelerationStructures` **must** have been created, allocated, or retrieved from the same VkDevice

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Outside | Outside | Compute | Action |

To query acceleration structure size parameters call:

```
// Provided by VK_NV_ray_tracing
void vkCmdWriteAccelerationStructuresPropertiesNV(
    VkCommandBuffer                             commandBuffer,
    uint32_t                                    accelerationStructureCount,
    const VkAccelerationStructureNV*            pAccelerationStructures,
    VkQueryType                                 queryType,
    VkQueryPool                                 queryPool,
    uint32_t                                    firstQuery);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `accelerationStructureCount` is the count of acceleration structures for which to query the property.
- `pAccelerationStructures` is a pointer to an array of existing previously built acceleration structures.
- `queryType` is a VkQueryType value specifying the type of queries managed by the pool.

- queryPool is the query pool that will manage the results of the query.
- firstQuery is the first query index within the query pool that will contain the accelerationStructureCount number of results.

Accesses to any of the acceleration structures listed in pAccelerationStructures **must** be synchronized with the VK_PIPELINE_STAGE_ACCELERATION_STRUCTURE_BUILD_BIT_KHR pipeline stage and an access type of VK_ACCESS_ACCELERATION_STRUCTURE_READ_BIT_KHR.

## Valid Usage

- VUID-vkCmdWriteAccelerationStructuresPropertiesNV-queryPool-03755
  queryPool **must** have been created with a queryType matching queryType
- VUID-vkCmdWriteAccelerationStructuresPropertiesNV-queryPool-03756
  The queries identified by queryPool and firstQuery **must** be *unavailable*
- VUID-vkCmdWriteAccelerationStructuresPropertiesNV-accelerationStructure-03757
  accelerationStructure **must** be bound completely and contiguously to a single VkDeviceMemory object via vkBindAccelerationStructureMemoryNV
- VUID-vkCmdWriteAccelerationStructuresPropertiesNV-pAccelerationStructures-04958
  All acceleration structures in pAccelerationStructures **must** have been built prior to the execution of this command
- VUID-vkCmdWriteAccelerationStructuresPropertiesNV-pAccelerationStructures-06215
  All acceleration structures in pAccelerationStructures **must** have been built with VK_BUILD_ACCELERATION_STRUCTURE_ALLOW_COMPACTION_BIT_KHR if queryType is VK_QUERY_TYPE_ACCELERATION_STRUCTURE_COMPACTED_SIZE_NV
- VUID-vkCmdWriteAccelerationStructuresPropertiesNV-queryType-06216
  queryType **must** be VK_QUERY_TYPE_ACCELERATION_STRUCTURE_COMPACTED_SIZE_NV

## Valid Usage (Implicit)

- VUID-vkCmdWriteAccelerationStructuresPropertiesNV-commandBuffer-parameter
  commandBuffer **must** be a valid VkCommandBuffer handle
- VUID-vkCmdWriteAccelerationStructuresPropertiesNV-pAccelerationStructures-parameter
  pAccelerationStructures **must** be a valid pointer to an array of accelerationStructureCount valid VkAccelerationStructureNV handles
- VUID-vkCmdWriteAccelerationStructuresPropertiesNV-queryType-parameter
  queryType **must** be a valid VkQueryType value
- VUID-vkCmdWriteAccelerationStructuresPropertiesNV-queryPool-parameter
  queryPool **must** be a valid VkQueryPool handle
- VUID-vkCmdWriteAccelerationStructuresPropertiesNV-commandBuffer-recording
  commandBuffer **must** be in the recording state
- VUID-vkCmdWriteAccelerationStructuresPropertiesNV-commandBuffer-cmdpool

The `VkCommandPool` that `commandBuffer` was allocated from **must** support compute operations

- VUID-vkCmdWriteAccelerationStructuresPropertiesNV-renderpass
  This command **must** only be called outside of a render pass instance
- VUID-vkCmdWriteAccelerationStructuresPropertiesNV-videocoding
  This command **must** only be called outside of a video coding scope
- VUID-vkCmdWriteAccelerationStructuresPropertiesNV-accelerationStructureCount-arraylength
  `accelerationStructureCount` **must** be greater than `0`
- VUID-vkCmdWriteAccelerationStructuresPropertiesNV-commonparent
  Each of `commandBuffer`, `queryPool`, and the elements of `pAccelerationStructures` **must** have been created, allocated, or retrieved from the same VkDevice

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Outside | Outside | Compute | Action |

To copy an acceleration structure call:

```
// Provided by VK_NV_ray_tracing
void vkCmdCopyAccelerationStructureNV(
    VkCommandBuffer                             commandBuffer,
    VkAccelerationStructureNV                   dst,
    VkAccelerationStructureNV                   src,
    VkCopyAccelerationStructureModeKHR          mode);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `dst` is the target acceleration structure for the copy.
- `src` is the source acceleration structure for the copy.
- `mode` is a VkCopyAccelerationStructureModeKHR value specifying additional operations to perform during the copy.

Accesses to src and dst **must** be synchronized with the VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_COPY_BIT_KHR pipeline stage or the VK_PIPELINE_STAGE_ACCELERATION_STRUCTURE_BUILD_BIT_KHR pipeline stage, and an access type of VK_ACCESS_ACCELERATION_STRUCTURE_READ_BIT_KHR or VK_ACCESS_ACCELERATION_STRUCTURE_WRITE_BIT_KHR as appropriate.

## Valid Usage

- VUID-vkCmdCopyAccelerationStructureNV-mode-03410
  mode **must** be VK_COPY_ACCELERATION_STRUCTURE_MODE_COMPACT_KHR or VK_COPY_ACCELERATION_STRUCTURE_MODE_CLONE_KHR
- VUID-vkCmdCopyAccelerationStructureNV-src-04963
  The source acceleration structure src **must** have been constructed prior to the execution of this command
- VUID-vkCmdCopyAccelerationStructureNV-src-03411
  If mode is VK_COPY_ACCELERATION_STRUCTURE_MODE_COMPACT_KHR, src **must** have been constructed with VK_BUILD_ACCELERATION_STRUCTURE_ALLOW_COMPACTION_BIT_KHR in the build
- VUID-vkCmdCopyAccelerationStructureNV-buffer-03718
  The buffer used to create src **must** be bound to device memory
- VUID-vkCmdCopyAccelerationStructureNV-buffer-03719
  The buffer used to create dst **must** be bound to device memory
- VUID-vkCmdCopyAccelerationStructureNV-dst-07791
  The range of memory backing dst that is accessed by this command **must** not overlap the memory backing src that is accessed by this command
- VUID-vkCmdCopyAccelerationStructureNV-dst-07792
  dst **must** be bound completely and contiguously to a single VkDeviceMemory object via vkBindAccelerationStructureMemoryNV

## Valid Usage (Implicit)

- VUID-vkCmdCopyAccelerationStructureNV-commandBuffer-parameter
  commandBuffer **must** be a valid VkCommandBuffer handle
- VUID-vkCmdCopyAccelerationStructureNV-dst-parameter
  dst **must** be a valid VkAccelerationStructureNV handle
- VUID-vkCmdCopyAccelerationStructureNV-src-parameter
  src **must** be a valid VkAccelerationStructureNV handle
- VUID-vkCmdCopyAccelerationStructureNV-mode-parameter
  mode **must** be a valid VkCopyAccelerationStructureModeKHR value
- VUID-vkCmdCopyAccelerationStructureNV-commandBuffer-recording
  commandBuffer **must** be in the recording state
- VUID-vkCmdCopyAccelerationStructureNV-commandBuffer-cmdpool
  The VkCommandPool that commandBuffer was allocated from **must** support compute

operations

- VUID-vkCmdCopyAccelerationStructureNV-renderpass
  This command **must** only be called outside of a render pass instance
- VUID-vkCmdCopyAccelerationStructureNV-videocoding
  This command **must** only be called outside of a video coding scope
- VUID-vkCmdCopyAccelerationStructureNV-commonparent
  Each of `commandBuffer`, `dst`, and `src` **must** have been created, allocated, or retrieved from the same VkDevice

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Outside | Outside | Compute | Action |

To copy an acceleration structure call:

```
// Provided by VK_KHR_acceleration_structure
void vkCmdCopyAccelerationStructureKHR(
    VkCommandBuffer                             commandBuffer,
    const VkCopyAccelerationStructureInfoKHR*   pInfo);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `pInfo` is a pointer to a VkCopyAccelerationStructureInfoKHR structure defining the copy operation.

This command copies the `pInfo->src` acceleration structure to the `pInfo->dst` acceleration structure in the manner specified by `pInfo->mode`.

Accesses to `pInfo->src` and `pInfo->dst` **must** be synchronized with the `VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_COPY_BIT_KHR` pipeline stage or the `VK_PIPELINE_STAGE_ACCELERATION_STRUCTURE_BUILD_BIT_KHR` pipeline stage, and an access type of `VK_ACCESS_ACCELERATION_STRUCTURE_READ_BIT_KHR` or `VK_ACCESS_ACCELERATION_STRUCTURE_WRITE_BIT_KHR` as appropriate.

## Valid Usage

- VUID-vkCmdCopyAccelerationStructureKHR-accelerationStructure-08925
  The `VkPhysicalDeviceAccelerationStructureFeaturesKHR::accelerationStructure` feature **must** be enabled
- VUID-vkCmdCopyAccelerationStructureKHR-buffer-03737
  The `buffer` used to create `pInfo->src` **must** be bound to device memory
- VUID-vkCmdCopyAccelerationStructureKHR-buffer-03738
  The `buffer` used to create `pInfo->dst` **must** be bound to device memory

## Valid Usage (Implicit)

- VUID-vkCmdCopyAccelerationStructureKHR-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdCopyAccelerationStructureKHR-pInfo-parameter
  `pInfo` **must** be a valid pointer to a valid VkCopyAccelerationStructureInfoKHR structure
- VUID-vkCmdCopyAccelerationStructureKHR-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdCopyAccelerationStructureKHR-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support compute operations
- VUID-vkCmdCopyAccelerationStructureKHR-renderpass
  This command **must** only be called outside of a render pass instance
- VUID-vkCmdCopyAccelerationStructureKHR-videocoding
  This command **must** only be called outside of a video coding scope

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Outside | Outside | Compute | Action |

The VkCopyAccelerationStructureInfoKHR structure is defined as:

```
// Provided by VK_KHR_acceleration_structure
typedef struct VkCopyAccelerationStructureInfoKHR {
    VkStructureType                         sType;
    const void*                             pNext;
    VkAccelerationStructureKHR              src;
    VkAccelerationStructureKHR              dst;
    VkCopyAccelerationStructureModeKHR      mode;
} VkCopyAccelerationStructureInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- src is the source acceleration structure for the copy.
- dst is the target acceleration structure for the copy.
- mode is a VkCopyAccelerationStructureModeKHR value specifying additional operations to perform during the copy.

## Valid Usage

- VUID-VkCopyAccelerationStructureInfoKHR-mode-03410
  mode **must** be VK_COPY_ACCELERATION_STRUCTURE_MODE_COMPACT_KHR or VK_COPY_ACCELERATION_STRUCTURE_MODE_CLONE_KHR
- VUID-VkCopyAccelerationStructureInfoKHR-src-04963
  The source acceleration structure src **must** have been constructed prior to the execution of this command
- VUID-VkCopyAccelerationStructureInfoKHR-src-03411
  If mode is VK_COPY_ACCELERATION_STRUCTURE_MODE_COMPACT_KHR, src **must** have been constructed with VK_BUILD_ACCELERATION_STRUCTURE_ALLOW_COMPACTION_BIT_KHR in the build
- VUID-VkCopyAccelerationStructureInfoKHR-buffer-03718
  The buffer used to create src **must** be bound to device memory
- VUID-VkCopyAccelerationStructureInfoKHR-buffer-03719
  The buffer used to create dst **must** be bound to device memory
- VUID-VkCopyAccelerationStructureInfoKHR-dst-07791
  The range of memory backing dst that is accessed by this command **must** not overlap the memory backing src that is accessed by this command

## Valid Usage (Implicit)

- VUID-VkCopyAccelerationStructureInfoKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_COPY_ACCELERATION_STRUCTURE_INFO_KHR
- VUID-VkCopyAccelerationStructureInfoKHR-pNext-pNext

pNext **must** be NULL

- VUID-VkCopyAccelerationStructureInfoKHR-src-parameter
  src **must** be a valid VkAccelerationStructureKHR handle
- VUID-VkCopyAccelerationStructureInfoKHR-dst-parameter
  dst **must** be a valid VkAccelerationStructureKHR handle
- VUID-VkCopyAccelerationStructureInfoKHR-mode-parameter
  mode **must** be a valid VkCopyAccelerationStructureModeKHR value
- VUID-VkCopyAccelerationStructureInfoKHR-commonparent
  Both of dst, and src **must** have been created, allocated, or retrieved from the same VkDevice

Possible values of mode specifying additional operations to perform during the copy, are:

```
// Provided by VK_KHR_acceleration_structure
typedef enum VkCopyAccelerationStructureModeKHR {
    VK_COPY_ACCELERATION_STRUCTURE_MODE_CLONE_KHR = 0,
    VK_COPY_ACCELERATION_STRUCTURE_MODE_COMPACT_KHR = 1,
    VK_COPY_ACCELERATION_STRUCTURE_MODE_SERIALIZE_KHR = 2,
    VK_COPY_ACCELERATION_STRUCTURE_MODE_DESERIALIZE_KHR = 3,
  // Provided by VK_NV_ray_tracing
    VK_COPY_ACCELERATION_STRUCTURE_MODE_CLONE_NV =
VK_COPY_ACCELERATION_STRUCTURE_MODE_CLONE_KHR,
  // Provided by VK_NV_ray_tracing
    VK_COPY_ACCELERATION_STRUCTURE_MODE_COMPACT_NV =
VK_COPY_ACCELERATION_STRUCTURE_MODE_COMPACT_KHR,
} VkCopyAccelerationStructureModeKHR;
```

or the equivalent

```
// Provided by VK_NV_ray_tracing
typedef VkCopyAccelerationStructureModeKHR VkCopyAccelerationStructureModeNV;
```

- VK_COPY_ACCELERATION_STRUCTURE_MODE_CLONE_KHR creates a direct copy of the acceleration structure specified in src into the one specified by dst. The dst acceleration structure **must** have been created with the same parameters as src. If src contains references to other acceleration structures, dst will reference the same acceleration structures.
- VK_COPY_ACCELERATION_STRUCTURE_MODE_COMPACT_KHR creates a more compact version of an acceleration structure src into dst. The acceleration structure dst **must** have been created with a size at least as large as that returned by vkCmdWriteAccelerationStructuresPropertiesKHR or vkWriteAccelerationStructuresPropertiesKHR after the build of the acceleration structure specified by src. If src contains references to other acceleration structures, dst will reference the same acceleration structures.
- VK_COPY_ACCELERATION_STRUCTURE_MODE_SERIALIZE_KHR serializes the acceleration structure to a semi-opaque format which can be reloaded on a compatible implementation.

- VK_COPY_ACCELERATION_STRUCTURE_MODE_DESERIALIZE_KHR deserializes the semi-opaque serialization format in the buffer to the acceleration structure.

To copy an acceleration structure to device memory call:

```
// Provided by VK_KHR_acceleration_structure
void vkCmdCopyAccelerationStructureToMemoryKHR(
    VkCommandBuffer                             commandBuffer,
    const VkCopyAccelerationStructureToMemoryInfoKHR* pInfo);
```

- commandBuffer is the command buffer into which the command will be recorded.
- pInfo is an a pointer to a VkCopyAccelerationStructureToMemoryInfoKHR structure defining the copy operation.

Accesses to pInfo->src **must** be synchronized with the VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_COPY_BIT_KHR pipeline stage or the VK_PIPELINE_STAGE_ACCELERATION_STRUCTURE_BUILD_BIT_KHR pipeline stage, and an access type of VK_ACCESS_ACCELERATION_STRUCTURE_READ_BIT_KHR. Accesses to the buffer indicated by pInfo->dst.deviceAddress **must** be synchronized with the VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_COPY_BIT_KHR pipeline stage or the VK_PIPELINE_STAGE_ACCELERATION_STRUCTURE_BUILD_BIT_KHR pipeline stage, and an and an access type of VK_ACCESS_TRANSFER_WRITE_BIT.

This command produces the same results as vkCopyAccelerationStructureToMemoryKHR, but writes its result to a device address, and is executed on the device rather than the host. The output **may** not necessarily be bit-for-bit identical, but it can be equally used by either vkCmdCopyMemoryToAccelerationStructureKHR or vkCopyMemoryToAccelerationStructureKHR.

The defined header structure for the serialized data consists of:

- VK_UUID_SIZE bytes of data matching VkPhysicalDeviceIDProperties::driverUUID
- VK_UUID_SIZE bytes of data identifying the compatibility for comparison using vkGetDeviceAccelerationStructureCompatibilityKHR
- A 64-bit integer of the total size matching the value queried using VK_QUERY_TYPE_ACCELERATION_STRUCTURE_SERIALIZATION_SIZE_KHR
- A 64-bit integer of the deserialized size to be passed in to VkAccelerationStructureCreateInfoKHR::size
- A 64-bit integer of the count of the number of acceleration structure handles following. This value matches the value queried using VK_QUERY_TYPE_ACCELERATION_STRUCTURE_SERIALIZATION_BOTTOM_LEVEL_POINTERS_KHR. This will be zero for a bottom-level acceleration structure. For top-level acceleration structures this number is implementation-dependent; the number of and ordering of the handles may not match the instance descriptions which were used to build the acceleration structure.

The corresponding handles matching the values returned by vkGetAccelerationStructureDeviceAddressKHR or vkGetAccelerationStructureHandleNV are tightly

packed in the buffer following the count. The application is expected to store a mapping between those handles and the original application-generated bottom-level acceleration structures to provide when deserializing. The serialized data is written to the buffer (or read from the buffer) according to the host endianness.

## Valid Usage

- VUID-vkCmdCopyAccelerationStructureToMemoryKHR-accelerationStructure-08926
  The `VkPhysicalDeviceAccelerationStructureFeaturesKHR::accelerationStructure` feature **must** be enabled
- VUID-vkCmdCopyAccelerationStructureToMemoryKHR-pInfo-03739
  `pInfo->dst.deviceAddress` **must** be a valid device address for a buffer bound to device memory
- VUID-vkCmdCopyAccelerationStructureToMemoryKHR-pInfo-03740
  `pInfo->dst.deviceAddress` **must** be aligned to `256` bytes
- VUID-vkCmdCopyAccelerationStructureToMemoryKHR-pInfo-03741
  If the buffer pointed to by `pInfo->dst.deviceAddress` is non-sparse then it **must** be bound completely and contiguously to a single VkDeviceMemory object
- VUID-vkCmdCopyAccelerationStructureToMemoryKHR-None-03559
  The `buffer` used to create `pInfo->src` **must** be bound to device memory

## Valid Usage (Implicit)

- VUID-vkCmdCopyAccelerationStructureToMemoryKHR-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdCopyAccelerationStructureToMemoryKHR-pInfo-parameter
  `pInfo` **must** be a valid pointer to a valid VkCopyAccelerationStructureToMemoryInfoKHR structure
- VUID-vkCmdCopyAccelerationStructureToMemoryKHR-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdCopyAccelerationStructureToMemoryKHR-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support compute operations
- VUID-vkCmdCopyAccelerationStructureToMemoryKHR-renderpass
  This command **must** only be called outside of a render pass instance
- VUID-vkCmdCopyAccelerationStructureToMemoryKHR-videocoding
  This command **must** only be called outside of a video coding scope

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized

- Host access to the VkCommandPool that commandBuffer was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Outside | Outside | Compute | Action |

```
// Provided by VK_KHR_acceleration_structure
typedef struct VkCopyAccelerationStructureToMemoryInfoKHR {
    VkStructureType                         sType;
    const void*                             pNext;
    VkAccelerationStructureKHR              src;
    VkDeviceOrHostAddressKHR                dst;
    VkCopyAccelerationStructureModeKHR      mode;
} VkCopyAccelerationStructureToMemoryInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- src is the source acceleration structure for the copy
- dst is the device or host address to memory which is the target for the copy
- mode is a VkCopyAccelerationStructureModeKHR value specifying additional operations to perform during the copy.

## Valid Usage

- VUID-VkCopyAccelerationStructureToMemoryInfoKHR-src-04959
  The source acceleration structure src **must** have been constructed prior to the execution of this command
- VUID-VkCopyAccelerationStructureToMemoryInfoKHR-dst-03561
  The memory pointed to by dst **must** be at least as large as the serialization size of src, as reported by vkWriteAccelerationStructuresPropertiesKHR or vkCmdWriteAccelerationStructuresPropertiesKHR with a query type of VK_QUERY_TYPE_ACCELERATION_STRUCTURE_SERIALIZATION_SIZE_KHR
- VUID-VkCopyAccelerationStructureToMemoryInfoKHR-mode-03412
  mode **must** be VK_COPY_ACCELERATION_STRUCTURE_MODE_SERIALIZE_KHR

### Valid Usage (Implicit)

- VUID-VkCopyAccelerationStructureToMemoryInfoKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_COPY_ACCELERATION_STRUCTURE_TO_MEMORY_INFO_KHR`
- VUID-VkCopyAccelerationStructureToMemoryInfoKHR-pNext-pNext
  `pNext` **must** be `NULL`
- VUID-VkCopyAccelerationStructureToMemoryInfoKHR-src-parameter
  `src` **must** be a valid VkAccelerationStructureKHR handle
- VUID-VkCopyAccelerationStructureToMemoryInfoKHR-mode-parameter
  `mode` **must** be a valid VkCopyAccelerationStructureModeKHR value

To copy device memory to an acceleration structure call:

```
// Provided by VK_KHR_acceleration_structure
void vkCmdCopyMemoryToAccelerationStructureKHR(
    VkCommandBuffer                             commandBuffer,
    const VkCopyMemoryToAccelerationStructureInfoKHR* pInfo);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `pInfo` is a pointer to a VkCopyMemoryToAccelerationStructureInfoKHR structure defining the copy operation.

Accesses to `pInfo->dst` **must** be synchronized with the `VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_COPY_BIT_KHR` pipeline stage or the `VK_PIPELINE_STAGE_ACCELERATION_STRUCTURE_BUILD_BIT_KHR` pipeline stage, and an access type of `VK_ACCESS_ACCELERATION_STRUCTURE_WRITE_BIT_KHR`. Accesses to the buffer indicated by `pInfo->src.deviceAddress` **must** be synchronized with the `VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_COPY_BIT_KHR` pipeline stage or the `VK_PIPELINE_STAGE_ACCELERATION_STRUCTURE_BUILD_BIT_KHR` pipeline stage, and an access type of `VK_ACCESS_TRANSFER_READ_BIT`.

This command can accept acceleration structures produced by either vkCmdCopyAccelerationStructureToMemoryKHR or vkCopyAccelerationStructureToMemoryKHR.

The structure provided as input to deserialize is as described in vkCmdCopyAccelerationStructureToMemoryKHR, with any acceleration structure handles filled in with the newly-queried handles to bottom level acceleration structures created before deserialization. These do not need to be built at deserialize time, but **must** be created.

### Valid Usage

- VUID-vkCmdCopyMemoryToAccelerationStructureKHR-accelerationStructure-08927
  The `VkPhysicalDeviceAccelerationStructureFeaturesKHR::accelerationStructure` feature **must** be enabled

- VUID-vkCmdCopyMemoryToAccelerationStructureKHR-pInfo-03742
  `pInfo->src.deviceAddress` **must** be a valid device address for a buffer bound to device memory
- VUID-vkCmdCopyMemoryToAccelerationStructureKHR-pInfo-03743
  `pInfo->src.deviceAddress` **must** be aligned to `256` bytes
- VUID-vkCmdCopyMemoryToAccelerationStructureKHR-pInfo-03744
  If the buffer pointed to by `pInfo->src.deviceAddress` is non-sparse then it **must** be bound completely and contiguously to a single VkDeviceMemory object
- VUID-vkCmdCopyMemoryToAccelerationStructureKHR-buffer-03745
  The `buffer` used to create `pInfo->dst` **must** be bound to device memory

## Valid Usage (Implicit)

- VUID-vkCmdCopyMemoryToAccelerationStructureKHR-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdCopyMemoryToAccelerationStructureKHR-pInfo-parameter
  `pInfo` **must** be a valid pointer to a valid VkCopyMemoryToAccelerationStructureInfoKHR structure
- VUID-vkCmdCopyMemoryToAccelerationStructureKHR-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdCopyMemoryToAccelerationStructureKHR-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support compute operations
- VUID-vkCmdCopyMemoryToAccelerationStructureKHR-renderpass
  This command **must** only be called outside of a render pass instance
- VUID-vkCmdCopyMemoryToAccelerationStructureKHR-videocoding
  This command **must** only be called outside of a video coding scope

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Outside | Outside | Compute | Action |

The VkCopyMemoryToAccelerationStructureInfoKHR structure is defined as:

```
// Provided by VK_KHR_acceleration_structure
typedef struct VkCopyMemoryToAccelerationStructureInfoKHR {
    VkStructureType                         sType;
    const void*                             pNext;
    VkDeviceOrHostAddressConstKHR           src;
    VkAccelerationStructureKHR              dst;
    VkCopyAccelerationStructureModeKHR      mode;
} VkCopyMemoryToAccelerationStructureInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- src is the device or host address to memory containing the source data for the copy.
- dst is the target acceleration structure for the copy.
- mode is a VkCopyAccelerationStructureModeKHR value specifying additional operations to perform during the copy.

## Valid Usage

- VUID-VkCopyMemoryToAccelerationStructureInfoKHR-src-04960
  The source memory pointed to by src **must** contain data previously serialized using vkCmdCopyAccelerationStructureToMemoryKHR, potentially modified to relocate acceleration structure references as described in that command
- VUID-VkCopyMemoryToAccelerationStructureInfoKHR-mode-03413
  mode **must** be VK_COPY_ACCELERATION_STRUCTURE_MODE_DESERIALIZE_KHR
- VUID-VkCopyMemoryToAccelerationStructureInfoKHR-pInfo-03414
  The data in src **must** have a format compatible with the destination physical device as returned by vkGetDeviceAccelerationStructureCompatibilityKHR
- VUID-VkCopyMemoryToAccelerationStructureInfoKHR-dst-03746
  dst **must** have been created with a size greater than or equal to that used to serialize the data in src

## Valid Usage (Implicit)

- VUID-VkCopyMemoryToAccelerationStructureInfoKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_COPY_MEMORY_TO_ACCELERATION_STRUCTURE_INFO_KHR
- VUID-VkCopyMemoryToAccelerationStructureInfoKHR-pNext-pNext
  pNext **must** be NULL
- VUID-VkCopyMemoryToAccelerationStructureInfoKHR-dst-parameter
  dst **must** be a valid VkAccelerationStructureKHR handle
- VUID-VkCopyMemoryToAccelerationStructureInfoKHR-mode-parameter

mode **must** be a valid VkCopyAccelerationStructureModeKHR value

To check if a serialized acceleration structure is compatible with the current device call:

```
// Provided by VK_KHR_acceleration_structure
void vkGetDeviceAccelerationStructureCompatibilityKHR(
    VkDevice                                    device,
    const VkAccelerationStructureVersionInfoKHR* pVersionInfo,
    VkAccelerationStructureCompatibilityKHR*    pCompatibility);
```

- device is the device to check the version against.
- pVersionInfo is a pointer to a VkAccelerationStructureVersionInfoKHR structure specifying version information to check against the device.
- pCompatibility is a pointer to a VkAccelerationStructureCompatibilityKHR value in which compatibility information is returned.

### Valid Usage

- VUID-vkGetDeviceAccelerationStructureCompatibilityKHR-accelerationStructure-08928
  The VkPhysicalDeviceAccelerationStructureFeaturesKHR::accelerationStructure feature **must** be enabled

### Valid Usage (Implicit)

- VUID-vkGetDeviceAccelerationStructureCompatibilityKHR-device-parameter
  device **must** be a valid VkDevice handle
- VUID-vkGetDeviceAccelerationStructureCompatibilityKHR-pVersionInfo-parameter
  pVersionInfo **must** be a valid pointer to a valid VkAccelerationStructureVersionInfoKHR structure
- VUID-vkGetDeviceAccelerationStructureCompatibilityKHR-pCompatibility-parameter
  pCompatibility **must** be a valid pointer to a VkAccelerationStructureCompatibilityKHR value

The VkAccelerationStructureVersionInfoKHR structure is defined as:

```
// Provided by VK_KHR_acceleration_structure
typedef struct VkAccelerationStructureVersionInfoKHR {
    VkStructureType    sType;
    const void*        pNext;
    const uint8_t*     pVersionData;
} VkAccelerationStructureVersionInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- pVersionData is a pointer to the version header of an acceleration structure as defined in vkCmdCopyAccelerationStructureToMemoryKHR

**NOTE** pVersionData is a *pointer* to an array of 2×VK_UUID_SIZE uint8_t values instead of two VK_UUID_SIZE arrays as the expected use case for this member is to be pointed at the header of a previously serialized acceleration structure (via vkCmdCopyAccelerationStructureToMemoryKHR or vkCopyAccelerationStructureToMemoryKHR) that is loaded in memory. Using arrays would necessitate extra memory copies of the UUIDs.

### Valid Usage (Implicit)

- VUID-VkAccelerationStructureVersionInfoKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_ACCELERATION_STRUCTURE_VERSION_INFO_KHR
- VUID-VkAccelerationStructureVersionInfoKHR-pNext-pNext
  pNext **must** be NULL
- VUID-VkAccelerationStructureVersionInfoKHR-pVersionData-parameter
  pVersionData **must** be a valid pointer to an array of $2 \times VK_UUID_SIZE$ uint8_t values

Possible values of pCompatibility returned by vkGetDeviceAccelerationStructureCompatibilityKHR are:

```
// Provided by VK_KHR_acceleration_structure
typedef enum VkAccelerationStructureCompatibilityKHR {
    VK_ACCELERATION_STRUCTURE_COMPATIBILITY_COMPATIBLE_KHR = 0,
    VK_ACCELERATION_STRUCTURE_COMPATIBILITY_INCOMPATIBLE_KHR = 1,
} VkAccelerationStructureCompatibilityKHR;
```

- VK_ACCELERATION_STRUCTURE_COMPATIBILITY_COMPATIBLE_KHR if the pVersionData version acceleration structure is compatible with device.
- VK_ACCELERATION_STRUCTURE_COMPATIBILITY_INCOMPATIBLE_KHR if the pVersionData version acceleration structure is not compatible with device.

## 37.2. Host Acceleration Structure Operations

Implementations are also required to provide host implementations of the acceleration structure operations if the accelerationStructureHostCommands feature is enabled:

- vkBuildAccelerationStructuresKHR corresponding to vkCmdBuildAccelerationStructuresKHR
- vkCopyAccelerationStructureKHR corresponding to vkCmdCopyAccelerationStructureKHR
- vkCopyAccelerationStructureToMemoryKHR corresponding to

vkCmdCopyAccelerationStructureToMemoryKHR

- vkCopyMemoryToAccelerationStructureKHR corresponding to vkCmdCopyMemoryToAccelerationStructureKHR
- vkWriteAccelerationStructuresPropertiesKHR corresponding to vkCmdWriteAccelerationStructuresPropertiesKHR

These commands are functionally equivalent to their device counterparts, except that they are executed on the host timeline, rather than being enqueued into command buffers.

All acceleration structures used by the host commands **must** be bound to host-visible memory, and all input data for acceleration structure builds **must** be referenced using host addresses instead of device addresses. Applications are not required to map acceleration structure memory when using the host commands.

**NOTE**

The vkBuildAccelerationStructuresKHR and vkCmdBuildAccelerationStructuresKHR **may** use different algorithms, and thus are not required to produce identical structures. The structures produced by these two commands **may** exhibit different memory footprints or traversal performance, but should strive to be similar where possible.

Apart from these details, the host and device operations are interchangeable. For example, an application **can** use vkBuildAccelerationStructuresKHR to build a structure, compact it on the device using vkCmdCopyAccelerationStructureKHR, and serialize the result using vkCopyAccelerationStructureToMemoryKHR.

**NOTE**

For efficient execution, acceleration structures manipulated using these commands should always be bound to host cached memory, as the implementation may need to repeatedly read and write this memory during the execution of the command.

To build acceleration structures on the host, call:

```
// Provided by VK_KHR_acceleration_structure
VkResult vkBuildAccelerationStructuresKHR(
    VkDevice                                    device,
    VkDeferredOperationKHR                      deferredOperation,
    uint32_t                                    infoCount,
    const VkAccelerationStructureBuildGeometryInfoKHR* pInfos,
    const VkAccelerationStructureBuildRangeInfoKHR* const* ppBuildRangeInfos);
```

- `device` is the `VkDevice` for which the acceleration structures are being built.
- `deferredOperation` is an optional VkDeferredOperationKHR to request deferral for this command.
- `infoCount` is the number of acceleration structures to build. It specifies the number of the `pInfos` structures and `ppBuildRangeInfos` pointers that **must** be provided.
- `pInfos` is a pointer to an array of `infoCount` VkAccelerationStructureBuildGeometryInfoKHR structures defining the geometry used to build each acceleration structure.

- ppBuildRangeInfos is a pointer to an array of infoCount pointers to arrays of VkAccelerationStructureBuildRangeInfoKHR structures. Each ppBuildRangeInfos[i] is a pointer to an array of pInfos[i].geometryCount VkAccelerationStructureBuildRangeInfoKHR structures defining dynamic offsets to the addresses where geometry data is stored, as defined by pInfos[i].

This command fulfills the same task as vkCmdBuildAccelerationStructuresKHR but is executed by the host.

The vkBuildAccelerationStructuresKHR command provides the ability to initiate multiple acceleration structures builds, however there is no ordering or synchronization implied between any of the individual acceleration structure builds.

**NOTE** This means that an application **cannot** build a top-level acceleration structure in the same vkBuildAccelerationStructuresKHR call as the associated bottom-level or instance acceleration structures are being built. There also **cannot** be any memory aliasing between any acceleration structure memories or scratch memories being used by any of the builds.

## Valid Usage

- VUID-vkBuildAccelerationStructuresKHR-accelerationStructureHostCommands-03581
  The VkPhysicalDeviceAccelerationStructureFeaturesKHR::accelerationStructureHostCommands feature **must** be enabled
- VUID-vkBuildAccelerationStructuresKHR-mode-04628
  The mode member of each element of pInfos **must** be a valid VkBuildAccelerationStructureModeKHR value
- VUID-vkBuildAccelerationStructuresKHR-srcAccelerationStructure-04629
  If the srcAccelerationStructure member of any element of pInfos is not VK_NULL_HANDLE, the srcAccelerationStructure member **must** be a valid VkAccelerationStructureKHR handle
- VUID-vkBuildAccelerationStructuresKHR-pInfos-04630
  For each element of pInfos, if its mode member is VK_BUILD_ACCELERATION_STRUCTURE_MODE_UPDATE_KHR, its srcAccelerationStructure member **must** not be VK_NULL_HANDLE
- VUID-vkBuildAccelerationStructuresKHR-pInfos-03403
  The srcAccelerationStructure member of any element of pInfos **must** not be the same acceleration structure as the dstAccelerationStructure member of any other element of pInfos
- VUID-vkBuildAccelerationStructuresKHR-dstAccelerationStructure-03698
  The dstAccelerationStructure member of any element of pInfos **must** not be the same acceleration structure as the dstAccelerationStructure member of any other element of pInfos
- VUID-vkBuildAccelerationStructuresKHR-dstAccelerationStructure-03800
  The dstAccelerationStructure member of any element of pInfos **must** be a valid VkAccelerationStructureKHR handle

- VUID-vkBuildAccelerationStructuresKHR-pInfos-03699
  For each element of pInfos, if its type member is VK_ACCELERATION_STRUCTURE_TYPE_TOP_LEVEL_KHR, its dstAccelerationStructure member **must** have been created with a value of VkAccelerationStructureCreateInfoKHR::type equal to either VK_ACCELERATION_STRUCTURE_TYPE_TOP_LEVEL_KHR or VK_ACCELERATION_STRUCTURE_TYPE_GENERIC_KHR
- VUID-vkBuildAccelerationStructuresKHR-pInfos-03700
  For each element of pInfos, if its type member is VK_ACCELERATION_STRUCTURE_TYPE_BOTTOM_LEVEL_KHR, its dstAccelerationStructure member **must** have been created with a value of VkAccelerationStructureCreateInfoKHR::type equal to either VK_ACCELERATION_STRUCTURE_TYPE_BOTTOM_LEVEL_KHR or VK_ACCELERATION_STRUCTURE_TYPE_GENERIC_KHR
- VUID-vkBuildAccelerationStructuresKHR-pInfos-03663
  For each element of pInfos, if its mode member is VK_BUILD_ACCELERATION_STRUCTURE_MODE_UPDATE_KHR, inactive primitives in its srcAccelerationStructure member **must** not be made active
- VUID-vkBuildAccelerationStructuresKHR-pInfos-03664
  For each element of pInfos, if its mode member is VK_BUILD_ACCELERATION_STRUCTURE_MODE_UPDATE_KHR, active primitives in its srcAccelerationStructure member **must** not be made inactive
- VUID-vkBuildAccelerationStructuresKHR-None-03407
  The dstAccelerationStructure member of any element of pInfos **must** not be referenced by the geometry.instances.data member of any element of pGeometries or ppGeometries with a geometryType of VK_GEOMETRY_TYPE_INSTANCES_KHR in any other element of pInfos
- VUID-vkBuildAccelerationStructuresKHR-dstAccelerationStructure-03701
  The range of memory backing the dstAccelerationStructure member of any element of pInfos that is accessed by this command **must** not overlap the memory backing the srcAccelerationStructure member of any other element of pInfos with a mode equal to VK_BUILD_ACCELERATION_STRUCTURE_MODE_UPDATE_KHR, which is accessed by this command
- VUID-vkBuildAccelerationStructuresKHR-dstAccelerationStructure-03702
  The range of memory backing the dstAccelerationStructure member of any element of pInfos that is accessed by this command **must** not overlap the memory backing the dstAccelerationStructure member of any other element of pInfos, which is accessed by this command
- VUID-vkBuildAccelerationStructuresKHR-dstAccelerationStructure-03703
  The range of memory backing the dstAccelerationStructure member of any element of pInfos that is accessed by this command **must** not overlap the memory backing the scratchData member of any element of pInfos (including the same element), which is accessed by this command
- VUID-vkBuildAccelerationStructuresKHR-scratchData-03704
  The range of memory backing the scratchData member of any element of pInfos that is accessed by this command **must** not overlap the memory backing the scratchData member of any other element of pInfos, which is accessed by this command
- VUID-vkBuildAccelerationStructuresKHR-scratchData-03705

The range of memory backing the scratchData member of any element of pInfos that is accessed by this command **must** not overlap the memory backing the srcAccelerationStructure member of any element of pInfos with a mode equal to VK_BUILD_ACCELERATION_STRUCTURE_MODE_UPDATE_KHR (including the same element), which is accessed by this command

- VUID-vkBuildAccelerationStructuresKHR-dstAccelerationStructure-03706
  The range of memory backing the dstAccelerationStructure member of any element of pInfos that is accessed by this command **must** not overlap the memory backing any acceleration structure referenced by the geometry.instances.data member of any element of pGeometries or ppGeometries with a geometryType of VK_GEOMETRY_TYPE_INSTANCES_KHR in any other element of pInfos, which is accessed by this command
- VUID-vkBuildAccelerationStructuresKHR-pInfos-03667
  For each element of pInfos, if its mode member is VK_BUILD_ACCELERATION_STRUCTURE_MODE_UPDATE_KHR, its srcAccelerationStructure member **must** have previously been constructed with VK_BUILD_ACCELERATION_STRUCTURE_ALLOW_UPDATE_BIT_KHR set in VkAccelerationStructureBuildGeometryInfoKHR::flags in the build
- VUID-vkBuildAccelerationStructuresKHR-pInfos-03668
  For each element of pInfos, if its mode member is VK_BUILD_ACCELERATION_STRUCTURE_MODE_UPDATE_KHR, its srcAccelerationStructure and dstAccelerationStructure members **must** either be the same VkAccelerationStructureKHR, or not have any memory aliasing
- VUID-vkBuildAccelerationStructuresKHR-pInfos-03758
  For each element of pInfos, if its mode member is VK_BUILD_ACCELERATION_STRUCTURE_MODE_UPDATE_KHR, its geometryCount member **must** have the same value which was specified when srcAccelerationStructure was last built
- VUID-vkBuildAccelerationStructuresKHR-pInfos-03759
  For each element of pInfos, if its mode member is VK_BUILD_ACCELERATION_STRUCTURE_MODE_UPDATE_KHR, its flags member **must** have the same value which was specified when srcAccelerationStructure was last built
- VUID-vkBuildAccelerationStructuresKHR-pInfos-03760
  For each element of pInfos, if its mode member is VK_BUILD_ACCELERATION_STRUCTURE_MODE_UPDATE_KHR, its type member **must** have the same value which was specified when srcAccelerationStructure was last built
- VUID-vkBuildAccelerationStructuresKHR-pInfos-03761
  For each element of pInfos, if its mode member is VK_BUILD_ACCELERATION_STRUCTURE_MODE_UPDATE_KHR, then for each VkAccelerationStructureGeometryKHR structure referred to by its pGeometries or ppGeometries members, its geometryType member **must** have the same value which was specified when srcAccelerationStructure was last built
- VUID-vkBuildAccelerationStructuresKHR-pInfos-03762
  For each element of pInfos, if its mode member is VK_BUILD_ACCELERATION_STRUCTURE_MODE_UPDATE_KHR, then for each VkAccelerationStructureGeometryKHR structure referred to by its pGeometries or ppGeometries members, its flags member **must** have the same value which was specified

when srcAccelerationStructure was last built

- VUID-vkBuildAccelerationStructuresKHR-pInfos-03763
  For each element of pInfos, if its mode member is VK_BUILD_ACCELERATION_STRUCTURE_MODE_UPDATE_KHR, then for each VkAccelerationStructureGeometryKHR structure referred to by its pGeometries or ppGeometries members, if geometryType is VK_GEOMETRY_TYPE_TRIANGLES_KHR, its geometry.triangles.vertexFormat member **must** have the same value which was specified when srcAccelerationStructure was last built
- VUID-vkBuildAccelerationStructuresKHR-pInfos-03764
  For each element of pInfos, if its mode member is VK_BUILD_ACCELERATION_STRUCTURE_MODE_UPDATE_KHR, then for each VkAccelerationStructureGeometryKHR structure referred to by its pGeometries or ppGeometries members, if geometryType is VK_GEOMETRY_TYPE_TRIANGLES_KHR, its geometry.triangles.maxVertex member **must** have the same value which was specified when srcAccelerationStructure was last built
- VUID-vkBuildAccelerationStructuresKHR-pInfos-03765
  For each element of pInfos, if its mode member is VK_BUILD_ACCELERATION_STRUCTURE_MODE_UPDATE_KHR, then for each VkAccelerationStructureGeometryKHR structure referred to by its pGeometries or ppGeometries members, if geometryType is VK_GEOMETRY_TYPE_TRIANGLES_KHR, its geometry.triangles.indexType member **must** have the same value which was specified when srcAccelerationStructure was last built
- VUID-vkBuildAccelerationStructuresKHR-pInfos-03766
  For each element of pInfos, if its mode member is VK_BUILD_ACCELERATION_STRUCTURE_MODE_UPDATE_KHR, then for each VkAccelerationStructureGeometryKHR structure referred to by its pGeometries or ppGeometries members, if geometryType is VK_GEOMETRY_TYPE_TRIANGLES_KHR, if its geometry.triangles.transformData address was NULL when srcAccelerationStructure was last built, then it **must** be NULL
- VUID-vkBuildAccelerationStructuresKHR-pInfos-03767
  For each element of pInfos, if its mode member is VK_BUILD_ACCELERATION_STRUCTURE_MODE_UPDATE_KHR, then for each VkAccelerationStructureGeometryKHR structure referred to by its pGeometries or ppGeometries members, if geometryType is VK_GEOMETRY_TYPE_TRIANGLES_KHR, if its geometry.triangles.transformData address was not NULL when srcAccelerationStructure was last built, then it **must** not be NULL
- VUID-vkBuildAccelerationStructuresKHR-pInfos-03768
  For each element of pInfos, if its mode member is VK_BUILD_ACCELERATION_STRUCTURE_MODE_UPDATE_KHR, then for each VkAccelerationStructureGeometryKHR structure referred to by its pGeometries or ppGeometries members, if geometryType is VK_GEOMETRY_TYPE_TRIANGLES_KHR, and geometry.triangles.indexType is not VK_INDEX_TYPE_NONE_KHR, then the value of each index referenced **must** be the same as the corresponding index value when srcAccelerationStructure was last built
- VUID-vkBuildAccelerationStructuresKHR-primitiveCount-03769

For each element of pInfos, if its mode member is VK_BUILD_ACCELERATION_STRUCTURE_MODE_UPDATE_KHR, then for each VkAccelerationStructureGeometryKHR structure referred to by its pGeometries or ppGeometries members, the primitiveCount member of its corresponding VkAccelerationStructureBuildRangeInfoKHR structure **must** have the same value which was specified when srcAccelerationStructure was last built

- VUID-vkBuildAccelerationStructuresKHR-pInfos-03801
  For each element of pInfos[i].pGeometries or pInfos[i].ppGeometries with a geometryType of VK_GEOMETRY_TYPE_INSTANCES_KHR, the corresponding ppBuildRangeInfos[i][j].primitiveCount **must** be less than or equal to VkPhysicalDeviceAccelerationStructurePropertiesKHR::maxInstanceCount

- VUID-vkBuildAccelerationStructuresKHR-pInfos-10126
  For each pInfos[i], dstAccelerationStructure **must** have been created with a value of VkAccelerationStructureCreateInfoKHR::size greater than or equal to either:
  - the memory size required by the build operation, as returned by vkGetAccelerationStructureBuildSizesKHR with pBuildInfo = pInfos[i] and with each element of the pMaxPrimitiveCounts array greater than or equal to the equivalent ppBuildRangeInfos[i][j].primitiveCount values for j in [0,pInfos[i].geometryCount) or,
  - the result of querying the corresponding VK_QUERY_TYPE_ACCELERATION_STRUCTURE_COMPACTED_SIZE_KHR, if updating a compacted acceleration structure

- VUID-vkBuildAccelerationStructuresKHR-ppBuildRangeInfos-03676
  Each element of ppBuildRangeInfos[i] **must** be a valid pointer to an array of pInfos[i].geometryCount VkAccelerationStructureBuildRangeInfoKHR structures

- VUID-vkBuildAccelerationStructuresKHR-deferredOperation-03678
  Any previous deferred operation that was associated with deferredOperation **must** be complete

- VUID-vkBuildAccelerationStructuresKHR-pInfos-03722
  For each element of pInfos, the buffer used to create its dstAccelerationStructure member **must** be bound to host-visible device memory

- VUID-vkBuildAccelerationStructuresKHR-pInfos-03723
  For each element of pInfos, if its mode member is VK_BUILD_ACCELERATION_STRUCTURE_MODE_UPDATE_KHR the buffer used to create its srcAccelerationStructure member **must** be bound to host-visible device memory

- VUID-vkBuildAccelerationStructuresKHR-pInfos-03724
  For each element of pInfos, the buffer used to create each acceleration structure referenced by the geometry.instances.data member of any element of pGeometries or ppGeometries with a geometryType of VK_GEOMETRY_TYPE_INSTANCES_KHR **must** be bound to host-visible device memory

- VUID-vkBuildAccelerationStructuresKHR-pInfos-03725
  If pInfos[i].mode is VK_BUILD_ACCELERATION_STRUCTURE_MODE_BUILD_KHR, all addresses between pInfos[i].scratchData.hostAddress and pInfos[i].scratchData.hostAddress + N - 1 **must** be valid host memory, where N is given by the buildScratchSize member of the

VkAccelerationStructureBuildSizesInfoKHR structure returned from a call to vkGetAccelerationStructureBuildSizesKHR with an identical VkAccelerationStructureBuildGeometryInfoKHR structure and primitive count

- VUID-vkBuildAccelerationStructuresKHR-pInfos-03726
  If pInfos[i].mode is VK_BUILD_ACCELERATION_STRUCTURE_MODE_UPDATE_KHR, all addresses between pInfos[i].scratchData.hostAddress and pInfos[i].scratchData.hostAddress + N - 1 **must** be valid host memory, where N is given by the updateScratchSize member of the VkAccelerationStructureBuildSizesInfoKHR structure returned from a call to vkGetAccelerationStructureBuildSizesKHR with an identical VkAccelerationStructureBuildGeometryInfoKHR structure and primitive count
- VUID-vkBuildAccelerationStructuresKHR-pInfos-03771
  For any element of pInfos[i].pGeometries or pInfos[i].ppGeometries with a geometryType of VK_GEOMETRY_TYPE_TRIANGLES_KHR, geometry.triangles.vertexData.hostAddress **must** be a valid host address
- VUID-vkBuildAccelerationStructuresKHR-pInfos-03772
  For any element of pInfos[i].pGeometries or pInfos[i].ppGeometries with a geometryType of VK_GEOMETRY_TYPE_TRIANGLES_KHR, if geometry.triangles.indexType is not VK_INDEX_TYPE_NONE_KHR, geometry.triangles.indexData.hostAddress **must** be a valid host address
- VUID-vkBuildAccelerationStructuresKHR-pInfos-03773
  For any element of pInfos[i].pGeometries or pInfos[i].ppGeometries with a geometryType of VK_GEOMETRY_TYPE_TRIANGLES_KHR, if geometry.triangles.transformData.hostAddress is not 0, it **must** be a valid host address
- VUID-vkBuildAccelerationStructuresKHR-pInfos-03774
  For any element of pInfos[i].pGeometries or pInfos[i].ppGeometries with a geometryType of VK_GEOMETRY_TYPE_AABBS_KHR, geometry.aabbs.data.hostAddress **must** be a valid host address
- VUID-vkBuildAccelerationStructuresKHR-pInfos-03775
  For each element of pInfos, the buffer used to create its dstAccelerationStructure member **must** be bound to memory that was not allocated with multiple instances
- VUID-vkBuildAccelerationStructuresKHR-pInfos-03776
  For each element of pInfos, if its mode member is VK_BUILD_ACCELERATION_STRUCTURE_MODE_UPDATE_KHR the buffer used to create its srcAccelerationStructure member **must** be bound to memory that was not allocated with multiple instances
- VUID-vkBuildAccelerationStructuresKHR-pInfos-03777
  For each element of pInfos, the buffer used to create each acceleration structure referenced by the geometry.instances.data member of any element of pGeometries or ppGeometries with a geometryType of VK_GEOMETRY_TYPE_INSTANCES_KHR **must** be bound to memory that was not allocated with multiple instances
- VUID-vkBuildAccelerationStructuresKHR-pInfos-03778
  For any element of pInfos[i].pGeometries or pInfos[i].ppGeometries with a geometryType of VK_GEOMETRY_TYPE_INSTANCES_KHR, geometry.instances.data.hostAddress **must** be a valid host address
- VUID-vkBuildAccelerationStructuresKHR-pInfos-03779

For any element of pInfos[i].pGeometries or pInfos[i].ppGeometries with a geometryType of VK_GEOMETRY_TYPE_INSTANCES_KHR, each VkAccelerationStructureInstanceKHR::accelerationStructureReference value in geometry.instances.data.hostAddress **must** be a valid VkAccelerationStructureKHR object

- VUID-vkBuildAccelerationStructuresKHR-pInfos-04930
  For any element of pInfos[i].pGeometries or pInfos[i].ppGeometries with a geometryType of VK_GEOMETRY_TYPE_INSTANCES_KHR with VK_BUILD_ACCELERATION_STRUCTURE_MOTION_BIT_NV set, each accelerationStructureReference in any structure in VkAccelerationStructureMotionInstanceNV value in geometry.instances.data.hostAddress **must** be a valid VkAccelerationStructureKHR object

## Valid Usage (Implicit)

- VUID-vkBuildAccelerationStructuresKHR-device-parameter
  device **must** be a valid VkDevice handle
- VUID-vkBuildAccelerationStructuresKHR-deferredOperation-parameter
  If deferredOperation is not VK_NULL_HANDLE, deferredOperation **must** be a valid VkDeferredOperationKHR handle
- VUID-vkBuildAccelerationStructuresKHR-pInfos-parameter
  pInfos **must** be a valid pointer to an array of infoCount valid VkAccelerationStructureBuildGeometryInfoKHR structures
- VUID-vkBuildAccelerationStructuresKHR-ppBuildRangeInfos-parameter
  ppBuildRangeInfos **must** be a valid pointer to an array of infoCount VkAccelerationStructureBuildRangeInfoKHR structures
- VUID-vkBuildAccelerationStructuresKHR-infoCount-arraylength
  infoCount **must** be greater than 0
- VUID-vkBuildAccelerationStructuresKHR-deferredOperation-parent
  If deferredOperation is a valid handle, it **must** have been created, allocated, or retrieved from device

## Return Codes

### Success

- VK_SUCCESS
- VK_OPERATION_DEFERRED_KHR
- VK_OPERATION_NOT_DEFERRED_KHR

### Failure

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY

To copy or compact an acceleration structure on the host, call:

```
// Provided by VK_KHR_acceleration_structure
VkResult vkCopyAccelerationStructureKHR(
    VkDevice                                    device,
    VkDeferredOperationKHR                      deferredOperation,
    const VkCopyAccelerationStructureInfoKHR*   pInfo);
```

- `device` is the device which owns the acceleration structures.
- `deferredOperation` is an optional VkDeferredOperationKHR to request deferral for this command.
- `pInfo` is a pointer to a VkCopyAccelerationStructureInfoKHR structure defining the copy operation.

This command fulfills the same task as vkCmdCopyAccelerationStructureKHR but is executed by the host.

## Valid Usage

- VUID-vkCopyAccelerationStructureKHR-accelerationStructureHostCommands-03582
  The `VkPhysicalDeviceAccelerationStructureFeaturesKHR::accelerationStructureHostCommands` feature **must** be enabled
- VUID-vkCopyAccelerationStructureKHR-deferredOperation-03678
  Any previous deferred operation that was associated with `deferredOperation` **must** be complete
- VUID-vkCopyAccelerationStructureKHR-buffer-03727
  The `buffer` used to create `pInfo->src` **must** be bound to host-visible device memory
- VUID-vkCopyAccelerationStructureKHR-buffer-03728
  The `buffer` used to create `pInfo->dst` **must** be bound to host-visible device memory
- VUID-vkCopyAccelerationStructureKHR-buffer-03780
  The `buffer` used to create `pInfo->src` **must** be bound to memory that was not allocated with multiple instances
- VUID-vkCopyAccelerationStructureKHR-buffer-03781
  The `buffer` used to create `pInfo->dst` **must** be bound to memory that was not allocated with multiple instances

## Valid Usage (Implicit)

- VUID-vkCopyAccelerationStructureKHR-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkCopyAccelerationStructureKHR-deferredOperation-parameter
  If `deferredOperation` is not VK_NULL_HANDLE, `deferredOperation` **must** be a valid VkDeferredOperationKHR handle
- VUID-vkCopyAccelerationStructureKHR-pInfo-parameter

pInfo **must** be a valid pointer to a valid VkCopyAccelerationStructureInfoKHR structure

- VUID-vkCopyAccelerationStructureKHR-deferredOperation-parent
  If deferredOperation is a valid handle, it **must** have been created, allocated, or retrieved from device

## Return Codes

**Success**

- VK_SUCCESS
- VK_OPERATION_DEFERRED_KHR
- VK_OPERATION_NOT_DEFERRED_KHR

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY

To copy host accessible memory to an acceleration structure, call:

```
// Provided by VK_KHR_acceleration_structure
VkResult vkCopyMemoryToAccelerationStructureKHR(
    VkDevice                                    device,
    VkDeferredOperationKHR                      deferredOperation,
    const VkCopyMemoryToAccelerationStructureInfoKHR* pInfo);
```

- device is the device which owns pInfo->dst.
- deferredOperation is an optional VkDeferredOperationKHR to request deferral for this command.
- pInfo is a pointer to a VkCopyMemoryToAccelerationStructureInfoKHR structure defining the copy operation.

This command fulfills the same task as vkCmdCopyMemoryToAccelerationStructureKHR but is executed by the host.

This command can accept acceleration structures produced by either vkCmdCopyAccelerationStructureToMemoryKHR or vkCopyAccelerationStructureToMemoryKHR.

## Valid Usage

- VUID-vkCopyMemoryToAccelerationStructureKHR-accelerationStructureHostCommands-03583
  The VkPhysicalDeviceAccelerationStructureFeaturesKHR::accelerationStructureHostCommands feature **must** be enabled
- VUID-vkCopyMemoryToAccelerationStructureKHR-deferredOperation-03678

Any previous deferred operation that was associated with `deferredOperation` **must** be complete

- VUID-vkCopyMemoryToAccelerationStructureKHR-pInfo-03729
  `pInfo->src.hostAddress` **must** be a valid host pointer
- VUID-vkCopyMemoryToAccelerationStructureKHR-pInfo-03750
  `pInfo->src.hostAddress` **must** be aligned to 16 bytes
- VUID-vkCopyMemoryToAccelerationStructureKHR-buffer-03730
  The `buffer` used to create `pInfo->dst` **must** be bound to host-visible device memory
- VUID-vkCopyMemoryToAccelerationStructureKHR-buffer-03782
  The `buffer` used to create `pInfo->dst` **must** be bound to memory that was not allocated with multiple instances

## Valid Usage (Implicit)

- VUID-vkCopyMemoryToAccelerationStructureKHR-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkCopyMemoryToAccelerationStructureKHR-deferredOperation-parameter
  If `deferredOperation` is not VK_NULL_HANDLE, `deferredOperation` **must** be a valid VkDeferredOperationKHR handle
- VUID-vkCopyMemoryToAccelerationStructureKHR-pInfo-parameter
  `pInfo` **must** be a valid pointer to a valid VkCopyMemoryToAccelerationStructureInfoKHR structure
- VUID-vkCopyMemoryToAccelerationStructureKHR-deferredOperation-parent
  If `deferredOperation` is a valid handle, it **must** have been created, allocated, or retrieved from `device`

## Return Codes

**Success**

- `VK_SUCCESS`
- `VK_OPERATION_DEFERRED_KHR`
- `VK_OPERATION_NOT_DEFERRED_KHR`

**Failure**

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`

To copy an acceleration structure to host accessible memory, call:

```
// Provided by VK_KHR_acceleration_structure
VkResult vkCopyAccelerationStructureToMemoryKHR(
    VkDevice                                    device,
    VkDeferredOperationKHR                      deferredOperation,
    const VkCopyAccelerationStructureToMemoryInfoKHR* pInfo);
```

- `device` is the device which owns `pInfo->src`.
- `deferredOperation` is an optional VkDeferredOperationKHR to request deferral for this command.
- `pInfo` is a pointer to a VkCopyAccelerationStructureToMemoryInfoKHR structure defining the copy operation.

This command fulfills the same task as vkCmdCopyAccelerationStructureToMemoryKHR but is executed by the host.

This command produces the same results as vkCmdCopyAccelerationStructureToMemoryKHR, but writes its result directly to a host pointer, and is executed on the host rather than the device. The output **may** not necessarily be bit-for-bit identical, but it can be equally used by either vkCmdCopyMemoryToAccelerationStructureKHR or vkCopyMemoryToAccelerationStructureKHR.

### Valid Usage

- VUID-vkCopyAccelerationStructureToMemoryKHR-accelerationStructureHostCommands-03584
  The `VkPhysicalDeviceAccelerationStructureFeaturesKHR::accelerationStructureHostCommands` feature **must** be enabled
- VUID-vkCopyAccelerationStructureToMemoryKHR-deferredOperation-03678
  Any previous deferred operation that was associated with `deferredOperation` **must** be complete
- VUID-vkCopyAccelerationStructureToMemoryKHR-buffer-03731
  The `buffer` used to create `pInfo->src` **must** be bound to host-visible device memory
- VUID-vkCopyAccelerationStructureToMemoryKHR-pInfo-03732
  `pInfo->dst.hostAddress` **must** be a valid host pointer
- VUID-vkCopyAccelerationStructureToMemoryKHR-pInfo-03751
  `pInfo->dst.hostAddress` **must** be aligned to 16 bytes
- VUID-vkCopyAccelerationStructureToMemoryKHR-buffer-03783
  The `buffer` used to create `pInfo->src` **must** be bound to memory that was not allocated with multiple instances

### Valid Usage (Implicit)

- VUID-vkCopyAccelerationStructureToMemoryKHR-device-parameter
  `device` **must** be a valid VkDevice handle

- VUID-vkCopyAccelerationStructureToMemoryKHR-deferredOperation-parameter
  If `deferredOperation` is not VK_NULL_HANDLE, `deferredOperation` **must** be a valid VkDeferredOperationKHR handle
- VUID-vkCopyAccelerationStructureToMemoryKHR-pInfo-parameter
  `pInfo` **must** be a valid pointer to a valid VkCopyAccelerationStructureToMemoryInfoKHR structure
- VUID-vkCopyAccelerationStructureToMemoryKHR-deferredOperation-parent
  If `deferredOperation` is a valid handle, it **must** have been created, allocated, or retrieved from `device`

## Return Codes

**Success**

- `VK_SUCCESS`
- `VK_OPERATION_DEFERRED_KHR`
- `VK_OPERATION_NOT_DEFERRED_KHR`

**Failure**

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`

To query acceleration structure size parameters on the host, call:

```
// Provided by VK_KHR_acceleration_structure
VkResult vkWriteAccelerationStructuresPropertiesKHR(
    VkDevice                                    device,
    uint32_t                                    accelerationStructureCount,
    const VkAccelerationStructureKHR*           pAccelerationStructures,
    VkQueryType                                 queryType,
    size_t                                      dataSize,
    void*                                       pData,
    size_t                                      stride);
```

- `device` is the device which owns the acceleration structures in `pAccelerationStructures`.
- `accelerationStructureCount` is the count of acceleration structures for which to query the property.
- `pAccelerationStructures` is a pointer to an array of existing previously built acceleration structures.
- `queryType` is a VkQueryType value specifying the property to be queried.
- `dataSize` is the size in bytes of the buffer pointed to by `pData`.
- `pData` is a pointer to an application-allocated buffer where the results will be written.
- `stride` is the stride in bytes between results for individual queries within `pData`.

This command fulfills the same task as vkCmdWriteAccelerationStructuresPropertiesKHR but is executed by the host.

## Valid Usage

- VUID-vkWriteAccelerationStructuresPropertiesKHR-accelerationStructureHostCommands-03585
  The VkPhysicalDeviceAccelerationStructureFeaturesKHR::accelerationStructureHostCommands feature **must** be enabled
- VUID-vkWriteAccelerationStructuresPropertiesKHR-pAccelerationStructures-04964
  All acceleration structures in pAccelerationStructures **must** have been built prior to the execution of this command
- VUID-vkWriteAccelerationStructuresPropertiesKHR-accelerationStructures-03431
  All acceleration structures in pAccelerationStructures **must** have been built with VK_BUILD_ACCELERATION_STRUCTURE_ALLOW_COMPACTION_BIT_KHR if queryType is VK_QUERY_TYPE_ACCELERATION_STRUCTURE_COMPACTED_SIZE_KHR
- VUID-vkWriteAccelerationStructuresPropertiesKHR-queryType-06742
  queryType **must** be VK_QUERY_TYPE_ACCELERATION_STRUCTURE_SIZE_KHR, VK_QUERY_TYPE_ACCELERATION_STRUCTURE_SERIALIZATION_BOTTOM_LEVEL_POINTERS_KHR, VK_QUERY_TYPE_ACCELERATION_STRUCTURE_COMPACTED_SIZE_KHR, or VK_QUERY_TYPE_ACCELERATION_STRUCTURE_SERIALIZATION_SIZE_KHR
- VUID-vkWriteAccelerationStructuresPropertiesKHR-queryType-03448
  If queryType is VK_QUERY_TYPE_ACCELERATION_STRUCTURE_COMPACTED_SIZE_KHR, then stride **must** be a multiple of the size of VkDeviceSize
- VUID-vkWriteAccelerationStructuresPropertiesKHR-queryType-03449
  If queryType is VK_QUERY_TYPE_ACCELERATION_STRUCTURE_COMPACTED_SIZE_KHR, then pData **must** point to a VkDeviceSize
- VUID-vkWriteAccelerationStructuresPropertiesKHR-queryType-03450
  If queryType is VK_QUERY_TYPE_ACCELERATION_STRUCTURE_SERIALIZATION_SIZE_KHR, then stride **must** be a multiple of the size of VkDeviceSize
- VUID-vkWriteAccelerationStructuresPropertiesKHR-queryType-03451
  If queryType is VK_QUERY_TYPE_ACCELERATION_STRUCTURE_SERIALIZATION_SIZE_KHR, then pData **must** point to a VkDeviceSize
- VUID-vkWriteAccelerationStructuresPropertiesKHR-queryType-06731
  If queryType is VK_QUERY_TYPE_ACCELERATION_STRUCTURE_SIZE_KHR, then stride **must** be a multiple of the size of VkDeviceSize
- VUID-vkWriteAccelerationStructuresPropertiesKHR-queryType-06732
  If queryType is VK_QUERY_TYPE_ACCELERATION_STRUCTURE_SIZE_KHR, then pData **must** point to a VkDeviceSize
- VUID-vkWriteAccelerationStructuresPropertiesKHR-queryType-06733
  If queryType is VK_QUERY_TYPE_ACCELERATION_STRUCTURE_SERIALIZATION_BOTTOM_LEVEL_POINTERS_KHR, then stride **must** be a multiple of the size of VkDeviceSize

- VUID-vkWriteAccelerationStructuresPropertiesKHR-queryType-06734
  If `queryType` is `VK_QUERY_TYPE_ACCELERATION_STRUCTURE_SERIALIZATION_BOTTOM_LEVEL_POINTERS_KHR`, then `pData` **must** point to a VkDeviceSize
- VUID-vkWriteAccelerationStructuresPropertiesKHR-dataSize-03452
  `dataSize` **must** be greater than or equal to `accelerationStructureCount`*`stride`
- VUID-vkWriteAccelerationStructuresPropertiesKHR-buffer-03733
  The `buffer` used to create each acceleration structure in `pAccelerationStructures` **must** be bound to host-visible device memory
- VUID-vkWriteAccelerationStructuresPropertiesKHR-buffer-03784
  The `buffer` used to create each acceleration structure in `pAccelerationStructures` **must** be bound to memory that was not allocated with multiple instances

## Valid Usage (Implicit)

- VUID-vkWriteAccelerationStructuresPropertiesKHR-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkWriteAccelerationStructuresPropertiesKHR-pAccelerationStructures-parameter
  `pAccelerationStructures` **must** be a valid pointer to an array of `accelerationStructureCount` valid VkAccelerationStructureKHR handles
- VUID-vkWriteAccelerationStructuresPropertiesKHR-queryType-parameter
  `queryType` **must** be a valid VkQueryType value
- VUID-vkWriteAccelerationStructuresPropertiesKHR-pData-parameter
  `pData` **must** be a valid pointer to an array of `dataSize` bytes
- VUID-vkWriteAccelerationStructuresPropertiesKHR-accelerationStructureCount-arraylength
  `accelerationStructureCount` **must** be greater than `0`
- VUID-vkWriteAccelerationStructuresPropertiesKHR-dataSize-arraylength
  `dataSize` **must** be greater than `0`
- VUID-vkWriteAccelerationStructuresPropertiesKHR-pAccelerationStructures-parent
  Each element of `pAccelerationStructures` **must** have been created, allocated, or retrieved from `device`

## Return Codes

### Success

- `VK_SUCCESS`

### Failure

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`

# Chapter 38. Micromap

## 38.1. Micromaps

*Acceleration structures* store and organize geometry for ray tracing, but in some cases it is beneficial to include some information within the geometry, particularly for triangles. A *micromap* organizes this data around a map of values corresponding to subdivided microtriangles which can be added to a triangle geometry when building a *bottom level acceleration structure*.

An *opacity micromap* is a type of micromap which stores information to control intersection opacity as described in Ray Opacity Micromap.

A *displacement micromap* is a type of micromap which stores information to displace sub-triangle vertices as described in Displacement Micromap.

A micromap is considered to be constructed if a micromap build command or copy command has been executed with the given acceleration structure as the destination.

### 38.1.1. Building Micromaps

To build micromaps call:

```
// Provided by VK_EXT_opacity_micromap
void vkCmdBuildMicromapsEXT(
    VkCommandBuffer                             commandBuffer,
    uint32_t                                    infoCount,
    const VkMicromapBuildInfoEXT*               pInfos);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `infoCount` is the number of micromaps to build. It specifies the number of the `pInfos` structures that **must** be provided.
- `pInfos` is a pointer to an array of `infoCount` VkMicromapBuildInfoEXT structures defining the data used to build each micromap.

The `vkCmdBuildMicromapsEXT` command provides the ability to initiate multiple micromaps builds, however there is no ordering or synchronization implied between any of the individual micromap builds.

| | |
|---|---|
| **NOTE** | This means that there **cannot** be any memory aliasing between any micromap memories or scratch memories being used by any of the builds. |

Accesses to the micromap scratch buffers as identified by the VkMicromapBuildInfoEXT::`scratchData` buffer device addresses **must** be synchronized with the `VK_PIPELINE_STAGE_2_MICROMAP_BUILD_BIT_EXT` pipeline stage and an access type of (`VK_ACCESS_2_MICROMAP_READ_BIT_EXT` | `VK_ACCESS_2_MICROMAP_WRITE_BIT_EXT`). Accesses to VkMicromapBuildInfoEXT::`dstMicromap` **must** be synchronized with the

VK_PIPELINE_STAGE_2_MICROMAP_BUILD_BIT_EXT pipeline stage and an access type of VK_ACCESS_2_MICROMAP_WRITE_BIT_EXT.

Accesses to other input buffers as identified by any used values of VkMicromapBuildInfoEXT::data or VkMicromapBuildInfoEXT::triangleArray **must** be synchronized with the VK_PIPELINE_STAGE_2_MICROMAP_BUILD_BIT_EXT pipeline stage and an access type of VK_ACCESS_SHADER_READ_BIT.

## Valid Usage

- VUID-vkCmdBuildMicromapsEXT-pInfos-07461
  For each pInfos[i], dstMicromap **must** have been created with a value of VkMicromapCreateInfoEXT::size greater than or equal to the memory size required by the build operation, as returned by vkGetMicromapBuildSizesEXT with pBuildInfo = pInfos[i]
- VUID-vkCmdBuildMicromapsEXT-mode-07462
  The mode member of each element of pInfos **must** be a valid VkBuildMicromapModeEXT value
- VUID-vkCmdBuildMicromapsEXT-dstMicromap-07463
  The dstMicromap member of any element of pInfos **must** be a valid VkMicromapEXT handle
- VUID-vkCmdBuildMicromapsEXT-pInfos-07464
  For each element of pInfos its type member **must** match the value of VkMicromapCreateInfoEXT::type when its dstMicromap was created
- VUID-vkCmdBuildMicromapsEXT-dstMicromap-07465
  The range of memory backing the dstMicromap member of any element of pInfos that is accessed by this command **must** not overlap the memory backing the dstMicromap member of any other element of pInfos, which is accessed by this command
- VUID-vkCmdBuildMicromapsEXT-dstMicromap-07466
  The range of memory backing the dstMicromap member of any element of pInfos that is accessed by this command **must** not overlap the memory backing the scratchData member of any element of pInfos (including the same element), which is accessed by this command
- VUID-vkCmdBuildMicromapsEXT-scratchData-07467
  The range of memory backing the scratchData member of any element of pInfos that is accessed by this command **must** not overlap the memory backing the scratchData member of any other element of pInfos, which is accessed by this command
- VUID-vkCmdBuildMicromapsEXT-pInfos-07508
  For each element of pInfos, the buffer used to create its dstMicromap member **must** be bound to device memory
- VUID-vkCmdBuildMicromapsEXT-pInfos-07509
  If pInfos[i].mode is VK_BUILD_MICROMAP_MODE_BUILD_EXT, all addresses between pInfos[i].scratchData.deviceAddress and pInfos[i].scratchData.deviceAddress + N - 1 **must** be in the buffer device address range of the same buffer, where N is given by the

buildScratchSize member of the VkMicromapBuildSizesInfoEXT structure returned from a call to vkGetMicromapBuildSizesEXT with an identical VkMicromapBuildInfoEXT structure and primitive count

- VUID-vkCmdBuildMicromapsEXT-data-07510
  The buffers from which the buffer device addresses for all of the data and triangleArray members of all pInfos[i] are queried **must** have been created with the VK_BUFFER_USAGE_MICROMAP_BUILD_INPUT_READ_ONLY_BIT_EXT usage flag
- VUID-vkCmdBuildMicromapsEXT-pInfos-07511
  For each element of pInfos[i] the buffer from which the buffer device address pInfos[i].scratchData.deviceAddress is queried **must** have been created with VK_BUFFER_USAGE_STORAGE_BUFFER_BIT usage flag
- VUID-vkCmdBuildMicromapsEXT-pInfos-07512
  For each element of pInfos, its scratchData.deviceAddress, data.deviceAddress, and triangleArray.deviceAddress members **must** be valid device addresses obtained from vkGetBufferDeviceAddressKHR
- VUID-vkCmdBuildMicromapsEXT-pInfos-07513
  For each element of pInfos, if scratchData.deviceAddress, data.deviceAddress, or triangleArray.deviceAddress is the address of a non-sparse buffer then it **must** be bound completely and contiguously to a single VkDeviceMemory object
- VUID-vkCmdBuildMicromapsEXT-pInfos-07514
  For each element of pInfos, its scratchData.deviceAddress member **must** be a multiple of VkPhysicalDeviceAccelerationStructurePropertiesKHR::minAccelerationStructureScratchOffsetAlignment
- VUID-vkCmdBuildMicromapsEXT-pInfos-07515
  For each element of pInfos, its triangleArray.deviceAddress and data.deviceAddress members **must** be a multiple of 256

## Valid Usage (Implicit)

- VUID-vkCmdBuildMicromapsEXT-commandBuffer-parameter
  commandBuffer **must** be a valid VkCommandBuffer handle
- VUID-vkCmdBuildMicromapsEXT-pInfos-parameter
  pInfos **must** be a valid pointer to an array of infoCount valid VkMicromapBuildInfoEXT structures
- VUID-vkCmdBuildMicromapsEXT-commandBuffer-recording
  commandBuffer **must** be in the recording state
- VUID-vkCmdBuildMicromapsEXT-commandBuffer-cmdpool
  The VkCommandPool that commandBuffer was allocated from **must** support compute operations
- VUID-vkCmdBuildMicromapsEXT-renderpass
  This command **must** only be called outside of a render pass instance
- VUID-vkCmdBuildMicromapsEXT-videocoding

This command **must** only be called outside of a video coding scope

- VUID-vkCmdBuildMicromapsEXT-infoCount-arraylength
  `infoCount` **must** be greater than `0`

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Outside | Outside | Compute | Action |

Formats which **can** be set in VkMicromapUsageEXT::`format` and VkMicromapTriangleEXT::`format` for micromap builds, are:

```
// Provided by VK_EXT_opacity_micromap
typedef enum VkOpacityMicromapFormatEXT {
    VK_OPACITY_MICROMAP_FORMAT_2_STATE_EXT = 1,
    VK_OPACITY_MICROMAP_FORMAT_4_STATE_EXT = 2,
} VkOpacityMicromapFormatEXT;
```

- `VK_OPACITY_MICROMAP_FORMAT_2_STATE_EXT` indicates that the given micromap format has one bit per subtriangle encoding either fully opaque or fully transparent.
- `VK_OPACITY_MICROMAP_FORMAT_4_STATE_EXT` indicates that the given micromap format has two bits per subtriangle encoding four modes which can be interpreted as described in ray traversal.

**NOTE** For compactness, these values are stored as 16-bit in some structures.

Formats which **can** be set in VkMicromapUsageEXT::`format` and VkMicromapTriangleEXT::`format` for micromap builds, are:

```
// Provided by VK_NV_displacement_micromap
typedef enum VkDisplacementMicromapFormatNV {
    VK_DISPLACEMENT_MICROMAP_FORMAT_64_TRIANGLES_64_BYTES_NV = 1,
    VK_DISPLACEMENT_MICROMAP_FORMAT_256_TRIANGLES_128_BYTES_NV = 2,
    VK_DISPLACEMENT_MICROMAP_FORMAT_1024_TRIANGLES_128_BYTES_NV = 3,
} VkDisplacementMicromapFormatNV;
```

- VK_DISPLACEMENT_MICROMAP_FORMAT_64_TRIANGLES_64_BYTES_NV indicates that the given micromap format encodes 64 micro-triangles worth of displacements in 64 bytes as described in Displacement Micromap Encoding.
- VK_DISPLACEMENT_MICROMAP_FORMAT_256_TRIANGLES_128_BYTES_NV indicates that the given micromap format encodes 256 micro-triangles worth of displacements in 128 bytes as described in Displacement Micromap Encoding.
- VK_DISPLACEMENT_MICROMAP_FORMAT_1024_TRIANGLES_128_BYTES_NV indicates that the given micromap format encodes 1024 micro-triangles worth of displacements in 128 bytes as described in Displacement Micromap Encoding.

**NOTE** For compactness, these values are stored as 16-bit in some structures.

The VkMicromapBuildInfoEXT structure is defined as:

```
// Provided by VK_EXT_opacity_micromap
typedef struct VkMicromapBuildInfoEXT {
    VkStructureType                     sType;
    const void*                         pNext;
    VkMicromapTypeEXT                   type;
    VkBuildMicromapFlagsEXT             flags;
    VkBuildMicromapModeEXT              mode;
    VkMicromapEXT                       dstMicromap;
    uint32_t                            usageCountsCount;
    const VkMicromapUsageEXT*           pUsageCounts;
    const VkMicromapUsageEXT* const*    ppUsageCounts;
    VkDeviceOrHostAddressConstKHR       data;
    VkDeviceOrHostAddressKHR            scratchData;
    VkDeviceOrHostAddressConstKHR       triangleArray;
    VkDeviceSize                        triangleArrayStride;
} VkMicromapBuildInfoEXT;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- type is a VkMicromapTypeEXT value specifying the type of micromap being built.
- flags is a bitmask of VkBuildMicromapFlagBitsEXT specifying additional parameters of the micromap.
- mode is a VkBuildMicromapModeEXT value specifying the type of operation to perform.
- dstMicromap is a pointer to the target micromap for the build.
- usageCountsCount specifies the number of usage counts structures that will be used to determine the size of this micromap.
- pUsageCounts is a pointer to an array of VkMicromapUsageEXT structures.
- ppUsageCounts is a pointer to an array of pointers to VkMicromapUsageEXT structures.
- data is the device or host address to memory which contains the data for the micromap.

- scratchData is the device or host address to memory that will be used as scratch memory for the build.
- triangleArray is the device or host address to memory containing the VkMicromapTriangleEXT data
- triangleArrayStride is the stride in bytes between each element of triangleArray

Only one of pUsageCounts or ppUsageCounts **can** be a valid pointer, the other **must** be NULL. The elements of the non-NULL array describe the total counts used to build each micromap. Each element contains a count which is the number of micromap triangles of that format and subdivisionLevel contained in the micromap. Multiple elements with the same format and subdivisionLevel are allowed and the total count for that format and subdivisionLevel is the sum of the count for each element.

Each micromap triangle refers to one element in triangleArray which contains the format and subdivisionLevel for that particular triangle as well as a dataOffset in bytes which is the location relative to data where that triangle's micromap data begins. The data at triangleArray is laid out as a 4 byte unsigned integer for the dataOffset followed by a 2 byte unsigned integer for the subdivision level then a 2 byte unsigned integer for the format. In practice, compilers compile VkMicromapTriangleEXT to match this pattern.

For opacity micromaps, the data at data is packed as either one bit per element for VK_OPACITY_MICROMAP_FORMAT_2_STATE_EXT or two bits per element for VK_OPACITY_MICROMAP_FORMAT_4_STATE_EXT and is packed from LSB to MSB in each byte. The data at each index in those bytes is interpreted as discussed in Ray Opacity Micromap.

For displacement micromaps, the data at data is interpreted as discussed in Displacement Micromap Encoding.

### Valid Usage

- VUID-VkMicromapBuildInfoEXT-pUsageCounts-07516
  Only one of pUsageCounts or ppUsageCounts **can** be a valid pointer, the other **must** be NULL
- VUID-VkMicromapBuildInfoEXT-type-07517
  If type is VK_MICROMAP_TYPE_OPACITY_MICROMAP_EXT the format member of VkMicromapUsageEXT **must** be a valid value from VkOpacityMicromapFormatEXT
- VUID-VkMicromapBuildInfoEXT-type-07518
  If type is VK_MICROMAP_TYPE_OPACITY_MICROMAP_EXT the format member of VkMicromapTriangleEXT **must** be a valid value from VkOpacityMicromapFormatEXT
- VUID-VkMicromapBuildInfoEXT-type-08704
  If type is VK_MICROMAP_TYPE_DISPLACEMENT_MICROMAP_NV the format member of VkMicromapUsageEXT **must** be a valid value from VkDisplacementMicromapFormatNV
- VUID-VkMicromapBuildInfoEXT-type-08705
  If type is VK_MICROMAP_TYPE_DISPLACEMENT_MICROMAP_NV the format member of VkMicromapTriangleEXT **must** be a valid value from VkDisplacementMicromapFormatNV

### Valid Usage (Implicit)

- VUID-VkMicromapBuildInfoEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_MICROMAP_BUILD_INFO_EXT`
- VUID-VkMicromapBuildInfoEXT-pNext-pNext
  `pNext` **must** be `NULL`
- VUID-VkMicromapBuildInfoEXT-type-parameter
  `type` **must** be a valid VkMicromapTypeEXT value
- VUID-VkMicromapBuildInfoEXT-flags-parameter
  `flags` **must** be a valid combination of VkBuildMicromapFlagBitsEXT values
- VUID-VkMicromapBuildInfoEXT-pUsageCounts-parameter
  If `usageCountsCount` is not `0`, and `pUsageCounts` is not `NULL`, `pUsageCounts` **must** be a valid pointer to an array of `usageCountsCount` VkMicromapUsageEXT structures
- VUID-VkMicromapBuildInfoEXT-ppUsageCounts-parameter
  If `usageCountsCount` is not `0`, and `ppUsageCounts` is not `NULL`, `ppUsageCounts` **must** be a valid pointer to an array of `usageCountsCount` valid pointers to VkMicromapUsageEXT structures

The `VkBuildMicromapModeEXT` enumeration is defined as:

```
// Provided by VK_EXT_opacity_micromap
typedef enum VkBuildMicromapModeEXT {
    VK_BUILD_MICROMAP_MODE_BUILD_EXT = 0,
} VkBuildMicromapModeEXT;
```

- `VK_BUILD_MICROMAP_MODE_BUILD_EXT` specifies that the destination micromap will be built using the specified data.

The `VkMicromapUsageEXT` structure is defined as:

```
// Provided by VK_EXT_opacity_micromap
typedef struct VkMicromapUsageEXT {
    uint32_t    count;
    uint32_t    subdivisionLevel;
    uint32_t    format;
} VkMicromapUsageEXT;
```

- `count` is the number of triangles in the usage format defined by the `subdivisionLevel` and `format` below in the micromap
- `subdivisionLevel` is the subdivision level of this usage format
- `format` is the format of this usage format

### Valid Usage

- VUID-VkMicromapUsageEXT-format-07519
  If the VkMicromapTypeEXT of the micromap is VK_MICROMAP_TYPE_OPACITY_MICROMAP_EXT then format **must** be VK_OPACITY_MICROMAP_FORMAT_2_STATE_EXT or VK_OPACITY_MICROMAP_FORMAT_4_STATE_EXT
- VUID-VkMicromapUsageEXT-format-07520
  If the VkMicromapTypeEXT of the micromap is VK_MICROMAP_TYPE_OPACITY_MICROMAP_EXT and format is VK_OPACITY_MICROMAP_FORMAT_2_STATE_EXT then subdivisionLevel **must** be less than or equal to VkPhysicalDeviceOpacityMicromapPropertiesEXT::maxOpacity2StateSubdivisionLevel
- VUID-VkMicromapUsageEXT-format-07521
  If the VkMicromapTypeEXT of the micromap is VK_MICROMAP_TYPE_OPACITY_MICROMAP_EXT and format is VK_OPACITY_MICROMAP_FORMAT_4_STATE_EXT then subdivisionLevel **must** be less than or equal to VkPhysicalDeviceOpacityMicromapPropertiesEXT::maxOpacity4StateSubdivisionLevel
- VUID-VkMicromapUsageEXT-format-08706
  If the VkMicromapTypeEXT of the micromap is VK_MICROMAP_TYPE_DISPLACEMENT_MICROMAP_NV then format **must** be VK_DISPLACEMENT_MICROMAP_FORMAT_64_TRIANGLES_64_BYTES_NV, VK_DISPLACEMENT_MICROMAP_FORMAT_256_TRIANGLES_128_BYTES_NV or VK_DISPLACEMENT_MICROMAP_FORMAT_1024_TRIANGLES_128_BYTES_NV
- VUID-VkMicromapUsageEXT-subdivisionLevel-08707
  If the VkMicromapTypeEXT of the micromap is VK_MICROMAP_TYPE_DISPLACEMENT_MICROMAP_NV then subdivisionLevel **must** be less than or equal to VkPhysicalDeviceDisplacementMicromapPropertiesNV::maxDisplacementMicromapSubdivisionLevel

The format is interpreted based on the type of the micromap using it.

The VkMicromapTriangleEXT structure is defined as:

```
// Provided by VK_EXT_opacity_micromap
typedef struct VkMicromapTriangleEXT {
    uint32_t    dataOffset;
    uint16_t    subdivisionLevel;
    uint16_t    format;
} VkMicromapTriangleEXT;
```

- dataOffset is the offset in bytes of the start of the data for this triangle. This is a byte aligned value.
- subdivisionLevel is the subdivision level of this triangle
- format is the format of this triangle

### Valid Usage

- VUID-VkMicromapTriangleEXT-format-07522
  If the VkMicromapTypeEXT of the micromap is VK_MICROMAP_TYPE_OPACITY_MICROMAP_EXT then format **must** be VK_OPACITY_MICROMAP_FORMAT_2_STATE_EXT or VK_OPACITY_MICROMAP_FORMAT_4_STATE_EXT
- VUID-VkMicromapTriangleEXT-format-07523
  If the VkMicromapTypeEXT of the micromap is VK_MICROMAP_TYPE_OPACITY_MICROMAP_EXT and format is VK_OPACITY_MICROMAP_FORMAT_2_STATE_EXT then subdivisionLevel **must** be less than or equal to VkPhysicalDeviceOpacityMicromapPropertiesEXT::maxOpacity2StateSubdivisionLevel
- VUID-VkMicromapTriangleEXT-format-07524
  If the VkMicromapTypeEXT of the micromap is VK_MICROMAP_TYPE_OPACITY_MICROMAP_EXT and format is VK_OPACITY_MICROMAP_FORMAT_4_STATE_EXT then subdivisionLevel **must** be less than or equal to VkPhysicalDeviceOpacityMicromapPropertiesEXT::maxOpacity4StateSubdivisionLevel
- VUID-VkMicromapTriangleEXT-format-08708
  If the VkMicromapTypeEXT of the micromap is VK_MICROMAP_TYPE_DISPLACEMENT_MICROMAP_NV then format **must** be VK_DISPLACEMENT_MICROMAP_FORMAT_64_TRIANGLES_64_BYTES_NV, VK_DISPLACEMENT_MICROMAP_FORMAT_256_TRIANGLES_128_BYTES_NV or VK_DISPLACEMENT_MICROMAP_FORMAT_1024_TRIANGLES_128_BYTES_NV
- VUID-VkMicromapTriangleEXT-subdivisionLevel-08709
  If the VkMicromapTypeEXT of the micromap is VK_MICROMAP_TYPE_DISPLACEMENT_MICROMAP_NV then subdivisionLevel **must** be less than or equal to VkPhysicalDeviceDisplacementMicromapPropertiesNV::maxDisplacementMicromapSubdivisionLevel

The format is interpreted based on the type of the micromap using it.

## 38.1.2. Copying Micromaps

An additional command exists for copying micromaps without updating their contents. Before copying, an application **must** query the size of the resulting micromap.

To query micromap size parameters call:

```
// Provided by VK_EXT_opacity_micromap
void vkCmdWriteMicromapsPropertiesEXT(
    VkCommandBuffer                                 commandBuffer,
    uint32_t                                        micromapCount,
    const VkMicromapEXT*                            pMicromaps,
    VkQueryType                                     queryType,
    VkQueryPool                                     queryPool,
    uint32_t                                        firstQuery);
```

- commandBuffer is the command buffer into which the command will be recorded.
- micromapCount is the count of micromaps for which to query the property.
- pMicromaps is a pointer to an array of existing previously built micromaps.
- queryType is a VkQueryType value specifying the type of queries managed by the pool.
- queryPool is the query pool that will manage the results of the query.
- firstQuery is the first query index within the query pool that will contain the micromapCount number of results.

Accesses to any of the micromaps listed in pMicromaps **must** be synchronized with the VK_PIPELINE_STAGE_2_MICROMAP_BUILD_BIT_EXT pipeline stage and an access type of VK_ACCESS_2_MICROMAP_READ_BIT_EXT.

- If queryType is VK_QUERY_TYPE_MICROMAP_SERIALIZATION_SIZE_EXT, then the value written out is the number of bytes required by a serialized micromap.
- If queryType is VK_QUERY_TYPE_MICROMAP_COMPACTED_SIZE_EXT, then the value written out is the number of bytes required by a compacted micromap.

## Valid Usage

- VUID-vkCmdWriteMicromapsPropertiesEXT-queryPool-07525
  queryPool **must** have been created with a queryType matching queryType
- VUID-vkCmdWriteMicromapsPropertiesEXT-queryPool-07526
  The queries identified by queryPool and firstQuery **must** be *unavailable*
- VUID-vkCmdWriteMicromapsPropertiesEXT-buffer-07527
  The buffer used to create each micromap in pMicrmaps **must** be bound to device memory
- VUID-vkCmdWriteMicromapsPropertiesEXT-query-07528
  The sum of query plus micromapCount **must** be less than or equal to the number of queries in queryPool
- VUID-vkCmdWriteMicromapsPropertiesEXT-pMicromaps-07501
  All micromaps in pMicromaps **must** have been constructed prior to the execution of this command
- VUID-vkCmdWriteMicromapsPropertiesEXT-pMicromaps-07502
  All micromaps in pMicromaps **must** have been constructed with VK_BUILD_MICROMAP_ALLOW_COMPACTION_BIT_EXT if queryType is VK_QUERY_TYPE_MICROMAP_COMPACTED_SIZE_EXT
- VUID-vkCmdWriteMicromapsPropertiesEXT-queryType-07503
  queryType **must** be VK_QUERY_TYPE_MICROMAP_COMPACTED_SIZE_EXT or VK_QUERY_TYPE_MICROMAP_SERIALIZATION_SIZE_EXT

## Valid Usage (Implicit)

- VUID-vkCmdWriteMicromapsPropertiesEXT-commandBuffer-parameter

commandBuffer **must** be a valid VkCommandBuffer handle

- VUID-vkCmdWriteMicromapsPropertiesEXT-pMicromaps-parameter
  pMicromaps **must** be a valid pointer to an array of micromapCount valid VkMicromapEXT handles
- VUID-vkCmdWriteMicromapsPropertiesEXT-queryType-parameter
  queryType **must** be a valid VkQueryType value
- VUID-vkCmdWriteMicromapsPropertiesEXT-queryPool-parameter
  queryPool **must** be a valid VkQueryPool handle
- VUID-vkCmdWriteMicromapsPropertiesEXT-commandBuffer-recording
  commandBuffer **must** be in the recording state
- VUID-vkCmdWriteMicromapsPropertiesEXT-commandBuffer-cmdpool
  The VkCommandPool that commandBuffer was allocated from **must** support compute operations
- VUID-vkCmdWriteMicromapsPropertiesEXT-renderpass
  This command **must** only be called outside of a render pass instance
- VUID-vkCmdWriteMicromapsPropertiesEXT-videocoding
  This command **must** only be called outside of a video coding scope
- VUID-vkCmdWriteMicromapsPropertiesEXT-micromapCount-arraylength
  micromapCount **must** be greater than 0
- VUID-vkCmdWriteMicromapsPropertiesEXT-commonparent
  Each of commandBuffer, queryPool, and the elements of pMicromaps **must** have been created, allocated, or retrieved from the same VkDevice

## Host Synchronization

- Host access to commandBuffer **must** be externally synchronized
- Host access to the VkCommandPool that commandBuffer was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Outside | Outside | Compute | Action |

To copy a micromap call:

```
// Provided by VK_EXT_opacity_micromap
void vkCmdCopyMicromapEXT(
    VkCommandBuffer                             commandBuffer,
    const VkCopyMicromapInfoEXT*                pInfo);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `pInfo` is a pointer to a VkCopyMicromapInfoEXT structure defining the copy operation.

This command copies the `pInfo->src` micromap to the `pInfo->dst` micromap in the manner specified by `pInfo->mode`.

Accesses to `pInfo->src` and `pInfo->dst` **must** be synchronized with the `VK_PIPELINE_STAGE_2_MICROMAP_BUILD_BIT_EXT` pipeline stage and an access type of `VK_ACCESS_2_MICROMAP_READ_BIT_EXT` or `VK_ACCESS_2_MICROMAP_WRITE_BIT_EXT` as appropriate.

## Valid Usage

- VUID-vkCmdCopyMicromapEXT-buffer-07529
  The `buffer` used to create `pInfo->src` **must** be bound to device memory
- VUID-vkCmdCopyMicromapEXT-buffer-07530
  The `buffer` used to create `pInfo->dst` **must** be bound to device memory

## Valid Usage (Implicit)

- VUID-vkCmdCopyMicromapEXT-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdCopyMicromapEXT-pInfo-parameter
  `pInfo` **must** be a valid pointer to a valid VkCopyMicromapInfoEXT structure
- VUID-vkCmdCopyMicromapEXT-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdCopyMicromapEXT-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support compute operations
- VUID-vkCmdCopyMicromapEXT-renderpass
  This command **must** only be called outside of a render pass instance
- VUID-vkCmdCopyMicromapEXT-videocoding
  This command **must** only be called outside of a video coding scope

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally

synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Outside | Outside | Compute | Action |

The `VkCopyMicromapInfoEXT` structure is defined as:

```
// Provided by VK_EXT_opacity_micromap
typedef struct VkCopyMicromapInfoEXT {
    VkStructureType          sType;
    const void*              pNext;
    VkMicromapEXT            src;
    VkMicromapEXT            dst;
    VkCopyMicromapModeEXT    mode;
} VkCopyMicromapInfoEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `src` is the source micromap for the copy.
- `dst` is the target micromap for the copy.
- `mode` is a VkCopyMicromapModeEXT value specifying additional operations to perform during the copy.

## Valid Usage

- VUID-VkCopyMicromapInfoEXT-mode-07531
  `mode` **must** be `VK_COPY_MICROMAP_MODE_COMPACT_EXT` or `VK_COPY_MICROMAP_MODE_CLONE_EXT`
- VUID-VkCopyMicromapInfoEXT-src-07532
  The source acceleration structure `src` **must** have been constructed prior to the execution of this command
- VUID-VkCopyMicromapInfoEXT-mode-07533
  If `mode` is `VK_COPY_MICROMAP_MODE_COMPACT_EXT`, `src` **must** have been constructed with `VK_BUILD_MICROMAP_ALLOW_COMPACTION_BIT_EXT` in the build
- VUID-VkCopyMicromapInfoEXT-buffer-07534
  The `buffer` used to create `src` **must** be bound to device memory
- VUID-VkCopyMicromapInfoEXT-buffer-07535
  The `buffer` used to create `dst` **must** be bound to device memory

### Valid Usage (Implicit)

- VUID-VkCopyMicromapInfoEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_COPY_MICROMAP_INFO_EXT`
- VUID-VkCopyMicromapInfoEXT-pNext-pNext
  `pNext` **must** be `NULL`
- VUID-VkCopyMicromapInfoEXT-src-parameter
  `src` **must** be a valid VkMicromapEXT handle
- VUID-VkCopyMicromapInfoEXT-dst-parameter
  `dst` **must** be a valid VkMicromapEXT handle
- VUID-VkCopyMicromapInfoEXT-mode-parameter
  `mode` **must** be a valid VkCopyMicromapModeEXT value
- VUID-VkCopyMicromapInfoEXT-commonparent
  Both of `dst`, and `src` **must** have been created, allocated, or retrieved from the same VkDevice

Possible values of `mode` specifying additional operations to perform during the copy, are:

```
// Provided by VK_EXT_opacity_micromap
typedef enum VkCopyMicromapModeEXT {
    VK_COPY_MICROMAP_MODE_CLONE_EXT = 0,
    VK_COPY_MICROMAP_MODE_SERIALIZE_EXT = 1,
    VK_COPY_MICROMAP_MODE_DESERIALIZE_EXT = 2,
    VK_COPY_MICROMAP_MODE_COMPACT_EXT = 3,
} VkCopyMicromapModeEXT;
```

- `VK_COPY_MICROMAP_MODE_CLONE_EXT` creates a direct copy of the micromap specified in `src` into the one specified by `dst`. The `dst` micromap **must** have been created with the same parameters as `src`.
- `VK_COPY_MICROMAP_MODE_SERIALIZE_EXT` serializes the micromap to a semi-opaque format which can be reloaded on a compatible implementation.
- `VK_COPY_MICROMAP_MODE_DESERIALIZE_EXT` deserializes the semi-opaque serialization format in the buffer to the micromap.
- `VK_COPY_MICROMAP_MODE_COMPACT_EXT` creates a more compact version of a micromap `src` into `dst`. The micromap `dst` **must** have been created with a size at least as large as that returned by vkCmdWriteMicromapsPropertiesEXT after the build of the micromap specified by `src`.

To copy a micromap to device memory call:

```
// Provided by VK_EXT_opacity_micromap
void vkCmdCopyMicromapToMemoryEXT(
    VkCommandBuffer                             commandBuffer,
    const VkCopyMicromapToMemoryInfoEXT*        pInfo);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `pInfo` is an a pointer to a VkCopyMicromapToMemoryInfoEXT structure defining the copy operation.

Accesses to `pInfo->src` **must** be synchronized with the `VK_PIPELINE_STAGE_2_MICROMAP_BUILD_BIT_EXT` pipeline stage and an access type of `VK_ACCESS_2_MICROMAP_READ_BIT_EXT`. Accesses to the buffer indicated by `pInfo->dst.deviceAddress` **must** be synchronized with the `VK_PIPELINE_STAGE_2_MICROMAP_BUILD_BIT_EXT` pipeline stage and an access type of `VK_ACCESS_TRANSFER_WRITE_BIT`.

This command produces the same results as vkCopyMicromapToMemoryEXT, but writes its result to a device address, and is executed on the device rather than the host. The output **may** not necessarily be bit-for-bit identical, but it can be equally used by either vkCmdCopyMemoryToMicromapEXT or vkCopyMemoryToMicromapEXT.

The defined header structure for the serialized data consists of:

- `VK_UUID_SIZE` bytes of data matching `VkPhysicalDeviceIDProperties::driverUUID`
- `VK_UUID_SIZE` bytes of data identifying the compatibility for comparison using vkGetDeviceMicromapCompatibilityEXT The serialized data is written to the buffer (or read from the buffer) according to the host endianness.

## Valid Usage

- VUID-vkCmdCopyMicromapToMemoryEXT-pInfo-07536
  `pInfo->dst.deviceAddress` **must** be a valid device address for a buffer bound to device memory
- VUID-vkCmdCopyMicromapToMemoryEXT-pInfo-07537
  `pInfo->dst.deviceAddress` **must** be aligned to `256` bytes
- VUID-vkCmdCopyMicromapToMemoryEXT-pInfo-07538
  If the buffer pointed to by `pInfo->dst.deviceAddress` is non-sparse then it **must** be bound completely and contiguously to a single VkDeviceMemory object
- VUID-vkCmdCopyMicromapToMemoryEXT-buffer-07539
  The `buffer` used to create `pInfo->src` **must** be bound to device memory

## Valid Usage (Implicit)

- VUID-vkCmdCopyMicromapToMemoryEXT-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle

- VUID-vkCmdCopyMicromapToMemoryEXT-pInfo-parameter
  `pInfo` **must** be a valid pointer to a valid VkCopyMicromapToMemoryInfoEXT structure
- VUID-vkCmdCopyMicromapToMemoryEXT-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdCopyMicromapToMemoryEXT-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support compute operations
- VUID-vkCmdCopyMicromapToMemoryEXT-renderpass
  This command **must** only be called outside of a render pass instance
- VUID-vkCmdCopyMicromapToMemoryEXT-videocoding
  This command **must** only be called outside of a video coding scope

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Outside | Outside | Compute | Action |

```
// Provided by VK_EXT_opacity_micromap
typedef struct VkCopyMicromapToMemoryInfoEXT {
    VkStructureType             sType;
    const void*                 pNext;
    VkMicromapEXT               src;
    VkDeviceOrHostAddressKHR    dst;
    VkCopyMicromapModeEXT       mode;
} VkCopyMicromapToMemoryInfoEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `src` is the source micromap for the copy
- `dst` is the device or host address to memory which is the target for the copy
- `mode` is a VkCopyMicromapModeEXT value specifying additional operations to perform during the copy.

### Valid Usage

- VUID-VkCopyMicromapToMemoryInfoEXT-src-07540
  The source micromap `src` **must** have been constructed prior to the execution of this command
- VUID-VkCopyMicromapToMemoryInfoEXT-dst-07541
  The memory pointed to by `dst` **must** be at least as large as the serialization size of `src`, as reported by vkWriteMicromapsPropertiesEXT or vkCmdWriteMicromapsPropertiesEXT with a query type of `VK_QUERY_TYPE_MICROMAP_SERIALIZATION_SIZE_EXT`
- VUID-VkCopyMicromapToMemoryInfoEXT-mode-07542
  `mode` **must** be `VK_COPY_MICROMAP_MODE_SERIALIZE_EXT`

### Valid Usage (Implicit)

- VUID-VkCopyMicromapToMemoryInfoEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_COPY_MICROMAP_TO_MEMORY_INFO_EXT`
- VUID-VkCopyMicromapToMemoryInfoEXT-pNext-pNext
  `pNext` **must** be `NULL`
- VUID-VkCopyMicromapToMemoryInfoEXT-src-parameter
  `src` **must** be a valid VkMicromapEXT handle
- VUID-VkCopyMicromapToMemoryInfoEXT-mode-parameter
  `mode` **must** be a valid VkCopyMicromapModeEXT value

To copy device memory to a micromap call:

```
// Provided by VK_EXT_opacity_micromap
void vkCmdCopyMemoryToMicromapEXT(
    VkCommandBuffer                             commandBuffer,
    const VkCopyMemoryToMicromapInfoEXT*        pInfo);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `pInfo` is a pointer to a VkCopyMicromapToMemoryInfoEXT structure defining the copy operation.

Accesses to `pInfo->dst` **must** be synchronized with the `VK_PIPELINE_STAGE_2_MICROMAP_BUILD_BIT_EXT` pipeline stage and an access type of `VK_ACCESS_2_MICROMAP_READ_BIT_EXT`. Accesses to the buffer indicated by `pInfo->src.deviceAddress` **must** be synchronized with the `VK_PIPELINE_STAGE_2_MICROMAP_BUILD_BIT_EXT` pipeline stage and an access type of `VK_ACCESS_TRANSFER_READ_BIT`.

This command can accept micromaps produced by either vkCmdCopyMicromapToMemoryEXT or vkCopyMicromapToMemoryEXT.

## Valid Usage

- VUID-vkCmdCopyMemoryToMicromapEXT-pInfo-07543
  `pInfo->src.deviceAddress` **must** be a valid device address for a buffer bound to device memory
- VUID-vkCmdCopyMemoryToMicromapEXT-pInfo-07544
  `pInfo->src.deviceAddress` **must** be aligned to `256` bytes
- VUID-vkCmdCopyMemoryToMicromapEXT-pInfo-07545
  If the buffer pointed to by `pInfo->src.deviceAddress` is non-sparse then it **must** be bound completely and contiguously to a single VkDeviceMemory object
- VUID-vkCmdCopyMemoryToMicromapEXT-buffer-07546
  The `buffer` used to create `pInfo->dst` **must** be bound to device memory

## Valid Usage (Implicit)

- VUID-vkCmdCopyMemoryToMicromapEXT-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdCopyMemoryToMicromapEXT-pInfo-parameter
  `pInfo` **must** be a valid pointer to a valid VkCopyMemoryToMicromapInfoEXT structure
- VUID-vkCmdCopyMemoryToMicromapEXT-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdCopyMemoryToMicromapEXT-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support compute operations
- VUID-vkCmdCopyMemoryToMicromapEXT-renderpass
  This command **must** only be called outside of a render pass instance
- VUID-vkCmdCopyMemoryToMicromapEXT-videocoding
  This command **must** only be called outside of a video coding scope

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Outside | Outside | Compute | Action |

The `VkCopyMemoryToMicromapInfoEXT` structure is defined as:

```
// Provided by VK_EXT_opacity_micromap
typedef struct VkCopyMemoryToMicromapInfoEXT {
    VkStructureType                 sType;
    const void*                     pNext;
    VkDeviceOrHostAddressConstKHR   src;
    VkMicromapEXT                   dst;
    VkCopyMicromapModeEXT           mode;
} VkCopyMemoryToMicromapInfoEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `src` is the device or host address to memory containing the source data for the copy.
- `dst` is the target micromap for the copy.
- `mode` is a VkCopyMicromapModeEXT value specifying additional operations to perform during the copy.

## Valid Usage

- VUID-VkCopyMemoryToMicromapInfoEXT-src-07547
  The source memory pointed to by `src` **must** contain data previously serialized using vkCmdCopyMicromapToMemoryEXT
- VUID-VkCopyMemoryToMicromapInfoEXT-mode-07548
  `mode` **must** be `VK_COPY_MICROMAP_MODE_DESERIALIZE_EXT`
- VUID-VkCopyMemoryToMicromapInfoEXT-src-07549
  The data in `src` **must** have a format compatible with the destination physical device as returned by vkGetDeviceMicromapCompatibilityEXT
- VUID-VkCopyMemoryToMicromapInfoEXT-dst-07550
  `dst` **must** have been created with a `size` greater than or equal to that used to serialize the data in `src`

### Valid Usage (Implicit)

- VUID-VkCopyMemoryToMicromapInfoEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_COPY_MEMORY_TO_MICROMAP_INFO_EXT`
- VUID-VkCopyMemoryToMicromapInfoEXT-pNext-pNext
  `pNext` **must** be `NULL`
- VUID-VkCopyMemoryToMicromapInfoEXT-dst-parameter
  `dst` **must** be a valid VkMicromapEXT handle
- VUID-VkCopyMemoryToMicromapInfoEXT-mode-parameter
  `mode` **must** be a valid VkCopyMicromapModeEXT value

To check if a serialized micromap is compatible with the current device call:

```
// Provided by VK_EXT_opacity_micromap
void vkGetDeviceMicromapCompatibilityEXT(
    VkDevice                                    device,
    const VkMicromapVersionInfoEXT*             pVersionInfo,
    VkAccelerationStructureCompatibilityKHR*    pCompatibility);
```

- `device` is the device to check the version against.
- `pVersionInfo` is a pointer to a VkMicromapVersionInfoEXT structure specifying version information to check against the device.
- `pCompatibility` is a pointer to a VkAccelerationStructureCompatibilityKHR value in which compatibility information is returned.

### Valid Usage

- VUID-vkGetDeviceMicromapCompatibilityEXT-micromap-07551
  The `micromap` feature **must** be enabled

### Valid Usage (Implicit)

- VUID-vkGetDeviceMicromapCompatibilityEXT-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkGetDeviceMicromapCompatibilityEXT-pVersionInfo-parameter
  `pVersionInfo` **must** be a valid pointer to a valid VkMicromapVersionInfoEXT structure
- VUID-vkGetDeviceMicromapCompatibilityEXT-pCompatibility-parameter
  `pCompatibility` **must** be a valid pointer to a VkAccelerationStructureCompatibilityKHR value

The `VkMicromapVersionInfoEXT` structure is defined as:

```
// Provided by VK_EXT_opacity_micromap
typedef struct VkMicromapVersionInfoEXT {
    VkStructureType    sType;
    const void*        pNext;
    const uint8_t*     pVersionData;
} VkMicromapVersionInfoEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `pVersionData` is a pointer to the version header of a micromap as defined in vkCmdCopyMicromapToMemoryEXT

**NOTE** `pVersionData` is a *pointer* to an array of 2×`VK_UUID_SIZE` `uint8_t` values instead of two `VK_UUID_SIZE` arrays as the expected use case for this member is to be pointed at the header of a previously serialized micromap (via vkCmdCopyMicromapToMemoryEXT or vkCopyMicromapToMemoryEXT) that is loaded in memory. Using arrays would necessitate extra memory copies of the UUIDs.

**Valid Usage (Implicit)**

- VUID-VkMicromapVersionInfoEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_MICROMAP_VERSION_INFO_EXT`
- VUID-VkMicromapVersionInfoEXT-pNext-pNext
  `pNext` **must** be `NULL`
- VUID-VkMicromapVersionInfoEXT-pVersionData-parameter
  `pVersionData` **must** be a valid pointer to an array of $2 \times VK_UUID_SIZE$ `uint8_t` values

## 38.2. Host Micromap Operations

Implementations are also required to provide host implementations of the micromap operations if the `micromapHostCommands` feature is enabled:

- vkBuildMicromapsEXT corresponding to vkCmdBuildMicromapsEXT
- vkCopyMicromapEXT corresponding to vkCmdCopyMicromapEXT
- vkCopyMicromapToMemoryEXT corresponding to vkCmdCopyMicromapToMemoryEXT
- vkCopyMemoryToMicromapEXT corresponding to vkCmdCopyMemoryToMicromapEXT
- vkWriteMicromapsPropertiesEXT corresponding to vkCmdWriteMicromapsPropertiesEXT

These commands are functionally equivalent to their device counterparts, except that they are executed on the host timeline, rather than being enqueued into command buffers.

All micromaps used by the host commands **must** be bound to host-visible memory, and all input

data for micromap builds **must** be referenced using host addresses instead of device addresses. Applications are not required to map micromap memory when using the host commands.

**NOTE** The vkBuildMicromapsEXT and vkCmdBuildMicromapsEXT **may** use different algorithms, and thus are not required to produce identical structures.

Apart from these details, the host and device operations are interchangeable.

**NOTE** For efficient execution, micromaps manipulated using these commands should always be bound to host cached memory, as the implementation may need to repeatedly read and write this memory during the execution of the command.

To build micromaps on the host, call:

```
// Provided by VK_EXT_opacity_micromap
VkResult vkBuildMicromapsEXT(
    VkDevice                                    device,
    VkDeferredOperationKHR                      deferredOperation,
    uint32_t                                    infoCount,
    const VkMicromapBuildInfoEXT*               pInfos);
```

- `device` is the `VkDevice` for which the micromaps are being built.
- `deferredOperation` is an optional VkDeferredOperationKHR to request deferral for this command.
- `infoCount` is the number of micromaps to build. It specifies the number of the `pInfos` that **must** be provided.
- `pInfos` is a pointer to an array of `infoCount` VkMicromapBuildInfoEXT structures defining the geometry used to build each micromap.

This command fulfills the same task as vkCmdBuildMicromapsEXT but is executed by the host.

The `vkBuildMicromapsEXT` command provides the ability to initiate multiple micromaps builds, however there is no ordering or synchronization implied between any of the individual micromap builds.

**NOTE** This means that there **cannot** be any memory aliasing between any micromap memories or scratch memories being used by any of the builds.

### Valid Usage

- VUID-vkBuildMicromapsEXT-pInfos-07461
  For each `pInfos`[i], `dstMicromap` **must** have been created with a value of VkMicromapCreateInfoEXT::`size` greater than or equal to the memory size required by the build operation, as returned by vkGetMicromapBuildSizesEXT with `pBuildInfo` = `pInfos`[i]

- VUID-vkBuildMicromapsEXT-mode-07462
  The `mode` member of each element of `pInfos` **must** be a valid VkBuildMicromapModeEXT value
- VUID-vkBuildMicromapsEXT-dstMicromap-07463
  The `dstMicromap` member of any element of `pInfos` **must** be a valid VkMicromapEXT handle
- VUID-vkBuildMicromapsEXT-pInfos-07464
  For each element of `pInfos` its `type` member **must** match the value of VkMicromapCreateInfoEXT::`type` when its `dstMicromap` was created
- VUID-vkBuildMicromapsEXT-dstMicromap-07465
  The range of memory backing the `dstMicromap` member of any element of `pInfos` that is accessed by this command **must** not overlap the memory backing the `dstMicromap` member of any other element of `pInfos`, which is accessed by this command
- VUID-vkBuildMicromapsEXT-dstMicromap-07466
  The range of memory backing the `dstMicromap` member of any element of `pInfos` that is accessed by this command **must** not overlap the memory backing the `scratchData` member of any element of `pInfos` (including the same element), which is accessed by this command
- VUID-vkBuildMicromapsEXT-scratchData-07467
  The range of memory backing the `scratchData` member of any element of `pInfos` that is accessed by this command **must** not overlap the memory backing the `scratchData` member of any other element of `pInfos`, which is accessed by this command
- VUID-vkBuildMicromapsEXT-pInfos-07552
  For each element of `pInfos`, the `buffer` used to create its `dstMicromap` member **must** be bound to host-visible device memory
- VUID-vkBuildMicromapsEXT-pInfos-07553
  For each element of `pInfos`, all referenced addresses of `pInfos`[i].`data.hostAddress` **must** be valid host memory
- VUID-vkBuildMicromapsEXT-pInfos-07554
  For each element of `pInfos`, all referenced addresses of `pInfos`[i].`triangleArray.hostAddress` **must** be valid host memory
- VUID-vkBuildMicromapsEXT-micromapHostCommands-07555
  The `VkPhysicalDeviceOpacityMicromapFeaturesEXT::micromapHostCommands` feature **must** be enabled
- VUID-vkBuildMicromapsEXT-pInfos-07556
  If `pInfos`[i].`mode` is `VK_BUILD_MICROMAP_MODE_BUILD_EXT`, all addresses between `pInfos`[i].`scratchData.hostAddress` and `pInfos`[i].`scratchData.hostAddress` + N - 1 **must** be valid host memory, where N is given by the `buildScratchSize` member of the VkMicromapBuildSizesInfoEXT structure returned from a call to vkGetMicromapBuildSizesEXT with an identical VkMicromapBuildInfoEXT structure and primitive count
- VUID-vkBuildMicromapsEXT-pInfos-07557
  For each element of `pInfos`, the `buffer` used to create its `dstMicromap` member **must** be

bound to memory that was not allocated with multiple instances

### Valid Usage (Implicit)

- VUID-vkBuildMicromapsEXT-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkBuildMicromapsEXT-deferredOperation-parameter
  If `deferredOperation` is not VK_NULL_HANDLE, `deferredOperation` **must** be a valid VkDeferredOperationKHR handle
- VUID-vkBuildMicromapsEXT-pInfos-parameter
  `pInfos` **must** be a valid pointer to an array of `infoCount` valid VkMicromapBuildInfoEXT structures
- VUID-vkBuildMicromapsEXT-infoCount-arraylength
  `infoCount` **must** be greater than `0`
- VUID-vkBuildMicromapsEXT-deferredOperation-parent
  If `deferredOperation` is a valid handle, it **must** have been created, allocated, or retrieved from `device`

### Return Codes

**Success**

- `VK_SUCCESS`
- `VK_OPERATION_DEFERRED_KHR`
- `VK_OPERATION_NOT_DEFERRED_KHR`

**Failure**

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`

To copy or compact a micromap on the host, call:

```
// Provided by VK_EXT_opacity_micromap
VkResult vkCopyMicromapEXT(
    VkDevice                                    device,
    VkDeferredOperationKHR                      deferredOperation,
    const VkCopyMicromapInfoEXT*                pInfo);
```

- `device` is the device which owns the micromaps.
- `deferredOperation` is an optional VkDeferredOperationKHR to request deferral for this command.
- `pInfo` is a pointer to a VkCopyMicromapInfoEXT structure defining the copy operation.

This command fulfills the same task as vkCmdCopyMicromapEXT but is executed by the host.

## Valid Usage

- VUID-vkCopyMicromapEXT-deferredOperation-03678
  Any previous deferred operation that was associated with `deferredOperation` **must** be complete
- VUID-vkCopyMicromapEXT-buffer-07558
  The `buffer` used to create `pInfo->src` **must** be bound to host-visible device memory
- VUID-vkCopyMicromapEXT-buffer-07559
  The `buffer` used to create `pInfo->dst` **must** be bound to host-visible device memory
- VUID-vkCopyMicromapEXT-micromapHostCommands-07560
  The `VkPhysicalDeviceOpacityMicromapFeaturesEXT::micromapHostCommands` feature **must** be enabled
- VUID-vkCopyMicromapEXT-buffer-07561
  The `buffer` used to create `pInfo->src` **must** be bound to memory that was not allocated with multiple instances
- VUID-vkCopyMicromapEXT-buffer-07562
  The `buffer` used to create `pInfo->dst` **must** be bound to memory that was not allocated with multiple instances

## Valid Usage (Implicit)

- VUID-vkCopyMicromapEXT-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkCopyMicromapEXT-deferredOperation-parameter
  If `deferredOperation` is not VK_NULL_HANDLE, `deferredOperation` **must** be a valid VkDeferredOperationKHR handle
- VUID-vkCopyMicromapEXT-pInfo-parameter
  `pInfo` **must** be a valid pointer to a valid VkCopyMicromapInfoEXT structure
- VUID-vkCopyMicromapEXT-deferredOperation-parent
  If `deferredOperation` is a valid handle, it **must** have been created, allocated, or retrieved from `device`

## Return Codes

**Success**

- `VK_SUCCESS`
- `VK_OPERATION_DEFERRED_KHR`
- `VK_OPERATION_NOT_DEFERRED_KHR`

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY

To copy host accessible memory to a micromap, call:

```
// Provided by VK_EXT_opacity_micromap
VkResult vkCopyMemoryToMicromapEXT(
    VkDevice                                    device,
    VkDeferredOperationKHR                      deferredOperation,
    const VkCopyMemoryToMicromapInfoEXT*        pInfo);
```

- device is the device which owns pInfo->dst.
- deferredOperation is an optional VkDeferredOperationKHR to request deferral for this command.
- pInfo is a pointer to a VkCopyMemoryToMicromapInfoEXT structure defining the copy operation.

This command fulfills the same task as vkCmdCopyMemoryToMicromapEXT but is executed by the host.

This command can accept micromaps produced by either vkCmdCopyMicromapToMemoryEXT or vkCopyMicromapToMemoryEXT.

## Valid Usage

- VUID-vkCopyMemoryToMicromapEXT-deferredOperation-03678
  Any previous deferred operation that was associated with deferredOperation **must** be complete
- VUID-vkCopyMemoryToMicromapEXT-pInfo-07563
  pInfo->src.hostAddress **must** be a valid host pointer
- VUID-vkCopyMemoryToMicromapEXT-pInfo-07564
  pInfo->src.hostAddress **must** be aligned to 16 bytes
- VUID-vkCopyMemoryToMicromapEXT-buffer-07565
  The buffer used to create pInfo->dst **must** be bound to host-visible device memory
- VUID-vkCopyMemoryToMicromapEXT-micromapHostCommands-07566
  The VkPhysicalDeviceOpacityMicromapFeaturesEXT::micromapHostCommands feature **must** be enabled
- VUID-vkCopyMemoryToMicromapEXT-buffer-07567
  The buffer used to create pInfo->dst **must** be bound to memory that was not allocated with multiple instances

### Valid Usage (Implicit)

- VUID-vkCopyMemoryToMicromapEXT-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkCopyMemoryToMicromapEXT-deferredOperation-parameter
  If `deferredOperation` is not VK_NULL_HANDLE, `deferredOperation` **must** be a valid VkDeferredOperationKHR handle
- VUID-vkCopyMemoryToMicromapEXT-pInfo-parameter
  `pInfo` **must** be a valid pointer to a valid VkCopyMemoryToMicromapInfoEXT structure
- VUID-vkCopyMemoryToMicromapEXT-deferredOperation-parent
  If `deferredOperation` is a valid handle, it **must** have been created, allocated, or retrieved from `device`

### Return Codes

**Success**

- `VK_SUCCESS`
- `VK_OPERATION_DEFERRED_KHR`
- `VK_OPERATION_NOT_DEFERRED_KHR`

**Failure**

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`

To copy a micromap to host accessible memory, call:

```
// Provided by VK_EXT_opacity_micromap
VkResult vkCopyMicromapToMemoryEXT(
    VkDevice                                    device,
    VkDeferredOperationKHR                      deferredOperation,
    const VkCopyMicromapToMemoryInfoEXT*        pInfo);
```

- `device` is the device which owns `pInfo->src`.
- `deferredOperation` is an optional VkDeferredOperationKHR to request deferral for this command.
- `pInfo` is a pointer to a VkCopyMicromapToMemoryInfoEXT structure defining the copy operation.

This command fulfills the same task as vkCmdCopyMicromapToMemoryEXT but is executed by the host.

This command produces the same results as vkCmdCopyMicromapToMemoryEXT, but writes its

result directly to a host pointer, and is executed on the host rather than the device. The output **may** not necessarily be bit-for-bit identical, but it can be equally used by either vkCmdCopyMemoryToMicromapEXT or vkCopyMemoryToMicromapEXT.

## Valid Usage

- VUID-vkCopyMicromapToMemoryEXT-deferredOperation-03678
  Any previous deferred operation that was associated with `deferredOperation` **must** be complete
- VUID-vkCopyMicromapToMemoryEXT-buffer-07568
  The `buffer` used to create `pInfo->src` **must** be bound to host-visible device memory
- VUID-vkCopyMicromapToMemoryEXT-pInfo-07569
  `pInfo->dst.hostAddress` **must** be a valid host pointer
- VUID-vkCopyMicromapToMemoryEXT-pInfo-07570
  `pInfo->dst.hostAddress` **must** be aligned to 16 bytes
- VUID-vkCopyMicromapToMemoryEXT-micromapHostCommands-07571
  The `VkPhysicalDeviceOpacityMicromapFeaturesEXT::micromapHostCommands` feature **must** be enabled
- VUID-vkCopyMicromapToMemoryEXT-buffer-07572
  The `buffer` used to create `pInfo->src` **must** be bound to memory that was not allocated with multiple instances

## Valid Usage (Implicit)

- VUID-vkCopyMicromapToMemoryEXT-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkCopyMicromapToMemoryEXT-deferredOperation-parameter
  If `deferredOperation` is not VK_NULL_HANDLE, `deferredOperation` **must** be a valid VkDeferredOperationKHR handle
- VUID-vkCopyMicromapToMemoryEXT-pInfo-parameter
  `pInfo` **must** be a valid pointer to a valid VkCopyMicromapToMemoryInfoEXT structure
- VUID-vkCopyMicromapToMemoryEXT-deferredOperation-parent
  If `deferredOperation` is a valid handle, it **must** have been created, allocated, or retrieved from `device`

## Return Codes

**Success**

- `VK_SUCCESS`
- `VK_OPERATION_DEFERRED_KHR`
- `VK_OPERATION_NOT_DEFERRED_KHR`

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY

To query micromap size parameters on the host, call:

```
// Provided by VK_EXT_opacity_micromap
VkResult vkWriteMicromapsPropertiesEXT(
    VkDevice                                    device,
    uint32_t                                    micromapCount,
    const VkMicromapEXT*                        pMicromaps,
    VkQueryType                                 queryType,
    size_t                                      dataSize,
    void*                                       pData,
    size_t                                      stride);
```

- device is the device which owns the micromaps in pMicromaps.
- micromapCount is the count of micromaps for which to query the property.
- pMicromaps is a pointer to an array of existing previously built micromaps.
- queryType is a VkQueryType value specifying the property to be queried.
- dataSize is the size in bytes of the buffer pointed to by pData.
- pData is a pointer to an application-allocated buffer where the results will be written.
- stride is the stride in bytes between results for individual queries within pData.

This command fulfills the same task as vkCmdWriteMicromapsPropertiesEXT but is executed by the host.

## Valid Usage

- VUID-vkWriteMicromapsPropertiesEXT-pMicromaps-07501
  All micromaps in pMicromaps **must** have been constructed prior to the execution of this command
- VUID-vkWriteMicromapsPropertiesEXT-pMicromaps-07502
  All micromaps in pMicromaps **must** have been constructed with VK_BUILD_MICROMAP_ALLOW_COMPACTION_BIT_EXT if queryType is VK_QUERY_TYPE_MICROMAP_COMPACTED_SIZE_EXT
- VUID-vkWriteMicromapsPropertiesEXT-queryType-07503
  queryType **must** be VK_QUERY_TYPE_MICROMAP_COMPACTED_SIZE_EXT or VK_QUERY_TYPE_MICROMAP_SERIALIZATION_SIZE_EXT
- VUID-vkWriteMicromapsPropertiesEXT-queryType-10071
  If queryType is VK_QUERY_TYPE_MICROMAP_SERIALIZATION_SIZE_EXT or VK_QUERY_TYPE_MICROMAP_COMPACTED_SIZE_EXT then stride **must** be a multiple of the size of VkDeviceSize

- VUID-vkWriteMicromapsPropertiesEXT-queryType-10072
  If queryType is VK_QUERY_TYPE_MICROMAP_SERIALIZATION_SIZE_EXT or VK_QUERY_TYPE_MICROMAP_COMPACTED_SIZE_EXT then pData **must** point to a VkDeviceSize
- VUID-vkWriteMicromapsPropertiesEXT-dataSize-07576
  dataSize **must** be greater than or equal to micromapCount*stride
- VUID-vkWriteMicromapsPropertiesEXT-buffer-07577
  The buffer used to create each micromap in pMicromaps **must** be bound to host-visible device memory
- VUID-vkWriteMicromapsPropertiesEXT-micromapHostCommands-07578
  The VkPhysicalDeviceOpacityMicromapFeaturesEXT::micromapHostCommands feature **must** be enabled
- VUID-vkWriteMicromapsPropertiesEXT-buffer-07579
  The buffer used to create each micromap in pMicromaps **must** be bound to memory that was not allocated with multiple instances

## Valid Usage (Implicit)

- VUID-vkWriteMicromapsPropertiesEXT-device-parameter
  device **must** be a valid VkDevice handle
- VUID-vkWriteMicromapsPropertiesEXT-pMicromaps-parameter
  pMicromaps **must** be a valid pointer to an array of micromapCount valid VkMicromapEXT handles
- VUID-vkWriteMicromapsPropertiesEXT-queryType-parameter
  queryType **must** be a valid VkQueryType value
- VUID-vkWriteMicromapsPropertiesEXT-pData-parameter
  pData **must** be a valid pointer to an array of dataSize bytes
- VUID-vkWriteMicromapsPropertiesEXT-micromapCount-arraylength
  micromapCount **must** be greater than 0
- VUID-vkWriteMicromapsPropertiesEXT-dataSize-arraylength
  dataSize **must** be greater than 0
- VUID-vkWriteMicromapsPropertiesEXT-pMicromaps-parent
  Each element of pMicromaps **must** have been created, allocated, or retrieved from device

## Return Codes

### Success

- VK_SUCCESS

### Failure

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY

### 38.2.1. Displacement Micromap

A displacement micromap in an acceleration structure includes information in the VkAccelerationStructureTrianglesDisplacementMicromapNV to define a base triangle and displacement directions then uses displacement information encoded in the micromap to apply to those values to generate the final position.

#### Displacement Base Triangle

If `displacementBiasAndScaleBuffer` is provided the bias and scale are fetched from that buffer. If `displacementBiasAndScaleBuffer` is zero the bias and scale are assumed to be 0.0 and 1.0, respectively.

Given an input position from the geometry, the base position and displacement vector used by the displacement are computed by:

basePosition = inputPosition + displacementVector × bias

baseDisplacementVector = displacementVector × scale

The parameters of each micro-vertex are derived from a combination of the base triangle parameters extracted from the bottom-level acceleration structure, the barycentrics of that micro-vertex, and the displacement value fetched from the displacement micromap corresponding to that micro-vertex.

microVertexBasePosition = lerp(basePositions, microVertexBarycentrics)

microVertexDisplacementVector = lerp(displacementVectors, microVertexBarycentrics)

microVertexDisplacedPosition = microVertexBasePosition + microVertexDisplacementVector × micromapDisplacementValue

#### Displacement Micromap Encoding

Displacement amounts are stored in displacement blocks, each covering a triangular region of microvertices. Depending on the subdivision level and encoding format, one or more displacement blocks combine to store all displacement values for a given displacement micromap.

Displacement blocks are organized along a space filling curve within a displacement micromap if more than one block is required, then micro-vertices are organized along the same space filling curve within a displacement micromap.

The space-filling curve is purely hierarchical with recursive splitting, similar to that for opacity micromaps but operating on vertices instead of triangles. To maintain that the hierarchical ordering is contiguous while keeping continuous winding, some triangles are flipped and wound differently.

The `VK_DISPLACEMENT_MICROMAP_FORMAT_64_TRIANGLES_64_BYTES_NV` format is an uncompressed, packed format which covers 64 microtriangles (subdivision level 3) in a block. The block contains 45 displacement values encoded as 11 bit unorm values and stored tightly packed in the vertex order described above, occupying 495 bits. This is followed by 15 unused bits then 2 reserved bits which

**must** be 0. If this block is used to store displacement for a subdivision level below 3 the later unused values are ignored.

| Section | Field | Entries | Bits per entry | Starting bit offset |
|---|---|---|---|---|
| Displacement amounts | Vertex 0 - 44 | 45 | 11 | 0 |
| Unused | | 1 | 15 | 495 |
| Reserved | Must be 0 | 1 | 2 | 510 |

The `VK_DISPLACEMENT_MICROMAP_FORMAT_256_TRIANGLES_128_BYTES_NV` and `VK_DISPLACEMENT_MICROMAP_FORMAT_1024_TRIANGLES_128_BYTES_NV` formats store displacements in a compressed form to save space. Both formats use the same compression algorithm, differing in the number of bits used in the different fields.

The compression algorithm works by starting with fully specified anchor vertices, then for each level, predicting the value for the displacement and encoding the correction for that value, using fewer bits for each level of subdivision.

When adding a vertex in the recursive subdivision process between two adjacent displacement values, the predicted value is given by the rounded average of the two adjacent values as integers:

prediction = (A + B + 1) / 2

The decoded value after applying the correction is given by:

decoded = prediction + ( SignExtend(correction) << shift )

where correction is given by the corrections field for a given level and micro vertex and shift is given by the shifts field indexed from the level then by 4 values, selected from interior or the 3 edges in vertex order in that order.

The bit encoding for `VK_DISPLACEMENT_MICROMAP_FORMAT_256_TRIANGLES_128_BYTES_NV`

| Section | Field | Entries | Bits per entry | Starting bit offset |
|---|---|---|---|---|
| Anchors | Vertex 0 - 2 | 3 | 11 | 0 |
| Corrections | Level 1 | 3 | 11 | 33 |
| | Level 2 | 9 | 11 | 66 |
| | Level 3 | 30 | 10 | 165 |
| | Level 4 | 108 | 5 | 465 |
| Unused | | 1 | 1 | 1005 |
| Shifts | Level 4 | 4 | 3 | 1006 |
| | Level 3 | 4 | 1 | 1018 |
| Reserved | Must be 0 | 1 | 2 | 1022 |

The bit encoding for `VK_DISPLACEMENT_MICROMAP_FORMAT_1024_TRIANGLES_128_BYTES_NV`

| Section | Field | Entries | Bits per entry | Starting bit offset |
|---|---|---|---|---|
| Anchors | Vertex 0 - 2 | 3 | 11 | 0 |
| Corrections | Level 1 | 3 | 11 | 33 |
| | Level 2 | 9 | 8 | 66 |
| | Level 3 | 30 | 4 | 138 |
| | Level 4 | 108 | 2 | 258 |
| | Level 5 | 408 | 1 | 474 |
| Unused | | 1 | 88 | 882 |
| Shifts | Level 5 | 4 | 4 | 970 |
| | Level 4 | 4 | 4 | 986 |
| | Level 3 | 4 | 3 | 1002 |
| | Level 2 | 4 | 2 | 1014 |
| Reserved | Must be 0 | 1 | 2 | 1022 |

# Chapter 39. Ray Traversal

The ray traversal process identifies and handles intersections between a ray and geometries in an acceleration structure.

Ray traversal cannot be started by a Vulkan API command directly - a shader **must** execute `OpRayQueryProceedKHR` or a pipeline trace ray instruction . When the `rayTracingPipeline` feature is enabled, `OpTraceRayKHR` **can** be used for ray tracing in a ray tracing pipeline. When the `rayQuery` feature is enabled, `OpRayQueryProceedKHR` **can** be used in any shader stage.

## 39.1. Ray Intersection Candidate Determination

Once tracing begins, rays are first tested against instances in a top-level acceleration structure. A ray that intersects an instance will be transformed into the space of the instance to continue traversal within that instance; therefore the transform matrix stored in the instance **must** be invertible.

In case multiple instances are intersected by a ray, the ray transformation into the space of the instance is invariant under the order in which these instances are encountered in the top-level acceleration structure.

| | |
|---|---|
| **NOTE** | Applying multiple forward and reverse transforms to a ray to transition from one instance to another could result in accumulated errors. Thus an implementation should behave as if the ray is transformed from the origin for each instance independently. |

Next, rays are tested against geometries in a bottom-level acceleration structure to determine if a hit occurred between them, initially based only on their geometric properties (i.e. their vertices). The implementation performs similar operations to that of rasterization, but with the effective viewport determined by the parameters of the ray, and the geometry transformed into a space determined by that viewport.

The vertices of each primitive are transformed from acceleration structure space $_{as}$ to ray space $_{r}$ according to the ray origin and direction as follows:

$$\begin{pmatrix} x_r \\ y_r \\ z_r \end{pmatrix} = \begin{pmatrix} a_x^2(1-c)+c & a_x a_y(1-c)-sa_z & a_x a_z(1-c)+sa_y \\ a_x a_y(1-c)+sa_z & a_y^2(1-c)+c & a_y a_z(1-c)-sa_x \\ a_x a_z(1-c)-sa_y & a_y a_z(1-c)+sa_x & a_z^2(1-c)+c \end{pmatrix} \begin{pmatrix} x_{as}-o_x \\ y_{as}-o_y \\ z_{as}-o_z \end{pmatrix}$$

$\mathbf{a}$ is the axis of rotation from the unnormalized ray direction vector $\mathbf{d}$ to the axis vector $\mathbf{k}$:

$$\mathbf{a} = \begin{cases} \dfrac{\mathbf{d} \times \mathbf{k}}{\|\mathbf{d} \times \mathbf{k}\|} & \text{if } \|\mathbf{d} \times \mathbf{k}\| \neq 0 \\ \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} & \text{if } \|\mathbf{d} \times \mathbf{k}\| = 0 \end{cases}$$

$s$ and $c$ are the sine and cosine of the angle of rotation about $\mathbf{a}$ from $\mathbf{d}$ to $\mathbf{k}$:

$$c = \frac{\mathbf{d} \cdot \mathbf{k}}{\|\mathbf{d}\|}$$
$$s = \sqrt{1 - c^2}$$

$\mathbf{k}$ is the unit vector:

$$\mathbf{k} = \begin{pmatrix} 0 \\ 0 \\ -1 \end{pmatrix}$$

$\mathbf{o}$ and $\mathbf{d}$ are the ray origin and unnormalized direction, respectively; the vector described by $x_{as}$, $y_{as}$, and $z_{as}$ is any position in acceleration structure space; and the vector described by $x_r$, $y_r$, and $z_r$ is the same position in ray space.

An *intersection candidate* is a unique point of intersection between a ray and a geometric primitive. For any primitive that has within its bounds a position $\mathbf{xyz_{as}}$ such that

$$\begin{aligned} x_r &= 0 \\ y_r &= 0 \\ t_{min} < -\frac{z_r}{\|\mathbf{d}\|} &< t_{max} \qquad \text{if the primitive is a triangle,} \\ t_{min} \leq -\frac{z_r}{\|\mathbf{d}\|} &\leq t_{max} \qquad \text{otherwise} \end{aligned}$$

(where $t = -\frac{z_r}{\|\mathbf{d}\|}$), an intersection candidate exists.

Triangle primitive bounds consist of all points on the plane formed by the three vertices and within the bounds of the edges between the vertices, subject to the watertightness constraints below. AABB primitive bounds consist of all points within an implementation-defined bound which includes the specified box.

| | |
|---|---|
| **NOTE** | The bounds of the AABB including all points internal to the bound implies that a ray started within the AABB will hit that AABB. |

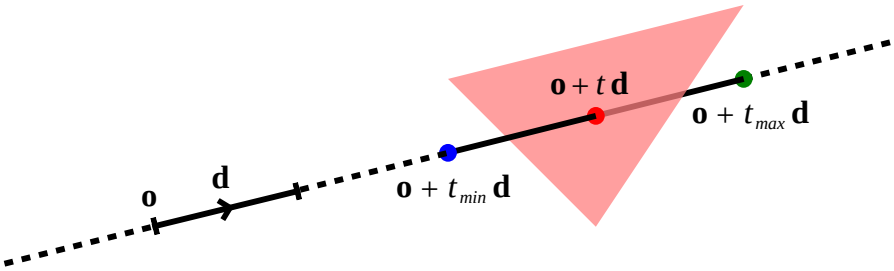



*Figure 28. Ray intersection candidate*

The determination of this condition is performed in an implementation specific manner, and **may** be performed with floating-point operations. Due to the complexity and number of operations involved, inaccuracies are expected, particularly as the scale of values involved begins to diverge.

Implementations **should** take efforts to maintain as much precision as possible.

**NOTE**

One very common case is when geometries are close to each other at some distance from the origin in acceleration structure space, where an effect similar to "z-fighting" is likely to be observed. Applications can mitigate this by ensuring their detailed geometries remain close to the origin.

Another likely case is when the origin of a ray is set to a position on a previously intersected surface, and its $t_{min}$ is zero or near zero; an intersection may be detected on the emitting surface. This case can usually be mitigated by offsetting $t_{min}$ slightly.

For a motion primitive or a motion instance, the positions for intersection are evaluated at the time specified in the `time` parameter to `OpTraceRayMotionNV` by interpolating between the two endpoints as specified for the given motion type. If a motion acceleration structure is traced with `OpTraceRayKHR`, it behaves as a `OpTraceRayMotionNV` with `time` of 0.0.

In the case of AABB geometries, implementations **may** increase their size in an acceleration structure in order to mitigate precision issues. This **may** result in false positive intersections being reported to the application.

For triangle intersection candidates, the b and c barycentric coordinates on the triangle where the above condition is met are made available to future shading. If the ray was traced with a pipeline trace ray instruction, these values are available as a vector of 2 32-bit floating-point values in the `HitAttributeKHR` storage class.

Once an intersection candidate is determined, it proceeds through the following operations, in order:

1. Ray Intersection Culling
2. Ray Intersection Confirmation
3. Ray Closest Hit Determination
4. Ray Result Determination

The sections below describe the exact details of these tests. There is no ordering guarantee between operations performed on different intersection candidates.

### 39.1.1. Watertightness

For a set of triangles with identical transforms, within a single instance:

- Any set of two or more triangles where all triangles have one vertex with an identical position value, that vertex is a *shared vertex*.
- Any set of two triangles with two shared vertices that were specified in the same winding order in each triangle have a *shared edge* defined by those vertices.

A *closed fan* is a set of three or more triangles where:

- All triangles in the set have the same shared vertex as one of their vertices.

- All edges that include the above vertex are shared edges.
- All above shared edges are shared by exactly two triangles from the set.
- No two triangles in the set intersect, except at shared edges.
- Every triangle in the set is joined to every other triangle in the set by a series of the above shared edges.

Implementations **should** not double-hit or miss when a ray intersects a shared edge, or a shared vertex of a closed fan.

## 39.2. Ray Intersection Culling

Candidate intersections go through several phases of culling before confirmation as an actual hit. There is no particular ordering dependency between the different culling operations.

### 39.2.1. Ray Primitive Culling

If the `rayTraversalPrimitiveCulling` or `rayQuery` features are enabled, the `SkipTrianglesKHR` and `SkipAABBsKHR` ray flags **can** be specified when tracing a ray. `SkipTrianglesKHR` and `SkipAABBsKHR` are mutually exclusive. `SkipTrianglesKHR` is also mutually exclusive with `CullBackFacingTrianglesKHR` and `CullFrontFacingTrianglesKHR`.

If `SkipTrianglesKHR` was included in the `Ray Flags` operand of the ray trace instruction, and the intersection is with a triangle primitive, the intersection is dropped, and no further processing of this intersection occurs. If `VK_PIPELINE_CREATE_RAY_TRACING_SKIP_TRIANGLES_BIT_KHR` was included in the pipeline, traversal with pipeline trace ray instructions will all behave as if `SkipTrianglesKHR` was included in their `Ray Flags` operand.

If `SkipAABBsKHR` was included in the `Ray Flags` operand of the ray trace instruction, and the intersection is with an AABB primitive, the intersection is dropped, and no further processing of this intersection occurs. If `VK_PIPELINE_CREATE_RAY_TRACING_SKIP_AABBS_BIT_KHR` was included in the pipeline, traversal with pipeline trace ray instructions will all behave as if `SkipAABBsKHR` was included in their `Ray Flags` operand.

### 39.2.2. Ray Mask Culling

Instances **can** be made invisible to particular rays based on the value of VkAccelerationStructureInstanceKHR::`mask` used to add that instance to a top-level acceleration structure, and the `Cull Mask` parameter used to trace the ray.

For the instance which is intersected, if `mask` & `Cull Mask` == 0, the intersection is dropped, and no further processing occurs.

### 39.2.3. Ray Face Culling

As in polygon rasterization, one of the stages of ray traversal is to determine if a triangle primitive is back- or front-facing, and primitives **can** be culled based on that facing.

If the intersection candidate is with an AABB primitive, this operation is skipped.

*Determination*

When a ray intersects a triangle primitive, the order that vertices are specified for the polygon affects whether the ray intersects the front or back face. Front or back facing is determined in the same way as they are for rasterization, based on the sign of the polygon's area but using the ray space coordinates instead of framebuffer coordinates. One way to compute this area is:

$$a = -\frac{1}{2}\sum_{i=0}^{n-1} x_r^i y_r^{i \oplus 1} - x_r^{i \oplus 1} y_r^i$$

where $x_r^i$ and $y_r^i$ are the x and y ray space coordinates of the ith vertex of the n-vertex polygon (vertices are numbered starting at zero for the purposes of this computation) and i □ 1 is (i + 1) mod n.

By default, if a is negative then the intersection is with the front face of the triangle, otherwise it is with the back face. If `VK_GEOMETRY_INSTANCE_TRIANGLE_FLIP_FACING_BIT_KHR` is included in VkAccelerationStructureInstanceKHR::`flags` for the instance containing the intersected triangle, this determination is reversed. Additionally, if a is 0, the intersection candidate is treated as not intersecting with any face, irrespective of the sign.

| | |
|---|---|
| **NOTE** | In a left-handed coordinate system, an intersection will be with the front face of a triangle if the vertices of the triangle, as defined in index order, appear from the ray's perspective in a clockwise rotation order. `VK_GEOMETRY_INSTANCE_TRIANGLE_FLIP_FACING_BIT_KHR` was previously annotated as `VK_GEOMETRY_INSTANCE_TRIANGLE_FRONT_COUNTERCLOCKWISE_BIT_KHR` because of this. |

If the ray was traced with a pipeline trace ray instruction, the `HitKindKHR` built-in is set to `HitKindFrontFacingTriangleKHR` if the intersection is with front-facing geometry, and `HitKindBackFacingTriangleKHR` if the intersection is with back-facing geometry, for shader stages considering this intersection.

If the ray was traced with `OpRayQueryProceedKHR`, `OpRayQueryGetIntersectionFrontFaceKHR` will return true for intersection candidates with front faces, or false for back faces.

*Culling*

If `CullBackFacingTrianglesKHR` was included in the `Ray Flags` parameter of the ray trace instruction, and the intersection is determined as with the back face of a triangle primitive, the intersection is dropped, and no further processing of this intersection occurs.

If `CullFrontFacingTrianglesKHR` was included in the `Ray Flags` parameter of the ray trace instruction, and the intersection is determined as with the front face of a triangle primitive, the intersection is dropped, and no further processing of this intersection occurs.

This culling is disabled if `VK_GEOMETRY_INSTANCE_TRIANGLE_FACING_CULL_DISABLE_BIT_KHR` was included in VkAccelerationStructureInstanceKHR::`flags` for the instance which the intersected geometry belongs to.

Intersection candidates that have not intersected with any face (a == 0) are unconditionally culled, irrespective of ray flags and geometry instance flags.

The `CullBackFacingTrianglesKHR` and `CullFrontFacingTrianglesKHR` `Ray Flags` are mutually exclusive.

### 39.2.4. Ray Opacity Culling

Each geometry in the acceleration structure **may** be considered either opaque or not. Opaque geometries continue through traversal as normal, whereas non-opaque geometries need to be either confirmed or discarded by shader code. Intersection candidates **can** also be culled based on their opacity.

*Determination*

Each individual intersection candidate is initially determined as opaque if `VK_GEOMETRY_OPAQUE_BIT_KHR` was included in the VkAccelerationStructureGeometryKHR::`flags` when the geometry it intersected with was built, otherwise it is considered non-opaque.

If the geometry includes an opacity micromap, the opacity of the intersection at this point is instead derived as described in Ray Opacity Micromap.

If the intersection candidate was generated by an intersection shader, the intersection is initially considered to have opacity matching the AABB candidate that it was generated from.

However, this opacity can be overridden when it is built into an instance. Setting `VK_GEOMETRY_INSTANCE_FORCE_OPAQUE_BIT_KHR` in VkAccelerationStructureInstanceKHR::`flags` will force all geometries in the instance to be considered opaque. Similarly, setting `VK_GEOMETRY_INSTANCE_FORCE_NO_OPAQUE_BIT_KHR` will force all geometries in the instance to be considered non-opaque.

This can again be overridden by including `OpaqueKHR` or `NoOpaqueKHR` in the `Ray Flags` parameter when tracing a ray. `OpaqueKHR` forces all geometries to behave as if they are opaque, regardless of their build parameters. Similarly, `NoOpaqueKHR` forces all geometries to behave as if they are non-opaque.

If the ray was traced with `OpRayQueryProceedKHR`, to determine the opacity of AABB intersection candidates, `OpRayQueryGetIntersectionCandidateAABBOpaqueKHR` **can** be used. This instruction will return `true` for opaque intersection candidates, and `false` for non-opaque intersection candidates.

*Culling*

If `CullOpaqueKHR` is included in the `Ray Flags` parameter when tracing a ray, an intersection with a geometry that is considered opaque is dropped, and no further processing occurs.

If `CullNoOpaqueKHR` is included in the `Ray Flags` parameter when tracing a ray, an intersection with a geometry that is considered non-opaque is dropped, and no further processing occurs.

The `OpaqueKHR`, `NoOpaqueKHR`, `CullOpaqueKHR`, and `CullNoOpaqueKHR` `Ray Flags` are mutually exclusive.

### 39.2.5. Ray Opacity Micromap

A `VK_GEOMETRY_TYPE_TRIANGLES_KHR` geometry in the acceleration structure **may** have an opacity micromap associated with it to give finer-grained opacity information.

If the intersection candidate is with a geometry with an associated opacity micromap and

VK_GEOMETRY_INSTANCE_DISABLE_OPACITY_MICROMAPS_EXT is not set in its instance then the micromap is used to determine geometry opacity instead of the VK_GEOMETRY_OPAQUE_BIT_KHR flag in the geometry.

The opacity information in the micromap object is accessed using the candidate intersection u and v coordinates. The integer u and v are computed from ⌊u⌋ + ⌊v⌋, clamping ⌊u⌋ as needed to keep the sum less than or equal to 1 << subdivisionlevel. These values are mapped into a linear index with a space filling curve which is defined recursively by traversing into the sub-triangle nearest vertex 0, then the middle triangle with ordering flipped, then nearest vertex 1 then nearest vertex 2.

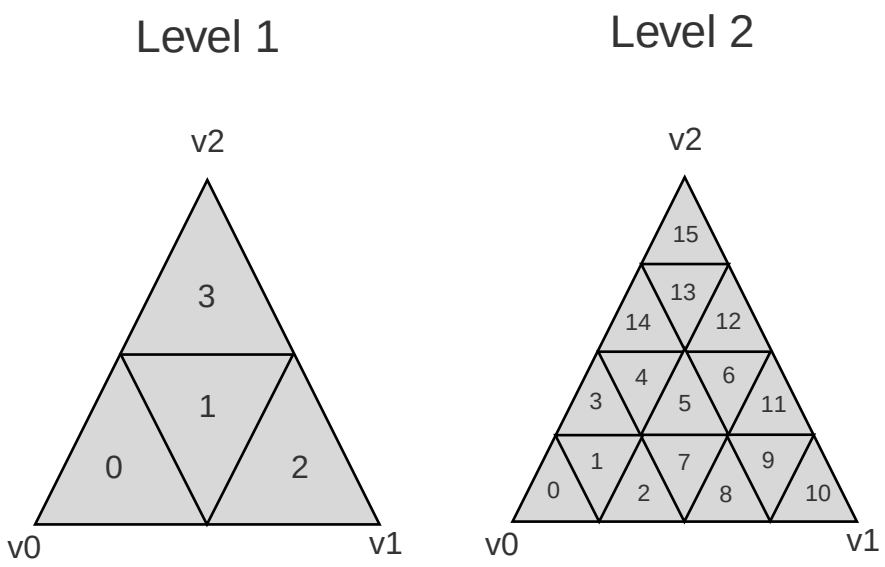



*Figure 29. Example ordering for micromap data*

**NOTE**

This encoding is spatially coherent, purely hierarchical, and allows a bit-parallel conversion between barycentric address and index values.

See the appendix for reference code implementing this mapping.

The result of the opacity micromap lookup and operations is to treat the intersection as opaque, non-opaque, or ignored. The interpretation of the values depends on VK_GEOMETRY_INSTANCE_FORCE_OPACITY_MICROMAP_2_STATE_EXT in the instance of the candidate intersection or ForceOpacityMicromap2StateEXT ray flags on the ray. If either is set, the opacity micromap information is interpreted in 2 state override mode. If the result of the micromap lookup is to treat the intersection candidate as ignored, no further processing of that candidate is done.

If the associated opacity micromap has format VK_OPACITY_MICROMAP_FORMAT_2_STATE_EXT, each element of the micromap is represented by a single bit at the index derived above.

If the associated opacity micromap has format VK_OPACITY_MICROMAP_FORMAT_4_STATE_EXT, each element is represented by a two bit value at the index derived above.

| 4 State value | 2 State value | Special index value | 2 State override | Result |
|---|---|---|---|---|
| 0 | 0 | VK_OPACITY_MICROMAP_SPECIAL_INDEX_FULLY_TRANSPARENT_EXT | Y | Ignored |
| 0 | 0 | VK_OPACITY_MICROMAP_SPECIAL_INDEX_FULLY_TRANSPARENT_EXT | N | Ignored |
| 1 | 1 | VK_OPACITY_MICROMAP_SPECIAL_INDEX_FULLY_OPAQUE_EXT | Y | Opaque |
| 1 | 1 | VK_OPACITY_MICROMAP_SPECIAL_INDEX_FULLY_OPAQUE_EXT | N | Opaque |
| 2 | | VK_OPACITY_MICROMAP_SPECIAL_INDEX_FULLY_UNKNOWN_TRANSPARENT_EXT | Y | Ignored |
| 2 | | VK_OPACITY_MICROMAP_SPECIAL_INDEX_FULLY_UNKNOWN_TRANSPARENT_EXT | N | Non-opaque |
| 3 | | VK_OPACITY_MICROMAP_SPECIAL_INDEX_FULLY_UNKNOWN_OPAQUE_EXT | Y | Opaque |
| 3 | | VK_OPACITY_MICROMAP_SPECIAL_INDEX_FULLY_UNKNOWN_OPAQUE_EXT | N | Non-opaque |

# 39.3. Ray Intersection Confirmation

Depending on the opacity of intersected geometry and whether it is a triangle or an AABB, candidate intersections are further processed to determine the eventual hit result. Candidates generated from AABB intersections run through the same confirmation process as triangle hits.

## 39.3.1. AABB Intersection Candidates

For an intersection candidate with an AABB geometry generated by Ray Intersection Candidate Determination, shader code is executed to determine whether any hits should be reported to the traversal infrastructure; no further processing of this intersection candidate occurs. The occurrence of an AABB intersection candidate does not guarantee the ray intersects the primitive bounds. To avoid propagating false intersections the application **should** verify the intersection candidate before reporting any hits.

If the ray was traced with a pipeline trace ray instruction, an intersection shader is invoked from the Shader Binding Table according to the specified indexing for the intersected geometry. If this

shader calls OpReportIntersectionKHR, a new intersection candidate is generated as described below. If the intersection shader is VK_SHADER_UNUSED_KHR (which is only allowed for a zero shader group) then no further processing of the intersection candidate occurs.

Each new candidate generated as a result of this processing is a generated intersection candidate that intersects the AABB geometry, with a t value equal to the Hit parameter of the OpReportIntersectionKHR instruction. The new generated candidate is then independently run through Ray Intersection Confirmation as a generated intersection.

If the ray was traced with OpRayQueryProceedKHR, control is returned to the shader which executed OpRayQueryProceedKHR, returning true. The resulting ray query has a candidate intersection type of RayQueryCandidateIntersectionAABBKHR. OpRayQueryGenerateIntersectionKHR **can** be called to commit a new intersection candidate with committed intersection type of RayQueryCommittedIntersectionGeneratedKHR. Further ray query processing **can** be continued by executing OpRayQueryProceedKHR with the same ray query, or intersection **can** be terminated with OpRayQueryTerminateKHR. Unlike rays traced with a pipeline trace ray instruction, candidates generated in this way skip generated intersection candidate confirmation; applications **should** make this determination before generating the intersection.

This operation **may** be executed multiple times for the same intersection candidate.

### 39.3.2. Triangle and Generated Intersection Candidates

For triangle and generated intersection candidates, additional shader code **may** be executed based on the intersection's opacity.

If the intersection is opaque, the candidate is immediately confirmed as a valid hit and passes to the next stage of processing.

For non-opaque intersection candidates, shader code is executed to determine whether a hit occurred or not.

If the ray was traced with a pipeline trace ray instruction, an any-hit shader is invoked from the Shader Binding Table according to the specified indexing. If this shader calls OpIgnoreIntersectionKHR, the candidate is dropped and no further processing of the candidate occurs. If the any-hit shader identified is VK_SHADER_UNUSED_KHR, the candidate is immediately confirmed as a valid hit and passes to the next stage of processing.

If the ray was traced with OpRayQueryProceedKHR, control is returned to the shader which executed OpRayQueryProceedKHR, returning true. As only triangle candidates participate in this operation with ray queries, the resulting candidate intersection type is always RayQueryCandidateIntersectionTriangleKHR. OpRayQueryConfirmIntersectionKHR **can** be called on the ray query to confirm the candidate as a hit with committed intersection type of RayQueryCommittedIntersectionTriangleKHR. Further ray query processing **can** be continued by executing OpRayQueryProceedKHR with the same ray query, or intersection **can** be terminated with OpRayQueryTerminateKHR. If OpRayQueryConfirmIntersectionKHR has not been executed, the candidate is dropped and no further processing of the candidate occurs.

This operation **may** be executed multiple times for the same intersection candidate unless VK_GEOMETRY_NO_DUPLICATE_ANY_HIT_INVOCATION_BIT_KHR was specified for the intersected geometry.

## 39.4. Ray Closest Hit Determination

Unless the ray was traced with the `TerminateOnFirstHitKHR` ray flag, the implementation **must** track the closest confirmed hit until all geometries have been tested and either confirmed or dropped.

After an intersection candidate is confirmed, its t value is compared to $t_{max}$ to determine which intersection is closer, where t is the parametric distance along the ray at which the intersection occurred.

- If $t < t_{max}$, $t_{max}$ is set to t and the candidate is set as the current closest hit.
- If $t > t_{max}$, the candidate is dropped and no further processing of that candidate occurs.
- If $t = t_{max}$, the candidate **may** be set as the current closest hit or dropped.

If `TerminateOnFirstHitKHR` was included in the `Ray Flags` used to trace the ray, once the first hit is confirmed, the ray trace is terminated.

## 39.5. Ray Result Determination

Once all candidates have finished processing the prior stages, or if the ray is forcibly terminated, the final result of the ray trace is determined.

If a closest hit result was identified by Ray Closest Hit Determination, a closest hit has occurred, otherwise the final result is a miss.

For rays traced with pipeline trace ray instructions which **can** invoke a closest hit shader, if a closest hit result was identified, a closest hit shader is invoked from the Shader Binding Table according to the specified indexing for the intersected geometry. Control returns to the shader that executed the pipeline trace ray instruction once this shader returns. This shader is skipped if either the ray flags included `SkipClosestHitShaderKHR`, or if the closest hit shader identified is `VK_SHADER_UNUSED_KHR`.

For rays traced with a pipeline trace ray instruction where no hit result was identified, the miss shader identified by the `Miss Index` parameter of the instruction is invoked. Control returns to the shader that executed the pipeline trace ray instruction once this shader returns. This shader is skipped if the miss shader identified is `VK_SHADER_UNUSED_KHR`.

If the ray was traced with `OpRayQueryProceedKHR`, control is returned to the shader which executed `OpRayQueryProceedKHR`, returning `false`. If a closest hit was identified by Ray Closest Hit Determination, the ray query will now have a committed intersection type of `RayQueryCommittedIntersectionGeneratedKHR` or `RayQueryCommittedIntersectionTriangleKHR`. If no closest hit was identified, the committed intersection type will be `RayQueryCommittedIntersectionNoneKHR`.

No further processing of a ray query occurs after this result is determined.

# Chapter 40. Ray Tracing

Ray tracing uses a separate rendering pipeline from both the graphics and compute pipelines (see Ray Tracing Pipeline).

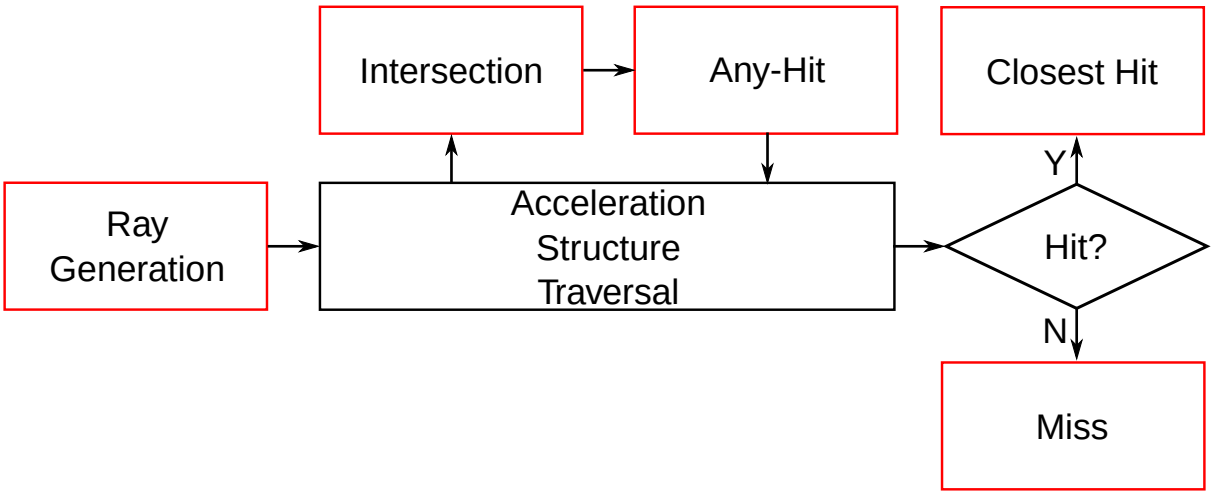



*Figure 30. Ray tracing pipeline execution*

**Caption**

Interaction between the different shader stages in the ray tracing pipeline

Within the ray tracing pipeline, a pipeline trace ray instruction **can** be called to perform a ray traversal that invokes the various ray tracing shader stages during its execution. The relationship between the ray tracing pipeline object and the geometries present in the acceleration structure traversed is passed into the ray tracing command in a VkBuffer object known as a *shader binding table*. `OpExecuteCallableKHR` can also be used in ray tracing pipelines to invoke a callable shader.

During execution, control alternates between scheduling and other operations. The scheduling functionality is implementation-specific and is responsible for workload execution. The shader stages are programmable. *Traversal*, which refers to the process of traversing acceleration structures to find potential intersections of rays with geometry, is fixed function.

The programmable portions of the pipeline are exposed in a single-ray programming model, with each invocation handling one ray at a time. Memory operations **can** be synchronized using standard memory barriers. The `Workgroup` scope and variables with a storage class of `Workgroup` **must** not be used in the ray tracing pipeline.

## 40.1. Shader Call Instructions

A *shader call* is an instruction which **may** cause execution to continue elsewhere by creating one or more invocations that execute a different shader stage.

The following table lists all shader call instructions and which stages each one **can** directly call.

| Instruction | Intersection | Any-Hit | Closest Hit | Miss | Callable |
|---|---|---|---|---|---|
| OpTraceRayKHR | X | X | X | X | |
| OpTraceRayMotionNV | X | X | X | X | |
| OpReportIntersectionKHR | | X | | | |
| OpExecuteCallableKHR | | | | | X |
| OpHitObjectTraceRayNV | X | X | | | |
| OpHitObjectTraceRayMotionNV | X | X | | | |
| OpHitObjectExecuteShaderNV | | | X | X | |

The invocations created by shader call instructions are grouped into subgroups by the implementation. Those subgroups **may** be unrelated to the subgroup of the parent invocation.

*Pipeline trace ray instructions* **can** be used recursively; invoked shaders **can** themselves execute pipeline trace ray instructions, to a maximum depth defined by the maxRecursionDepth or maxRayRecursionDepth limit.

Shaders directly invoked from the API always have a recursion depth of 0; each shader executed by a pipeline trace ray instruction has a recursion depth one higher than the recursion depth of the shader which invoked it. Applications **must** not invoke a shader with a recursion depth greater than the value of maxRecursionDepth or maxPipelineRayRecursionDepth specified in the pipeline.

There is no explicit recursion limit for other shader call instructions which may recurse (e.g. OpExecuteCallableKHR) but there is an upper bound determined by the stack size.

An *invocation repack instruction* is a ray tracing instruction where the implementation **may** change the set of invocations that are executing. When a repack instruction is encountered, the invocation is suspended and a new invocation begins and executes the instruction. After executing the repack instruction (which **may** result in other ray tracing shader stages executing) the new invocation ends and the original invocation is resumed, but it **may** be resumed in a different subgroup or at a different SubgroupLocalInvocationId within the same subgroup. When a subset of invocations in a subgroup execute the invocation repack instruction, those that do not execute it remain in the same subgroup at the same SubgroupLocalInvocationId.

The OpTraceRayKHR, OpTraceRayMotionNV, OpReorderThreadWithHintNV, OpReorderThreadWithHitObjectNV, OpReportIntersectionKHR, and OpExecuteCallableKHR instructions are invocation repack instructions.

The invocations that are executing before a shader call instruction, after the instruction, or are created by the instruction, are shader-call-related.

If the implementation changes the composition of subgroups, the values of SubgroupSize, SubgroupLocalInvocationId, SMIDNV, WarpIDNV, and builtin variables that are derived from them (SubgroupEqMask, SubgroupGeMask, SubgroupGtMask, SubgroupLeMask, SubgroupLtMask) **must** be changed accordingly by the invocation repack instruction. The application **must** use Volatile semantics on

these BuiltIn variables when used in the ray generation, closest hit, miss, intersection, and callable shaders. Similarly, the application **must** use Volatile semantics on any RayTmaxKHR decorated Builtin used in an intersection shader.

> Subgroup operations are permitted in the programmable ray tracing shader stages. However, shader call instructions place a bound on where results of subgroup instructions or subgroup-scoped instructions that execute the dynamic instance of that instruction are potentially valid. For example, care **must** be taken when using the result of a ballot operation that was computed before an invocation repack instruction, after that repack instruction. The ballot **may** be incorrect as the set of invocations could have changed.
>
> **NOTE** While the SubgroupSize built-in is required to be declared Volatile, its value will never change unless VK_PIPELINE_SHADER_STAGE_CREATE_ALLOW_VARYING_SUBGROUP_SIZE_BIT is set on pipeline creation, as without that bit set, its value is required to match that of VkPhysicalDeviceSubgroupProperties::subgroupSize.
>
> For clock operations, the value of a Subgroup scoped OpReadClockKHR read before the dynamic instance of a repack instruction **should** not be compared to the result of that clock instruction after the repack instruction.

When a ray tracing shader executes a dynamic instance of an invocation repack instruction which results in another ray tracing shader being invoked, their instructions are related by shader-call-order.

For ray tracing invocations that are shader-call-related:

- memory operations on StorageBuffer, Image, and ShaderRecordBufferKHR storage classes **can** be synchronized using the ShaderCallKHR scope.
- the CallableDataKHR, IncomingCallableDataKHR, RayPayloadKHR, HitAttributeKHR, and IncomingRayPayloadKHR storage classes are system-synchronized and no application availability and visibility operations are required.
- memory operations within a single invocation before and after the shader call instruction are ordered by program-order and do not require explicit synchronization.

## 40.2. Ray Tracing Commands

*Ray tracing commands* provoke work in the ray tracing pipeline. Ray tracing commands are recorded into a command buffer and when executed by a queue will produce work that executes according to the currently bound ray tracing pipeline. A ray tracing pipeline **must** be bound to a command buffer before any ray tracing commands are recorded in that command buffer.

To dispatch ray tracing use:

```
// Provided by VK_NV_ray_tracing
void vkCmdTraceRaysNV(
    VkCommandBuffer                                 commandBuffer,
    VkBuffer                                        raygenShaderBindingTableBuffer,
    VkDeviceSize                                    raygenShaderBindingOffset,
    VkBuffer                                        missShaderBindingTableBuffer,
    VkDeviceSize                                    missShaderBindingOffset,
    VkDeviceSize                                    missShaderBindingStride,
    VkBuffer                                        hitShaderBindingTableBuffer,
    VkDeviceSize                                    hitShaderBindingOffset,
    VkDeviceSize                                    hitShaderBindingStride,
    VkBuffer                                        callableShaderBindingTableBuffer,
    VkDeviceSize                                    callableShaderBindingOffset,
    VkDeviceSize                                    callableShaderBindingStride,
    uint32_t                                        width,
    uint32_t                                        height,
    uint32_t                                        depth);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `raygenShaderBindingTableBuffer` is the buffer object that holds the shader binding table data for the ray generation shader stage.
- `raygenShaderBindingOffset` is the offset in bytes (relative to `raygenShaderBindingTableBuffer`) of the ray generation shader being used for the trace.
- `missShaderBindingTableBuffer` is the buffer object that holds the shader binding table data for the miss shader stage.
- `missShaderBindingOffset` is the offset in bytes (relative to `missShaderBindingTableBuffer`) of the miss shader being used for the trace.
- `missShaderBindingStride` is the size in bytes of each shader binding table record in `missShaderBindingTableBuffer`.
- `hitShaderBindingTableBuffer` is the buffer object that holds the shader binding table data for the hit shader stages.
- `hitShaderBindingOffset` is the offset in bytes (relative to `hitShaderBindingTableBuffer`) of the hit shader group being used for the trace.
- `hitShaderBindingStride` is the size in bytes of each shader binding table record in `hitShaderBindingTableBuffer`.
- `callableShaderBindingTableBuffer` is the buffer object that holds the shader binding table data for the callable shader stage.
- `callableShaderBindingOffset` is the offset in bytes (relative to `callableShaderBindingTableBuffer`) of the callable shader being used for the trace.
- `callableShaderBindingStride` is the size in bytes of each shader binding table record in `callableShaderBindingTableBuffer`.
- `width` is the width of the ray trace query dimensions.
- `height` is height of the ray trace query dimensions.

- depth is depth of the ray trace query dimensions.

When the command is executed, a ray generation group of width × height × depth rays is assembled.

### Valid Usage

- VUID-vkCmdTraceRaysNV-magFilter-04553
  If a VkSampler created with magFilter or minFilter equal to VK_FILTER_LINEAR, reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE, and compareEnable equal to VK_FALSE is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT
- VUID-vkCmdTraceRaysNV-magFilter-09598
  If a VkSampler created with magFilter or minFilter equal to VK_FILTER_LINEAR and reductionMode equal to either VK_SAMPLER_REDUCTION_MODE_MIN or VK_SAMPLER_REDUCTION_MODE_MAX is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT
- VUID-vkCmdTraceRaysNV-mipmapMode-04770
  If a VkSampler created with mipmapMode equal to VK_SAMPLER_MIPMAP_MODE_LINEAR, reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE, and compareEnable equal to VK_FALSE is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT
- VUID-vkCmdTraceRaysNV-mipmapMode-09599
  If a VkSampler created with mipmapMode equal to VK_SAMPLER_MIPMAP_MODE_LINEAR and reductionMode equal to either VK_SAMPLER_REDUCTION_MODE_MIN or VK_SAMPLER_REDUCTION_MODE_MAX is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT
- VUID-vkCmdTraceRaysNV-unnormalizedCoordinates-09635
  If a VkSampler created with unnormalizedCoordinates equal to VK_TRUE is used to sample a VkImageView as a result of this command, then the image view's levelCount and layerCount **must** be 1
- VUID-vkCmdTraceRaysNV-unnormalizedCoordinates-09636
  If a VkSampler created with unnormalizedCoordinates equal to VK_TRUE is used to sample a VkImageView as a result of this command, then the image view's viewType **must** be VK_IMAGE_VIEW_TYPE_1D or VK_IMAGE_VIEW_TYPE_2D
- VUID-vkCmdTraceRaysNV-None-06479
  If a VkImageView is sampled with depth comparison, the image view's format features **must** contain VK_FORMAT_FEATURE_2_SAMPLED_IMAGE_DEPTH_COMPARISON_BIT
- VUID-vkCmdTraceRaysNV-None-02691
  If a VkImageView is accessed using atomic operations as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_STORAGE_IMAGE_ATOMIC_BIT

- VUID-vkCmdTraceRaysNV-None-07888
  If a VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER descriptor is accessed using atomic operations as a result of this command, then the storage texel buffer's format features **must** contain VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_ATOMIC_BIT
- VUID-vkCmdTraceRaysNV-None-02692
  If a VkImageView is sampled with VK_FILTER_CUBIC_EXT as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_CUBIC_BIT_EXT
- VUID-vkCmdTraceRaysNV-None-02693
  If the VK_EXT_filter_cubic extension is not enabled and any VkImageView is sampled with VK_FILTER_CUBIC_EXT as a result of this command, it **must** not have a VkImageViewType of VK_IMAGE_VIEW_TYPE_3D, VK_IMAGE_VIEW_TYPE_CUBE, or VK_IMAGE_VIEW_TYPE_CUBE_ARRAY
- VUID-vkCmdTraceRaysNV-filterCubic-02694
  Any VkImageView being sampled with VK_FILTER_CUBIC_EXT as a result of this command **must** have a VkImageViewType and format that supports cubic filtering, as specified by VkFilterCubicImageViewImageFormatPropertiesEXT::filterCubic returned by vkGetPhysicalDeviceImageFormatProperties2
- VUID-vkCmdTraceRaysNV-filterCubicMinmax-02695
  Any VkImageView being sampled with VK_FILTER_CUBIC_EXT with a reduction mode of either VK_SAMPLER_REDUCTION_MODE_MIN or VK_SAMPLER_REDUCTION_MODE_MAX as a result of this command **must** have a VkImageViewType and format that supports cubic filtering together with minmax filtering, as specified by VkFilterCubicImageViewImageFormatPropertiesEXT::filterCubicMinmax returned by vkGetPhysicalDeviceImageFormatProperties2
- VUID-vkCmdTraceRaysNV-cubicRangeClamp-09212
  If the cubicRangeClamp feature is not enabled, then any VkImageView being sampled with VK_FILTER_CUBIC_EXT as a result of this command **must** not have a VkSamplerReductionModeCreateInfo::reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE_RANGECLAMP_QCOM
- VUID-vkCmdTraceRaysNV-reductionMode-09213
  Any VkImageView being sampled with a VkSamplerReductionModeCreateInfo::reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE_RANGECLAMP_QCOM as a result of this command **must** sample with VK_FILTER_CUBIC_EXT
- VUID-vkCmdTraceRaysNV-selectableCubicWeights-09214
  If the selectableCubicWeights feature is not enabled, then any VkImageView being sampled with VK_FILTER_CUBIC_EXT as a result of this command **must** have VkSamplerCubicWeightsCreateInfoQCOM::cubicWeights equal to VK_CUBIC_FILTER_WEIGHTS_CATMULL_ROM_QCOM
- VUID-vkCmdTraceRaysNV-flags-02696
  Any VkImage created with a VkImageCreateInfo::flags containing VK_IMAGE_CREATE_CORNER_SAMPLED_BIT_NV sampled as a result of this command **must** only be sampled using a VkSamplerAddressMode of VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_EDGE
- VUID-vkCmdTraceRaysNV-OpTypeImage-07027
  For any VkImageView being written as a storage image where the image format field of

the OpTypeImage is Unknown, the view's format features **must** contain VK_FORMAT_FEATURE_2_STORAGE_WRITE_WITHOUT_FORMAT_BIT

- VUID-vkCmdTraceRaysNV-OpTypeImage-07028
  For any VkImageView being read as a storage image where the image format field of the OpTypeImage is Unknown, the view's format features **must** contain VK_FORMAT_FEATURE_2_STORAGE_READ_WITHOUT_FORMAT_BIT
- VUID-vkCmdTraceRaysNV-OpTypeImage-07029
  For any VkBufferView being written as a storage texel buffer where the image format field of the OpTypeImage is Unknown, the view's buffer features **must** contain VK_FORMAT_FEATURE_2_STORAGE_WRITE_WITHOUT_FORMAT_BIT
- VUID-vkCmdTraceRaysNV-OpTypeImage-07030
  Any VkBufferView being read as a storage texel buffer where the image format field of the OpTypeImage is Unknown then the view's buffer features **must** contain VK_FORMAT_FEATURE_2_STORAGE_READ_WITHOUT_FORMAT_BIT
- VUID-vkCmdTraceRaysNV-None-08600
  For each set *n* that is statically used by a bound shader, a descriptor set **must** have been bound to *n* at the same pipeline bind point, with a VkPipelineLayout that is compatible for set *n*, with the VkPipelineLayout used to create the current VkPipeline or the VkDescriptorSetLayout array used to create the current VkShaderEXT , as described in Pipeline Layout Compatibility
- VUID-vkCmdTraceRaysNV-None-08601
  For each push constant that is statically used by a bound shader, a push constant value **must** have been set for the same pipeline bind point, with a VkPipelineLayout that is compatible for push constants, with the VkPipelineLayout used to create the current VkPipeline or the VkDescriptorSetLayout array used to create the current VkShaderEXT , as described in Pipeline Layout Compatibility
- VUID-vkCmdTraceRaysNV-None-10068
  For each array of resources that is used by a bound shader, the indices used to access members of the array **must** be less than the descriptor count for the identified binding in the descriptor sets used by this command
- VUID-vkCmdTraceRaysNV-maintenance4-08602
  If the maintenance4 feature is not enabled, then for each push constant that is statically used by a bound shader, a push constant value **must** have been set for the same pipeline bind point, with a VkPipelineLayout that is compatible for push constants, with the VkPipelineLayout used to create the current VkPipeline or the VkDescriptorSetLayout and VkPushConstantRange arrays used to create the current VkShaderEXT , as described in Pipeline Layout Compatibility
- VUID-vkCmdTraceRaysNV-None-08114
  Descriptors in each bound descriptor set, specified via vkCmdBindDescriptorSets, **must** be valid as described by descriptor validity if they are statically used by the VkPipeline bound to the pipeline bind point used by this command and the bound VkPipeline was not created with VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT
- VUID-vkCmdTraceRaysNV-None-08115
  If the descriptors used by the VkPipeline bound to the pipeline bind point were specified

via vkCmdBindDescriptorSets, the bound VkPipeline **must** have been created without VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT

- VUID-vkCmdTraceRaysNV-None-08116
  Descriptors in bound descriptor buffers, specified via vkCmdSetDescriptorBufferOffsetsEXT, **must** be valid if they are dynamically used by the VkPipeline bound to the pipeline bind point used by this command and the bound VkPipeline was created with VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT
- VUID-vkCmdTraceRaysNV-None-08604
  Descriptors in bound descriptor buffers, specified via vkCmdSetDescriptorBufferOffsetsEXT, **must** be valid if they are dynamically used by any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command
- VUID-vkCmdTraceRaysNV-None-08117
  If the descriptors used by the VkPipeline bound to the pipeline bind point were specified via vkCmdSetDescriptorBufferOffsetsEXT, the bound VkPipeline **must** have been created with VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT
- VUID-vkCmdTraceRaysNV-None-08119
  If a descriptor is dynamically used with a VkPipeline created with VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT, the descriptor memory **must** be resident
- VUID-vkCmdTraceRaysNV-None-08605
  If a descriptor is dynamically used with a VkShaderEXT created with a VkDescriptorSetLayout that was created with VK_DESCRIPTOR_SET_LAYOUT_CREATE_DESCRIPTOR_BUFFER_BIT_EXT, the descriptor memory **must** be resident
- VUID-vkCmdTraceRaysNV-None-08606
  If the shaderObject feature is not enabled, a valid pipeline **must** be bound to the pipeline bind point used by this command
- VUID-vkCmdTraceRaysNV-None-08608
  If a pipeline is bound to the pipeline bind point used by this command, there **must** not have been any calls to dynamic state setting commands for any state specified statically in the VkPipeline object bound to the pipeline bind point used by this command, since that pipeline was bound
- VUID-vkCmdTraceRaysNV-None-08609
  If the VkPipeline object bound to the pipeline bind point used by this command or any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a VkSampler object that uses unnormalized coordinates, that sampler **must** not be used to sample from any VkImage with a VkImageView of the type VK_IMAGE_VIEW_TYPE_3D, VK_IMAGE_VIEW_TYPE_CUBE, VK_IMAGE_VIEW_TYPE_1D_ARRAY, VK_IMAGE_VIEW_TYPE_2D_ARRAY or VK_IMAGE_VIEW_TYPE_CUBE_ARRAY, in any shader stage
- VUID-vkCmdTraceRaysNV-None-08610
  If the VkPipeline object bound to the pipeline bind point used by this command or any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a VkSampler object that uses unnormalized coordinates, that sampler **must** not be used with any of the SPIR-V OpImageSample* or OpImageSparseSample*

instructions with `ImplicitLod`, `Dref` or `Proj` in their name, in any shader stage

- VUID-vkCmdTraceRaysNV-None-08611
  If the VkPipeline object bound to the pipeline bind point used by this command or any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a VkSampler object that uses unnormalized coordinates, that sampler **must** not be used with any of the SPIR-V `OpImageSample*` or `OpImageSparseSample*` instructions that includes a LOD bias or any offset values, in any shader stage

- VUID-vkCmdTraceRaysNV-None-08607
  If the `shaderObject` is enabled, either a valid pipeline **must** be bound to the pipeline bind point used by this command, or a valid combination of valid and VK_NULL_HANDLE shader objects **must** be bound to every supported shader stage corresponding to the pipeline bind point used by this command

- VUID-vkCmdTraceRaysNV-uniformBuffers-06935
  If any stage of the VkPipeline object bound to the pipeline bind point used by this command accesses a uniform buffer, and that stage was created without enabling either `VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_EXT` or `VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_2_EXT` for `uniformBuffers`, and the `robustBufferAccess` feature is not enabled, that stage **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point

- VUID-vkCmdTraceRaysNV-None-08612
  If the `robustBufferAccess` feature is not enabled, and any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a uniform buffer, it **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point

- VUID-vkCmdTraceRaysNV-storageBuffers-06936
  If any stage of the VkPipeline object bound to the pipeline bind point used by this command accesses a storage buffer, and that stage was created without enabling either `VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_EXT` or `VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_2_EXT` for `storageBuffers`, and the `robustBufferAccess` feature is not enabled, that stage **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point

- VUID-vkCmdTraceRaysNV-None-08613
  If the `robustBufferAccess` feature is not enabled, and any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a storage buffer, it **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point

- VUID-vkCmdTraceRaysNV-commandBuffer-02707
  If `commandBuffer` is an unprotected command buffer and `protectedNoFault` is not supported, any resource accessed by bound shaders **must** not be a protected resource

- VUID-vkCmdTraceRaysNV-None-06550
  If a bound shader accesses a VkSampler or VkImageView object that enables sampler $Y'C_BC_R$ conversion, that object **must** only be used with `OpImageSample*` or `OpImageSparseSample*` instructions

- VUID-vkCmdTraceRaysNV-ConstOffset-06551
  If a bound shader accesses a VkSampler or VkImageView object that enables sampler Y'C$_B$C$_R$ conversion, that object **must** not use the `ConstOffset` and `Offset` operands
- VUID-vkCmdTraceRaysNV-viewType-07752
  If a VkImageView is accessed as a result of this command, then the image view's `viewType` **must** match the `Dim` operand of the `OpTypeImage` as described in Compatibility Between SPIR-V Image Dimensions and Vulkan ImageView Types
- VUID-vkCmdTraceRaysNV-format-07753
  If a VkImageView is accessed as a result of this command, then the numeric type of the image view's `format` and the `Sampled Type` operand of the `OpTypeImage` **must** match
- VUID-vkCmdTraceRaysNV-OpImageWrite-08795
  If a VkImageView created with a format other than `VK_FORMAT_A8_UNORM_KHR` is accessed using `OpImageWrite` as a result of this command, then the `Type` of the `Texel` operand of that instruction **must** have at least as many components as the image view's format
- VUID-vkCmdTraceRaysNV-OpImageWrite-08796
  If a VkImageView created with the format `VK_FORMAT_A8_UNORM_KHR` is accessed using `OpImageWrite` as a result of this command, then the `Type` of the `Texel` operand of that instruction **must** have four components
- VUID-vkCmdTraceRaysNV-OpImageWrite-04469
  If a VkBufferView is accessed using `OpImageWrite` as a result of this command, then the `Type` of the `Texel` operand of that instruction **must** have at least as many components as the buffer view's format
- VUID-vkCmdTraceRaysNV-SampledType-04470
  If a VkImageView with a VkFormat that has a 64-bit component width is accessed as a result of this command, the `SampledType` of the `OpTypeImage` operand of that instruction **must** have a `Width` of 64
- VUID-vkCmdTraceRaysNV-SampledType-04471
  If a VkImageView with a VkFormat that has a component width less than 64-bit is accessed as a result of this command, the `SampledType` of the `OpTypeImage` operand of that instruction **must** have a `Width` of 32
- VUID-vkCmdTraceRaysNV-SampledType-04472
  If a VkBufferView with a VkFormat that has a 64-bit component width is accessed as a result of this command, the `SampledType` of the `OpTypeImage` operand of that instruction **must** have a `Width` of 64
- VUID-vkCmdTraceRaysNV-SampledType-04473
  If a VkBufferView with a VkFormat that has a component width less than 64-bit is accessed as a result of this command, the `SampledType` of the `OpTypeImage` operand of that instruction **must** have a `Width` of 32
- VUID-vkCmdTraceRaysNV-sparseImageInt64Atomics-04474
  If the `sparseImageInt64Atomics` feature is not enabled, VkImage objects created with the `VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT` flag **must** not be accessed by atomic instructions through an `OpTypeImage` with a `SampledType` with a `Width` of 64 by this command
- VUID-vkCmdTraceRaysNV-sparseImageInt64Atomics-04475

If the sparseImageInt64Atomics feature is not enabled, VkBuffer objects created with the VK_BUFFER_CREATE_SPARSE_RESIDENCY_BIT flag **must** not be accessed by atomic instructions through an OpTypeImage with a SampledType with a Width of 64 by this command

- VUID-vkCmdTraceRaysNV-OpImageWeightedSampleQCOM-06971
  If OpImageWeightedSampleQCOM is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_WEIGHT_SAMPLED_IMAGE_BIT_QCOM
- VUID-vkCmdTraceRaysNV-OpImageWeightedSampleQCOM-06972
  If OpImageWeightedSampleQCOM uses a VkImageView as a sample weight image as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_WEIGHT_IMAGE_BIT_QCOM
- VUID-vkCmdTraceRaysNV-OpImageBoxFilterQCOM-06973
  If OpImageBoxFilterQCOM is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_BOX_FILTER_SAMPLED_BIT_QCOM
- VUID-vkCmdTraceRaysNV-OpImageBlockMatchSSDQCOM-06974
  If OpImageBlockMatchSSDQCOM is used to read from an VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM
- VUID-vkCmdTraceRaysNV-OpImageBlockMatchSADQCOM-06975
  If OpImageBlockMatchSADQCOM is used to read from an VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM
- VUID-vkCmdTraceRaysNV-OpImageBlockMatchSADQCOM-06976
  If OpImageBlockMatchSADQCOM or OpImageBlockMatchSSDQCOM is used to read from a reference image as result of this command, then the specified reference coordinates **must** not fail integer texel coordinate validation
- VUID-vkCmdTraceRaysNV-OpImageWeightedSampleQCOM-06977
  If OpImageWeightedSampleQCOM, OpImageBoxFilterQCOM, OpImageBlockMatchWindowSSDQCOM, OpImageBlockMatchWindowSADQCOM, OpImageBlockMatchGatherSSDQCOM, OpImageBlockMatchGatherSADQCOM, OpImageBlockMatchSSDQCOM, or OpImageBlockMatchSADQCOM uses a VkSampler as a result of this command, then the sampler **must** have been created with VK_SAMPLER_CREATE_IMAGE_PROCESSING_BIT_QCOM
- VUID-vkCmdTraceRaysNV-OpImageWeightedSampleQCOM-06978
  If any command other than OpImageWeightedSampleQCOM, OpImageBoxFilterQCOM, OpImageBlockMatchWindowSSDQCOM, OpImageBlockMatchWindowSADQCOM, OpImageBlockMatchGatherSSDQCOM, OpImageBlockMatchGatherSADQCOM, OpImageBlockMatchSSDQCOM, or OpImageBlockMatchSADQCOM uses a VkSampler as a result of this command, then the sampler **must** not have been created with VK_SAMPLER_CREATE_IMAGE_PROCESSING_BIT_QCOM
- VUID-vkCmdTraceRaysNV-OpImageBlockMatchWindow-09215
  If a OpImageBlockMatchWindow*QCOM or OpImageBlockMatchGather*QCOM instruction is used to read from an VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM

- VUID-vkCmdTraceRaysNV-OpImageBlockMatchWindow-09216
  If a `OpImageBlockMatchWindow*QCOM` or `OpImageBlockMatchGather*QCOM` instruction is used to read from an VkImageView as a result of this command, then the image view's format **must** be a single-component format
- VUID-vkCmdTraceRaysNV-OpImageBlockMatchWindow-09217
  If a `OpImageBlockMatchWindow*QCOM` or `OpImageBlockMatchGather*QCOM` read from a reference image as result of this command, then the specified reference coordinates **must** not fail integer texel coordinate validation
- VUID-vkCmdTraceRaysNV-None-07288
  Any shader invocation executed by this command **must** terminate
- VUID-vkCmdTraceRaysNV-None-09600
  If a descriptor with type equal to any of `VK_DESCRIPTOR_TYPE_SAMPLE_WEIGHT_IMAGE_QCOM`, `VK_DESCRIPTOR_TYPE_BLOCK_MATCH_IMAGE_QCOM`, `VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE`, `VK_DESCRIPTOR_TYPE_STORAGE_IMAGE`, or `VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT` is accessed as a result of this command, the image subresource identified by that descriptor **must** be in the image layout identified when the descriptor was written
- VUID-vkCmdTraceRaysNV-None-03429
  Any shader group handle referenced by this call **must** have been queried from the currently bound ray tracing pipeline
- VUID-vkCmdTraceRaysNV-None-09458
  If the bound ray tracing pipeline state was created with the `VK_DYNAMIC_STATE_RAY_TRACING_PIPELINE_STACK_SIZE_KHR` dynamic state enabled then vkCmdSetRayTracingPipelineStackSizeKHR **must** have been called in the current command buffer prior to this trace command
- VUID-vkCmdTraceRaysNV-commandBuffer-04624
  `commandBuffer` **must** not be a protected command buffer
- VUID-vkCmdTraceRaysNV-maxRecursionDepth-03625
  This command **must** not cause a pipeline trace ray instruction to be executed from a shader invocation with a recursion depth greater than the value of `maxRecursionDepth` used to create the bound ray tracing pipeline
- VUID-vkCmdTraceRaysNV-raygenShaderBindingTableBuffer-04042
  If `raygenShaderBindingTableBuffer` is non-sparse then it **must** be bound completely and contiguously to a single `VkDeviceMemory` object
- VUID-vkCmdTraceRaysNV-raygenShaderBindingOffset-02455
  `raygenShaderBindingOffset` **must** be less than the size of `raygenShaderBindingTableBuffer`
- VUID-vkCmdTraceRaysNV-raygenShaderBindingOffset-02456
  `raygenShaderBindingOffset` **must** be a multiple of `VkPhysicalDeviceRayTracingPropertiesNV::shaderGroupBaseAlignment`
- VUID-vkCmdTraceRaysNV-missShaderBindingTableBuffer-04043
  If `missShaderBindingTableBuffer` is non-sparse then it **must** be bound completely and contiguously to a single `VkDeviceMemory` object
- VUID-vkCmdTraceRaysNV-missShaderBindingOffset-02457
  `missShaderBindingOffset` **must** be less than the size of `missShaderBindingTableBuffer`

- VUID-vkCmdTraceRaysNV-missShaderBindingOffset-02458
  `missShaderBindingOffset` **must** be a multiple of `VkPhysicalDeviceRayTracingPropertiesNV::shaderGroupBaseAlignment`
- VUID-vkCmdTraceRaysNV-hitShaderBindingTableBuffer-04044
  If `hitShaderBindingTableBuffer` is non-sparse then it **must** be bound completely and contiguously to a single `VkDeviceMemory` object
- VUID-vkCmdTraceRaysNV-hitShaderBindingOffset-02459
  `hitShaderBindingOffset` **must** be less than the size of `hitShaderBindingTableBuffer`
- VUID-vkCmdTraceRaysNV-hitShaderBindingOffset-02460
  `hitShaderBindingOffset` **must** be a multiple of `VkPhysicalDeviceRayTracingPropertiesNV::shaderGroupBaseAlignment`
- VUID-vkCmdTraceRaysNV-callableShaderBindingTableBuffer-04045
  If `callableShaderBindingTableBuffer` is non-sparse then it **must** be bound completely and contiguously to a single `VkDeviceMemory` object
- VUID-vkCmdTraceRaysNV-callableShaderBindingOffset-02461
  `callableShaderBindingOffset` **must** be less than the size of `callableShaderBindingTableBuffer`
- VUID-vkCmdTraceRaysNV-callableShaderBindingOffset-02462
  `callableShaderBindingOffset` **must** be a multiple of `VkPhysicalDeviceRayTracingPropertiesNV::shaderGroupBaseAlignment`
- VUID-vkCmdTraceRaysNV-missShaderBindingStride-02463
  `missShaderBindingStride` **must** be a multiple of `VkPhysicalDeviceRayTracingPropertiesNV::shaderGroupHandleSize`
- VUID-vkCmdTraceRaysNV-hitShaderBindingStride-02464
  `hitShaderBindingStride` **must** be a multiple of `VkPhysicalDeviceRayTracingPropertiesNV::shaderGroupHandleSize`
- VUID-vkCmdTraceRaysNV-callableShaderBindingStride-02465
  `callableShaderBindingStride` **must** be a multiple of `VkPhysicalDeviceRayTracingPropertiesNV::shaderGroupHandleSize`
- VUID-vkCmdTraceRaysNV-missShaderBindingStride-02466
  `missShaderBindingStride` **must** be less than or equal to `VkPhysicalDeviceRayTracingPropertiesNV::maxShaderGroupStride`
- VUID-vkCmdTraceRaysNV-hitShaderBindingStride-02467
  `hitShaderBindingStride` **must** be less than or equal to `VkPhysicalDeviceRayTracingPropertiesNV::maxShaderGroupStride`
- VUID-vkCmdTraceRaysNV-callableShaderBindingStride-02468
  `callableShaderBindingStride` **must** be less than or equal to `VkPhysicalDeviceRayTracingPropertiesNV::maxShaderGroupStride`
- VUID-vkCmdTraceRaysNV-width-02469
  `width` **must** be less than or equal to `VkPhysicalDeviceLimits::maxComputeWorkGroupCount`[0]
- VUID-vkCmdTraceRaysNV-height-02470
  `height` **must** be less than or equal to `VkPhysicalDeviceLimits::maxComputeWorkGroupCount`[1]

- VUID-vkCmdTraceRaysNV-depth-02471
  `depth` **must** be less than or equal to `VkPhysicalDeviceLimits::maxComputeWorkGroupCount`[2]

## Valid Usage (Implicit)

- VUID-vkCmdTraceRaysNV-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdTraceRaysNV-raygenShaderBindingTableBuffer-parameter
  `raygenShaderBindingTableBuffer` **must** be a valid VkBuffer handle
- VUID-vkCmdTraceRaysNV-missShaderBindingTableBuffer-parameter
  If `missShaderBindingTableBuffer` is not VK_NULL_HANDLE, `missShaderBindingTableBuffer` **must** be a valid VkBuffer handle
- VUID-vkCmdTraceRaysNV-hitShaderBindingTableBuffer-parameter
  If `hitShaderBindingTableBuffer` is not VK_NULL_HANDLE, `hitShaderBindingTableBuffer` **must** be a valid VkBuffer handle
- VUID-vkCmdTraceRaysNV-callableShaderBindingTableBuffer-parameter
  If `callableShaderBindingTableBuffer` is not VK_NULL_HANDLE, `callableShaderBindingTableBuffer` **must** be a valid VkBuffer handle
- VUID-vkCmdTraceRaysNV-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdTraceRaysNV-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support compute operations
- VUID-vkCmdTraceRaysNV-renderpass
  This command **must** only be called outside of a render pass instance
- VUID-vkCmdTraceRaysNV-videocoding
  This command **must** only be called outside of a video coding scope
- VUID-vkCmdTraceRaysNV-commonparent
  Each of `callableShaderBindingTableBuffer`, `commandBuffer`, `hitShaderBindingTableBuffer`, `missShaderBindingTableBuffer`, and `raygenShaderBindingTableBuffer` that are valid handles of non-ignored parameters **must** have been created, allocated, or retrieved from the same VkDevice

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Outside | Outside | Compute | Action |

To dispatch ray tracing use:

```
// Provided by VK_KHR_ray_tracing_pipeline
void vkCmdTraceRaysKHR(
    VkCommandBuffer                             commandBuffer,
    const VkStridedDeviceAddressRegionKHR*      pRaygenShaderBindingTable,
    const VkStridedDeviceAddressRegionKHR*      pMissShaderBindingTable,
    const VkStridedDeviceAddressRegionKHR*      pHitShaderBindingTable,
    const VkStridedDeviceAddressRegionKHR*      pCallableShaderBindingTable,
    uint32_t                                    width,
    uint32_t                                    height,
    uint32_t                                    depth);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `pRaygenShaderBindingTable` is a VkStridedDeviceAddressRegionKHR that holds the shader binding table data for the ray generation shader stage.
- `pMissShaderBindingTable` is a VkStridedDeviceAddressRegionKHR that holds the shader binding table data for the miss shader stage.
- `pHitShaderBindingTable` is a VkStridedDeviceAddressRegionKHR that holds the shader binding table data for the hit shader stage.
- `pCallableShaderBindingTable` is a VkStridedDeviceAddressRegionKHR that holds the shader binding table data for the callable shader stage.
- `width` is the width of the ray trace query dimensions.
- `height` is height of the ray trace query dimensions.
- `depth` is depth of the ray trace query dimensions.

When the command is executed, a ray generation group of `width` × `height` × `depth` rays is assembled.

## Valid Usage

- VUID-vkCmdTraceRaysKHR-magFilter-04553
  If a VkSampler created with `magFilter` or `minFilter` equal to `VK_FILTER_LINEAR`, `reductionMode` equal to `VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE`, and `compareEnable` equal to `VK_FALSE` is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT`

- VUID-vkCmdTraceRaysKHR-magFilter-09598
  If a VkSampler created with magFilter or minFilter equal to VK_FILTER_LINEAR and reductionMode equal to either VK_SAMPLER_REDUCTION_MODE_MIN or VK_SAMPLER_REDUCTION_MODE_MAX is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT
- VUID-vkCmdTraceRaysKHR-mipmapMode-04770
  If a VkSampler created with mipmapMode equal to VK_SAMPLER_MIPMAP_MODE_LINEAR, reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE, and compareEnable equal to VK_FALSE is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT
- VUID-vkCmdTraceRaysKHR-mipmapMode-09599
  If a VkSampler created with mipmapMode equal to VK_SAMPLER_MIPMAP_MODE_LINEAR and reductionMode equal to either VK_SAMPLER_REDUCTION_MODE_MIN or VK_SAMPLER_REDUCTION_MODE_MAX is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT
- VUID-vkCmdTraceRaysKHR-unnormalizedCoordinates-09635
  If a VkSampler created with unnormalizedCoordinates equal to VK_TRUE is used to sample a VkImageView as a result of this command, then the image view's levelCount and layerCount **must** be 1
- VUID-vkCmdTraceRaysKHR-unnormalizedCoordinates-09636
  If a VkSampler created with unnormalizedCoordinates equal to VK_TRUE is used to sample a VkImageView as a result of this command, then the image view's viewType **must** be VK_IMAGE_VIEW_TYPE_1D or VK_IMAGE_VIEW_TYPE_2D
- VUID-vkCmdTraceRaysKHR-None-06479
  If a VkImageView is sampled with depth comparison, the image view's format features **must** contain VK_FORMAT_FEATURE_2_SAMPLED_IMAGE_DEPTH_COMPARISON_BIT
- VUID-vkCmdTraceRaysKHR-None-02691
  If a VkImageView is accessed using atomic operations as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_STORAGE_IMAGE_ATOMIC_BIT
- VUID-vkCmdTraceRaysKHR-None-07888
  If a VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER descriptor is accessed using atomic operations as a result of this command, then the storage texel buffer's format features **must** contain VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_ATOMIC_BIT
- VUID-vkCmdTraceRaysKHR-None-02692
  If a VkImageView is sampled with VK_FILTER_CUBIC_EXT as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_CUBIC_BIT_EXT
- VUID-vkCmdTraceRaysKHR-None-02693
  If the VK_EXT_filter_cubic extension is not enabled and any VkImageView is sampled with VK_FILTER_CUBIC_EXT as a result of this command, it **must** not have a VkImageViewType of

VK_IMAGE_VIEW_TYPE_3D, VK_IMAGE_VIEW_TYPE_CUBE, or VK_IMAGE_VIEW_TYPE_CUBE_ARRAY

- VUID-vkCmdTraceRaysKHR-filterCubic-02694
  Any VkImageView being sampled with VK_FILTER_CUBIC_EXT as a result of this command **must** have a VkImageViewType and format that supports cubic filtering, as specified by VkFilterCubicImageViewImageFormatPropertiesEXT::filterCubic returned by vkGetPhysicalDeviceImageFormatProperties2
- VUID-vkCmdTraceRaysKHR-filterCubicMinmax-02695
  Any VkImageView being sampled with VK_FILTER_CUBIC_EXT with a reduction mode of either VK_SAMPLER_REDUCTION_MODE_MIN or VK_SAMPLER_REDUCTION_MODE_MAX as a result of this command **must** have a VkImageViewType and format that supports cubic filtering together with minmax filtering, as specified by VkFilterCubicImageViewImageFormatPropertiesEXT::filterCubicMinmax returned by vkGetPhysicalDeviceImageFormatProperties2
- VUID-vkCmdTraceRaysKHR-cubicRangeClamp-09212
  If the cubicRangeClamp feature is not enabled, then any VkImageView being sampled with VK_FILTER_CUBIC_EXT as a result of this command **must** not have a VkSamplerReductionModeCreateInfo::reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE_RANGECLAMP_QCOM
- VUID-vkCmdTraceRaysKHR-reductionMode-09213
  Any VkImageView being sampled with a VkSamplerReductionModeCreateInfo::reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE_RANGECLAMP_QCOM as a result of this command **must** sample with VK_FILTER_CUBIC_EXT
- VUID-vkCmdTraceRaysKHR-selectableCubicWeights-09214
  If the selectableCubicWeights feature is not enabled, then any VkImageView being sampled with VK_FILTER_CUBIC_EXT as a result of this command **must** have VkSamplerCubicWeightsCreateInfoQCOM::cubicWeights equal to VK_CUBIC_FILTER_WEIGHTS_CATMULL_ROM_QCOM
- VUID-vkCmdTraceRaysKHR-flags-02696
  Any VkImage created with a VkImageCreateInfo::flags containing VK_IMAGE_CREATE_CORNER_SAMPLED_BIT_NV sampled as a result of this command **must** only be sampled using a VkSamplerAddressMode of VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_EDGE
- VUID-vkCmdTraceRaysKHR-OpTypeImage-07027
  For any VkImageView being written as a storage image where the image format field of the OpTypeImage is Unknown, the view's format features **must** contain VK_FORMAT_FEATURE_2_STORAGE_WRITE_WITHOUT_FORMAT_BIT
- VUID-vkCmdTraceRaysKHR-OpTypeImage-07028
  For any VkImageView being read as a storage image where the image format field of the OpTypeImage is Unknown, the view's format features **must** contain VK_FORMAT_FEATURE_2_STORAGE_READ_WITHOUT_FORMAT_BIT
- VUID-vkCmdTraceRaysKHR-OpTypeImage-07029
  For any VkBufferView being written as a storage texel buffer where the image format field of the OpTypeImage is Unknown, the view's buffer features **must** contain VK_FORMAT_FEATURE_2_STORAGE_WRITE_WITHOUT_FORMAT_BIT
- VUID-vkCmdTraceRaysKHR-OpTypeImage-07030

Any VkBufferView being read as a storage texel buffer where the image format field of the `OpTypeImage` is `Unknown` then the view's buffer features **must** contain `VK_FORMAT_FEATURE_2_STORAGE_READ_WITHOUT_FORMAT_BIT`

- VUID-vkCmdTraceRaysKHR-None-08600
  For each set *n* that is statically used by a bound shader, a descriptor set **must** have been bound to *n* at the same pipeline bind point, with a VkPipelineLayout that is compatible for set *n*, with the VkPipelineLayout used to create the current VkPipeline or the VkDescriptorSetLayout array used to create the current VkShaderEXT , as described in Pipeline Layout Compatibility

- VUID-vkCmdTraceRaysKHR-None-08601
  For each push constant that is statically used by a bound shader, a push constant value **must** have been set for the same pipeline bind point, with a VkPipelineLayout that is compatible for push constants, with the VkPipelineLayout used to create the current VkPipeline or the VkDescriptorSetLayout array used to create the current VkShaderEXT , as described in Pipeline Layout Compatibility

- VUID-vkCmdTraceRaysKHR-None-10068
  For each array of resources that is used by a bound shader, the indices used to access members of the array **must** be less than the descriptor count for the identified binding in the descriptor sets used by this command

- VUID-vkCmdTraceRaysKHR-maintenance4-08602
  If the `maintenance4` feature is not enabled, then for each push constant that is statically used by a bound shader, a push constant value **must** have been set for the same pipeline bind point, with a VkPipelineLayout that is compatible for push constants, with the VkPipelineLayout used to create the current VkPipeline or the VkDescriptorSetLayout and VkPushConstantRange arrays used to create the current VkShaderEXT , as described in Pipeline Layout Compatibility

- VUID-vkCmdTraceRaysKHR-None-08114
  Descriptors in each bound descriptor set, specified via vkCmdBindDescriptorSets, **must** be valid as described by descriptor validity if they are statically used by the VkPipeline bound to the pipeline bind point used by this command and the bound VkPipeline was not created with `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`

- VUID-vkCmdTraceRaysKHR-None-08115
  If the descriptors used by the VkPipeline bound to the pipeline bind point were specified via vkCmdBindDescriptorSets, the bound VkPipeline **must** have been created without `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`

- VUID-vkCmdTraceRaysKHR-None-08116
  Descriptors in bound descriptor buffers, specified via vkCmdSetDescriptorBufferOffsetsEXT, **must** be valid if they are dynamically used by the VkPipeline bound to the pipeline bind point used by this command and the bound VkPipeline was created with `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`

- VUID-vkCmdTraceRaysKHR-None-08604
  Descriptors in bound descriptor buffers, specified via vkCmdSetDescriptorBufferOffsetsEXT, **must** be valid if they are dynamically used by any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command

- VUID-vkCmdTraceRaysKHR-None-08117
  If the descriptors used by the VkPipeline bound to the pipeline bind point were specified via vkCmdSetDescriptorBufferOffsetsEXT, the bound VkPipeline **must** have been created with VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT
- VUID-vkCmdTraceRaysKHR-None-08119
  If a descriptor is dynamically used with a VkPipeline created with VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT, the descriptor memory **must** be resident
- VUID-vkCmdTraceRaysKHR-None-08605
  If a descriptor is dynamically used with a VkShaderEXT created with a VkDescriptorSetLayout that was created with VK_DESCRIPTOR_SET_LAYOUT_CREATE_DESCRIPTOR_BUFFER_BIT_EXT, the descriptor memory **must** be resident
- VUID-vkCmdTraceRaysKHR-None-08606
  If the shaderObject feature is not enabled, a valid pipeline **must** be bound to the pipeline bind point used by this command
- VUID-vkCmdTraceRaysKHR-None-08608
  If a pipeline is bound to the pipeline bind point used by this command, there **must** not have been any calls to dynamic state setting commands for any state specified statically in the VkPipeline object bound to the pipeline bind point used by this command, since that pipeline was bound
- VUID-vkCmdTraceRaysKHR-None-08609
  If the VkPipeline object bound to the pipeline bind point used by this command or any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a VkSampler object that uses unnormalized coordinates, that sampler **must** not be used to sample from any VkImage with a VkImageView of the type VK_IMAGE_VIEW_TYPE_3D, VK_IMAGE_VIEW_TYPE_CUBE, VK_IMAGE_VIEW_TYPE_1D_ARRAY, VK_IMAGE_VIEW_TYPE_2D_ARRAY or VK_IMAGE_VIEW_TYPE_CUBE_ARRAY, in any shader stage
- VUID-vkCmdTraceRaysKHR-None-08610
  If the VkPipeline object bound to the pipeline bind point used by this command or any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a VkSampler object that uses unnormalized coordinates, that sampler **must** not be used with any of the SPIR-V OpImageSample* or OpImageSparseSample* instructions with ImplicitLod, Dref or Proj in their name, in any shader stage
- VUID-vkCmdTraceRaysKHR-None-08611
  If the VkPipeline object bound to the pipeline bind point used by this command or any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a VkSampler object that uses unnormalized coordinates, that sampler **must** not be used with any of the SPIR-V OpImageSample* or OpImageSparseSample* instructions that includes a LOD bias or any offset values, in any shader stage
- VUID-vkCmdTraceRaysKHR-None-08607
  If the shaderObject is enabled, either a valid pipeline **must** be bound to the pipeline bind point used by this command, or a valid combination of valid and VK_NULL_HANDLE shader objects **must** be bound to every supported shader stage corresponding to the pipeline bind point used by this command

- VUID-vkCmdTraceRaysKHR-uniformBuffers-06935
  If any stage of the VkPipeline object bound to the pipeline bind point used by this command accesses a uniform buffer, and that stage was created without enabling either `VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_EXT` or `VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_2_EXT` for `uniformBuffers`, and the `robustBufferAccess` feature is not enabled, that stage **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point
- VUID-vkCmdTraceRaysKHR-None-08612
  If the `robustBufferAccess` feature is not enabled, and any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a uniform buffer, it **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point
- VUID-vkCmdTraceRaysKHR-storageBuffers-06936
  If any stage of the VkPipeline object bound to the pipeline bind point used by this command accesses a storage buffer, and that stage was created without enabling either `VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_EXT` or `VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_2_EXT` for `storageBuffers`, and the `robustBufferAccess` feature is not enabled, that stage **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point
- VUID-vkCmdTraceRaysKHR-None-08613
  If the `robustBufferAccess` feature is not enabled, and any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a storage buffer, it **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point
- VUID-vkCmdTraceRaysKHR-commandBuffer-02707
  If `commandBuffer` is an unprotected command buffer and `protectedNoFault` is not supported, any resource accessed by bound shaders **must** not be a protected resource
- VUID-vkCmdTraceRaysKHR-None-06550
  If a bound shader accesses a VkSampler or VkImageView object that enables sampler $Y'C_BC_R$ conversion, that object **must** only be used with `OpImageSample*` or `OpImageSparseSample*` instructions
- VUID-vkCmdTraceRaysKHR-ConstOffset-06551
  If a bound shader accesses a VkSampler or VkImageView object that enables sampler $Y'C_BC_R$ conversion, that object **must** not use the `ConstOffset` and `Offset` operands
- VUID-vkCmdTraceRaysKHR-viewType-07752
  If a VkImageView is accessed as a result of this command, then the image view's `viewType` **must** match the `Dim` operand of the `OpTypeImage` as described in Compatibility Between SPIR-V Image Dimensions and Vulkan ImageView Types
- VUID-vkCmdTraceRaysKHR-format-07753
  If a VkImageView is accessed as a result of this command, then the numeric type of the image view's `format` and the `Sampled Type` operand of the `OpTypeImage` **must** match
- VUID-vkCmdTraceRaysKHR-OpImageWrite-08795

If a VkImageView created with a format other than VK_FORMAT_A8_UNORM_KHR is accessed using OpImageWrite as a result of this command, then the Type of the Texel operand of that instruction **must** have at least as many components as the image view's format

- VUID-vkCmdTraceRaysKHR-OpImageWrite-08796
  If a VkImageView created with the format VK_FORMAT_A8_UNORM_KHR is accessed using OpImageWrite as a result of this command, then the Type of the Texel operand of that instruction **must** have four components
- VUID-vkCmdTraceRaysKHR-OpImageWrite-04469
  If a VkBufferView is accessed using OpImageWrite as a result of this command, then the Type of the Texel operand of that instruction **must** have at least as many components as the buffer view's format
- VUID-vkCmdTraceRaysKHR-SampledType-04470
  If a VkImageView with a VkFormat that has a 64-bit component width is accessed as a result of this command, the SampledType of the OpTypeImage operand of that instruction **must** have a Width of 64
- VUID-vkCmdTraceRaysKHR-SampledType-04471
  If a VkImageView with a VkFormat that has a component width less than 64-bit is accessed as a result of this command, the SampledType of the OpTypeImage operand of that instruction **must** have a Width of 32
- VUID-vkCmdTraceRaysKHR-SampledType-04472
  If a VkBufferView with a VkFormat that has a 64-bit component width is accessed as a result of this command, the SampledType of the OpTypeImage operand of that instruction **must** have a Width of 64
- VUID-vkCmdTraceRaysKHR-SampledType-04473
  If a VkBufferView with a VkFormat that has a component width less than 64-bit is accessed as a result of this command, the SampledType of the OpTypeImage operand of that instruction **must** have a Width of 32
- VUID-vkCmdTraceRaysKHR-sparseImageInt64Atomics-04474
  If the sparseImageInt64Atomics feature is not enabled, VkImage objects created with the VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT flag **must** not be accessed by atomic instructions through an OpTypeImage with a SampledType with a Width of 64 by this command
- VUID-vkCmdTraceRaysKHR-sparseImageInt64Atomics-04475
  If the sparseImageInt64Atomics feature is not enabled, VkBuffer objects created with the VK_BUFFER_CREATE_SPARSE_RESIDENCY_BIT flag **must** not be accessed by atomic instructions through an OpTypeImage with a SampledType with a Width of 64 by this command
- VUID-vkCmdTraceRaysKHR-OpImageWeightedSampleQCOM-06971
  If OpImageWeightedSampleQCOM is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_WEIGHT_SAMPLED_IMAGE_BIT_QCOM
- VUID-vkCmdTraceRaysKHR-OpImageWeightedSampleQCOM-06972
  If OpImageWeightedSampleQCOM uses a VkImageView as a sample weight image as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_WEIGHT_IMAGE_BIT_QCOM

- VUID-vkCmdTraceRaysKHR-OpImageBoxFilterQCOM-06973
  If `OpImageBoxFilterQCOM` is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_2_BOX_FILTER_SAMPLED_BIT_QCOM`
- VUID-vkCmdTraceRaysKHR-OpImageBlockMatchSSDQCOM-06974
  If `OpImageBlockMatchSSDQCOM` is used to read from an VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM`
- VUID-vkCmdTraceRaysKHR-OpImageBlockMatchSADQCOM-06975
  If `OpImageBlockMatchSADQCOM` is used to read from an VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM`
- VUID-vkCmdTraceRaysKHR-OpImageBlockMatchSADQCOM-06976
  If `OpImageBlockMatchSADQCOM` or OpImageBlockMatchSSDQCOM is used to read from a reference image as result of this command, then the specified reference coordinates **must** not fail integer texel coordinate validation
- VUID-vkCmdTraceRaysKHR-OpImageWeightedSampleQCOM-06977
  If `OpImageWeightedSampleQCOM`, `OpImageBoxFilterQCOM`, `OpImageBlockMatchWindowSSDQCOM`, `OpImageBlockMatchWindowSADQCOM`, `OpImageBlockMatchGatherSSDQCOM`, `OpImageBlockMatchGatherSADQCOM`, `OpImageBlockMatchSSDQCOM`, or `OpImageBlockMatchSADQCOM` uses a VkSampler as a result of this command, then the sampler **must** have been created with `VK_SAMPLER_CREATE_IMAGE_PROCESSING_BIT_QCOM`
- VUID-vkCmdTraceRaysKHR-OpImageWeightedSampleQCOM-06978
  If any command other than `OpImageWeightedSampleQCOM`, `OpImageBoxFilterQCOM`, `OpImageBlockMatchWindowSSDQCOM`, `OpImageBlockMatchWindowSADQCOM`, `OpImageBlockMatchGatherSSDQCOM`, `OpImageBlockMatchGatherSADQCOM`, `OpImageBlockMatchSSDQCOM`, or `OpImageBlockMatchSADQCOM` uses a VkSampler as a result of this command, then the sampler **must** not have been created with `VK_SAMPLER_CREATE_IMAGE_PROCESSING_BIT_QCOM`
- VUID-vkCmdTraceRaysKHR-OpImageBlockMatchWindow-09215
  If a `OpImageBlockMatchWindow*QCOM` or `OpImageBlockMatchGather*QCOM` instruction is used to read from an VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM`
- VUID-vkCmdTraceRaysKHR-OpImageBlockMatchWindow-09216
  If a `OpImageBlockMatchWindow*QCOM` or `OpImageBlockMatchGather*QCOM` instruction is used to read from an VkImageView as a result of this command, then the image view's format **must** be a single-component format
- VUID-vkCmdTraceRaysKHR-OpImageBlockMatchWindow-09217
  If a `OpImageBlockMatchWindow*QCOM` or `OpImageBlockMatchGather*QCOM` read from a reference image as result of this command, then the specified reference coordinates **must** not fail integer texel coordinate validation
- VUID-vkCmdTraceRaysKHR-None-07288
  Any shader invocation executed by this command **must** terminate
- VUID-vkCmdTraceRaysKHR-None-09600

If a descriptor with type equal to any of VK_DESCRIPTOR_TYPE_SAMPLE_WEIGHT_IMAGE_QCOM, VK_DESCRIPTOR_TYPE_BLOCK_MATCH_IMAGE_QCOM, VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE, VK_DESCRIPTOR_TYPE_STORAGE_IMAGE, or VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT is accessed as a result of this command, the image subresource identified by that descriptor **must** be in the image layout identified when the descriptor was written

- VUID-vkCmdTraceRaysKHR-None-03429
  Any shader group handle referenced by this call **must** have been queried from the currently bound ray tracing pipeline
- VUID-vkCmdTraceRaysKHR-None-09458
  If the bound ray tracing pipeline state was created with the VK_DYNAMIC_STATE_RAY_TRACING_PIPELINE_STACK_SIZE_KHR dynamic state enabled then vkCmdSetRayTracingPipelineStackSizeKHR **must** have been called in the current command buffer prior to this trace command
- VUID-vkCmdTraceRaysKHR-maxPipelineRayRecursionDepth-03679
  This command **must** not cause a shader call instruction to be executed from a shader invocation with a recursion depth greater than the value of maxPipelineRayRecursionDepth used to create the bound ray tracing pipeline
- VUID-vkCmdTraceRaysKHR-commandBuffer-03635
  commandBuffer **must** not be a protected command buffer
- VUID-vkCmdTraceRaysKHR-size-04023
  The size member of pRayGenShaderBindingTable **must** be equal to its stride member
- VUID-vkCmdTraceRaysKHR-pRayGenShaderBindingTable-03680
  If the buffer from which pRayGenShaderBindingTable->deviceAddress was queried is non-sparse then it **must** be bound completely and contiguously to a single VkDeviceMemory object
- VUID-vkCmdTraceRaysKHR-pRayGenShaderBindingTable-03681
  The buffer from which the pRayGenShaderBindingTable->deviceAddress is queried **must** have been created with the VK_BUFFER_USAGE_SHADER_BINDING_TABLE_BIT_KHR usage flag
- VUID-vkCmdTraceRaysKHR-pRayGenShaderBindingTable-03682
  pRayGenShaderBindingTable->deviceAddress **must** be a multiple of VkPhysicalDeviceRayTracingPipelinePropertiesKHR::shaderGroupBaseAlignment
- VUID-vkCmdTraceRaysKHR-pMissShaderBindingTable-03683
  If the buffer from which pMissShaderBindingTable->deviceAddress was queried is non-sparse then it **must** be bound completely and contiguously to a single VkDeviceMemory object
- VUID-vkCmdTraceRaysKHR-pMissShaderBindingTable-03684
  The buffer from which the pMissShaderBindingTable->deviceAddress is queried **must** have been created with the VK_BUFFER_USAGE_SHADER_BINDING_TABLE_BIT_KHR usage flag
- VUID-vkCmdTraceRaysKHR-pMissShaderBindingTable-03685
  pMissShaderBindingTable->deviceAddress **must** be a multiple of VkPhysicalDeviceRayTracingPipelinePropertiesKHR::shaderGroupBaseAlignment
- VUID-vkCmdTraceRaysKHR-stride-03686

pMissShaderBindingTable->stride **must** be a multiple of VkPhysicalDeviceRayTracingPipelinePropertiesKHR::shaderGroupHandleAlignment

- VUID-vkCmdTraceRaysKHR-stride-04029
  pMissShaderBindingTable->stride **must** be less than or equal to VkPhysicalDeviceRayTracingPipelinePropertiesKHR::maxShaderGroupStride
- VUID-vkCmdTraceRaysKHR-pHitShaderBindingTable-03687
  If the buffer from which pHitShaderBindingTable->deviceAddress was queried is non-sparse then it **must** be bound completely and contiguously to a single VkDeviceMemory object
- VUID-vkCmdTraceRaysKHR-pHitShaderBindingTable-03688
  The buffer from which the pHitShaderBindingTable->deviceAddress is queried **must** have been created with the VK_BUFFER_USAGE_SHADER_BINDING_TABLE_BIT_KHR usage flag
- VUID-vkCmdTraceRaysKHR-pHitShaderBindingTable-03689
  pHitShaderBindingTable->deviceAddress **must** be a multiple of VkPhysicalDeviceRayTracingPipelinePropertiesKHR::shaderGroupBaseAlignment
- VUID-vkCmdTraceRaysKHR-stride-03690
  pHitShaderBindingTable->stride **must** be a multiple of VkPhysicalDeviceRayTracingPipelinePropertiesKHR::shaderGroupHandleAlignment
- VUID-vkCmdTraceRaysKHR-stride-04035
  pHitShaderBindingTable->stride **must** be less than or equal to VkPhysicalDeviceRayTracingPipelinePropertiesKHR::maxShaderGroupStride
- VUID-vkCmdTraceRaysKHR-pCallableShaderBindingTable-03691
  If the buffer from which pCallableShaderBindingTable->deviceAddress was queried is non-sparse then it **must** be bound completely and contiguously to a single VkDeviceMemory object
- VUID-vkCmdTraceRaysKHR-pCallableShaderBindingTable-03692
  The buffer from which the pCallableShaderBindingTable->deviceAddress is queried **must** have been created with the VK_BUFFER_USAGE_SHADER_BINDING_TABLE_BIT_KHR usage flag
- VUID-vkCmdTraceRaysKHR-pCallableShaderBindingTable-03693
  pCallableShaderBindingTable->deviceAddress **must** be a multiple of VkPhysicalDeviceRayTracingPipelinePropertiesKHR::shaderGroupBaseAlignment
- VUID-vkCmdTraceRaysKHR-stride-03694
  pCallableShaderBindingTable->stride **must** be a multiple of VkPhysicalDeviceRayTracingPipelinePropertiesKHR::shaderGroupHandleAlignment
- VUID-vkCmdTraceRaysKHR-stride-04041
  pCallableShaderBindingTable->stride **must** be less than or equal to VkPhysicalDeviceRayTracingPipelinePropertiesKHR::maxShaderGroupStride
- VUID-vkCmdTraceRaysKHR-flags-03696
  If the currently bound ray tracing pipeline was created with flags that included VK_PIPELINE_CREATE_RAY_TRACING_NO_NULL_CLOSEST_HIT_SHADERS_BIT_KHR, pHitShaderBindingTable->deviceAddress **must** not be zero
- VUID-vkCmdTraceRaysKHR-flags-03697
  If the currently bound ray tracing pipeline was created with flags that included

VK_PIPELINE_CREATE_RAY_TRACING_NO_NULL_INTERSECTION_SHADERS_BIT_KHR, pHitShaderBindingTable->deviceAddress **must** not be zero

- VUID-vkCmdTraceRaysKHR-flags-03511
  If the currently bound ray tracing pipeline was created with flags that included VK_PIPELINE_CREATE_RAY_TRACING_NO_NULL_MISS_SHADERS_BIT_KHR, the shader group handle identified by pMissShaderBindingTable->deviceAddress **must** not be set to zero
- VUID-vkCmdTraceRaysKHR-flags-03512
  If the currently bound ray tracing pipeline was created with flags that included VK_PIPELINE_CREATE_RAY_TRACING_NO_NULL_ANY_HIT_SHADERS_BIT_KHR, entries in the table identified by pHitShaderBindingTable->deviceAddress accessed as a result of this command in order to execute an any-hit shader **must** not be set to zero
- VUID-vkCmdTraceRaysKHR-flags-03513
  If the currently bound ray tracing pipeline was created with flags that included VK_PIPELINE_CREATE_RAY_TRACING_NO_NULL_CLOSEST_HIT_SHADERS_BIT_KHR, entries in the table identified by pHitShaderBindingTable->deviceAddress accessed as a result of this command in order to execute a closest hit shader **must** not be set to zero
- VUID-vkCmdTraceRaysKHR-flags-03514
  If the currently bound ray tracing pipeline was created with flags that included VK_PIPELINE_CREATE_RAY_TRACING_NO_NULL_INTERSECTION_SHADERS_BIT_KHR, entries in the table identified by pHitShaderBindingTable->deviceAddress accessed as a result of this command in order to execute an intersection shader **must** not be set to zero
- VUID-vkCmdTraceRaysKHR-pHitShaderBindingTable-04735
  Any non-zero hit shader group entries in the table identified by pHitShaderBindingTable->deviceAddress accessed by this call from a geometry with a geometryType of VK_GEOMETRY_TYPE_TRIANGLES_KHR **must** have been created with VK_RAY_TRACING_SHADER_GROUP_TYPE_TRIANGLES_HIT_GROUP_KHR
- VUID-vkCmdTraceRaysKHR-pHitShaderBindingTable-04736
  Any non-zero hit shader group entries in the table identified by pHitShaderBindingTable->deviceAddress accessed by this call from a geometry with a geometryType of VK_GEOMETRY_TYPE_AABBS_KHR **must** have been created with VK_RAY_TRACING_SHADER_GROUP_TYPE_PROCEDURAL_HIT_GROUP_KHR
- VUID-vkCmdTraceRaysKHR-width-03638
  width **must** be less than or equal to VkPhysicalDeviceLimits::maxComputeWorkGroupCount[0] × VkPhysicalDeviceLimits::maxComputeWorkGroupSize[0]
- VUID-vkCmdTraceRaysKHR-height-03639
  height **must** be less than or equal to VkPhysicalDeviceLimits::maxComputeWorkGroupCount[1] × VkPhysicalDeviceLimits::maxComputeWorkGroupSize[1]
- VUID-vkCmdTraceRaysKHR-depth-03640
  depth **must** be less than or equal to VkPhysicalDeviceLimits::maxComputeWorkGroupCount[2] × VkPhysicalDeviceLimits::maxComputeWorkGroupSize[2]
- VUID-vkCmdTraceRaysKHR-width-03641
  width × height × depth **must** be less than or equal to VkPhysicalDeviceRayTracingPipelinePropertiesKHR::maxRayDispatchInvocationCount

## Valid Usage (Implicit)

- VUID-vkCmdTraceRaysKHR-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdTraceRaysKHR-pRaygenShaderBindingTable-parameter
  `pRaygenShaderBindingTable` **must** be a valid pointer to a valid VkStridedDeviceAddressRegionKHR structure
- VUID-vkCmdTraceRaysKHR-pMissShaderBindingTable-parameter
  `pMissShaderBindingTable` **must** be a valid pointer to a valid VkStridedDeviceAddressRegionKHR structure
- VUID-vkCmdTraceRaysKHR-pHitShaderBindingTable-parameter
  `pHitShaderBindingTable` **must** be a valid pointer to a valid VkStridedDeviceAddressRegionKHR structure
- VUID-vkCmdTraceRaysKHR-pCallableShaderBindingTable-parameter
  `pCallableShaderBindingTable` **must** be a valid pointer to a valid VkStridedDeviceAddressRegionKHR structure
- VUID-vkCmdTraceRaysKHR-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdTraceRaysKHR-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support compute operations
- VUID-vkCmdTraceRaysKHR-renderpass
  This command **must** only be called outside of a render pass instance
- VUID-vkCmdTraceRaysKHR-videocoding
  This command **must** only be called outside of a video coding scope

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Outside | Outside | Compute | Action |

When invocation mask image usage is enabled in the bound ray tracing pipeline, the pipeline uses

an invocation mask image specified by the command:

```
// Provided by VK_HUAWEI_invocation_mask
void vkCmdBindInvocationMaskHUAWEI(
    VkCommandBuffer                             commandBuffer,
    VkImageView                                 imageView,
    VkImageLayout                               imageLayout);
```

- `commandBuffer` is the command buffer into which the command will be recorded
- `imageView` is an image view handle specifying the invocation mask image `imageView` **may** be set to VK_NULL_HANDLE, which is equivalent to specifying a view of an image filled with ones value.
- `imageLayout` is the layout that the image subresources accessible from `imageView` will be in when the invocation mask image is accessed

## Valid Usage

- VUID-vkCmdBindInvocationMaskHUAWEI-None-04976
  The `invocationMask` feature **must** be enabled
- VUID-vkCmdBindInvocationMaskHUAWEI-imageView-04977
  If `imageView` is not VK_NULL_HANDLE, it **must** be a valid VkImageView handle of type `VK_IMAGE_VIEW_TYPE_2D`
- VUID-vkCmdBindInvocationMaskHUAWEI-imageView-04978
  If `imageView` is not VK_NULL_HANDLE, it **must** have a format of `VK_FORMAT_R8_UINT`
- VUID-vkCmdBindInvocationMaskHUAWEI-imageView-04979
  If `imageView` is not VK_NULL_HANDLE, it **must** have been created with `VK_IMAGE_USAGE_INVOCATION_MASK_BIT_HUAWEI` set
- VUID-vkCmdBindInvocationMaskHUAWEI-imageView-04980
  If `imageView` is not VK_NULL_HANDLE, `imageLayout` **must** be `VK_IMAGE_LAYOUT_GENERAL`
- VUID-vkCmdBindInvocationMaskHUAWEI-width-04981
  Thread mask image resolution **must** match the `width` and `height` in vkCmdTraceRaysKHR
- VUID-vkCmdBindInvocationMaskHUAWEI-None-04982
  Each element in the invocation mask image **must** have the value `0` or `1`. The value 1 means the invocation is active
- VUID-vkCmdBindInvocationMaskHUAWEI-depth-04983
  `depth` in vkCmdTraceRaysKHR **must** be 1

## Valid Usage (Implicit)

- VUID-vkCmdBindInvocationMaskHUAWEI-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdBindInvocationMaskHUAWEI-imageView-parameter
  If `imageView` is not VK_NULL_HANDLE, `imageView` **must** be a valid VkImageView handle

- VUID-vkCmdBindInvocationMaskHUAWEI-imageLayout-parameter
  `imageLayout` **must** be a valid VkImageLayout value
- VUID-vkCmdBindInvocationMaskHUAWEI-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdBindInvocationMaskHUAWEI-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support compute operations
- VUID-vkCmdBindInvocationMaskHUAWEI-renderpass
  This command **must** only be called outside of a render pass instance
- VUID-vkCmdBindInvocationMaskHUAWEI-videocoding
  This command **must** only be called outside of a video coding scope
- VUID-vkCmdBindInvocationMaskHUAWEI-commonparent
  Both of `commandBuffer`, and `imageView` that are valid handles of non-ignored parameters **must** have been created, allocated, or retrieved from the same VkDevice

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Outside | Outside | Compute | State |

To dispatch ray tracing, with some parameters sourced on the device, use:

```
// Provided by VK_KHR_ray_tracing_pipeline
void vkCmdTraceRaysIndirectKHR(
    VkCommandBuffer                             commandBuffer,
    const VkStridedDeviceAddressRegionKHR*      pRaygenShaderBindingTable,
    const VkStridedDeviceAddressRegionKHR*      pMissShaderBindingTable,
    const VkStridedDeviceAddressRegionKHR*      pHitShaderBindingTable,
    const VkStridedDeviceAddressRegionKHR*      pCallableShaderBindingTable,
    VkDeviceAddress                             indirectDeviceAddress);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `pRaygenShaderBindingTable` is a VkStridedDeviceAddressRegionKHR that holds the shader

binding table data for the ray generation shader stage.

- pMissShaderBindingTable is a VkStridedDeviceAddressRegionKHR that holds the shader binding table data for the miss shader stage.
- pHitShaderBindingTable is a VkStridedDeviceAddressRegionKHR that holds the shader binding table data for the hit shader stage.
- pCallableShaderBindingTable is a VkStridedDeviceAddressRegionKHR that holds the shader binding table data for the callable shader stage.
- indirectDeviceAddress is a buffer device address which is a pointer to a VkTraceRaysIndirectCommandKHR structure containing the trace ray parameters.

vkCmdTraceRaysIndirectKHR behaves similarly to vkCmdTraceRaysKHR except that the ray trace query dimensions are read by the device from indirectDeviceAddress during execution.

## Valid Usage

- VUID-vkCmdTraceRaysIndirectKHR-magFilter-04553
  If a VkSampler created with magFilter or minFilter equal to VK_FILTER_LINEAR, reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE, and compareEnable equal to VK_FALSE is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT
- VUID-vkCmdTraceRaysIndirectKHR-magFilter-09598
  If a VkSampler created with magFilter or minFilter equal to VK_FILTER_LINEAR and reductionMode equal to either VK_SAMPLER_REDUCTION_MODE_MIN or VK_SAMPLER_REDUCTION_MODE_MAX is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT
- VUID-vkCmdTraceRaysIndirectKHR-mipmapMode-04770
  If a VkSampler created with mipmapMode equal to VK_SAMPLER_MIPMAP_MODE_LINEAR, reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE, and compareEnable equal to VK_FALSE is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT
- VUID-vkCmdTraceRaysIndirectKHR-mipmapMode-09599
  If a VkSampler created with mipmapMode equal to VK_SAMPLER_MIPMAP_MODE_LINEAR and reductionMode equal to either VK_SAMPLER_REDUCTION_MODE_MIN or VK_SAMPLER_REDUCTION_MODE_MAX is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT
- VUID-vkCmdTraceRaysIndirectKHR-unnormalizedCoordinates-09635
  If a VkSampler created with unnormalizedCoordinates equal to VK_TRUE is used to sample a VkImageView as a result of this command, then the image view's levelCount and layerCount **must** be 1
- VUID-vkCmdTraceRaysIndirectKHR-unnormalizedCoordinates-09636

If a VkSampler created with unnormalizedCoordinates equal to VK_TRUE is used to sample a VkImageView as a result of this command, then the image view's viewType **must** be VK_IMAGE_VIEW_TYPE_1D or VK_IMAGE_VIEW_TYPE_2D

- VUID-vkCmdTraceRaysIndirectKHR-None-06479
  If a VkImageView is sampled with depth comparison, the image view's format features **must** contain VK_FORMAT_FEATURE_2_SAMPLED_IMAGE_DEPTH_COMPARISON_BIT
- VUID-vkCmdTraceRaysIndirectKHR-None-02691
  If a VkImageView is accessed using atomic operations as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_STORAGE_IMAGE_ATOMIC_BIT
- VUID-vkCmdTraceRaysIndirectKHR-None-07888
  If a VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER descriptor is accessed using atomic operations as a result of this command, then the storage texel buffer's format features **must** contain VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_ATOMIC_BIT
- VUID-vkCmdTraceRaysIndirectKHR-None-02692
  If a VkImageView is sampled with VK_FILTER_CUBIC_EXT as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_CUBIC_BIT_EXT
- VUID-vkCmdTraceRaysIndirectKHR-None-02693
  If the VK_EXT_filter_cubic extension is not enabled and any VkImageView is sampled with VK_FILTER_CUBIC_EXT as a result of this command, it **must** not have a VkImageViewType of VK_IMAGE_VIEW_TYPE_3D, VK_IMAGE_VIEW_TYPE_CUBE, or VK_IMAGE_VIEW_TYPE_CUBE_ARRAY
- VUID-vkCmdTraceRaysIndirectKHR-filterCubic-02694
  Any VkImageView being sampled with VK_FILTER_CUBIC_EXT as a result of this command **must** have a VkImageViewType and format that supports cubic filtering, as specified by VkFilterCubicImageViewImageFormatPropertiesEXT::filterCubic returned by vkGetPhysicalDeviceImageFormatProperties2
- VUID-vkCmdTraceRaysIndirectKHR-filterCubicMinmax-02695
  Any VkImageView being sampled with VK_FILTER_CUBIC_EXT with a reduction mode of either VK_SAMPLER_REDUCTION_MODE_MIN or VK_SAMPLER_REDUCTION_MODE_MAX as a result of this command **must** have a VkImageViewType and format that supports cubic filtering together with minmax filtering, as specified by VkFilterCubicImageViewImageFormatPropertiesEXT::filterCubicMinmax returned by vkGetPhysicalDeviceImageFormatProperties2
- VUID-vkCmdTraceRaysIndirectKHR-cubicRangeClamp-09212
  If the cubicRangeClamp feature is not enabled, then any VkImageView being sampled with VK_FILTER_CUBIC_EXT as a result of this command **must** not have a VkSamplerReductionModeCreateInfo::reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE_RANGECLAMP_QCOM
- VUID-vkCmdTraceRaysIndirectKHR-reductionMode-09213
  Any VkImageView being sampled with a VkSamplerReductionModeCreateInfo::reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE_RANGECLAMP_QCOM as a result of this command **must** sample with VK_FILTER_CUBIC_EXT
- VUID-vkCmdTraceRaysIndirectKHR-selectableCubicWeights-09214

If the selectableCubicWeights feature is not enabled, then any VkImageView being sampled with VK_FILTER_CUBIC_EXT as a result of this command **must** have VkSamplerCubicWeightsCreateInfoQCOM::cubicWeights equal to VK_CUBIC_FILTER_WEIGHTS_CATMULL_ROM_QCOM

- VUID-vkCmdTraceRaysIndirectKHR-flags-02696
  Any VkImage created with a VkImageCreateInfo::flags containing VK_IMAGE_CREATE_CORNER_SAMPLED_BIT_NV sampled as a result of this command **must** only be sampled using a VkSamplerAddressMode of VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_EDGE
- VUID-vkCmdTraceRaysIndirectKHR-OpTypeImage-07027
  For any VkImageView being written as a storage image where the image format field of the OpTypeImage is Unknown, the view's format features **must** contain VK_FORMAT_FEATURE_2_STORAGE_WRITE_WITHOUT_FORMAT_BIT
- VUID-vkCmdTraceRaysIndirectKHR-OpTypeImage-07028
  For any VkImageView being read as a storage image where the image format field of the OpTypeImage is Unknown, the view's format features **must** contain VK_FORMAT_FEATURE_2_STORAGE_READ_WITHOUT_FORMAT_BIT
- VUID-vkCmdTraceRaysIndirectKHR-OpTypeImage-07029
  For any VkBufferView being written as a storage texel buffer where the image format field of the OpTypeImage is Unknown, the view's buffer features **must** contain VK_FORMAT_FEATURE_2_STORAGE_WRITE_WITHOUT_FORMAT_BIT
- VUID-vkCmdTraceRaysIndirectKHR-OpTypeImage-07030
  Any VkBufferView being read as a storage texel buffer where the image format field of the OpTypeImage is Unknown then the view's buffer features **must** contain VK_FORMAT_FEATURE_2_STORAGE_READ_WITHOUT_FORMAT_BIT
- VUID-vkCmdTraceRaysIndirectKHR-None-08600
  For each set *n* that is statically used by a bound shader, a descriptor set **must** have been bound to *n* at the same pipeline bind point, with a VkPipelineLayout that is compatible for set *n*, with the VkPipelineLayout used to create the current VkPipeline or the VkDescriptorSetLayout array used to create the current VkShaderEXT , as described in Pipeline Layout Compatibility
- VUID-vkCmdTraceRaysIndirectKHR-None-08601
  For each push constant that is statically used by a bound shader, a push constant value **must** have been set for the same pipeline bind point, with a VkPipelineLayout that is compatible for push constants, with the VkPipelineLayout used to create the current VkPipeline or the VkDescriptorSetLayout array used to create the current VkShaderEXT , as described in Pipeline Layout Compatibility
- VUID-vkCmdTraceRaysIndirectKHR-None-10068
  For each array of resources that is used by a bound shader, the indices used to access members of the array **must** be less than the descriptor count for the identified binding in the descriptor sets used by this command
- VUID-vkCmdTraceRaysIndirectKHR-maintenance4-08602
  If the maintenance4 feature is not enabled, then for each push constant that is statically used by a bound shader, a push constant value **must** have been set for the same pipeline bind point, with a VkPipelineLayout that is compatible for push constants, with the

VkPipelineLayout used to create the current VkPipeline or the VkDescriptorSetLayout and VkPushConstantRange arrays used to create the current VkShaderEXT , as described in Pipeline Layout Compatibility

- VUID-vkCmdTraceRaysIndirectKHR-None-08114
  Descriptors in each bound descriptor set, specified via vkCmdBindDescriptorSets, **must** be valid as described by descriptor validity if they are statically used by the VkPipeline bound to the pipeline bind point used by this command and the bound VkPipeline was not created with `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`
- VUID-vkCmdTraceRaysIndirectKHR-None-08115
  If the descriptors used by the VkPipeline bound to the pipeline bind point were specified via vkCmdBindDescriptorSets, the bound VkPipeline **must** have been created without `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`
- VUID-vkCmdTraceRaysIndirectKHR-None-08116
  Descriptors in bound descriptor buffers, specified via vkCmdSetDescriptorBufferOffsetsEXT, **must** be valid if they are dynamically used by the VkPipeline bound to the pipeline bind point used by this command and the bound VkPipeline was created with `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`
- VUID-vkCmdTraceRaysIndirectKHR-None-08604
  Descriptors in bound descriptor buffers, specified via vkCmdSetDescriptorBufferOffsetsEXT, **must** be valid if they are dynamically used by any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command
- VUID-vkCmdTraceRaysIndirectKHR-None-08117
  If the descriptors used by the VkPipeline bound to the pipeline bind point were specified via vkCmdSetDescriptorBufferOffsetsEXT, the bound VkPipeline **must** have been created with `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`
- VUID-vkCmdTraceRaysIndirectKHR-None-08119
  If a descriptor is dynamically used with a VkPipeline created with `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`, the descriptor memory **must** be resident
- VUID-vkCmdTraceRaysIndirectKHR-None-08605
  If a descriptor is dynamically used with a VkShaderEXT created with a `VkDescriptorSetLayout` that was created with `VK_DESCRIPTOR_SET_LAYOUT_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`, the descriptor memory **must** be resident
- VUID-vkCmdTraceRaysIndirectKHR-None-08606
  If the `shaderObject` feature is not enabled, a valid pipeline **must** be bound to the pipeline bind point used by this command
- VUID-vkCmdTraceRaysIndirectKHR-None-08608
  If a pipeline is bound to the pipeline bind point used by this command, there **must** not have been any calls to dynamic state setting commands for any state specified statically in the VkPipeline object bound to the pipeline bind point used by this command, since that pipeline was bound
- VUID-vkCmdTraceRaysIndirectKHR-None-08609
  If the VkPipeline object bound to the pipeline bind point used by this command or any

VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a VkSampler object that uses unnormalized coordinates, that sampler **must** not be used to sample from any VkImage with a VkImageView of the type VK_IMAGE_VIEW_TYPE_3D, VK_IMAGE_VIEW_TYPE_CUBE, VK_IMAGE_VIEW_TYPE_1D_ARRAY, VK_IMAGE_VIEW_TYPE_2D_ARRAY or VK_IMAGE_VIEW_TYPE_CUBE_ARRAY, in any shader stage

- VUID-vkCmdTraceRaysIndirectKHR-None-08610
  If the VkPipeline object bound to the pipeline bind point used by this command or any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a VkSampler object that uses unnormalized coordinates, that sampler **must** not be used with any of the SPIR-V OpImageSample* or OpImageSparseSample* instructions with ImplicitLod, Dref or Proj in their name, in any shader stage

- VUID-vkCmdTraceRaysIndirectKHR-None-08611
  If the VkPipeline object bound to the pipeline bind point used by this command or any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a VkSampler object that uses unnormalized coordinates, that sampler **must** not be used with any of the SPIR-V OpImageSample* or OpImageSparseSample* instructions that includes a LOD bias or any offset values, in any shader stage

- VUID-vkCmdTraceRaysIndirectKHR-None-08607
  If the shaderObject is enabled, either a valid pipeline **must** be bound to the pipeline bind point used by this command, or a valid combination of valid and VK_NULL_HANDLE shader objects **must** be bound to every supported shader stage corresponding to the pipeline bind point used by this command

- VUID-vkCmdTraceRaysIndirectKHR-uniformBuffers-06935
  If any stage of the VkPipeline object bound to the pipeline bind point used by this command accesses a uniform buffer, and that stage was created without enabling either VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_EXT or VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_2_EXT for uniformBuffers, and the robustBufferAccess feature is not enabled, that stage **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point

- VUID-vkCmdTraceRaysIndirectKHR-None-08612
  If the robustBufferAccess feature is not enabled, and any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a uniform buffer, it **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point

- VUID-vkCmdTraceRaysIndirectKHR-storageBuffers-06936
  If any stage of the VkPipeline object bound to the pipeline bind point used by this command accesses a storage buffer, and that stage was created without enabling either VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_EXT or VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_2_EXT for storageBuffers, and the robustBufferAccess feature is not enabled, that stage **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point

- VUID-vkCmdTraceRaysIndirectKHR-None-08613
  If the robustBufferAccess feature is not enabled, and any VkShaderEXT bound to a stage

corresponding to the pipeline bind point used by this command accesses a storage buffer, it **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point

- VUID-vkCmdTraceRaysIndirectKHR-commandBuffer-02707
  If `commandBuffer` is an unprotected command buffer and `protectedNoFault` is not supported, any resource accessed by bound shaders **must** not be a protected resource
- VUID-vkCmdTraceRaysIndirectKHR-None-06550
  If a bound shader accesses a VkSampler or VkImageView object that enables sampler $Y'C_BC_R$ conversion, that object **must** only be used with `OpImageSample*` or `OpImageSparseSample*` instructions
- VUID-vkCmdTraceRaysIndirectKHR-ConstOffset-06551
  If a bound shader accesses a VkSampler or VkImageView object that enables sampler $Y'C_BC_R$ conversion, that object **must** not use the `ConstOffset` and `Offset` operands
- VUID-vkCmdTraceRaysIndirectKHR-viewType-07752
  If a VkImageView is accessed as a result of this command, then the image view's `viewType` **must** match the `Dim` operand of the `OpTypeImage` as described in Compatibility Between SPIR-V Image Dimensions and Vulkan ImageView Types
- VUID-vkCmdTraceRaysIndirectKHR-format-07753
  If a VkImageView is accessed as a result of this command, then the numeric type of the image view's `format` and the `Sampled Type` operand of the `OpTypeImage` **must** match
- VUID-vkCmdTraceRaysIndirectKHR-OpImageWrite-08795
  If a VkImageView created with a format other than `VK_FORMAT_A8_UNORM_KHR` is accessed using `OpImageWrite` as a result of this command, then the `Type` of the `Texel` operand of that instruction **must** have at least as many components as the image view's format
- VUID-vkCmdTraceRaysIndirectKHR-OpImageWrite-08796
  If a VkImageView created with the format `VK_FORMAT_A8_UNORM_KHR` is accessed using `OpImageWrite` as a result of this command, then the `Type` of the `Texel` operand of that instruction **must** have four components
- VUID-vkCmdTraceRaysIndirectKHR-OpImageWrite-04469
  If a VkBufferView is accessed using `OpImageWrite` as a result of this command, then the `Type` of the `Texel` operand of that instruction **must** have at least as many components as the buffer view's format
- VUID-vkCmdTraceRaysIndirectKHR-SampledType-04470
  If a VkImageView with a VkFormat that has a 64-bit component width is accessed as a result of this command, the `SampledType` of the `OpTypeImage` operand of that instruction **must** have a `Width` of 64
- VUID-vkCmdTraceRaysIndirectKHR-SampledType-04471
  If a VkImageView with a VkFormat that has a component width less than 64-bit is accessed as a result of this command, the `SampledType` of the `OpTypeImage` operand of that instruction **must** have a `Width` of 32
- VUID-vkCmdTraceRaysIndirectKHR-SampledType-04472
  If a VkBufferView with a VkFormat that has a 64-bit component width is accessed as a result of this command, the `SampledType` of the `OpTypeImage` operand of that instruction

**must** have a `Width` of 64

- VUID-vkCmdTraceRaysIndirectKHR-SampledType-04473
  If a VkBufferView with a VkFormat that has a component width less than 64-bit is accessed as a result of this command, the `SampledType` of the `OpTypeImage` operand of that instruction **must** have a `Width` of 32
- VUID-vkCmdTraceRaysIndirectKHR-sparseImageInt64Atomics-04474
  If the `sparseImageInt64Atomics` feature is not enabled, VkImage objects created with the `VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT` flag **must** not be accessed by atomic instructions through an `OpTypeImage` with a `SampledType` with a `Width` of 64 by this command
- VUID-vkCmdTraceRaysIndirectKHR-sparseImageInt64Atomics-04475
  If the `sparseImageInt64Atomics` feature is not enabled, VkBuffer objects created with the `VK_BUFFER_CREATE_SPARSE_RESIDENCY_BIT` flag **must** not be accessed by atomic instructions through an `OpTypeImage` with a `SampledType` with a `Width` of 64 by this command
- VUID-vkCmdTraceRaysIndirectKHR-OpImageWeightedSampleQCOM-06971
  If `OpImageWeightedSampleQCOM` is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_2_WEIGHT_SAMPLED_IMAGE_BIT_QCOM`
- VUID-vkCmdTraceRaysIndirectKHR-OpImageWeightedSampleQCOM-06972
  If `OpImageWeightedSampleQCOM` uses a VkImageView as a sample weight image as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_2_WEIGHT_IMAGE_BIT_QCOM`
- VUID-vkCmdTraceRaysIndirectKHR-OpImageBoxFilterQCOM-06973
  If `OpImageBoxFilterQCOM` is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_2_BOX_FILTER_SAMPLED_BIT_QCOM`
- VUID-vkCmdTraceRaysIndirectKHR-OpImageBlockMatchSSDQCOM-06974
  If `OpImageBlockMatchSSDQCOM` is used to read from an VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM`
- VUID-vkCmdTraceRaysIndirectKHR-OpImageBlockMatchSADQCOM-06975
  If `OpImageBlockMatchSADQCOM` is used to read from an VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM`
- VUID-vkCmdTraceRaysIndirectKHR-OpImageBlockMatchSADQCOM-06976
  If `OpImageBlockMatchSADQCOM` or OpImageBlockMatchSSDQCOM is used to read from a reference image as result of this command, then the specified reference coordinates **must** not fail integer texel coordinate validation
- VUID-vkCmdTraceRaysIndirectKHR-OpImageWeightedSampleQCOM-06977
  If `OpImageWeightedSampleQCOM`, `OpImageBoxFilterQCOM`, `OpImageBlockMatchWindowSSDQCOM`, `OpImageBlockMatchWindowSADQCOM`, `OpImageBlockMatchGatherSSDQCOM`, `OpImageBlockMatchGatherSADQCOM`, `OpImageBlockMatchSSDQCOM`, or `OpImageBlockMatchSADQCOM` uses a VkSampler as a result of this command, then the sampler **must** have been created with `VK_SAMPLER_CREATE_IMAGE_PROCESSING_BIT_QCOM`

- VUID-vkCmdTraceRaysIndirectKHR-OpImageWeightedSampleQCOM-06978
  If any command other than `OpImageWeightedSampleQCOM`, `OpImageBoxFilterQCOM`, `OpImageBlockMatchWindowSSDQCOM`, `OpImageBlockMatchWindowSADQCOM`, `OpImageBlockMatchGatherSSDQCOM`, `OpImageBlockMatchGatherSADQCOM`, `OpImageBlockMatchSSDQCOM`, or `OpImageBlockMatchSADQCOM` uses a VkSampler as a result of this command, then the sampler **must** not have been created with `VK_SAMPLER_CREATE_IMAGE_PROCESSING_BIT_QCOM`
- VUID-vkCmdTraceRaysIndirectKHR-OpImageBlockMatchWindow-09215
  If a `OpImageBlockMatchWindow*QCOM` or `OpImageBlockMatchGather*QCOM` instruction is used to read from an VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM`
- VUID-vkCmdTraceRaysIndirectKHR-OpImageBlockMatchWindow-09216
  If a `OpImageBlockMatchWindow*QCOM` or `OpImageBlockMatchGather*QCOM` instruction is used to read from an VkImageView as a result of this command, then the image view's format **must** be a single-component format
- VUID-vkCmdTraceRaysIndirectKHR-OpImageBlockMatchWindow-09217
  If a `OpImageBlockMatchWindow*QCOM` or `OpImageBlockMatchGather*QCOM` read from a reference image as result of this command, then the specified reference coordinates **must** not fail integer texel coordinate validation
- VUID-vkCmdTraceRaysIndirectKHR-None-07288
  Any shader invocation executed by this command **must** terminate
- VUID-vkCmdTraceRaysIndirectKHR-None-09600
  If a descriptor with type equal to any of `VK_DESCRIPTOR_TYPE_SAMPLE_WEIGHT_IMAGE_QCOM`, `VK_DESCRIPTOR_TYPE_BLOCK_MATCH_IMAGE_QCOM`, `VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE`, `VK_DESCRIPTOR_TYPE_STORAGE_IMAGE`, or `VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT` is accessed as a result of this command, the image subresource identified by that descriptor **must** be in the image layout identified when the descriptor was written
- VUID-vkCmdTraceRaysIndirectKHR-None-03429
  Any shader group handle referenced by this call **must** have been queried from the currently bound ray tracing pipeline
- VUID-vkCmdTraceRaysIndirectKHR-None-09458
  If the bound ray tracing pipeline state was created with the `VK_DYNAMIC_STATE_RAY_TRACING_PIPELINE_STACK_SIZE_KHR` dynamic state enabled then vkCmdSetRayTracingPipelineStackSizeKHR **must** have been called in the current command buffer prior to this trace command
- VUID-vkCmdTraceRaysIndirectKHR-maxPipelineRayRecursionDepth-03679
  This command **must** not cause a shader call instruction to be executed from a shader invocation with a recursion depth greater than the value of `maxPipelineRayRecursionDepth` used to create the bound ray tracing pipeline
- VUID-vkCmdTraceRaysIndirectKHR-commandBuffer-03635
  `commandBuffer` **must** not be a protected command buffer
- VUID-vkCmdTraceRaysIndirectKHR-size-04023
  The `size` member of `pRayGenShaderBindingTable` **must** be equal to its `stride` member

- VUID-vkCmdTraceRaysIndirectKHR-pRayGenShaderBindingTable-03680
  If the buffer from which `pRayGenShaderBindingTable->deviceAddress` was queried is non-sparse then it **must** be bound completely and contiguously to a single `VkDeviceMemory` object
- VUID-vkCmdTraceRaysIndirectKHR-pRayGenShaderBindingTable-03681
  The buffer from which the `pRayGenShaderBindingTable->deviceAddress` is queried **must** have been created with the `VK_BUFFER_USAGE_SHADER_BINDING_TABLE_BIT_KHR` usage flag
- VUID-vkCmdTraceRaysIndirectKHR-pRayGenShaderBindingTable-03682
  `pRayGenShaderBindingTable->deviceAddress` **must** be a multiple of `VkPhysicalDeviceRayTracingPipelinePropertiesKHR::shaderGroupBaseAlignment`
- VUID-vkCmdTraceRaysIndirectKHR-pMissShaderBindingTable-03683
  If the buffer from which `pMissShaderBindingTable->deviceAddress` was queried is non-sparse then it **must** be bound completely and contiguously to a single `VkDeviceMemory` object
- VUID-vkCmdTraceRaysIndirectKHR-pMissShaderBindingTable-03684
  The buffer from which the `pMissShaderBindingTable->deviceAddress` is queried **must** have been created with the `VK_BUFFER_USAGE_SHADER_BINDING_TABLE_BIT_KHR` usage flag
- VUID-vkCmdTraceRaysIndirectKHR-pMissShaderBindingTable-03685
  `pMissShaderBindingTable->deviceAddress` **must** be a multiple of `VkPhysicalDeviceRayTracingPipelinePropertiesKHR::shaderGroupBaseAlignment`
- VUID-vkCmdTraceRaysIndirectKHR-stride-03686
  `pMissShaderBindingTable->stride` **must** be a multiple of `VkPhysicalDeviceRayTracingPipelinePropertiesKHR::shaderGroupHandleAlignment`
- VUID-vkCmdTraceRaysIndirectKHR-stride-04029
  `pMissShaderBindingTable->stride` **must** be less than or equal to `VkPhysicalDeviceRayTracingPipelinePropertiesKHR::maxShaderGroupStride`
- VUID-vkCmdTraceRaysIndirectKHR-pHitShaderBindingTable-03687
  If the buffer from which `pHitShaderBindingTable->deviceAddress` was queried is non-sparse then it **must** be bound completely and contiguously to a single `VkDeviceMemory` object
- VUID-vkCmdTraceRaysIndirectKHR-pHitShaderBindingTable-03688
  The buffer from which the `pHitShaderBindingTable->deviceAddress` is queried **must** have been created with the `VK_BUFFER_USAGE_SHADER_BINDING_TABLE_BIT_KHR` usage flag
- VUID-vkCmdTraceRaysIndirectKHR-pHitShaderBindingTable-03689
  `pHitShaderBindingTable->deviceAddress` **must** be a multiple of `VkPhysicalDeviceRayTracingPipelinePropertiesKHR::shaderGroupBaseAlignment`
- VUID-vkCmdTraceRaysIndirectKHR-stride-03690
  `pHitShaderBindingTable->stride` **must** be a multiple of `VkPhysicalDeviceRayTracingPipelinePropertiesKHR::shaderGroupHandleAlignment`
- VUID-vkCmdTraceRaysIndirectKHR-stride-04035
  `pHitShaderBindingTable->stride` **must** be less than or equal to `VkPhysicalDeviceRayTracingPipelinePropertiesKHR::maxShaderGroupStride`
- VUID-vkCmdTraceRaysIndirectKHR-pCallableShaderBindingTable-03691

If the buffer from which `pCallableShaderBindingTable->deviceAddress` was queried is non-sparse then it **must** be bound completely and contiguously to a single `VkDeviceMemory` object

- VUID-vkCmdTraceRaysIndirectKHR-pCallableShaderBindingTable-03692
  The buffer from which the `pCallableShaderBindingTable->deviceAddress` is queried **must** have been created with the `VK_BUFFER_USAGE_SHADER_BINDING_TABLE_BIT_KHR` usage flag
- VUID-vkCmdTraceRaysIndirectKHR-pCallableShaderBindingTable-03693
  `pCallableShaderBindingTable->deviceAddress` **must** be a multiple of `VkPhysicalDeviceRayTracingPipelinePropertiesKHR::shaderGroupBaseAlignment`
- VUID-vkCmdTraceRaysIndirectKHR-stride-03694
  `pCallableShaderBindingTable->stride` **must** be a multiple of `VkPhysicalDeviceRayTracingPipelinePropertiesKHR::shaderGroupHandleAlignment`
- VUID-vkCmdTraceRaysIndirectKHR-stride-04041
  `pCallableShaderBindingTable->stride` **must** be less than or equal to `VkPhysicalDeviceRayTracingPipelinePropertiesKHR::maxShaderGroupStride`
- VUID-vkCmdTraceRaysIndirectKHR-flags-03696
  If the currently bound ray tracing pipeline was created with `flags` that included `VK_PIPELINE_CREATE_RAY_TRACING_NO_NULL_CLOSEST_HIT_SHADERS_BIT_KHR`, `pHitShaderBindingTable->deviceAddress` **must** not be zero
- VUID-vkCmdTraceRaysIndirectKHR-flags-03697
  If the currently bound ray tracing pipeline was created with `flags` that included `VK_PIPELINE_CREATE_RAY_TRACING_NO_NULL_INTERSECTION_SHADERS_BIT_KHR`, `pHitShaderBindingTable->deviceAddress` **must** not be zero
- VUID-vkCmdTraceRaysIndirectKHR-flags-03511
  If the currently bound ray tracing pipeline was created with `flags` that included `VK_PIPELINE_CREATE_RAY_TRACING_NO_NULL_MISS_SHADERS_BIT_KHR`, the shader group handle identified by `pMissShaderBindingTable->deviceAddress` **must** not be set to zero
- VUID-vkCmdTraceRaysIndirectKHR-flags-03512
  If the currently bound ray tracing pipeline was created with `flags` that included `VK_PIPELINE_CREATE_RAY_TRACING_NO_NULL_ANY_HIT_SHADERS_BIT_KHR`, entries in the table identified by `pHitShaderBindingTable->deviceAddress` accessed as a result of this command in order to execute an any-hit shader **must** not be set to zero
- VUID-vkCmdTraceRaysIndirectKHR-flags-03513
  If the currently bound ray tracing pipeline was created with `flags` that included `VK_PIPELINE_CREATE_RAY_TRACING_NO_NULL_CLOSEST_HIT_SHADERS_BIT_KHR`, entries in the table identified by `pHitShaderBindingTable->deviceAddress` accessed as a result of this command in order to execute a closest hit shader **must** not be set to zero
- VUID-vkCmdTraceRaysIndirectKHR-flags-03514
  If the currently bound ray tracing pipeline was created with `flags` that included `VK_PIPELINE_CREATE_RAY_TRACING_NO_NULL_INTERSECTION_SHADERS_BIT_KHR`, entries in the table identified by `pHitShaderBindingTable->deviceAddress` accessed as a result of this command in order to execute an intersection shader **must** not be set to zero
- VUID-vkCmdTraceRaysIndirectKHR-pHitShaderBindingTable-04735

Any non-zero hit shader group entries in the table identified by `pHitShaderBindingTable->deviceAddress` accessed by this call from a geometry with a `geometryType` of `VK_GEOMETRY_TYPE_TRIANGLES_KHR` **must** have been created with `VK_RAY_TRACING_SHADER_GROUP_TYPE_TRIANGLES_HIT_GROUP_KHR`

- VUID-vkCmdTraceRaysIndirectKHR-pHitShaderBindingTable-04736
  Any non-zero hit shader group entries in the table identified by `pHitShaderBindingTable->deviceAddress` accessed by this call from a geometry with a `geometryType` of `VK_GEOMETRY_TYPE_AABBS_KHR` **must** have been created with `VK_RAY_TRACING_SHADER_GROUP_TYPE_PROCEDURAL_HIT_GROUP_KHR`

- VUID-vkCmdTraceRaysIndirectKHR-indirectDeviceAddress-03632
  If the buffer from which `indirectDeviceAddress` was queried is non-sparse then it **must** be bound completely and contiguously to a single `VkDeviceMemory` object
- VUID-vkCmdTraceRaysIndirectKHR-indirectDeviceAddress-03633
  The buffer from which `indirectDeviceAddress` was queried **must** have been created with the `VK_BUFFER_USAGE_INDIRECT_BUFFER_BIT` bit set
- VUID-vkCmdTraceRaysIndirectKHR-indirectDeviceAddress-03634
  `indirectDeviceAddress` **must** be a multiple of `4`
- VUID-vkCmdTraceRaysIndirectKHR-indirectDeviceAddress-03636
  All device addresses between `indirectDeviceAddress` and `indirectDeviceAddress` + `sizeof`(`VkTraceRaysIndirectCommandKHR`) - 1 **must** be in the buffer device address range of the same buffer
- VUID-vkCmdTraceRaysIndirectKHR-rayTracingPipelineTraceRaysIndirect-03637
  The `rayTracingPipelineTraceRaysIndirect` feature **must** be enabled
- VUID-vkCmdTraceRaysIndirectKHR-rayTracingMotionBlurPipelineTraceRaysIndirect-04951
  If the bound ray tracing pipeline was created with `VK_PIPELINE_CREATE_RAY_TRACING_ALLOW_MOTION_BIT_NV` `VkPhysicalDeviceRayTracingMotionBlurFeaturesNV::rayTracingMotionBlurPipelineTraceRaysIndirect` feature **must** be enabled

## Valid Usage (Implicit)

- VUID-vkCmdTraceRaysIndirectKHR-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdTraceRaysIndirectKHR-pRaygenShaderBindingTable-parameter
  `pRaygenShaderBindingTable` **must** be a valid pointer to a valid VkStridedDeviceAddressRegionKHR structure
- VUID-vkCmdTraceRaysIndirectKHR-pMissShaderBindingTable-parameter
  `pMissShaderBindingTable` **must** be a valid pointer to a valid VkStridedDeviceAddressRegionKHR structure
- VUID-vkCmdTraceRaysIndirectKHR-pHitShaderBindingTable-parameter
  `pHitShaderBindingTable` **must** be a valid pointer to a valid

VkStridedDeviceAddressRegionKHR structure

- VUID-vkCmdTraceRaysIndirectKHR-pCallableShaderBindingTable-parameter
  `pCallableShaderBindingTable` **must** be a valid pointer to a valid VkStridedDeviceAddressRegionKHR structure
- VUID-vkCmdTraceRaysIndirectKHR-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdTraceRaysIndirectKHR-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support compute operations
- VUID-vkCmdTraceRaysIndirectKHR-renderpass
  This command **must** only be called outside of a render pass instance
- VUID-vkCmdTraceRaysIndirectKHR-videocoding
  This command **must** only be called outside of a video coding scope

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Outside | Outside | Compute | Action |

The `VkTraceRaysIndirectCommandKHR` structure is defined as:

```
// Provided by VK_KHR_ray_tracing_pipeline
typedef struct VkTraceRaysIndirectCommandKHR {
    uint32_t    width;
    uint32_t    height;
    uint32_t    depth;
} VkTraceRaysIndirectCommandKHR;
```

- `width` is the width of the ray trace query dimensions.
- `height` is height of the ray trace query dimensions.
- `depth` is depth of the ray trace query dimensions.

The members of `VkTraceRaysIndirectCommandKHR` have the same meaning as the similarly named

parameters of vkCmdTraceRaysKHR.

### Valid Usage

- VUID-VkTraceRaysIndirectCommandKHR-width-03638
  width **must** be less than or equal to VkPhysicalDeviceLimits::maxComputeWorkGroupCount[0] × VkPhysicalDeviceLimits::maxComputeWorkGroupSize[0]
- VUID-VkTraceRaysIndirectCommandKHR-height-03639
  height **must** be less than or equal to VkPhysicalDeviceLimits::maxComputeWorkGroupCount[1] × VkPhysicalDeviceLimits::maxComputeWorkGroupSize[1]
- VUID-VkTraceRaysIndirectCommandKHR-depth-03640
  depth **must** be less than or equal to VkPhysicalDeviceLimits::maxComputeWorkGroupCount[2] × VkPhysicalDeviceLimits::maxComputeWorkGroupSize[2]
- VUID-VkTraceRaysIndirectCommandKHR-width-03641
  width × height × depth **must** be less than or equal to VkPhysicalDeviceRayTracingPipelinePropertiesKHR::maxRayDispatchInvocationCount

To dispatch ray tracing, with some parameters sourced on the device, use:

```
// Provided by VK_KHR_ray_tracing_maintenance1 with VK_KHR_ray_tracing_pipeline
void vkCmdTraceRaysIndirect2KHR(
    VkCommandBuffer                             commandBuffer,
    VkDeviceAddress                             indirectDeviceAddress);
```

- commandBuffer is the command buffer into which the command will be recorded.
- indirectDeviceAddress is a buffer device address which is a pointer to a VkTraceRaysIndirectCommand2KHR structure containing the trace ray parameters.

vkCmdTraceRaysIndirect2KHR behaves similarly to vkCmdTraceRaysIndirectKHR except that shader binding table parameters as well as dispatch dimensions are read by the device from indirectDeviceAddress during execution.

### Valid Usage

- VUID-vkCmdTraceRaysIndirect2KHR-magFilter-04553
  If a VkSampler created with magFilter or minFilter equal to VK_FILTER_LINEAR, reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE, and compareEnable equal to VK_FALSE is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT
- VUID-vkCmdTraceRaysIndirect2KHR-magFilter-09598
  If a VkSampler created with magFilter or minFilter equal to VK_FILTER_LINEAR and reductionMode equal to either VK_SAMPLER_REDUCTION_MODE_MIN or VK_SAMPLER_REDUCTION_MODE_MAX is used to sample a VkImageView as a result of this

command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT

- VUID-vkCmdTraceRaysIndirect2KHR-mipmapMode-04770
  If a VkSampler created with mipmapMode equal to VK_SAMPLER_MIPMAP_MODE_LINEAR, reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE, and compareEnable equal to VK_FALSE is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT
- VUID-vkCmdTraceRaysIndirect2KHR-mipmapMode-09599
  If a VkSampler created with mipmapMode equal to VK_SAMPLER_MIPMAP_MODE_LINEAR and reductionMode equal to either VK_SAMPLER_REDUCTION_MODE_MIN or VK_SAMPLER_REDUCTION_MODE_MAX is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT
- VUID-vkCmdTraceRaysIndirect2KHR-unnormalizedCoordinates-09635
  If a VkSampler created with unnormalizedCoordinates equal to VK_TRUE is used to sample a VkImageView as a result of this command, then the image view's levelCount and layerCount **must** be 1
- VUID-vkCmdTraceRaysIndirect2KHR-unnormalizedCoordinates-09636
  If a VkSampler created with unnormalizedCoordinates equal to VK_TRUE is used to sample a VkImageView as a result of this command, then the image view's viewType **must** be VK_IMAGE_VIEW_TYPE_1D or VK_IMAGE_VIEW_TYPE_2D
- VUID-vkCmdTraceRaysIndirect2KHR-None-06479
  If a VkImageView is sampled with depth comparison, the image view's format features **must** contain VK_FORMAT_FEATURE_2_SAMPLED_IMAGE_DEPTH_COMPARISON_BIT
- VUID-vkCmdTraceRaysIndirect2KHR-None-02691
  If a VkImageView is accessed using atomic operations as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_STORAGE_IMAGE_ATOMIC_BIT
- VUID-vkCmdTraceRaysIndirect2KHR-None-07888
  If a VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER descriptor is accessed using atomic operations as a result of this command, then the storage texel buffer's format features **must** contain VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_ATOMIC_BIT
- VUID-vkCmdTraceRaysIndirect2KHR-None-02692
  If a VkImageView is sampled with VK_FILTER_CUBIC_EXT as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_CUBIC_BIT_EXT
- VUID-vkCmdTraceRaysIndirect2KHR-None-02693
  If the VK_EXT_filter_cubic extension is not enabled and any VkImageView is sampled with VK_FILTER_CUBIC_EXT as a result of this command, it **must** not have a VkImageViewType of VK_IMAGE_VIEW_TYPE_3D, VK_IMAGE_VIEW_TYPE_CUBE, or VK_IMAGE_VIEW_TYPE_CUBE_ARRAY
- VUID-vkCmdTraceRaysIndirect2KHR-filterCubic-02694
  Any VkImageView being sampled with VK_FILTER_CUBIC_EXT as a result of this command **must** have a VkImageViewType and format that supports cubic filtering, as specified by

VkFilterCubicImageViewImageFormatPropertiesEXT::filterCubic returned by vkGetPhysicalDeviceImageFormatProperties2

- VUID-vkCmdTraceRaysIndirect2KHR-filterCubicMinmax-02695
  Any VkImageView being sampled with VK_FILTER_CUBIC_EXT with a reduction mode of either VK_SAMPLER_REDUCTION_MODE_MIN or VK_SAMPLER_REDUCTION_MODE_MAX as a result of this command **must** have a VkImageViewType and format that supports cubic filtering together with minmax filtering, as specified by VkFilterCubicImageViewImageFormatPropertiesEXT::filterCubicMinmax returned by vkGetPhysicalDeviceImageFormatProperties2
- VUID-vkCmdTraceRaysIndirect2KHR-cubicRangeClamp-09212
  If the cubicRangeClamp feature is not enabled, then any VkImageView being sampled with VK_FILTER_CUBIC_EXT as a result of this command **must** not have a VkSamplerReductionModeCreateInfo::reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE_RANGECLAMP_QCOM
- VUID-vkCmdTraceRaysIndirect2KHR-reductionMode-09213
  Any VkImageView being sampled with a VkSamplerReductionModeCreateInfo::reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE_RANGECLAMP_QCOM as a result of this command **must** sample with VK_FILTER_CUBIC_EXT
- VUID-vkCmdTraceRaysIndirect2KHR-selectableCubicWeights-09214
  If the selectableCubicWeights feature is not enabled, then any VkImageView being sampled with VK_FILTER_CUBIC_EXT as a result of this command **must** have VkSamplerCubicWeightsCreateInfoQCOM::cubicWeights equal to VK_CUBIC_FILTER_WEIGHTS_CATMULL_ROM_QCOM
- VUID-vkCmdTraceRaysIndirect2KHR-flags-02696
  Any VkImage created with a VkImageCreateInfo::flags containing VK_IMAGE_CREATE_CORNER_SAMPLED_BIT_NV sampled as a result of this command **must** only be sampled using a VkSamplerAddressMode of VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_EDGE
- VUID-vkCmdTraceRaysIndirect2KHR-OpTypeImage-07027
  For any VkImageView being written as a storage image where the image format field of the OpTypeImage is Unknown, the view's format features **must** contain VK_FORMAT_FEATURE_2_STORAGE_WRITE_WITHOUT_FORMAT_BIT
- VUID-vkCmdTraceRaysIndirect2KHR-OpTypeImage-07028
  For any VkImageView being read as a storage image where the image format field of the OpTypeImage is Unknown, the view's format features **must** contain VK_FORMAT_FEATURE_2_STORAGE_READ_WITHOUT_FORMAT_BIT
- VUID-vkCmdTraceRaysIndirect2KHR-OpTypeImage-07029
  For any VkBufferView being written as a storage texel buffer where the image format field of the OpTypeImage is Unknown, the view's buffer features **must** contain VK_FORMAT_FEATURE_2_STORAGE_WRITE_WITHOUT_FORMAT_BIT
- VUID-vkCmdTraceRaysIndirect2KHR-OpTypeImage-07030
  Any VkBufferView being read as a storage texel buffer where the image format field of the OpTypeImage is Unknown then the view's buffer features **must** contain VK_FORMAT_FEATURE_2_STORAGE_READ_WITHOUT_FORMAT_BIT
- VUID-vkCmdTraceRaysIndirect2KHR-None-08600

For each set *n* that is statically used by a bound shader, a descriptor set **must** have been bound to *n* at the same pipeline bind point, with a VkPipelineLayout that is compatible for set *n*, with the VkPipelineLayout used to create the current VkPipeline or the VkDescriptorSetLayout array used to create the current VkShaderEXT , as described in Pipeline Layout Compatibility

- VUID-vkCmdTraceRaysIndirect2KHR-None-08601
  For each push constant that is statically used by a bound shader, a push constant value **must** have been set for the same pipeline bind point, with a VkPipelineLayout that is compatible for push constants, with the VkPipelineLayout used to create the current VkPipeline or the VkDescriptorSetLayout array used to create the current VkShaderEXT , as described in Pipeline Layout Compatibility

- VUID-vkCmdTraceRaysIndirect2KHR-None-10068
  For each array of resources that is used by a bound shader, the indices used to access members of the array **must** be less than the descriptor count for the identified binding in the descriptor sets used by this command

- VUID-vkCmdTraceRaysIndirect2KHR-maintenance4-08602
  If the `maintenance4` feature is not enabled, then for each push constant that is statically used by a bound shader, a push constant value **must** have been set for the same pipeline bind point, with a VkPipelineLayout that is compatible for push constants, with the VkPipelineLayout used to create the current VkPipeline or the VkDescriptorSetLayout and VkPushConstantRange arrays used to create the current VkShaderEXT , as described in Pipeline Layout Compatibility

- VUID-vkCmdTraceRaysIndirect2KHR-None-08114
  Descriptors in each bound descriptor set, specified via vkCmdBindDescriptorSets, **must** be valid as described by descriptor validity if they are statically used by the VkPipeline bound to the pipeline bind point used by this command and the bound VkPipeline was not created with `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`

- VUID-vkCmdTraceRaysIndirect2KHR-None-08115
  If the descriptors used by the VkPipeline bound to the pipeline bind point were specified via vkCmdBindDescriptorSets, the bound VkPipeline **must** have been created without `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`

- VUID-vkCmdTraceRaysIndirect2KHR-None-08116
  Descriptors in bound descriptor buffers, specified via vkCmdSetDescriptorBufferOffsetsEXT, **must** be valid if they are dynamically used by the VkPipeline bound to the pipeline bind point used by this command and the bound VkPipeline was created with `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`

- VUID-vkCmdTraceRaysIndirect2KHR-None-08604
  Descriptors in bound descriptor buffers, specified via vkCmdSetDescriptorBufferOffsetsEXT, **must** be valid if they are dynamically used by any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command

- VUID-vkCmdTraceRaysIndirect2KHR-None-08117
  If the descriptors used by the VkPipeline bound to the pipeline bind point were specified via vkCmdSetDescriptorBufferOffsetsEXT, the bound VkPipeline **must** have been created with `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`

- VUID-vkCmdTraceRaysIndirect2KHR-None-08119
  If a descriptor is dynamically used with a VkPipeline created with VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT, the descriptor memory **must** be resident
- VUID-vkCmdTraceRaysIndirect2KHR-None-08605
  If a descriptor is dynamically used with a VkShaderEXT created with a VkDescriptorSetLayout that was created with VK_DESCRIPTOR_SET_LAYOUT_CREATE_DESCRIPTOR_BUFFER_BIT_EXT, the descriptor memory **must** be resident
- VUID-vkCmdTraceRaysIndirect2KHR-None-08606
  If the shaderObject feature is not enabled, a valid pipeline **must** be bound to the pipeline bind point used by this command
- VUID-vkCmdTraceRaysIndirect2KHR-None-08608
  If a pipeline is bound to the pipeline bind point used by this command, there **must** not have been any calls to dynamic state setting commands for any state specified statically in the VkPipeline object bound to the pipeline bind point used by this command, since that pipeline was bound
- VUID-vkCmdTraceRaysIndirect2KHR-None-08609
  If the VkPipeline object bound to the pipeline bind point used by this command or any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a VkSampler object that uses unnormalized coordinates, that sampler **must** not be used to sample from any VkImage with a VkImageView of the type VK_IMAGE_VIEW_TYPE_3D, VK_IMAGE_VIEW_TYPE_CUBE, VK_IMAGE_VIEW_TYPE_1D_ARRAY, VK_IMAGE_VIEW_TYPE_2D_ARRAY or VK_IMAGE_VIEW_TYPE_CUBE_ARRAY, in any shader stage
- VUID-vkCmdTraceRaysIndirect2KHR-None-08610
  If the VkPipeline object bound to the pipeline bind point used by this command or any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a VkSampler object that uses unnormalized coordinates, that sampler **must** not be used with any of the SPIR-V OpImageSample* or OpImageSparseSample* instructions with ImplicitLod, Dref or Proj in their name, in any shader stage
- VUID-vkCmdTraceRaysIndirect2KHR-None-08611
  If the VkPipeline object bound to the pipeline bind point used by this command or any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a VkSampler object that uses unnormalized coordinates, that sampler **must** not be used with any of the SPIR-V OpImageSample* or OpImageSparseSample* instructions that includes a LOD bias or any offset values, in any shader stage
- VUID-vkCmdTraceRaysIndirect2KHR-None-08607
  If the shaderObject is enabled, either a valid pipeline **must** be bound to the pipeline bind point used by this command, or a valid combination of valid and VK_NULL_HANDLE shader objects **must** be bound to every supported shader stage corresponding to the pipeline bind point used by this command
- VUID-vkCmdTraceRaysIndirect2KHR-uniformBuffers-06935
  If any stage of the VkPipeline object bound to the pipeline bind point used by this command accesses a uniform buffer, and that stage was created without enabling either VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_EXT or VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_2_EXT for uniformBuffers,

and the robustBufferAccess feature is not enabled, that stage **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point

- VUID-vkCmdTraceRaysIndirect2KHR-None-08612
  If the robustBufferAccess feature is not enabled, and any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a uniform buffer, it **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point

- VUID-vkCmdTraceRaysIndirect2KHR-storageBuffers-06936
  If any stage of the VkPipeline object bound to the pipeline bind point used by this command accesses a storage buffer, and that stage was created without enabling either VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_EXT or VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_2_EXT for storageBuffers, and the robustBufferAccess feature is not enabled, that stage **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point

- VUID-vkCmdTraceRaysIndirect2KHR-None-08613
  If the robustBufferAccess feature is not enabled, and any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a storage buffer, it **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point

- VUID-vkCmdTraceRaysIndirect2KHR-commandBuffer-02707
  If commandBuffer is an unprotected command buffer and protectedNoFault is not supported, any resource accessed by bound shaders **must** not be a protected resource

- VUID-vkCmdTraceRaysIndirect2KHR-None-06550
  If a bound shader accesses a VkSampler or VkImageView object that enables sampler $Y'C_BC_R$ conversion, that object **must** only be used with OpImageSample* or OpImageSparseSample* instructions

- VUID-vkCmdTraceRaysIndirect2KHR-ConstOffset-06551
  If a bound shader accesses a VkSampler or VkImageView object that enables sampler $Y'C_BC_R$ conversion, that object **must** not use the ConstOffset and Offset operands

- VUID-vkCmdTraceRaysIndirect2KHR-viewType-07752
  If a VkImageView is accessed as a result of this command, then the image view's viewType **must** match the Dim operand of the OpTypeImage as described in Compatibility Between SPIR-V Image Dimensions and Vulkan ImageView Types

- VUID-vkCmdTraceRaysIndirect2KHR-format-07753
  If a VkImageView is accessed as a result of this command, then the numeric type of the image view's format and the Sampled Type operand of the OpTypeImage **must** match

- VUID-vkCmdTraceRaysIndirect2KHR-OpImageWrite-08795
  If a VkImageView created with a format other than VK_FORMAT_A8_UNORM_KHR is accessed using OpImageWrite as a result of this command, then the Type of the Texel operand of that instruction **must** have at least as many components as the image view's format

- VUID-vkCmdTraceRaysIndirect2KHR-OpImageWrite-08796
  If a VkImageView created with the format VK_FORMAT_A8_UNORM_KHR is accessed using

OpImageWrite as a result of this command, then the Type of the Texel operand of that instruction **must** have four components

- VUID-vkCmdTraceRaysIndirect2KHR-OpImageWrite-04469
  If a VkBufferView is accessed using OpImageWrite as a result of this command, then the Type of the Texel operand of that instruction **must** have at least as many components as the buffer view's format
- VUID-vkCmdTraceRaysIndirect2KHR-SampledType-04470
  If a VkImageView with a VkFormat that has a 64-bit component width is accessed as a result of this command, the SampledType of the OpTypeImage operand of that instruction **must** have a Width of 64
- VUID-vkCmdTraceRaysIndirect2KHR-SampledType-04471
  If a VkImageView with a VkFormat that has a component width less than 64-bit is accessed as a result of this command, the SampledType of the OpTypeImage operand of that instruction **must** have a Width of 32
- VUID-vkCmdTraceRaysIndirect2KHR-SampledType-04472
  If a VkBufferView with a VkFormat that has a 64-bit component width is accessed as a result of this command, the SampledType of the OpTypeImage operand of that instruction **must** have a Width of 64
- VUID-vkCmdTraceRaysIndirect2KHR-SampledType-04473
  If a VkBufferView with a VkFormat that has a component width less than 64-bit is accessed as a result of this command, the SampledType of the OpTypeImage operand of that instruction **must** have a Width of 32
- VUID-vkCmdTraceRaysIndirect2KHR-sparseImageInt64Atomics-04474
  If the sparseImageInt64Atomics feature is not enabled, VkImage objects created with the VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT flag **must** not be accessed by atomic instructions through an OpTypeImage with a SampledType with a Width of 64 by this command
- VUID-vkCmdTraceRaysIndirect2KHR-sparseImageInt64Atomics-04475
  If the sparseImageInt64Atomics feature is not enabled, VkBuffer objects created with the VK_BUFFER_CREATE_SPARSE_RESIDENCY_BIT flag **must** not be accessed by atomic instructions through an OpTypeImage with a SampledType with a Width of 64 by this command
- VUID-vkCmdTraceRaysIndirect2KHR-OpImageWeightedSampleQCOM-06971
  If OpImageWeightedSampleQCOM is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_WEIGHT_SAMPLED_IMAGE_BIT_QCOM
- VUID-vkCmdTraceRaysIndirect2KHR-OpImageWeightedSampleQCOM-06972
  If OpImageWeightedSampleQCOM uses a VkImageView as a sample weight image as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_WEIGHT_IMAGE_BIT_QCOM
- VUID-vkCmdTraceRaysIndirect2KHR-OpImageBoxFilterQCOM-06973
  If OpImageBoxFilterQCOM is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_BOX_FILTER_SAMPLED_BIT_QCOM
- VUID-vkCmdTraceRaysIndirect2KHR-OpImageBlockMatchSSDQCOM-06974

If OpImageBlockMatchSSDQCOM is used to read from an VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM

- VUID-vkCmdTraceRaysIndirect2KHR-OpImageBlockMatchSADQCOM-06975
  If OpImageBlockMatchSADQCOM is used to read from an VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM
- VUID-vkCmdTraceRaysIndirect2KHR-OpImageBlockMatchSADQCOM-06976
  If OpImageBlockMatchSADQCOM or OpImageBlockMatchSSDQCOM is used to read from a reference image as result of this command, then the specified reference coordinates **must** not fail integer texel coordinate validation
- VUID-vkCmdTraceRaysIndirect2KHR-OpImageWeightedSampleQCOM-06977
  If OpImageWeightedSampleQCOM, OpImageBoxFilterQCOM, OpImageBlockMatchWindowSSDQCOM, OpImageBlockMatchWindowSADQCOM, OpImageBlockMatchGatherSSDQCOM, OpImageBlockMatchGatherSADQCOM, OpImageBlockMatchSSDQCOM, or OpImageBlockMatchSADQCOM uses a VkSampler as a result of this command, then the sampler **must** have been created with VK_SAMPLER_CREATE_IMAGE_PROCESSING_BIT_QCOM
- VUID-vkCmdTraceRaysIndirect2KHR-OpImageWeightedSampleQCOM-06978
  If any command other than OpImageWeightedSampleQCOM, OpImageBoxFilterQCOM, OpImageBlockMatchWindowSSDQCOM, OpImageBlockMatchWindowSADQCOM, OpImageBlockMatchGatherSSDQCOM, OpImageBlockMatchGatherSADQCOM, OpImageBlockMatchSSDQCOM, or OpImageBlockMatchSADQCOM uses a VkSampler as a result of this command, then the sampler **must** not have been created with VK_SAMPLER_CREATE_IMAGE_PROCESSING_BIT_QCOM
- VUID-vkCmdTraceRaysIndirect2KHR-OpImageBlockMatchWindow-09215
  If a OpImageBlockMatchWindow*QCOM or OpImageBlockMatchGather*QCOM instruction is used to read from an VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM
- VUID-vkCmdTraceRaysIndirect2KHR-OpImageBlockMatchWindow-09216
  If a OpImageBlockMatchWindow*QCOM or OpImageBlockMatchGather*QCOM instruction is used to read from an VkImageView as a result of this command, then the image view's format **must** be a single-component format
- VUID-vkCmdTraceRaysIndirect2KHR-OpImageBlockMatchWindow-09217
  If a OpImageBlockMatchWindow*QCOM or OpImageBlockMatchGather*QCOM read from a reference image as result of this command, then the specified reference coordinates **must** not fail integer texel coordinate validation
- VUID-vkCmdTraceRaysIndirect2KHR-None-07288
  Any shader invocation executed by this command **must** terminate
- VUID-vkCmdTraceRaysIndirect2KHR-None-09600
  If a descriptor with type equal to any of VK_DESCRIPTOR_TYPE_SAMPLE_WEIGHT_IMAGE_QCOM, VK_DESCRIPTOR_TYPE_BLOCK_MATCH_IMAGE_QCOM, VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE, VK_DESCRIPTOR_TYPE_STORAGE_IMAGE, or VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT is accessed as a result of this command, the image subresource identified by that descriptor **must** be in the image layout identified when the descriptor was written

- VUID-vkCmdTraceRaysIndirect2KHR-None-03429
  Any shader group handle referenced by this call **must** have been queried from the currently bound ray tracing pipeline
- VUID-vkCmdTraceRaysIndirect2KHR-None-09458
  If the bound ray tracing pipeline state was created with the `VK_DYNAMIC_STATE_RAY_TRACING_PIPELINE_STACK_SIZE_KHR` dynamic state enabled then vkCmdSetRayTracingPipelineStackSizeKHR **must** have been called in the current command buffer prior to this trace command
- VUID-vkCmdTraceRaysIndirect2KHR-maxPipelineRayRecursionDepth-03679
  This command **must** not cause a shader call instruction to be executed from a shader invocation with a recursion depth greater than the value of `maxPipelineRayRecursionDepth` used to create the bound ray tracing pipeline
- VUID-vkCmdTraceRaysIndirect2KHR-commandBuffer-03635
  `commandBuffer` **must** not be a protected command buffer
- VUID-vkCmdTraceRaysIndirect2KHR-indirectDeviceAddress-03632
  If the buffer from which `indirectDeviceAddress` was queried is non-sparse then it **must** be bound completely and contiguously to a single `VkDeviceMemory` object
- VUID-vkCmdTraceRaysIndirect2KHR-indirectDeviceAddress-03633
  The buffer from which `indirectDeviceAddress` was queried **must** have been created with the `VK_BUFFER_USAGE_INDIRECT_BUFFER_BIT` bit set
- VUID-vkCmdTraceRaysIndirect2KHR-indirectDeviceAddress-03634
  `indirectDeviceAddress` **must** be a multiple of `4`
- VUID-vkCmdTraceRaysIndirect2KHR-indirectDeviceAddress-03636
  All device addresses between `indirectDeviceAddress` and `indirectDeviceAddress` + `sizeof`(`VkTraceRaysIndirectCommand2KHR`) - 1 **must** be in the buffer device address range of the same buffer
- VUID-vkCmdTraceRaysIndirect2KHR-rayTracingPipelineTraceRaysIndirect2-03637
  The `rayTracingPipelineTraceRaysIndirect2` feature **must** be enabled
- VUID-vkCmdTraceRaysIndirect2KHR-rayTracingMotionBlurPipelineTraceRaysIndirect-04951
  If the bound ray tracing pipeline was created with `VK_PIPELINE_CREATE_RAY_TRACING_ALLOW_MOTION_BIT_NV` `VkPhysicalDeviceRayTracingMotionBlurFeaturesNV`::`rayTracingMotionBlurPipelineTraceRaysIndirect` feature **must** be enabled

## Valid Usage (Implicit)

- VUID-vkCmdTraceRaysIndirect2KHR-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdTraceRaysIndirect2KHR-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdTraceRaysIndirect2KHR-commandBuffer-cmdpool

The VkCommandPool that commandBuffer was allocated from **must** support compute operations

- VUID-vkCmdTraceRaysIndirect2KHR-renderpass
  This command **must** only be called outside of a render pass instance
- VUID-vkCmdTraceRaysIndirect2KHR-videocoding
  This command **must** only be called outside of a video coding scope

## Host Synchronization

- Host access to commandBuffer **must** be externally synchronized
- Host access to the VkCommandPool that commandBuffer was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Outside | Outside | Compute | Action |

The VkTraceRaysIndirectCommand2KHR structure is defined as:

```
// Provided by VK_KHR_ray_tracing_maintenance1 with VK_KHR_ray_tracing_pipeline
typedef struct VkTraceRaysIndirectCommand2KHR {
    VkDeviceAddress    raygenShaderRecordAddress;
    VkDeviceSize       raygenShaderRecordSize;
    VkDeviceAddress    missShaderBindingTableAddress;
    VkDeviceSize       missShaderBindingTableSize;
    VkDeviceSize       missShaderBindingTableStride;
    VkDeviceAddress    hitShaderBindingTableAddress;
    VkDeviceSize       hitShaderBindingTableSize;
    VkDeviceSize       hitShaderBindingTableStride;
    VkDeviceAddress    callableShaderBindingTableAddress;
    VkDeviceSize       callableShaderBindingTableSize;
    VkDeviceSize       callableShaderBindingTableStride;
    uint32_t           width;
    uint32_t           height;
    uint32_t           depth;
} VkTraceRaysIndirectCommand2KHR;
```

- raygenShaderRecordAddress is a VkDeviceAddress of the ray generation shader binding table record used by this command.
- raygenShaderRecordSize is a VkDeviceSize number of bytes corresponding to the ray generation

shader binding table record at base address raygenShaderRecordAddress.

- missShaderBindingTableAddress is a VkDeviceAddress of the first record in the miss shader binding table used by this command.
- missShaderBindingTableSize is a VkDeviceSize number of bytes corresponding to the total size of the miss shader binding table at missShaderBindingTableAddress that may be accessed by this command.
- missShaderBindingTableStride is a VkDeviceSize number of bytes between records of the miss shader binding table.
- hitShaderBindingTableAddress is a VkDeviceAddress of the first record in the hit shader binding table used by this command.
- hitShaderBindingTableSize is a VkDeviceSize number of bytes corresponding to the total size of the hit shader binding table at hitShaderBindingTableAddress that may be accessed by this command.
- hitShaderBindingTableStride is a VkDeviceSize number of bytes between records of the hit shader binding table.
- callableShaderBindingTableAddress is a VkDeviceAddress of the first record in the callable shader binding table used by this command.
- callableShaderBindingTableSize is a VkDeviceSize number of bytes corresponding to the total size of the callable shader binding table at callableShaderBindingTableAddress that may be accessed by this command.
- callableShaderBindingTableStride is a VkDeviceSize number of bytes between records of the callable shader binding table.
- width is the width of the ray trace query dimensions.
- height is height of the ray trace query dimensions.
- depth is depth of the ray trace query dimensions.

The members of VkTraceRaysIndirectCommand2KHR have the same meaning as the similarly named parameters of vkCmdTraceRaysKHR.

Indirect shader binding table buffer parameters **must** satisfy the same memory alignment and binding requirements as their counterparts in vkCmdTraceRaysIndirectKHR and vkCmdTraceRaysKHR.

### Valid Usage

- VUID-VkTraceRaysIndirectCommand2KHR-pRayGenShaderBindingTable-03680
  If the buffer from which raygenShaderRecordAddress was queried is non-sparse then it **must** be bound completely and contiguously to a single VkDeviceMemory object
- VUID-VkTraceRaysIndirectCommand2KHR-pRayGenShaderBindingTable-03681
  The buffer from which the raygenShaderRecordAddress is queried **must** have been created with the VK_BUFFER_USAGE_SHADER_BINDING_TABLE_BIT_KHR usage flag
- VUID-VkTraceRaysIndirectCommand2KHR-pRayGenShaderBindingTable-03682

raygenShaderRecordAddress **must** be a multiple of VkPhysicalDeviceRayTracingPipelinePropertiesKHR::shaderGroupBaseAlignment

- VUID-VkTraceRaysIndirectCommand2KHR-pMissShaderBindingTable-03683
  If the buffer from which missShaderBindingTableAddress was queried is non-sparse then it **must** be bound completely and contiguously to a single VkDeviceMemory object
- VUID-VkTraceRaysIndirectCommand2KHR-pMissShaderBindingTable-03684
  The buffer from which the missShaderBindingTableAddress is queried **must** have been created with the VK_BUFFER_USAGE_SHADER_BINDING_TABLE_BIT_KHR usage flag
- VUID-VkTraceRaysIndirectCommand2KHR-pMissShaderBindingTable-03685
  missShaderBindingTableAddress **must** be a multiple of VkPhysicalDeviceRayTracingPipelinePropertiesKHR::shaderGroupBaseAlignment
- VUID-VkTraceRaysIndirectCommand2KHR-stride-03686
  missShaderBindingTableStride **must** be a multiple of VkPhysicalDeviceRayTracingPipelinePropertiesKHR::shaderGroupHandleAlignment
- VUID-VkTraceRaysIndirectCommand2KHR-stride-04029
  missShaderBindingTableStride **must** be less than or equal to VkPhysicalDeviceRayTracingPipelinePropertiesKHR::maxShaderGroupStride
- VUID-VkTraceRaysIndirectCommand2KHR-pHitShaderBindingTable-03687
  If the buffer from which hitShaderBindingTableAddress was queried is non-sparse then it **must** be bound completely and contiguously to a single VkDeviceMemory object
- VUID-VkTraceRaysIndirectCommand2KHR-pHitShaderBindingTable-03688
  The buffer from which the hitShaderBindingTableAddress is queried **must** have been created with the VK_BUFFER_USAGE_SHADER_BINDING_TABLE_BIT_KHR usage flag
- VUID-VkTraceRaysIndirectCommand2KHR-pHitShaderBindingTable-03689
  hitShaderBindingTableAddress **must** be a multiple of VkPhysicalDeviceRayTracingPipelinePropertiesKHR::shaderGroupBaseAlignment
- VUID-VkTraceRaysIndirectCommand2KHR-stride-03690
  hitShaderBindingTableStride **must** be a multiple of VkPhysicalDeviceRayTracingPipelinePropertiesKHR::shaderGroupHandleAlignment
- VUID-VkTraceRaysIndirectCommand2KHR-stride-04035
  hitShaderBindingTableStride **must** be less than or equal to VkPhysicalDeviceRayTracingPipelinePropertiesKHR::maxShaderGroupStride
- VUID-VkTraceRaysIndirectCommand2KHR-pCallableShaderBindingTable-03691
  If the buffer from which callableShaderBindingTableAddress was queried is non-sparse then it **must** be bound completely and contiguously to a single VkDeviceMemory object
- VUID-VkTraceRaysIndirectCommand2KHR-pCallableShaderBindingTable-03692
  The buffer from which the callableShaderBindingTableAddress is queried **must** have been created with the VK_BUFFER_USAGE_SHADER_BINDING_TABLE_BIT_KHR usage flag
- VUID-VkTraceRaysIndirectCommand2KHR-pCallableShaderBindingTable-03693
  callableShaderBindingTableAddress **must** be a multiple of VkPhysicalDeviceRayTracingPipelinePropertiesKHR::shaderGroupBaseAlignment
- VUID-VkTraceRaysIndirectCommand2KHR-stride-03694

callableShaderBindingTableStride **must** be a multiple of VkPhysicalDeviceRayTracingPipelinePropertiesKHR::shaderGroupHandleAlignment

- VUID-VkTraceRaysIndirectCommand2KHR-stride-04041
  callableShaderBindingTableStride **must** be less than or equal to VkPhysicalDeviceRayTracingPipelinePropertiesKHR::maxShaderGroupStride
- VUID-VkTraceRaysIndirectCommand2KHR-flags-03696
  If the currently bound ray tracing pipeline was created with flags that included VK_PIPELINE_CREATE_RAY_TRACING_NO_NULL_CLOSEST_HIT_SHADERS_BIT_KHR, hitShaderBindingTableAddress **must** not be zero
- VUID-VkTraceRaysIndirectCommand2KHR-flags-03697
  If the currently bound ray tracing pipeline was created with flags that included VK_PIPELINE_CREATE_RAY_TRACING_NO_NULL_INTERSECTION_SHADERS_BIT_KHR, hitShaderBindingTableAddress **must** not be zero
- VUID-VkTraceRaysIndirectCommand2KHR-flags-03511
  If the currently bound ray tracing pipeline was created with flags that included VK_PIPELINE_CREATE_RAY_TRACING_NO_NULL_MISS_SHADERS_BIT_KHR, the shader group handle identified by missShaderBindingTableAddress **must** not be set to zero
- VUID-VkTraceRaysIndirectCommand2KHR-flags-03512
  If the currently bound ray tracing pipeline was created with flags that included VK_PIPELINE_CREATE_RAY_TRACING_NO_NULL_ANY_HIT_SHADERS_BIT_KHR, entries in the table identified by hitShaderBindingTableAddress accessed as a result of this command in order to execute an any-hit shader **must** not be set to zero
- VUID-VkTraceRaysIndirectCommand2KHR-flags-03513
  If the currently bound ray tracing pipeline was created with flags that included VK_PIPELINE_CREATE_RAY_TRACING_NO_NULL_CLOSEST_HIT_SHADERS_BIT_KHR, entries in the table identified by hitShaderBindingTableAddress accessed as a result of this command in order to execute a closest hit shader **must** not be set to zero
- VUID-VkTraceRaysIndirectCommand2KHR-flags-03514
  If the currently bound ray tracing pipeline was created with flags that included VK_PIPELINE_CREATE_RAY_TRACING_NO_NULL_INTERSECTION_SHADERS_BIT_KHR, entries in the table identified by hitShaderBindingTableAddress accessed as a result of this command in order to execute an intersection shader **must** not be set to zero
- VUID-VkTraceRaysIndirectCommand2KHR-pHitShaderBindingTable-04735
  Any non-zero hit shader group entries in the table identified by hitShaderBindingTableAddress accessed by this call from a geometry with a geometryType of VK_GEOMETRY_TYPE_TRIANGLES_KHR **must** have been created with VK_RAY_TRACING_SHADER_GROUP_TYPE_TRIANGLES_HIT_GROUP_KHR
- VUID-VkTraceRaysIndirectCommand2KHR-pHitShaderBindingTable-04736
  Any non-zero hit shader group entries in the table identified by hitShaderBindingTableAddress accessed by this call from a geometry with a geometryType of VK_GEOMETRY_TYPE_AABBS_KHR **must** have been created with VK_RAY_TRACING_SHADER_GROUP_TYPE_PROCEDURAL_HIT_GROUP_KHR
- VUID-VkTraceRaysIndirectCommand2KHR-width-03638

width **must** be less than or equal to VkPhysicalDeviceLimits::maxComputeWorkGroupCount[0] × VkPhysicalDeviceLimits::maxComputeWorkGroupSize[0]

- VUID-VkTraceRaysIndirectCommand2KHR-height-03639
  height **must** be less than or equal to VkPhysicalDeviceLimits::maxComputeWorkGroupCount[1] × VkPhysicalDeviceLimits::maxComputeWorkGroupSize[1]
- VUID-VkTraceRaysIndirectCommand2KHR-depth-03640
  depth **must** be less than or equal to VkPhysicalDeviceLimits::maxComputeWorkGroupCount[2] × VkPhysicalDeviceLimits::maxComputeWorkGroupSize[2]
- VUID-VkTraceRaysIndirectCommand2KHR-width-03641
  width × height × depth **must** be less than or equal to VkPhysicalDeviceRayTracingPipelinePropertiesKHR::maxRayDispatchInvocationCount

## 40.3. Shader Binding Table

A *shader binding table* is a resource which establishes the relationship between the ray tracing pipeline and the acceleration structures that were built for the ray tracing pipeline. It indicates the shaders that operate on each geometry in an acceleration structure. In addition, it contains the resources accessed by each shader, including indices of textures, buffer device addresses, and constants. The application allocates and manages *shader binding tables* as VkBuffer objects.

Each entry in the shader binding table consists of shaderGroupHandleSize bytes of data, either as queried by vkGetRayTracingShaderGroupHandlesKHR to refer to those specified shaders, or all zeros to refer to a zero shader group. A zero shader group behaves as though it is a shader group consisting entirely of VK_SHADER_UNUSED_KHR. The remainder of the data specified by the stride is application-visible data that can be referenced by a ShaderRecordBufferKHR block in the shader.

The shader binding tables to use in a ray tracing pipeline are passed to the vkCmdTraceRaysNV, vkCmdTraceRaysKHR, or vkCmdTraceRaysIndirectKHR commands. Shader binding tables are read-only in shaders that are executing on the ray tracing pipeline.

Shader variables identified with the ShaderRecordBufferKHR storage class are used to access the provided shader binding table. Such variables **must** be:

- typed as OpTypeStruct, or an array of this type,
- identified with a Block decoration, and
- laid out explicitly using the Offset, ArrayStride, and MatrixStride decorations as specified in Offset and Stride Assignment.

The Offset decoration for any member of a Block-decorated variable in the ShaderRecordBufferKHR storage class **must** not cause the space required for that variable to extend outside the range [0, maxStorageBufferRange).

Accesses to the shader binding table from ray tracing pipelines **must** be synchronized with the VK_PIPELINE_STAGE_RAY_TRACING_SHADER_BIT_KHR pipeline stage and an access type of VK_ACCESS_SHADER_READ_BIT.

| | |
|---|---|
| **NOTE** | Because different shader record buffers can be associated with the same shader, a shader variable with `ShaderRecordBufferKHR` storage class will not be dynamically uniform if different invocations of the same shader can reference different data in the shader record buffer, such as if the same shader occurs twice in the shader binding table with a different shader record buffer. In this case, indexing resources based on values in the `ShaderRecordBufferKHR` storage class, the index should be decorated as `NonUniform`. |

### 40.3.1. Indexing Rules

In order to execute the correct shaders and access the correct resources during a ray tracing dispatch, the implementation **must** be able to locate shader binding table entries at various stages of execution. This is accomplished by defining a set of indexing rules that compute shader binding table record positions relative to the buffer's base address in memory. The application **must** organize the contents of the shader binding table's memory in a way that application of the indexing rules will lead to correct records.

#### Ray Generation Shaders

Only one ray generation shader is executed per ray tracing dispatch.

For vkCmdTraceRaysKHR, the location of the ray generation shader is specified by the `pRaygenShaderBindingTable->deviceAddress` parameter — there is no indexing. All data accessed **must** be less than `pRaygenShaderBindingTable->size` bytes from `deviceAddress`. `pRaygenShaderBindingTable->stride` is unused, and **must** be equal to `pRaygenShaderBindingTable->size`.

For vkCmdTraceRaysNV, the location of the ray generation shader is specified by the `raygenShaderBindingTableBuffer` and `raygenShaderBindingOffset` parameters — there is no indexing.

#### Hit Shaders

The base for the computation of intersection, any-hit, and closest hit shader locations is the `instanceShaderBindingTableRecordOffset` value stored with each instance of a top-level acceleration structure (VkAccelerationStructureInstanceKHR). This value determines the beginning of the shader binding table records for a given instance.

In the following rule, `geometryIndex` refers to the geometry index of the intersected geometry within the instance.

The `sbtRecordOffset` and `sbtRecordStride` values are passed in as parameters to `traceNV`() or `traceRayEXT`() calls made in the shaders. See Section 8.19 (Ray Tracing Functions) of the OpenGL Shading Language Specification for more details. In SPIR-V, these correspond to the `SBTOffset` and `SBTStride` parameters to the pipeline trace ray instructions.

The result of this computation is then added to `pHitShaderBindingTable->deviceAddress`, a device address passed to vkCmdTraceRaysKHR , or `hitShaderBindingOffset`, a base offset passed to vkCmdTraceRaysNV .

For vkCmdTraceRaysKHR, the complete rule to compute a hit shader binding table record address in the pHitShaderBindingTable is:

pHitShaderBindingTable->deviceAddress + pHitShaderBindingTable->stride × ( instanceShaderBindingTableRecordOffset + geometryIndex × sbtRecordStride + sbtRecordOffset )

All data accessed **must** be less than pHitShaderBindingTable->size bytes from the base address.

For vkCmdTraceRaysNV, the offset and stride come from direct parameters, so the full rule to compute a hit shader binding table record address in the hitShaderBindingTableBuffer is:

hitShaderBindingOffset + hitShaderBindingStride × ( instanceShaderBindingTableRecordOffset + geometryIndex × sbtRecordStride + sbtRecordOffset )

#### Miss Shaders

A miss shader is executed whenever a ray query fails to find an intersection for the given scene geometry. Multiple miss shaders **may** be executed throughout a ray tracing dispatch.

The base for the computation of miss shader locations is pMissShaderBindingTable->deviceAddress, a device address passed into vkCmdTraceRaysKHR , or missShaderBindingOffset, a base offset passed into vkCmdTraceRaysNV .

The missIndex value is passed in as a parameter to traceNV() or traceRayEXT() calls made in the shaders. See Section 8.19 (Ray Tracing Functions) of the OpenGL Shading Language Specification for more details. In SPIR-V, this corresponds to the MissIndex parameter to the pipeline trace ray instructions.

For vkCmdTraceRaysKHR, the complete rule to compute a miss shader binding table record address in the pMissShaderBindingTable is:

pMissShaderBindingTable->deviceAddress + pMissShaderBindingTable->stride × missIndex

All data accessed **must** be less than pMissShaderBindingTable->size bytes from the base address.

For vkCmdTraceRaysNV, the offset and stride come from direct parameters, so the full rule to compute a miss shader binding table record address in the missShaderBindingTableBuffer is:

missShaderBindingOffset + missShaderBindingStride × missIndex

#### Callable Shaders

A callable shader is executed when requested by a ray tracing shader. Multiple callable shaders **may** be executed throughout a ray tracing dispatch.

The base for the computation of callable shader locations is pCallableShaderBindingTable-

>deviceAddress, a device address passed into vkCmdTraceRaysKHR , or callableShaderBindingOffset, a base offset passed into vkCmdTraceRaysNV .

The sbtRecordIndex value is passed in as a parameter to executeCallableNV() or executeCallableEXT() calls made in the shaders. See Section 8.19 (Ray Tracing Functions) of the OpenGL Shading Language Specification for more details. In SPIR-V, this corresponds to the SBTIndex parameter to the OpExecuteCallableNV or OpExecuteCallableKHR instruction.

For vkCmdTraceRaysKHR, the complete rule to compute a callable shader binding table record address in the pCallableShaderBindingTable is:

pCallableShaderBindingTable->deviceAddress + pCallableShaderBindingTable->stride × sbtRecordIndex

All data accessed **must** be less than pCallableShaderBindingTable->size bytes from the base address.

For vkCmdTraceRaysNV, the offset and stride come from direct parameters, so the full rule to compute a callable shader binding table record address in the callableShaderBindingTableBuffer is:

callableShaderBindingOffset + callableShaderBindingStride × sbtRecordIndex

## 40.4. Ray Tracing Pipeline Stack

Ray tracing pipelines have a potentially large set of shaders which **may** be invoked in various call chain combinations to perform ray tracing. To store parameters for a given shader execution, an implementation **may** use a stack of data in memory. This stack **must** be sized to the sum of the stack sizes of all shaders in any call chain executed by the application.

If the stack size is not set explicitly, the stack size for a pipeline is:

rayGenStackMax + min(1, maxPipelineRayRecursionDepth) × max(closestHitStackMax, missStackMax, intersectionStackMax + anyHitStackMax) + max(0, maxPipelineRayRecursionDepth -1) × max(closestHitStackMax, missStackMax) + 2 × callableStackMax

where rayGenStackMax, closestHitStackMax, missStackMax, anyHitStackMax, intersectionStackMax, and callableStackMax are the maximum stack values queried by the respective shader stages for any shaders in any shader groups defined by the pipeline.

This stack size is potentially significant, so an application **may** want to provide a more accurate stack size after pipeline compilation. The value that the application provides is the maximum value of the sum of all shaders in a call chain across all possible call chains, taking into account any application specific knowledge about the properties of the call chains.

**NOTE** For example, if an application has two types of closest hit and miss shaders that it can use but the first level of rays will only use the first kind (possibly reflection) and the second level will only use the second kind (occlusion or shadow ray, for

example) then the application can compute the stack size by something similar to:

```
rayGenStack + max(closestHit1Stack, miss1Stack) + max(closestHit2Stack, miss2Stack)
```

This is guaranteed to be no larger than the default stack size computation which assumes that both call levels may be the larger of the two.

## 40.5. Ray Tracing Capture Replay

In a similar way to bufferDeviceAddressCaptureReplay, the `rayTracingPipelineShaderGroupHandleCaptureReplay` feature allows the querying of opaque data which **can** be used in a future replay.

During the capture phase, capture/replay tools are expected to query opaque data for shader group handle replay using vkGetRayTracingCaptureReplayShaderGroupHandlesKHR.

Providing the opaque data during replay, using VkRayTracingShaderGroupCreateInfoKHR::`pShaderGroupCaptureReplayHandle` at pipeline creation time, causes the implementation to generate identical shader group handles to those in the capture phase, allowing capture/replay tools to reuse previously recorded shader binding table buffer contents or to obtain the same handles by calling vkGetRayTracingCaptureReplayShaderGroupHandlesKHR again.

## 40.6. Ray Tracing Validation

Ray tracing validation **can** help root cause application issues and improve performance. Unlike existing validation layers, ray tracing validation performs checks at an implementation level, which helps identify potential problems that **may** not be caught by the layer.

By enabling the ray tracing validation feature, warnings and errors **can** be delivered straight from a ray tracing implementation to the application through a messenger callback registered with the implementation, where they **can** be processed through existing application-side debugging or logging systems.

# Chapter 41. Memory Decompression

To decompress data between one or more memory regions call:

```
// Provided by VK_NV_memory_decompression
void vkCmdDecompressMemoryNV(
    VkCommandBuffer                             commandBuffer,
    uint32_t                                    decompressRegionCount,
    const VkDecompressMemoryRegionNV*           pDecompressMemoryRegions);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `decompressRegionCount` is the number of memory regions to decompress.
- `pDecompressMemoryRegions` is a pointer to an array of `decompressRegionCount` VkDecompressMemoryRegionNV structures specifying decompression parameters.

Each region specified in `pDecompressMemoryRegions` is decompressed from the source to destination region based on the specified decompression method.

### Valid Usage

- VUID-vkCmdDecompressMemoryNV-None-07684
  The `memoryDecompression` feature **must** be enabled

### Valid Usage (Implicit)

- VUID-vkCmdDecompressMemoryNV-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdDecompressMemoryNV-pDecompressMemoryRegions-parameter
  `pDecompressMemoryRegions` **must** be a valid pointer to an array of `decompressRegionCount` valid VkDecompressMemoryRegionNV structures
- VUID-vkCmdDecompressMemoryNV-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdDecompressMemoryNV-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics, or compute operations
- VUID-vkCmdDecompressMemoryNV-renderpass
  This command **must** only be called outside of a render pass instance
- VUID-vkCmdDecompressMemoryNV-videocoding
  This command **must** only be called outside of a video coding scope
- VUID-vkCmdDecompressMemoryNV-decompressRegionCount-arraylength
  `decompressRegionCount` **must** be greater than `0`

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Outside | Outside | Graphics<br>Compute | Action |

The `VkDecompressMemoryRegionNV` structure is defined as:

```
// Provided by VK_NV_memory_decompression
typedef struct VkDecompressMemoryRegionNV {
    VkDeviceAddress                     srcAddress;
    VkDeviceAddress                     dstAddress;
    VkDeviceSize                        compressedSize;
    VkDeviceSize                        decompressedSize;
    VkMemoryDecompressionMethodFlagsNV  decompressionMethod;
} VkDecompressMemoryRegionNV;
```

- `srcAddress` is the address where compressed data is stored.
- `dstAddress` is the destination address where decompressed data will be written.
- `compressedSize` is the size of compressed data in bytes.
- `decompressedSize` is the size of decompressed data in bytes.
- `decompressionMethod` is a bitmask of `VkMemoryDecompressionMethodFlagBitsNV` with a single bit set specifying the method used to decompress data.

## Valid Usage

- VUID-VkDecompressMemoryRegionNV-srcAddress-07685
  The `srcAddress` **must** be 4 byte aligned
- VUID-VkDecompressMemoryRegionNV-srcAddress-07686
  The memory in range `srcAddress` and `srcAddress` + `compressedSize` **must** be valid and bound to a `VkDeviceMemory` object
- VUID-VkDecompressMemoryRegionNV-dstAddress-07687
  The `dstAddress` **must** be 4 byte aligned
- VUID-VkDecompressMemoryRegionNV-decompressionMethod-09395

If decompressionMethod is VK_MEMORY_DECOMPRESSION_METHOD_GDEFLATE_1_0_BIT_NV, then decompressedSize **must** be less than or equal to 65536 bytes

- VUID-VkDecompressMemoryRegionNV-dstAddress-07688
  The memory in range dstAddress and dstAddress + decompressedSize **must** be valid and bound to a VkDeviceMemory object
- VUID-VkDecompressMemoryRegionNV-decompressedSize-07689
  The decompressedSize **must** be large enough to hold the decompressed data based on the decompressionMethod
- VUID-VkDecompressMemoryRegionNV-decompressionMethod-07690
  The decompressionMethod **must** have a single bit set
- VUID-VkDecompressMemoryRegionNV-srcAddress-07691
  The srcAddress to srcAddress + compressedSize region **must** not overlap with the dstAddress and dstAddress + decompressedSize region

### Valid Usage (Implicit)

- VUID-VkDecompressMemoryRegionNV-decompressionMethod-parameter
  decompressionMethod **must** be a valid combination of VkMemoryDecompressionMethodFlagBitsNV values
- VUID-VkDecompressMemoryRegionNV-decompressionMethod-requiredbitmask
  decompressionMethod **must** not be 0

To decompress data between one or more memory regions by specifying decompression parameters indirectly in a buffer, call:

```
// Provided by VK_NV_memory_decompression
void vkCmdDecompressMemoryIndirectCountNV(
    VkCommandBuffer                             commandBuffer,
    VkDeviceAddress                             indirectCommandsAddress,
    VkDeviceAddress                             indirectCommandsCountAddress,
    uint32_t                                    stride);
```

- commandBuffer is the command buffer into which the command will be recorded.
- indirectCommandsAddress is the device address containing decompression parameters laid out as an array of VkDecompressMemoryRegionNV structures.
- indirectCommandsCountAddress is the device address containing the decompression count.
- stride is the byte stride between successive sets of decompression parameters located starting from indirectCommandsAddress.

Each region specified in indirectCommandsAddress is decompressed from the source to destination region based on the specified decompression method.

## Valid Usage

- VUID-vkCmdDecompressMemoryIndirectCountNV-None-07692
  The `memoryDecompression` feature **must** be enabled
- VUID-vkCmdDecompressMemoryIndirectCountNV-indirectCommandsAddress-07693
  If `indirectCommandsAddress` comes from a non-sparse buffer then it **must** be bound completely and contiguously to a single `VkDeviceMemory` object
- VUID-vkCmdDecompressMemoryIndirectCountNV-indirectCommandsAddress-07694
  The VkBuffer that `indirectCommandsAddress` comes from **must** have been created with the `VK_BUFFER_USAGE_INDIRECT_BUFFER_BIT` bit set
- VUID-vkCmdDecompressMemoryIndirectCountNV-offset-07695
  `offset` **must** be a multiple of `4`
- VUID-vkCmdDecompressMemoryIndirectCountNV-indirectCommandsCountAddress-07696
  If `indirectCommandsCountAddress` comes from a non-sparse buffer then it **must** be bound completely and contiguously to a single `VkDeviceMemory` object
- VUID-vkCmdDecompressMemoryIndirectCountNV-indirectCommandsCountAddress-07697
  The VkBuffer that `indirectCommandsCountAddress` comes from **must** have been created with the `VK_BUFFER_USAGE_INDIRECT_BUFFER_BIT` bit set
- VUID-vkCmdDecompressMemoryIndirectCountNV-indirectCommandsCountAddress-07698
  `indirectCommandsCountAddress` **must** be a multiple of `4`
- VUID-vkCmdDecompressMemoryIndirectCountNV-indirectCommandsCountAddress-07699
  The count stored in `indirectCommandsCountAddress` **must** be less than or equal to `VkPhysicalDeviceMemoryDecompressionPropertiesNV::maxDecompressionIndirectCount`
- VUID-vkCmdDecompressMemoryIndirectCountNV-stride-07700
  `stride` **must** be a multiple of `4` and **must** be greater than or equal to sizeof(`VkDecompressMemoryRegionNV`)
- VUID-vkCmdDecompressMemoryIndirectCountNV-indirectCommandsCountAddress-07701
  If the count stored in `indirectCommandsCountAddress` is equal to `1`, (`offset` + sizeof(`VkDecompressMemoryRegionNV`)) **must** be less than or equal to the size of the VkBuffer that `indirectCommandsAddress` comes from
- VUID-vkCmdDecompressMemoryIndirectCountNV-indirectCommandsCountAddress-07702
  If the count stored in `indirectCommandsCountAddress` is greater than `1`, `indirectCommandsAddress` + sizeof(`VkDecompressMemoryRegionNV`) + (`stride` × (count stored in `countBuffer` - 1)) **must** be less than or equal to the last valid address in the VkBuffer that `indirectCommandsAddress` was created from

## Valid Usage (Implicit)

- VUID-vkCmdDecompressMemoryIndirectCountNV-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdDecompressMemoryIndirectCountNV-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdDecompressMemoryIndirectCountNV-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics, or compute operations
- VUID-vkCmdDecompressMemoryIndirectCountNV-renderpass
  This command **must** only be called outside of a render pass instance
- VUID-vkCmdDecompressMemoryIndirectCountNV-videocoding
  This command **must** only be called outside of a video coding scope

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Outside | Outside | Graphics<br>Compute | Action |

Bits which **can** be set in `VkDecompressMemoryRegionNV::decompressionMethod` specifying the decompression method to select, or returned in `VkPhysicalDeviceMemoryDecompressionPropertiesNV::decompressionMethods` specifying the available decompression methods are:

```
// Provided by VK_NV_memory_decompression
// Flag bits for VkMemoryDecompressionMethodFlagBitsNV
typedef VkFlags64 VkMemoryDecompressionMethodFlagBitsNV;
static const VkMemoryDecompressionMethodFlagBitsNV
VK_MEMORY_DECOMPRESSION_METHOD_GDEFLATE_1_0_BIT_NV = 0x00000001ULL;
```

- `VK_MEMORY_DECOMPRESSION_METHOD_GDEFLATE_1_0_BIT_NV` specifies that the GDeflate 1.0 algorithm is used to decompress data.

```
// Provided by VK_NV_memory_decompression
typedef VkFlags64 VkMemoryDecompressionMethodFlagsNV;
```

VkMemoryDecompressionMethodFlagsNV is a bitmask type for specifying a mask of one or more VkMemoryDecompressionMethodFlagBitsNV:

# Chapter 42. Video Coding

Vulkan implementations **may** expose one or more queue families supporting video coding operations. These operations are performed by recording them into a command buffer within a video coding scope, and submitting them to queues with compatible video coding capabilities.

The Vulkan video functionalities are designed to be made available through a set of APIs built on top of each other, consisting of:

- A core API providing common video coding functionalities,
- APIs providing codec-independent video decode and video encode related functionalities, respectively,
- Additional codec-specific APIs built on top of those.

This chapter details the fundamental components and operations of these.

## 42.1. Video Picture Resources

In the context of video coding, multidimensional arrays of image data that **can** be used as the source or target of video coding operations are referred to as *video picture resources*. They **may** store additional metadata that includes implementation-private information used during the execution of video coding operations, as discussed later.

Video picture resources are backed by VkImage objects. Individual subregions of VkImageView objects created from such resources **can** be used as decode output pictures, encode input pictures, reconstructed pictures, and/or reference pictures.

The parameters of a video picture resource are specified using a `VkVideoPictureResourceInfoKHR` structure.

The `VkVideoPictureResourceInfoKHR` structure is defined as:

```
// Provided by VK_KHR_video_queue
typedef struct VkVideoPictureResourceInfoKHR {
    VkStructureType    sType;
    const void*        pNext;
    VkOffset2D         codedOffset;
    VkExtent2D         codedExtent;
    uint32_t           baseArrayLayer;
    VkImageView        imageViewBinding;
} VkVideoPictureResourceInfoKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `codedOffset` is the offset in texels of the image subregion to use.
- `codedExtent` is the size in pixels of the coded image data.

- `baseArrayLayer` is the array layer of the image view specified in `imageViewBinding` to use as the video picture resource.
- `imageViewBinding` is an image view representing the video picture resource.

The image subresource referred to by such a structure is defined as the image array layer index specified in `baseArrayLayer` relative to the image subresource range the image view specified in `imageViewBinding` was created with.

The meaning of the `codedOffset` and `codedExtent` depends on the command and context the video picture resource is used in, as well as on the used video profile and corresponding codec-specific semantics, as described later.

A video picture resource is uniquely defined by the image subresource referred to by an instance of this structure, together with the `codedOffset` and `codedExtent` members that identify the image subregion within the image subresource referenced corresponding to the video picture resource according to the particular codec-specific semantics.

Accesses to image data within a video picture resource happen at the granularity indicated by VkVideoCapabilitiesKHR::`pictureAccessGranularity`, as returned by vkGetPhysicalDeviceVideoCapabilitiesKHR for the used video profile. As a result, given an effective image subregion corresponding to a video picture resource, the actual image subregion accessed **may** be larger than that as it **may** include additional padding texels due to the picture access granularity. Any writes performed by video coding operations to such padding texels will result in undefined texel values.

Two video picture resources match if they refer to the same image subresource and they specify identical `codedOffset` and `codedExtent` values.

### Valid Usage

- VUID-VkVideoPictureResourceInfoKHR-baseArrayLayer-07175
  `baseArrayLayer` **must** be less than the VkImageViewCreateInfo::`subresourceRange.layerCount` specified when the image view `imageViewBinding` was created

### Valid Usage (Implicit)

- VUID-VkVideoPictureResourceInfoKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_VIDEO_PICTURE_RESOURCE_INFO_KHR`
- VUID-VkVideoPictureResourceInfoKHR-pNext-pNext
  `pNext` **must** be `NULL`
- VUID-VkVideoPictureResourceInfoKHR-imageViewBinding-parameter
  `imageViewBinding` **must** be a valid VkImageView handle

## 42.2. Decoded Picture Buffer

An integral part of video coding pipelines is the reconstruction of pictures from a compressed video bitstream. A *reconstructed picture* is a video picture resource resulting from this process.

Such reconstructed pictures **can** be used as *reference pictures* in subsequent video coding operations to provide predictions of the values of samples of subsequently decoded or encoded pictures. The correct use of such reconstructed pictures as reference pictures is driven by the video compression standard, the implementation, and the application-specific use cases.

The list of reference pictures used to provide such predictions within a single video coding operation is referred to as the list of *active reference pictures*.

The *decoded picture buffer (DPB)* is an indexed data structure that maintains the set of reference pictures available to be used in video coding operations. Individual indexed entries of the DPB are referred to as the *decoded picture buffer (DPB) slots*. The range of valid DPB slot indices is between zero and `N-1`, where `N` is the capacity of the DPB. Each DPB slot **can** refer to a reference picture containing a *video frame* or **can** refer to up to two reference pictures containing the top and/or bottom *fields* that, when both present, together represent a full *video frame* .

In Vulkan, the state and the backing store of the DPB is separated as follows:

- The state of individual DPB slots is maintained by video session objects.
- The backing store of DPB slots is provided by subregions of VkImage objects used as video picture resources.

In addition, the implementation **may** also maintain opaque metadata associated with DPB slots, including:

- *Reference picture metadata* corresponding to the video picture resource associated with the DPB slot.

Such metadata **may** be stored by the implementation as part of the DPB slot state maintained by the video session, or as part of the video picture resource backing the DPB slot.

Any metadata stored in the video picture resources backing DPB slots are independent of the video session used to store it, hence such video picture resources **can** be shared with other video sessions. Correspondingly, any metadata that is dependent on the video session will always be stored as part of the DPB slot state maintained by that video session.

The responsibility of managing the DPB is split between the application and the implementation as follows:

- The application maintains the association between DPB slot indices and corresponding video picture resources.
- The implementation maintains global and per-slot opaque reference picture metadata.

In addition, the application is also responsible for managing the mapping between the codec-specific picture IDs and DPB slots, and any other codec-specific states unless otherwise specified.

### 42.2.1. DPB Slot States

At a given time, each DPB slot is either in *active* or *inactive* state. Initially, all DPB slots managed by a video session are in *inactive* state.

A DPB slot **can** be *activated* by using it as the target of picture reconstruction in a video coding operation with the reconstructed picture requested to be set up as a reference picture, according to the codec-specific semantics, changing its state to *active* and associating it with a *picture reference* to the reconstructed pictures.

Some video coding standards allow multiple picture references to be associated with a single DPB slot. In this case the state of the individual picture references **can** be independently updated.

| | |
|---|---|
| **NOTE** | As an example, H.264 decoding allows associating a separate top field and bottom field picture with the same DPB slot. |

As part of reference picture setup, the implementation **may** also generate reference picture metadata. Such reference picture metadata is specific to each picture reference associated with the DPB slot.

If such a video coding operation completes successfully, the activated DPB slot will have a *valid picture reference* and the reconstructed picture is associated with the DPB slot. This is true even if the DPB slot is used as the target of a picture reconstruction that only sets up a top field or bottom field reference picture and thus does not yet refer to a complete frame. However, if any data provided as input to such a video coding operation is not compliant with the video compression standard used, that video coding operation **may** complete unsuccessfully, in which case the activated DPB slot will have an *invalid picture reference*. This is true even if the DPB slot previously had a valid picture reference to a top field or bottom field reference picture, but the reconstruction of the other field corresponding to the DPB slot failed.

The application **can** use queries to get feedback about the outcome of video coding operations and use the resulting VkQueryResultStatusKHR value to determine whether the video coding operation completed successfully (result status is positive) or unsuccessfully (result status is negative).

Using a reference picture associated with a DPB slot that has an *invalid picture reference* as an active reference picture in subsequent video coding operations is legal, however, the contents of the outputs of such operations are undefined, and any DPB slots activated by such video coding operations will also have an *invalid picture reference*. This is true even if such video coding operations **may** otherwise complete successfully.

A DPB slot **can** also be *deactivated* by the application, changing its state to *inactive* and invalidating any picture references and reference picture metadata associated with the DPB slot.

If an already active DPB slot is used as the target of picture reconstruction in a video coding operation, but the decoded picture is not requested to be set up as a reference picture, according to the codec-specific semantics, no reference picture setup happens and the corresponding picture reference and reference picture metadata is invalidated within the DPB slot. If the DPB slot no longer has any associated picture references after such an operation, the DPB slot is implicitly deactivated.

If an already active DPB slot is used as the target of picture reconstruction when decoding a field picture that is not marked as reference, then the behavior is as follows:

- If the DPB slot is currently associated with a frame, then the DPB slot is deactivated.
- If the DPB slot is not currently associated with a top field picture and the decoded picture is a top field picture, or if the DPB slot is not currently associated with a bottom field picture and the decoded picture is a bottom field picture, then the other field picture association of the DPB slot, if any, is not disturbed.
- If the DPB slot is currently associated with a top field picture and the decoded picture is a top field picture, or if the DPB slot is currently associated with a bottom field picture and the decoded picture is a bottom field picture, then that picture association is invalidated, without disturbing the other field picture association, if any. If the DPB slot no longer has any associated picture references after such an operation, the DPB slot is implicitly deactivated.

A DPB slot **can** be activated with a new frame even if it is already active. In this case all previous associations of the DPB slots with reference pictures are replaced with an association with the reconstructed picture used to activate it.

If an already active DPB slot is activated with a reconstructed field picture, then the behavior is as follows:

- If the DPB slot is currently associated with a frame, then that association is replaced with an association with the reconstructed field picture used to activate it.
- If the DPB slot is not currently associated with a top field picture and the DPB slot is activated with a top field picture, or if the DPB slot is not currently associated with a bottom field picture and the DPB slot is activated with a bottom field picture, then the DPB slot is associated with the reconstructed field picture used to activate it, without disturbing the other field picture association, if any.
- If the DPB slot is currently associated with a top field picture and the DPB slot is activated with a new top field picture, or if the DPB slot is currently associated with a bottom field picture and the DPB slot is activated with a new bottom field picture, then that association is replaced with an association with the reconstructed field picture used to activate it, without disturbing the other field picture association, if any.

## 42.3. Video Profiles

The `VkVideoProfileInfoKHR` structure is defined as follows:

```
// Provided by VK_KHR_video_queue
typedef struct VkVideoProfileInfoKHR {
    VkStructureType                     sType;
    const void*                         pNext;
    VkVideoCodecOperationFlagBitsKHR    videoCodecOperation;
    VkVideoChromaSubsamplingFlagsKHR    chromaSubsampling;
    VkVideoComponentBitDepthFlagsKHR    lumaBitDepth;
    VkVideoComponentBitDepthFlagsKHR    chromaBitDepth;
} VkVideoProfileInfoKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `videoCodecOperation` is a VkVideoCodecOperationFlagBitsKHR value specifying a video codec operation.
- `chromaSubsampling` is a bitmask of VkVideoChromaSubsamplingFlagBitsKHR specifying video chroma subsampling information.
- `lumaBitDepth` is a bitmask of VkVideoComponentBitDepthFlagBitsKHR specifying video luma bit depth information.
- `chromaBitDepth` is a bitmask of VkVideoComponentBitDepthFlagBitsKHR specifying video chroma bit depth information.

Video profiles are provided as input to video capability queries such as vkGetPhysicalDeviceVideoCapabilitiesKHR or vkGetPhysicalDeviceVideoFormatPropertiesKHR, as well as when creating resources to be used by video coding operations such as images, buffers, query pools, and video sessions.

The full description of a video profile is specified by an instance of this structure, and the codec-specific and auxiliary structures provided in its `pNext` chain.

When this structure is specified as an input parameter to vkGetPhysicalDeviceVideoCapabilitiesKHR, or through the `pProfiles` member of a VkVideoProfileListInfoKHR structure in the `pNext` chain of the input parameter of a query command such as vkGetPhysicalDeviceVideoFormatPropertiesKHR or vkGetPhysicalDeviceImageFormatProperties2, the following error codes indicate specific causes of the failure of the query operation:

- `VK_ERROR_VIDEO_PICTURE_LAYOUT_NOT_SUPPORTED_KHR` indicates that the requested video picture layout (e.g. through the `pictureLayout` member of a VkVideoDecodeH264ProfileInfoKHR structure included in the `pNext` chain of `VkVideoProfileInfoKHR`) is not supported.
- `VK_ERROR_VIDEO_PROFILE_OPERATION_NOT_SUPPORTED_KHR` indicates that a video profile operation specified by `videoCodecOperation` is not supported.
- `VK_ERROR_VIDEO_PROFILE_FORMAT_NOT_SUPPORTED_KHR` indicates that video format parameters specified by `chromaSubsampling`, `lumaBitDepth`, or `chromaBitDepth` are not supported.
- `VK_ERROR_VIDEO_PROFILE_CODEC_NOT_SUPPORTED_KHR` indicates that the codec-specific parameters corresponding to the video codec operation are not supported.

## Valid Usage

- VUID-VkVideoProfileInfoKHR-chromaSubsampling-07013
  `chromaSubsampling` **must** have a single bit set
- VUID-VkVideoProfileInfoKHR-lumaBitDepth-07014
  `lumaBitDepth` **must** have a single bit set
- VUID-VkVideoProfileInfoKHR-chromaSubsampling-07015
  If `chromaSubsampling` is not `VK_VIDEO_CHROMA_SUBSAMPLING_MONOCHROME_BIT_KHR`, then `chromaBitDepth` **must** have a single bit set
- VUID-VkVideoProfileInfoKHR-videoCodecOperation-07179
  If `videoCodecOperation` is `VK_VIDEO_CODEC_OPERATION_DECODE_H264_BIT_KHR`, then the `pNext` chain **must** include a VkVideoDecodeH264ProfileInfoKHR structure
- VUID-VkVideoProfileInfoKHR-videoCodecOperation-07180
  If `videoCodecOperation` is `VK_VIDEO_CODEC_OPERATION_DECODE_H265_BIT_KHR`, then the `pNext` chain **must** include a VkVideoDecodeH265ProfileInfoKHR structure
- VUID-VkVideoProfileInfoKHR-videoCodecOperation-09256
  If `videoCodecOperation` is `VK_VIDEO_CODEC_OPERATION_DECODE_AV1_BIT_KHR`, then the `pNext` chain **must** include a VkVideoDecodeAV1ProfileInfoKHR structure
- VUID-VkVideoProfileInfoKHR-videoCodecOperation-07181
  If `videoCodecOperation` is `VK_VIDEO_CODEC_OPERATION_ENCODE_H264_BIT_KHR`, then the `pNext` chain **must** include a VkVideoEncodeH264ProfileInfoKHR structure
- VUID-VkVideoProfileInfoKHR-videoCodecOperation-07182
  If `videoCodecOperation` is `VK_VIDEO_CODEC_OPERATION_ENCODE_H265_BIT_KHR`, then the `pNext` chain **must** include a VkVideoEncodeH265ProfileInfoKHR structure

## Valid Usage (Implicit)

- VUID-VkVideoProfileInfoKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_VIDEO_PROFILE_INFO_KHR`
- VUID-VkVideoProfileInfoKHR-videoCodecOperation-parameter
  `videoCodecOperation` **must** be a valid VkVideoCodecOperationFlagBitsKHR value
- VUID-VkVideoProfileInfoKHR-chromaSubsampling-parameter
  `chromaSubsampling` **must** be a valid combination of VkVideoChromaSubsamplingFlagBitsKHR values
- VUID-VkVideoProfileInfoKHR-chromaSubsampling-requiredbitmask
  `chromaSubsampling` **must** not be `0`
- VUID-VkVideoProfileInfoKHR-lumaBitDepth-parameter
  `lumaBitDepth` **must** be a valid combination of VkVideoComponentBitDepthFlagBitsKHR values
- VUID-VkVideoProfileInfoKHR-lumaBitDepth-requiredbitmask
  `lumaBitDepth` **must** not be `0`

- VUID-VkVideoProfileInfoKHR-chromaBitDepth-parameter
  `chromaBitDepth` **must** be a valid combination of VkVideoComponentBitDepthFlagBitsKHR values

Possible values of VkVideoProfileInfoKHR::`videoCodecOperation`, specifying the type of video coding operation and video compression standard used by a video profile, are:

```
// Provided by VK_KHR_video_queue
typedef enum VkVideoCodecOperationFlagBitsKHR {
    VK_VIDEO_CODEC_OPERATION_NONE_KHR = 0,
  // Provided by VK_KHR_video_encode_h264
    VK_VIDEO_CODEC_OPERATION_ENCODE_H264_BIT_KHR = 0x00010000,
  // Provided by VK_KHR_video_encode_h265
    VK_VIDEO_CODEC_OPERATION_ENCODE_H265_BIT_KHR = 0x00020000,
  // Provided by VK_KHR_video_decode_h264
    VK_VIDEO_CODEC_OPERATION_DECODE_H264_BIT_KHR = 0x00000001,
  // Provided by VK_KHR_video_decode_h265
    VK_VIDEO_CODEC_OPERATION_DECODE_H265_BIT_KHR = 0x00000002,
  // Provided by VK_KHR_video_decode_av1
    VK_VIDEO_CODEC_OPERATION_DECODE_AV1_BIT_KHR = 0x00000004,
} VkVideoCodecOperationFlagBitsKHR;
```

- `VK_VIDEO_CODEC_OPERATION_NONE_KHR` indicates no support for any video codec operations.
- `VK_VIDEO_CODEC_OPERATION_DECODE_H264_BIT_KHR` specifies support for H.264 decode operations.
- `VK_VIDEO_CODEC_OPERATION_DECODE_H265_BIT_KHR` specifies support for H.265 decode operations.
- `VK_VIDEO_CODEC_OPERATION_DECODE_AV1_BIT_KHR` specifies support for AV1 decode operations.
- `VK_VIDEO_CODEC_OPERATION_ENCODE_H264_BIT_KHR` specifies support for H.264 encode operations.
- `VK_VIDEO_CODEC_OPERATION_ENCODE_H265_BIT_KHR` specifies support for H.265 encode operations.

```
// Provided by VK_KHR_video_queue
typedef VkFlags VkVideoCodecOperationFlagsKHR;
```

`VkVideoCodecOperationFlagsKHR` is a bitmask type for setting a mask of zero or more VkVideoCodecOperationFlagBitsKHR.

The video format chroma subsampling is defined with the following enums:

```
// Provided by VK_KHR_video_queue
typedef enum VkVideoChromaSubsamplingFlagBitsKHR {
    VK_VIDEO_CHROMA_SUBSAMPLING_INVALID_KHR = 0,
    VK_VIDEO_CHROMA_SUBSAMPLING_MONOCHROME_BIT_KHR = 0x00000001,
    VK_VIDEO_CHROMA_SUBSAMPLING_420_BIT_KHR = 0x00000002,
    VK_VIDEO_CHROMA_SUBSAMPLING_422_BIT_KHR = 0x00000004,
    VK_VIDEO_CHROMA_SUBSAMPLING_444_BIT_KHR = 0x00000008,
} VkVideoChromaSubsamplingFlagBitsKHR;
```

- VK_VIDEO_CHROMA_SUBSAMPLING_MONOCHROME_BIT_KHR specifies that the format is monochrome.
- VK_VIDEO_CHROMA_SUBSAMPLING_420_BIT_KHR specified that the format is 4:2:0 chroma subsampled, i.e. the two chroma components are sampled horizontally and vertically at half the sample rate of the luma component.
- VK_VIDEO_CHROMA_SUBSAMPLING_422_BIT_KHR - the format is 4:2:2 chroma subsampled, i.e. the two chroma components are sampled horizontally at half the sample rate of luma component.
- VK_VIDEO_CHROMA_SUBSAMPLING_444_BIT_KHR - the format is 4:4:4 chroma sampled, i.e. all three components of the $Y'C_BC_R$ format are sampled at the same rate, thus there is no chroma subsampling.

Chroma subsampling is described in more detail in the Chroma Reconstruction section.

```
// Provided by VK_KHR_video_queue
typedef VkFlags VkVideoChromaSubsamplingFlagsKHR;
```

VkVideoChromaSubsamplingFlagsKHR is a bitmask type for setting a mask of zero or more VkVideoChromaSubsamplingFlagBitsKHR.

Possible values for the video format component bit depth are:

```
// Provided by VK_KHR_video_queue
typedef enum VkVideoComponentBitDepthFlagBitsKHR {
    VK_VIDEO_COMPONENT_BIT_DEPTH_INVALID_KHR = 0,
    VK_VIDEO_COMPONENT_BIT_DEPTH_8_BIT_KHR = 0x00000001,
    VK_VIDEO_COMPONENT_BIT_DEPTH_10_BIT_KHR = 0x00000004,
    VK_VIDEO_COMPONENT_BIT_DEPTH_12_BIT_KHR = 0x00000010,
} VkVideoComponentBitDepthFlagBitsKHR;
```

- VK_VIDEO_COMPONENT_BIT_DEPTH_8_BIT_KHR specifies a component bit depth of 8 bits.
- VK_VIDEO_COMPONENT_BIT_DEPTH_10_BIT_KHR specifies a component bit depth of 10 bits.
- VK_VIDEO_COMPONENT_BIT_DEPTH_12_BIT_KHR specifies a component bit depth of 12 bits.

```
// Provided by VK_KHR_video_queue
typedef VkFlags VkVideoComponentBitDepthFlagsKHR;
```

VkVideoComponentBitDepthFlagsKHR is a bitmask type for setting a mask of zero or more VkVideoComponentBitDepthFlagBitsKHR.

Additional information about the video decode use case **can** be provided by adding a VkVideoDecodeUsageInfoKHR structure to the pNext chain of VkVideoProfileInfoKHR.

The VkVideoDecodeUsageInfoKHR structure is defined as:

```
// Provided by VK_KHR_video_decode_queue
typedef struct VkVideoDecodeUsageInfoKHR {
    VkStructureType                 sType;
    const void*                     pNext;
    VkVideoDecodeUsageFlagsKHR      videoUsageHints;
} VkVideoDecodeUsageInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- videoUsageHints is a bitmask of VkVideoDecodeUsageFlagBitsKHR specifying hints about the intended use of the video decode profile.

### Valid Usage (Implicit)

- VUID-VkVideoDecodeUsageInfoKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_VIDEO_DECODE_USAGE_INFO_KHR
- VUID-VkVideoDecodeUsageInfoKHR-videoUsageHints-parameter
  videoUsageHints **must** be a valid combination of VkVideoDecodeUsageFlagBitsKHR values

The following bits **can** be specified in VkVideoDecodeUsageInfoKHR::videoUsageHints as a hint about the video decode use case:

```
// Provided by VK_KHR_video_decode_queue
typedef enum VkVideoDecodeUsageFlagBitsKHR {
    VK_VIDEO_DECODE_USAGE_DEFAULT_KHR = 0,
    VK_VIDEO_DECODE_USAGE_TRANSCODING_BIT_KHR = 0x00000001,
    VK_VIDEO_DECODE_USAGE_OFFLINE_BIT_KHR = 0x00000002,
    VK_VIDEO_DECODE_USAGE_STREAMING_BIT_KHR = 0x00000004,
} VkVideoDecodeUsageFlagBitsKHR;
```

- VK_VIDEO_DECODE_USAGE_TRANSCODING_BIT_KHR specifies that video decoding is intended to be used in conjunction with video encoding to transcode a video bitstream with the same and/or different codecs.
- VK_VIDEO_DECODE_USAGE_OFFLINE_BIT_KHR specifies that video decoding is intended to be used to consume a local video bitstream.
- VK_VIDEO_DECODE_USAGE_STREAMING_BIT_KHR specifies that video decoding is intended to be used to

consume a video bitstream received as a continuous flow over network.

| | |
|---|---|
| **NOTE** | There are no restrictions on the combination of bits that **can** be specified by the application. However, applications **should** use reasonable combinations in order for the implementation to be able to select the most appropriate mode of operation for the particular use case. |

```
// Provided by VK_KHR_video_decode_queue
typedef VkFlags VkVideoDecodeUsageFlagsKHR;
```

`VkVideoDecodeUsageFlagsKHR` is a bitmask type for setting a mask of zero or more VkVideoDecodeUsageFlagBitsKHR.

Additional information about the video encode use case **can** be provided by adding a `VkVideoEncodeUsageInfoKHR` structure to the `pNext` chain of VkVideoProfileInfoKHR.

The `VkVideoEncodeUsageInfoKHR` structure is defined as:

```
// Provided by VK_KHR_video_encode_queue
typedef struct VkVideoEncodeUsageInfoKHR {
    VkStructureType                 sType;
    const void*                     pNext;
    VkVideoEncodeUsageFlagsKHR      videoUsageHints;
    VkVideoEncodeContentFlagsKHR    videoContentHints;
    VkVideoEncodeTuningModeKHR      tuningMode;
} VkVideoEncodeUsageInfoKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `videoUsageHints` is a bitmask of VkVideoEncodeUsageFlagBitsKHR specifying hints about the intended use of the video encode profile.
- `videoContentHints` is a bitmask of VkVideoEncodeContentFlagBitsKHR specifying hints about the content to be encoded using the video encode profile.
- `tuningMode` is a VkVideoEncodeTuningModeKHR value specifying the tuning mode to use when encoding with the video profile.

### Valid Usage (Implicit)

- VUID-VkVideoEncodeUsageInfoKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_VIDEO_ENCODE_USAGE_INFO_KHR`
- VUID-VkVideoEncodeUsageInfoKHR-videoUsageHints-parameter
  `videoUsageHints` **must** be a valid combination of VkVideoEncodeUsageFlagBitsKHR values
- VUID-VkVideoEncodeUsageInfoKHR-videoContentHints-parameter
  `videoContentHints` **must** be a valid combination of VkVideoEncodeContentFlagBitsKHR

values

- VUID-VkVideoEncodeUsageInfoKHR-tuningMode-parameter
  If `tuningMode` is not `0`, `tuningMode` **must** be a valid VkVideoEncodeTuningModeKHR value

The following bits **can** be specified in VkVideoEncodeUsageInfoKHR::`videoUsageHints` as a hint about the video encode use case:

```
// Provided by VK_KHR_video_encode_queue
typedef enum VkVideoEncodeUsageFlagBitsKHR {
    VK_VIDEO_ENCODE_USAGE_DEFAULT_KHR = 0,
    VK_VIDEO_ENCODE_USAGE_TRANSCODING_BIT_KHR = 0x00000001,
    VK_VIDEO_ENCODE_USAGE_STREAMING_BIT_KHR = 0x00000002,
    VK_VIDEO_ENCODE_USAGE_RECORDING_BIT_KHR = 0x00000004,
    VK_VIDEO_ENCODE_USAGE_CONFERENCING_BIT_KHR = 0x00000008,
} VkVideoEncodeUsageFlagBitsKHR;
```

- `VK_VIDEO_ENCODE_USAGE_TRANSCODING_BIT_KHR` specifies that video encoding is intended to be used in conjunction with video decoding to transcode a video bitstream with the same and/or different codecs.
- `VK_VIDEO_ENCODE_USAGE_STREAMING_BIT_KHR` specifies that video encoding is intended to be used to produce a video bitstream that is expected to be sent as a continuous flow over network.
- `VK_VIDEO_ENCODE_USAGE_RECORDING_BIT_KHR` specifies that video encoding is intended to be used for real-time recording for offline consumption.
- `VK_VIDEO_ENCODE_USAGE_CONFERENCING_BIT_KHR` specifies that video encoding is intended to be used in a video conferencing scenario.

**NOTE** There are no restrictions on the combination of bits that **can** be specified by the application. However, applications **should** use reasonable combinations in order for the implementation to be able to select the most appropriate mode of operation for the particular use case.

```
// Provided by VK_KHR_video_encode_queue
typedef VkFlags VkVideoEncodeUsageFlagsKHR;
```

`VkVideoEncodeUsageFlagsKHR` is a bitmask type for setting a mask of zero or more VkVideoEncodeUsageFlagBitsKHR.

The following bits **can** be specified in VkVideoEncodeUsageInfoKHR::`videoContentHints` as a hint about the encoded video content:

```
// Provided by VK_KHR_video_encode_queue
typedef enum VkVideoEncodeContentFlagBitsKHR {
    VK_VIDEO_ENCODE_CONTENT_DEFAULT_KHR = 0,
    VK_VIDEO_ENCODE_CONTENT_CAMERA_BIT_KHR = 0x00000001,
    VK_VIDEO_ENCODE_CONTENT_DESKTOP_BIT_KHR = 0x00000002,
    VK_VIDEO_ENCODE_CONTENT_RENDERED_BIT_KHR = 0x00000004,
} VkVideoEncodeContentFlagBitsKHR;
```

- VK_VIDEO_ENCODE_CONTENT_CAMERA_BIT_KHR specifies that video encoding is intended to be used to encode camera content.
- VK_VIDEO_ENCODE_CONTENT_DESKTOP_BIT_KHR specifies that video encoding is intended to be used to encode desktop content.
- VK_VIDEO_ENCODE_CONTENT_RENDERED_BIT_KHR specified that video encoding is intended to be used to encode rendered (e.g. game) content.

**NOTE** There are no restrictions on the combination of bits that **can** be specified by the application. However, applications **should** use reasonable combinations in order for the implementation to be able to select the most appropriate mode of operation for the particular content type.

```
// Provided by VK_KHR_video_encode_queue
typedef VkFlags VkVideoEncodeContentFlagsKHR;
```

VkVideoEncodeContentFlagsKHR is a bitmask type for setting a mask of zero or more VkVideoEncodeContentFlagBitsKHR.

Possible video encode tuning mode values are as follows:

```
// Provided by VK_KHR_video_encode_queue
typedef enum VkVideoEncodeTuningModeKHR {
    VK_VIDEO_ENCODE_TUNING_MODE_DEFAULT_KHR = 0,
    VK_VIDEO_ENCODE_TUNING_MODE_HIGH_QUALITY_KHR = 1,
    VK_VIDEO_ENCODE_TUNING_MODE_LOW_LATENCY_KHR = 2,
    VK_VIDEO_ENCODE_TUNING_MODE_ULTRA_LOW_LATENCY_KHR = 3,
    VK_VIDEO_ENCODE_TUNING_MODE_LOSSLESS_KHR = 4,
} VkVideoEncodeTuningModeKHR;
```

- VK_VIDEO_ENCODE_TUNING_MODE_DEFAULT_KHR specifies the default tuning mode.
- VK_VIDEO_ENCODE_TUNING_MODE_HIGH_QUALITY_KHR specifies that video encoding is tuned for high quality. When using this tuning mode, the implementation **may** compromise the latency of video encoding operations to improve quality.
- VK_VIDEO_ENCODE_TUNING_MODE_LOW_LATENCY_KHR specifies that video encoding is tuned for low latency. When using this tuning mode, the implementation **may** compromise quality to increase the performance and lower the latency of video encode operations.

- VK_VIDEO_ENCODE_TUNING_MODE_ULTRA_LOW_LATENCY_KHR specifies that video encoding is tuned for ultra-low latency. When using this tuning mode, the implementation **may** compromise quality to maximize the performance and minimize the latency of video encoding operations.
- VK_VIDEO_ENCODE_TUNING_MODE_LOSSLESS_KHR specifies that video encoding is tuned for lossless encoding. When using this tuning mode, video encode operations produce lossless output.

The VkVideoProfileListInfoKHR structure is defined as:

```
// Provided by VK_KHR_video_queue
typedef struct VkVideoProfileListInfoKHR {
    VkStructureType                 sType;
    const void*                     pNext;
    uint32_t                        profileCount;
    const VkVideoProfileInfoKHR*    pProfiles;
} VkVideoProfileListInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- profileCount is the number of elements in the pProfiles array.
- pProfiles is a pointer to an array of VkVideoProfileInfoKHR structures.

**NOTE** Video transcoding is an example of a use case that necessitates the specification of multiple profiles in various contexts.

When the application provides a video decode profile and one or more video encode profiles in the profile list, the implementation ensures that any capabilitities returned or resources created are suitable for the video transcoding use cases without the need for manual data transformations.

### Valid Usage

- VUID-VkVideoProfileListInfoKHR-pProfiles-06813
  pProfiles **must** not contain more than one element whose videoCodecOperation member specifies a decode operation

### Valid Usage (Implicit)

- VUID-VkVideoProfileListInfoKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_VIDEO_PROFILE_LIST_INFO_KHR
- VUID-VkVideoProfileListInfoKHR-pProfiles-parameter
  If profileCount is not 0, pProfiles **must** be a valid pointer to an array of profileCount valid VkVideoProfileInfoKHR structures

# 42.4. Video Capabilities

## 42.4.1. Video Coding Capabilities

To query video coding capabilities for a specific video profile, call:

```
// Provided by VK_KHR_video_queue
VkResult vkGetPhysicalDeviceVideoCapabilitiesKHR(
    VkPhysicalDevice                            physicalDevice,
    const VkVideoProfileInfoKHR*                pVideoProfile,
    VkVideoCapabilitiesKHR*                     pCapabilities);
```

- `physicalDevice` is the physical device from which to query the video decode or encode capabilities.
- `pVideoProfile` is a pointer to a VkVideoProfileInfoKHR structure.
- `pCapabilities` is a pointer to a VkVideoCapabilitiesKHR structure in which the capabilities are returned.

If the video profile described by `pVideoProfile` is supported by the implementation, then this command returns `VK_SUCCESS` and `pCapabilities` is filled with the capabilities supported with the specified video profile. Otherwise, one of the video-profile-specific error codes are returned.

### Valid Usage

- VUID-vkGetPhysicalDeviceVideoCapabilitiesKHR-pVideoProfile-07183
  If `pVideoProfile->videoCodecOperation` specifies a decode operation, then the `pNext` chain of `pCapabilities` **must** include a VkVideoDecodeCapabilitiesKHR structure
- VUID-vkGetPhysicalDeviceVideoCapabilitiesKHR-pVideoProfile-07184
  If `pVideoProfile->videoCodecOperation` is `VK_VIDEO_CODEC_OPERATION_DECODE_H264_BIT_KHR`, then the `pNext` chain of `pCapabilities` **must** include a VkVideoDecodeH264CapabilitiesKHR structure
- VUID-vkGetPhysicalDeviceVideoCapabilitiesKHR-pVideoProfile-07185
  If `pVideoProfile->videoCodecOperation` is `VK_VIDEO_CODEC_OPERATION_DECODE_H265_BIT_KHR`, then the `pNext` chain of `pCapabilities` **must** include a VkVideoDecodeH265CapabilitiesKHR structure
- VUID-vkGetPhysicalDeviceVideoCapabilitiesKHR-pVideoProfile-09257
  If `pVideoProfile->videoCodecOperation` is `VK_VIDEO_CODEC_OPERATION_DECODE_AV1_BIT_KHR`, then the `pNext` chain of `pCapabilities` **must** include a VkVideoDecodeAV1CapabilitiesKHR structure
- VUID-vkGetPhysicalDeviceVideoCapabilitiesKHR-pVideoProfile-07186
  If `pVideoProfile->videoCodecOperation` specifies an encode operation, then the `pNext` chain of `pCapabilities` **must** include a VkVideoEncodeCapabilitiesKHR structure
- VUID-vkGetPhysicalDeviceVideoCapabilitiesKHR-pVideoProfile-07187
  If `pVideoProfile->videoCodecOperation` is `VK_VIDEO_CODEC_OPERATION_ENCODE_H264_BIT_KHR`,

then the `pNext` chain of `pCapabilities` **must** include a VkVideoEncodeH264CapabilitiesKHR structure

- VUID-vkGetPhysicalDeviceVideoCapabilitiesKHR-pVideoProfile-07188
  If `pVideoProfile->videoCodecOperation` is `VK_VIDEO_CODEC_OPERATION_ENCODE_H265_BIT_KHR`, then the `pNext` chain of `pCapabilities` **must** include a VkVideoEncodeH265CapabilitiesKHR structure

## Valid Usage (Implicit)

- VUID-vkGetPhysicalDeviceVideoCapabilitiesKHR-physicalDevice-parameter
  `physicalDevice` **must** be a valid VkPhysicalDevice handle
- VUID-vkGetPhysicalDeviceVideoCapabilitiesKHR-pVideoProfile-parameter
  `pVideoProfile` **must** be a valid pointer to a valid VkVideoProfileInfoKHR structure
- VUID-vkGetPhysicalDeviceVideoCapabilitiesKHR-pCapabilities-parameter
  `pCapabilities` **must** be a valid pointer to a VkVideoCapabilitiesKHR structure

## Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_VIDEO_PROFILE_OPERATION_NOT_SUPPORTED_KHR`
- `VK_ERROR_VIDEO_PROFILE_FORMAT_NOT_SUPPORTED_KHR`
- `VK_ERROR_VIDEO_PICTURE_LAYOUT_NOT_SUPPORTED_KHR`
- `VK_ERROR_VIDEO_PROFILE_CODEC_NOT_SUPPORTED_KHR`

The `VkVideoCapabilitiesKHR` structure is defined as:

```
// Provided by VK_KHR_video_queue
typedef struct VkVideoCapabilitiesKHR {
    VkStructureType                 sType;
    void*                           pNext;
    VkVideoCapabilityFlagsKHR       flags;
    VkDeviceSize                    minBitstreamBufferOffsetAlignment;
    VkDeviceSize                    minBitstreamBufferSizeAlignment;
    VkExtent2D                      pictureAccessGranularity;
    VkExtent2D                      minCodedExtent;
    VkExtent2D                      maxCodedExtent;
    uint32_t                        maxDpbSlots;
    uint32_t                        maxActiveReferencePictures;
    VkExtensionProperties           stdHeaderVersion;
} VkVideoCapabilitiesKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is a bitmask of VkVideoCapabilityFlagBitsKHR specifying capability flags.
- `minBitstreamBufferOffsetAlignment` is the minimum alignment for bitstream buffer offsets.
- `minBitstreamBufferSizeAlignment` is the minimum alignment for bitstream buffer range sizes.
- `pictureAccessGranularity` is the granularity at which image access to video picture resources happen.
- `minCodedExtent` is the minimum width and height of the coded frames.
- `maxCodedExtent` is the maximum width and height of the coded frames.
- `maxDpbSlots` is the maximum number of DPB slots supported by a single video session.
- `maxActiveReferencePictures` is the maximum number of active reference pictures a single video coding operation **can** use.
- `stdHeaderVersion` is a VkExtensionProperties structure reporting the Video Std header name and version supported for the video profile.

**NOTE** It is common for video compression standards to allow using all reference pictures associated with active DPB slots as active reference pictures, hence for video decode profiles the values returned in `maxDpbSlots` and `maxActiveReferencePictures` are often equal. Similarly, in case of video decode profiles supporting field pictures the value of `maxActiveReferencePictures` often equals `maxDpbSlots` × 2.

### Valid Usage (Implicit)

- VUID-VkVideoCapabilitiesKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_VIDEO_CAPABILITIES_KHR`
- VUID-VkVideoCapabilitiesKHR-pNext-pNext
  Each `pNext` member of any structure (including this one) in the `pNext` chain **must** be either `NULL` or a pointer to a valid instance of VkVideoDecodeAV1CapabilitiesKHR,

VkVideoDecodeCapabilitiesKHR, VkVideoDecodeH264CapabilitiesKHR, VkVideoDecodeH265CapabilitiesKHR, VkVideoEncodeCapabilitiesKHR, VkVideoEncodeH264CapabilitiesKHR, or VkVideoEncodeH265CapabilitiesKHR

- VUID-VkVideoCapabilitiesKHR-sType-unique
  The sType value of each struct in the pNext chain **must** be unique

Bits which **can** be set in VkVideoCapabilitiesKHR::flags are:

```
// Provided by VK_KHR_video_queue
typedef enum VkVideoCapabilityFlagBitsKHR {
    VK_VIDEO_CAPABILITY_PROTECTED_CONTENT_BIT_KHR = 0x00000001,
    VK_VIDEO_CAPABILITY_SEPARATE_REFERENCE_IMAGES_BIT_KHR = 0x00000002,
} VkVideoCapabilityFlagBitsKHR;
```

- VK_VIDEO_CAPABILITY_PROTECTED_CONTENT_BIT_KHR indicates that video sessions support producing and consuming protected content.
- VK_VIDEO_CAPABILITY_SEPARATE_REFERENCE_IMAGES_BIT_KHR indicates that the video picture resources associated with the DPB slots of a video session **can** be backed by separate VkImage objects. If this capability flag is not present, then all DPB slots of a video session **must** be associated with video picture resources backed by the same VkImage object (e.g. using different layers of the same image).

```
// Provided by VK_KHR_video_queue
typedef VkFlags VkVideoCapabilityFlagsKHR;
```

VkVideoCapabilityFlagsKHR is a bitmask type for setting a mask of zero or more VkVideoCapabilityFlagBitsKHR.

### 42.4.2. Video Format Capabilities

To enumerate the supported output, input and DPB image formats and corresponding capabilities for a specific video profile, call:

```
// Provided by VK_KHR_video_queue
VkResult vkGetPhysicalDeviceVideoFormatPropertiesKHR(
    VkPhysicalDevice                            physicalDevice,
    const VkPhysicalDeviceVideoFormatInfoKHR*   pVideoFormatInfo,
    uint32_t*                                   pVideoFormatPropertyCount,
    VkVideoFormatPropertiesKHR*                 pVideoFormatProperties);
```

- physicalDevice is the physical device from which to query the video format properties.
- pVideoFormatInfo is a pointer to a VkPhysicalDeviceVideoFormatInfoKHR structure specifying the usage and video profiles for which supported image formats and capabilities are returned.
- pVideoFormatPropertyCount is a pointer to an integer related to the number of video format

properties available or queried, as described below.

- pVideoFormatProperties is a pointer to an array of VkVideoFormatPropertiesKHR structures in which supported image formats and capabilities are returned.

If pVideoFormatProperties is NULL, then the number of video format properties supported for the given physicalDevice is returned in pVideoFormatPropertyCount. Otherwise, pVideoFormatPropertyCount **must** point to a variable set by the application to the number of elements in the pVideoFormatProperties array, and on return the variable is overwritten with the number of values actually written to pVideoFormatProperties. If the value of pVideoFormatPropertyCount is less than the number of video format properties supported, at most pVideoFormatPropertyCount values will be written to pVideoFormatProperties, and VK_INCOMPLETE will be returned instead of VK_SUCCESS, to indicate that not all the available values were returned.

Video format properties are always queried with respect to a specific set of video profiles. These are specified by chaining the VkVideoProfileListInfoKHR structure to pVideoFormatInfo.

For most use cases, the images are used by a single video session and a single video profile is provided. For a use case such as video transcoding, where a decode session output image **can** be used as encode input in one or more encode sessions, multiple video profiles corresponding to the video sessions that will share the image **must** be provided.

If any of the video profiles specified via VkVideoProfileListInfoKHR::pProfiles are not supported, then this command returns one of the video-profile-specific error codes. Furthermore, if VkPhysicalDeviceVideoFormatInfoKHR::imageUsage includes any image usage flags not supported by the specified video profiles, then this command returns VK_ERROR_IMAGE_USAGE_NOT_SUPPORTED_KHR.

This command also returns VK_ERROR_IMAGE_USAGE_NOT_SUPPORTED_KHR if VkPhysicalDeviceVideoFormatInfoKHR::imageUsage does not include the appropriate flags as dictated by the decode capability flags returned in VkVideoDecodeCapabilitiesKHR::flags for any of the profiles specified in the VkVideoProfileListInfoKHR structure provided in the pNext chain of pVideoFormatInfo.

If the decode capability flags include VK_VIDEO_DECODE_CAPABILITY_DPB_AND_OUTPUT_COINCIDE_BIT_KHR but not VK_VIDEO_DECODE_CAPABILITY_DPB_AND_OUTPUT_DISTINCT_BIT_KHR, then in order to query video format properties for decode DPB and output usage, VkPhysicalDeviceVideoFormatInfoKHR::imageUsage **must** include both VK_IMAGE_USAGE_VIDEO_DECODE_DPB_BIT_KHR and VK_IMAGE_USAGE_VIDEO_DECODE_DST_BIT_KHR. Otherwise, the call will fail with VK_ERROR_IMAGE_USAGE_NOT_SUPPORTED_KHR.

If the decode capability flags include VK_VIDEO_DECODE_CAPABILITY_DPB_AND_OUTPUT_DISTINCT_BIT_KHR but not VK_VIDEO_DECODE_CAPABILITY_DPB_AND_OUTPUT_COINCIDE_BIT_KHR, then in order to query video format properties for decode DPB usage, VkPhysicalDeviceVideoFormatInfoKHR::imageUsage **must** include VK_IMAGE_USAGE_VIDEO_DECODE_DPB_BIT_KHR, but not VK_IMAGE_USAGE_VIDEO_DECODE_DST_BIT_KHR. Otherwise, the call will fail with VK_ERROR_IMAGE_USAGE_NOT_SUPPORTED_KHR. Similarly, to query video format properties for decode output usage, VkPhysicalDeviceVideoFormatInfoKHR::imageUsage **must** include VK_IMAGE_USAGE_VIDEO_DECODE_DST_BIT_KHR, but not VK_IMAGE_USAGE_VIDEO_DECODE_DPB_BIT_KHR. Otherwise, the call will fail with VK_ERROR_IMAGE_USAGE_NOT_SUPPORTED_KHR.

The imageUsage member of the VkPhysicalDeviceVideoFormatInfoKHR structure specifies the expected video usage flags that the returned video formats **must** support. Correspondingly, the imageUsageFlags member of each VkVideoFormatPropertiesKHR structure returned will contain at least the same set of image usage flags.

If the implementation supports using video input, output, or DPB images of a particular format in operations other than video decode/encode then the imageUsageFlags member of the corresponding VkVideoFormatPropertiesKHR structure returned will include additional image usage flags indicating that.

| | |
|---|---|
| **NOTE** | For most use cases, only decode or encode related usage flags are going to be specified. For a use case such as transcode, if the image were to be shared between decode and encode session(s), then both decode and encode related usage flags **can** be set. |

Multiple VkVideoFormatPropertiesKHR entries **may** be returned with the same format member with different componentMapping, imageType, or imageTiling values, as described later.

In addition, a different set of VkVideoFormatPropertiesKHR entries **may** be returned depending on the imageUsage member of the VkPhysicalDeviceVideoFormatInfoKHR structure, even for the same set of video profiles, for example, based on whether encode input, encode DPB, decode output, and/or decode DPB usage is requested.

The application **can** select the parameters returned in the VkVideoFormatPropertiesKHR entries and use compatible parameters when creating the input, output, and DPB images. The implementation will report all image creation and usage flags that are valid for images used with the requested video profiles but applications **should** create images only with those that are necessary for the particular use case.

Before creating an image, the application **can** obtain the complete set of supported image format features by calling vkGetPhysicalDeviceImageFormatProperties2 using parameters derived from the members of one of the reported VkVideoFormatPropertiesKHR entries and adding the same VkVideoProfileListInfoKHR structure to the pNext chain of VkPhysicalDeviceImageFormatInfo2.

The following applies to all VkVideoFormatPropertiesKHR entries returned by vkGetPhysicalDeviceVideoFormatPropertiesKHR:

- vkGetPhysicalDeviceFormatProperties2 **must** succeed when called with VkVideoFormatPropertiesKHR::format
- If VkVideoFormatPropertiesKHR::imageTiling is VK_IMAGE_TILING_OPTIMAL, then the optimalTilingFeatures returned by vkGetPhysicalDeviceFormatProperties2 **must** include all format features required by the image usage flags reported in VkVideoFormatPropertiesKHR::imageUsageFlags for the format, as indicated in the Format Feature Dependent Usage Flags section.
- If VkVideoFormatPropertiesKHR::imageTiling is VK_IMAGE_TILING_LINEAR, then the linearTilingFeatures returned by vkGetPhysicalDeviceFormatProperties2 **must** include all format features required by the image usage flags reported in VkVideoFormatPropertiesKHR::imageUsageFlags for the format, as indicated in the Format Feature Dependent Usage Flags

section.

- vkGetPhysicalDeviceImageFormatProperties2 **must** succeed when called with a VkPhysicalDeviceImageFormatInfo2 structure containing the following information:
  - The `pNext` chain including the same VkVideoProfileListInfoKHR structure used to call `vkGetPhysicalDeviceVideoFormatPropertiesKHR`.
  - `format` set to the value of `VkVideoFormatPropertiesKHR::format`.
  - `type` set to the value of `VkVideoFormatPropertiesKHR::imageType`.
  - `tiling` set to the value of `VkVideoFormatPropertiesKHR::imageTiling`.
  - `usage` set to the value of `VkVideoFormatPropertiesKHR::imageUsageFlags`.
  - `flags` set to the value of `VkVideoFormatPropertiesKHR::imageCreateFlags`.

The `componentMapping` member of `VkVideoFormatPropertiesKHR` defines the ordering of the $Y'C_BC_R$ color channels from the perspective of the video codec operations specified in VkVideoProfileListInfoKHR. For example, if the implementation produces video decode output with the format `VK_FORMAT_G8_B8R8_2PLANE_420_UNORM` where the blue and red chrominance channels are swapped then the `componentMapping` member of the corresponding `VkVideoFormatPropertiesKHR` structure will have the following member values:

```
components.r = VK_COMPONENT_SWIZZLE_B;        // Cb component
components.g = VK_COMPONENT_SWIZZLE_IDENTITY; // Y component
components.b = VK_COMPONENT_SWIZZLE_R;        // Cr component
components.a = VK_COMPONENT_SWIZZLE_IDENTITY; // unused, defaults to 1.0
```

## Valid Usage

- VUID-vkGetPhysicalDeviceVideoFormatPropertiesKHR-pNext-06812
  The `pNext` chain of `pVideoFormatInfo` **must** include a VkVideoProfileListInfoKHR structure with `profileCount` greater than `0`

## Valid Usage (Implicit)

- VUID-vkGetPhysicalDeviceVideoFormatPropertiesKHR-physicalDevice-parameter
  `physicalDevice` **must** be a valid VkPhysicalDevice handle
- VUID-vkGetPhysicalDeviceVideoFormatPropertiesKHR-pVideoFormatInfo-parameter
  `pVideoFormatInfo` **must** be a valid pointer to a valid VkPhysicalDeviceVideoFormatInfoKHR structure
- VUID-vkGetPhysicalDeviceVideoFormatPropertiesKHR-pVideoFormatPropertyCount-parameter
  `pVideoFormatPropertyCount` **must** be a valid pointer to a `uint32_t` value
- VUID-vkGetPhysicalDeviceVideoFormatPropertiesKHR-pVideoFormatProperties-parameter
  If the value referenced by `pVideoFormatPropertyCount` is not `0`, and `pVideoFormatProperties`

is not NULL, pVideoFormatProperties **must** be a valid pointer to an array of pVideoFormatPropertyCount VkVideoFormatPropertiesKHR structures

## Return Codes

**Success**

- VK_SUCCESS
- VK_INCOMPLETE

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY
- VK_ERROR_IMAGE_USAGE_NOT_SUPPORTED_KHR
- VK_ERROR_VIDEO_PROFILE_OPERATION_NOT_SUPPORTED_KHR
- VK_ERROR_VIDEO_PROFILE_FORMAT_NOT_SUPPORTED_KHR
- VK_ERROR_VIDEO_PICTURE_LAYOUT_NOT_SUPPORTED_KHR
- VK_ERROR_VIDEO_PROFILE_CODEC_NOT_SUPPORTED_KHR

The VkPhysicalDeviceVideoFormatInfoKHR structure is defined as:

```
// Provided by VK_KHR_video_queue
typedef struct VkPhysicalDeviceVideoFormatInfoKHR {
    VkStructureType      sType;
    const void*          pNext;
    VkImageUsageFlags    imageUsage;
} VkPhysicalDeviceVideoFormatInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- imageUsage is a bitmask of VkImageUsageFlagBits specifying the intended usage of the video images.

## Valid Usage (Implicit)

- VUID-VkPhysicalDeviceVideoFormatInfoKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_VIDEO_FORMAT_INFO_KHR
- VUID-VkPhysicalDeviceVideoFormatInfoKHR-pNext-pNext
  pNext **must** be NULL or a pointer to a valid instance of VkVideoProfileListInfoKHR
- VUID-VkPhysicalDeviceVideoFormatInfoKHR-sType-unique
  The sType value of each struct in the pNext chain **must** be unique
- VUID-VkPhysicalDeviceVideoFormatInfoKHR-imageUsage-parameter

imageUsage **must** be a valid combination of VkImageUsageFlagBits values

- VUID-VkPhysicalDeviceVideoFormatInfoKHR-imageUsage-requiredbitmask
  imageUsage **must** not be 0

The VkVideoFormatPropertiesKHR structure is defined as:

```
// Provided by VK_KHR_video_queue
typedef struct VkVideoFormatPropertiesKHR {
    VkStructureType       sType;
    void*                 pNext;
    VkFormat              format;
    VkComponentMapping    componentMapping;
    VkImageCreateFlags    imageCreateFlags;
    VkImageType           imageType;
    VkImageTiling         imageTiling;
    VkImageUsageFlags     imageUsageFlags;
} VkVideoFormatPropertiesKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- format is a VkFormat that specifies the format that **can** be used with the specified video profiles and image usages.
- componentMapping defines the color channel order used for the format. format along with componentMapping describe how the color channels are ordered when producing video decoder output or are expected to be ordered in video encoder input, when applicable. If the format reported does not require component swizzling then all members of componentMapping will be set to VK_COMPONENT_SWIZZLE_IDENTITY.
- imageCreateFlags is a bitmask of VkImageCreateFlagBits specifying the supported image creation flags for the format.
- imageType is a VkImageType that specifies the image type the format **can** be used with.
- imageTiling is a VkImageTiling that specifies the image tiling the format **can** be used with.
- imageUsageFlags is a bitmask of VkImageUsageFlagBits specifying the supported image usage flags for the format.

### Valid Usage (Implicit)

- VUID-VkVideoFormatPropertiesKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_VIDEO_FORMAT_PROPERTIES_KHR
- VUID-VkVideoFormatPropertiesKHR-pNext-pNext
  pNext **must** be NULL

# 42.5. Video Sessions

Video sessions are objects that represent and maintain the state needed to perform video decode or encode operations using a specific video profile.

In case of video encode profiles this includes the current rate control configuration and the currently set video encode quality level.

Video sessions are represented by VkVideoSessionKHR handles:

```
// Provided by VK_KHR_video_queue
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkVideoSessionKHR)
```

## 42.5.1. Creating a Video Session

To create a video session object, call:

```
// Provided by VK_KHR_video_queue
VkResult vkCreateVideoSessionKHR(
    VkDevice                                    device,
    const VkVideoSessionCreateInfoKHR*          pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkVideoSessionKHR*                          pVideoSession);
```

- device is the logical device that creates the video session.
- pCreateInfo is a pointer to a VkVideoSessionCreateInfoKHR structure containing parameters to be used to create the video session.
- pAllocator controls host memory allocation as described in the Memory Allocation chapter.
- pVideoSession is a pointer to a VkVideoSessionKHR handle in which the resulting video session object is returned.

The resulting video session object is said to be created with the video codec operation specified in pCreateInfo->pVideoProfile->videoCodecOperation.

The name and version of the codec-specific Video Std header to be used with the video session is specified by the VkExtensionProperties structure pointed to by pCreateInfo->pStdHeaderVersion. If a non-existent or unsupported Video Std header version is specified in pCreateInfo->pStdHeaderVersion->specVersion, then this command returns VK_ERROR_VIDEO_STD_VERSION_NOT_SUPPORTED_KHR.

Video session objects are created in *uninitialized* state. In order to transition the video session into *initial* state, the application **must** issue a vkCmdControlVideoCodingKHR command with VkVideoCodingControlInfoKHR::flags including VK_VIDEO_CODING_CONTROL_RESET_BIT_KHR.

Video session objects also maintain the state of the DPB. The number of DPB slots usable with the created video session is specified in pCreateInfo->maxDpbSlots, and each slot is initially in the

inactive state.

Each DPB slot maintained by the created video session **can** refer to a reference picture representing a video frame.

In addition, if the `videoCodecOperation` member of the VkVideoProfileInfoKHR structure pointed to by `pCreateInfo->pVideoProfile` is `VK_VIDEO_CODEC_OPERATION_DECODE_H264_BIT_KHR` and the `pictureLayout` member of the VkVideoDecodeH264ProfileInfoKHR structure provided in the VkVideoProfileInfoKHR::`pNext` chain is not `VK_VIDEO_DECODE_H264_PICTURE_LAYOUT_PROGRESSIVE_KHR`, then the created video session supports *interlaced* frames and each DPB slot maintained by the created video session **can** instead refer to separate top field and bottom field reference pictures that together **can** represent a full video frame. In this case, it is up to the application, driven by the video content, whether it associates any individual DPB slot with separate top and/or bottom field pictures or a single picture representing a full frame.

The created video session **can** be used to perform video coding operations using video frames up to the maximum size specified in `pCreateInfo->maxCodedExtent`. The minimum frame size allowed is implicitly derived from VkVideoCapabilitiesKHR::`minCodedExtent`, as returned by vkGetPhysicalDeviceVideoCapabilitiesKHR for the video profile specified by `pCreateInfo->pVideoProfile`. Accordingly, the created video session is said to be created with a `minCodedExtent` equal to that.

In case of video session objects created with a video encode operation, implementations **may** return the `VK_ERROR_INVALID_VIDEO_STD_PARAMETERS_KHR` error if any of the specified Video Std parameters do not adhere to the syntactic or semantic requirements of the used video compression standard, or if values derived from parameters according to the rules defined by the used video compression standard do not adhere to the capabilities of the video compression standard or the implementation.

| | |
|---|---|
| **NOTE** | Applications **should** not rely on the `VK_ERROR_INVALID_VIDEO_STD_PARAMETERS_KHR` error being returned by any command as a means to verify Video Std parameters, as implementations are not required to report the error in any specific set of cases. |

### Valid Usage (Implicit)

- VUID-vkCreateVideoSessionKHR-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkCreateVideoSessionKHR-pCreateInfo-parameter
  `pCreateInfo` **must** be a valid pointer to a valid VkVideoSessionCreateInfoKHR structure
- VUID-vkCreateVideoSessionKHR-pAllocator-parameter
  If `pAllocator` is not `NULL`, `pAllocator` **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkCreateVideoSessionKHR-pVideoSession-parameter
  `pVideoSession` **must** be a valid pointer to a VkVideoSessionKHR handle

### Return Codes

**Success**

- VK_SUCCESS

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY
- VK_ERROR_INITIALIZATION_FAILED
- VK_ERROR_VIDEO_STD_VERSION_NOT_SUPPORTED_KHR
- VK_ERROR_INVALID_VIDEO_STD_PARAMETERS_KHR

The VkVideoSessionCreateInfoKHR structure is defined as:

```
// Provided by VK_KHR_video_queue
typedef struct VkVideoSessionCreateInfoKHR {
    VkStructureType                 sType;
    const void*                     pNext;
    uint32_t                        queueFamilyIndex;
    VkVideoSessionCreateFlagsKHR    flags;
    const VkVideoProfileInfoKHR*    pVideoProfile;
    VkFormat                        pictureFormat;
    VkExtent2D                      maxCodedExtent;
    VkFormat                        referencePictureFormat;
    uint32_t                        maxDpbSlots;
    uint32_t                        maxActiveReferencePictures;
    const VkExtensionProperties*    pStdHeaderVersion;
} VkVideoSessionCreateInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- queueFamilyIndex is the index of the queue family the created video session will be used with.
- flags is a bitmask of VkVideoSessionCreateFlagBitsKHR specifying creation flags.
- pVideoProfile is a pointer to a VkVideoProfileInfoKHR structure specifying the video profile the created video session will be used with.
- pictureFormat is the image format the created video session will be used with. If pVideoProfile->videoCodecOperation specifies a decode operation, then pictureFormat is the image format of decode output pictures usable with the created video session. If pVideoProfile->videoCodecOperation specifies an encode operation, then pictureFormat is the image format of encode input pictures usable with the created video session.
- maxCodedExtent is the maximum width and height of the coded frames the created video session will be used with.
- referencePictureFormat is the image format of reference pictures stored in the DPB the created

video session will be used with.

- maxDpbSlots is the maximum number of DPB Slots that **can** be used with the created video session.
- maxActiveReferencePictures is the maximum number of active reference pictures that **can** be used in a single video coding operation using the created video session.
- pStdHeaderVersion is a pointer to a VkExtensionProperties structure requesting the Video Std header version to use for the videoCodecOperation specified in pVideoProfile.

## Valid Usage

- VUID-VkVideoSessionCreateInfoKHR-protectedMemory-07189
  If the protectedMemory feature is not enabled or if VkVideoCapabilitiesKHR::flags does not include VK_VIDEO_CAPABILITY_PROTECTED_CONTENT_BIT_KHR, as returned by vkGetPhysicalDeviceVideoCapabilitiesKHR for the video profile specified by pVideoProfile, then flags **must** not include VK_VIDEO_SESSION_CREATE_PROTECTED_CONTENT_BIT_KHR
- VUID-VkVideoSessionCreateInfoKHR-flags-08371
  If flags includes VK_VIDEO_SESSION_CREATE_INLINE_QUERIES_BIT_KHR, then videoMaintenance1 **must** be enabled
- VUID-VkVideoSessionCreateInfoKHR-pVideoProfile-04845
  pVideoProfile **must** be a supported video profile
- VUID-VkVideoSessionCreateInfoKHR-maxDpbSlots-04847
  maxDpbSlots **must** be less than or equal to VkVideoCapabilitiesKHR::maxDpbSlots, as returned by vkGetPhysicalDeviceVideoCapabilitiesKHR for the video profile specified by pVideoProfile
- VUID-VkVideoSessionCreateInfoKHR-maxActiveReferencePictures-04849
  maxActiveReferencePictures **must** be less than or equal to VkVideoCapabilitiesKHR::maxActiveReferencePictures, as returned by vkGetPhysicalDeviceVideoCapabilitiesKHR for the video profile specified by pVideoProfile
- VUID-VkVideoSessionCreateInfoKHR-maxDpbSlots-04850
  If either maxDpbSlots or maxActiveReferencePictures is 0, then both **must** be 0
- VUID-VkVideoSessionCreateInfoKHR-maxCodedExtent-04851
  maxCodedExtent **must** be between VkVideoCapabilitiesKHR::minCodedExtent and VkVideoCapabilitiesKHR::maxCodedExtent, inclusive, as returned by vkGetPhysicalDeviceVideoCapabilitiesKHR for the video profile specified by pVideoProfile
- VUID-VkVideoSessionCreateInfoKHR-referencePictureFormat-04852
  If pVideoProfile->videoCodecOperation specifies a decode operation and maxActiveReferencePictures is greater than 0, then referencePictureFormat **must** be one of the supported decode DPB formats, as returned by vkGetPhysicalDeviceVideoFormatPropertiesKHR in VkVideoFormatPropertiesKHR::format when called with the imageUsage member of its pVideoFormatInfo parameter containing VK_IMAGE_USAGE_VIDEO_DECODE_DPB_BIT_KHR, and with a VkVideoProfileListInfoKHR structure specified in the pNext chain of its pVideoFormatInfo parameter whose pProfiles

member contains an element matching pVideoProfile

- VUID-VkVideoSessionCreateInfoKHR-referencePictureFormat-06814
  If pVideoProfile->videoCodecOperation specifies an encode operation and maxActiveReferencePictures is greater than 0, then referencePictureFormat **must** be one of the supported decode DPB formats, as returned by then referencePictureFormat **must** be one of the supported encode DPB formats, as returned by vkGetPhysicalDeviceVideoFormatPropertiesKHR in VkVideoFormatPropertiesKHR::format when called with the imageUsage member of its pVideoFormatInfo parameter containing VK_IMAGE_USAGE_VIDEO_ENCODE_DPB_BIT_KHR, and with a VkVideoProfileListInfoKHR structure specified in the pNext chain of its pVideoFormatInfo parameter whose pProfiles member contains an element matching pVideoProfile
- VUID-VkVideoSessionCreateInfoKHR-pictureFormat-04853
  If pVideoProfile->videoCodecOperation specifies a decode operation, then pictureFormat **must** be one of the supported decode output formats, as returned by vkGetPhysicalDeviceVideoFormatPropertiesKHR in VkVideoFormatPropertiesKHR::format when called with the imageUsage member of its pVideoFormatInfo parameter containing VK_IMAGE_USAGE_VIDEO_DECODE_DST_BIT_KHR, and with a VkVideoProfileListInfoKHR structure specified in the pNext chain of its pVideoFormatInfo parameter whose pProfiles member contains an element matching pVideoProfile
- VUID-VkVideoSessionCreateInfoKHR-pictureFormat-04854
  If pVideoProfile->videoCodecOperation specifies an encode operation, then pictureFormat **must** be one of the supported encode input formats, as returned by vkGetPhysicalDeviceVideoFormatPropertiesKHR in VkVideoFormatPropertiesKHR::format when called with the imageUsage member of its pVideoFormatInfo parameter containing VK_IMAGE_USAGE_VIDEO_ENCODE_SRC_BIT_KHR, and with a VkVideoProfileListInfoKHR structure specified in the pNext chain of its pVideoFormatInfo parameter whose pProfiles member contains an element matching pVideoProfile
- VUID-VkVideoSessionCreateInfoKHR-pStdHeaderVersion-07190
  pStdHeaderVersion->extensionName **must** match VkVideoCapabilitiesKHR::stdHeaderVersion.extensionName, as returned by vkGetPhysicalDeviceVideoCapabilitiesKHR for the video profile specified by pVideoProfile
- VUID-VkVideoSessionCreateInfoKHR-pStdHeaderVersion-07191
  pStdHeaderVersion->specVersion **must** be less than or equal to VkVideoCapabilitiesKHR::stdHeaderVersion.specVersion, as returned by vkGetPhysicalDeviceVideoCapabilitiesKHR for the video profile specified by pVideoProfile
- VUID-VkVideoSessionCreateInfoKHR-pVideoProfile-08251
  If pVideoProfile->videoCodecOperation is VK_VIDEO_CODEC_OPERATION_ENCODE_H264_BIT_KHR and the pNext chain of this structure includes a VkVideoEncodeH264SessionCreateInfoKHR structure, then its maxLevelIdc member **must** be less than or equal to VkVideoEncodeH264CapabilitiesKHR::maxLevelIdc, as returned by vkGetPhysicalDeviceVideoCapabilitiesKHR for the video profile specified in pVideoProfile
- VUID-VkVideoSessionCreateInfoKHR-pVideoProfile-08252
  If pVideoProfile->videoCodecOperation is VK_VIDEO_CODEC_OPERATION_ENCODE_H265_BIT_KHR and the pNext chain of this structure includes a VkVideoEncodeH265SessionCreateInfoKHR structure, then its maxLevelIdc member **must**

be less than or equal to VkVideoEncodeH265CapabilitiesKHR::maxLevelIdc, as returned by vkGetPhysicalDeviceVideoCapabilitiesKHR for the video profile specified in pVideoProfile

### Valid Usage (Implicit)

- VUID-VkVideoSessionCreateInfoKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_VIDEO_SESSION_CREATE_INFO_KHR
- VUID-VkVideoSessionCreateInfoKHR-pNext-pNext
  Each pNext member of any structure (including this one) in the pNext chain **must** be either NULL or a pointer to a valid instance of VkVideoEncodeH264SessionCreateInfoKHR or VkVideoEncodeH265SessionCreateInfoKHR
- VUID-VkVideoSessionCreateInfoKHR-sType-unique
  The sType value of each struct in the pNext chain **must** be unique
- VUID-VkVideoSessionCreateInfoKHR-flags-parameter
  flags **must** be a valid combination of VkVideoSessionCreateFlagBitsKHR values
- VUID-VkVideoSessionCreateInfoKHR-pVideoProfile-parameter
  pVideoProfile **must** be a valid pointer to a valid VkVideoProfileInfoKHR structure
- VUID-VkVideoSessionCreateInfoKHR-pictureFormat-parameter
  pictureFormat **must** be a valid VkFormat value
- VUID-VkVideoSessionCreateInfoKHR-referencePictureFormat-parameter
  referencePictureFormat **must** be a valid VkFormat value
- VUID-VkVideoSessionCreateInfoKHR-pStdHeaderVersion-parameter
  pStdHeaderVersion **must** be a valid pointer to a valid VkExtensionProperties structure

Bits which **can** be set in VkVideoSessionCreateInfoKHR::flags are:

```
// Provided by VK_KHR_video_queue
typedef enum VkVideoSessionCreateFlagBitsKHR {
    VK_VIDEO_SESSION_CREATE_PROTECTED_CONTENT_BIT_KHR = 0x00000001,
  // Provided by VK_KHR_video_encode_queue
    VK_VIDEO_SESSION_CREATE_ALLOW_ENCODE_PARAMETER_OPTIMIZATIONS_BIT_KHR = 0x00000002,
  // Provided by VK_KHR_video_maintenance1
    VK_VIDEO_SESSION_CREATE_INLINE_QUERIES_BIT_KHR = 0x00000004,
} VkVideoSessionCreateFlagBitsKHR;
```

- VK_VIDEO_SESSION_CREATE_PROTECTED_CONTENT_BIT_KHR specifies that the video session uses protected video content.
- VK_VIDEO_SESSION_CREATE_ALLOW_ENCODE_PARAMETER_OPTIMIZATIONS_BIT_KHR specifies that the implementation is allowed to override video session parameters and other codec-specific encoding parameters to optimize video encode operations based on the use case information specified in the video profile and the used video encode quality level.

**NOTE** Not specifying

> VK_VIDEO_SESSION_CREATE_ALLOW_ENCODE_PARAMETER_OPTIMIZATIONS_BIT_KHR does not guarantee that the implementation will not do any codec-specific parameter overrides, as certain overrides are necessary for the correct operation of the video encoder implementation due to limitations to the available encoding tools on that implementation. This flag, however, enables the implementation to apply further optimizing overrides.

- VK_VIDEO_SESSION_CREATE_INLINE_QUERIES_BIT_KHR specifies that queries within video coding scopes using the created video session are executed inline with video coding operations.

```
// Provided by VK_KHR_video_queue
typedef VkFlags VkVideoSessionCreateFlagsKHR;
```

VkVideoSessionCreateFlagsKHR is a bitmask type for setting a mask of zero or more VkVideoSessionCreateFlagBitsKHR.

### 42.5.2. Destroying a Video Session

To destroy a video session, call:

```
// Provided by VK_KHR_video_queue
void vkDestroyVideoSessionKHR(
    VkDevice                                    device,
    VkVideoSessionKHR                           videoSession,
    const VkAllocationCallbacks*                pAllocator);
```

- device is the logical device that destroys the video session.
- videoSession is the video session to destroy.
- pAllocator controls host memory allocation as described in the Memory Allocation chapter.

#### Valid Usage

- VUID-vkDestroyVideoSessionKHR-videoSession-07192
  All submitted commands that refer to videoSession **must** have completed execution
- VUID-vkDestroyVideoSessionKHR-videoSession-07193
  If VkAllocationCallbacks were provided when videoSession was created, a compatible set of callbacks **must** be provided here
- VUID-vkDestroyVideoSessionKHR-videoSession-07194
  If no VkAllocationCallbacks were provided when videoSession was created, pAllocator **must** be NULL

### Valid Usage (Implicit)

- VUID-vkDestroyVideoSessionKHR-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkDestroyVideoSessionKHR-videoSession-parameter
  If `videoSession` is not VK_NULL_HANDLE, `videoSession` **must** be a valid VkVideoSessionKHR handle
- VUID-vkDestroyVideoSessionKHR-pAllocator-parameter
  If `pAllocator` is not `NULL`, `pAllocator` **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkDestroyVideoSessionKHR-videoSession-parent
  If `videoSession` is a valid handle, it **must** have been created, allocated, or retrieved from `device`

### Host Synchronization

- Host access to `videoSession` **must** be externally synchronized

## 42.5.3. Video Session Memory Association

After creating a video session object, and before the object **can** be used to record video coding operations into command buffers using it, the application **must** allocate and bind device memory to the video session. Device memory is allocated separately (see Device Memory) and then associated with the video session.

Video sessions **may** have multiple memory bindings identified by unique unsigned integer values. Appropriate device memory **must** be bound to each such memory binding before using the video session to record command buffer commands with it.

To determine the memory requirements for a video session object, call:

```
// Provided by VK_KHR_video_queue
VkResult vkGetVideoSessionMemoryRequirementsKHR(
    VkDevice                                    device,
    VkVideoSessionKHR                           videoSession,
    uint32_t*                                   pMemoryRequirementsCount,
    VkVideoSessionMemoryRequirementsKHR*        pMemoryRequirements);
```

- `device` is the logical device that owns the video session.
- `videoSession` is the video session to query.
- `pMemoryRequirementsCount` is a pointer to an integer related to the number of memory binding requirements available or queried, as described below.
- `pMemoryRequirements` is `NULL` or a pointer to an array of

VkVideoSessionMemoryRequirementsKHR structures in which the memory binding requirements of the video session are returned.

If pMemoryRequirements is NULL, then the number of memory bindings required for the video session is returned in pMemoryRequirementsCount. Otherwise, pMemoryRequirementsCount **must** point to a variable set by the application to the number of elements in the pMemoryRequirements array, and on return the variable is overwritten with the number of memory binding requirements actually written to pMemoryRequirements. If pMemoryRequirementsCount is less than the number of memory bindings required for the video session, then at most pMemoryRequirementsCount elements will be written to pMemoryRequirements, and VK_INCOMPLETE will be returned, instead of VK_SUCCESS, to indicate that not all required memory binding requirements were returned.

## Valid Usage (Implicit)

- VUID-vkGetVideoSessionMemoryRequirementsKHR-device-parameter
  device **must** be a valid VkDevice handle
- VUID-vkGetVideoSessionMemoryRequirementsKHR-videoSession-parameter
  videoSession **must** be a valid VkVideoSessionKHR handle
- VUID-vkGetVideoSessionMemoryRequirementsKHR-pMemoryRequirementsCount-parameter
  pMemoryRequirementsCount **must** be a valid pointer to a uint32_t value
- VUID-vkGetVideoSessionMemoryRequirementsKHR-pMemoryRequirements-parameter
  If the value referenced by pMemoryRequirementsCount is not 0, and pMemoryRequirements is not NULL, pMemoryRequirements **must** be a valid pointer to an array of pMemoryRequirementsCount VkVideoSessionMemoryRequirementsKHR structures
- VUID-vkGetVideoSessionMemoryRequirementsKHR-videoSession-parent
  videoSession **must** have been created, allocated, or retrieved from device

## Return Codes

**Success**

- VK_SUCCESS
- VK_INCOMPLETE

**Failure**

None

The VkVideoSessionMemoryRequirementsKHR structure is defined as:

```
// Provided by VK_KHR_video_queue
typedef struct VkVideoSessionMemoryRequirementsKHR {
    VkStructureType         sType;
    void*                   pNext;
    uint32_t                memoryBindIndex;
    VkMemoryRequirements    memoryRequirements;
} VkVideoSessionMemoryRequirementsKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `memoryBindIndex` is the index of the memory binding.
- `memoryRequirements` is a VkMemoryRequirements structure in which the requested memory binding requirements for the binding index specified by `memoryBindIndex` are returned.

### Valid Usage (Implicit)

- VUID-VkVideoSessionMemoryRequirementsKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_VIDEO_SESSION_MEMORY_REQUIREMENTS_KHR`
- VUID-VkVideoSessionMemoryRequirementsKHR-pNext-pNext
  `pNext` **must** be `NULL`

To attach memory to a video session object, call:

```
// Provided by VK_KHR_video_queue
VkResult vkBindVideoSessionMemoryKHR(
    VkDevice                                    device,
    VkVideoSessionKHR                           videoSession,
    uint32_t                                    bindSessionMemoryInfoCount,
    const VkBindVideoSessionMemoryInfoKHR*      pBindSessionMemoryInfos);
```

- `device` is the logical device that owns the video session.
- `videoSession` is the video session to be bound with device memory.
- `bindSessionMemoryInfoCount` is the number of elements in `pBindSessionMemoryInfos`.
- `pBindSessionMemoryInfos` is a pointer to an array of `bindSessionMemoryInfoCount` VkBindVideoSessionMemoryInfoKHR structures specifying memory regions to be bound to specific memory bindings of the video session.

The valid usage statements below refer to the VkMemoryRequirements structure corresponding to a specific element of `pBindSessionMemoryInfos`, which is defined as follows:

- If the `memoryBindIndex` member of the element of `pBindSessionMemoryInfos` in question matches the `memoryBindIndex` member of one of the elements returned in `pMemoryRequirements` when vkGetVideoSessionMemoryRequirementsKHR is called with the same `videoSession` and with

pMemoryRequirementsCount equal to bindSessionMemoryInfoCount, then the memoryRequirements member of that element of pMemoryRequirements is the VkMemoryRequirements structure corresponding to the element of pBindSessionMemoryInfos in question.

- Otherwise the element of pBindSessionMemoryInfos in question is said to not have a corresponding VkMemoryRequirements structure.

## Valid Usage

- VUID-vkBindVideoSessionMemoryKHR-videoSession-07195
  The memory binding of videoSession identified by the memoryBindIndex member of any element of pBindSessionMemoryInfos **must** not already be backed by a memory object
- VUID-vkBindVideoSessionMemoryKHR-memoryBindIndex-07196
  The memoryBindIndex member of each element of pBindSessionMemoryInfos **must** be unique within pBindSessionMemoryInfos
- VUID-vkBindVideoSessionMemoryKHR-pBindSessionMemoryInfos-07197
  Each element of pBindSessionMemoryInfos **must** have a corresponding VkMemoryRequirements structure
- VUID-vkBindVideoSessionMemoryKHR-pBindSessionMemoryInfos-07198
  If an element of pBindSessionMemoryInfos has a corresponding VkMemoryRequirements structure, then the memory member of that element of pBindSessionMemoryInfos **must** have been allocated using one of the memory types allowed in the memoryTypeBits member of the corresponding VkMemoryRequirements structure
- VUID-vkBindVideoSessionMemoryKHR-pBindSessionMemoryInfos-07199
  If an element of pBindSessionMemoryInfos has a corresponding VkMemoryRequirements structure, then the memoryOffset member of that element of pBindSessionMemoryInfos **must** be an integer multiple of the alignment member of the corresponding VkMemoryRequirements structure
- VUID-vkBindVideoSessionMemoryKHR-pBindSessionMemoryInfos-07200
  If an element of pBindSessionMemoryInfos has a corresponding VkMemoryRequirements structure, then the memorySize member of that element of pBindSessionMemoryInfos **must** equal the size member of the corresponding VkMemoryRequirements structure

## Valid Usage (Implicit)

- VUID-vkBindVideoSessionMemoryKHR-device-parameter
  device **must** be a valid VkDevice handle
- VUID-vkBindVideoSessionMemoryKHR-videoSession-parameter
  videoSession **must** be a valid VkVideoSessionKHR handle
- VUID-vkBindVideoSessionMemoryKHR-pBindSessionMemoryInfos-parameter
  pBindSessionMemoryInfos **must** be a valid pointer to an array of bindSessionMemoryInfoCount valid VkBindVideoSessionMemoryInfoKHR structures
- VUID-vkBindVideoSessionMemoryKHR-bindSessionMemoryInfoCount-arraylength
  bindSessionMemoryInfoCount **must** be greater than 0

- VUID-vkBindVideoSessionMemoryKHR-videoSession-parent
  `videoSession` **must** have been created, allocated, or retrieved from `device`

## Host Synchronization

- Host access to `videoSession` **must** be externally synchronized

## Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`

The `VkBindVideoSessionMemoryInfoKHR` structure is defined as:

```
// Provided by VK_KHR_video_queue
typedef struct VkBindVideoSessionMemoryInfoKHR {
    VkStructureType    sType;
    const void*        pNext;
    uint32_t           memoryBindIndex;
    VkDeviceMemory     memory;
    VkDeviceSize       memoryOffset;
    VkDeviceSize       memorySize;
} VkBindVideoSessionMemoryInfoKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `memoryBindIndex` is the memory binding index to bind memory to.
- `memory` is the allocated device memory to be bound to the video session's memory binding with index `memoryBindIndex`.
- `memoryOffset` is the start offset of the region of `memory` which is to be bound.
- `memorySize` is the size in bytes of the region of `memory`, starting from `memoryOffset` bytes, to be bound.

## Valid Usage

- VUID-VkBindVideoSessionMemoryInfoKHR-memoryOffset-07201
  `memoryOffset` **must** be less than the size of `memory`
- VUID-VkBindVideoSessionMemoryInfoKHR-memorySize-07202

memorySize **must** be less than or equal to the size of memory minus memoryOffset

### Valid Usage (Implicit)

- VUID-VkBindVideoSessionMemoryInfoKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_BIND_VIDEO_SESSION_MEMORY_INFO_KHR
- VUID-VkBindVideoSessionMemoryInfoKHR-pNext-pNext
  pNext **must** be NULL
- VUID-VkBindVideoSessionMemoryInfoKHR-memory-parameter
  memory **must** be a valid VkDeviceMemory handle

## 42.6. Video Profile Compatibility

Resources and query pools used with a particular video session **must** be compatible with the video profile the video session was created with.

A VkBuffer is compatible with a video profile if it was created with the VkBufferCreateInfo::pNext chain including a VkVideoProfileListInfoKHR structure with its pProfiles member containing an element matching the VkVideoProfileInfoKHR structure chain describing the video profile, and VkBufferCreateInfo::usage including at least one bit specific to video coding usage.

- VK_BUFFER_USAGE_VIDEO_DECODE_SRC_BIT_KHR
- VK_BUFFER_USAGE_VIDEO_DECODE_DST_BIT_KHR
- VK_BUFFER_USAGE_VIDEO_ENCODE_SRC_BIT_KHR
- VK_BUFFER_USAGE_VIDEO_ENCODE_DST_BIT_KHR

A VkBuffer is also compatible with a video profile if it was created with VkBufferCreateInfo::flags including VK_BUFFER_CREATE_VIDEO_PROFILE_INDEPENDENT_BIT_KHR.

A VkImage is compatible with a video profile if it was created with the VkImageCreateInfo::pNext chain including a VkVideoProfileListInfoKHR structure with its pProfiles member containing an element matching the VkVideoProfileInfoKHR structure chain describing the video profile, and VkImageCreateInfo::usage including at least one bit specific to video coding usage.

- VK_IMAGE_USAGE_VIDEO_DECODE_SRC_BIT_KHR
- VK_IMAGE_USAGE_VIDEO_DECODE_DST_BIT_KHR
- VK_IMAGE_USAGE_VIDEO_DECODE_DPB_BIT_KHR
- VK_IMAGE_USAGE_VIDEO_ENCODE_SRC_BIT_KHR
- VK_IMAGE_USAGE_VIDEO_ENCODE_DST_BIT_KHR
- VK_IMAGE_USAGE_VIDEO_ENCODE_DPB_BIT_KHR

A VkImage is also compatible with a video profile if all of the following conditions are true for the VkImageCreateInfo structure the image was created with:

- VkImageCreateInfo::flags included VK_IMAGE_CREATE_VIDEO_PROFILE_INDEPENDENT_BIT_KHR.
- The list of VkVideoFormatPropertiesKHR structures, obtained by calling vkGetPhysicalDeviceVideoFormatPropertiesKHR with VkPhysicalDeviceVideoFormatInfoKHR::imageUsage equal to the VkImageCreateInfo::usage the image was created with and the VkPhysicalDeviceVideoFormatInfoKHR::pNext chain including a VkVideoProfileListInfoKHR structure with its pProfiles member containing a single array element specifying the VkVideoProfileInfoKHR structure chain describing the video profile in question, contains an element for which all of the following conditions are true with respect to the VkImageCreateInfo structure the image was created with:
  - VkImageCreateInfo::format equals VkVideoFormatPropertiesKHR::format.
  - VkImageCreateInfo::flags only contains bits also set in VkVideoFormatPropertiesKHR::imageCreateFlags.
  - VkImageCreateInfo::imageType equals VkVideoFormatPropertiesKHR::imageType.
  - VkImageCreateInfo::tiling equals VkVideoFormatPropertiesKHR::imageTiling.
  - VkImageCreateInfo::usage only contains bits also set in VkVideoFormatPropertiesKHR::imageUsageFlags.

**NOTE** While some of these rules allow creating buffer or image resources that **may** be compatible with any video profile, applications **should** still prefer to include the specific video profiles the buffer or image resource is expected to be used with (through a VkVideoProfileListInfoKHR structure included in the pNext chain of the corresponding create info structure) whenever the information about the complete set of video profiles is available at resource creation time, to enable the implementation to optimize the created resource for the specific use case. In the absence of that information, the implementation **may** have to make conservative decisions about the memory requirements or representation of the resource.

A VkImageView is compatible with a video profile if the VkImage it was created from is also compatible with that video profile.

A VkQueryPool is compatible with a video profile if it was created with the VkQueryPoolCreateInfo::pNext chain including a VkVideoProfileInfoKHR structure chain describing the same video profile, and VkQueryPoolCreateInfo::queryType having one of the following values:

- VK_QUERY_TYPE_RESULT_STATUS_ONLY_KHR
- VK_QUERY_TYPE_VIDEO_ENCODE_FEEDBACK_KHR

## 42.7. Video Session Parameters

Video session parameters objects **can** store preprocessed codec-specific parameters used with a compatible video session, and enable reducing the number of parameters needed to be provided and processed by the implementation while recording video coding operations into command buffers.

Parameters stored in such objects are *immutable* to facilitate the concurrent use of the stored

parameters in multiple threads. At the same time, new parameters **can** be added to existing objects using the vkUpdateVideoSessionParametersKHR command.

In order to support concurrent use of the stored immutable parameters while also allowing the video session parameters object to be extended with new parameters, each video session parameters object maintains an *update sequence counter* that is set to `0` at object creation time and **must** be incremented by each subsequent update operation.

Certain video sequences that adhere to particular video compression standards permit updating previously supplied parameters. If a parameter update is necessary, the application has the following options:

- Cache the set of parameters on the application side and create a new video session parameters object adding all the parameters with appropriate changes, as necessary; or
- Create a new video session parameters object providing only the updated parameters and the previously used object as the template, which ensures that parameters not specified at creation time will be copied unmodified from the template object.

The actual types of parameters that **can** be stored and the capacity for individual parameter types, and the methods of initializing, updating, and referring to individual parameters are specific to the video codec operation the video session parameters object was created with.

- For `VK_VIDEO_CODEC_OPERATION_DECODE_H264_BIT_KHR` these are defined in the H.264 Decode Parameter Sets section.
- For `VK_VIDEO_CODEC_OPERATION_DECODE_H265_BIT_KHR` these are defined in the H.265 Decode Parameter Sets section.
- For `VK_VIDEO_CODEC_OPERATION_DECODE_AV1_BIT_KHR` these are defined in the AV1 Decode Parameter Sets section.
- For `VK_VIDEO_CODEC_OPERATION_ENCODE_H264_BIT_KHR` these are defined in the H.264 Encode Parameter Sets section.
- For `VK_VIDEO_CODEC_OPERATION_ENCODE_H265_BIT_KHR` these are defined in the H.265 Encode Parameter Sets section.

Video session parameters objects created with an encode operation are further specialized based on the video encode quality level the video session parameters are used with, as implementations **may** apply different sets of parameter overrides depending on the used quality level. This enables implementations to store the potentially optimized set of parameters in these objects, further limiting the necessary processing required while recording video encode operations into command buffers.

Video session parameters are represented by `VkVideoSessionParametersKHR` handles:

```
// Provided by VK_KHR_video_queue
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkVideoSessionParametersKHR)
```

### 42.7.1. Creating Video Session Parameters

To create a video session parameters object, call:

```
// Provided by VK_KHR_video_queue
VkResult vkCreateVideoSessionParametersKHR(
    VkDevice                                    device,
    const VkVideoSessionParametersCreateInfoKHR* pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkVideoSessionParametersKHR*                pVideoSessionParameters);
```

- `device` is the logical device that creates the video session parameters object.
- `pCreateInfo` is a pointer to VkVideoSessionParametersCreateInfoKHR structure containing parameters to be used to create the video session parameters object.
- `pAllocator` controls host memory allocation as described in the Memory Allocation chapter.
- `pVideoSessionParameters` is a pointer to a VkVideoSessionParametersKHR handle in which the resulting video session parameters object is returned.

The resulting video session parameters object is said to be created with the video codec operation `pCreateInfo->videoSession` was created with.

Video session parameters objects created with an encode operation are always created with respect to a video encode quality level. By default, the created video session parameters objects are created with quality level zero, unless otherwise specified by including a VkVideoEncodeQualityLevelInfoKHR structure in the `pCreateInfo->pNext` chain, in which case the video session parameters object is created with the quality level specified in VkVideoEncodeQualityLevelInfoKHR::`qualityLevel`.

If `pCreateInfo->videoSessionParametersTemplate` is not `VK_NULL_HANDLE`, then it will be used as a template for constructing the new video session parameters object. This happens by first adding any parameters according to the additional creation parameters provided in the `pCreateInfo->pNext` chain, followed by adding any parameters from the template object that have a key that does not match the key of any of the already added parameters.

For video session parameters objects created with an encode operation, the template object specified in `pCreateInfo->videoSessionParametersTemplate` **must** have been created with the same video encode quality level as the newly created object.

**NOTE** This means that codec-specific parameters stored in video session parameters objects **can** only be reused across different video encode quality levels by re-specifying them, as previously created video session parameters against other quality levels **cannot** be used as template because the original codec-specific parameters (before the implementation **may** have applied parameter overrides) **may** no longer be available in them for the purposes of constructing the derived object.

If `pCreateInfo->videoSession` was created with the video codec operation

VK_VIDEO_CODEC_OPERATION_DECODE_H264_BIT_KHR, then the created video session parameters object will initially contain the following sets of parameter entries:

- StdVideoH264SequenceParameterSet structures representing H.264 SPS entries, as follows:
  - If the pParametersAddInfo member of the VkVideoDecodeH264SessionParametersCreateInfoKHR structure provided in the pCreateInfo->pNext chain is not NULL, then the set of StdVideoH264SequenceParameterSet entries specified in pParametersAddInfo->pStdSPSs are added first;
  - If pCreateInfo->videoSessionParametersTemplate is not VK_NULL_HANDLE, then each StdVideoH264SequenceParameterSet entry stored in it is copied to the created video session parameters object if the created object does not already contain such an entry with the same seq_parameter_set_id.
- StdVideoH264PictureParameterSet structures representing H.264 PPS entries, as follows:
  - If the pParametersAddInfo member of the VkVideoDecodeH264SessionParametersCreateInfoKHR structure provided in the pCreateInfo->pNext chain is not NULL, then the set of StdVideoH264PictureParameterSet entries specified in pParametersAddInfo->pStdPPSs are added first;
  - If pCreateInfo->videoSessionParametersTemplate is not VK_NULL_HANDLE, then each StdVideoH264PictureParameterSet entry stored in it is copied to the created video session parameters object if the created object does not already contain such an entry with the same seq_parameter_set_id and pic_parameter_set_id.

If pCreateInfo->videoSession was created with the video codec operation VK_VIDEO_CODEC_OPERATION_DECODE_H265_BIT_KHR, then the created video session parameters object will initially contain the following sets of parameter entries:

- StdVideoH265VideoParameterSet structures representing H.265 VPS entries, as follows:
  - If the pParametersAddInfo member of the VkVideoDecodeH265SessionParametersCreateInfoKHR structure provided in the pCreateInfo->pNext chain is not NULL, then the set of StdVideoH265VideoParameterSet entries specified in pParametersAddInfo->pStdVPSs are added first;
  - If pCreateInfo->videoSessionParametersTemplate is not VK_NULL_HANDLE, then each StdVideoH265VideoParameterSet entry stored in it is copied to the created video session parameters object if the created object does not already contain such an entry with the same vps_video_parameter_set_id.
- StdVideoH265SequenceParameterSet structures representing H.265 SPS entries, as follows:
  - If the pParametersAddInfo member of the VkVideoDecodeH265SessionParametersCreateInfoKHR structure provided in the pCreateInfo->pNext chain is not NULL, then the set of StdVideoH265SequenceParameterSet entries specified in pParametersAddInfo->pStdSPSs are added first;
  - If pCreateInfo->videoSessionParametersTemplate is not VK_NULL_HANDLE, then each StdVideoH265SequenceParameterSet entry stored in it is copied to the created video session parameters object if the created object does not already contain such an entry with the same sps_video_parameter_set_id and sps_seq_parameter_set_id.

- StdVideoH265PictureParameterSet structures representing H.265 PPS entries, as follows:
  - If the pParametersAddInfo member of the VkVideoDecodeH265SessionParametersCreateInfoKHR structure provided in the pCreateInfo->pNext chain is not NULL, then the set of StdVideoH265PictureParameterSet entries specified in pParametersAddInfo->pStdPPSs are added first;
  - If pCreateInfo->videoSessionParametersTemplate is not VK_NULL_HANDLE, then each StdVideoH265PictureParameterSet entry stored in it is copied to the created video session parameters object if the created object does not already contain such an entry with the same sps_video_parameter_set_id, pps_seq_parameter_set_id, and pps_pic_parameter_set_id.

If pCreateInfo->videoSession was created with the video codec operation VK_VIDEO_CODEC_OPERATION_DECODE_AV1_BIT_KHR, then the created video session parameters object will contain a single AV1 sequence header represented by a StdVideoAV1SequenceHeader structure specified through the pStdSequenceHeader member of the VkVideoDecodeAV1SessionParametersCreateInfoKHR structure provided in the pCreateInfo->pNext chain. As such video session parameters objects **can** only contain a single AV1 sequence header, it is not possible to use a previously created object as a template or subsequently update the created video session parameters object.

If pCreateInfo->videoSession was created with the video codec operation VK_VIDEO_CODEC_OPERATION_ENCODE_H264_BIT_KHR, then the created video session parameters object will initially contain the following sets of parameter entries:

- StdVideoH264SequenceParameterSet structures representing H.264 SPS entries, as follows:
  - If the pParametersAddInfo member of the VkVideoEncodeH264SessionParametersCreateInfoKHR structure provided in the pCreateInfo->pNext chain is not NULL, then the set of StdVideoH264SequenceParameterSet entries specified in pParametersAddInfo->pStdSPSs are added first;
  - If pCreateInfo->videoSessionParametersTemplate is not VK_NULL_HANDLE, then each StdVideoH264SequenceParameterSet entry stored in it is copied to the created video session parameters object if the created object does not already contain such an entry with the same seq_parameter_set_id.
- StdVideoH264PictureParameterSet structures representing H.264 PPS entries, as follows:
  - If the pParametersAddInfo member of the VkVideoEncodeH264SessionParametersCreateInfoKHR structure provided in the pCreateInfo->pNext chain is not NULL, then the set of StdVideoH264PictureParameterSet entries specified in pParametersAddInfo->pStdPPSs are added first;
  - If pCreateInfo->videoSessionParametersTemplate is not VK_NULL_HANDLE, then each StdVideoH264PictureParameterSet entry stored in it is copied to the created video session parameters object if the created object does not already contain such an entry with the same seq_parameter_set_id and pic_parameter_set_id.

If pCreateInfo->videoSession was created with the video codec operation VK_VIDEO_CODEC_OPERATION_ENCODE_H265_BIT_KHR, then the created video session parameters object will initially contain the following sets of parameter entries:

- StdVideoH265VideoParameterSet structures representing H.265 VPS entries, as follows:
  - If the pParametersAddInfo member of the VkVideoEncodeH265SessionParametersCreateInfoKHR structure provided in the pCreateInfo->pNext chain is not NULL, then the set of StdVideoH265VideoParameterSet entries specified in pParametersAddInfo->pStdVPSs are added first;
  - If pCreateInfo->videoSessionParametersTemplate is not VK_NULL_HANDLE, then each StdVideoH265VideoParameterSet entry stored in it is copied to the created video session parameters object if the created object does not already contain such an entry with the same vps_video_parameter_set_id.
- StdVideoH265SequenceParameterSet structures representing H.265 SPS entries, as follows:
  - If the pParametersAddInfo member of the VkVideoEncodeH265SessionParametersCreateInfoKHR structure provided in the pCreateInfo->pNext chain is not NULL, then the set of StdVideoH265SequenceParameterSet entries specified in pParametersAddInfo->pStdSPSs are added first;
  - If pCreateInfo->videoSessionParametersTemplate is not VK_NULL_HANDLE, then each StdVideoH265SequenceParameterSet entry stored in it is copied to the created video session parameters object if the created object does not already contain such an entry with the same sps_video_parameter_set_id and sps_seq_parameter_set_id.
- StdVideoH265PictureParameterSet structures representing H.265 PPS entries, as follows:
  - If the pParametersAddInfo member of the VkVideoEncodeH265SessionParametersCreateInfoKHR structure provided in the pCreateInfo->pNext chain is not NULL, then the set of StdVideoH265PictureParameterSet entries specified in pParametersAddInfo->pStdPPSs are added first;
  - If pCreateInfo->videoSessionParametersTemplate is not VK_NULL_HANDLE, then each StdVideoH265PictureParameterSet entry stored in it is copied to the created video session parameters object if the created object does not already contain such an entry with the same sps_video_parameter_set_id, pps_seq_parameter_set_id, and pps_pic_parameter_set_id.

In case of video session parameters objects created with a video encode operation, implementations **may** return the VK_ERROR_INVALID_VIDEO_STD_PARAMETERS_KHR error if any of the specified Video Std parameters do not adhere to the syntactic or semantic requirements of the used video compression standard, or if values derived from parameters according to the rules defined by the used video compression standard do not adhere to the capabilities of the video compression standard or the implementation.

| | |
|---|---|
| **NOTE** | Applications **should** not rely on the VK_ERROR_INVALID_VIDEO_STD_PARAMETERS_KHR error being returned by any command as a means to verify Video Std parameters, as implementations are not required to report the error in any specific set of cases. |

### Valid Usage (Implicit)

- VUID-vkCreateVideoSessionParametersKHR-device-parameter
  device **must** be a valid VkDevice handle
- VUID-vkCreateVideoSessionParametersKHR-pCreateInfo-parameter

pCreateInfo **must** be a valid pointer to a valid VkVideoSessionParametersCreateInfoKHR structure

- VUID-vkCreateVideoSessionParametersKHR-pAllocator-parameter
  If pAllocator is not NULL, pAllocator **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkCreateVideoSessionParametersKHR-pVideoSessionParameters-parameter
  pVideoSessionParameters **must** be a valid pointer to a VkVideoSessionParametersKHR handle

## Return Codes

**Success**

- VK_SUCCESS

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY
- VK_ERROR_INITIALIZATION_FAILED
- VK_ERROR_INVALID_VIDEO_STD_PARAMETERS_KHR

The VkVideoSessionParametersCreateInfoKHR structure is defined as:

```
// Provided by VK_KHR_video_queue
typedef struct VkVideoSessionParametersCreateInfoKHR {
    VkStructureType                             sType;
    const void*                                 pNext;
    VkVideoSessionParametersCreateFlagsKHR      flags;
    VkVideoSessionParametersKHR                 videoSessionParametersTemplate;
    VkVideoSessionKHR                           videoSession;
} VkVideoSessionParametersCreateInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- flags is reserved for future use.
- videoSessionParametersTemplate is VK_NULL_HANDLE or a valid handle to a VkVideoSessionParametersKHR object used as a template for constructing the new video session parameters object.
- videoSession is the video session object against which the video session parameters object is going to be created.

Limiting values are defined below that are referenced by the relevant valid usage statements of this structure.

- If videoSession was created with the codec operation VK_VIDEO_CODEC_OPERATION_DECODE_H264_BIT_KHR, then let StdVideoH264SequenceParameterSet spsAddList[] be the list of H.264 SPS entries to add to the created video session parameters object, defined as follows:
  - If the pParametersAddInfo member of the VkVideoDecodeH264SessionParametersCreateInfoKHR structure provided in the pNext chain is not NULL, then the set of StdVideoH264SequenceParameterSet entries specified in pParametersAddInfo->pStdSPSs are added to spsAddList;
  - If videoSessionParametersTemplate is not VK_NULL_HANDLE, then each StdVideoH264SequenceParameterSet entry stored in it with seq_parameter_set_id not matching any of the entries already in spsAddList is added to spsAddList.
- If videoSession was created with the codec operation VK_VIDEO_CODEC_OPERATION_DECODE_H264_BIT_KHR, then let StdVideoH264PictureParameterSet ppsAddList[] be the list of H.264 PPS entries to add to the created video session parameters object, defined as follows:
  - If the pParametersAddInfo member of the VkVideoDecodeH264SessionParametersCreateInfoKHR structure provided in the pNext chain is not NULL, then the set of StdVideoH264PictureParameterSet entries specified in pParametersAddInfo->pStdPPSs are added to ppsAddList;
  - If videoSessionParametersTemplate is not VK_NULL_HANDLE, then each StdVideoH264PictureParameterSet entry stored in it with seq_parameter_set_id or pic_parameter_set_id not matching any of the entries already in ppsAddList is added to ppsAddList.
- If videoSession was created with the codec operation VK_VIDEO_CODEC_OPERATION_DECODE_H265_BIT_KHR, then let StdVideoH265VideoParameterSet vpsAddList[] be the list of H.265 VPS entries to add to the created video session parameters object, defined as follows:
  - If the pParametersAddInfo member of the VkVideoDecodeH265SessionParametersCreateInfoKHR structure provided in the pNext chain is not NULL, then the set of StdVideoH265VideoParameterSet entries specified in pParametersAddInfo->pStdVPSs are added to vpsAddList;
  - If videoSessionParametersTemplate is not VK_NULL_HANDLE, then each StdVideoH265VideoParameterSet entry stored in it with vps_video_parameter_set_id not matching any of the entries already in vpsAddList is added to vpsAddList.
- If videoSession was created with the codec operation VK_VIDEO_CODEC_OPERATION_DECODE_H265_BIT_KHR, then let StdVideoH265SequenceParameterSet spsAddList[] be the list of H.265 SPS entries to add to the created video session parameters object, defined as follows:
  - If the pParametersAddInfo member of the VkVideoDecodeH265SessionParametersCreateInfoKHR structure provided in the pNext chain is not NULL, then the set of StdVideoH265SequenceParameterSet entries specified in pParametersAddInfo->pStdSPSs are added to spsAddList;
  - If videoSessionParametersTemplate is not VK_NULL_HANDLE, then each

StdVideoH265SequenceParameterSet entry stored in it with sps_video_parameter_set_id or sps_seq_parameter_set_id not matching any of the entries already in spsAddList is added to spsAddList.

- If videoSession was created with the codec operation VK_VIDEO_CODEC_OPERATION_DECODE_H265_BIT_KHR, then let StdVideoH265PictureParameterSet ppsAddList[] be the list of H.265 PPS entries to add to the created video session parameters object, defined as follows:
  - If the pParametersAddInfo member of the VkVideoDecodeH265SessionParametersCreateInfoKHR structure provided in the pNext chain is not NULL, then the set of StdVideoH265PictureParameterSet entries specified in pParametersAddInfo->pStdPPSs are added to ppsAddList;
  - If videoSessionParametersTemplate is not VK_NULL_HANDLE, then each StdVideoH265PictureParameterSet entry stored in it with sps_video_parameter_set_id, pps_seq_parameter_set_id, or pps_pic_parameter_set_id not matching any of the entries already in ppsAddList is added to ppsAddList.
- If videoSession was created with an encode operation, then let uint32_t qualityLevel be the video encode quality level of the created video session parameters object, defined as follows:
  - If the pNext chain of this structure includes a VkVideoEncodeQualityLevelInfoKHR structure, then qualityLevel is equal to VkVideoEncodeQualityLevelInfoKHR::qualityLevel.
  - Otherwise qualityLevel is 0
- If videoSession was created with the codec operation VK_VIDEO_CODEC_OPERATION_ENCODE_H264_BIT_KHR, then let StdVideoH264SequenceParameterSet spsAddList[] be the list of H.264 SPS entries to add to the created video session parameters object, defined as follows:
  - If the pParametersAddInfo member of the VkVideoEncodeH264SessionParametersCreateInfoKHR structure provided in the pNext chain is not NULL, then the set of StdVideoH264SequenceParameterSet entries specified in pParametersAddInfo->pStdSPSs are added to spsAddList;
  - If videoSessionParametersTemplate is not VK_NULL_HANDLE, then each StdVideoH264SequenceParameterSet entry stored in it with seq_parameter_set_id not matching any of the entries already in spsAddList is added to spsAddList.
- If videoSession was created with the codec operation VK_VIDEO_CODEC_OPERATION_ENCODE_H264_BIT_KHR, then let StdVideoH264PictureParameterSet ppsAddList[] be the list of H.264 PPS entries to add to the created video session parameters object, defined as follows:
  - If the pParametersAddInfo member of the VkVideoEncodeH264SessionParametersCreateInfoKHR structure provided in the pNext chain is not NULL, then the set of StdVideoH264PictureParameterSet entries specified in pParametersAddInfo->pStdPPSs are added to ppsAddList;
  - If videoSessionParametersTemplate is not VK_NULL_HANDLE, then each StdVideoH264PictureParameterSet entry stored in it with seq_parameter_set_id or pic_parameter_set_id not matching any of the entries already in ppsAddList is added to ppsAddList.

- If videoSession was created with the codec operation VK_VIDEO_CODEC_OPERATION_ENCODE_H265_BIT_KHR, then let StdVideoH265VideoParameterSet vpsAddList[] be the list of H.265 VPS entries to add to the created video session parameters object, defined as follows:
  - If the pParametersAddInfo member of the VkVideoEncodeH265SessionParametersCreateInfoKHR structure provided in the pNext chain is not NULL, then the set of StdVideoH265VideoParameterSet entries specified in pParametersAddInfo->pStdVPSs are added to vpsAddList;
  - If videoSessionParametersTemplate is not VK_NULL_HANDLE, then each StdVideoH265VideoParameterSet entry stored in it with vps_video_parameter_set_id not matching any of the entries already in vpsAddList is added to vpsAddList.
- If videoSession was created with the codec operation VK_VIDEO_CODEC_OPERATION_ENCODE_H265_BIT_KHR, then let StdVideoH265SequenceParameterSet spsAddList[] be the list of H.265 SPS entries to add to the created video session parameters object, defined as follows:
  - If the pParametersAddInfo member of the VkVideoEncodeH265SessionParametersCreateInfoKHR structure provided in the pNext chain is not NULL, then the set of StdVideoH265SequenceParameterSet entries specified in pParametersAddInfo->pStdSPSs are added to spsAddList;
  - If videoSessionParametersTemplate is not VK_NULL_HANDLE, then each StdVideoH265SequenceParameterSet entry stored in it with sps_video_parameter_set_id or sps_seq_parameter_set_id not matching any of the entries already in spsAddList is added to spsAddList.
- If videoSession was created with the codec operation VK_VIDEO_CODEC_OPERATION_ENCODE_H265_BIT_KHR, then let StdVideoH265PictureParameterSet ppsAddList[] be the list of H.265 PPS entries to add to the created video session parameters object, defined as follows:
  - If the pParametersAddInfo member of the VkVideoEncodeH265SessionParametersCreateInfoKHR structure provided in the pNext chain is not NULL, then the set of StdVideoH265PictureParameterSet entries specified in pParametersAddInfo->pStdPPSs are added to ppsAddList;
  - If videoSessionParametersTemplate is not VK_NULL_HANDLE, then each StdVideoH265PictureParameterSet entry stored in it with sps_video_parameter_set_id, pps_seq_parameter_set_id, or pps_pic_parameter_set_id not matching any of the entries already in ppsAddList is added to ppsAddList.

### Valid Usage

- VUID-VkVideoSessionParametersCreateInfoKHR-videoSessionParametersTemplate-04855
  If videoSessionParametersTemplate is not VK_NULL_HANDLE, it **must** have been created against videoSession
- VUID-VkVideoSessionParametersCreateInfoKHR-videoSessionParametersTemplate-08310
  If videoSessionParametersTemplate is not VK_NULL_HANDLE and videoSession was created with an encode operation, then qualityLevel **must** equal the video encode quality level

videoSessionParametersTemplate was created with

- VUID-VkVideoSessionParametersCreateInfoKHR-videoSession-07203
  If videoSession was created with the video codec operation VK_VIDEO_CODEC_OPERATION_DECODE_H264_BIT_KHR, then the pNext chain **must** include a VkVideoDecodeH264SessionParametersCreateInfoKHR structure
- VUID-VkVideoSessionParametersCreateInfoKHR-videoSession-07204
  If videoSession was created with the video codec operation VK_VIDEO_CODEC_OPERATION_DECODE_H264_BIT_KHR, then the number of elements of spsAddList **must** be less than or equal to the maxStdSPSCount specified in the VkVideoDecodeH264SessionParametersCreateInfoKHR structure included in the pNext chain
- VUID-VkVideoSessionParametersCreateInfoKHR-videoSession-07205
  If videoSession was created with the video codec operation VK_VIDEO_CODEC_OPERATION_DECODE_H264_BIT_KHR, then the number of elements of ppsAddList **must** be less than or equal to the maxStdPPSCount specified in the VkVideoDecodeH264SessionParametersCreateInfoKHR structure included in the pNext chain
- VUID-VkVideoSessionParametersCreateInfoKHR-videoSession-07206
  If videoSession was created with the video codec operation VK_VIDEO_CODEC_OPERATION_DECODE_H265_BIT_KHR, then the pNext chain **must** include a VkVideoDecodeH265SessionParametersCreateInfoKHR structure
- VUID-VkVideoSessionParametersCreateInfoKHR-videoSession-07207
  If videoSession was created with the video codec operation VK_VIDEO_CODEC_OPERATION_DECODE_H265_BIT_KHR, then the number of elements of vpsAddList **must** be less than or equal to the maxStdVPSCount specified in the VkVideoDecodeH265SessionParametersCreateInfoKHR structure included in the pNext chain
- VUID-VkVideoSessionParametersCreateInfoKHR-videoSession-07208
  If videoSession was created with the video codec operation VK_VIDEO_CODEC_OPERATION_DECODE_H265_BIT_KHR, then the number of elements of spsAddList **must** be less than or equal to the maxStdSPSCount specified in the VkVideoDecodeH265SessionParametersCreateInfoKHR structure included in the pNext chain
- VUID-VkVideoSessionParametersCreateInfoKHR-videoSession-07209
  If videoSession was created with the video codec operation VK_VIDEO_CODEC_OPERATION_DECODE_H265_BIT_KHR, then the number of elements of ppsAddList **must** be less than or equal to the maxStdPPSCount specified in the VkVideoDecodeH265SessionParametersCreateInfoKHR structure included in the pNext chain
- VUID-VkVideoSessionParametersCreateInfoKHR-videoSession-09258
  If videoSession was created with the video codec operation VK_VIDEO_CODEC_OPERATION_DECODE_AV1_BIT_KHR, then videoSessionParametersTemplate **must** be VK_NULL_HANDLE
- VUID-VkVideoSessionParametersCreateInfoKHR-videoSession-09259

If videoSession was created with the video codec operation VK_VIDEO_CODEC_OPERATION_DECODE_AV1_BIT_KHR, then the pNext chain **must** include a VkVideoDecodeAV1SessionParametersCreateInfoKHR structure

- VUID-VkVideoSessionParametersCreateInfoKHR-videoSession-07210
  If videoSession was created with the video codec operation VK_VIDEO_CODEC_OPERATION_ENCODE_H264_BIT_KHR, then the pNext chain **must** include a VkVideoEncodeH264SessionParametersCreateInfoKHR structure
- VUID-VkVideoSessionParametersCreateInfoKHR-videoSession-04839
  If videoSession was created with the video codec operation VK_VIDEO_CODEC_OPERATION_ENCODE_H264_BIT_KHR, then the number of elements of spsAddList **must** be less than or equal to the maxStdSPSCount specified in the VkVideoEncodeH264SessionParametersCreateInfoKHR structure included in the pNext chain
- VUID-VkVideoSessionParametersCreateInfoKHR-videoSession-04840
  If videoSession was created with the video codec operation VK_VIDEO_CODEC_OPERATION_ENCODE_H264_BIT_KHR, then the number of elements of ppsAddList **must** be less than or equal to the maxStdPPSCount specified in the VkVideoEncodeH264SessionParametersCreateInfoKHR structure included in the pNext chain
- VUID-VkVideoSessionParametersCreateInfoKHR-videoSession-07211
  If videoSession was created with the video codec operation VK_VIDEO_CODEC_OPERATION_ENCODE_H265_BIT_KHR, then the pNext chain **must** include a VkVideoEncodeH265SessionParametersCreateInfoKHR structure
- VUID-VkVideoSessionParametersCreateInfoKHR-videoSession-04841
  If videoSession was created with the video codec operation VK_VIDEO_CODEC_OPERATION_ENCODE_H265_BIT_KHR, then the number of elements of vpsAddList **must** be less than or equal to the maxStdVPSCount specified in the VkVideoEncodeH265SessionParametersCreateInfoKHR structure included in the pNext chain
- VUID-VkVideoSessionParametersCreateInfoKHR-videoSession-04842
  If videoSession was created with the video codec operation VK_VIDEO_CODEC_OPERATION_ENCODE_H265_BIT_KHR, then the number of elements of spsAddList **must** be less than or equal to the maxStdSPSCount specified in the VkVideoEncodeH265SessionParametersCreateInfoKHR structure included in the pNext chain
- VUID-VkVideoSessionParametersCreateInfoKHR-videoSession-04843
  If videoSession was created with the video codec operation VK_VIDEO_CODEC_OPERATION_ENCODE_H265_BIT_KHR, then the number of elements of ppsAddList **must** be less than or equal to the maxStdPPSCount specified in the VkVideoEncodeH265SessionParametersCreateInfoKHR structure included in the pNext chain
- VUID-VkVideoSessionParametersCreateInfoKHR-videoSession-08319
  If videoSession was created with the video codec operation VK_VIDEO_CODEC_OPERATION_ENCODE_H265_BIT_KHR, then num_tile_columns_minus1 **must** be less than VkVideoEncodeH265CapabilitiesKHR::maxTiles.width, as returned by

vkGetPhysicalDeviceVideoCapabilitiesKHR for the video profile videoSession was created with, for each element of ppsAddList

- VUID-VkVideoSessionParametersCreateInfoKHR-videoSession-08320
  If videoSession was created with the video codec operation VK_VIDEO_CODEC_OPERATION_ENCODE_H265_BIT_KHR, then num_tile_rows_minus1 **must** be less than VkVideoEncodeH265CapabilitiesKHR::maxTiles.height, as returned by vkGetPhysicalDeviceVideoCapabilitiesKHR for the video profile videoSession was created with, for each element of ppsAddList

## Valid Usage (Implicit)

- VUID-VkVideoSessionParametersCreateInfoKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_VIDEO_SESSION_PARAMETERS_CREATE_INFO_KHR
- VUID-VkVideoSessionParametersCreateInfoKHR-pNext-pNext
  Each pNext member of any structure (including this one) in the pNext chain **must** be either NULL or a pointer to a valid instance of VkVideoDecodeAV1SessionParametersCreateInfoKHR, VkVideoDecodeH264SessionParametersCreateInfoKHR, VkVideoDecodeH265SessionParametersCreateInfoKHR, VkVideoEncodeH264SessionParametersCreateInfoKHR, VkVideoEncodeH265SessionParametersCreateInfoKHR, or VkVideoEncodeQualityLevelInfoKHR
- VUID-VkVideoSessionParametersCreateInfoKHR-sType-unique
  The sType value of each struct in the pNext chain **must** be unique
- VUID-VkVideoSessionParametersCreateInfoKHR-flags-zerobitmask
  flags **must** be 0
- VUID-VkVideoSessionParametersCreateInfoKHR-videoSessionParametersTemplate-parameter
  If videoSessionParametersTemplate is not VK_NULL_HANDLE, videoSessionParametersTemplate **must** be a valid VkVideoSessionParametersKHR handle
- VUID-VkVideoSessionParametersCreateInfoKHR-videoSession-parameter
  videoSession **must** be a valid VkVideoSessionKHR handle
- VUID-VkVideoSessionParametersCreateInfoKHR-videoSessionParametersTemplate-parent
  If videoSessionParametersTemplate is a valid handle, it **must** have been created, allocated, or retrieved from videoSession
- VUID-VkVideoSessionParametersCreateInfoKHR-commonparent
  Both of videoSession, and videoSessionParametersTemplate that are valid handles of non-ignored parameters **must** have been created, allocated, or retrieved from the same VkDevice

```
// Provided by VK_KHR_video_queue
typedef VkFlags VkVideoSessionParametersCreateFlagsKHR;
```

VkVideoSessionParametersCreateFlagsKHR is a bitmask type for setting a mask, but is currently reserved for future use.

### 42.7.2. Destroying Video Session Parameters

To destroy a video session parameters object, call:

```
// Provided by VK_KHR_video_queue
void vkDestroyVideoSessionParametersKHR(
    VkDevice                                    device,
    VkVideoSessionParametersKHR                 videoSessionParameters,
    const VkAllocationCallbacks*                pAllocator);
```

- device is the logical device that destroys the video session parameters object.
- videoSessionParameters is the video session parameters object to destroy.
- pAllocator controls host memory allocation as described in the Memory Allocation chapter.

#### Valid Usage

- VUID-vkDestroyVideoSessionParametersKHR-videoSessionParameters-07212
  All submitted commands that refer to videoSessionParameters **must** have completed execution
- VUID-vkDestroyVideoSessionParametersKHR-videoSessionParameters-07213
  If VkAllocationCallbacks were provided when videoSessionParameters was created, a compatible set of callbacks **must** be provided here
- VUID-vkDestroyVideoSessionParametersKHR-videoSessionParameters-07214
  If no VkAllocationCallbacks were provided when videoSessionParameters was created, pAllocator **must** be NULL

#### Valid Usage (Implicit)

- VUID-vkDestroyVideoSessionParametersKHR-device-parameter
  device **must** be a valid VkDevice handle
- VUID-vkDestroyVideoSessionParametersKHR-videoSessionParameters-parameter
  If videoSessionParameters is not VK_NULL_HANDLE, videoSessionParameters **must** be a valid VkVideoSessionParametersKHR handle
- VUID-vkDestroyVideoSessionParametersKHR-pAllocator-parameter
  If pAllocator is not NULL, pAllocator **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkDestroyVideoSessionParametersKHR-videoSessionParameters-parent
  If videoSessionParameters is a valid handle, it **must** have been created, allocated, or retrieved from device

**Host Synchronization**

- Host access to videoSessionParameters **must** be externally synchronized

### 42.7.3. Updating Video Session Parameters

To update video session parameters object with new parameters, call:

```
// Provided by VK_KHR_video_queue
VkResult vkUpdateVideoSessionParametersKHR(
    VkDevice                                    device,
    VkVideoSessionParametersKHR                 videoSessionParameters,
    const VkVideoSessionParametersUpdateInfoKHR* pUpdateInfo);
```

- device is the logical device that updates the video session parameters.
- videoSessionParameters is the video session parameters object to update.
- pUpdateInfo is a pointer to a VkVideoSessionParametersUpdateInfoKHR structure specifying the parameter update information.

After a successful call to this command, the update sequence counter of videoSessionParameters is changed to the value specified in pUpdateInfo->updateSequenceCount.

**NOTE** As each update issued to a video session parameters object needs to specify the next available update sequence count value, concurrent updates of the same video session parameters object are inherently disallowed. However, recording video coding operations to command buffers referring to parameters previously added to the video session parameters object is allowed, even if there is a concurrent update in progress adding some new entries to the object.

If videoSessionParameters was created with the video codec operation VK_VIDEO_CODEC_OPERATION_DECODE_H264_BIT_KHR and the pUpdateInfo->pNext chain includes a VkVideoDecodeH264SessionParametersAddInfoKHR structure, then this command adds the following parameter entries to videoSessionParameters:

- The H.264 SPS entries specified in VkVideoDecodeH264SessionParametersAddInfoKHR::pStdSPSs.
- The H.264 PPS entries specified in VkVideoDecodeH264SessionParametersAddInfoKHR::pStdPPSs.

If videoSessionParameters was created with the video codec operation VK_VIDEO_CODEC_OPERATION_DECODE_H265_BIT_KHR and the pUpdateInfo->pNext chain includes a VkVideoDecodeH265SessionParametersAddInfoKHR structure, then this command adds the following parameter entries to videoSessionParameters:

- The H.265 VPS entries specified in VkVideoDecodeH265SessionParametersAddInfoKHR::pStdVPSs.

- The H.265 SPS entries specified in VkVideoDecodeH265SessionParametersAddInfoKHR::pStdSPSs.
- The H.265 PPS entries specified in VkVideoDecodeH265SessionParametersAddInfoKHR::pStdPPSs.

If videoSessionParameters was created with the video codec operation VK_VIDEO_CODEC_OPERATION_ENCODE_H264_BIT_KHR and the pUpdateInfo->pNext chain includes a VkVideoEncodeH264SessionParametersAddInfoKHR structure, then this command adds the following parameter entries to videoSessionParameters:

- The H.264 SPS entries specified in VkVideoEncodeH264SessionParametersAddInfoKHR::pStdSPSs.
- The H.264 PPS entries specified in VkVideoEncodeH264SessionParametersAddInfoKHR::pStdPPSs.

If videoSessionParameters was created with the video codec operation VK_VIDEO_CODEC_OPERATION_ENCODE_H265_BIT_KHR and the pUpdateInfo->pNext chain includes a VkVideoEncodeH265SessionParametersAddInfoKHR structure, then this command adds the following parameter entries to videoSessionParameters:

- The H.265 VPS entries specified in VkVideoEncodeH265SessionParametersAddInfoKHR::pStdVPSs.
- The H.265 SPS entries specified in VkVideoEncodeH265SessionParametersAddInfoKHR::pStdSPSs.
- The H.265 PPS entries specified in VkVideoEncodeH265SessionParametersAddInfoKHR::pStdPPSs.

In case of video session parameters objects created with a video encode operation, implementations **may** return the VK_ERROR_INVALID_VIDEO_STD_PARAMETERS_KHR error if any of the specified Video Std parameters do not adhere to the syntactic or semantic requirements of the used video compression standard, or if values derived from parameters according to the rules defined by the used video compression standard do not adhere to the capabilities of the video compression standard or the implementation.

**NOTE** Applications **should** not rely on the VK_ERROR_INVALID_VIDEO_STD_PARAMETERS_KHR error being returned by any command as a means to verify Video Std parameters, as implementations are not required to report the error in any specific set of cases.

### Valid Usage

- VUID-vkUpdateVideoSessionParametersKHR-pUpdateInfo-07215
  pUpdateInfo->updateSequenceCount **must** equal the current update sequence counter of videoSessionParameters plus one
- VUID-vkUpdateVideoSessionParametersKHR-videoSessionParameters-07216
  If videoSessionParameters was created with the video codec operation VK_VIDEO_CODEC_OPERATION_DECODE_H264_BIT_KHR and the pNext chain of pUpdateInfo includes

a VkVideoDecodeH264SessionParametersAddInfoKHR structure, then videoSessionParameters **must** not already contain a StdVideoH264SequenceParameterSet entry with seq_parameter_set_id matching any of the elements of VkVideoDecodeH264SessionParametersAddInfoKHR::pStdSPSs

- VUID-vkUpdateVideoSessionParametersKHR-videoSessionParameters-07217
  If videoSessionParameters was created with the video codec operation VK_VIDEO_CODEC_OPERATION_DECODE_H264_BIT_KHR, then the number of StdVideoH264SequenceParameterSet entries already stored in it plus the value of the stdSPSCount member of the VkVideoDecodeH264SessionParametersAddInfoKHR structure included in the pUpdateInfo->pNext chain **must** be less than or equal to the VkVideoDecodeH264SessionParametersCreateInfoKHR::maxStdSPSCount videoSessionParameters was created with
- VUID-vkUpdateVideoSessionParametersKHR-videoSessionParameters-07218
  If videoSessionParameters was created with the video codec operation VK_VIDEO_CODEC_OPERATION_DECODE_H264_BIT_KHR and the pNext chain of pUpdateInfo includes a VkVideoDecodeH264SessionParametersAddInfoKHR structure, then videoSessionParameters **must** not already contain a StdVideoH264PictureParameterSet entry with both seq_parameter_set_id and pic_parameter_set_id matching any of the elements of VkVideoDecodeH264SessionParametersAddInfoKHR::pStdPPSs
- VUID-vkUpdateVideoSessionParametersKHR-videoSessionParameters-07219
  If videoSessionParameters was created with the video codec operation VK_VIDEO_CODEC_OPERATION_DECODE_H264_BIT_KHR, then the number of StdVideoH264PictureParameterSet entries already stored in it plus the value of the stdPPSCount member of the VkVideoDecodeH264SessionParametersAddInfoKHR structure included in the pUpdateInfo->pNext chain **must** be less than or equal to the VkVideoDecodeH264SessionParametersCreateInfoKHR::maxStdPPSCount videoSessionParameters was created with
- VUID-vkUpdateVideoSessionParametersKHR-videoSessionParameters-07220
  If videoSessionParameters was created with the video codec operation VK_VIDEO_CODEC_OPERATION_DECODE_H265_BIT_KHR and the pNext chain of pUpdateInfo includes a VkVideoDecodeH265SessionParametersAddInfoKHR structure, then videoSessionParameters **must** not already contain a StdVideoH265VideoParameterSet entry with vps_video_parameter_set_id matching any of the elements of VkVideoDecodeH265SessionParametersAddInfoKHR::pStdVPSs
- VUID-vkUpdateVideoSessionParametersKHR-videoSessionParameters-07221
  If videoSessionParameters was created with the video codec operation VK_VIDEO_CODEC_OPERATION_DECODE_H265_BIT_KHR, then the number of StdVideoH265VideoParameterSet entries already stored in it plus the value of the stdVPSCount member of the VkVideoDecodeH265SessionParametersAddInfoKHR structure included in the pUpdateInfo->pNext chain **must** be less than or equal to the VkVideoDecodeH265SessionParametersCreateInfoKHR::maxStdVPSCount videoSessionParameters was created with
- VUID-vkUpdateVideoSessionParametersKHR-videoSessionParameters-07222
  If videoSessionParameters was created with the video codec operation VK_VIDEO_CODEC_OPERATION_DECODE_H265_BIT_KHR and the pNext chain of pUpdateInfo includes

a VkVideoDecodeH265SessionParametersAddInfoKHR structure, then videoSessionParameters **must** not already contain a StdVideoH265SequenceParameterSet entry with both sps_video_parameter_set_id and sps_seq_parameter_set_id matching any of the elements of VkVideoDecodeH265SessionParametersAddInfoKHR::pStdSPSs

- VUID-vkUpdateVideoSessionParametersKHR-videoSessionParameters-07223
  If videoSessionParameters was created with the video codec operation VK_VIDEO_CODEC_OPERATION_DECODE_H265_BIT_KHR, then the number of StdVideoH265SequenceParameterSet entries already stored in it plus the value of the stdSPSCount member of the VkVideoDecodeH265SessionParametersAddInfoKHR structure included in the pUpdateInfo->pNext chain **must** be less than or equal to the VkVideoDecodeH265SessionParametersCreateInfoKHR::maxStdSPSCount videoSessionParameters was created with
- VUID-vkUpdateVideoSessionParametersKHR-videoSessionParameters-07224
  If videoSessionParameters was created with the video codec operation VK_VIDEO_CODEC_OPERATION_DECODE_H265_BIT_KHR and the pNext chain of pUpdateInfo includes a VkVideoDecodeH265SessionParametersAddInfoKHR structure, then videoSessionParameters **must** not already contain a StdVideoH265PictureParameterSet entry with sps_video_parameter_set_id, pps_seq_parameter_set_id, and pps_pic_parameter_set_id all matching any of the elements of VkVideoDecodeH265SessionParametersAddInfoKHR ::pStdPPSs
- VUID-vkUpdateVideoSessionParametersKHR-videoSessionParameters-07225
  If videoSessionParameters was created with the video codec operation VK_VIDEO_CODEC_OPERATION_DECODE_H265_BIT_KHR, then the number of StdVideoH265PictureParameterSet entries already stored in it plus the value of the stdPPSCount member of the VkVideoDecodeH265SessionParametersAddInfoKHR structure included in the pUpdateInfo->pNext chain **must** be less than or equal to the VkVideoDecodeH265SessionParametersCreateInfoKHR::maxStdPPSCount videoSessionParameters was created with
- VUID-vkUpdateVideoSessionParametersKHR-videoSessionParameters-09260
  videoSessionParameters **must** not have been created with the video codec operation VK_VIDEO_CODEC_OPERATION_DECODE_AV1_BIT_KHR
- VUID-vkUpdateVideoSessionParametersKHR-videoSessionParameters-07226
  If videoSessionParameters was created with the video codec operation VK_VIDEO_CODEC_OPERATION_ENCODE_H264_BIT_KHR and the pNext chain of pUpdateInfo includes a VkVideoEncodeH264SessionParametersAddInfoKHR structure, then videoSessionParameters **must** not already contain a StdVideoH264SequenceParameterSet entry with seq_parameter_set_id matching any of the elements of VkVideoEncodeH264SessionParametersAddInfoKHR::pStdSPSs
- VUID-vkUpdateVideoSessionParametersKHR-videoSessionParameters-06441
  If videoSessionParameters was created with the video codec operation VK_VIDEO_CODEC_OPERATION_ENCODE_H264_BIT_KHR, then the number of StdVideoH264SequenceParameterSet entries already stored in it plus the value of the stdSPSCount member of the VkVideoEncodeH264SessionParametersAddInfoKHR structure included in the pUpdateInfo->pNext chain **must** be less than or equal to the VkVideoEncodeH264SessionParametersCreateInfoKHR::maxStdSPSCount

videoSessionParameters was created with

- VUID-vkUpdateVideoSessionParametersKHR-videoSessionParameters-07227
  If videoSessionParameters was created with the video codec operation VK_VIDEO_CODEC_OPERATION_ENCODE_H264_BIT_KHR and the pNext chain of pUpdateInfo includes a VkVideoEncodeH264SessionParametersAddInfoKHR structure, then videoSessionParameters **must** not already contain a StdVideoH264PictureParameterSet entry with both seq_parameter_set_id and pic_parameter_set_id matching any of the elements of VkVideoEncodeH264SessionParametersAddInfoKHR::pStdPPSs
- VUID-vkUpdateVideoSessionParametersKHR-videoSessionParameters-06442
  If videoSessionParameters was created with the video codec operation VK_VIDEO_CODEC_OPERATION_ENCODE_H264_BIT_KHR, then the number of StdVideoH264PictureParameterSet entries already stored in it plus the value of the stdPPSCount member of the VkVideoEncodeH264SessionParametersAddInfoKHR structure included in the pUpdateInfo->pNext chain **must** be less than or equal to the VkVideoEncodeH264SessionParametersCreateInfoKHR::maxStdPPSCount videoSessionParameters was created with
- VUID-vkUpdateVideoSessionParametersKHR-videoSessionParameters-07228
  If videoSessionParameters was created with the video codec operation VK_VIDEO_CODEC_OPERATION_ENCODE_H265_BIT_KHR and the pNext chain of pUpdateInfo includes a VkVideoEncodeH265SessionParametersAddInfoKHR structure, then videoSessionParameters **must** not already contain a StdVideoH265VideoParameterSet entry with vps_video_parameter_set_id matching any of the elements of VkVideoEncodeH265SessionParametersAddInfoKHR::pStdVPSs
- VUID-vkUpdateVideoSessionParametersKHR-videoSessionParameters-06443
  If videoSessionParameters was created with the video codec operation VK_VIDEO_CODEC_OPERATION_ENCODE_H265_BIT_KHR, then the number of StdVideoH265VideoParameterSet entries already stored in it plus the value of the stdVPSCount member of the VkVideoEncodeH265SessionParametersAddInfoKHR structure included in the pUpdateInfo->pNext chain **must** be less than or equal to the VkVideoEncodeH265SessionParametersCreateInfoKHR::maxStdVPSCount videoSessionParameters was created with
- VUID-vkUpdateVideoSessionParametersKHR-videoSessionParameters-07229
  If videoSessionParameters was created with the video codec operation VK_VIDEO_CODEC_OPERATION_ENCODE_H265_BIT_KHR and the pNext chain of pUpdateInfo includes a VkVideoEncodeH265SessionParametersAddInfoKHR structure, then videoSessionParameters **must** not already contain a StdVideoH265SequenceParameterSet entry with both sps_video_parameter_set_id and sps_seq_parameter_set_id matching any of the elements of VkVideoEncodeH265SessionParametersAddInfoKHR::pStdSPSs
- VUID-vkUpdateVideoSessionParametersKHR-videoSessionParameters-06444
  If videoSessionParameters was created with the video codec operation VK_VIDEO_CODEC_OPERATION_ENCODE_H265_BIT_KHR, then the number of StdVideoH265SequenceParameterSet entries already stored in it plus the value of the stdSPSCount member of the VkVideoEncodeH265SessionParametersAddInfoKHR structure included in the pUpdateInfo->pNext chain **must** be less than or equal to the VkVideoEncodeH265SessionParametersCreateInfoKHR::maxStdSPSCount

videoSessionParameters was created with

- VUID-vkUpdateVideoSessionParametersKHR-videoSessionParameters-07230
  If videoSessionParameters was created with the video codec operation VK_VIDEO_CODEC_OPERATION_ENCODE_H265_BIT_KHR and the pNext chain of pUpdateInfo includes a VkVideoEncodeH265SessionParametersAddInfoKHR structure, then videoSessionParameters **must** not already contain a StdVideoH265PictureParameterSet entry with sps_video_parameter_set_id, pps_seq_parameter_set_id, and pps_pic_parameter_set_id all matching any of the elements of VkVideoEncodeH265SessionParametersAddInfoKHR::pStdPPSs
- VUID-vkUpdateVideoSessionParametersKHR-videoSessionParameters-06445
  If videoSessionParameters was created with the video codec operation VK_VIDEO_CODEC_OPERATION_ENCODE_H265_BIT_KHR, then the number of StdVideoH265PictureParameterSet entries already stored in it plus the value of the stdPPSCount member of the VkVideoEncodeH265SessionParametersAddInfoKHR structure included in the pUpdateInfo->pNext chain **must** be less than or equal to the VkVideoEncodeH265SessionParametersCreateInfoKHR::maxStdPPSCount videoSessionParameters was created with
- VUID-vkUpdateVideoSessionParametersKHR-videoSessionParameters-08321
  If videoSessionParameters was created with the video codec operation VK_VIDEO_CODEC_OPERATION_ENCODE_H265_BIT_KHR and the pNext chain of pUpdateInfo includes a VkVideoEncodeH265SessionParametersAddInfoKHR structure, then num_tile_columns_minus1 **must** be less than VkVideoEncodeH265CapabilitiesKHR::maxTiles.width, as returned by vkGetPhysicalDeviceVideoCapabilitiesKHR for the video profile videoSessionParameters was created with, for each element of VkVideoEncodeH265SessionParametersAddInfoKHR::pStdPPSs
- VUID-vkUpdateVideoSessionParametersKHR-videoSessionParameters-08322
  If videoSessionParameters was created with the video codec operation VK_VIDEO_CODEC_OPERATION_ENCODE_H265_BIT_KHR and the pNext chain of pUpdateInfo includes a VkVideoEncodeH265SessionParametersAddInfoKHR structure, then num_tile_rows_minus1 **must** be less than VkVideoEncodeH265CapabilitiesKHR::maxTiles.height, as returned by vkGetPhysicalDeviceVideoCapabilitiesKHR for the video profile videoSessionParameters was created with, for each element of VkVideoEncodeH265SessionParametersAddInfoKHR::pStdPPSs

## Valid Usage (Implicit)

- VUID-vkUpdateVideoSessionParametersKHR-device-parameter
  device **must** be a valid VkDevice handle
- VUID-vkUpdateVideoSessionParametersKHR-videoSessionParameters-parameter
  videoSessionParameters **must** be a valid VkVideoSessionParametersKHR handle
- VUID-vkUpdateVideoSessionParametersKHR-pUpdateInfo-parameter
  pUpdateInfo **must** be a valid pointer to a valid VkVideoSessionParametersUpdateInfoKHR structure
- VUID-vkUpdateVideoSessionParametersKHR-videoSessionParameters-parent

videoSessionParameters **must** have been created, allocated, or retrieved from device

### Return Codes

**Success**

- VK_SUCCESS

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY
- VK_ERROR_INVALID_VIDEO_STD_PARAMETERS_KHR

The VkVideoSessionParametersUpdateInfoKHR structure is defined as:

```
// Provided by VK_KHR_video_queue
typedef struct VkVideoSessionParametersUpdateInfoKHR {
    VkStructureType    sType;
    const void*        pNext;
    uint32_t           updateSequenceCount;
} VkVideoSessionParametersUpdateInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- updateSequenceCount is the new update sequence count to set for the video session parameters object.

### Valid Usage (Implicit)

- VUID-VkVideoSessionParametersUpdateInfoKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_VIDEO_SESSION_PARAMETERS_UPDATE_INFO_KHR
- VUID-VkVideoSessionParametersUpdateInfoKHR-pNext-pNext
  Each pNext member of any structure (including this one) in the pNext chain **must** be either NULL or a pointer to a valid instance of VkVideoDecodeH264SessionParametersAddInfoKHR, VkVideoDecodeH265SessionParametersAddInfoKHR, VkVideoEncodeH264SessionParametersAddInfoKHR, or VkVideoEncodeH265SessionParametersAddInfoKHR
- VUID-VkVideoSessionParametersUpdateInfoKHR-sType-unique
  The sType value of each struct in the pNext chain **must** be unique

## 42.8. Video Coding Scope

Applications **can** record video coding commands for a video session only within a video coding scope.

To begin a video coding scope, call:

```
// Provided by VK_KHR_video_queue
void vkCmdBeginVideoCodingKHR(
    VkCommandBuffer                             commandBuffer,
    const VkVideoBeginCodingInfoKHR*            pBeginInfo);
```

- `commandBuffer` is the command buffer in which to record the command.
- `pBeginInfo` is a pointer to a VkVideoBeginCodingInfoKHR structure specifying the parameters of the video coding scope, including the video session and video session parameters object to use.

After beginning a video coding scope, the video session object specified in `pBeginInfo->videoSession` is *bound* to the command buffer, and the command buffer is ready to record video coding operations. Similarly, if `pBeginInfo->videoSessionParameters` is not `VK_NULL_HANDLE`, it is also *bound* to the command buffer, and video coding operations **can** refer to the codec-specific parameters stored in it.

This command also establishes the set of *bound reference picture resources* that **can** be used as reconstructed pictures or reference pictures within the video coding scope. Each element of this set consists of a video picture resource and the DPB slot index associated with it, if there is one.

The set of bound reference picture resources is immutable within a video coding scope, however, the DPB slot index associated with any of the bound reference picture resources **can** change during the video coding scope in response to video coding operations.

The VkVideoReferenceSlotInfoKHR structures provided as the elements of `pBeginInfo->pReferenceSlots` are interpreted by this command as follows:

- If `slotIndex` is non-negative and `pPictureResource` is not `NULL`, then the video picture resource defined by the VkVideoPictureResourceInfoKHR structure pointed to by `pPictureResource` is added to the set of bound reference picture resources and is associated with the DPB slot index specified in `slotIndex`.
- If `slotIndex` is non-negative and `pPictureResource` is `NULL`, then the DPB slot with index `slotIndex` is deactivated by this command.
- If `slotIndex` is negative and `pPictureResource` is not `NULL`, then the video picture resource defined by the VkVideoPictureResourceInfoKHR structure pointed to by `pPictureResource` is added to the set of bound reference picture resources without an associated DPB slot. Such a picture resource **can** be subsequently used as a reconstructed picture to associate it with a DPB slot.
- If `slotIndex` is negative and `pPictureResource` is `NULL`, then the element is ignored.

**NOTE** It is possible for multiple bound reference picture resources to be associated with the same DPB slot index, or for a single bound reference picture to refer to multiple

separate reference pictures. For example, in case of an H.264 decode profile with interlaced frame support a single DPB slot can refer to two separate pictures for the top and bottom fields. Depending on the picture layout used by the H.264 decode profile, the following special cases **may** arise:

- If the picture layout is `VK_VIDEO_DECODE_H264_PICTURE_LAYOUT_INTERLACED_INTERLEAVED_LINES_BIT_KHR`, then the top and bottom field pictures are physically co-located in the same video picture resource with even scanlines corresponding to the top field and odd scanlines corresponding to the bottom field, respectively.
- If the picture layout is `VK_VIDEO_DECODE_H264_PICTURE_LAYOUT_INTERLACED_SEPARATE_PLANES_BIT_KHR`, then the top and bottom field pictures are stored in separate video picture resources (in separate subregions of the same image layer, in separate layers of the same image, or in entirely separate images), hence two elements of VkVideoBeginCodingInfoKHR::`pReferenceSlots` **can** contain the same `slotIndex` but specify different video picture resources in their `pPictureResource` members.

All non-negative `slotIndex` values specified in the elements of `pBeginInfo->pReferenceSlots` **must** identify DPB slots of the video session that are in the active state at the time this command is executed on the device.

**NOTE** The application does not have to specify an entry in `pBeginInfo->pReferenceSlots` corresponding to all active DPB slots of the video session, but only for those which are intended to be used in the video coding scope. This way the application can avoid any potential runtime cost associated with binding the corresponding picture resources to the command buffer.

In case of a video encode session, the application is also responsible for providing information about the current rate control state configured for the video session by including an instance of the VkVideoEncodeRateControlInfoKHR structure in the `pNext` chain of `pBeginInfo`. If no VkVideoEncodeRateControlInfoKHR is included, then the presence of an empty VkVideoEncodeRateControlInfoKHR structure is implied which indicates that the current rate control mode is `VK_VIDEO_ENCODE_RATE_CONTROL_MODE_DEFAULT_KHR`. The specified state **must** match the effective rate control state configured for the video session at the time the recorded command is executed on the device.

**NOTE** Including an instance of the VkVideoEncodeRateControlInfoKHR structure in the `pNext` chain of `pBeginInfo` does not change the rate control state configured for the video session, but only specifies the expected rate control state configured at the time the recorded command is executed on the device which allows the implementation to have information about the configured rate control state at command buffer recording time. In order to change the current rate control state of a video session, the application has to issue an appropriate vkCmdControlVideoCodingKHR command as described in the Video Coding Control and Rate Control State sections.

## Valid Usage

- VUID-vkCmdBeginVideoCodingKHR-commandBuffer-07231
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support the video codec operation `pBeginInfo->videoSession` was created with, as returned by vkGetPhysicalDeviceQueueFamilyProperties2 in VkQueueFamilyVideoPropertiesKHR::`videoCodecOperations`
- VUID-vkCmdBeginVideoCodingKHR-None-07232
  There **must** be no active queries
- VUID-vkCmdBeginVideoCodingKHR-commandBuffer-07233
  If `commandBuffer` is an unprotected command buffer and `protectedNoFault` is not supported, then `pBeginInfo->videoSession` **must** not have been created with `VK_VIDEO_SESSION_CREATE_PROTECTED_CONTENT_BIT_KHR`
- VUID-vkCmdBeginVideoCodingKHR-commandBuffer-07234
  If `commandBuffer` is a protected command buffer and `protectedNoFault` is not supported, then `pBeginInfo->videoSession` **must** have been created with `VK_VIDEO_SESSION_CREATE_PROTECTED_CONTENT_BIT_KHR`
- VUID-vkCmdBeginVideoCodingKHR-commandBuffer-07235
  If `commandBuffer` is an unprotected command buffer, `protectedNoFault` is not supported, and the `pPictureResource` member of any element of `pBeginInfo->pReferenceSlots` is not `NULL`, then `pPictureResource->imageViewBinding` for that element **must** not specify an image view created from a protected image
- VUID-vkCmdBeginVideoCodingKHR-commandBuffer-07236
  If `commandBuffer` is a protected command buffer `protectedNoFault` is not supported, and the `pPictureResource` member of any element of `pBeginInfo->pReferenceSlots` is not `NULL`, then `pPictureResource->imageViewBinding` for that element **must** specify an image view created from a protected image
- VUID-vkCmdBeginVideoCodingKHR-slotIndex-07239
  If the `slotIndex` member of any element of `pBeginInfo->pReferenceSlots` is not negative, then it **must** specify the index of a DPB slot that is in the active state in `pBeginInfo->videoSession` at the time the command is executed on the device
- VUID-vkCmdBeginVideoCodingKHR-pPictureResource-07265
  Each video picture resource specified by any non-`NULL` `pPictureResource` member specified in the elements of `pBeginInfo->pReferenceSlots` for which `slotIndex` is not negative **must** match one of the video picture resources currently associated with the DPB slot index of `pBeginInfo->videoSession` specified by `slotIndex` at the time the command is executed on the device
- VUID-vkCmdBeginVideoCodingKHR-pBeginInfo-08253
  If `pBeginInfo->videoSession` was created with a video encode operation and the `pNext` chain of `pBeginInfo` does not include an instance of the VkVideoEncodeRateControlInfoKHR structure, then the rate control mode configured for `pBeginInfo->videoSession` at the time the command is executed on the device **must** be `VK_VIDEO_ENCODE_RATE_CONTROL_MODE_DEFAULT_KHR`
- VUID-vkCmdBeginVideoCodingKHR-pBeginInfo-08254

If pBeginInfo->videoSession was created with a video encode operation and the pNext chain of pBeginInfo includes an instance of the VkVideoEncodeRateControlInfoKHR structure, then it **must** match the rate control state configured for pBeginInfo->videoSession at the time the command is executed on the device

- VUID-vkCmdBeginVideoCodingKHR-pBeginInfo-08255
  If pBeginInfo->videoSession was created with the video codec operation VK_VIDEO_CODEC_OPERATION_ENCODE_H264_BIT_KHR, the current rate control mode is not VK_VIDEO_ENCODE_RATE_CONTROL_MODE_DEFAULT_KHR or VK_VIDEO_ENCODE_RATE_CONTROL_MODE_DISABLED_BIT_KHR, and VkVideoEncodeH264CapabilitiesKHR::requiresGopRemainingFrames is VK_TRUE, as returned by vkGetPhysicalDeviceVideoCapabilitiesKHR for the video profile the pBeginInfo->videoSession was created with, then the pNext chain of pBeginInfo **must** include an instance of the VkVideoEncodeH264GopRemainingFrameInfoKHR with its useGopRemainingFrames member set to VK_TRUE

- VUID-vkCmdBeginVideoCodingKHR-pBeginInfo-08256
  If pBeginInfo->videoSession was created with the video codec operation VK_VIDEO_CODEC_OPERATION_ENCODE_H265_BIT_KHR, the current rate control mode is not VK_VIDEO_ENCODE_RATE_CONTROL_MODE_DEFAULT_KHR or VK_VIDEO_ENCODE_RATE_CONTROL_MODE_DISABLED_BIT_KHR, and VkVideoEncodeH265CapabilitiesKHR::requiresGopRemainingFrames is VK_TRUE, as returned by vkGetPhysicalDeviceVideoCapabilitiesKHR for the video profile the pBeginInfo->videoSession was created with, then the pNext chain of pBeginInfo **must** include an instance of the VkVideoEncodeH265GopRemainingFrameInfoKHR with its useGopRemainingFrames member set to VK_TRUE

## Valid Usage (Implicit)

- VUID-vkCmdBeginVideoCodingKHR-commandBuffer-parameter
  commandBuffer **must** be a valid VkCommandBuffer handle
- VUID-vkCmdBeginVideoCodingKHR-pBeginInfo-parameter
  pBeginInfo **must** be a valid pointer to a valid VkVideoBeginCodingInfoKHR structure
- VUID-vkCmdBeginVideoCodingKHR-commandBuffer-recording
  commandBuffer **must** be in the recording state
- VUID-vkCmdBeginVideoCodingKHR-commandBuffer-cmdpool
  The VkCommandPool that commandBuffer was allocated from **must** support decode, or encode operations
- VUID-vkCmdBeginVideoCodingKHR-renderpass
  This command **must** only be called outside of a render pass instance
- VUID-vkCmdBeginVideoCodingKHR-videocoding
  This command **must** only be called outside of a video coding scope
- VUID-vkCmdBeginVideoCodingKHR-bufferlevel
  commandBuffer **must** be a primary VkCommandBuffer

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary | Outside | Outside | Decode<br>Encode | Action<br>State |

The VkVideoBeginCodingInfoKHR structure is defined as:

```
// Provided by VK_KHR_video_queue
typedef struct VkVideoBeginCodingInfoKHR {
    VkStructureType                         sType;
    const void*                             pNext;
    VkVideoBeginCodingFlagsKHR              flags;
    VkVideoSessionKHR                       videoSession;
    VkVideoSessionParametersKHR             videoSessionParameters;
    uint32_t                                referenceSlotCount;
    const VkVideoReferenceSlotInfoKHR*      pReferenceSlots;
} VkVideoBeginCodingInfoKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is reserved for future use.
- `videoSession` is the video session object to be bound for the processing of the video commands.
- `videoSessionParameters` is `VK_NULL_HANDLE` or a handle of a VkVideoSessionParametersKHR object to be used for the processing of the video commands. If `VK_NULL_HANDLE`, then no video session parameters object is bound for the duration of the video coding scope.
- `referenceSlotCount` is the number of elements in the `pReferenceSlots` array.
- `pReferenceSlots` is a pointer to an array of VkVideoReferenceSlotInfoKHR structures specifying the information used to determine the set of bound reference picture resources for the video coding scope and their initial association with DPB slot indices.

Limiting values are defined below that are referenced by the relevant valid usage statements of this structure.

- Let `VkOffset2D codedOffsetGranularity` be the minimum alignment requirement for the coded

offset of video picture resources. Unless otherwise defined, the value of the x and y members of codedOffsetGranularity are 0.

- If videoSession was created with an H.264 decode profile with a VkVideoDecodeH264ProfileInfoKHR::pictureLayout of VK_VIDEO_DECODE_H264_PICTURE_LAYOUT_INTERLACED_SEPARATE_PLANES_BIT_KHR, then codedOffsetGranularity is equal to VkVideoDecodeH264CapabilitiesKHR::fieldOffsetGranularity, as returned by vkGetPhysicalDeviceVideoCapabilitiesKHR for that video profile.

## Valid Usage

- VUID-VkVideoBeginCodingInfoKHR-videoSession-07237
  videoSession **must** have memory bound to all of its memory bindings returned by vkGetVideoSessionMemoryRequirementsKHR for videoSession
- VUID-VkVideoBeginCodingInfoKHR-slotIndex-04856
  Each non-negative VkVideoReferenceSlotInfoKHR::slotIndex specified in the elements of pReferenceSlots **must** be less than the VkVideoSessionCreateInfoKHR::maxDpbSlots specified when videoSession was created
- VUID-VkVideoBeginCodingInfoKHR-pPictureResource-07238
  Each video picture resource corresponding to any non-NULL pPictureResource member specified in the elements of pReferenceSlots **must** be unique within pReferenceSlots
- VUID-VkVideoBeginCodingInfoKHR-pPictureResource-07240
  If the pPictureResource member of any element of pReferenceSlots is not NULL, then the image view specified in pPictureResource->imageViewBinding for that element **must** be compatible with the video profile videoSession was created with
- VUID-VkVideoBeginCodingInfoKHR-pPictureResource-07241
  If the pPictureResource member of any element of pReferenceSlots is not NULL, then the format of the image view specified in pPictureResource->imageViewBinding for that element **must** match the VkVideoSessionCreateInfoKHR::referencePictureFormat videoSession was created with
- VUID-VkVideoBeginCodingInfoKHR-pPictureResource-07242
  If the pPictureResource member of any element of pReferenceSlots is not NULL, then its codedOffset member **must** be an integer multiple of codedOffsetGranularity
- VUID-VkVideoBeginCodingInfoKHR-pPictureResource-07243
  If the pPictureResource member of any element of pReferenceSlots is not NULL, then its codedExtent member **must** be between minCodedExtent and maxCodedExtent, inclusive, videoSession was created with
- VUID-VkVideoBeginCodingInfoKHR-flags-07244
  If VkVideoCapabilitiesKHR::flags does not include VK_VIDEO_CAPABILITY_SEPARATE_REFERENCE_IMAGES_BIT_KHR, as returned by vkGetPhysicalDeviceVideoCapabilitiesKHR for the video profile videoSession was created with, then pPictureResource->imageViewBinding of all elements of pReferenceSlots with a non-NULL pPictureResource member **must** specify image views created from the same image

- VUID-VkVideoBeginCodingInfoKHR-slotIndex-07245
  If `videoSession` was created with a decode operation and the `slotIndex` member of any element of `pReferenceSlots` is not negative, then the image view specified in `pPictureResource->imageViewBinding` for that element **must** have been created with `VK_IMAGE_USAGE_VIDEO_DECODE_DPB_BIT_KHR`
- VUID-VkVideoBeginCodingInfoKHR-slotIndex-07246
  If `videoSession` was created with an encode operation and the `slotIndex` member of any element of `pReferenceSlots` is not negative, then the image view specified in `pPictureResource->imageViewBinding` for that element **must** have been created with `VK_IMAGE_USAGE_VIDEO_ENCODE_DPB_BIT_KHR`
- VUID-VkVideoBeginCodingInfoKHR-videoSession-07247
  If `videoSession` was created with the video codec operation `VK_VIDEO_CODEC_OPERATION_DECODE_H264_BIT_KHR`, then `videoSessionParameters` **must** not be `VK_NULL_HANDLE`
- VUID-VkVideoBeginCodingInfoKHR-videoSession-07248
  If `videoSession` was created with the video codec operation `VK_VIDEO_CODEC_OPERATION_DECODE_H265_BIT_KHR`, then `videoSessionParameters` **must** not be `VK_NULL_HANDLE`
- VUID-VkVideoBeginCodingInfoKHR-videoSession-09261
  If `videoSession` was created with the video codec operation `VK_VIDEO_CODEC_OPERATION_DECODE_AV1_BIT_KHR`, then `videoSessionParameters` **must** not be `VK_NULL_HANDLE`
- VUID-VkVideoBeginCodingInfoKHR-videoSession-07249
  If `videoSession` was created with the video codec operation `VK_VIDEO_CODEC_OPERATION_ENCODE_H264_BIT_KHR`, then `videoSessionParameters` **must** not be `VK_NULL_HANDLE`
- VUID-VkVideoBeginCodingInfoKHR-videoSession-07250
  If `videoSession` was created with the video codec operation `VK_VIDEO_CODEC_OPERATION_ENCODE_H265_BIT_KHR`, then `videoSessionParameters` **must** not be `VK_NULL_HANDLE`
- VUID-VkVideoBeginCodingInfoKHR-videoSessionParameters-04857
  If `videoSessionParameters` is not `VK_NULL_HANDLE`, it **must** have been created with `videoSession` specified in VkVideoSessionParametersCreateInfoKHR::`videoSession`

## Valid Usage (Implicit)

- VUID-VkVideoBeginCodingInfoKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_VIDEO_BEGIN_CODING_INFO_KHR`
- VUID-VkVideoBeginCodingInfoKHR-pNext-pNext
  Each `pNext` member of any structure (including this one) in the `pNext` chain **must** be either `NULL` or a pointer to a valid instance of VkVideoEncodeH264GopRemainingFrameInfoKHR, VkVideoEncodeH264RateControlInfoKHR, VkVideoEncodeH265GopRemainingFrameInfoKHR,

VkVideoEncodeH265RateControlInfoKHR, or VkVideoEncodeRateControlInfoKHR

- VUID-VkVideoBeginCodingInfoKHR-sType-unique
  The `sType` value of each struct in the `pNext` chain **must** be unique
- VUID-VkVideoBeginCodingInfoKHR-flags-zerobitmask
  `flags` **must** be `0`
- VUID-VkVideoBeginCodingInfoKHR-videoSession-parameter
  `videoSession` **must** be a valid VkVideoSessionKHR handle
- VUID-VkVideoBeginCodingInfoKHR-videoSessionParameters-parameter
  If `videoSessionParameters` is not VK_NULL_HANDLE, `videoSessionParameters` **must** be a valid VkVideoSessionParametersKHR handle
- VUID-VkVideoBeginCodingInfoKHR-pReferenceSlots-parameter
  If `referenceSlotCount` is not `0`, `pReferenceSlots` **must** be a valid pointer to an array of `referenceSlotCount` valid VkVideoReferenceSlotInfoKHR structures
- VUID-VkVideoBeginCodingInfoKHR-videoSessionParameters-parent
  If `videoSessionParameters` is a valid handle, it **must** have been created, allocated, or retrieved from `videoSession`
- VUID-VkVideoBeginCodingInfoKHR-commonparent
  Both of `videoSession`, and `videoSessionParameters` that are valid handles of non-ignored parameters **must** have been created, allocated, or retrieved from the same VkDevice

```
// Provided by VK_KHR_video_queue
typedef VkFlags VkVideoBeginCodingFlagsKHR;
```

`VkVideoBeginCodingFlagsKHR` is a bitmask type for setting a mask, but is currently reserved for future use.

The `VkVideoReferenceSlotInfoKHR` structure is defined as:

```
// Provided by VK_KHR_video_queue
typedef struct VkVideoReferenceSlotInfoKHR {
    VkStructureType                          sType;
    const void*                              pNext;
    int32_t                                  slotIndex;
    const VkVideoPictureResourceInfoKHR*     pPictureResource;
} VkVideoReferenceSlotInfoKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `slotIndex` is the index of the DPB slot or a negative integer value.
- `pPictureResource` is `NULL` or a pointer to a VkVideoPictureResourceInfoKHR structure describing the video picture resource associated with the DPB slot index specified by `slotIndex`.

### Valid Usage (Implicit)

- VUID-VkVideoReferenceSlotInfoKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_VIDEO_REFERENCE_SLOT_INFO_KHR`
- VUID-VkVideoReferenceSlotInfoKHR-pNext-pNext
  Each `pNext` member of any structure (including this one) in the `pNext` chain **must** be either `NULL` or a pointer to a valid instance of VkVideoDecodeAV1DpbSlotInfoKHR, VkVideoDecodeH264DpbSlotInfoKHR, VkVideoDecodeH265DpbSlotInfoKHR, VkVideoEncodeH264DpbSlotInfoKHR, or VkVideoEncodeH265DpbSlotInfoKHR
- VUID-VkVideoReferenceSlotInfoKHR-sType-unique
  The `sType` value of each struct in the `pNext` chain **must** be unique
- VUID-VkVideoReferenceSlotInfoKHR-pPictureResource-parameter
  If `pPictureResource` is not `NULL`, `pPictureResource` **must** be a valid pointer to a valid VkVideoPictureResourceInfoKHR structure

To end a video coding scope, call:

```
// Provided by VK_KHR_video_queue
void vkCmdEndVideoCodingKHR(
    VkCommandBuffer                             commandBuffer,
    const VkVideoEndCodingInfoKHR*              pEndCodingInfo);
```

- `commandBuffer` is the command buffer in which to record the command.
- `pEndCodingInfo` is a pointer to a VkVideoEndCodingInfoKHR structure specifying the parameters for ending the video coding scope.

After ending a video coding scope, the video session object, the optional video session parameters object, and all reference picture resources previously bound by the corresponding vkCmdBeginVideoCodingKHR command are *unbound*.

### Valid Usage

- VUID-vkCmdEndVideoCodingKHR-None-07251
  There **must** be no active queries

### Valid Usage (Implicit)

- VUID-vkCmdEndVideoCodingKHR-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdEndVideoCodingKHR-pEndCodingInfo-parameter
  `pEndCodingInfo` **must** be a valid pointer to a valid VkVideoEndCodingInfoKHR structure
- VUID-vkCmdEndVideoCodingKHR-commandBuffer-recording

commandBuffer **must** be in the recording state

- VUID-vkCmdEndVideoCodingKHR-commandBuffer-cmdpool
  The VkCommandPool that commandBuffer was allocated from **must** support decode, or encode operations
- VUID-vkCmdEndVideoCodingKHR-renderpass
  This command **must** only be called outside of a render pass instance
- VUID-vkCmdEndVideoCodingKHR-videocoding
  This command **must** only be called inside of a video coding scope
- VUID-vkCmdEndVideoCodingKHR-bufferlevel
  commandBuffer **must** be a primary VkCommandBuffer

## Host Synchronization

- Host access to commandBuffer **must** be externally synchronized
- Host access to the VkCommandPool that commandBuffer was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary | Outside | Inside | Decode<br>Encode | Action<br>State |

The VkVideoEndCodingInfoKHR structure is defined as:

```
// Provided by VK_KHR_video_queue
typedef struct VkVideoEndCodingInfoKHR {
    VkStructureType             sType;
    const void*                 pNext;
    VkVideoEndCodingFlagsKHR    flags;
} VkVideoEndCodingInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- flags is reserved for future use.

## Valid Usage (Implicit)

- VUID-VkVideoEndCodingInfoKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_VIDEO_END_CODING_INFO_KHR

- VUID-VkVideoEndCodingInfoKHR-pNext-pNext
  `pNext` **must** be `NULL`
- VUID-VkVideoEndCodingInfoKHR-flags-zerobitmask
  `flags` **must** be `0`

```
// Provided by VK_KHR_video_queue
typedef VkFlags VkVideoEndCodingFlagsKHR;
```

`VkVideoEndCodingFlagsKHR` is a bitmask type for setting a mask, but is currently reserved for future use.

## 42.9. Video Coding Control

To apply dynamic controls to the currently bound video session object, call:

```
// Provided by VK_KHR_video_queue
void vkCmdControlVideoCodingKHR(
    VkCommandBuffer                             commandBuffer,
    const VkVideoCodingControlInfoKHR*          pCodingControlInfo);
```

- `commandBuffer` is the command buffer in which to record the command.
- `pCodingControlInfo` is a pointer to a VkVideoCodingControlInfoKHR structure specifying the control parameters.

The control parameters provided in this call are applied to the video session at the time the command executes on the device and are in effect until a subsequent call to this command with the same video session bound changes the corresponding control parameters.

A newly created video session **must** be reset before performing video coding operations using it by including `VK_VIDEO_CODING_CONTROL_RESET_BIT_KHR` in `pCodingControlInfo->flags`. The reset operation also returns all DPB slots of the video session to the inactive state. Correspondingly, any DPB slot index associated with the bound reference picture resources is removed.

For encode sessions, the reset operation returns rate control configuration to implementation default settings and sets the video encode quality level to zero.

After video coding operations are performed using a video session, the reset operation **can** be used to return the video session to the same *initial* state as after the reset of a newly created video session. This **can** be used, for example, when different video sequences are needed to be processed with the same video session object.

If `pCodingControlInfo->flags` includes `VK_VIDEO_CODING_CONTROL_ENCODE_RATE_CONTROL_BIT_KHR`, then the command replaces the rate control configuration maintained by the video session with the configuration specified in the VkVideoEncodeRateControlInfoKHR structure included in the `pCodingControlInfo->pNext` chain.

If `pCodingControlInfo->flags` includes `VK_VIDEO_CODING_CONTROL_ENCODE_QUALITY_LEVEL_BIT_KHR`, then the command changes the current video encode quality level to the value specified in the `qualityLevel` member of the VkVideoEncodeQualityLevelInfoKHR structure included in the `pCodingControlInfo->pNext` chain.

## Valid Usage

- VUID-vkCmdControlVideoCodingKHR-flags-07017
  If `pCodingControlInfo->flags` does not include `VK_VIDEO_CODING_CONTROL_RESET_BIT_KHR`, then the bound video session **must** not be in uninitialized state at the time the command is executed on the device
- VUID-vkCmdControlVideoCodingKHR-pCodingControlInfo-08243
  If the bound video session was not created with an encode operation, then `pCodingControlInfo->flags` **must** not include `VK_VIDEO_CODING_CONTROL_ENCODE_RATE_CONTROL_BIT_KHR` or `VK_VIDEO_CODING_CONTROL_ENCODE_QUALITY_LEVEL_BIT_KHR`

## Valid Usage (Implicit)

- VUID-vkCmdControlVideoCodingKHR-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdControlVideoCodingKHR-pCodingControlInfo-parameter
  `pCodingControlInfo` **must** be a valid pointer to a valid VkVideoCodingControlInfoKHR structure
- VUID-vkCmdControlVideoCodingKHR-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdControlVideoCodingKHR-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support decode, or encode operations
- VUID-vkCmdControlVideoCodingKHR-renderpass
  This command **must** only be called outside of a render pass instance
- VUID-vkCmdControlVideoCodingKHR-videocoding
  This command **must** only be called inside of a video coding scope
- VUID-vkCmdControlVideoCodingKHR-bufferlevel
  `commandBuffer` **must** be a primary `VkCommandBuffer`

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary | Outside | Inside | Decode<br>Encode | Action |

The VkVideoCodingControlInfoKHR structure is defined as:

```
// Provided by VK_KHR_video_queue
typedef struct VkVideoCodingControlInfoKHR {
    VkStructureType                 sType;
    const void*                     pNext;
    VkVideoCodingControlFlagsKHR    flags;
} VkVideoCodingControlInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- flags is a bitmask of VkVideoCodingControlFlagsKHR specifying control flags.

## Valid Usage

- VUID-VkVideoCodingControlInfoKHR-flags-07018
  If flags includes VK_VIDEO_CODING_CONTROL_ENCODE_RATE_CONTROL_BIT_KHR, then the pNext chain **must** include a VkVideoEncodeRateControlInfoKHR structure
- VUID-VkVideoCodingControlInfoKHR-flags-08349
  If flags includes VK_VIDEO_CODING_CONTROL_ENCODE_QUALITY_LEVEL_BIT_KHR, then the pNext chain **must** include a VkVideoEncodeQualityLevelInfoKHR structure

## Valid Usage (Implicit)

- VUID-VkVideoCodingControlInfoKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_VIDEO_CODING_CONTROL_INFO_KHR
- VUID-VkVideoCodingControlInfoKHR-pNext-pNext
  Each pNext member of any structure (including this one) in the pNext chain **must** be either NULL or a pointer to a valid instance of VkVideoEncodeH264RateControlInfoKHR, VkVideoEncodeH265RateControlInfoKHR, VkVideoEncodeQualityLevelInfoKHR, or VkVideoEncodeRateControlInfoKHR
- VUID-VkVideoCodingControlInfoKHR-sType-unique
  The sType value of each struct in the pNext chain **must** be unique
- VUID-VkVideoCodingControlInfoKHR-flags-parameter
  flags **must** be a valid combination of VkVideoCodingControlFlagBitsKHR values

- VUID-VkVideoCodingControlInfoKHR-flags-requiredbitmask
  `flags` **must** not be `0`

Bits which **can** be set in VkVideoCodingControlInfoKHR::`flags`, specifying the video coding control parameters to be modified, are:

```
// Provided by VK_KHR_video_queue
typedef enum VkVideoCodingControlFlagBitsKHR {
    VK_VIDEO_CODING_CONTROL_RESET_BIT_KHR = 0x00000001,
  // Provided by VK_KHR_video_encode_queue
    VK_VIDEO_CODING_CONTROL_ENCODE_RATE_CONTROL_BIT_KHR = 0x00000002,
  // Provided by VK_KHR_video_encode_queue
    VK_VIDEO_CODING_CONTROL_ENCODE_QUALITY_LEVEL_BIT_KHR = 0x00000004,
} VkVideoCodingControlFlagBitsKHR;
```

- `VK_VIDEO_CODING_CONTROL_RESET_BIT_KHR` indicates a request for the bound video session to be reset before other coding control parameters are applied.
- `VK_VIDEO_CODING_CONTROL_ENCODE_RATE_CONTROL_BIT_KHR` indicates that the coding control parameters include video encode rate control parameters (see VkVideoEncodeRateControlInfoKHR).
- `VK_VIDEO_CODING_CONTROL_ENCODE_QUALITY_LEVEL_BIT_KHR` indicates that the coding control parameters include video encode quality level parameters (see VkVideoEncodeQualityLevelInfoKHR).

```
// Provided by VK_KHR_video_queue
typedef VkFlags VkVideoCodingControlFlagsKHR;
```

`VkVideoCodingControlFlagsKHR` is a bitmask type for setting a mask of zero or more VkVideoCodingControlFlagBitsKHR.

## 42.10. Inline Queries

If a video session was created with `VK_VIDEO_SESSION_CREATE_INLINE_QUERIES_BIT_KHR`, beginning queries using commands such as vkCmdBeginQuery within a video coding scope is not allowed. Instead, queries are executed inline by including an instance of the VkVideoInlineQueryInfoKHR structure in the `pNext` chain of the parameters of one of the video coding commands, with its `queryPool` member set to a valid `VkQueryPool` handle.

The `VkVideoInlineQueryInfoKHR` structure is defined as:

```
// Provided by VK_KHR_video_maintenance1
typedef struct VkVideoInlineQueryInfoKHR {
    VkStructureType    sType;
    const void*        pNext;
    VkQueryPool        queryPool;
    uint32_t           firstQuery;
    uint32_t           queryCount;
} VkVideoInlineQueryInfoKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `queryPool` is `VK_NULL_HANDLE` or a valid handle to a VkQueryPool object that will manage the results of the queries.
- `firstQuery` is the query index within the query pool that will contain the query results for the first video coding operation. The query results of subsequent video coding operations will be contained by subsequent query indices.
- `queryCount` is the number of queries to execute.

**NOTE** In practice, if `queryPool` is not `VK_NULL_HANDLE`, then `queryCount` will always have to match the number of video coding operations issued by the video coding command this structure is specified to, meaning that using inline queries in a video coding command will always execute a query for each issued video coding operation.

This structure **can** be included in the `pNext` chain of the input parameter structure of video coding commands.

- In the `pNext` chain of the `pDecodeInfo` parameter of the vkCmdDecodeVideoKHR command to execute a query for each video decode operation issued by the command.
- In the `pNext` chain of the `pEncodeInfo` parameter of the vkCmdEncodeVideoKHR command to execute a query for each video encode operation issued by the command.

## Valid Usage

- VUID-VkVideoInlineQueryInfoKHR-queryPool-08372
  If `queryPool` is not `VK_NULL_HANDLE`, then `firstQuery` **must** be less than the number of queries in `queryPool`
- VUID-VkVideoInlineQueryInfoKHR-queryPool-08373
  If `queryPool` is not `VK_NULL_HANDLE`, then the sum of `firstQuery` and `queryCount` **must** be less than or equal to the number of queries in `queryPool`

## Valid Usage (Implicit)

- VUID-VkVideoInlineQueryInfoKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_VIDEO_INLINE_QUERY_INFO_KHR`
- VUID-VkVideoInlineQueryInfoKHR-queryPool-parameter
  If `queryPool` is not VK_NULL_HANDLE, `queryPool` **must** be a valid VkQueryPool handle

# 42.11. Video Decode Operations

Video decode operations consume compressed video data from a video bitstream buffer and zero or more reference pictures, and produce a *decode output picture* and an optional reconstructed picture.

| | |
|---|---|
| **NOTE** | Such decode output pictures can be shared with the Decoded Picture Buffer, and can also be used as the input of video encode operations, with graphics or compute operations, or with Window System Integration APIs, depending on the capabilities of the implementation. |

Video decode operations **may** access the following resources in the `VK_PIPELINE_STAGE_2_VIDEO_DECODE_BIT_KHR` stage:

- The source video bitstream buffer range and the image subregions corresponding to the list of active reference pictures with access `VK_ACCESS_2_VIDEO_DECODE_READ_BIT_KHR`.
- The image subregions corresponding to the target decode output picture and reconstructed picture with access `VK_ACCESS_2_VIDEO_DECODE_WRITE_BIT_KHR`.

The image subresource of each video picture resource accessed by the video decode operation is specified using a corresponding VkVideoPictureResourceInfoKHR structure. Each such image subresource **must** be in the appropriate image layout as follows:

- If the image subresource is used in the video decode operation only as decode output picture, then it **must** be in the `VK_IMAGE_LAYOUT_VIDEO_DECODE_DST_KHR` layout.
- If the image subresource is used in the video decode operation both as decode output picture and reconstructed picture, then it **must** be in the `VK_IMAGE_LAYOUT_VIDEO_DECODE_DPB_KHR` layout.
- If the image subresource is used in the video decode operation only as reconstructed picture, then it **must** be in the `VK_IMAGE_LAYOUT_VIDEO_DECODE_DPB_KHR` layout.
- If the image subresource is used in the video decode operation as a reference picture, then it **must** be in the `VK_IMAGE_LAYOUT_VIDEO_DECODE_DPB_KHR` layout.

A video decode operation **may** complete unsuccessfully. In this case the decode output picture will have undefined contents. Similarly, if reference picture setup is requested, the reconstructed picture will also have undefined contents, and the activated DPB slot will have an invalid picture reference.

## 42.11.1. Codec-Specific Semantics

The following aspects of video decode operations are codec-specific:

- The interpretation of the contents of the source video bitstream buffer range.
- The construction and interpretation of the list of active reference pictures and the interpretation of the picture data referred to by the corresponding image subregions.
- The construction and interpretation of information related to the decode output picture and the generation of picture data to the corresponding image subregion.
- The decision on reference picture setup.
- The construction and interpretation of information related to the optional reconstructed picture and the generation of picture data to the corresponding image subregion.

These codec-specific behaviors are defined for each video codec operation separately.

- If the used video codec operation is `VK_VIDEO_CODEC_OPERATION_DECODE_H264_BIT_KHR`, then the codec-specific aspects of the video decoding process are performed as defined in the H.264 Decode Operations section.
- If the used video codec operation is `VK_VIDEO_CODEC_OPERATION_DECODE_H265_BIT_KHR`, then the codec-specific aspects of the video decoding process are performed as defined in the H.265 Decode Operations section.
- If the used video codec operation is `VK_VIDEO_CODEC_OPERATION_DECODE_AV1_BIT_KHR`, then the codec-specific aspects of the video decoding process are performed as defined in the AV1 Decode Operations section.

### 42.11.2. Video Decode Operation Steps

Each video decode operation performs the following steps in the `VK_PIPELINE_STAGE_2_VIDEO_DECODE_BIT_KHR` stage:

1. Reads the encoded video data from the source video bitstream buffer range.
2. Performs picture reconstruction of the encoded video data according to the codec-specific semantics, applying any prediction data read from the active reference pictures in the process;
3. Writes the decoded picture data to the decode output picture, and optionally to the reconstructed picture, if one is specified and is different from the decode output picture, according to the codec-specific semantics;
4. If reference picture setup is requested, the DPB slot index specified in the reconstructed picture information is activated with the reconstructed picture.

When reconstructed picture information is provided, the specified DPB slot index is associated with the corresponding bound reference picture resource, indifferent of whether reference picture setup is requested.

### 42.11.3. Capabilities

When calling vkGetPhysicalDeviceVideoCapabilitiesKHR with `pVideoProfile->videoCodecOperation` specifying a decode operation, the `VkVideoDecodeCapabilitiesKHR` structure **must** be included in the `pNext` chain of the VkVideoCapabilitiesKHR structure to retrieve capabilities specific to video decoding.

The VkVideoDecodeCapabilitiesKHR structure is defined as:

```
// Provided by VK_KHR_video_decode_queue
typedef struct VkVideoDecodeCapabilitiesKHR {
    VkStructureType                     sType;
    void*                               pNext;
    VkVideoDecodeCapabilityFlagsKHR     flags;
} VkVideoDecodeCapabilitiesKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- flags is a bitmask of VkVideoDecodeCapabilityFlagBitsKHR describing the supported video decoding capabilities.

### Valid Usage (Implicit)

- VUID-VkVideoDecodeCapabilitiesKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_VIDEO_DECODE_CAPABILITIES_KHR

Bits which **may** be set in VkVideoDecodeCapabilitiesKHR::flags, indicating the decoding capabilities supported, are:

```
// Provided by VK_KHR_video_decode_queue
typedef enum VkVideoDecodeCapabilityFlagBitsKHR {
    VK_VIDEO_DECODE_CAPABILITY_DPB_AND_OUTPUT_COINCIDE_BIT_KHR = 0x00000001,
    VK_VIDEO_DECODE_CAPABILITY_DPB_AND_OUTPUT_DISTINCT_BIT_KHR = 0x00000002,
} VkVideoDecodeCapabilityFlagBitsKHR;
```

- VK_VIDEO_DECODE_CAPABILITY_DPB_AND_OUTPUT_COINCIDE_BIT_KHR indicates support for using the same video picture resource as the reconstructed picture and decode output picture in a video decode operation.
- VK_VIDEO_DECODE_CAPABILITY_DPB_AND_OUTPUT_DISTINCT_BIT_KHR indicates support for using distinct video picture resources as the reconstructed picture and decode output picture in a video decode operation.

**NOTE** Some video profiles allow using distinct video picture resources as the reconstructed picture and decode output picture in specific video decode operations even when the video decode profile does not support VK_VIDEO_DECODE_CAPABILITY_DPB_AND_OUTPUT_DISTINCT_BIT_KHR. Even if the implementation only reports coincide, the decode output picture for film grain enabled frames must be a different video picture resource from the reconstructed picture because film grain is applied outside of the coding loop.

Implementations are only **required** to support one of

VK_VIDEO_DECODE_CAPABILITY_DPB_AND_OUTPUT_COINCIDE_BIT_KHR and VK_VIDEO_DECODE_CAPABILITY_DPB_AND_OUTPUT_DISTINCT_BIT_KHR. Accordingly, applications **should** handle both cases to maximize portability.

**NOTE** If both VK_VIDEO_DECODE_CAPABILITY_DPB_AND_OUTPUT_COINCIDE_BIT_KHR and VK_VIDEO_DECODE_CAPABILITY_DPB_AND_OUTPUT_DISTINCT_BIT_KHR are supported, an application can choose to create separate images for decode DPB and decode output. E.g. in cases when linear tiling is preferred (and supported) for the decode output picture and the DPB requires optimal tiling, this avoids the need for a separate copy at the expense of additional memory bandwidth requirements during decoding.

```
// Provided by VK_KHR_video_decode_queue
typedef VkFlags VkVideoDecodeCapabilityFlagsKHR;
```

VkVideoDecodeCapabilityFlagsKHR is a bitmask type for setting a mask of zero or more VkVideoDecodeCapabilityFlagBitsKHR.

### 42.11.4. Video Decode Commands

To launch video decode operations, call:

```
// Provided by VK_KHR_video_decode_queue
void vkCmdDecodeVideoKHR(
    VkCommandBuffer                             commandBuffer,
    const VkVideoDecodeInfoKHR*                 pDecodeInfo);
```

- commandBuffer is the command buffer in which to record the command.
- pDecodeInfo is a pointer to a VkVideoDecodeInfoKHR structure specifying the parameters of the video decode operations.

Each call issues one or more video decode operations. The implicit parameter opCount corresponds to the number of video decode operations issued by the command. After calling this command, the active query index of each active query is incremented by opCount.

Currently each call to this command results in the issue of a single video decode operation.

If the bound video session was created with VK_VIDEO_SESSION_CREATE_INLINE_QUERIES_BIT_KHR and the pNext chain of pDecodeInfo includes a VkVideoInlineQueryInfoKHR structure with its queryPool member specifying a valid VkQueryPool handle, then this command will execute a query for each video decode operation issued by it.

**Active Reference Picture Information**

The list of active reference pictures used by a video decode operation is a list of image subregions used as the source of reference picture data and related parameters, and is derived from the VkVideoReferenceSlotInfoKHR structures provided as the elements of the pDecodeInfo-

>pReferenceSlots array. For each element of pDecodeInfo->pReferenceSlots, one or more elements are added to the active reference picture list, as defined by the codec-specific semantics. Each element of this list contains the following information:

- The image subregion within the image subresource referred to by the video picture resource used as the reference picture.
- The DPB slot index the reference picture is associated with.
- The codec-specific reference information related to the reference picture.

**Reconstructed Picture Information**

Information related to the optional reconstructed picture used by a video decode operation is derived from the VkVideoReferenceSlotInfoKHR structure pointed to by pDecodeInfo->pSetupReferenceSlot, if not NULL, as defined by the codec-specific semantics, and consists of the following:

- The image subregion within the image subresource referred to by the video picture resource used as the reconstructed picture.
- The DPB slot index to use for picture reconstruction.
- The codec-specific reference information related to the reconstructed picture.

Specifying a valid VkVideoReferenceSlotInfoKHR structure in pDecodeInfo->pSetupReferenceSlot is always required, unless the video session was created with VkVideoSessionCreateInfoKHR::maxDpbSlot equal to zero. However, the DPB slot identified by pDecodeInfo->pSetupReferenceSlot->slotIndex is only activated with the reconstructed picture specified in pDecodeInfo->pSetupReferenceSlot->pPictureResource if reference picture setup is requested according to the codec-specific semantics.

If reconstructed picture information is specified, and pDecodeInfo->pSetupReferenceSlot->pPictureResource refers to a video picture resource different than that of the decode output picture, but reference picture setup is not requested, the contents of the video picture resource corresponding to the reconstructed picture will be undefined after the video decode operation.

| | |
|---|---|
| **NOTE** | Some implementations may always output the reconstructed picture or use it as temporary storage during the video decode operation even when the reconstructed picture is not marked for future reference. |

**Decode Output Picture Information**

Information related to the decode output picture used by a video decode operation is derived from pDecodeInfo->dstPictureResource and any codec-specific parameters provided in the pDecodeInfo->pNext chain, as defined by the codec-specific semantics, and consists of the following:

- The image subregion within the image subresource referred to by the video picture resource used as the decode output picture.
- The codec-specific picture information related to the decode output picture.

Several limiting values are defined below that are referenced by the relevant valid usage

statements of this command.

- Let `uint32_t activeReferencePictureCount` be the size of the list of active reference pictures used by the video decode operation. Unless otherwise defined, `activeReferencePictureCount` is set to the value of `pDecodeInfo->referenceSlotCount`.
  - If the bound video session was created with an H.264 decode profile, then let `activeReferencePictureCount` be the value of `pDecodeInfo->referenceSlotCount` plus the number of elements of the `pDecodeInfo->pReferenceSlots` array that have a VkVideoDecodeH264DpbSlotInfoKHR structure included in their `pNext` chain with both `pStdReferenceInfo->flags.top_field_flag` and `pStdReferenceInfo->flags.bottom_field_flag` set.

| | |
|---|---|
| **NOTE** | This means that the elements of `pDecodeInfo->pReferenceSlots` that include both a top and bottom field reference are counted as two separate active reference pictures, as described in the active reference picture list construction rules for H.264 decode operations. |

- Let `VkOffset2D codedOffsetGranularity` be the minimum alignment requirement for the coded offset of video picture resources. Unless otherwise defined, the value of the `x` and `y` members of `codedOffsetGranularity` are `0`.
  - If the bound video session was created with an H.264 decode profile with a VkVideoDecodeH264ProfileInfoKHR::`pictureLayout` of `VK_VIDEO_DECODE_H264_PICTURE_LAYOUT_INTERLACED_SEPARATE_PLANES_BIT_KHR`, then `codedOffsetGranularity` is equal to VkVideoDecodeH264CapabilitiesKHR::`fieldOffsetGranularity`, as returned by vkGetPhysicalDeviceVideoCapabilitiesKHR for that video profile.
- Let `uint32_t dpbFrameUseCount[]` be an array of size `maxDpbSlots`, where `maxDpbSlots` is the VkVideoSessionCreateInfoKHR::`maxDpbSlots` the bound video session was created with, with each element indicating the number of times a frame associated with the corresponding DPB slot index is referred to by the video coding operation. Let the initial value of each element of the array be `0`.
  - If `pDecodeInfo->pSetupReferenceSlot` is not `NULL`, then `dpbFrameUseCount[i]` is incremented by one, where `i` equals `pDecodeInfo->pSetupReferenceSlot->slotIndex`. If the bound video session object was created with an H.264 decode profile, then `dpbFrameUseCount[i]` is decremented by one if either `pStdReferenceInfo->flags.top_field_flag` or `pStdReferenceInfo->flags.bottom_field_flag` is set in the VkVideoDecodeH264DpbSlotInfoKHR structure in the `pDecodeInfo->pSetupReferenceSlot->pNext` chain.
  - For each element of `pDecodeInfo->pReferenceSlots`, `dpbFrameUseCount[i]` is incremented by one, where `i` equals the `slotIndex` member of the corresponding element. If the bound video session object was created with an H.264 decode profile, then `dpbFrameUseCount[i]` is decremented by one if either `pStdReferenceInfo->flags.top_field_flag` or `pStdReferenceInfo->flags.bottom_field_flag` is set in the VkVideoDecodeH264DpbSlotInfoKHR structure in the `pNext` chain of the corresponding element of `pDecodeInfo->pReferenceSlots`.

- Let `uint32_t dpbTopFieldUseCount[]` and `uint32_t dpbBottomFieldUseCount[]` be arrays of size `maxDpbSlots`, where `maxDpbSlots` is the VkVideoSessionCreateInfoKHR::`maxDpbSlots` the bound video session was created with, with each element indicating the number of times the top field or the bottom field, respectively, associated with the corresponding DPB slot index is referred to by the video coding operation. Let the initial value of each element of the arrays be `0`.
  - If the bound video session object was created with an H.264 decode profile and `pDecodeInfo->pSetupReferenceSlot` is not `NULL`, then perform the following:
    - If `pStdReferenceInfo->flags.top_field_flag` is set in the VkVideoDecodeH264DpbSlotInfoKHR structure in the `pDecodeInfo->pSetupReferenceSlot->pNext` chain, then `dpbTopFieldUseCount[i]` is incremented by one, where `i` equals `pDecodeInfo->pSetupReferenceSlot->slotIndex`.
    - If `pStdReferenceInfo->flags.bottom_field_flag` is set in the VkVideoDecodeH264DpbSlotInfoKHR structure in the `pDecodeInfo->pSetupReferenceSlot->pNext` chain, then `dpbBottomFieldUseCount[i]` is incremented by one, where `i` equals `pDecodeInfo->pSetupReferenceSlot->slotIndex`.
  - If the bound video session object was created with an H.264 decode profile, then perform the following for each element of `pDecodeInfo->pReferenceSlots`:
    - If `pStdReferenceInfo->flags.top_field_flag` is set in the VkVideoDecodeH264DpbSlotInfoKHR structure in the `pNext` chain of the element, then `dpbTopFieldUseCount[i]` is incremented by one, where `i` equals the `slotIndex` member of the element.
    - If `pStdReferenceInfo->flags.bottom_field_flag` is set in the VkVideoDecodeH264DpbSlotInfoKHR structure in the `pNext` chain of the element, then `dpbBottomFieldUseCount[i]` is incremented by one, where `i` equals the `slotIndex` member of the element.

## Valid Usage

- VUID-vkCmdDecodeVideoKHR-None-08249
  The bound video session **must** have been created with a decode operation
- VUID-vkCmdDecodeVideoKHR-None-07011
  The bound video session **must** not be in uninitialized state at the time the command is executed on the device
- VUID-vkCmdDecodeVideoKHR-opCount-07134
  For each active query, the active query index corresponding to the query type of that query plus `opCount` **must** be less than or equal to the last activatable query index corresponding to the query type of that query plus one
- VUID-vkCmdDecodeVideoKHR-pNext-08365
  If the bound video session was created with `VK_VIDEO_SESSION_CREATE_INLINE_QUERIES_BIT_KHR`, and the `pNext` chain of `pDecodeInfo` includes a VkVideoInlineQueryInfoKHR structure with its `queryPool` member specifying a valid `VkQueryPool` handle, then VkVideoInlineQueryInfoKHR::queryCount **must** equal `opCount`

- VUID-vkCmdDecodeVideoKHR-pNext-08366
  If the bound video session was created with `VK_VIDEO_SESSION_CREATE_INLINE_QUERIES_BIT_KHR`, and the `pNext` chain of `pDecodeInfo` includes a VkVideoInlineQueryInfoKHR structure with its `queryPool` member specifying a valid `VkQueryPool` handle, then all the queries used by the command, as specified by the VkVideoInlineQueryInfoKHR structure, **must** be *unavailable*
- VUID-vkCmdDecodeVideoKHR-queryType-08367
  If the bound video session was created with `VK_VIDEO_SESSION_CREATE_INLINE_QUERIES_BIT_KHR`, then the `queryType` used to create the `queryPool` specified in the VkVideoInlineQueryInfoKHR structure included in the `pNext` chain of `pDecodeInfo` **must** be `VK_QUERY_TYPE_RESULT_STATUS_ONLY_KHR`
- VUID-vkCmdDecodeVideoKHR-queryPool-08368
  If the bound video session was created with `VK_VIDEO_SESSION_CREATE_INLINE_QUERIES_BIT_KHR`, then the `queryPool` specified in the VkVideoInlineQueryInfoKHR structure included in the `pNext` chain of `pDecodeInfo` **must** have been created with a VkVideoProfileInfoKHR structure included in the `pNext` chain of VkQueryPoolCreateInfo identical to the one specified in VkVideoSessionCreateInfoKHR::`pVideoProfile` the bound video session was created with
- VUID-vkCmdDecodeVideoKHR-queryType-08369
  If the bound video session was created with `VK_VIDEO_SESSION_CREATE_INLINE_QUERIES_BIT_KHR`, and the `queryType` used to create the `queryPool` specified in the VkVideoInlineQueryInfoKHR structure included in the `pNext` chain of `pDecodeInfo` is `VK_QUERY_TYPE_RESULT_STATUS_ONLY_KHR`, then the `VkCommandPool` that `commandBuffer` was allocated from **must** have been created with a queue family index that supports result status queries, as indicated by VkQueueFamilyQueryResultStatusPropertiesKHR::`queryResultStatusSupport`
- VUID-vkCmdDecodeVideoKHR-pDecodeInfo-07135
  `pDecodeInfo->srcBuffer` **must** be compatible with the video profile the bound video session was created with
- VUID-vkCmdDecodeVideoKHR-commandBuffer-07136
  If `commandBuffer` is an unprotected command buffer and `protectedNoFault` is not supported, then `pDecodeInfo->srcBuffer` **must** not be a protected buffer
- VUID-vkCmdDecodeVideoKHR-commandBuffer-07137
  If `commandBuffer` is a protected command buffer and `protectedNoFault` is not supported, then `pDecodeInfo->srcBuffer` **must** be a protected buffer
- VUID-vkCmdDecodeVideoKHR-pDecodeInfo-07138
  `pDecodeInfo->srcBufferOffset` **must** be an integer multiple of VkVideoCapabilitiesKHR::`minBitstreamBufferOffsetAlignment`, as returned by vkGetPhysicalDeviceVideoCapabilitiesKHR for the video profile the bound video session was created with
- VUID-vkCmdDecodeVideoKHR-pDecodeInfo-07139
  `pDecodeInfo->srcBufferRange` **must** be an integer multiple of VkVideoCapabilitiesKHR::`minBitstreamBufferSizeAlignment`, as returned by vkGetPhysicalDeviceVideoCapabilitiesKHR for the video profile the bound video session was created with

- VUID-vkCmdDecodeVideoKHR-pDecodeInfo-07140
  If `pDecodeInfo->pSetupReferenceSlot` is not `NULL` and VkVideoDecodeCapabilitiesKHR::`flags` does not include `VK_VIDEO_DECODE_CAPABILITY_DPB_AND_OUTPUT_COINCIDE_BIT_KHR`, as returned by vkGetPhysicalDeviceVideoCapabilitiesKHR for the video profile the bound video session was created with, then the video picture resources specified by `pDecodeInfo->dstPictureResource` and `pDecodeInfo->pSetupReferenceSlot->pPictureResource` **must** not match
- VUID-vkCmdDecodeVideoKHR-pDecodeInfo-07141
  If `pDecodeInfo->pSetupReferenceSlot` is not `NULL` and none of the following is true:
  - VkVideoDecodeCapabilitiesKHR::`flags` includes `VK_VIDEO_DECODE_CAPABILITY_DPB_AND_OUTPUT_DISTINCT_BIT_KHR`, as returned by vkGetPhysicalDeviceVideoCapabilitiesKHR for the video profile the bound video session was created with
  - the bound video session was created with the video codec operation `VK_VIDEO_CODEC_OPERATION_DECODE_AV1_BIT_KHR` and VkVideoDecodeAV1ProfileInfoKHR::`filmGrainSupport` set to `VK_TRUE`, and film grain is enabled for the decoded picture

  then the video picture resources specified by `pDecodeInfo->dstPictureResource` and `pDecodeInfo->pSetupReferenceSlot->pPictureResource` **must** match
- VUID-vkCmdDecodeVideoKHR-pDecodeInfo-07142
  `pDecodeInfo->dstPictureResource.imageViewBinding` **must** be compatible with the video profile the bound video session was created with
- VUID-vkCmdDecodeVideoKHR-pDecodeInfo-07143
  The format of `pDecodeInfo->dstPictureResource.imageViewBinding` **must** match the VkVideoSessionCreateInfoKHR::`pictureFormat` the bound video session was created with
- VUID-vkCmdDecodeVideoKHR-pDecodeInfo-07144
  `pDecodeInfo->dstPictureResource.codedOffset` **must** be an integer multiple of `codedOffsetGranularity`
- VUID-vkCmdDecodeVideoKHR-pDecodeInfo-07145
  `pDecodeInfo->dstPictureResource.codedExtent` **must** be between `minCodedExtent` and `maxCodedExtent`, inclusive, the bound video session was created with
- VUID-vkCmdDecodeVideoKHR-pDecodeInfo-07146
  `pDecodeInfo->dstPictureResource.imageViewBinding` **must** have been created with `VK_IMAGE_USAGE_VIDEO_DECODE_DST_BIT_KHR`
- VUID-vkCmdDecodeVideoKHR-commandBuffer-07147
  If `commandBuffer` is an unprotected command buffer and `protectedNoFault` is not supported, then `pDecodeInfo->dstPictureResource.imageViewBinding` **must** not have been created from a protected image
- VUID-vkCmdDecodeVideoKHR-commandBuffer-07148
  If `commandBuffer` is a protected command buffer and `protectedNoFault` is not supported, then `pDecodeInfo->dstPictureResource.imageViewBinding` **must** have been created from a protected image
- VUID-vkCmdDecodeVideoKHR-pDecodeInfo-08376

pDecodeInfo->pSetupReferenceSlot **must** not be NULL unless the bound video session was created with VkVideoSessionCreateInfoKHR::maxDpbSlots equal to zero

- VUID-vkCmdDecodeVideoKHR-pDecodeInfo-07170
  If pDecodeInfo->pSetupReferenceSlot is not NULL, then pDecodeInfo->pSetupReferenceSlot->slotIndex **must** be less than the VkVideoSessionCreateInfoKHR::maxDpbSlots specified when the bound video session was created
- VUID-vkCmdDecodeVideoKHR-pDecodeInfo-07173
  If pDecodeInfo->pSetupReferenceSlot is not NULL, then pDecodeInfo->pSetupReferenceSlot->pPictureResource->codedOffset **must** be an integer multiple of codedOffsetGranularity
- VUID-vkCmdDecodeVideoKHR-pDecodeInfo-07149
  If pDecodeInfo->pSetupReferenceSlot is not NULL, then pDecodeInfo->pSetupReferenceSlot->pPictureResource **must** match one of the bound reference picture resource
- VUID-vkCmdDecodeVideoKHR-activeReferencePictureCount-07150
  activeReferencePictureCount **must** be less than or equal to the VkVideoSessionCreateInfoKHR::maxActiveReferencePictures specified when the bound video session was created
- VUID-vkCmdDecodeVideoKHR-slotIndex-07256
  The slotIndex member of each element of pDecodeInfo->pReferenceSlots **must** be less than the VkVideoSessionCreateInfoKHR::maxDpbSlots specified when the bound video session was created
- VUID-vkCmdDecodeVideoKHR-codedOffset-07257
  The codedOffset member of the VkVideoPictureResourceInfoKHR structure pointed to by the pPictureResource member of each element of pDecodeInfo->pReferenceSlots **must** be an integer multiple of codedOffsetGranularity
- VUID-vkCmdDecodeVideoKHR-pDecodeInfo-07151
  The pPictureResource member of each element of pDecodeInfo->pReferenceSlots **must** match one of the bound reference picture resource associated with the DPB slot index specified in the slotIndex member of that element
- VUID-vkCmdDecodeVideoKHR-pDecodeInfo-07264
  Each video picture resource corresponding to the pPictureResource member specified in the elements of pDecodeInfo->pReferenceSlots **must** be unique within pDecodeInfo->pReferenceSlots
- VUID-vkCmdDecodeVideoKHR-dpbFrameUseCount-07176
  All elements of dpbFrameUseCount **must** be less than or equal to 1
- VUID-vkCmdDecodeVideoKHR-dpbTopFieldUseCount-07177
  All elements of dpbTopFieldUseCount **must** be less than or equal to 1
- VUID-vkCmdDecodeVideoKHR-dpbBottomFieldUseCount-07178
  All elements of dpbBottomFieldUseCount **must** be less than or equal to 1
- VUID-vkCmdDecodeVideoKHR-pDecodeInfo-07252
  If pDecodeInfo->pSetupReferenceSlot is NULL or pDecodeInfo->pSetupReferenceSlot->pPictureResource does not refer to the same image subresource as pDecodeInfo->dstPictureResource, then the image subresource referred to by pDecodeInfo->dstPictureResource **must** be in the VK_IMAGE_LAYOUT_VIDEO_DECODE_DST_KHR layout at the

time the video decode operation is executed on the device

- VUID-vkCmdDecodeVideoKHR-pDecodeInfo-07253
  If `pDecodeInfo->pSetupReferenceSlot` is not `NULL` and `pDecodeInfo->pSetupReferenceSlot->pPictureResource` refers to the same image subresource as `pDecodeInfo->dstPictureResource`, then the image subresource referred to by `pDecodeInfo->dstPictureResource` **must** be in the `VK_IMAGE_LAYOUT_VIDEO_DECODE_DPB_KHR` layout at the time the video decode operation is executed on the device
- VUID-vkCmdDecodeVideoKHR-pDecodeInfo-07254
  If `pDecodeInfo->pSetupReferenceSlot` is not `NULL`, then the image subresource referred to by `pDecodeInfo->pSetupReferenceSlot->pPictureResource` **must** be in the `VK_IMAGE_LAYOUT_VIDEO_DECODE_DPB_KHR` layout at the time the video decode operation is executed on the device
- VUID-vkCmdDecodeVideoKHR-pPictureResource-07255
  The image subresource referred to by the `pPictureResource` member of each element of `pDecodeInfo->pReferenceSlots` **must** be in the `VK_IMAGE_LAYOUT_VIDEO_DECODE_DPB_KHR` layout at the time the video decode operation is executed on the device
- VUID-vkCmdDecodeVideoKHR-pNext-07152
  If the bound video session was created with the video codec operation `VK_VIDEO_CODEC_OPERATION_DECODE_H264_BIT_KHR`, then the `pNext` chain of `pDecodeInfo` **must** include a VkVideoDecodeH264PictureInfoKHR structure
- VUID-vkCmdDecodeVideoKHR-None-07258
  If the bound video session was created with the video codec operation `VK_VIDEO_CODEC_OPERATION_DECODE_H264_BIT_KHR` but was not created with interlaced frame support, then the decode output picture **must** represent a frame
- VUID-vkCmdDecodeVideoKHR-pSliceOffsets-07153
  If the bound video session was created with the video codec operation `VK_VIDEO_CODEC_OPERATION_DECODE_H264_BIT_KHR`, then all elements of the `pSliceOffsets` member of the VkVideoDecodeH264PictureInfoKHR structure included in the `pNext` chain of `pDecodeInfo` **must** be less than `pDecodeInfo->srcBufferRange`
- VUID-vkCmdDecodeVideoKHR-StdVideoH264SequenceParameterSet-07154
  If the bound video session was created with the video codec operation `VK_VIDEO_CODEC_OPERATION_DECODE_H264_BIT_KHR`, then the bound video session parameters object **must** contain a `StdVideoH264SequenceParameterSet` entry with `seq_parameter_set_id` matching `StdVideoDecodeH264PictureInfo::seq_parameter_set_id` that is provided in the `pStdPictureInfo` member of the VkVideoDecodeH264PictureInfoKHR structure included in the `pNext` chain of `pDecodeInfo`
- VUID-vkCmdDecodeVideoKHR-StdVideoH264PictureParameterSet-07155
  If the bound video session was created with the video codec operation `VK_VIDEO_CODEC_OPERATION_DECODE_H264_BIT_KHR`, then the bound video session parameters object **must** contain a `StdVideoH264PictureParameterSet` entry with `seq_parameter_set_id` and `pic_parameter_set_id` matching `StdVideoDecodeH264PictureInfo::seq_parameter_set_id` and `StdVideoDecodeH264PictureInfo::pic_parameter_set_id`, respectively, that are provided in the `pStdPictureInfo` member of the VkVideoDecodeH264PictureInfoKHR structure included in the `pNext` chain of `pDecodeInfo`

- VUID-vkCmdDecodeVideoKHR-pDecodeInfo-07156
  If the bound video session was created with the video codec operation `VK_VIDEO_CODEC_OPERATION_DECODE_H264_BIT_KHR` and `pDecodeInfo->pSetupReferenceSlot` is not `NULL`, then the `pNext` chain of `pDecodeInfo->pSetupReferenceSlot` **must** include a VkVideoDecodeH264DpbSlotInfoKHR structure
- VUID-vkCmdDecodeVideoKHR-pDecodeInfo-07259
  If the bound video session was created with the video codec operation `VK_VIDEO_CODEC_OPERATION_DECODE_H264_BIT_KHR` but was not created with interlaced frame support, and `pDecodeInfo->pSetupReferenceSlot` is not `NULL`, then the reconstructed picture **must** represent a frame
- VUID-vkCmdDecodeVideoKHR-pNext-07157
  If the bound video session was created with the video codec operation `VK_VIDEO_CODEC_OPERATION_DECODE_H264_BIT_KHR`, then the `pNext` chain of each element of `pDecodeInfo->pReferenceSlots` **must** include a VkVideoDecodeH264DpbSlotInfoKHR structure
- VUID-vkCmdDecodeVideoKHR-pDecodeInfo-07260
  If the bound video session was created with the video codec operation `VK_VIDEO_CODEC_OPERATION_DECODE_H264_BIT_KHR` but was not created with interlaced frame support, then each active reference picture corresponding to the elements of `pDecodeInfo->pReferenceSlots` **must** represent a frame
- VUID-vkCmdDecodeVideoKHR-pDecodeInfo-07261
  If the bound video session was created with the video codec operation `VK_VIDEO_CODEC_OPERATION_DECODE_H264_BIT_KHR`, `pDecodeInfo->pSetupReferenceSlot` is not `NULL`, and the decode output picture represents a frame, then the reconstructed picture **must** also represent a frame
- VUID-vkCmdDecodeVideoKHR-pDecodeInfo-07262
  If the bound video session was created with the video codec operation `VK_VIDEO_CODEC_OPERATION_DECODE_H264_BIT_KHR`, `pDecodeInfo->pSetupReferenceSlot` is not `NULL`, and the decode output picture represents a top field, then the reconstructed picture **must** also represent a top field
- VUID-vkCmdDecodeVideoKHR-pDecodeInfo-07263
  If the bound video session was created with the video codec operation `VK_VIDEO_CODEC_OPERATION_DECODE_H264_BIT_KHR`, `pDecodeInfo->pSetupReferenceSlot` is not `NULL`, and the decode output picture represents a bottom field, then the reconstructed picture **must** also represent a bottom field
- VUID-vkCmdDecodeVideoKHR-pDecodeInfo-07266
  If the bound video session was created with the video codec operation `VK_VIDEO_CODEC_OPERATION_DECODE_H264_BIT_KHR` and an active reference picture corresponding to any element of `pDecodeInfo->pReferenceSlots` represents a frame, then the DPB slot index of the bound video session specified by the `slotIndex` member of that element **must** be currently associated with a frame picture matching the video picture resource specified by the `pPictureResource` member of the same element at the time the command is executed on the device
- VUID-vkCmdDecodeVideoKHR-pDecodeInfo-07267
  If the bound video session was created with the video codec operation

VK_VIDEO_CODEC_OPERATION_DECODE_H264_BIT_KHR and an active reference picture corresponding to any element of pDecodeInfo->pReferenceSlots represents a top field, then the DPB slot index of the bound video session specified by the slotIndex member of that element **must** be currently associated with a top field picture matching the video picture resource specified by the pPictureResource member of the same element at the time the command is executed on the device

- VUID-vkCmdDecodeVideoKHR-pDecodeInfo-07268
  If the bound video session was created with the video codec operation VK_VIDEO_CODEC_OPERATION_DECODE_H264_BIT_KHR and an active reference picture corresponding to any element of pDecodeInfo->pReferenceSlots represents a bottom field, then the DPB slot index of the bound video session specified by the slotIndex member of that element **must** be currently associated with a bottom field picture matching the video picture resource specified by the pPictureResource member of the same element at the time the command is executed on the device

- VUID-vkCmdDecodeVideoKHR-pNext-07158
  If the bound video session was created with the video codec operation VK_VIDEO_CODEC_OPERATION_DECODE_H265_BIT_KHR, then the pNext chain of pDecodeInfo **must** include a VkVideoDecodeH265PictureInfoKHR structure

- VUID-vkCmdDecodeVideoKHR-pSliceSegmentOffsets-07159
  If the bound video session was created with the video codec operation VK_VIDEO_CODEC_OPERATION_DECODE_H265_BIT_KHR, then all elements of the pSliceSegmentOffsets member of the VkVideoDecodeH265PictureInfoKHR structure included in the pNext chain of pDecodeInfo **must** be less than pDecodeInfo->srcBufferRange

- VUID-vkCmdDecodeVideoKHR-StdVideoH265VideoParameterSet-07160
  If the bound video session was created with the video codec operation VK_VIDEO_CODEC_OPERATION_DECODE_H265_BIT_KHR, then the bound video session parameters object **must** contain a StdVideoH265VideoParameterSet entry with vps_video_parameter_set_id matching StdVideoDecodeH265PictureInfo::sps_video_parameter_set_id that is provided in the pStdPictureInfo member of the VkVideoDecodeH265PictureInfoKHR structure included in the pNext chain of pDecodeInfo

- VUID-vkCmdDecodeVideoKHR-StdVideoH265SequenceParameterSet-07161
  If the bound video session was created with the video codec operation VK_VIDEO_CODEC_OPERATION_DECODE_H265_BIT_KHR, then the bound video session parameters object **must** contain a StdVideoH265SequenceParameterSet entry with sps_video_parameter_set_id and sps_seq_parameter_set_id matching StdVideoDecodeH265PictureInfo::sps_video_parameter_set_id and StdVideoDecodeH265PictureInfo::pps_seq_parameter_set_id, respectively, that are provided in the pStdPictureInfo member of the VkVideoDecodeH265PictureInfoKHR structure included in the pNext chain of pDecodeInfo

- VUID-vkCmdDecodeVideoKHR-StdVideoH265PictureParameterSet-07162
  If the bound video session was created with the video codec operation VK_VIDEO_CODEC_OPERATION_DECODE_H265_BIT_KHR, then the bound video session parameters object **must** contain a StdVideoH265PictureParameterSet entry with sps_video_parameter_set_id, pps_seq_parameter_set_id, and pps_pic_parameter_set_id matching StdVideoDecodeH265PictureInfo::sps_video_parameter_set_id,

StdVideoDecodeH265PictureInfo::pps_seq_parameter_set_id, and StdVideoDecodeH265PictureInfo::pps_pic_parameter_set_id, respectively, that are provided in the pStdPictureInfo member of the VkVideoDecodeH265PictureInfoKHR structure included in the pNext chain of pDecodeInfo

- VUID-vkCmdDecodeVideoKHR-pDecodeInfo-07163
  If the bound video session was created with the video codec operation VK_VIDEO_CODEC_OPERATION_DECODE_H265_BIT_KHR and pDecodeInfo->pSetupReferenceSlot is not NULL, then the pNext chain of pDecodeInfo->pSetupReferenceSlot **must** include a VkVideoDecodeH265DpbSlotInfoKHR structure
- VUID-vkCmdDecodeVideoKHR-pNext-07164
  If the bound video session was created with the video codec operation VK_VIDEO_CODEC_OPERATION_DECODE_H265_BIT_KHR, then the pNext chain of each element of pDecodeInfo->pReferenceSlots **must** include a VkVideoDecodeH265DpbSlotInfoKHR structure
- VUID-vkCmdDecodeVideoKHR-filmGrainSupport-09248
  If the bound video session was created with the video codec operation VK_VIDEO_CODEC_OPERATION_DECODE_AV1_BIT_KHR and VkVideoDecodeAV1ProfileInfoKHR::filmGrainSupport set to VK_FALSE, then film grain **must** not be enabled for the decoded picture
- VUID-vkCmdDecodeVideoKHR-pDecodeInfo-09249
  If the bound video session was created with the video codec operation VK_VIDEO_CODEC_OPERATION_DECODE_AV1_BIT_KHR, pDecodeInfo->pSetupReferenceSlot is not NULL, and film grain is enabled for the decoded picture, then the video picture resources specified by pDecodeInfo->dstPictureResource and pDecodeInfo->pSetupReferenceSlot->pPictureResource **must** not match
- VUID-vkCmdDecodeVideoKHR-pNext-09250
  If the bound video session was created with the video codec operation VK_VIDEO_CODEC_OPERATION_DECODE_AV1_BIT_KHR, then the pNext chain of pDecodeInfo **must** include a VkVideoDecodeAV1PictureInfoKHR structure
- VUID-vkCmdDecodeVideoKHR-frameHeaderOffset-09251
  If the bound video session was created with the video codec operation VK_VIDEO_CODEC_OPERATION_DECODE_AV1_BIT_KHR, then the frameHeaderOffset member of the VkVideoDecodeAV1PictureInfoKHR structure included in the pNext chain of pDecodeInfo **must** be less than the minimum of pDecodeInfo->srcBufferRange
- VUID-vkCmdDecodeVideoKHR-pTileOffsets-09253
  If the bound video session was created with the video codec operation VK_VIDEO_CODEC_OPERATION_DECODE_AV1_BIT_KHR, then all elements of the pTileOffsets member of the VkVideoDecodeAV1PictureInfoKHR structure included in the pNext chain of pDecodeInfo **must** be less than pDecodeInfo->srcBufferRange
- VUID-vkCmdDecodeVideoKHR-pTileOffsets-09252
  If the bound video session was created with the video codec operation VK_VIDEO_CODEC_OPERATION_DECODE_AV1_BIT_KHR, then for each element i of the pTileOffsets and pTileSizes members of the VkVideoDecodeAV1PictureInfoKHR structure included in the pNext chain of pDecodeInfo the sum of pTileOffsets[i] and pTileSizes[i] **must** be less than or equal to pDecodeInfo->srcBufferRange

- VUID-vkCmdDecodeVideoKHR-pDecodeInfo-09254
  If the bound video session was created with the video codec operation `VK_VIDEO_CODEC_OPERATION_DECODE_AV1_BIT_KHR` and `pDecodeInfo->pSetupReferenceSlot` is not `NULL`, then the `pNext` chain of `pDecodeInfo->pSetupReferenceSlot` **must** include a VkVideoDecodeAV1DpbSlotInfoKHR structure
- VUID-vkCmdDecodeVideoKHR-pNext-09255
  If the bound video session was created with the video codec operation `VK_VIDEO_CODEC_OPERATION_DECODE_AV1_BIT_KHR`, then the `pNext` chain of each element of `pDecodeInfo->pReferenceSlots` **must** include a VkVideoDecodeAV1DpbSlotInfoKHR structure
- VUID-vkCmdDecodeVideoKHR-referenceNameSlotIndices-09262
  If the bound video session was created with the video codec operation `VK_VIDEO_CODEC_OPERATION_DECODE_AV1_BIT_KHR`, then each element of the `referenceNameSlotIndices` array member of the VkVideoDecodeAV1PictureInfoKHR structure included in the `pNext` chain of `pDecodeInfo` **must** either be negative or **must** equal the `slotIndex` member of one of the elements of `pDecodeInfo->pReferenceSlots`
- VUID-vkCmdDecodeVideoKHR-slotIndex-09263
  If the bound video session was created with the video codec operation `VK_VIDEO_CODEC_OPERATION_DECODE_AV1_BIT_KHR`, then the `slotIndex` member of each element of `pDecodeInfo->pReferenceSlots` **must** equal one of the elements of the `referenceNameSlotIndices` array member of the VkVideoDecodeAV1PictureInfoKHR structure included in the `pNext` chain of `pDecodeInfo`

## Valid Usage (Implicit)

- VUID-vkCmdDecodeVideoKHR-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdDecodeVideoKHR-pDecodeInfo-parameter
  `pDecodeInfo` **must** be a valid pointer to a valid VkVideoDecodeInfoKHR structure
- VUID-vkCmdDecodeVideoKHR-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdDecodeVideoKHR-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support decode operations
- VUID-vkCmdDecodeVideoKHR-renderpass
  This command **must** only be called outside of a render pass instance
- VUID-vkCmdDecodeVideoKHR-videocoding
  This command **must** only be called inside of a video coding scope
- VUID-vkCmdDecodeVideoKHR-bufferlevel
  `commandBuffer` **must** be a primary `VkCommandBuffer`

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary | Outside | Inside | Decode | Action |

The `VkVideoDecodeInfoKHR` structure is defined as:

```
// Provided by VK_KHR_video_decode_queue
typedef struct VkVideoDecodeInfoKHR {
    VkStructureType                          sType;
    const void*                              pNext;
    VkVideoDecodeFlagsKHR                    flags;
    VkBuffer                                 srcBuffer;
    VkDeviceSize                             srcBufferOffset;
    VkDeviceSize                             srcBufferRange;
    VkVideoPictureResourceInfoKHR            dstPictureResource;
    const VkVideoReferenceSlotInfoKHR*       pSetupReferenceSlot;
    uint32_t                                 referenceSlotCount;
    const VkVideoReferenceSlotInfoKHR*       pReferenceSlots;
} VkVideoDecodeInfoKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is reserved for future use.
- `srcBuffer` is the source video bitstream buffer to read the encoded bitstream from.
- `srcBufferOffset` is the starting offset in bytes from the start of `srcBuffer` to read the encoded bitstream from.
- `srcBufferRange` is the size in bytes of the encoded bitstream to decode from `srcBuffer`, starting from `srcBufferOffset`.
- `dstPictureResource` is the video picture resource to use as the decode output picture.
- `pSetupReferenceSlot` is `NULL` or a pointer to a VkVideoReferenceSlotInfoKHR structure specifying the reconstructed picture information.
- `referenceSlotCount` is the number of elements in the `pReferenceSlots` array.
- `pReferenceSlots` is `NULL` or a pointer to an array of VkVideoReferenceSlotInfoKHR structures

describing the DPB slots and corresponding reference picture resources to use in this video decode operation (the set of active reference pictures).

## Valid Usage

- VUID-VkVideoDecodeInfoKHR-srcBuffer-07165
  `srcBuffer` **must** have been created with `VK_BUFFER_USAGE_VIDEO_DECODE_SRC_BIT_KHR` set
- VUID-VkVideoDecodeInfoKHR-srcBufferOffset-07166
  `srcBufferOffset` **must** be less than the size of `srcBuffer`
- VUID-VkVideoDecodeInfoKHR-srcBufferRange-07167
  `srcBufferRange` **must** be less than or equal to the size of `srcBuffer` minus `srcBufferOffset`
- VUID-VkVideoDecodeInfoKHR-pSetupReferenceSlot-07168
  If `pSetupReferenceSlot` is not `NULL`, then its `slotIndex` member **must** not be negative
- VUID-VkVideoDecodeInfoKHR-pSetupReferenceSlot-07169
  If `pSetupReferenceSlot` is not `NULL`, then its `pPictureResource` **must** not be `NULL`
- VUID-VkVideoDecodeInfoKHR-slotIndex-07171
  The `slotIndex` member of each element of `pReferenceSlots` **must** not be negative
- VUID-VkVideoDecodeInfoKHR-pPictureResource-07172
  The `pPictureResource` member of each element of `pReferenceSlots` **must** not be `NULL`

## Valid Usage (Implicit)

- VUID-VkVideoDecodeInfoKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_VIDEO_DECODE_INFO_KHR`
- VUID-VkVideoDecodeInfoKHR-pNext-pNext
  Each `pNext` member of any structure (including this one) in the `pNext` chain **must** be either `NULL` or a pointer to a valid instance of VkVideoDecodeAV1PictureInfoKHR, VkVideoDecodeH264PictureInfoKHR, VkVideoDecodeH265PictureInfoKHR, or VkVideoInlineQueryInfoKHR
- VUID-VkVideoDecodeInfoKHR-sType-unique
  The `sType` value of each struct in the `pNext` chain **must** be unique
- VUID-VkVideoDecodeInfoKHR-flags-zerobitmask
  `flags` **must** be `0`
- VUID-VkVideoDecodeInfoKHR-srcBuffer-parameter
  `srcBuffer` **must** be a valid VkBuffer handle
- VUID-VkVideoDecodeInfoKHR-dstPictureResource-parameter
  `dstPictureResource` **must** be a valid VkVideoPictureResourceInfoKHR structure
- VUID-VkVideoDecodeInfoKHR-pSetupReferenceSlot-parameter
  If `pSetupReferenceSlot` is not `NULL`, `pSetupReferenceSlot` **must** be a valid pointer to a valid VkVideoReferenceSlotInfoKHR structure
- VUID-VkVideoDecodeInfoKHR-pReferenceSlots-parameter

If `referenceSlotCount` is not `0`, `pReferenceSlots` **must** be a valid pointer to an array of `referenceSlotCount` valid VkVideoReferenceSlotInfoKHR structures

```
// Provided by VK_KHR_video_decode_queue
typedef VkFlags VkVideoDecodeFlagsKHR;
```

`VkVideoDecodeFlagsKHR` is a bitmask type for setting a mask, but is currently reserved for future use.

## 42.12. H.264 Decode Operations

Video decode operations using an H.264 decode profile **can** be used to decode elementary video stream sequences compliant to the ITU-T H.264 Specification.

| | |
|---|---|
| **NOTE** | Refer to the Preamble for information on how the Khronos Intellectual Property Rights Policy relates to normative references to external materials not created by Khronos. |

This process is performed according to the video decode operation steps with the codec-specific semantics defined in section 8 of the ITU-T H.264 Specification as follows:

- Syntax elements, derived values, and other parameters are applied from the following structures:
  - The `StdVideoH264SequenceParameterSet` structure corresponding to the active SPS specifying the H.264 sequence parameter set.
  - The `StdVideoH264PictureParameterSet` structure corresponding to the active PPS specifying the H.264 picture parameter set.
  - The `StdVideoDecodeH264PictureInfo` structure specifying the H.264 picture information.
  - The `StdVideoDecodeH264ReferenceInfo` structures specifying the H.264 reference information corresponding to the optional reconstructed picture and any active reference pictures.
- The contents of the provided video bitstream buffer range are interpreted as defined in the H.264 Decode Bitstream Data Access section.
- Picture data in the video picture resources corresponding to the used active reference pictures, decode output picture, and optional reconstructed picture is accessed as defined in the H.264 Decode Picture Data Access section.
- The decision on reference picture setup is made according to the parameters specified in the H.264 picture information.

If the parameters and the bitstream adhere to the syntactic and semantic requirements defined in the corresponding sections of the ITU-T H.264 Specification, as described above, and the DPB slots associated with the active reference pictures all refer to valid picture references, then the video decode operation will complete successfully. Otherwise, the video decode operation **may** complete unsuccessfully.

### 42.12.1. H.264 Decode Bitstream Data Access

If the target decode output picture is a frame, then the video bitstream buffer range **should** contain a VCL NAL unit comprised of the slice headers and data of a picture representing an entire frame, as defined in sections 7.3.3 and 7.3.4, and this data is interpreted as defined in sections 7.4.3 and 7.4.4 of the ITU-T H.264 Specification, respectively.

If the target decode output picture is a field, then the video bitstream buffer range should contain a VCL NAL unit comprised of the slice headers and data of a picture representing a field, as defined in sections 7.3.3 and 7.3.4, and this data is interpreted as defined in sections 7.4.3 and 7.4.4 of the ITU-T H.264 Specification, respectively.

The offsets provided in VkVideoDecodeH264PictureInfoKHR::`pSliceOffsets` **should** specify the starting offsets corresponding to each slice header within the video bitstream buffer range.

### 42.12.2. H.264 Decode Picture Data Access

The effective `imageOffset` and `imageExtent` corresponding to a decode output picture, reference picture, or reconstructed picture used in video decode operations with an H.264 decode profile are defined as follows:

- `imageOffset` is (`codedOffset.x`,`codedOffset.y`) and `imageExtent` is (`codedExtent.width`, `codedExtent.height`), if the picture represents a frame.
- `imageOffset` is (`codedOffset.x`,`codedOffset.y`) and `imageExtent` is (`codedExtent.width`, `codedExtent.height`), if the picture represents a field and the picture layout of the used H.264 decode profile is `VK_VIDEO_DECODE_H264_PICTURE_LAYOUT_INTERLACED_INTERLEAVED_LINES_BIT_KHR`.
- `imageOffset` is (`codedOffset.x`,`codedOffset.y`) and `imageExtent` is (`codedExtent.width`, `codedExtent.height` / 2), if the picture represents a field and the picture layout of the used H.264 decode profile is `VK_VIDEO_DECODE_H264_PICTURE_LAYOUT_INTERLACED_SEPARATE_PLANES_BIT_KHR`.

Where `codedOffset` and `codedExtent` are the members of the VkVideoPictureResourceInfoKHR structure corresponding to the picture.

However, accesses to image data within a video picture resource happen at the granularity indicated by VkVideoCapabilitiesKHR::`pictureAccessGranularity`, as returned by vkGetPhysicalDeviceVideoCapabilitiesKHR for the used video profile. This means that the complete image subregion accessed by video coding operations using an H.264 decode profile for the video picture resource is defined as the set of texels within the coordinate range:

([`startX`,`endX`), [`startY`,`endY`))

Where:

- `startX` equals `imageOffset.x` rounded down to the nearest integer multiple of `pictureAccessGranularity.width`;
- `endX` equals `imageOffset.x` + `imageExtent.width` rounded up to the nearest integer multiple of `pictureAccessGranularity.width` and clamped to the width of the image subresource referred to

by the corresponding VkVideoPictureResourceInfoKHR structure;

- startY equals imageOffset.y rounded down to the nearest integer multiple of pictureAccessGranularity.height;
- endY equals imageOffset.y + imageExtent.height rounded up to the nearest integer multiple of pictureAccessGranularity.height and clamped to the height of the image subresource referred to by the corresponding VkVideoPictureResourceInfoKHR structure.

In case of video decode operations using an H.264 decode profile, any access to a picture at the coordinates (x,y), as defined by the ITU-T H.264 Specification, is an access to the image subresource referred to by the corresponding VkVideoPictureResourceInfoKHR structure at the texel coordinates specified below:

- (x,y), if the accessed picture represents a frame.
- (x,y × 2), if the accessed picture represents a top field and the picture layout of the used H.264 decode profile is VK_VIDEO_DECODE_H264_PICTURE_LAYOUT_INTERLACED_INTERLEAVED_LINES_BIT_KHR.
- (x,y × 2 + 1), if the accessed picture represents a bottom field and the picture layout of the used H.264 decode profile is VK_VIDEO_DECODE_H264_PICTURE_LAYOUT_INTERLACED_INTERLEAVED_LINES_BIT_KHR.
- (x,y), if the accessed picture represents a top field and the picture layout of the used H.264 decode profile is VK_VIDEO_DECODE_H264_PICTURE_LAYOUT_INTERLACED_SEPARATE_PLANES_BIT_KHR.
- (codedOffset.x + x,codedOffset.y + y), if the accessed picture represents a bottom field and the picture layout of the used H.264 decode profile is VK_VIDEO_DECODE_H264_PICTURE_LAYOUT_INTERLACED_SEPARATE_PLANES_BIT_KHR.

Where codedOffset is the member of the corresponding VkVideoPictureResourceInfoKHR structure.

### 42.12.3. H.264 Decode Profile

A video profile supporting H.264 video decode operations is specified by setting VkVideoProfileInfoKHR::videoCodecOperation to VK_VIDEO_CODEC_OPERATION_DECODE_H264_BIT_KHR and adding a VkVideoDecodeH264ProfileInfoKHR structure to the VkVideoProfileInfoKHR::pNext chain.

The VkVideoDecodeH264ProfileInfoKHR structure is defined as:

```
// Provided by VK_KHR_video_decode_h264
typedef struct VkVideoDecodeH264ProfileInfoKHR {
    VkStructureType                              sType;
    const void*                                  pNext;
    StdVideoH264ProfileIdc                       stdProfileIdc;
    VkVideoDecodeH264PictureLayoutFlagBitsKHR    pictureLayout;
} VkVideoDecodeH264ProfileInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- stdProfileIdc is a StdVideoH264ProfileIdc value specifying the H.264 codec profile IDC, as

defined in section A.2 of the ITU-T H.264 Specification.

- pictureLayout is a VkVideoDecodeH264PictureLayoutFlagBitsKHR value specifying the picture layout used by the H.264 video sequence to be decoded.

### Valid Usage (Implicit)

- VUID-VkVideoDecodeH264ProfileInfoKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_VIDEO_DECODE_H264_PROFILE_INFO_KHR
- VUID-VkVideoDecodeH264ProfileInfoKHR-pictureLayout-parameter
  If pictureLayout is not 0, pictureLayout **must** be a valid VkVideoDecodeH264PictureLayoutFlagBitsKHR value

The H.264 video decode picture layout flags are defined as follows:

```
// Provided by VK_KHR_video_decode_h264
typedef enum VkVideoDecodeH264PictureLayoutFlagBitsKHR {
    VK_VIDEO_DECODE_H264_PICTURE_LAYOUT_PROGRESSIVE_KHR = 0,
    VK_VIDEO_DECODE_H264_PICTURE_LAYOUT_INTERLACED_INTERLEAVED_LINES_BIT_KHR =
0x00000001,
    VK_VIDEO_DECODE_H264_PICTURE_LAYOUT_INTERLACED_SEPARATE_PLANES_BIT_KHR =
0x00000002,
} VkVideoDecodeH264PictureLayoutFlagBitsKHR;
```

- VK_VIDEO_DECODE_H264_PICTURE_LAYOUT_PROGRESSIVE_KHR specifies support for progressive content. This flag has the value 0.
- VK_VIDEO_DECODE_H264_PICTURE_LAYOUT_INTERLACED_INTERLEAVED_LINES_BIT_KHR specifies support for or use of a picture layout for interlaced content where all lines belonging to the top field are decoded to the even-numbered lines within the picture resource, and all lines belonging to the bottom field are decoded to the odd-numbered lines within the picture resource.
- VK_VIDEO_DECODE_H264_PICTURE_LAYOUT_INTERLACED_SEPARATE_PLANES_BIT_KHR specifies support for or use of a picture layout for interlaced content where all lines belonging to a field are grouped together in a single image subregion, and the two fields comprising the frame **can** be stored in separate image subregions of the same image subresource or in separate image subresources.

```
// Provided by VK_KHR_video_decode_h264
typedef VkFlags VkVideoDecodeH264PictureLayoutFlagsKHR;
```

VkVideoDecodeH264PictureLayoutFlagsKHR is a bitmask type for setting a mask of zero or more VkVideoDecodeH264PictureLayoutFlagBitsKHR.

## 42.12.4. H.264 Decode Capabilities

When calling vkGetPhysicalDeviceVideoCapabilitiesKHR to query the capabilities for an H.264 decode profile, the VkVideoCapabilitiesKHR::pNext chain **must** include a

VkVideoDecodeH264CapabilitiesKHR structure that will be filled with the profile-specific capabilities.

The VkVideoDecodeH264CapabilitiesKHR structure is defined as:

```
// Provided by VK_KHR_video_decode_h264
typedef struct VkVideoDecodeH264CapabilitiesKHR {
    VkStructureType         sType;
    void*                   pNext;
    StdVideoH264LevelIdc    maxLevelIdc;
    VkOffset2D              fieldOffsetGranularity;
} VkVideoDecodeH264CapabilitiesKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- maxLevelIdc is a StdVideoH264LevelIdc value indicating the maximum H.264 level supported by the profile, where enum constant STD_VIDEO_H264_LEVEL_IDC_<major>_<minor> identifies H.264 level <major>.<minor> as defined in section A.3 of the ITU-T H.264 Specification.
- fieldOffsetGranularity is the minimum alignment for VkVideoPictureResourceInfoKHR::codedOffset specified for a video picture resource when using the picture layout VK_VIDEO_DECODE_H264_PICTURE_LAYOUT_INTERLACED_SEPARATE_PLANES_BIT_KHR.

**Valid Usage (Implicit)**

- VUID-VkVideoDecodeH264CapabilitiesKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_VIDEO_DECODE_H264_CAPABILITIES_KHR

## 42.12.5. H.264 Decode Parameter Sets

Video session parameters objects created with the video codec operation VK_VIDEO_CODEC_OPERATION_DECODE_H264_BIT_KHR **can** contain the following types of parameters:

**H.264 Sequence Parameter Sets (SPS)**

Represented by StdVideoH264SequenceParameterSet structures and interpreted as follows:

- reserved1 and reserved2 are used only for padding purposes and are otherwise ignored;
- seq_parameter_set_id is used as the key of the SPS entry;
- level_idc is one of the enum constants STD_VIDEO_H264_LEVEL_IDC_<major>_<minor> identifying the H.264 level <major>.<minor> as defined in section A.3 of the ITU-T H.264 Specification;
- if flags.seq_scaling_matrix_present_flag is set, then the StdVideoH264ScalingLists structure pointed to by pScalingLists is interpreted as follows:
  - scaling_list_present_mask is a bitmask where bit index i corresponds to seq_scaling_list_present_flag[i] as defined in section 7.4.2.1 of the ITU-T H.264

Specification;

  - `use_default_scaling_matrix_mask` is a bitmask where bit index i corresponds to `UseDefaultScalingMatrix4x4Flag[i]`, when i < 6, or corresponds to `UseDefaultScalingMatrix8x8Flag[i-6]`, otherwise, as defined in section 7.3.2.1 of the ITU-T H.264 Specification;
  - `ScalingList4x4` and `ScalingList8x8` correspond to the identically named syntax elements defined in section 7.3.2.1 of the ITU-T H.264 Specification;

- if `flags.vui_parameters_present_flag` is set, then `pSequenceParameterSetVui` is a pointer to a `StdVideoH264SequenceParameterSetVui` structure that is interpreted as follows:
  - `reserved1` is used only for padding purposes and is otherwise ignored;
  - if `flags.nal_hrd_parameters_present_flag` or `flags.vcl_hrd_parameters_present_flag` is set, then the `StdVideoH264HrdParameters` structure pointed to by `pHrdParameters` is interpreted as follows:
    - `reserved1` is used only for padding purposes and is otherwise ignored;
    - all other members of `StdVideoH264HrdParameters` are interpreted as defined in section E.2.2 of the ITU-T H.264 Specification;
  - all other members of `StdVideoH264SequenceParameterSetVui` are interpreted as defined in section E.2.1 of the ITU-T H.264 Specification;
- all other members of `StdVideoH264SequenceParameterSet` are interpreted as defined in section 7.4.2.1 of the ITU-T H.264 Specification.

**H.264 Picture Parameter Sets (PPS)**

Represented by `StdVideoH264PictureParameterSet` structures and interpreted as follows:

- the pair constructed from `seq_parameter_set_id` and `pic_parameter_set_id` is used as the key of the PPS entry;
- if `flags.pic_scaling_matrix_present_flag` is set, then the `StdVideoH264ScalingLists` structure pointed to by `pScalingLists` is interpreted as follows:
  - `scaling_list_present_mask` is a bitmask where bit index i corresponds to `pic_scaling_list_present_flag[i]` as defined in section 7.4.2.2 of the ITU-T H.264 Specification;
  - `use_default_scaling_matrix_mask` is a bitmask where bit index i corresponds to `UseDefaultScalingMatrix4x4Flag[i]`, when i < 6, or corresponds to `UseDefaultScalingMatrix8x8Flag[i-6]`, otherwise, as defined in section 7.3.2.2 of the ITU-T H.264 Specification;
  - `ScalingList4x4` and `ScalingList8x8` correspond to the identically named syntax elements defined in section 7.3.2.2 of the ITU-T H.264 Specification;
- all other members of `StdVideoH264PictureParameterSet` are interpreted as defined in section 7.4.2.2 of the ITU-T H.264 Specification.

When a video session parameters object is created with the codec operation

VK_VIDEO_CODEC_OPERATION_DECODE_H264_BIT_KHR, the VkVideoSessionParametersCreateInfoKHR::pNext chain **must** include a VkVideoDecodeH264SessionParametersCreateInfoKHR structure specifying the capacity and initial contents of the object.

The VkVideoDecodeH264SessionParametersCreateInfoKHR structure is defined as:

```
// Provided by VK_KHR_video_decode_h264
typedef struct VkVideoDecodeH264SessionParametersCreateInfoKHR {
    VkStructureType                                         sType;
    const void*                                             pNext;
    uint32_t                                                maxStdSPSCount;
    uint32_t                                                maxStdPPSCount;
    const VkVideoDecodeH264SessionParametersAddInfoKHR*     pParametersAddInfo;
} VkVideoDecodeH264SessionParametersCreateInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- maxStdSPSCount is the maximum number of H.264 SPS entries the created VkVideoSessionParametersKHR **can** contain.
- maxStdPPSCount is the maximum number of H.264 PPS entries the created VkVideoSessionParametersKHR **can** contain.
- pParametersAddInfo is NULL or a pointer to a VkVideoDecodeH264SessionParametersAddInfoKHR structure specifying H.264 parameters to add upon object creation.

### Valid Usage (Implicit)

- VUID-VkVideoDecodeH264SessionParametersCreateInfoKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_VIDEO_DECODE_H264_SESSION_PARAMETERS_CREATE_INFO_KHR
- VUID-VkVideoDecodeH264SessionParametersCreateInfoKHR-pParametersAddInfo-parameter
  If pParametersAddInfo is not NULL, pParametersAddInfo **must** be a valid pointer to a valid VkVideoDecodeH264SessionParametersAddInfoKHR structure

The VkVideoDecodeH264SessionParametersAddInfoKHR structure is defined as:

```
// Provided by VK_KHR_video_decode_h264
typedef struct VkVideoDecodeH264SessionParametersAddInfoKHR {
    VkStructureType                             sType;
    const void*                                 pNext;
    uint32_t                                    stdSPSCount;
    const StdVideoH264SequenceParameterSet*     pStdSPSs;
    uint32_t                                    stdPPSCount;
    const StdVideoH264PictureParameterSet*      pStdPPSs;
} VkVideoDecodeH264SessionParametersAddInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- stdSPSCount is the number of elements in the pStdSPSs array.
- pStdSPSs is a pointer to an array of StdVideoH264SequenceParameterSet structures describing the H.264 SPS entries to add.
- stdPPSCount is the number of elements in the pStdPPSs array.
- pStdPPSs is a pointer to an array of StdVideoH264PictureParameterSet structures describing the H.264 PPS entries to add.

This structure **can** be specified in the following places:

- In the pParametersAddInfo member of the VkVideoDecodeH264SessionParametersCreateInfoKHR structure specified in the pNext chain of VkVideoSessionParametersCreateInfoKHR used to create a video session parameters object. In this case, if the video codec operation the video session parameters object is created with is VK_VIDEO_CODEC_OPERATION_DECODE_H264_BIT_KHR, then it defines the set of initial parameters to add to the created object (see Creating Video Session Parameters).
- In the pNext chain of VkVideoSessionParametersUpdateInfoKHR. In this case, if the video codec operation the video session parameters object to be updated was created with is VK_VIDEO_CODEC_OPERATION_DECODE_H264_BIT_KHR, then it defines the set of parameters to add to it (see Updating Video Session Parameters).

### Valid Usage

- VUID-VkVideoDecodeH264SessionParametersAddInfoKHR-None-04825
  The seq_parameter_set_id member of each StdVideoH264SequenceParameterSet structure specified in the elements of pStdSPSs **must** be unique within pStdSPSs
- VUID-VkVideoDecodeH264SessionParametersAddInfoKHR-None-04826
  The pair constructed from the seq_parameter_set_id and pic_parameter_set_id members of each StdVideoH264PictureParameterSet structure specified in the elements of pStdPPSs **must** be unique within pStdPPSs

### Valid Usage (Implicit)

- VUID-VkVideoDecodeH264SessionParametersAddInfoKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_VIDEO_DECODE_H264_SESSION_PARAMETERS_ADD_INFO_KHR
- VUID-VkVideoDecodeH264SessionParametersAddInfoKHR-pStdSPSs-parameter
  If stdSPSCount is not 0, pStdSPSs **must** be a valid pointer to an array of stdSPSCount StdVideoH264SequenceParameterSet values
- VUID-VkVideoDecodeH264SessionParametersAddInfoKHR-pStdPPSs-parameter
  If stdPPSCount is not 0, pStdPPSs **must** be a valid pointer to an array of stdPPSCount StdVideoH264PictureParameterSet values

### 42.12.6. H.264 Decoding Parameters

The VkVideoDecodeH264PictureInfoKHR structure is defined as:

```
// Provided by VK_KHR_video_decode_h264
typedef struct VkVideoDecodeH264PictureInfoKHR {
    VkStructureType                         sType;
    const void*                             pNext;
    const StdVideoDecodeH264PictureInfo*    pStdPictureInfo;
    uint32_t                                sliceCount;
    const uint32_t*                         pSliceOffsets;
} VkVideoDecodeH264PictureInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- pStdPictureInfo is a pointer to a StdVideoDecodeH264PictureInfo structure specifying H.264 picture information.
- sliceCount is the number of elements in pSliceOffsets.
- pSliceOffsets is a pointer to an array of sliceCount offsets specifying the start offset of the slices of the picture within the video bitstream buffer range specified in VkVideoDecodeInfoKHR.

This structure is specified in the pNext chain of the VkVideoDecodeInfoKHR structure passed to vkCmdDecodeVideoKHR to specify the codec-specific picture information for an H.264 decode operation.

**Decode Output Picture Information**

When this structure is specified in the pNext chain of the VkVideoDecodeInfoKHR structure passed to vkCmdDecodeVideoKHR, the information related to the decode output picture is defined as follows:

- If pStdPictureInfo->flags.field_pic_flag is not set, then the picture represents a frame.
- If pStdPictureInfo->flags.field_pic_flag is set, then the picture represents a field. Specifically:
  - If pStdPictureInfo->flags.bottom_field_flag is not set, then the picture represents the top field of the frame.
  - If pStdPictureInfo->flags.bottom_field_flag is set, then the picture represents the bottom field of the frame.
- The image subregion used is determined according to the H.264 Decode Picture Data Access section.
- The decode output picture is associated with the H.264 picture information provided in pStdPictureInfo.

**Std Picture Information**

The members of the StdVideoDecodeH264PictureInfo structure pointed to by pStdPictureInfo are interpreted as follows:

- reserved1 and reserved2 are used only for padding purposes and are otherwise ignored;
- flags.is_intra as defined in section 3.73 of the ITU-T H.264 Specification;
- flags.is_reference as defined in section 3.136 of the ITU-T H.264 Specification;
- flags.complementary_field_pair as defined in section 3.35 of the ITU-T H.264 Specification;
- seq_parameter_set_id and pic_parameter_set_id are used to identify the active parameter sets, as described below;
- all other members are interpreted as defined in section 7.4.3 of the ITU-T H.264 Specification.

Reference picture setup is controlled by the value of StdVideoDecodeH264PictureInfo::flags.is_reference. If it is set and a reconstructed picture is specified, then the latter is used as the target of picture reconstruction to activate the DPB slot specified in pDecodeInfo->pSetupReferenceSlot->slotIndex. If StdVideoDecodeH264PictureInfo::flags.is_reference is not set, but a reconstructed picture is specified, then the corresponding picture reference associated with the DPB slot is invalidated, as described in the DPB Slot States section.

**Active Parameter Sets**

The members of the StdVideoDecodeH264PictureInfo structure pointed to by pStdPictureInfo are used to select the active parameter sets to use from the bound video session parameters object, as follows:

- The *active SPS* is the SPS identified by the key specified in StdVideoDecodeH264PictureInfo::seq_parameter_set_id.
- The *active PPS* is the PPS identified by the key specified by the pair constructed from StdVideoDecodeH264PictureInfo::seq_parameter_set_id and StdVideoDecodeH264PictureInfo::pic_parameter_set_id.

### Valid Usage (Implicit)

- VUID-VkVideoDecodeH264PictureInfoKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_VIDEO_DECODE_H264_PICTURE_INFO_KHR
- VUID-VkVideoDecodeH264PictureInfoKHR-pStdPictureInfo-parameter
  pStdPictureInfo **must** be a valid pointer to a valid StdVideoDecodeH264PictureInfo value
- VUID-VkVideoDecodeH264PictureInfoKHR-pSliceOffsets-parameter
  pSliceOffsets **must** be a valid pointer to an array of sliceCount uint32_t values
- VUID-VkVideoDecodeH264PictureInfoKHR-sliceCount-arraylength
  sliceCount **must** be greater than 0

The VkVideoDecodeH264DpbSlotInfoKHR structure is defined as:

```
// Provided by VK_KHR_video_decode_h264
typedef struct VkVideoDecodeH264DpbSlotInfoKHR {
    VkStructureType                             sType;
    const void*                                 pNext;
    const StdVideoDecodeH264ReferenceInfo*      pStdReferenceInfo;
} VkVideoDecodeH264DpbSlotInfoKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `pStdReferenceInfo` is a pointer to a `StdVideoDecodeH264ReferenceInfo` structure specifying H.264 reference information.

This structure is specified in the `pNext` chain of VkVideoDecodeInfoKHR::`pSetupReferenceSlot`, if not `NULL`, and the `pNext` chain of the elements of VkVideoDecodeInfoKHR::`pReferenceSlots` to specify the codec-specific reference picture information for an H.264 decode operation.

**Active Reference Picture Information**

When this structure is specified in the `pNext` chain of the elements of VkVideoDecodeInfoKHR::`pReferenceSlots`, one or two elements are added to the list of active reference pictures used by the video decode operation for each element of VkVideoDecodeInfoKHR::`pReferenceSlots` as follows:

- If neither `pStdReferenceInfo->flags.top_field_flag` nor `pStdReferenceInfo->flags.bottom_field_flag` is set, then the picture is added as a frame reference to the list of active reference pictures.
- If `pStdReferenceInfo->flags.top_field_flag` is set, then the picture is added as a top field reference to the list of active reference pictures.
- If `pStdReferenceInfo->flags.bottom_field_flag` is set, then the picture is added as a bottom field reference to the list of active reference pictures.
- For each added reference picture, the corresponding image subregion used is determined according to the H.264 Decode Picture Data Access section.
- Each added reference picture is associated with the DPB slot index specified in the `slotIndex` member of the corresponding element of VkVideoDecodeInfoKHR::`pReferenceSlots`.
- Each added reference picture is associated with the H.264 reference information provided in `pStdReferenceInfo`.

**NOTE** When both the top and bottom field of an interlaced frame currently associated with a DPB slot is intended to be used as an active reference picture and both fields are stored in the same image subregion (which is the case when using `VK_VIDEO_DECODE_H264_PICTURE_LAYOUT_INTERLACED_INTERLEAVED_LINES_BIT_KHR` which stores the two fields at even and odd scanlines of the same image subregion), both references have to be provided through a single VkVideoReferenceSlotInfoKHR structure that has both `flags.top_field_flag` and `flags.bottom_field_flag` set in the `StdVideoDecodeH264ReferenceInfo` structure pointed to by the `pStdReferenceInfo` member of the VkVideoDecodeH264DpbSlotInfoKHR structure included in the

corresponding VkVideoReferenceSlotInfoKHR structure's pNext chain. However, this approach can only be used when both fields are stored in the same image subregion. If that is not the case (e.g. when using VK_VIDEO_DECODE_H264_PICTURE_LAYOUT_INTERLACED_SEPARATE_PLANES_BIT_KHR which requires separate codedOffset values for the two fields and also allows storing the two fields of a frame in separate image layers or entirely separate images), then a separate VkVideoReferenceSlotInfoKHR structure needs to be provided for referencing the two fields, each only setting one of flags.top_field_flag or flags.bottom_field_flag, and providing the appropriate video picture resource information in VkVideoReferenceSlotInfoKHR::pPictureResource.

**Reconstructed Picture Information**

When this structure is specified in the pNext chain of VkVideoDecodeInfoKHR::pSetupReferenceSlot, the information related to the reconstructed picture is defined as follows:

- If neither pStdReferenceInfo->flags.top_field_flag nor pStdReferenceInfo->flags.bottom_field_flag is set, then the picture represents a frame.
- If pStdReferenceInfo->flags.top_field_flag is set, then the picture represents a field, specifically, the top field of the frame.
- If pStdReferenceInfo->flags.bottom_field_flag is set, then the picture represents a field, specifically, the bottom field of the frame.
- The image subregion used is determined according to the H.264 Decode Picture Data Access section.
- If reference picture setup is requested, then the reconstructed picture is used to activate the DPB slot with the index specified in VkVideoDecodeInfoKHR::pSetupReferenceSlot->slotIndex.
- The reconstructed picture is associated with the H.264 reference information provided in pStdReferenceInfo.

**Std Reference Information**

The members of the StdVideoDecodeH264ReferenceInfo structure pointed to by pStdReferenceInfo are interpreted as follows:

- flags.top_field_flag is used to indicate whether the reference is used as top field reference;
- flags.bottom_field_flag is used to indicate whether the reference is used as bottom field reference;
- flags.used_for_long_term_reference is used to indicate whether the picture is marked as "used for long-term reference" as defined in section 8.2.5.1 of the ITU-T H.264 Specification;
- flags.is_non_existing is used to indicate whether the picture is marked as "non-existing" as defined in section 8.2.5.2 of the ITU-T H.264 Specification;
- all other members are interpreted as defined in section 8.2 of the ITU-T H.264 Specification.

### Valid Usage (Implicit)

- VUID-VkVideoDecodeH264DpbSlotInfoKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_VIDEO_DECODE_H264_DPB_SLOT_INFO_KHR`
- VUID-VkVideoDecodeH264DpbSlotInfoKHR-pStdReferenceInfo-parameter
  `pStdReferenceInfo` **must** be a valid pointer to a valid `StdVideoDecodeH264ReferenceInfo` value

## 42.12.7. H.264 Decode Requirements

This section describes the **required** H.264 decoding capabilities for physical devices that have at least one queue family that supports the video codec operation `VK_VIDEO_CODEC_OPERATION_DECODE_H264_BIT_KHR`, as returned by vkGetPhysicalDeviceQueueFamilyProperties2 in VkQueueFamilyVideoPropertiesKHR::`videoCodecOperations`.

*Table 51. Required Video Std Header Versions*

| Video Std Header Name | Version |
|---|---|
| `vulkan_video_codec_h264std_decode` | 1.0.0 |

*Table 52. Required Video Capabilities*

| Video Capability | Requirement | Requirement Type[1] |
|---|---|---|
| **VkVideoCapabilitiesKHR** | | |
| `flags` | - | min |
| `minBitstreamBufferOffsetAlignment` | 4096 | max |
| `minBitstreamBufferSizeAlignment` | 4096 | max |
| `pictureAccessGranularity` | (64,64) | max |
| `minCodedExtent` | - | max |
| `maxCodedExtent` | - | min |
| `maxDpbSlots` | 0 | min |
| `maxActiveReferencePictures` | 0 | min |
| **VkVideoDecodeCapabilitiesKHR** | | |
| `flags` | `VK_VIDEO_DECODE_CAPABILITY_DPB_AND_OUTPUT_COINCIDE_BIT_KHR` or `VK_VIDEO_DECODE_CAPABILITY_DPB_AND_OUTPUT_DISTINCT_BIT_KHR` | min |
| **VkVideoDecodeH264CapabilitiesKHR** | | |

| Video Capability | Requirement | Requirement Type[1] |
|---|---|---|
| `maxLevelIdc` | `STD_VIDEO_H264_LEVEL_IDC_1_0` | min |
| `fieldOffsetGranularity` | (0,0) except for profiles using `VK_VIDEO_DECODE_H264_PICTURE_LAYOUT_INTERLACED_SEPARATE_PLANES_BIT_KHR` | implementation-dependent |

1

The **Requirement Type** column specifies the requirement is either the minimum value all implementations **must** support, the maximum value all implementations **must** support, or the exact value all implementations **must** support. For bitmasks a minimum value is the least bits all implementations **must** set, but they **may** have additional bits set beyond this minimum.

## 42.13. H.265 Decode Operations

Video decode operations using an H.265 decode profile **can** be used to decode elementary video stream sequences compliant to the ITU-T H.265 Specification.

| | |
|---|---|
| **NOTE** | Refer to the Preamble for information on how the Khronos Intellectual Property Rights Policy relates to normative references to external materials not created by Khronos. |

This process is performed according to the video decode operation steps with the codec-specific semantics defined in section 8 of ITU-T H.265 Specification:

- Syntax elements, derived values, and other parameters are applied from the following structures:
  - The `StdVideoH265VideoParameterSet` structure corresponding to the active VPS specifying the H.265 video parameter set.
  - The `StdVideoH265SequenceParameterSet` structure corresponding to the active SPS specifying the H.265 sequence parameter set.
  - The `StdVideoH265PictureParameterSet` structure corresponding to the active PPS specifying the H.265 picture parameter set.
  - The `StdVideoDecodeH265PictureInfo` structure specifying the H.265 picture information.
  - The `StdVideoDecodeH265ReferenceInfo` structures specifying the H.265 reference information corresponding to the optional reconstructed picture and any active reference pictures.
- The contents of the provided video bitstream buffer range are interpreted as defined in the H.265 Decode Bitstream Data Access section.
- Picture data in the video picture resources corresponding to the used active reference pictures, decode output picture, and optional reconstructed picture is accessed as defined in the H.265 Decode Picture Data Access section.

- The decision on reference picture setup is made according to the parameters specified in the H.265 picture information.

If the parameters and the bitstream adhere to the syntactic and semantic requirements defined in the corresponding sections of the ITU-T H.265 Specification, as described above, and the DPB slots associated with the active reference pictures all refer to valid picture references, then the video decode operation will complete successfully. Otherwise, the video decode operation **may** complete unsuccessfully.

### 42.13.1. H.265 Decode Bitstream Data Access

The video bitstream buffer range **should** contain a VCL NAL unit comprised of the slice segment headers and data of a picture representing a frame, as defined in sections 7.3.6 and 7.3.8, and this data is interpreted as defined in sections 7.4.7 and 7.4.9 of the ITU-T H.265 Specification, respectively.

The offsets provided in VkVideoDecodeH265PictureInfoKHR::`pSliceSegmentOffsets` **should** specify the starting offsets corresponding to each slice segment header within the video bitstream buffer range.

### 42.13.2. H.265 Decode Picture Data Access

Accesses to image data within a video picture resource happen at the granularity indicated by VkVideoCapabilitiesKHR::`pictureAccessGranularity`, as returned by vkGetPhysicalDeviceVideoCapabilitiesKHR for the used video profile. Accordingly, the complete image subregion of a decode output picture, reference picture, or reconstructed picture accessed by video coding operations using an H.265 decode profile is defined as the set of texels within the coordinate range:

$$([0,endX), [0,endY))$$

Where:

- `endX` equals `codedExtent.width` rounded up to the nearest integer multiple of `pictureAccessGranularity.width` and clamped to the width of the image subresource referred to by the corresponding VkVideoPictureResourceInfoKHR structure;
- endY equals `codedExtent.height` rounded up to the nearest integer multiple of `pictureAccessGranularity.height` and clamped to the height of the image subresource referred to by the corresponding VkVideoPictureResourceInfoKHR structure;

Where `codedExtent` is the member of the VkVideoPictureResourceInfoKHR structure corresponding to the picture.

In case of video decode operations using an H.265 decode profile, any access to a picture at the coordinates (`x`,`y`), as defined by the ITU-T H.265 Specification, is an access to the image subresource referred to by the corresponding VkVideoPictureResourceInfoKHR structure at the texel coordinates (`x`,`y`).

### 42.13.3. H.265 Decode Profile

A video profile supporting H.265 video decode operations is specified by setting VkVideoProfileInfoKHR::videoCodecOperation to VK_VIDEO_CODEC_OPERATION_DECODE_H265_BIT_KHR and adding a VkVideoDecodeH265ProfileInfoKHR structure to the VkVideoProfileInfoKHR::pNext chain.

The VkVideoDecodeH265ProfileInfoKHR structure is defined as:

```
// Provided by VK_KHR_video_decode_h265
typedef struct VkVideoDecodeH265ProfileInfoKHR {
    VkStructureType           sType;
    const void*               pNext;
    StdVideoH265ProfileIdc    stdProfileIdc;
} VkVideoDecodeH265ProfileInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- stdProfileIdc is a StdVideoH265ProfileIdc value specifying the H.265 codec profile IDC, as defined in section A.3 of the ITU-T H.265 Specification.

**Valid Usage (Implicit)**

- VUID-VkVideoDecodeH265ProfileInfoKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_VIDEO_DECODE_H265_PROFILE_INFO_KHR

### 42.13.4. H.265 Decode Capabilities

When calling vkGetPhysicalDeviceVideoCapabilitiesKHR to query the capabilities for an H.265 decode profile, the VkVideoCapabilitiesKHR::pNext chain **must** include a VkVideoDecodeH265CapabilitiesKHR structure that will be filled with the profile-specific capabilities.

The VkVideoDecodeH265CapabilitiesKHR structure is defined as:

```
// Provided by VK_KHR_video_decode_h265
typedef struct VkVideoDecodeH265CapabilitiesKHR {
    VkStructureType         sType;
    void*                   pNext;
    StdVideoH265LevelIdc    maxLevelIdc;
} VkVideoDecodeH265CapabilitiesKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- maxLevelIdc is a StdVideoH265LevelIdc value indicating the maximum H.265 level supported by the profile, where enum constant STD_VIDEO_H265_LEVEL_IDC_<major>_<minor> identifies H.265 level <major>.<minor> as defined in section A.4 of the ITU-T H.265 Specification.

### Valid Usage (Implicit)

- VUID-VkVideoDecodeH265CapabilitiesKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_VIDEO_DECODE_H265_CAPABILITIES_KHR`

## 42.13.5. H.265 Decode Parameter Sets

Video session parameters objects created with the video codec operation `VK_VIDEO_CODEC_OPERATION_DECODE_H265_BIT_KHR` **can** contain the following types of parameters:

**H.265 Video Parameter Sets (VPS)**

Represented by `StdVideoH265VideoParameterSet` structures and interpreted as follows:

- `reserved1`, `reserved2`, and `reserved3` are used only for padding purposes and are otherwise ignored;
- `vps_video_parameter_set_id` is used as the key of the VPS entry;
- the `max_latency_increase_plus1`, `max_dec_pic_buffering_minus1`, and `max_num_reorder_pics` members of the `StdVideoH265DecPicBufMgr` structure pointed to by `pDecPicBufMgr` correspond to `vps_max_latency_increase_plus1`, `vps_max_dec_pic_buffering_minus1`, and `vps_max_num_reorder_pics`, respectively, as defined in section 7.4.3.1 of the ITU-T H.265 Specification;
- the `StdVideoH265HrdParameters` structure pointed to by `pHrdParameters` is interpreted as follows:
  - `reserved` is used only for padding purposes and is otherwise ignored;
  - `flags.fixed_pic_rate_general_flag` is a bitmask where bit index i corresponds to `fixed_pic_rate_general_flag[i]` as defined in section E.3.2 of the ITU-T H.265 Specification;
  - `flags.fixed_pic_rate_within_cvs_flag` is a bitmask where bit index i corresponds to `fixed_pic_rate_within_cvs_flag[i]` as defined in section E.3.2 of the ITU-T H.265 Specification;
  - `flags.low_delay_hrd_flag` is a bitmask where bit index i corresponds to `low_delay_hrd_flag[i]` as defined in section E.3.2 of the ITU-T H.265 Specification;
  - if `flags.nal_hrd_parameters_present_flag` is set, then `pSubLayerHrdParametersNal` is a pointer to an array of `vps_max_sub_layers_minus1` + 1 number of `StdVideoH265SubLayerHrdParameters` structures where `vps_max_sub_layers_minus1` is the corresponding member of the encompassing `StdVideoH265VideoParameterSet` structure and each element is interpreted as follows:
    - `cbr_flag` is a bitmask where bit index i corresponds to `cbr_flag[i]` as defined in section E.3.3 of the ITU-T H.265 Specification;
    - all other members of the `StdVideoH265SubLayerHrdParameters` structure are interpreted as defined in section E.3.3 of the ITU-T H.265 Specification;

- if flags.vcl_hrd_parameters_present_flag is set, then pSubLayerHrdParametersVcl is a pointer to an array of vps_max_sub_layers_minus1 + 1 number of StdVideoH265SubLayerHrdParameters structures where vps_max_sub_layers_minus1 is the corresponding member of the encompassing StdVideoH265VideoParameterSet structure and each element is interpreted as follows:
  - cbr_flag is a bitmask where bit index i corresponds to cbr_flag[i] as defined in section E.3.3 of the ITU-T H.265 Specification;
  - all other members of the StdVideoH265SubLayerHrdParameters structure are interpreted as defined in section E.3.3 of the ITU-T H.265 Specification;
- all other members of StdVideoH265HrdParameters are interpreted as defined in section E.3.2 of the ITU-T H.265 Specification;

- the StdVideoH265ProfileTierLevel structure pointed to by pProfileTierLevel are interpreted as follows:
  - general_level_idc is one of the enum constants STD_VIDEO_H265_LEVEL_IDC_<major>_<minor> identifying the H.265 level <major>.<minor> as defined in section A.4 of the ITU-T H.265 Specification;
  - all other members of StdVideoH265ProfileTierLevel are interpreted as defined in section 7.4.4 of the ITU-T H.265 Specification;
- all other members of StdVideoH265VideoParameterSet are interpreted as defined in section 7.4.3.1 of the ITU-T H.265 Specification.

#### H.265 Sequence Parameter Sets (SPS)

Represented by StdVideoH265SequenceParameterSet structures and interpreted as follows:

- reserved1 and reserved2 are used only for padding purposes and are otherwise ignored;
- the pair constructed from sps_video_parameter_set_id and sps_seq_parameter_set_id is used as the key of the SPS entry;
- the StdVideoH265ProfileTierLevel structure pointed to by pProfileTierLevel are interpreted as follows:
  - general_level_idc is one of the enum constants STD_VIDEO_H265_LEVEL_IDC_<major>_<minor> identifying the H.265 level <major>.<minor> as defined in section A.4 of the ITU-T H.265 Specification;
  - all other members of StdVideoH265ProfileTierLevel are interpreted as defined in section 7.4.4 of the ITU-T H.265 Specification;
- the max_latency_increase_plus1, max_dec_pic_buffering_minus1, and max_num_reorder_pics members of the StdVideoH265DecPicBufMgr structure pointed to by pDecPicBufMgr correspond to sps_max_latency_increase_plus1, sps_max_dec_pic_buffering_minus1, and sps_max_num_reorder_pics, respectively, as defined in section 7.4.3.2 of the ITU-T H.265 Specification;
- if flags.sps_scaling_list_data_present_flag is set, then the StdVideoH265ScalingLists structure pointed to by pScalingLists is interpreted as follows:

    - ScalingList4x4, ScalingList8x8, ScalingList16x16, and ScalingList32x32 correspond to ScalingList[0], ScalingList[1], ScalingList[2], and ScalingList[3], respectively, as defined in section 7.3.4 of the ITU-T H.265 Specification;
    - ScalingListDCCoef16x16 and ScalingListDCCoef32x32 correspond to scaling_list_dc_coef_minus8[0] and scaling_list_dc_coef_minus8[1], respectively, as defined in section 7.3.4 of the ITU-T H.265 Specification;
- pShortTermRefPicSet is a pointer to an array of num_short_term_ref_pic_sets number of StdVideoH265ShortTermRefPicSet structures where each element is interpreted as follows:
    - reserved1, reserved2, and reserved3 are used only for padding purposes and are otherwise ignored;
    - used_by_curr_pic_flag is a bitmask where bit index i corresponds to used_by_curr_pic_flag[i] as defined in section 7.4.8 of the ITU-T H.265 Specification;
    - use_delta_flag is a bitmask where bit index i corresponds to use_delta_flag[i] as defined in section 7.4.8 of the ITU-T H.265 Specification;
    - used_by_curr_pic_s0_flag is a bitmask where bit index i corresponds to used_by_curr_pic_s0_flag[i] as defined in section 7.4.8 of the ITU-T H.265 Specification;
    - used_by_curr_pic_s1_flag is a bitmask where bit index i corresponds to used_by_curr_pic_s1_flag[i] as defined in section 7.4.8 of the ITU-T H.265 Specification;
    - all other members of StdVideoH265ShortTermRefPicSet are interpreted as defined in section 7.4.8 of the ITU-T H.265 Specification;
- if flags.long_term_ref_pics_present_flag is set then the StdVideoH265LongTermRefPicsSps structure pointed to by pLongTermRefPicsSps is interpreted as follows:
    - used_by_curr_pic_lt_sps_flag is a bitmask where bit index i corresponds to used_by_curr_pic_lt_sps_flag[i] as defined in section 7.4.3.2 of the ITU-T H.265 Specification;
    - all other members of StdVideoH265LongTermRefPicsSps are interpreted as defined in section 7.4.3.2 of the ITU-T H.265 Specification;
- if flags.vui_parameters_present_flag is set, then the StdVideoH265SequenceParameterSetVui structure pointed to by pSequenceParameterSetVui is interpreted as follows:
    - reserved1, reserved2, and reserved3 are used only for padding purposes and are otherwise ignored;
    - the StdVideoH265HrdParameters structure pointed to by pHrdParameters is interpreted as follows:
        - flags.fixed_pic_rate_general_flag is a bitmask where bit index i corresponds to fixed_pic_rate_general_flag[i] as defined in section E.3.2 of the ITU-T H.265 Specification;
        - flags.fixed_pic_rate_within_cvs_flag is a bitmask where bit index i corresponds to fixed_pic_rate_within_cvs_flag[i] as defined in section E.3.2 of the ITU-T H.265 Specification;
        - flags.low_delay_hrd_flag is a bitmask where bit index i corresponds to low_delay_hrd_flag[i] as defined in section E.3.2 of the ITU-T H.265 Specification;

        - if `flags.nal_hrd_parameters_present_flag` is set, then `pSubLayerHrdParametersNal` is a pointer to an array of `sps_max_sub_layers_minus1` + 1 number of `StdVideoH265SubLayerHrdParameters` structures where `sps_max_sub_layers_minus1` is the corresponding member of the encompassing `StdVideoH265SequenceParameterSet` structure and each element is interpreted as follows:
            - `cbr_flag` is a bitmask where bit index i corresponds to `cbr_flag[i]` as defined in section E.3.3 of the ITU-T H.265 Specification;
            - all other members of the `StdVideoH265SubLayerHrdParameters` structure are interpreted as defined in section E.3.3 of the ITU-T H.265 Specification;
        - if `flags.vcl_hrd_parameters_present_flag` is set, then `pSubLayerHrdParametersVcl` is a pointer to an array of `sps_max_sub_layers_minus1` + 1 number of `StdVideoH265SubLayerHrdParameters` structures where `sps_max_sub_layers_minus1` is the corresponding member of the encompassing `StdVideoH265SequenceParameterSet` structure and each element is interpreted as follows:
            - `cbr_flag` is a bitmask where bit index i corresponds to `cbr_flag[i]` as defined in section E.3.3 of the ITU-T H.265 Specification;
            - all other members of the `StdVideoH265SubLayerHrdParameters` structure are interpreted as defined in section E.3.3 of the ITU-T H.265 Specification;
        - all other members of `StdVideoH265HrdParameters` are interpreted as defined in section E.3.2 of the ITU-T H.265 Specification;
    - all other members of `pSequenceParameterSetVui` are interpreted as defined in section E.3.1 of the ITU-T H.265 Specification;
- if `flags.sps_palette_predictor_initializer_present_flag` is set, then the `PredictorPaletteEntries` member of the `StdVideoH265PredictorPaletteEntries` structure pointed to by `pPredictorPaletteEntries` is interpreted as defined in section 7.4.9.13 of the ITU-T H.265 Specification;
- all other members of `StdVideoH265SequenceParameterSet` are interpreted as defined in section 7.4.3.1 of the ITU-T H.265 Specification.

#### H.265 Picture Parameter Sets (PPS)

Represented by `StdVideoH265PictureParameterSet` structures and interpreted as follows:

- `reserved1`, `reserved2`, and `reserved3` are used only for padding purposes and are otherwise ignored;
- the triplet constructed from `sps_video_parameter_set_id`, `pps_seq_parameter_set_id`, and `pps_pic_parameter_set_id` is used as the key of the PPS entry;
- if `flags.pps_scaling_list_data_present_flag` is set, then the `StdVideoH265ScalingLists` structure pointed to by `pScalingLists` is interpreted as follows:
    - `ScalingList4x4`, `ScalingList8x8`, `ScalingList16x16`, and `ScalingList32x32` correspond to `ScalingList[0]`, `ScalingList[1]`, `ScalingList[2]`, and `ScalingList[3]`, respectively, as defined in section 7.3.4 of the ITU-T H.265 Specification;

    - ScalingListDCCoef16x16 and ScalingListDCCoef32x32 correspond to scaling_list_dc_coef_minus8[0] and scaling_list_dc_coef_minus8[1], respectively, as defined in section 7.3.4 of the ITU-T H.265 Specification;
  - if flags.pps_palette_predictor_initializer_present_flag is set, then the PredictorPaletteEntries member of the StdVideoH265PredictorPaletteEntries structure pointed to by pPredictorPaletteEntries is interpreted as defined in section 7.4.9.13 of the ITU-T H.265 Specification;
  - all other members of StdVideoH265PictureParameterSet are interpreted as defined in section 7.4.3.3 of the ITU-T H.265 Specification.

When a video session parameters object is created with the codec operation VK_VIDEO_CODEC_OPERATION_DECODE_H265_BIT_KHR, the VkVideoSessionParametersCreateInfoKHR::pNext chain **must** include a VkVideoDecodeH265SessionParametersCreateInfoKHR structure specifying the capacity and initial contents of the object.

The VkVideoDecodeH265SessionParametersCreateInfoKHR structure is defined as:

```
// Provided by VK_KHR_video_decode_h265
typedef struct VkVideoDecodeH265SessionParametersCreateInfoKHR {
    VkStructureType                                         sType;
    const void*                                             pNext;
    uint32_t                                                maxStdVPSCount;
    uint32_t                                                maxStdSPSCount;
    uint32_t                                                maxStdPPSCount;
    const VkVideoDecodeH265SessionParametersAddInfoKHR*     pParametersAddInfo;
} VkVideoDecodeH265SessionParametersCreateInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- maxStdVPSCount is the maximum number of H.265 VPS entries the created VkVideoSessionParametersKHR **can** contain.
- maxStdSPSCount is the maximum number of H.265 SPS entries the created VkVideoSessionParametersKHR **can** contain.
- maxStdPPSCount is the maximum number of H.265 PPS entries the created VkVideoSessionParametersKHR **can** contain.
- pParametersAddInfo is NULL or a pointer to a VkVideoDecodeH265SessionParametersAddInfoKHR structure specifying H.265 parameters to add upon object creation.

### Valid Usage (Implicit)

- VUID-VkVideoDecodeH265SessionParametersCreateInfoKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_VIDEO_DECODE_H265_SESSION_PARAMETERS_CREATE_INFO_KHR
- VUID-VkVideoDecodeH265SessionParametersCreateInfoKHR-pParametersAddInfo-parameter

If pParametersAddInfo is not NULL, pParametersAddInfo **must** be a valid pointer to a valid VkVideoDecodeH265SessionParametersAddInfoKHR structure

The VkVideoDecodeH265SessionParametersAddInfoKHR structure is defined as:

```
// Provided by VK_KHR_video_decode_h265
typedef struct VkVideoDecodeH265SessionParametersAddInfoKHR {
    VkStructureType                             sType;
    const void*                                 pNext;
    uint32_t                                    stdVPSCount;
    const StdVideoH265VideoParameterSet*        pStdVPSs;
    uint32_t                                    stdSPSCount;
    const StdVideoH265SequenceParameterSet*     pStdSPSs;
    uint32_t                                    stdPPSCount;
    const StdVideoH265PictureParameterSet*      pStdPPSs;
} VkVideoDecodeH265SessionParametersAddInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- stdVPSCount is the number of elements in the pStdVPSs array.
- pStdVPSs is a pointer to an array of StdVideoH265VideoParameterSet structures describing the H.265 VPS entries to add.
- stdSPSCount is the number of elements in the pStdSPSs array.
- pStdSPSs is a pointer to an array of StdVideoH265SequenceParameterSet structures describing the H.265 SPS entries to add.
- stdPPSCount is the number of elements in the pStdPPSs array.
- pStdPPSs is a pointer to an array of StdVideoH265PictureParameterSet structures describing the H.265 PPS entries to add.

This structure **can** be specified in the following places:

- In the pParametersAddInfo member of the VkVideoDecodeH265SessionParametersCreateInfoKHR structure specified in the pNext chain of VkVideoSessionParametersCreateInfoKHR used to create a video session parameters object. In this case, if the video codec operation the video session parameters object is created with is VK_VIDEO_CODEC_OPERATION_DECODE_H265_BIT_KHR, then it defines the set of initial parameters to add to the created object (see Creating Video Session Parameters).
- In the pNext chain of VkVideoSessionParametersUpdateInfoKHR. In this case, if the video codec operation the video session parameters object to be updated was created with is VK_VIDEO_CODEC_OPERATION_DECODE_H265_BIT_KHR, then it defines the set of parameters to add to it (see Updating Video Session Parameters).

### Valid Usage

- VUID-VkVideoDecodeH265SessionParametersAddInfoKHR-None-04833
  The `vps_video_parameter_set_id` member of each `StdVideoH265VideoParameterSet` structure specified in the elements of `pStdVPSs` **must** be unique within `pStdVPSs`
- VUID-VkVideoDecodeH265SessionParametersAddInfoKHR-None-04834
  The pair constructed from the `sps_video_parameter_set_id` and `sps_seq_parameter_set_id` members of each `StdVideoH265SequenceParameterSet` structure specified in the elements of `pStdSPSs` **must** be unique within `pStdSPSs`
- VUID-VkVideoDecodeH265SessionParametersAddInfoKHR-None-04835
  The triplet constructed from the `sps_video_parameter_set_id`, `pps_seq_parameter_set_id`, and `pps_pic_parameter_set_id` members of each `StdVideoH265PictureParameterSet` structure specified in the elements of `pStdPPSs` **must** be unique within `pStdPPSs`

### Valid Usage (Implicit)

- VUID-VkVideoDecodeH265SessionParametersAddInfoKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_VIDEO_DECODE_H265_SESSION_PARAMETERS_ADD_INFO_KHR`
- VUID-VkVideoDecodeH265SessionParametersAddInfoKHR-pStdVPSs-parameter
  If `stdVPSCount` is not `0`, `pStdVPSs` **must** be a valid pointer to an array of `stdVPSCount` `StdVideoH265VideoParameterSet` values
- VUID-VkVideoDecodeH265SessionParametersAddInfoKHR-pStdSPSs-parameter
  If `stdSPSCount` is not `0`, `pStdSPSs` **must** be a valid pointer to an array of `stdSPSCount` `StdVideoH265SequenceParameterSet` values
- VUID-VkVideoDecodeH265SessionParametersAddInfoKHR-pStdPPSs-parameter
  If `stdPPSCount` is not `0`, `pStdPPSs` **must** be a valid pointer to an array of `stdPPSCount` `StdVideoH265PictureParameterSet` values

## 42.13.6. H.265 Decoding Parameters

The `VkVideoDecodeH265PictureInfoKHR` structure is defined as:

```
// Provided by VK_KHR_video_decode_h265
typedef struct VkVideoDecodeH265PictureInfoKHR {
    VkStructureType                         sType;
    const void*                             pNext;
    const StdVideoDecodeH265PictureInfo*    pStdPictureInfo;
    uint32_t                                sliceSegmentCount;
    const uint32_t*                         pSliceSegmentOffsets;
} VkVideoDecodeH265PictureInfoKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.

- pStdPictureInfo is a pointer to a StdVideoDecodeH265PictureInfo structure specifying H.265 picture information.
- sliceSegmentCount is the number of elements in pSliceSegmentOffsets.
- pSliceSegmentOffsets is a pointer to an array of sliceSegmentCount offsets specifying the start offset of the slice segments of the picture within the video bitstream buffer range specified in VkVideoDecodeInfoKHR.

This structure is specified in the pNext chain of the VkVideoDecodeInfoKHR structure passed to vkCmdDecodeVideoKHR to specify the codec-specific picture information for an H.265 decode operation.

**Decode Output Picture Information**

When this structure is specified in the pNext chain of the VkVideoDecodeInfoKHR structure passed to vkCmdDecodeVideoKHR, the information related to the decode output picture is defined as follows:

- The image subregion used is determined according to the H.265 Decode Picture Data Access section.
- The decode output picture is associated with the H.265 picture information provided in pStdPictureInfo.

**Std Picture Information**

The members of the StdVideoDecodeH265PictureInfo structure pointed to by pStdPictureInfo are interpreted as follows:

- reserved is used only for padding purposes and is otherwise ignored;
- flags.IrapPicFlag as defined in section 3.73 of the ITU-T H.265 Specification;
- flags.IdrPicFlag as defined in section 3.67 of the ITU-T H.265 Specification;
- flags.IsReference as defined in section 3.132 of the ITU-T H.265 Specification;
- sps_video_parameter_set_id, pps_seq_parameter_set_id, and pps_pic_parameter_set_id are used to identify the active parameter sets, as described below;
- PicOrderCntVal as defined in section 8.3.1 of the ITU-T H.265 Specification;
- NumBitsForSTRefPicSetInSlice is the number of bits used in st_ref_pic_set when short_term_ref_pic_set_sps_flag is 0, or 0 otherwise, as defined in sections 7.4.7 and 7.4.8 of the ITU-T H.265 Specification;
- NumDeltaPocsOfRefRpsIdx is the value of NumDeltaPocs[RefRpsIdx] when short_term_ref_pic_set_sps_flag is 1, or 0 otherwise, as defined in sections 7.4.7 and 7.4.8 of the ITU-T H.265 Specification;
- RefPicSetStCurrBefore, RefPicSetStCurrAfter, and RefPicSetLtCurr are interpreted as defined in section 8.3.2 of the ITU-T H.265 Specification where each element of these arrays either identifies an active reference picture using its DPB slot index or contains the value STD_VIDEO_H265_NO_REFERENCE_PICTURE to indicate “no reference picture”;
- all other members are interpreted as defined in section 8.3.2 of the ITU-T H.265 Specification.

Reference picture setup is controlled by the value of StdVideoDecodeH265PictureInfo::flags.IsReference. If it is set and a reconstructed picture is specified, then the latter is used as the target of picture reconstruction to activate the corresponding DPB slot. If StdVideoDecodeH265PictureInfo::flags.IsReference is not set, but a reconstructed picture is specified, then the corresponding picture reference associated with the DPB slot is invalidated, as described in the DPB Slot States section.

**Active Parameter Sets**

The members of the StdVideoDecodeH265PictureInfo structure pointed to by pStdPictureInfo are used to select the active parameter sets to use from the bound video session parameters object, as follows:

- The *active VPS* is the VPS identified by the key specified in StdVideoDecodeH265PictureInfo::sps_video_parameter_set_id.
- The *active SPS* is the SPS identified by the key specified by the pair constructed from StdVideoDecodeH265PictureInfo::sps_video_parameter_set_id and StdVideoDecodeH265PictureInfo::pps_seq_parameter_set_id.
- The *active PPS* is the PPS identified by the key specified by the triplet constructed from StdVideoDecodeH265PictureInfo::sps_video_parameter_set_id, StdVideoDecodeH265PictureInfo::pps_seq_parameter_set_id, and StdVideoDecodeH265PictureInfo::pps_pic_parameter_set_id.

### Valid Usage (Implicit)

- VUID-VkVideoDecodeH265PictureInfoKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_VIDEO_DECODE_H265_PICTURE_INFO_KHR
- VUID-VkVideoDecodeH265PictureInfoKHR-pStdPictureInfo-parameter
  pStdPictureInfo **must** be a valid pointer to a valid StdVideoDecodeH265PictureInfo value
- VUID-VkVideoDecodeH265PictureInfoKHR-pSliceSegmentOffsets-parameter
  pSliceSegmentOffsets **must** be a valid pointer to an array of sliceSegmentCount uint32_t values
- VUID-VkVideoDecodeH265PictureInfoKHR-sliceSegmentCount-arraylength
  sliceSegmentCount **must** be greater than 0

The VkVideoDecodeH265DpbSlotInfoKHR structure is defined as:

```
// Provided by VK_KHR_video_decode_h265
typedef struct VkVideoDecodeH265DpbSlotInfoKHR {
    VkStructureType                             sType;
    const void*                                 pNext;
    const StdVideoDecodeH265ReferenceInfo*      pStdReferenceInfo;
} VkVideoDecodeH265DpbSlotInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.

- pStdReferenceInfo is a pointer to a StdVideoDecodeH265ReferenceInfo structure specifying reference picture information described in section 8.3 of the ITU-T H.265 Specification.

This structure is specified in the pNext chain of VkVideoDecodeInfoKHR::pSetupReferenceSlot, if not NULL, and the pNext chain of the elements of VkVideoDecodeInfoKHR::pReferenceSlots to specify the codec-specific reference picture information for an H.265 decode operation.

**Active Reference Picture Information**

When this structure is specified in the pNext chain of the elements of VkVideoDecodeInfoKHR::pReferenceSlots, one element is added to the list of active reference pictures used by the video decode operation for each element of VkVideoDecodeInfoKHR::pReferenceSlots as follows:

- The image subregion used is determined according to the H.265 Decode Picture Data Access section.
- The reference picture is associated with the DPB slot index specified in the slotIndex member of the corresponding element of VkVideoDecodeInfoKHR::pReferenceSlots.
- The reference picture is associated with the H.265 reference information provided in pStdReferenceInfo.

**Reconstructed Picture Information**

When this structure is specified in the pNext chain of VkVideoDecodeInfoKHR::pSetupReferenceSlot, the information related to the reconstructed picture is defined as follows:

- The image subregion used is determined according to the H.265 Decode Picture Data Access section.
- If reference picture setup is requested, then the reconstructed picture is used to activate the DPB slot with the index specified in VkVideoDecodeInfoKHR::pSetupReferenceSlot->slotIndex.
- The reconstructed picture is associated with the H.265 reference information provided in pStdReferenceInfo.

**Std Reference Information**

The members of the StdVideoDecodeH265ReferenceInfo structure pointed to by pStdReferenceInfo are interpreted as follows:

- flags.used_for_long_term_reference is used to indicate whether the picture is marked as “used for long-term reference” as defined in section 8.3.2 of the ITU-T H.265 Specification;
- flags.unused_for_reference is used to indicate whether the picture is marked as “unused for reference” as defined in section 8.3.2 of the ITU-T H.265 Specification;
- all other members are interpreted as defined in section 8.3 of the ITU-T H.265 Specification.

## Valid Usage (Implicit)

- VUID-VkVideoDecodeH265DpbSlotInfoKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_VIDEO_DECODE_H265_DPB_SLOT_INFO_KHR
- VUID-VkVideoDecodeH265DpbSlotInfoKHR-pStdReferenceInfo-parameter

pStdReferenceInfo **must** be a valid pointer to a valid StdVideoDecodeH265ReferenceInfo value

### 42.13.7. H.265 Decode Requirements

This section describes the **required** H.265 decoding capabilities for physical devices that have at least one queue family that supports the video codec operation VK_VIDEO_CODEC_OPERATION_DECODE_H265_BIT_KHR, as returned by vkGetPhysicalDeviceQueueFamilyProperties2 in VkQueueFamilyVideoPropertiesKHR::videoCodecOperations.

*Table 53. Required Video Std Header Versions*

| Video Std Header Name | Version |
|---|---|
| vulkan_video_codec_h265std_decode | 1.0.0 |

*Table 54. Required Video Capabilities*

| Video Capability | Requirement | Requirement Type[1] |
|---|---|---|
| **VkVideoCapabilitiesKHR** | | |
| flags | - | min |
| minBitstreamBufferOffsetAlignment | 4096 | max |
| minBitstreamBufferSizeAlignment | 4096 | max |
| pictureAccessGranularity | (64,64) | max |
| minCodedExtent | - | max |
| maxCodedExtent | - | min |
| maxDpbSlots | 0 | min |
| maxActiveReferencePictures | 0 | min |
| **VkVideoDecodeCapabilitiesKHR** | | |
| flags | VK_VIDEO_DECODE_CAPABILITY_DPB_AND_OUTPUT_COINCIDE_BIT_KHR or VK_VIDEO_DECODE_CAPABILITY_DPB_AND_OUTPUT_DISTINCT_BIT_KHR | min |
| **VkVideoDecodeH265CapabilitiesKHR** | | |
| maxLevelIdc | STD_VIDEO_H265_LEVEL_IDC_1_0 | min |

**1**

The **Requirement Type** column specifies the requirement is either the minimum value all implementations **must** support, the maximum value all implementations **must** support, or the exact value all implementations **must** support. For bitmasks a minimum value is the least bits all

implementations **must** set, but they **may** have additional bits set beyond this minimum.

## 42.14. AV1 Decode Operations

Video decode operations using an AV1 decode profile **can** be used to decode elementary video stream sequences compliant with the AV1 Specification.

| | |
|---|---|
| **NOTE** | Refer to the Preamble for information on how the Khronos Intellectual Property Rights Policy relates to normative references to external materials not created by Khronos. |

This process is performed according to the video decode operation steps with the codec-specific semantics defined in section 7 of the AV1 Specification:

- Syntax elements, derived values, and other parameters are applied from the following structures:
  - The `StdVideoAV1SequenceHeader` structure stored in the bound video session parameters object specifying the active sequence header.
  - The `StdVideoDecodeAV1PictureInfo` structure specifying the AV1 picture information.
  - The `StdVideoDecodeAV1ReferenceInfo` structures specifying the AV1 reference information corresponding to the optional reconstructed picture and any active reference pictures.
- The contents of the provided video bitstream buffer range are interpreted as defined in the AV1 Decode Bitstream Data Access section.
- Picture data in the video picture resources corresponding to the used active reference pictures, decode output picture, and optional reconstructed picture is accessed as defined in the AV1 Decode Picture Data Access section.
- The decision on reference picture setup is made according to the parameters specified in the AV1 picture information.

If the parameters and the bitstream adhere to the syntactic and semantic requirements defined in the corresponding sections of the AV1 Specification, as described above, and the DPB slots associated with the active reference pictures all refer to valid picture references, then the video decode operation will complete successfully. Otherwise, the video decode operation **may** complete unsuccessfully.

### 42.14.1. AV1 Decode Bitstream Data Access

The video bitstream buffer range **should** contain one or more frame OBUs, comprised of a frame header OBU and tile group OBU, that together represent an entire frame, as defined in sections 5.10, 5.9, and 5.11, and this data is interpreted as defined in sections 6.9, 6.8, and 6.10 of the AV1 Specification, respectively.

The offset specified in VkVideoDecodeAV1PictureInfoKHR::`frameHeaderOffset` **should** specify the starting offset of the frame header OBU of the frame.

| | |
|---|---|
| **NOTE** | When the tiles of the frame are encoded into multiple tile groups, each frame OBU |

> has a separate frame header OBU but their content is expected to match per the requirements of the AV1 Specification. Accordingly, the offset specified in `frameHeaderOffset` **can** be the offset of any of the otherwise identical frame header OBUs when multiple tile groups are present.

The offsets and sizes provided in VkVideoDecodeAV1PictureInfoKHR::`pTileOffsets` and VkVideoDecodeAV1PictureInfoKHR::`pTileSizes`, respectively, **should** specify the starting offsets and sizes corresponding to each tile within the video bitstream buffer range.

### 42.14.2. AV1 Decode Picture Data Access

Accesses to image data within a video picture resource happen at the granularity indicated by VkVideoCapabilitiesKHR::`pictureAccessGranularity`, as returned by vkGetPhysicalDeviceVideoCapabilitiesKHR for the used video profile. Accordingly, the complete image subregion of a decode output picture, reference picture, or reconstructed picture accessed by video coding operations using an AV1 decode profile is defined as the set of texels within the coordinate range:

([0,`endX`), [0,`endY`))

Where:

- `endX` equals `codedExtent.width` rounded up to the nearest integer multiple of `pictureAccessGranularity.width` and clamped to the width of the image subresource referred to by the corresponding VkVideoPictureResourceInfoKHR structure;
- endY equals `codedExtent.height` rounded up to the nearest integer multiple of `pictureAccessGranularity.height` and clamped to the height of the image subresource referred to by the corresponding VkVideoPictureResourceInfoKHR structure;

Where `codedExtent` is the member of the VkVideoPictureResourceInfoKHR structure corresponding to the picture.

In case of video decode operations using an AV1 decode profile, any access to a picture at the coordinates (`x`,`y`), as defined by the AV1 Specification, is an access to the image subresource referred to by the corresponding VkVideoPictureResourceInfoKHR structure at the texel coordinates (`x`,`y`).

### 42.14.3. AV1 Reference Names and Semantics

Individual reference frames used in the decoding process have different semantics, as defined in section 6.10.24 of the AV1 Specification. The AV1 semantics associated with a reference picture are indicated by the corresponding enumeration constant defined in the Video Std enumeration type `StdVideoAV1ReferenceName`:

- `STD_VIDEO_AV1_REFERENCE_NAME_INTRA_FRAME` identifies the reference used for intra coding (`INTRA_FRAME`), as defined in sections 2 and 7.11.2 of the AV1 Specification.
- All other enumeration constants refer to backward or forward references used for inter coding, as defined in sections 2 and 7.11.3 of the AV1 Specification:

- STD_VIDEO_AV1_REFERENCE_NAME_LAST_FRAME identifies the LAST_FRAME reference
- STD_VIDEO_AV1_REFERENCE_NAME_LAST2_FRAME identifies the LAST2_FRAME reference
- STD_VIDEO_AV1_REFERENCE_NAME_LAST3_FRAME identifies the LAST3_FRAME reference
- STD_VIDEO_AV1_REFERENCE_NAME_GOLDEN_FRAME identifies the GOLDEN_FRAME reference
- STD_VIDEO_AV1_REFERENCE_NAME_BWDREF_FRAME identifies the BWDREF_FRAME reference
- STD_VIDEO_AV1_REFERENCE_NAME_ALTREF2_FRAME identifies the ALTREF2_FRAME reference
- STD_VIDEO_AV1_REFERENCE_NAME_ALTREF_FRAME identifies the ALTREF_FRAME reference

These enumeration constants are not directly used in any APIs but are used to indirectly index into certain Video Std and Vulkan API parameter arrays.

### 42.14.4. AV1 Decode Profile

A video profile supporting AV1 video decode operations is specified by setting VkVideoProfileInfoKHR::videoCodecOperation to VK_VIDEO_CODEC_OPERATION_DECODE_AV1_BIT_KHR and adding a VkVideoDecodeAV1ProfileInfoKHR structure to the VkVideoProfileInfoKHR::pNext chain.

The VkVideoDecodeAV1ProfileInfoKHR structure is defined as:

```
// Provided by VK_KHR_video_decode_av1
typedef struct VkVideoDecodeAV1ProfileInfoKHR {
    VkStructureType       sType;
    const void*           pNext;
    StdVideoAV1Profile    stdProfile;
    VkBool32              filmGrainSupport;
} VkVideoDecodeAV1ProfileInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- stdProfile is a StdVideoAV1Profile value specifying the AV1 codec profile, as defined in section A.2 of the AV1 Specification.
- filmGrainSupport specifies whether AV1 film grain, as defined in section 7.8.3 of the AV1 Specification, **can** be used with the video profile. When this member is set to VK_TRUE, video session objects created against the video profile will be able to decode pictures that have film grain enabled.

| | |
|---|---|
| **NOTE** | Enabling filmGrainSupport **may** increase the memory requirements of video sessions and/or video picture resources on some implementations. |

**Valid Usage (Implicit)**

- VUID-VkVideoDecodeAV1ProfileInfoKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_VIDEO_DECODE_AV1_PROFILE_INFO_KHR

### 42.14.5. AV1 Decode Capabilities

When calling vkGetPhysicalDeviceVideoCapabilitiesKHR to query the capabilities for an AV1 decode profile, the VkVideoCapabilitiesKHR::pNext chain **must** include a VkVideoDecodeAV1CapabilitiesKHR structure that will be filled with the profile-specific capabilities.

The VkVideoDecodeAV1CapabilitiesKHR structure is defined as:

```
// Provided by VK_KHR_video_decode_av1
typedef struct VkVideoDecodeAV1CapabilitiesKHR {
    VkStructureType     sType;
    void*               pNext;
    StdVideoAV1Level    maxLevel;
} VkVideoDecodeAV1CapabilitiesKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- maxLevel is a StdVideoAV1Level value specifying the maximum AV1 level supported by the profile, as defined in section A.3 of the AV1 Specification.

**Valid Usage (Implicit)**

- VUID-VkVideoDecodeAV1CapabilitiesKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_VIDEO_DECODE_AV1_CAPABILITIES_KHR

### 42.14.6. AV1 Decode Parameter Sets

Video session parameters objects created with the video codec operation VK_VIDEO_CODEC_OPERATION_DECODE_AV1_BIT_KHR contain a single instance of the following parameter set:

**AV1 Sequence Header**

Represented by StdVideoAV1SequenceHeader structures and interpreted as follows:

- flags.reserved and reserved1 are used only for padding purposes and are otherwise ignored;
- the StdVideoAV1ColorConfig structure pointed to by pColorConfig is interpreted as follows:
  - flags.reserved and reserved1 are used only for padding purposes and are otherwise ignored;
  - all other members of StdVideoAV1ColorConfig are interpreted as defined in section 6.4.2 of the AV1 Specification;
- if flags.timing_info_present_flag is set, then the StdVideoAV1TimingInfo structure pointed to by pTimingInfo is interpreted as follows:

    - flags.reserved is used only for padding purposes and is otherwise ignored;
    - all other members of StdVideoAV1TimingInfo are interpreted as defined in section 6.4.3 of the AV1 Specification;
- all other members of StdVideoAV1SequenceHeader are interpreted as defined in section 6.4 of the AV1 Specification.

When a video session parameters object is created with the codec operation VK_VIDEO_CODEC_OPERATION_DECODE_AV1_BIT_KHR, the VkVideoSessionParametersCreateInfoKHR::pNext chain **must** include a VkVideoDecodeAV1SessionParametersCreateInfoKHR structure specifying the contents of the object.

The VkVideoDecodeAV1SessionParametersCreateInfoKHR structure is defined as:

```
// Provided by VK_KHR_video_decode_av1
typedef struct VkVideoDecodeAV1SessionParametersCreateInfoKHR {
    VkStructureType                     sType;
    const void*                         pNext;
    const StdVideoAV1SequenceHeader*    pStdSequenceHeader;
} VkVideoDecodeAV1SessionParametersCreateInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- pStdSequenceHeader is a pointer to a StdVideoAV1SequenceHeader structure describing the AV1 sequence header entry to store in the created object.

**NOTE** As AV1 video session parameters objects will only ever contain a single AV1 sequence header, this has to be specified at object creation time and such video session parameters objects cannot be updated using the vkUpdateVideoSessionParametersKHR command. When a new AV1 sequence header is decoded from the input video bitstream the application needs to create a new video session parameters object to store it.

### Valid Usage (Implicit)

- VUID-VkVideoDecodeAV1SessionParametersCreateInfoKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_VIDEO_DECODE_AV1_SESSION_PARAMETERS_CREATE_INFO_KHR
- VUID-VkVideoDecodeAV1SessionParametersCreateInfoKHR-pStdSequenceHeader-parameter
  pStdSequenceHeader **must** be a valid pointer to a valid StdVideoAV1SequenceHeader value

## 42.14.7. AV1 Decoding Parameters

The VkVideoDecodeAV1PictureInfoKHR structure is defined as:

```
// Provided by VK_KHR_video_decode_av1
typedef struct VkVideoDecodeAV1PictureInfoKHR {
    VkStructureType                          sType;
    const void*                              pNext;
    const StdVideoDecodeAV1PictureInfo*      pStdPictureInfo;
    int32_t                                  referenceNameSlotIndices
[VK_MAX_VIDEO_AV1_REFERENCES_PER_FRAME_KHR];
    uint32_t                                 frameHeaderOffset;
    uint32_t                                 tileCount;
    const uint32_t*                          pTileOffsets;
    const uint32_t*                          pTileSizes;
} VkVideoDecodeAV1PictureInfoKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `pStdPictureInfo` is a pointer to a `StdVideoDecodeAV1PictureInfo` structure specifying AV1 picture information.
- `referenceNameSlotIndices` is an array of seven (`VK_MAX_VIDEO_AV1_REFERENCES_PER_FRAME_KHR`, which is equal to the Video Std definition `STD_VIDEO_AV1_REFS_PER_FRAME`) signed integer values specifying the index of the DPB slot or a negative integer value for each AV1 reference name used for inter coding. In particular, the DPB slot index for the AV1 reference name `frame` is specified in `referenceNameSlotIndices`[`frame` - `STD_VIDEO_AV1_REFERENCE_NAME_LAST_FRAME`].
- `frameHeaderOffset` is the byte offset of the AV1 frame header OBU, as defined in section 5.9 of the AV1 Specification, within the video bitstream buffer range specified in VkVideoDecodeInfoKHR.
- `tileCount` is the number of elements in `pTileOffsets` and `pTileSizes`.
- `pTileOffsets` is a pointer to an array of `tileCount` integers specifying the byte offset of the tiles of the picture within the video bitstream buffer range specified in VkVideoDecodeInfoKHR.
- `pTileSizes` is a pointer to an array of `tileCount` integers specifying the byte size of the tiles of the picture within the video bitstream buffer range specified in VkVideoDecodeInfoKHR.

This structure is specified in the `pNext` chain of the VkVideoDecodeInfoKHR structure passed to vkCmdDecodeVideoKHR to specify the codec-specific picture information for an AV1 decode operation.

**Decode Output Picture Information**

When this structure is specified in the `pNext` chain of the VkVideoDecodeInfoKHR structure passed to vkCmdDecodeVideoKHR, the information related to the decode output picture is defined as follows:

- The image subregion used is determined according to the AV1 Decode Picture Data Access section.
- The decode output picture is associated with the AV1 picture information provided in `pStdPictureInfo`.

**Std Picture Information**

The members of the `StdVideoDecodeAV1PictureInfo` structure pointed to by `pStdPictureInfo` are interpreted as follows:

- `flags.reserved`, `reserved1`, and `reserved2` are used only for padding purposes and are otherwise ignored;
- `flags.apply_grain` indicates that film grain is enabled for the decoded picture, as defined in section 6.8.20 of the AV1 Specification;
- `tg_start` and `tg_end` are interpreted as defined in section 6.10.1 of the AV1 Specification;
- `OrderHint`, `OrderHints`, and `expectedFrameId` are interpreted as defined in section 6.8.2 of the AV1 Specification;
- the `StdVideoAV1TileInfo` structure pointed to by `pTileInfo` is interpreted as follows:
  - `flags.reserved` and `reserved1` are used only for padding purposes and are otherwise ignored;
  - `pMiColStarts` is a pointer to an array of `TileCols` number of unsigned integers that corresponds to `MiColStarts` defined in section 6.8.14 of the AV1 Specification;
  - `pMiRowStarts` is a pointer to an array of `TileRows` number of unsigned integers that corresponds to `MiRowStarts` defined in section 6.8.14 of the AV1 Specification;
  - `pWidthInSbsMinus1` is a pointer to an array of `TileCols` number of unsigned integers that corresponds to `width_in_sbs_minus_1` defined in section 6.8.14 of the AV1 Specification;
  - `pHeightInSbsMinus1` is a pointer to an array of `TileRows` number of unsigned integers that corresponds to `height_in_sbs_minus_1` defined in section 6.8.14 of the AV1 Specification;
  - all other members of `StdVideoAV1TileInfo` are interpreted as defined in section 6.8.14 of the AV1 Specification;
- the `StdVideoAV1Quantization` structure pointed to by `pQuantization` is interpreted as follows:
  - `flags.reserved` is used only for padding purposes and is otherwise ignored;
  - all other members of `StdVideoAV1Quantization` are interpreted as defined in section 6.8.11 of the AV1 Specification;
- if `flags.segmentation_enabled` is set, then the `StdVideoAV1Segmentation` structure pointed to by `pSegmentation` is interpreted as follows:
  - the elements of `FeatureEnabled` are bitmasks where bit index j of element i corresponds to `FeatureEnabled[i][j]` as defined in section 6.8.13 of the AV1 Specification;
  - `FeatureData` is interpreted as defined in section 6.8.13 of the AV1 Specification;
- the `StdVideoAV1LoopFilter` structure pointed to by `pLoopFilter` is interpreted as follows:
  - `flags.reserved` is used only for padding purposes and is otherwise ignored;
  - `update_ref_delta` is a bitmask where bit index i is interpreted as the value of `update_ref_delta` corresponding to element i of `loop_filter_ref_deltas` as defined in section 6.8.10 of the AV1 Specification;
  - `update_mode_delta` is a bitmask where bit index i is interpreted as the value of `update_mode_delta` corresponding to element i of `loop_filter_mode_deltas` as defined in

section 6.8.10 of the AV1 Specification;

  - all other members of StdVideoAV1LoopFilter are interpreted as defined in section 6.8.10 of the AV1 Specification;

- if flags.enable_cdef is set in the active sequence header, then the members of the StdVideoAV1CDEF structure pointed to by pCDEF are interpreted as follows:
  - cdef_y_sec_strength and cdef_uv_sec_strength are the bitstream values of the corresponding syntax elements defined in section 5.9.19 of the AV1 Specification;
  - all other members of StdVideoAV1CDEF are interpreted as defined in section 6.10.14 of the AV1 Specification;
- the StdVideoAV1LoopRestoration structure pointed to by pLoopRestoration is interpreted as follows:
  - LoopRestorationSize[plane] is interpreted as log2(size) - 5, where size is the value of LoopRestorationSize[plane] as defined in section 6.10.15 of the AV1 Specification.
  - all other members of StdVideoAV1LoopRestoration are defined as in section 6.10.15 of the AV1 Specification;
- the members of the StdVideoAV1GlobalMotion structure provided in global_motion are interpreted as defined in section 7.10 of the AV1 Specification;
- if flags.film_grain_params_present is set in the active sequence header, then the StdVideoAV1FilmGrain structure pointed to by pFilmGrain is interpreted as follows:
  - flags.reserved is used only for padding purposes and is otherwise ignored;
  - all other members of StdVideoAV1FilmGrain are interpreted as defined in section 6.8.20 of the AV1 Specification;
- all other members are interpreted as defined in section 6.8 of the AV1 Specification.

When film grain is enabled for the decoded frame, the flags.update_grain and film_grain_params_ref_idx values specified in StdVideoAV1FilmGrain are ignored by AV1 decode operations and the load_grain_params function, as defined in section 6.8.20 of the AV1 Specification, is not executed. Instead, the application is responsible for specifying the effective film grain parameters for the frame in StdVideoAV1FilmGrain.

When film grain is enabled for the decoded frame, the application is required to specify a different decode output picture resource in VkVideoDecodeInfoKHR::dstPictureResource compared to the reconstructed picture specified in VkVideoDecodeInfoKHR::pSetupReferenceSlot->pPictureResource even if the implementation does not report support for VK_VIDEO_DECODE_CAPABILITY_DPB_AND_OUTPUT_DISTINCT_BIT_KHR in VkVideoDecodeCapabilitiesKHR::flags for the video decode profile.

Reference picture setup is controlled by the value of StdVideoDecodeAV1PictureInfo::refresh_frame_flags. If it is not zero and a reconstructed picture is specified, then the latter is used as the target of picture reconstruction to activate the DPB slot specified in pDecodeInfo->pSetupReferenceSlot->slotIndex. If StdVideoDecodeAV1PictureInfo::refresh_frame_flags is zero, but a reconstructed picture is specified, then the corresponding picture reference associated with the DPB slot is invalidated, as described in the DPB Slot States section.

**Active Parameter Sets**

The *active sequence header* is the AV1 sequence header stored in the bound video session parameters object.

### Valid Usage (Implicit)

- VUID-VkVideoDecodeAV1PictureInfoKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_VIDEO_DECODE_AV1_PICTURE_INFO_KHR`
- VUID-VkVideoDecodeAV1PictureInfoKHR-pStdPictureInfo-parameter
  `pStdPictureInfo` **must** be a valid pointer to a valid `StdVideoDecodeAV1PictureInfo` value
- VUID-VkVideoDecodeAV1PictureInfoKHR-pTileOffsets-parameter
  `pTileOffsets` **must** be a valid pointer to an array of `tileCount` `uint32_t` values
- VUID-VkVideoDecodeAV1PictureInfoKHR-pTileSizes-parameter
  `pTileSizes` **must** be a valid pointer to an array of `tileCount` `uint32_t` values
- VUID-VkVideoDecodeAV1PictureInfoKHR-tileCount-arraylength
  `tileCount` **must** be greater than `0`

The `VkVideoDecodeAV1DpbSlotInfoKHR` structure is defined as:

```
// Provided by VK_KHR_video_decode_av1
typedef struct VkVideoDecodeAV1DpbSlotInfoKHR {
    VkStructureType                          sType;
    const void*                              pNext;
    const StdVideoDecodeAV1ReferenceInfo*    pStdReferenceInfo;
} VkVideoDecodeAV1DpbSlotInfoKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `pStdReferenceInfo` is a pointer to a `StdVideoDecodeAV1ReferenceInfo` structure specifying AV1 reference information.

This structure is specified in the `pNext` chain of VkVideoDecodeInfoKHR::`pSetupReferenceSlot`, if not `NULL`, and the `pNext` chain of the elements of VkVideoDecodeInfoKHR::`pReferenceSlots` to specify the codec-specific reference picture information for an AV1 decode operation.

**Active Reference Picture Information**

When this structure is specified in the `pNext` chain of the elements of VkVideoDecodeInfoKHR::`pReferenceSlots`, one element is added to the list of active reference pictures used by the video decode operation for each element of VkVideoDecodeInfoKHR::`pReferenceSlots` as follows:

- The image subregion used is determined according to the AV1 Decode Picture Data Access section.
- The reference picture is associated with the DPB slot index specified in the `slotIndex` member of the corresponding element of VkVideoDecodeInfoKHR::`pReferenceSlots`.

- The reference picture is associated with the AV1 reference information provided in pStdReferenceInfo.

**Reconstructed Picture Information**

When this structure is specified in the pNext chain of VkVideoDecodeInfoKHR::pSetupReferenceSlot, the information related to the reconstructed picture is defined as follows:

- The image subregion used is determined according to the AV1 Decode Picture Data Access section.
- If reference picture setup is requested, then the reconstructed picture is used to activate the DPB slot with the index specified in VkVideoDecodeInfoKHR::pSetupReferenceSlot->slotIndex.
- The reconstructed picture is associated with the AV1 reference information provided in pStdReferenceInfo.

**Std Reference Information**

The members of the StdVideoDecodeAV1ReferenceInfo structure pointed to by pStdReferenceInfo are interpreted as follows:

- flags.reserved and reserved1 are used only for padding purposes and are otherwise ignored;
- flags.disable_frame_end_update_cdf is interpreted as defined in section 6.8.2 of the AV1 Specification;
- flags.segmentation_enabled is interpreted as defined in section 6.8.13 of the AV1 Specification;
- frame_type is interpreted as defined in section 6.8.2 of the AV1 Specification;

> **NOTE** The frame_type member is defined with the type uint8_t, but it takes the same values defined in the StdVideoAV1FrameType enumeration type as StdVideoDecodeAV1PictureInfo::frame_type.

- RefFrameSignBias is a bitmask where bit index i corresponds to RefFrameSignBias[i] as defined in section 6.8.2 of the AV1 Specification;
- OrderHint is interpreted as defined in section 6.8.2 of the AV1 Specification;
- SavedOrderHints is interpreted as defined in section 7.20 of the AV1 Specification.

> **NOTE** When the AV1 reference information is provided for the reconstructed picture, certain parameters (e.g. frame_type) are specified both in the AV1 picture information and in the AV1 reference information. This is necessary because unlike the AV1 picture information, which is only used for the purposes of the video decode operation in question, the AV1 reference information specified for the reconstructed picture **may** be associated with the activated DPB slot, meaning that some implementations **may** maintain it as part of the reference picture metadata corresponding to the video picture resource associated with the DPB slot. When the AV1 reference information is provided for an active reference picture, the specified parameters

correspond to the parameters specified when the DPB slot was activated (set up) with the reference picture, as usual, in order to communicate these parameters for implementations that do not maintain any subset of these parameters as part of the DPB slot's reference picture metadata.

### Valid Usage (Implicit)

- VUID-VkVideoDecodeAV1DpbSlotInfoKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_VIDEO_DECODE_AV1_DPB_SLOT_INFO_KHR`
- VUID-VkVideoDecodeAV1DpbSlotInfoKHR-pStdReferenceInfo-parameter
  `pStdReferenceInfo` **must** be a valid pointer to a valid `StdVideoDecodeAV1ReferenceInfo` value

## 42.14.8. AV1 Decode Requirements

This section describes the **required** AV1 decoding capabilities for physical devices that have at least one queue family that supports the video codec operation `VK_VIDEO_CODEC_OPERATION_DECODE_AV1_BIT_KHR`, as returned by vkGetPhysicalDeviceQueueFamilyProperties2 in VkQueueFamilyVideoPropertiesKHR::`videoCodecOperations`.

*Table 55. Required Video Std Header Versions*

| Video Std Header Name | Version |
|---|---|
| `vulkan_video_codec_av1std_decode` | 1.0.0 |

*Table 56. Required Video Capabilities*

| Video Capability | Requirement | Requirement Type[1] |
|---|---|---|
| **VkVideoCapabilitiesKHR** | | |
| `flags` | - | min |
| `minBitstreamBufferOffsetAlignment` | 4096 | max |
| `minBitstreamBufferSizeAlignment` | 4096 | max |
| `pictureAccessGranularity` | (64,64) | max |
| `minCodedExtent` | - | max |
| `maxCodedExtent` | - | min |
| `maxDpbSlots` | 0 | min |
| `maxActiveReferencePictures` | 0 | min |
| **VkVideoDecodeCapabilitiesKHR** | | |

| Video Capability | Requirement | Requirement Type[1] |
|---|---|---|
| flags | VK_VIDEO_DECODE_CAPABILITY_DPB_AND_OUTPUT_COINCIDE_BIT_KHR or VK_VIDEO_DECODE_CAPABILITY_DPB_AND_OUTPUT_DISTINCT_BIT_KHR | min |
| **VkVideoDecodeAV1CapabilitiesKHR** | | |
| maxLevel | STD_VIDEO_AV1_LEVEL_2_0 | min |

**1**

The **Requirement Type** column specifies the requirement is either the minimum value all implementations **must** support, the maximum value all implementations **must** support, or the exact value all implementations **must** support. For bitmasks a minimum value is the least bits all implementations **must** set, but they **may** have additional bits set beyond this minimum.

# 42.15. Video Encode Operations

Video encode operations consume an *encode input picture* and zero or more reference pictures, and produce compressed video data to a video bitstream buffer and an optional reconstructed picture.

| | |
|---|---|
| **NOTE** | Such encode input pictures can be used as the output of video decode operations, with graphics or compute operations, or with Window System Integration APIs, depending on the capabilities of the implementation. |

Video encode operations **may** access the following resources in the VK_PIPELINE_STAGE_2_VIDEO_ENCODE_BIT_KHR stage:

- The image subregions corresponding to the source encode input picture and active reference pictures with access VK_ACCESS_2_VIDEO_ENCODE_READ_BIT_KHR.
- The destination video bitstream buffer range and the optional reconstructed picture with access VK_ACCESS_2_VIDEO_ENCODE_WRITE_BIT_KHR.

The image subresource of each video picture resource accessed by the video encode operation is specified using a corresponding VkVideoPictureResourceInfoKHR structure. Each such image subresource **must** be in the appropriate image layout as follows:

- If the image subresource is used in the video encode operation as an encode input picture, then it **must** be in the VK_IMAGE_LAYOUT_VIDEO_ENCODE_SRC_KHR layout.
- If the image subresource is used in the video encode operation as a reconstructed picture or reference picture, then it **must** be in the VK_IMAGE_LAYOUT_VIDEO_ENCODE_DPB_KHR layout.

A video encode operation **may** complete unsuccessfully. In this case the target video bitstream buffer will have undefined contents. Similarly, if reference picture setup is requested, the reconstructed-picture will also have undefined contents, and the activated DPB slot will have an

invalid picture reference.

If a video encode operation completes successfully and the codec-specific parameters provided by the application adhere to the syntactic and semantic requirements defined in the corresponding video compression standard, then the target video bitstream buffer will contain compressed video data after the execution of the video encode operation according to the respective codec-specific semantics.

### 42.15.1. Codec-Specific Semantics

The following aspects of video encode operations are codec-specific:

- The compressed video data written to the target video bitstream buffer range.
- The construction and interpretation of the list of active reference pictures and the interpretation of the picture data referred to by the corresponding image subregions.
- The construction and interpretation of information related to the encode input picture and the interpretation of the picture data referred to by the corresponding image subregion.
- The decision on reference picture setup.
- The construction and interpretation of information related to the optional reconstructed picture and the generation of picture data to the corresponding image subregion.
- Certain aspects of rate control.

These codec-specific behaviors are defined for each video codec operation separately.

- If the used video codec operation is `VK_VIDEO_CODEC_OPERATION_ENCODE_H264_BIT_KHR`, then the codec-specific aspects of the video encoding process are performed as defined in the H.264 Encode Operations section.
- If the used video codec operation is `VK_VIDEO_CODEC_OPERATION_ENCODE_H265_BIT_KHR`, then the codec-specific aspects of the video encoding process are performed as defined in the H.265 Encode Operations section.

#### Video Encode Parameter Overrides

Implementations supporting video encode operations for any particular video codec operation often support only a subset of the available encoding tools defined by the corresponding video compression standards. Accordingly, certain implementation-dependent limitations **may** apply to codec-specific parameters provided through the structures defined in the Video Std headers corresponding to the used video codec operation.

Exposing all of these restrictions on particular codec-specific parameter values or combinations thereof in the form of application-queryable capabilities is impractical, hence this specification allows implementations to *override* the value of any of the codec-specific parameters, unless otherwise specified, as long as all of the following conditions are met:

- If the application-provided codec-specific parameters adhere to the syntactic and semantic requirements and rules defined by the used video compression standard, and thus would be usable to produce a video bitstream compliant with that standard, then the codec-specific

parameters resulting from the process of implementation overrides **must** also adhere to the same requirements and rules, and any video bitstream produced using the overridden parameters **must** also be compliant.

- The overridden codec-specific parameter values **must** not have an impact on the codec-independent behaviors defined for video encode operations.
- The implementation **must** not override any codec-specific parameters specified to a command that **may** cause application-provided codec-specific parameters specified to subsequent commands to no longer adhere to the semantic requirements and rules defined by the used video compression standard, unless the implementation also overrides those parameters to adhere to any such requirements and rules.
- The overridden codec-specific parameter values **must** not have an impact on the codec-specific picture data access semantics.
- The overridden codec-specific parameter values **may** change the contents of the codec-specific bitstream elements produced by video encode operations or otherwise retrieved by the application (e.g. using the vkGetEncodedVideoSessionParametersKHR command) but **must** still adhere to the codec-specific semantics defined for that video codec operation, including, but not limited to, the number, type, and order of the encoded codec-specific bitstream elements.

Besides codec-specific parameter overrides performed for implementation-dependent reasons, applications **can** enable the implementation to apply additional optimizing overrides that **may** improve the efficiency or performance of video encoding operations. However, implementations **must** meet the conditions listed above even in case of such optimizing overrides.

| | |
|---|---|
| **NOTE** | Unless the application opts in for optimizing overrides, implementations are not expected to override any of the codec-specific parameters, except when such overrides are necessary for the correct operation of video encoder implementation due to limitations to the available encoding tools on that implementation. |

### 42.15.2. Video Encode Operation Steps

Each video encode operation performs the following steps in the `VK_PIPELINE_STAGE_2_VIDEO_ENCODE_BIT_KHR` stage:

1. Reads the input picture data from the encode input picture;
2. Determine derived encoding quality parameters according to the codec-specific semantics and the current rate control state;
3. Compresses the input picture data according to the codec-specific semantics, applying any prediction data read from the active reference pictures and rate control restrictions in the process;
4. Writes the encoded bitstream data to the destination video bitstream buffer range;
5. Performs picture reconstruction of the encoded video data according to the codec-specific semantics, applying any prediction data read from the active reference pictures in the process, if a reconstructed picture is specified and reference picture setup is requested;
6. If reference picture setup is requested, the DPB slot index specified in the reconstructed picture

information is activated with the reconstructed picture;

7. Writes the reconstructed picture data to the reconstructed picture, if one is specified, according to the codec-specific semantics.

When reconstructed picture information is provided, the specified DPB slot index is associated with the corresponding bound reference picture resource, indifferent of whether reference picture setup is requested.

### 42.15.3. Capabilities

When calling vkGetPhysicalDeviceVideoCapabilitiesKHR with pVideoProfile->videoCodecOperation specifying an encode operation, the VkVideoEncodeCapabilitiesKHR structure **must** be included in the pNext chain of the VkVideoCapabilitiesKHR structure to retrieve capabilities specific to video encoding.

The VkVideoEncodeCapabilitiesKHR structure is defined as:

```
// Provided by VK_KHR_video_encode_queue
typedef struct VkVideoEncodeCapabilitiesKHR {
    VkStructureType                         sType;
    void*                                   pNext;
    VkVideoEncodeCapabilityFlagsKHR         flags;
    VkVideoEncodeRateControlModeFlagsKHR    rateControlModes;
    uint32_t                                maxRateControlLayers;
    uint64_t                                maxBitrate;
    uint32_t                                maxQualityLevels;
    VkExtent2D                              encodeInputPictureGranularity;
    VkVideoEncodeFeedbackFlagsKHR           supportedEncodeFeedbackFlags;
} VkVideoEncodeCapabilitiesKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- flags is a bitmask of VkVideoEncodeCapabilityFlagBitsKHR describing supported encoding features.
- rateControlModes is a bitmask of VkVideoEncodeRateControlModeFlagBitsKHR indicating supported rate control modes.
- maxRateControlLayers indicates the maximum number of rate control layers supported.
- maxBitrate indicates the maximum supported bitrate.
- maxQualityLevels indicates the number of discrete video encode quality levels supported. Implementations **must** support at least one quality level.
- encodeInputPictureGranularity indicates the granularity at which encode input picture data is encoded and **may** indicate a texel granularity up to the size of the largest supported codec-specific coding block. This capability does not impose any valid usage constraints on the application, however, depending on the contents of the encode input picture, it **may** have effects on the encoded bitstream, as described in more detail below.

- supportedEncodeFeedbackFlags is a bitmask of VkVideoEncodeFeedbackFlagBitsKHR values specifying the supported flags for video encode feedback queries.

Implementations **must** include support for at least VK_VIDEO_ENCODE_FEEDBACK_BITSTREAM_BUFFER_OFFSET_BIT_KHR and VK_VIDEO_ENCODE_FEEDBACK_BITSTREAM_BYTES_WRITTEN_BIT_KHR in supportedEncodeFeedbackFlags.

encodeInputPictureGranularity provides information about the way encode input picture data is used as input to video encode operations. In particular, some implementations **may** not be able to limit the set of texels used to encode the output video bitstream to the image subregion specified in the VkVideoPictureResourceInfoKHR structure corresponding to the encode input picture (i.e. to the resolution of the image data to encode specified in its codedExtent member).

**NOTE** For example, the application requests the coded extent to be 1920x1080, but the implementation is only able to source the encode input picture data at the granularity of the codec-specific coding block size which is 16x16 pixels (or as otherwise indicated in encodeInputPictureGranularity). In this example, the content is horizontally aligned with the coding block size, but not vertically aligned with it. Thus encoding of the last row of coding blocks will be impacted by the contents of the input image at texel rows 1080 to 1087 (the latter being the next row which is vertically aligned with the coding block size, assuming a zero-based texel row index).

If codedExtent rounded up to the next integer multiple of encodeInputPictureGranularity is greater than the extent of the image subresource specified for the encode input picture, then the texel values corresponding to texel coordinates outside of the bounds of the image subresource **may** be undefined. However, implementations **should** use well-defined default values for such texels in order to maximize the encoding efficiency for the last coding block row/column, and/or to ensure consistent encoding results across repeated encoding of the same input content. Nonetheless, the values used for such texels **must** not have an effect on whether the video encode operation produces a compliant bitstream, and **must** not have any other effects on the encoded picture data beyond what **may** otherwise result from using these texel values as input to any compression algorithm, as defined in the used video compression standard.

**NOTE** While not required, it is generally a good practice for applications to make sure that the image subresource used for the encode input picture has an extent that is an integer multiple of the codec-specific coding block size (or at least encodeInputPictureGranularity) and that this padding area is filled with known values in order to improve encoding efficiency, portability, and reproducibility.

### Valid Usage (Implicit)

- VUID-VkVideoEncodeCapabilitiesKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_VIDEO_ENCODE_CAPABILITIES_KHR

Bits which **may** be set in VkVideoEncodeCapabilitiesKHR::flags, indicating the encoding tools supported, are:

```
// Provided by VK_KHR_video_encode_queue
typedef enum VkVideoEncodeCapabilityFlagBitsKHR {
    VK_VIDEO_ENCODE_CAPABILITY_PRECEDING_EXTERNALLY_ENCODED_BYTES_BIT_KHR =
0x00000001,
    VK_VIDEO_ENCODE_CAPABILITY_INSUFFICIENT_BITSTREAM_BUFFER_RANGE_DETECTION_BIT_KHR =
0x00000002,
} VkVideoEncodeCapabilityFlagBitsKHR;
```

- `VK_VIDEO_ENCODE_CAPABILITY_PRECEDING_EXTERNALLY_ENCODED_BYTES_BIT_KHR` indicates that the implementation supports the use of VkVideoEncodeInfoKHR::`precedingExternallyEncodedBytes`.
- `VK_VIDEO_ENCODE_CAPABILITY_INSUFFICIENT_BITSTREAM_BUFFER_RANGE_DETECTION_BIT_KHR` indicates that the implementation is able to detect and report when the destination video bitstream buffer range provided by the application is not sufficiently large to fit the encoded bitstream data produced by a video encode operation by reporting the `VK_QUERY_RESULT_STATUS_INSUFFICIENT_BITSTREAM_BUFFER_RANGE_KHR` query result status code.

**NOTE**

Some implementations **may** not be able to reliably detect insufficient bitstream buffer range conditions in all situations. Such implementations will not report support for the `VK_VIDEO_ENCODE_CAPABILITY_INSUFFICIENT_BITSTREAM_BUFFER_RANGE_DETECTION_BIT_KHR` encode capability flag for the video profile, but **may** still report the `VK_QUERY_RESULT_STATUS_INSUFFICIENT_BITSTREAM_BUFFER_RANGE_KHR` query result status code in certain cases. Applications **should** always check for the specific query result status code `VK_QUERY_RESULT_STATUS_INSUFFICIENT_BITSTREAM_BUFFER_RANGE_KHR` even when this encode capability flag is not supported by the implementation for the video profile in question. However, applications **must** not assume that a different negative query result status code indicating an unsuccessful completion of a video encode operation is not the result of an insufficient bitstream buffer condition unless this encode capability flag is supported.

```
// Provided by VK_KHR_video_encode_queue
typedef VkFlags VkVideoEncodeCapabilityFlagsKHR;
```

`VkVideoEncodeCapabilityFlagsKHR` is a bitmask type for setting a mask of zero or more VkVideoEncodeCapabilityFlagBitsKHR.

### 42.15.4. Video Encode Quality Levels

Implementations **can** support more than one video encode quality levels for a video encode profile, which control the number and type of implementation-specific encoding tools and algorithms utilized in the encoding process.

**NOTE**

Generally, using higher video encode quality levels **may** produce higher quality video streams at the cost of additional processing time. However, as the final quality of an encoded picture depends on the contents of the encode input picture, the

> contents of the active reference pictures, the codec-specific encode parameters, and the particular implementation-specific tools used corresponding to the individual video encode quality levels, there are no guarantees that using a higher video encode quality level will always produce a higher quality encoded picture for any given set of inputs.

To query properties for a specific video encode quality level supported by a video encode profile, call:

```
// Provided by VK_KHR_video_encode_queue
VkResult vkGetPhysicalDeviceVideoEncodeQualityLevelPropertiesKHR(
    VkPhysicalDevice                            physicalDevice,
    const VkPhysicalDeviceVideoEncodeQualityLevelInfoKHR* pQualityLevelInfo,
    VkVideoEncodeQualityLevelPropertiesKHR*     pQualityLevelProperties);
```

- `physicalDevice` is the physical device to query the video encode quality level properties for.
- `pQualityLevelInfo` is a pointer to a VkPhysicalDeviceVideoEncodeQualityLevelInfoKHR structure specifying the video encode profile and quality level to query properties for.
- `pQualityLevelProperties` is a pointer to a VkVideoEncodeQualityLevelPropertiesKHR structure in which the properties are returned.

## Valid Usage

- VUID-vkGetPhysicalDeviceVideoEncodeQualityLevelPropertiesKHR-pQualityLevelInfo-08257
  If `pQualityLevelInfo->pVideoProfile->videoCodecOperation` is `VK_VIDEO_CODEC_OPERATION_ENCODE_H264_BIT_KHR`, then the `pNext` chain of `pQualityLevelProperties` **must** include a VkVideoEncodeH264QualityLevelPropertiesKHR structure
- VUID-vkGetPhysicalDeviceVideoEncodeQualityLevelPropertiesKHR-pQualityLevelInfo-08258
  If `pQualityLevelInfo->pVideoProfile->videoCodecOperation` is `VK_VIDEO_CODEC_OPERATION_ENCODE_H265_BIT_KHR`, then the `pNext` chain of `pQualityLevelProperties` **must** include a VkVideoEncodeH265QualityLevelPropertiesKHR structure

## Valid Usage (Implicit)

- VUID-vkGetPhysicalDeviceVideoEncodeQualityLevelPropertiesKHR-physicalDevice-parameter
  `physicalDevice` **must** be a valid VkPhysicalDevice handle
- VUID-vkGetPhysicalDeviceVideoEncodeQualityLevelPropertiesKHR-pQualityLevelInfo-parameter
  `pQualityLevelInfo` **must** be a valid pointer to a valid

VkPhysicalDeviceVideoEncodeQualityLevelInfoKHR structure

- VUID-vkGetPhysicalDeviceVideoEncodeQualityLevelPropertiesKHR-pQualityLevelProperties-parameter
  pQualityLevelProperties **must** be a valid pointer to a VkVideoEncodeQualityLevelPropertiesKHR structure

## Return Codes

**Success**

- VK_SUCCESS

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY
- VK_ERROR_VIDEO_PROFILE_OPERATION_NOT_SUPPORTED_KHR
- VK_ERROR_VIDEO_PROFILE_FORMAT_NOT_SUPPORTED_KHR
- VK_ERROR_VIDEO_PICTURE_LAYOUT_NOT_SUPPORTED_KHR
- VK_ERROR_VIDEO_PROFILE_CODEC_NOT_SUPPORTED_KHR

The VkPhysicalDeviceVideoEncodeQualityLevelInfoKHR structure is defined as:

```
// Provided by VK_KHR_video_encode_queue
typedef struct VkPhysicalDeviceVideoEncodeQualityLevelInfoKHR {
    VkStructureType                 sType;
    const void*                     pNext;
    const VkVideoProfileInfoKHR*    pVideoProfile;
    uint32_t                        qualityLevel;
} VkPhysicalDeviceVideoEncodeQualityLevelInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- pVideoProfile is a pointer to a VkVideoProfileInfoKHR structure specifying the video profile to query the video encode quality level properties for.
- qualityLevel is the video encode quality level to query properties for.

## Valid Usage

- VUID-VkPhysicalDeviceVideoEncodeQualityLevelInfoKHR-pVideoProfile-08259
  pVideoProfile **must** be a supported video profile
- VUID-VkPhysicalDeviceVideoEncodeQualityLevelInfoKHR-pVideoProfile-08260
  pVideoProfile->videoCodecOperation **must** specify an encode operation

- VUID-VkPhysicalDeviceVideoEncodeQualityLevelInfoKHR-qualityLevel-08261
  `qualityLevel` **must** be less than VkVideoEncodeCapabilitiesKHR::`maxQualityLevels`, as returned by vkGetPhysicalDeviceVideoCapabilitiesKHR for the video profile specified in `pVideoProfile`

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceVideoEncodeQualityLevelInfoKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_VIDEO_ENCODE_QUALITY_LEVEL_INFO_KHR`
- VUID-VkPhysicalDeviceVideoEncodeQualityLevelInfoKHR-pNext-pNext
  `pNext` **must** be `NULL`
- VUID-VkPhysicalDeviceVideoEncodeQualityLevelInfoKHR-pVideoProfile-parameter
  `pVideoProfile` **must** be a valid pointer to a valid VkVideoProfileInfoKHR structure

The `VkVideoEncodeQualityLevelPropertiesKHR` structure is defined as:

```
// Provided by VK_KHR_video_encode_queue
typedef struct VkVideoEncodeQualityLevelPropertiesKHR {
    VkStructureType                             sType;
    void*                                       pNext;
    VkVideoEncodeRateControlModeFlagBitsKHR     preferredRateControlMode;
    uint32_t                                    preferredRateControlLayerCount;
} VkVideoEncodeQualityLevelPropertiesKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `preferredRateControlMode` is a VkVideoEncodeRateControlModeFlagBitsKHR value indicating the preferred rate control mode to use with the video encode quality level.
- `preferredRateControlLayerCount` indicates the preferred number of rate control layers to use with the video encode quality level.

### Valid Usage (Implicit)

- VUID-VkVideoEncodeQualityLevelPropertiesKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_VIDEO_ENCODE_QUALITY_LEVEL_PROPERTIES_KHR`
- VUID-VkVideoEncodeQualityLevelPropertiesKHR-pNext-pNext
  Each `pNext` member of any structure (including this one) in the `pNext` chain **must** be either `NULL` or a pointer to a valid instance of VkVideoEncodeH264QualityLevelPropertiesKHR or VkVideoEncodeH265QualityLevelPropertiesKHR
- VUID-VkVideoEncodeQualityLevelPropertiesKHR-sType-unique
  The `sType` value of each struct in the `pNext` chain **must** be unique

The VkVideoEncodeQualityLevelInfoKHR structure is defined as:

```
// Provided by VK_KHR_video_encode_queue
typedef struct VkVideoEncodeQualityLevelInfoKHR {
    VkStructureType    sType;
    const void*        pNext;
    uint32_t           qualityLevel;
} VkVideoEncodeQualityLevelInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- qualityLevel is the used video encode quality level.

This structure **can** be specified in the following places:

- In the pNext chain of VkVideoSessionParametersCreateInfoKHR to specify the video encode quality level to use for a video session parameters object created for a video encode session. If no instance of this structure is included in the pNext chain of VkVideoSessionParametersCreateInfoKHR, then the video session parameters object is created with a video encode quality level of zero.
- In the pNext chain of VkVideoCodingControlInfoKHR to change the video encode quality level state of the bound video session.

### Valid Usage

- VUID-VkVideoEncodeQualityLevelInfoKHR-qualityLevel-08311
  qualityLevel **must** be less than VkVideoEncodeCapabilitiesKHR::maxQualityLevels, as returned by vkGetPhysicalDeviceVideoCapabilitiesKHR for the used video profile

### Valid Usage (Implicit)

- VUID-VkVideoEncodeQualityLevelInfoKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_VIDEO_ENCODE_QUALITY_LEVEL_INFO_KHR

## 42.15.5. Retrieving Encoded Session Parameters

Any codec-specific parameters stored in video session parameters objects **may** need to be separately encoded and included in the final video bitstream data, depending on the used video compression standard. In such cases the application **must** call the vkGetEncodedVideoSessionParametersKHR command to retrieve the encoded parameter data from the used video session parameters object in order to be able to produce a compliant video bitstream.

**NOTE** This is needed because implementations **may** have changed some of the codec-

specific parameters stored in the video session parameters object, as defined in the Video Encode Parameter Overrides section. In addition, the vkGetEncodedVideoSessionParametersKHR command enables the application to retrieve the encoded parameter data without having to encode these codec-specific parameters manually.

Encoded parameter data **can** be retrieved from a video session parameters object created with a video encode operation using the command:

```
// Provided by VK_KHR_video_encode_queue
VkResult vkGetEncodedVideoSessionParametersKHR(
    VkDevice                                    device,
    const VkVideoEncodeSessionParametersGetInfoKHR* pVideoSessionParametersInfo,
    VkVideoEncodeSessionParametersFeedbackInfoKHR* pFeedbackInfo,
    size_t*                                     pDataSize,
    void*                                       pData);
```

- `device` is the logical device that owns the video session parameters object.
- `pVideoSessionParametersInfo` is a pointer to a VkVideoEncodeSessionParametersGetInfoKHR structure specifying the parameters of the encoded parameter data to retrieve.
- `pFeedbackInfo` is either `NULL` or a pointer to a VkVideoEncodeSessionParametersFeedbackInfoKHR structure in which feedback about the requested parameter data is returned.
- `pDataSize` is a pointer to a `size_t` value related to the amount of encode parameter data returned, as described below.
- `pData` is either `NULL` or a pointer to a buffer to write the encoded parameter data to.

If `pData` is `NULL`, then the size of the encoded parameter data, in bytes, that **can** be retrieved is returned in `pDataSize`. Otherwise, `pDataSize` **must** point to a variable set by the application to the size of the buffer, in bytes, pointed to by `pData`, and on return the variable is overwritten with the number of bytes actually written to `pData`. If `pDataSize` is less than the size of the encoded parameter data that **can** be retrieved, then no data will be written to `pData`, zero will be written to `pDataSize`, and `VK_INCOMPLETE` will be returned instead of `VK_SUCCESS`, to indicate that no encoded parameter data was returned.

If `pFeedbackInfo` is not `NULL` then the members of the VkVideoEncodeSessionParametersFeedbackInfoKHR structure and any additional structures included in its `pNext` chain that are applicable to the video session parameters object specified in `pVideoSessionParametersInfo->videoSessionParameters` will be filled with feedback about the requested parameter data on all successful calls to this command.

**NOTE** This includes the cases when `pData` is `NULL` or when `VK_INCOMPLETE` is returned by the command, and enables the application to determine whether the implementation overrode any of the requested video session parameters without actually needing to retrieve the encoded parameter data itself.

## Valid Usage

- VUID-vkGetEncodedVideoSessionParametersKHR-pVideoSessionParametersInfo-08359 `pVideoSessionParametersInfo->videoSessionParameters` **must** have been created with an encode operation
- VUID-vkGetEncodedVideoSessionParametersKHR-pVideoSessionParametersInfo-08262 If `pVideoSessionParametersInfo->videoSessionParameters` was created with the video codec operation `VK_VIDEO_CODEC_OPERATION_ENCODE_H264_BIT_KHR`, then the `pNext` chain of `pVideoSessionParametersInfo` **must** include a VkVideoEncodeH264SessionParametersGetInfoKHR structure
- VUID-vkGetEncodedVideoSessionParametersKHR-pVideoSessionParametersInfo-08263 If `pVideoSessionParametersInfo->videoSessionParameters` was created with the video codec operation `VK_VIDEO_CODEC_OPERATION_ENCODE_H264_BIT_KHR`, then for the VkVideoEncodeH264SessionParametersGetInfoKHR structure included in the `pNext` chain of `pVideoSessionParametersInfo`, if its `writeStdSPS` member is `VK_TRUE`, then `pVideoSessionParametersInfo->videoSessionParameters` **must** contain a `StdVideoH264SequenceParameterSet` entry with `seq_parameter_set_id` matching VkVideoEncodeH264SessionParametersGetInfoKHR::`stdSPSId`
- VUID-vkGetEncodedVideoSessionParametersKHR-pVideoSessionParametersInfo-08264 If `pVideoSessionParametersInfo->videoSessionParameters` was created with the video codec operation `VK_VIDEO_CODEC_OPERATION_ENCODE_H264_BIT_KHR`, then for the VkVideoEncodeH264SessionParametersGetInfoKHR structure included in the `pNext` chain of `pVideoSessionParametersInfo`, if its `writeStdPPS` member is `VK_TRUE`, then `pVideoSessionParametersInfo->videoSessionParameters` **must** contain a `StdVideoH264PictureParameterSet` entry with `seq_parameter_set_id` and `pic_parameter_set_id` matching VkVideoEncodeH264SessionParametersGetInfoKHR::`stdSPSId` and VkVideoEncodeH264SessionParametersGetInfoKHR::`stdPPSId`, respectively
- VUID-vkGetEncodedVideoSessionParametersKHR-pVideoSessionParametersInfo-08265 If `pVideoSessionParametersInfo->videoSessionParameters` was created with the video codec operation `VK_VIDEO_CODEC_OPERATION_ENCODE_H265_BIT_KHR`, then the `pNext` chain of `pVideoSessionParametersInfo` **must** include a VkVideoEncodeH265SessionParametersGetInfoKHR structure
- VUID-vkGetEncodedVideoSessionParametersKHR-pVideoSessionParametersInfo-08266 If `pVideoSessionParametersInfo->videoSessionParameters` was created with the video codec operation `VK_VIDEO_CODEC_OPERATION_ENCODE_H265_BIT_KHR`, then for the VkVideoEncodeH265SessionParametersGetInfoKHR structure included in the `pNext` chain of `pVideoSessionParametersInfo`, if its `writeStdVPS` member is `VK_TRUE`, then `pVideoSessionParametersInfo->videoSessionParameters` **must** contain a `StdVideoH265VideoParameterSet` entry with `vps_video_parameter_set_id` matching VkVideoEncodeH265SessionParametersGetInfoKHR::`stdVPSId`
- VUID-vkGetEncodedVideoSessionParametersKHR-pVideoSessionParametersInfo-08267 If `pVideoSessionParametersInfo->videoSessionParameters` was created with the video codec operation `VK_VIDEO_CODEC_OPERATION_ENCODE_H265_BIT_KHR`, then for the VkVideoEncodeH265SessionParametersGetInfoKHR structure included in the `pNext` chain of `pVideoSessionParametersInfo`, if its `writeStdSPS` member is `VK_TRUE`, then

pVideoSessionParametersInfo->videoSessionParameters **must** contain a StdVideoH265SequenceParameterSet entry with sps_video_parameter_set_id and sps_seq_parameter_set_id matching VkVideoEncodeH265SessionParametersGetInfoKHR::stdVPSId and VkVideoEncodeH265SessionParametersGetInfoKHR::stdSPSId, respectively

- VUID-vkGetEncodedVideoSessionParametersKHR-pVideoSessionParametersInfo-08268
  If pVideoSessionParametersInfo->videoSessionParameters was created with the video codec operation VK_VIDEO_CODEC_OPERATION_ENCODE_H265_BIT_KHR, then for the VkVideoEncodeH265SessionParametersGetInfoKHR structure included in the pNext chain of pVideoSessionParametersInfo, if its writeStdPPS member is VK_TRUE, then pVideoSessionParametersInfo->videoSessionParameters **must** contain a StdVideoH265PictureParameterSet entry with sps_video_parameter_set_id, pps_seq_parameter_set_id, and pps_pic_parameter_set_id matching VkVideoEncodeH265SessionParametersGetInfoKHR::stdVPSId, VkVideoEncodeH265SessionParametersGetInfoKHR::stdSPSId, and VkVideoEncodeH265SessionParametersGetInfoKHR::stdPPSId, respectively

## Valid Usage (Implicit)

- VUID-vkGetEncodedVideoSessionParametersKHR-device-parameter
  device **must** be a valid VkDevice handle
- VUID-vkGetEncodedVideoSessionParametersKHR-pVideoSessionParametersInfo-parameter
  pVideoSessionParametersInfo **must** be a valid pointer to a valid VkVideoEncodeSessionParametersGetInfoKHR structure
- VUID-vkGetEncodedVideoSessionParametersKHR-pFeedbackInfo-parameter
  If pFeedbackInfo is not NULL, pFeedbackInfo **must** be a valid pointer to a VkVideoEncodeSessionParametersFeedbackInfoKHR structure
- VUID-vkGetEncodedVideoSessionParametersKHR-pDataSize-parameter
  pDataSize **must** be a valid pointer to a size_t value
- VUID-vkGetEncodedVideoSessionParametersKHR-pData-parameter
  If the value referenced by pDataSize is not 0, and pData is not NULL, pData **must** be a valid pointer to an array of pDataSize bytes

## Return Codes

### Success

- VK_SUCCESS
- VK_INCOMPLETE

### Failure

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY

The VkVideoEncodeSessionParametersGetInfoKHR structure is defined as:

```
// Provided by VK_KHR_video_encode_queue
typedef struct VkVideoEncodeSessionParametersGetInfoKHR {
    VkStructureType                sType;
    const void*                    pNext;
    VkVideoSessionParametersKHR    videoSessionParameters;
} VkVideoEncodeSessionParametersGetInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- videoSessionParameters is the VkVideoSessionParametersKHR object to retrieve encoded parameter data from.

Depending on the used video encode operation, additional codec-specific structures **may** need to be included in the pNext chain of this structure to identify the specific video session parameters to retrieve encoded parameter data for, as described in the corresponding sections.

### Valid Usage (Implicit)

- VUID-VkVideoEncodeSessionParametersGetInfoKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_VIDEO_ENCODE_SESSION_PARAMETERS_GET_INFO_KHR
- VUID-VkVideoEncodeSessionParametersGetInfoKHR-pNext-pNext
  Each pNext member of any structure (including this one) in the pNext chain **must** be either NULL or a pointer to a valid instance of VkVideoEncodeH264SessionParametersGetInfoKHR or VkVideoEncodeH265SessionParametersGetInfoKHR
- VUID-VkVideoEncodeSessionParametersGetInfoKHR-sType-unique
  The sType value of each struct in the pNext chain **must** be unique
- VUID-VkVideoEncodeSessionParametersGetInfoKHR-videoSessionParameters-parameter
  videoSessionParameters **must** be a valid VkVideoSessionParametersKHR handle

The VkVideoEncodeSessionParametersFeedbackInfoKHR structure is defined as:

```
// Provided by VK_KHR_video_encode_queue
typedef struct VkVideoEncodeSessionParametersFeedbackInfoKHR {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           hasOverrides;
} VkVideoEncodeSessionParametersFeedbackInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- hasOverrides indicates whether any of the requested parameter data were overridden by the

implementation.

Depending on the used video encode operation, additional codec-specific structures **can** be included in the pNext chain of this structure to capture codec-specific feedback information about the requested parameter data, as described in the corresponding sections.

### Valid Usage (Implicit)

- VUID-VkVideoEncodeSessionParametersFeedbackInfoKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_VIDEO_ENCODE_SESSION_PARAMETERS_FEEDBACK_INFO_KHR
- VUID-VkVideoEncodeSessionParametersFeedbackInfoKHR-pNext-pNext
  Each pNext member of any structure (including this one) in the pNext chain **must** be either NULL or a pointer to a valid instance of VkVideoEncodeH264SessionParametersFeedbackInfoKHR or VkVideoEncodeH265SessionParametersFeedbackInfoKHR
- VUID-VkVideoEncodeSessionParametersFeedbackInfoKHR-sType-unique
  The sType value of each struct in the pNext chain **must** be unique

## 42.15.6. Video Encode Commands

To launch video encode operations, call:

```
// Provided by VK_KHR_video_encode_queue
void vkCmdEncodeVideoKHR(
    VkCommandBuffer                             commandBuffer,
    const VkVideoEncodeInfoKHR*                 pEncodeInfo);
```

- commandBuffer is the command buffer in which to record the command.
- pEncodeInfo is a pointer to a VkVideoEncodeInfoKHR structure specifying the parameters of the video encode operations.

Each call issues one or more video encode operations. The implicit parameter opCount corresponds to the number of video encode operations issued by the command. After calling this command, the active query index of each active query is incremented by opCount.

Currently each call to this command results in the issue of a single video encode operation.

If the bound video session was created with VK_VIDEO_SESSION_CREATE_INLINE_QUERIES_BIT_KHR and the pNext chain of pEncodeInfo includes a VkVideoInlineQueryInfoKHR structure with its queryPool member specifying a valid VkQueryPool handle, then this command will execute a query for each video encode operation issued by it.

**Active Reference Picture Information**

The list of active reference pictures used by a video encode operation is a list of image subregions used as the source of reference picture data and related parameters, and is derived from the VkVideoReferenceSlotInfoKHR structures provided as the elements of the pEncodeInfo-

>pReferenceSlots array. For each element of pEncodeInfo->pReferenceSlots, one or more elements are added to the active reference picture list, as defined by the codec-specific semantics. Each element of this list contains the following information:

- The image subregion within the image subresource referred to by the video picture resource used as the reference picture.
- The DPB slot index the reference picture is associated with.
- The codec-specific reference information related to the reference picture.

**Reconstructed Picture Information**

Information related to the optional reconstructed picture used by a video encode operation is derived from the VkVideoReferenceSlotInfoKHR structure pointed to by pEncodeInfo->pSetupReferenceSlot, if not NULL, as defined by the codec-specific semantics, and consists of the following:

- The image subregion within the image subresource referred to by the video picture resource used as the reconstructed picture.
- The DPB slot index to use for picture reconstruction.
- The codec-specific reference information related to the reconstructed picture.

Specifying a valid VkVideoReferenceSlotInfoKHR structure in pEncodeInfo->pSetupReferenceSlot is always required, unless the video session was created with VkVideoSessionCreateInfoKHR::maxDpbSlot equal to zero. However, the DPB slot identified by pEncodeInfo->pSetupReferenceSlot->slotIndex is only activated with the reconstructed picture specified in pEncodeInfo->pSetupReferenceSlot->pPictureResource if reference picture setup is requested according to the codec-specific semantics.

If reconstructed picture information is specified, but reference picture setup is not requested, according to the codec-specific semantics, the contents of the video picture resource corresponding to the reconstructed picture will be undefined after the video encode operation.

| | |
|---|---|
| **NOTE** | Some implementations may always output the reconstructed picture or use it as temporary storage during the video encode operation even when the reconstructed picture is not marked for future reference. |

**Encode Input Picture Information**

Information related to the encode input picture used by a video encode operation is derived from pEncodeInfo->srcPictureResource and any codec-specific parameters provided in the pEncodeInfo->pNext chain, as defined by the codec-specific semantics, and consists of the following:

- The image subregion within the image subresource referred to by the video picture resource used as the encode input picture.
- The codec-specific picture information related to the encoded picture.

Several limiting values are defined below that are referenced by the relevant valid usage statements of this command.

- Let `uint32_t activeReferencePictureCount` be the size of the list of active reference pictures used by the video encode operation. Unless otherwise defined, `activeReferencePictureCount` is set to the value of `pEncodeInfo->referenceSlotCount`.
- Let `VkOffset2D codedOffsetGranularity` be the minimum alignment requirement for the coded offset of video picture resources. Unless otherwise defined, the value of the `x` and `y` members of `codedOffsetGranularity` are `0`.
- Let `uint32_t dpbFrameUseCount[]` be an array of size `maxDpbSlots`, where `maxDpbSlots` is the VkVideoSessionCreateInfoKHR::`maxDpbSlots` the bound video session was created with, with each element indicating the number of times a frame associated with the corresponding DPB slot index is referred to by the video coding operation. Let the initial value of each element of the array be `0`.
  - If `pEncodeInfo->pSetupReferenceSlot` is not `NULL`, then `dpbFrameUseCount[i]` is incremented by one, where `i` equals `pEncodeInfo->pSetupReferenceSlot->slotIndex`.
  - For each element of `pEncodeInfo->pReferenceSlots`, `dpbFrameUseCount[i]` is incremented by one, where `i` equals the `slotIndex` member of the corresponding element.
- Let `VkExtent2D maxCodingBlockSize` be the maximum codec-specific coding block size that **may** be used by the video encode operation.
  - If the bound video session object was created with an H.264 encode profile, then let `maxCodingBlockSize` be equal to the size of an H.264 macroblock, i.e. `{16,16}`.
  - If the bound video session object was created with an H.265 encode profile, then let `maxCodingBlockSize` be equal to the maximum H.265 coding block size that **may** be used by the video encode operation derived as the maximum of the CTB sizes corresponding to the VkVideoEncodeH265CtbSizeFlagBitsKHR bits set in VkVideoEncodeH265CapabilitiesKHR::`ctbSizes`, as returned by vkGetPhysicalDeviceVideoCapabilitiesKHR for the video profile the bound video session was created with.
  - Otherwise, `maxCodingBlockSize` is undefined.
- If `maxCodingBlockSize` is defined, then let `VkExtent2D minCodingBlockExtent` be the coded extent of the encode input picture expressed in terms of codec-specific coding blocks, assuming the maximum size of such coding blocks, as defined by `maxCodingBlockSize`, calculated from the value of the `codedExtent` member of `pEncodeInfo->srcPictureResource` as follows:
  - `minCodingBlockExtent.width` = (`codedExtent.width` `maxCodingBlockSize.width` - 1) / `maxCodingBlockSize.width`
  - `minCodingBlockExtent.height` = (`codedExtent.height` `maxCodingBlockSize.height` - 1) / `maxCodingBlockSize.height`
- If the bound video session object was created with an H.264 encode profile, then:
  - Let `StdVideoH264PictureType h264PictureType` be the picture type of the encoded picture set to the value of `pStdPictureInfo->primary_pic_type` specified in the VkVideoEncodeH264PictureInfoKHR structure included in the `pEncodeInfo->pNext` chain.
  - Let `StdVideoH264PictureType h264L0PictureTypes[]` and `StdVideoH264PictureType h264L1PictureTypes[]` be the picture types of the reference pictures in the L0 and L1 reference lists, respectively. If `pStdPictureInfo->pRefLists` specified in the VkVideoEncodeH264PictureInfoKHR structure included in the `pEncodeInfo->pNext` chain is

not NULL, then for each reference index specified in the elements of the pStdPictureInfo->pRefLists->RefPicList0 and pStdPictureInfo->pRefLists->RefPicList1 arrays, if the reference index is not STD_VIDEO_H264_NO_REFERENCE_PICTURE, pStdReferenceInfo->primary_pic_type is added to h264L0PictureTypes or h264L1PictureTypes, respectively, where pStdReferenceInfo is the member of the VkVideoEncodeH264DpbSlotInfoKHR structure included in the pNext chain of the element of pEncodeInfo->pReferenceSlots for which slotIndex equals the reference index in question.

- If the bound video session object was created with an H.265 encode profile, then:
  - Let StdVideoH265PictureType h265PictureType be the picture type of the encoded picture set to the value of pStdPictureInfo->pic_type specified in the VkVideoEncodeH265PictureInfoKHR structure included in the pEncodeInfo->pNext chain.
  - Let StdVideoH265PictureType h265L0PictureTypes[] and StdVideoH265PictureType h265L1PictureTypes[] be the picture types of the reference pictures in the L0 and L1 reference lists, respectively. If pStdPictureInfo->pRefLists specified in the VkVideoEncodeH265PictureInfoKHR structure included in the pEncodeInfo->pNext chain is not NULL, then for each reference index specified in the elements of the pStdPictureInfo->pRefLists->RefPicList0 and pStdPictureInfo->pRefLists->RefPicList1 arrays, if the reference index is not STD_VIDEO_H265_NO_REFERENCE_PICTURE, pStdReferenceInfo->pic_type is added to h265L0PictureTypes or h265L1PictureTypes, respectively, where pStdReferenceInfo is the member of the VkVideoEncodeH265DpbSlotInfoKHR structure included in the pNext chain of the element of pEncodeInfo->pReferenceSlots for which slotIndex equals the reference index in question.

## Valid Usage

- VUID-vkCmdEncodeVideoKHR-None-08250
  The bound video session **must** have been created with an encode operation
- VUID-vkCmdEncodeVideoKHR-None-07012
  The bound video session **must** not be in uninitialized state at the time the command is executed on the device
- VUID-vkCmdEncodeVideoKHR-None-08318
  The bound video session parameters object **must** have been created with the currently set video encode quality level for the bound video session at the time the command is executed on the device
- VUID-vkCmdEncodeVideoKHR-opCount-07174
  For each active query, the active query index corresponding to the query type of that query plus opCount **must** be less than or equal to the last activatable query index corresponding to the query type of that query plus one
- VUID-vkCmdEncodeVideoKHR-pNext-08360
  If the bound video session was created with VK_VIDEO_SESSION_CREATE_INLINE_QUERIES_BIT_KHR, and the pNext chain of pEncodeInfo includes a VkVideoInlineQueryInfoKHR structure with its queryPool member specifying a valid VkQueryPool handle, then VkVideoInlineQueryInfoKHR::queryCount **must** equal opCount

- VUID-vkCmdEncodeVideoKHR-pNext-08361
  If the bound video session was created with `VK_VIDEO_SESSION_CREATE_INLINE_QUERIES_BIT_KHR`, and the `pNext` chain of `pEncodeInfo` includes a VkVideoInlineQueryInfoKHR structure with its `queryPool` member specifying a valid `VkQueryPool` handle, then all the queries used by the command, as specified by the VkVideoInlineQueryInfoKHR structure, **must** be *unavailable*
- VUID-vkCmdEncodeVideoKHR-queryType-08362
  If the bound video session was created with `VK_VIDEO_SESSION_CREATE_INLINE_QUERIES_BIT_KHR`, then the `queryType` used to create the `queryPool` specified in the VkVideoInlineQueryInfoKHR structure included in the `pNext` chain of `pEncodeInfo` **must** be `VK_QUERY_TYPE_RESULT_STATUS_ONLY_KHR` or `VK_QUERY_TYPE_VIDEO_ENCODE_FEEDBACK_KHR`
- VUID-vkCmdEncodeVideoKHR-queryPool-08363
  If the bound video session was created with `VK_VIDEO_SESSION_CREATE_INLINE_QUERIES_BIT_KHR`, then the `queryPool` specified in the VkVideoInlineQueryInfoKHR structure included in the `pNext` chain of `pEncodeInfo` **must** have been created with a VkVideoProfileInfoKHR structure included in the `pNext` chain of VkQueryPoolCreateInfo identical to the one specified in VkVideoSessionCreateInfoKHR::`pVideoProfile` the bound video session was created with
- VUID-vkCmdEncodeVideoKHR-queryType-08364
  If the bound video session was created with `VK_VIDEO_SESSION_CREATE_INLINE_QUERIES_BIT_KHR`, and the `queryType` used to create the `queryPool` specified in the VkVideoInlineQueryInfoKHR structure included in the `pNext` chain of `pEncodeInfo` is `VK_QUERY_TYPE_RESULT_STATUS_ONLY_KHR`, then the `VkCommandPool` that `commandBuffer` was allocated from **must** have been created with a queue family index that supports result status queries, as indicated by VkQueueFamilyQueryResultStatusPropertiesKHR::`queryResultStatusSupport`
- VUID-vkCmdEncodeVideoKHR-pEncodeInfo-08201
  `pEncodeInfo->dstBuffer` **must** be compatible with the video profile the bound video session was created with
- VUID-vkCmdEncodeVideoKHR-commandBuffer-08202
  If `commandBuffer` is an unprotected command buffer and `protectedNoFault` is not supported, then `pEncodeInfo->dstBuffer` **must** not be a protected buffer
- VUID-vkCmdEncodeVideoKHR-commandBuffer-08203
  If `commandBuffer` is a protected command buffer and `protectedNoFault` is not supported, then `pEncodeInfo->dstBuffer` **must** be a protected buffer
- VUID-vkCmdEncodeVideoKHR-pEncodeInfo-08204
  `pEncodeInfo->dstBufferOffset` **must** be an integer multiple of VkVideoCapabilitiesKHR::`minBitstreamBufferOffsetAlignment`, as returned by vkGetPhysicalDeviceVideoCapabilitiesKHR for the video profile the bound video session was created with
- VUID-vkCmdEncodeVideoKHR-pEncodeInfo-08205
  `pEncodeInfo->dstBufferRange` **must** be an integer multiple of VkVideoCapabilitiesKHR::`minBitstreamBufferSizeAlignment`, as returned by vkGetPhysicalDeviceVideoCapabilitiesKHR for the video profile the bound video session

was created with

- VUID-vkCmdEncodeVideoKHR-pEncodeInfo-08206
  `pEncodeInfo->srcPictureResource.imageViewBinding` **must** be compatible with the video profile the bound video session was created with
- VUID-vkCmdEncodeVideoKHR-pEncodeInfo-08207
  The format of `pEncodeInfo->srcPictureResource.imageViewBinding` **must** match the VkVideoSessionCreateInfoKHR::`pictureFormat` the bound video session was created with
- VUID-vkCmdEncodeVideoKHR-pEncodeInfo-08208
  `pEncodeInfo->srcPictureResource.codedOffset` **must** be an integer multiple of `codedOffsetGranularity`
- VUID-vkCmdEncodeVideoKHR-pEncodeInfo-08209
  `pEncodeInfo->srcPictureResource.codedExtent` **must** be between `minCodedExtent` and `maxCodedExtent`, inclusive, the bound video session was created with
- VUID-vkCmdEncodeVideoKHR-pEncodeInfo-08210
  `pEncodeInfo->srcPictureResource.imageViewBinding` **must** have been created with `VK_IMAGE_USAGE_VIDEO_ENCODE_SRC_BIT_KHR`
- VUID-vkCmdEncodeVideoKHR-commandBuffer-08211
  If `commandBuffer` is an unprotected command buffer and `protectedNoFault` is not supported, then `pEncodeInfo->srcPictureResource.imageViewBinding` **must** not have been created from a protected image
- VUID-vkCmdEncodeVideoKHR-commandBuffer-08212
  If `commandBuffer` is a protected command buffer and `protectedNoFault` is not supported, then `pEncodeInfo->srcPictureResource.imageViewBinding` **must** have been created from a protected image
- VUID-vkCmdEncodeVideoKHR-pEncodeInfo-08377
  `pEncodeInfo->pSetupReferenceSlot` **must** not be `NULL` unless the bound video session was created with VkVideoSessionCreateInfoKHR::`maxDpbSlots` equal to zero
- VUID-vkCmdEncodeVideoKHR-pEncodeInfo-08213
  If `pEncodeInfo->pSetupReferenceSlot` is not `NULL`, then `pEncodeInfo->pSetupReferenceSlot->slotIndex` **must** be less than the VkVideoSessionCreateInfoKHR::`maxDpbSlots` specified when the bound video session was created
- VUID-vkCmdEncodeVideoKHR-pEncodeInfo-08214
  If `pEncodeInfo->pSetupReferenceSlot` is not `NULL`, then `pEncodeInfo->pSetupReferenceSlot->pPictureResource->codedOffset` **must** be an integer multiple of `codedOffsetGranularity`
- VUID-vkCmdEncodeVideoKHR-pEncodeInfo-08215
  If `pEncodeInfo->pSetupReferenceSlot` is not `NULL`, then `pEncodeInfo->pSetupReferenceSlot->pPictureResource` **must** match one of the bound reference picture resource
- VUID-vkCmdEncodeVideoKHR-activeReferencePictureCount-08216
  `activeReferencePictureCount` **must** be less than or equal to the VkVideoSessionCreateInfoKHR::`maxActiveReferencePictures` specified when the bound video session was created
- VUID-vkCmdEncodeVideoKHR-slotIndex-08217
  The `slotIndex` member of each element of `pEncodeInfo->pReferenceSlots` **must** be less than

the VkVideoSessionCreateInfoKHR::maxDpbSlots specified when the bound video session was created

- VUID-vkCmdEncodeVideoKHR-codedOffset-08218
  The codedOffset member of the VkVideoPictureResourceInfoKHR structure pointed to by the pPictureResource member of each element of pEncodeInfo->pReferenceSlots **must** be an integer multiple of codedOffsetGranularity
- VUID-vkCmdEncodeVideoKHR-pPictureResource-08219
  The pPictureResource member of each element of pEncodeInfo->pReferenceSlots **must** match one of the bound reference picture resource associated with the DPB slot index specified in the slotIndex member of that element
- VUID-vkCmdEncodeVideoKHR-pPictureResource-08220
  Each video picture resource corresponding to the pPictureResource member specified in the elements of pEncodeInfo->pReferenceSlots **must** be unique within pEncodeInfo->pReferenceSlots
- VUID-vkCmdEncodeVideoKHR-dpbFrameUseCount-08221
  All elements of dpbFrameUseCount **must** be less than or equal to 1
- VUID-vkCmdEncodeVideoKHR-pEncodeInfo-08222
  The image subresource referred to by pEncodeInfo->srcPictureResource **must** be in the VK_IMAGE_LAYOUT_VIDEO_ENCODE_SRC_KHR layout at the time the video encode operation is executed on the device
- VUID-vkCmdEncodeVideoKHR-pEncodeInfo-08223
  If pEncodeInfo->pSetupReferenceSlot is not NULL, then the image subresource referred to by pEncodeInfo->pSetupReferenceSlot->pPictureResource **must** be in the VK_IMAGE_LAYOUT_VIDEO_ENCODE_DPB_KHR layout at the time the video encode operation is executed on the device
- VUID-vkCmdEncodeVideoKHR-pPictureResource-08224
  The image subresource referred to by the pPictureResource member of each element of pEncodeInfo->pReferenceSlots **must** be in the VK_IMAGE_LAYOUT_VIDEO_ENCODE_DPB_KHR layout at the time the video encode operation is executed on the device
- VUID-vkCmdEncodeVideoKHR-pNext-08225
  If the bound video session was created with the video codec operation VK_VIDEO_CODEC_OPERATION_ENCODE_H264_BIT_KHR, then the pNext chain of pEncodeInfo **must** include a VkVideoEncodeH264PictureInfoKHR structure
- VUID-vkCmdEncodeVideoKHR-StdVideoH264SequenceParameterSet-08226
  If the bound video session was created with the video codec operation VK_VIDEO_CODEC_OPERATION_ENCODE_H264_BIT_KHR, then the bound video session parameters object **must** contain a StdVideoH264SequenceParameterSet entry with seq_parameter_set_id matching StdVideoEncodeH264PictureInfo::seq_parameter_set_id that is provided in the pStdPictureInfo member of the VkVideoEncodeH264PictureInfoKHR structure included in the pNext chain of pEncodeInfo
- VUID-vkCmdEncodeVideoKHR-StdVideoH264PictureParameterSet-08227
  If the bound video session was created with the video codec operation VK_VIDEO_CODEC_OPERATION_ENCODE_H264_BIT_KHR, then the bound video session parameters object **must** contain a StdVideoH264PictureParameterSet entry with seq_parameter_set_id

and pic_parameter_set_id matching StdVideoEncodeH264PictureInfo::seq_parameter_set_id and StdVideoEncodeH264PictureInfo::pic_parameter_set_id, respectively, that are provided in the pStdPictureInfo member of the VkVideoEncodeH264PictureInfoKHR structure included in the pNext chain of pEncodeInfo

- VUID-vkCmdEncodeVideoKHR-pEncodeInfo-08228
  If the bound video session was created with the video codec operation VK_VIDEO_CODEC_OPERATION_ENCODE_H264_BIT_KHR and pEncodeInfo->pSetupReferenceSlot is not NULL, then the pNext chain of pEncodeInfo->pSetupReferenceSlot **must** include a VkVideoEncodeH264DpbSlotInfoKHR structure
- VUID-vkCmdEncodeVideoKHR-pNext-08229
  If the bound video session was created with the video codec operation VK_VIDEO_CODEC_OPERATION_ENCODE_H264_BIT_KHR, then the pNext chain of each element of pEncodeInfo->pReferenceSlots **must** include a VkVideoEncodeH264DpbSlotInfoKHR structure
- VUID-vkCmdEncodeVideoKHR-constantQp-08269
  If the bound video session was created with the video codec operation VK_VIDEO_CODEC_OPERATION_ENCODE_H264_BIT_KHR and the current rate control mode is not VK_VIDEO_ENCODE_RATE_CONTROL_MODE_DISABLED_BIT_KHR, then VkVideoEncodeH264NaluSliceInfoKHR::constantQp **must** be zero for each element of the pNaluSliceEntries member of the VkVideoEncodeH264PictureInfoKHR structure included in the pNext chain of pEncodeInfo
- VUID-vkCmdEncodeVideoKHR-constantQp-08270
  If the bound video session was created with the video codec operation VK_VIDEO_CODEC_OPERATION_ENCODE_H264_BIT_KHR and the current rate control mode is VK_VIDEO_ENCODE_RATE_CONTROL_MODE_DISABLED_BIT_KHR, then VkVideoEncodeH264NaluSliceInfoKHR::constantQp **must** be between VkVideoEncodeH264CapabilitiesKHR::minQp and VkVideoEncodeH264CapabilitiesKHR::maxQp, as returned by vkGetPhysicalDeviceVideoCapabilitiesKHR for the video profile the bound video session was created with, for each element of the pNaluSliceEntries member of the VkVideoEncodeH264PictureInfoKHR structure included in the pNext chain of pEncodeInfo
- VUID-vkCmdEncodeVideoKHR-constantQp-08271
  If the bound video session was created with the video codec operation VK_VIDEO_CODEC_OPERATION_ENCODE_H264_BIT_KHR and VkVideoEncodeH264CapabilitiesKHR::flags does not include VK_VIDEO_ENCODE_H264_CAPABILITY_PER_SLICE_CONSTANT_QP_BIT_KHR, as returned by vkGetPhysicalDeviceVideoCapabilitiesKHR for the video profile the bound video session was created with, then VkVideoEncodeH264NaluSliceInfoKHR::constantQp **must** have the same value for each element of the pNaluSliceEntries member of the VkVideoEncodeH264PictureInfoKHR structure included in the pNext chain of pEncodeInfo
- VUID-vkCmdEncodeVideoKHR-naluSliceEntryCount-08302
  If the bound video session was created with the video codec operation VK_VIDEO_CODEC_OPERATION_ENCODE_H264_BIT_KHR, then the naluSliceEntryCount member of the VkVideoEncodeH264PictureInfoKHR structure included in the pNext chain of pEncodeInfo **must** be less than or equal to minCodingBlockExtent.width multiplied by minCodingBlockExtent.height

- VUID-vkCmdEncodeVideoKHR-naluSliceEntryCount-08312
  If the bound video session was created with the video codec operation VK_VIDEO_CODEC_OPERATION_ENCODE_H264_BIT_KHR and VkVideoEncodeH264CapabilitiesKHR::flags does not include VK_VIDEO_ENCODE_H264_CAPABILITY_ROW_UNALIGNED_SLICE_BIT_KHR, as returned by vkGetPhysicalDeviceVideoCapabilitiesKHR for the video profile the bound video session was created with, then the naluSliceEntryCount member of the VkVideoEncodeH264PictureInfoKHR structure included in the pNext chain of pEncodeInfo **must** be less than or equal to minCodingBlockExtent.height
- VUID-vkCmdEncodeVideoKHR-pNext-08352
  If the bound video session was created with the video codec operation VK_VIDEO_CODEC_OPERATION_ENCODE_H264_BIT_KHR, the pNext chain of pEncodeInfo includes a VkVideoEncodeH264PictureInfoKHR structure, and pEncodeInfo->referenceSlotCount is greater than zero, then VkVideoEncodeH264PictureInfoKHR::pStdPictureInfo->pRefLists **must** not be NULL
- VUID-vkCmdEncodeVideoKHR-pNext-08339
  If the bound video session was created with the video codec operation VK_VIDEO_CODEC_OPERATION_ENCODE_H264_BIT_KHR, the pNext chain of pEncodeInfo includes a VkVideoEncodeH264PictureInfoKHR structure, and VkVideoEncodeH264PictureInfoKHR::pStdPictureInfo->pRefLists is not NULL, then each element of the RefPicList0 and RefPicList1 array members of the StdVideoEncodeH264ReferenceListsInfo structure pointed to by VkVideoEncodeH264PictureInfoKHR::pStdPictureInfo->pRefLists **must** either be STD_VIDEO_H264_NO_REFERENCE_PICTURE or **must** equal the slotIndex member of one of the elements of pEncodeInfo->pReferenceSlots
- VUID-vkCmdEncodeVideoKHR-pNext-08353
  If the bound video session was created with the video codec operation VK_VIDEO_CODEC_OPERATION_ENCODE_H264_BIT_KHR, the pNext chain of pEncodeInfo includes a VkVideoEncodeH264PictureInfoKHR structure, and pEncodeInfo->referenceSlotCount is greater than zero, then the slotIndex member of each element of pEncodeInfo->pReferenceSlots **must** equal one of the elements of the RefPicList0 or RefPicList1 array members of the StdVideoEncodeH264ReferenceListsInfo structure pointed to by VkVideoEncodeH264PictureInfoKHR::pStdPictureInfo->pRefLists
- VUID-vkCmdEncodeVideoKHR-maxPPictureL0ReferenceCount-08340
  If the bound video session was created with the video codec operation VK_VIDEO_CODEC_OPERATION_ENCODE_H264_BIT_KHR and VkVideoEncodeH264CapabilitiesKHR::maxPPictureL0ReferenceCount is zero, as returned by vkGetPhysicalDeviceVideoCapabilitiesKHR for the video profile the bound video session was created with, then h264PictureType and each element of h264L0PictureTypes and h264L1PictureTypes **must** not be STD_VIDEO_H264_PICTURE_TYPE_P
- VUID-vkCmdEncodeVideoKHR-maxBPictureL0ReferenceCount-08341
  If the bound video session was created with the video codec operation VK_VIDEO_CODEC_OPERATION_ENCODE_H264_BIT_KHR and VkVideoEncodeH264CapabilitiesKHR::maxBPictureL0ReferenceCount and VkVideoEncodeH264CapabilitiesKHR::maxL1ReferenceCount are both zero, as returned by vkGetPhysicalDeviceVideoCapabilitiesKHR for the video profile the bound video session was created with, then h264PictureType and each element of h264L0PictureTypes and h264L1PictureTypes **must** not be STD_VIDEO_H264_PICTURE_TYPE_B

- VUID-vkCmdEncodeVideoKHR-flags-08342
  If the bound video session was created with the video codec operation VK_VIDEO_CODEC_OPERATION_ENCODE_H264_BIT_KHR and VkVideoEncodeH264CapabilitiesKHR::flags does not include VK_VIDEO_ENCODE_H264_CAPABILITY_B_FRAME_IN_L0_LIST_BIT_KHR, as returned by vkGetPhysicalDeviceVideoCapabilitiesKHR for the video profile the bound video session was created with, then each element of h264L0PictureTypes **must** not be STD_VIDEO_H264_PICTURE_TYPE_B
- VUID-vkCmdEncodeVideoKHR-flags-08343
  If the bound video session was created with the video codec operation VK_VIDEO_CODEC_OPERATION_ENCODE_H264_BIT_KHR and VkVideoEncodeH264CapabilitiesKHR::flags does not include VK_VIDEO_ENCODE_H264_CAPABILITY_B_FRAME_IN_L1_LIST_BIT_KHR, as returned by vkGetPhysicalDeviceVideoCapabilitiesKHR for the video profile the bound video session was created with, then each element of h264L1PictureTypes **must** not be STD_VIDEO_H264_PICTURE_TYPE_B
- VUID-vkCmdEncodeVideoKHR-pNext-08230
  If the bound video session was created with the video codec operation VK_VIDEO_CODEC_OPERATION_ENCODE_H265_BIT_KHR, then the pNext chain of pEncodeInfo **must** include a VkVideoEncodeH265PictureInfoKHR structure
- VUID-vkCmdEncodeVideoKHR-StdVideoH265VideoParameterSet-08231
  If the bound video session was created with the video codec operation VK_VIDEO_CODEC_OPERATION_ENCODE_H265_BIT_KHR, then the bound video session parameters object **must** contain a StdVideoH265VideoParameterSet entry with vps_video_parameter_set_id matching StdVideoEncodeH265PictureInfo::sps_video_parameter_set_id that is provided in the pStdPictureInfo member of the VkVideoEncodeH265PictureInfoKHR structure included in the pNext chain of pEncodeInfo
- VUID-vkCmdEncodeVideoKHR-StdVideoH265SequenceParameterSet-08232
  If the bound video session was created with the video codec operation VK_VIDEO_CODEC_OPERATION_ENCODE_H265_BIT_KHR, then the bound video session parameters object **must** contain a StdVideoH265SequenceParameterSet entry with sps_video_parameter_set_id and sps_seq_parameter_set_id matching StdVideoEncodeH265PictureInfo::sps_video_parameter_set_id and StdVideoEncodeH265PictureInfo::pps_seq_parameter_set_id, respectively, that are provided in the pStdPictureInfo member of the VkVideoEncodeH265PictureInfoKHR structure included in the pNext chain of pEncodeInfo
- VUID-vkCmdEncodeVideoKHR-StdVideoH265PictureParameterSet-08233
  If the bound video session was created with the video codec operation VK_VIDEO_CODEC_OPERATION_ENCODE_H265_BIT_KHR, then the bound video session parameters object **must** contain a StdVideoH265PictureParameterSet entry with sps_video_parameter_set_id, pps_seq_parameter_set_id, and pps_pic_parameter_set_id matching StdVideoEncodeH265PictureInfo::sps_video_parameter_set_id, StdVideoEncodeH265PictureInfo::pps_seq_parameter_set_id, and StdVideoEncodeH265PictureInfo::pps_pic_parameter_set_id, respectively, that are provided in the pStdPictureInfo member of the VkVideoEncodeH265PictureInfoKHR structure included in the pNext chain of pEncodeInfo
- VUID-vkCmdEncodeVideoKHR-pEncodeInfo-08234

If the bound video session was created with the video codec operation VK_VIDEO_CODEC_OPERATION_ENCODE_H265_BIT_KHR and pEncodeInfo->pSetupReferenceSlot is not NULL, then the pNext chain of pEncodeInfo->pSetupReferenceSlot **must** include a VkVideoEncodeH265DpbSlotInfoKHR structure

- VUID-vkCmdEncodeVideoKHR-pNext-08235
  If the bound video session was created with the video codec operation VK_VIDEO_CODEC_OPERATION_ENCODE_H265_BIT_KHR, then the pNext chain of each element of pEncodeInfo->pReferenceSlots **must** include a VkVideoEncodeH265DpbSlotInfoKHR structure

- VUID-vkCmdEncodeVideoKHR-constantQp-08272
  If the bound video session was created with the video codec operation VK_VIDEO_CODEC_OPERATION_ENCODE_H265_BIT_KHR and the current rate control mode is not VK_VIDEO_ENCODE_RATE_CONTROL_MODE_DISABLED_BIT_KHR, then VkVideoEncodeH265NaluSliceSegmentInfoKHR::constantQp **must** be zero for each element of the pNaluSliceSegmentEntries member of the VkVideoEncodeH265PictureInfoKHR structure included in the pNext chain of pEncodeInfo

- VUID-vkCmdEncodeVideoKHR-constantQp-08273
  If the bound video session was created with the video codec operation VK_VIDEO_CODEC_OPERATION_ENCODE_H265_BIT_KHR and the current rate control mode is VK_VIDEO_ENCODE_RATE_CONTROL_MODE_DISABLED_BIT_KHR, then VkVideoEncodeH265NaluSliceSegmentInfoKHR::constantQp **must** be between VkVideoEncodeH265CapabilitiesKHR::minQp and VkVideoEncodeH265CapabilitiesKHR::maxQp, as returned by vkGetPhysicalDeviceVideoCapabilitiesKHR for the video profile the bound video session was created with, for each element of the pNaluSliceSegmentEntries member of the VkVideoEncodeH265PictureInfoKHR structure included in the pNext chain of pEncodeInfo

- VUID-vkCmdEncodeVideoKHR-constantQp-08274
  If the bound video session was created with the video codec operation VK_VIDEO_CODEC_OPERATION_ENCODE_H265_BIT_KHR and VkVideoEncodeH265CapabilitiesKHR::flags does not include VK_VIDEO_ENCODE_H265_CAPABILITY_PER_SLICE_SEGMENT_CONSTANT_QP_BIT_KHR, as returned by vkGetPhysicalDeviceVideoCapabilitiesKHR for the video profile the bound video session was created with, then VkVideoEncodeH265NaluSliceSegmentInfoKHR::constantQp **must** have the same value for each element of the pNaluSliceSegmentEntries member of the VkVideoEncodeH264PictureInfoKHR structure included in the pNext chain of pEncodeInfo

- VUID-vkCmdEncodeVideoKHR-naluSliceSegmentEntryCount-08307
  If the bound video session was created with the video codec operation VK_VIDEO_CODEC_OPERATION_ENCODE_H265_BIT_KHR, then the naluSliceSegmentEntryCount member of the VkVideoEncodeH265PictureInfoKHR structure included in the pNext chain of pEncodeInfo **must** be less than or equal to minCodingBlockExtent.width multiplied by minCodingBlockExtent.height

- VUID-vkCmdEncodeVideoKHR-naluSliceSegmentEntryCount-08313
  If the bound video session was created with the video codec operation VK_VIDEO_CODEC_OPERATION_ENCODE_H265_BIT_KHR and VkVideoEncodeH265CapabilitiesKHR::flags does not include

VK_VIDEO_ENCODE_H265_CAPABILITY_ROW_UNALIGNED_SLICE_SEGMENT_BIT_KHR, as returned by vkGetPhysicalDeviceVideoCapabilitiesKHR for the video profile the bound video session was created with, then the naluSliceSegmentEntryCount member of the VkVideoEncodeH265PictureInfoKHR structure included in the pNext chain of pEncodeInfo **must** be less than or equal to minCodingBlockExtent.height

- VUID-vkCmdEncodeVideoKHR-pNext-08354
  If the bound video session was created with the video codec operation VK_VIDEO_CODEC_OPERATION_ENCODE_H265_BIT_KHR, the pNext chain of pEncodeInfo includes a VkVideoEncodeH265PictureInfoKHR structure, and pEncodeInfo->referenceSlotCount is greater than zero, then VkVideoEncodeH265PictureInfoKHR::pStdPictureInfo->pRefLists **must** not be NULL

- VUID-vkCmdEncodeVideoKHR-pNext-08344
  If the bound video session was created with the video codec operation VK_VIDEO_CODEC_OPERATION_ENCODE_H265_BIT_KHR, the pNext chain of pEncodeInfo includes a VkVideoEncodeH265PictureInfoKHR structure, and VkVideoEncodeH265PictureInfoKHR::pStdPictureInfo->pRefLists is not NULL, then each element of the RefPicList0 and RefPicList1 array members of the StdVideoEncodeH265ReferenceListsInfo structure pointed to by VkVideoEncodeH265PictureInfoKHR::pStdPictureInfo->pRefLists **must** either be STD_VIDEO_H265_NO_REFERENCE_PICTURE or **must** equal the slotIndex member of one of the elements of pEncodeInfo->pReferenceSlots

- VUID-vkCmdEncodeVideoKHR-pNext-08355
  If the bound video session was created with the video codec operation VK_VIDEO_CODEC_OPERATION_ENCODE_H265_BIT_KHR, the pNext chain of pEncodeInfo includes a VkVideoEncodeH265PictureInfoKHR structure, and pEncodeInfo->referenceSlotCount is greater than zero, then the slotIndex member of each element of pEncodeInfo->pReferenceSlots **must** equal one of the elements of the RefPicList0 or RefPicList1 array members of the StdVideoEncodeH265ReferenceListsInfo structure pointed to by VkVideoEncodeH265PictureInfoKHR::pStdPictureInfo->pRefLists

- VUID-vkCmdEncodeVideoKHR-maxPPictureL0ReferenceCount-08345
  If the bound video session was created with the video codec operation VK_VIDEO_CODEC_OPERATION_ENCODE_H265_BIT_KHR and VkVideoEncodeH265CapabilitiesKHR::maxPPictureL0ReferenceCount is zero, as returned by vkGetPhysicalDeviceVideoCapabilitiesKHR for the video profile the bound video session was created with, then h265PictureType and each element of h265L0PictureTypes and h265L1PictureTypes **must** not be STD_VIDEO_H265_PICTURE_TYPE_P

- VUID-vkCmdEncodeVideoKHR-maxBPictureL0ReferenceCount-08346
  If the bound video session was created with the video codec operation VK_VIDEO_CODEC_OPERATION_ENCODE_H265_BIT_KHR and VkVideoEncodeH265CapabilitiesKHR::maxBPictureL0ReferenceCount and VkVideoEncodeH265CapabilitiesKHR::maxL1ReferenceCount are both zero, as returned by vkGetPhysicalDeviceVideoCapabilitiesKHR for the video profile the bound video session was created with, then h265PictureType and each element of h265L0PictureTypes and h265L1PictureTypes **must** not be STD_VIDEO_H265_PICTURE_TYPE_B

- VUID-vkCmdEncodeVideoKHR-flags-08347
  If the bound video session was created with the video codec operation

VK_VIDEO_CODEC_OPERATION_ENCODE_H265_BIT_KHR and VkVideoEncodeH265CapabilitiesKHR::flags does not include VK_VIDEO_ENCODE_H265_CAPABILITY_B_FRAME_IN_L0_LIST_BIT_KHR, as returned by vkGetPhysicalDeviceVideoCapabilitiesKHR for the video profile the bound video session was created with, then each element of h265L0PictureTypes **must** not be STD_VIDEO_H264_PICTURE_TYPE_B

- VUID-vkCmdEncodeVideoKHR-flags-08348
  If the bound video session was created with the video codec operation VK_VIDEO_CODEC_OPERATION_ENCODE_H265_BIT_KHR and VkVideoEncodeH265CapabilitiesKHR::flags does not include VK_VIDEO_ENCODE_H265_CAPABILITY_B_FRAME_IN_L1_LIST_BIT_KHR, as returned by vkGetPhysicalDeviceVideoCapabilitiesKHR for the video profile the bound video session was created with, then each element of h265L1PictureTypes **must** not be STD_VIDEO_H265_PICTURE_TYPE_B

## Valid Usage (Implicit)

- VUID-vkCmdEncodeVideoKHR-commandBuffer-parameter
  commandBuffer **must** be a valid VkCommandBuffer handle
- VUID-vkCmdEncodeVideoKHR-pEncodeInfo-parameter
  pEncodeInfo **must** be a valid pointer to a valid VkVideoEncodeInfoKHR structure
- VUID-vkCmdEncodeVideoKHR-commandBuffer-recording
  commandBuffer **must** be in the recording state
- VUID-vkCmdEncodeVideoKHR-commandBuffer-cmdpool
  The VkCommandPool that commandBuffer was allocated from **must** support encode operations
- VUID-vkCmdEncodeVideoKHR-renderpass
  This command **must** only be called outside of a render pass instance
- VUID-vkCmdEncodeVideoKHR-videocoding
  This command **must** only be called inside of a video coding scope
- VUID-vkCmdEncodeVideoKHR-bufferlevel
  commandBuffer **must** be a primary VkCommandBuffer

## Host Synchronization

- Host access to commandBuffer **must** be externally synchronized
- Host access to the VkCommandPool that commandBuffer was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary | Outside | Inside | Encode | Action |

The `VkVideoEncodeInfoKHR` structure is defined as:

```
// Provided by VK_KHR_video_encode_queue
typedef struct VkVideoEncodeInfoKHR {
    VkStructureType                         sType;
    const void*                             pNext;
    VkVideoEncodeFlagsKHR                   flags;
    VkBuffer                                dstBuffer;
    VkDeviceSize                            dstBufferOffset;
    VkDeviceSize                            dstBufferRange;
    VkVideoPictureResourceInfoKHR           srcPictureResource;
    const VkVideoReferenceSlotInfoKHR*      pSetupReferenceSlot;
    uint32_t                                referenceSlotCount;
    const VkVideoReferenceSlotInfoKHR*      pReferenceSlots;
    uint32_t                                precedingExternallyEncodedBytes;
} VkVideoEncodeInfoKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is a pointer to a structure extending this structure.
- `flags` is reserved for future use.
- `dstBuffer` is the destination video bitstream buffer to write the encoded bitstream to.
- `dstBufferOffset` is the starting offset in bytes from the start of `dstBuffer` to write the encoded bitstream to.
- `dstBufferRange` is the maximum bitstream size in bytes that **can** be written to `dstBuffer`, starting from `dstBufferOffset`.
- `srcPictureResource` is the video picture resource to use as the encode input picture.
- `pSetupReferenceSlot` is `NULL` or a pointer to a VkVideoReferenceSlotInfoKHR structure specifying the reconstructed picture information.
- `referenceSlotCount` is the number of elements in the `pReferenceSlots` array.
- `pReferenceSlots` is `NULL` or a pointer to an array of VkVideoReferenceSlotInfoKHR structures describing the DPB slots and corresponding reference picture resources to use in this video encode operation (the set of active reference pictures).
- `precedingExternallyEncodedBytes` is the number of bytes externally encoded by the application to the video bitstream and is used to update the internal state of the implementation's rate control algorithm to account for the bitrate budget consumed by these externally encoded bytes.

## Valid Usage

- VUID-VkVideoEncodeInfoKHR-dstBuffer-08236
  `dstBuffer` **must** have been created with `VK_BUFFER_USAGE_VIDEO_ENCODE_DST_BIT_KHR` set
- VUID-VkVideoEncodeInfoKHR-dstBufferOffset-08237
  `dstBufferOffset` **must** be less than the size of `dstBuffer`
- VUID-VkVideoEncodeInfoKHR-dstBufferRange-08238
  `dstBufferRange` **must** be less than or equal to the size of `dstBuffer` minus `dstBufferOffset`
- VUID-VkVideoEncodeInfoKHR-pSetupReferenceSlot-08239
  If `pSetupReferenceSlot` is not `NULL`, then its `slotIndex` member **must** not be negative
- VUID-VkVideoEncodeInfoKHR-pSetupReferenceSlot-08240
  If `pSetupReferenceSlot` is not `NULL`, then its `pPictureResource` **must** not be `NULL`
- VUID-VkVideoEncodeInfoKHR-slotIndex-08241
  The `slotIndex` member of each element of `pReferenceSlots` **must** not be negative
- VUID-VkVideoEncodeInfoKHR-pPictureResource-08242
  The `pPictureResource` member of each element of `pReferenceSlots` **must** not be `NULL`

## Valid Usage (Implicit)

- VUID-VkVideoEncodeInfoKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_VIDEO_ENCODE_INFO_KHR`
- VUID-VkVideoEncodeInfoKHR-pNext-pNext
  Each `pNext` member of any structure (including this one) in the `pNext` chain **must** be either `NULL` or a pointer to a valid instance of VkVideoEncodeH264PictureInfoKHR, VkVideoEncodeH265PictureInfoKHR, or VkVideoInlineQueryInfoKHR
- VUID-VkVideoEncodeInfoKHR-sType-unique
  The `sType` value of each struct in the `pNext` chain **must** be unique
- VUID-VkVideoEncodeInfoKHR-flags-zerobitmask
  `flags` **must** be `0`
- VUID-VkVideoEncodeInfoKHR-dstBuffer-parameter
  `dstBuffer` **must** be a valid VkBuffer handle
- VUID-VkVideoEncodeInfoKHR-srcPictureResource-parameter
  `srcPictureResource` **must** be a valid VkVideoPictureResourceInfoKHR structure
- VUID-VkVideoEncodeInfoKHR-pSetupReferenceSlot-parameter
  If `pSetupReferenceSlot` is not `NULL`, `pSetupReferenceSlot` **must** be a valid pointer to a valid VkVideoReferenceSlotInfoKHR structure
- VUID-VkVideoEncodeInfoKHR-pReferenceSlots-parameter
  If `referenceSlotCount` is not `0`, `pReferenceSlots` **must** be a valid pointer to an array of `referenceSlotCount` valid VkVideoReferenceSlotInfoKHR structures

```
// Provided by VK_KHR_video_encode_queue
typedef VkFlags VkVideoEncodeFlagsKHR;
```

VkVideoEncodeFlagsKHR is a bitmask type for setting a mask, but is currently reserved for future use.

## 42.16. Video Encode Rate Control

The size of the encoded bitstream data produced by video encode operations is a function of the following set of constraints:

- The capabilities of the compression algorithms defined and employed by the used video compression standard;
- Restrictions imposed by the selected video profile according to the rules defined by the used video compression standard;
- Further restrictions imposed by the capabilities supported by the implementation for the selected video profile;
- The image data in the encode input picture and the set of active reference pictures (as these affect the effectiveness of the compression algorithms employed by the video encode operations);
- The set of codec-specific and codec-independent encoding parameters provided by the application.

These also inherently define the set of decoder capabilities required for reconstructing and processing the picture data in the encoded bitstream.

Video coding uses *bitrate* as the quantitative metric associated with encoded bitstream data size which expresses the rate at which video bitstream data **can** be transferred or processed, measured in number of bits per second. This bitrate is both a function of the encoded bitstream data size of the encoded pictures as well as the *frame rate* used by the video sequence.

Rate control algorithms are used by video encode operations to enable adjusting encoding parameters to achieve a target bitrate, or otherwise directly or indirectly control the bitrate of the generated video bitstream data. These algorithms are usually not defined by the used video compression standard, although some video compression standards do provide non-normative guidelines for implementations.

Accordingly, this specification does not mandate implementations to produce identical encoded bitstream data outputs in response to video encode operations, however, it does define a set of codec-independent and codec-specific parameters that enable the application to control the behavior of the rate control algorithms supported by the implementation. Some of these parameters guarantee certain implementation behavior while others provide guidance for implementations to apply various rate control heuristics.

| | |
|---|---|
| **NOTE** | Applications need to make sure that they configure rate control parameters appropriately and that they follow the promises made to the implementation |

> through parameters providing guidance for the implementation's rate control algorithms and heuristics in order to be able to get the desired rate control behavior and to be able to hit the set bitrate targets. In addition, the behavior of rate control may also differ across implementations even if the capabilities of the used video profile match between those implementations. This may happen due to implementations applying different rate control algorithms or heuristics internally, and thus even the same set of guidance parameter values may have different effects on the rate control behavior across implementations.

### 42.16.1. Rate Control Modes

After a video session is reset to the initial state, the default behavior and parameters of video encode rate control are entirely implementation-dependent and the application **cannot** affect the bitrate or quality parameters of the encoded bitstream data produced by video encode operations unless the application changes the rate control configuration of the video session, as described in the Video Coding Control section.

For each supported video profile, the implementation **may** expose a set of *rate control modes* that are available for use by the application when encoding bitstreams targeting that video profile. These modes allow using different rate control algorithms that fall into one of the following two categories:

1. Per-operation rate control
2. Stream-level rate control

In case of *per-operation rate control*, the bitrate of the generated video bitstream data is indirectly controlled by quality, size, or other encoding parameters specified by the application for each individual video encode operation.

In case of *stream-level rate control*, the application **can** directly specify target bitrates besides other encoding parameters to control the behavior of the rate control algorithm used by the implementation across multiple video encode operations.

The rate control modes are defined with the following enums:

```
// Provided by VK_KHR_video_encode_queue
typedef enum VkVideoEncodeRateControlModeFlagBitsKHR {
    VK_VIDEO_ENCODE_RATE_CONTROL_MODE_DEFAULT_KHR = 0,
    VK_VIDEO_ENCODE_RATE_CONTROL_MODE_DISABLED_BIT_KHR = 0x00000001,
    VK_VIDEO_ENCODE_RATE_CONTROL_MODE_CBR_BIT_KHR = 0x00000002,
    VK_VIDEO_ENCODE_RATE_CONTROL_MODE_VBR_BIT_KHR = 0x00000004,
} VkVideoEncodeRateControlModeFlagBitsKHR;
```

- `VK_VIDEO_ENCODE_RATE_CONTROL_MODE_DEFAULT_KHR` specifies the use of implementation-specific rate control.
- `VK_VIDEO_ENCODE_RATE_CONTROL_MODE_DISABLED_BIT_KHR` specifies that rate control is disabled and the application will specify per-operation rate control parameters controlling the encoding

quality. In this mode implementations will encode pictures independently of the output bitrate of prior video encode operations.

  - When using an H.264 encode profile, implementations will use the QP value specified in VkVideoEncodeH264NaluSliceInfoKHR::constantQp to control the quality of the encoded picture.
  - When using an H.265 encode profile, implementations will use the QP value specified in VkVideoEncodeH265NaluSliceSegmentInfoKHR::constantQp to control the quality of the encoded picture.

- VK_VIDEO_ENCODE_RATE_CONTROL_MODE_CBR_BIT_KHR specifies the use of constant bitrate (CBR) rate control mode. In this mode the implementation will attempt to produce the encoded bitstream at a constant bitrate while conforming to the constraints of other rate control parameters.
- VK_VIDEO_ENCODE_RATE_CONTROL_MODE_VBR_BIT_KHR specifies the use of variable bitrate (VBR) rate control mode. In this mode the implementation will produce the encoded bitstream at a variable bitrate according to the constraints of other rate control parameters.

```
// Provided by VK_KHR_video_encode_queue
typedef VkFlags VkVideoEncodeRateControlModeFlagsKHR;
```

VkVideoEncodeRateControlModeFlagsKHR is a bitmask type for setting a mask of zero or more VkVideoEncodeRateControlModeFlagBitsKHR.

### 42.16.2. Leaky Bucket Model

Video encoding implementations use the *leaky bucket model* for stream-level rate control. The leaky bucket is a concept referring to the interface between the video encoder and the consumer (for example, a network connection), where the video encoder produces encoded bitstream data corresponding to the encoded pictures and adds them in the leaky bucket while its content are drained by the consumer.

Analogously, a similar leaky bucket is considered to exist at the input interface of a video decoder, into which encoded bitstream data is continuously added and is subsequently consumed by the video decoder. It is desirable to avoid overflowing or underflowing this leaky bucked because:

- In case of an underflow, the video decoder will be unable to consume encoded bitstream data in order to decode pictures (and optionally display them).
- In case of an overflow, the leaky bucket will be unable to accommodate more encoded bitstream data and such data **may** need to be thrown away, leading to the loss of the corresponding encoded pictures.

These requirements **can** be satisfied by imposing various constraints on the encoder-side leaky bucket to avoid its overflow or underflow, depending on the used rate control algorithm and codec parameters. However, enumerating these constraints is outside the scope of this specification.

The term *virtual buffer* is often used as an alternative to refer to the leaky bucket.

This virtual buffer model is defined by the following parameters:

- The bitrate (R) at which the encoded bitstream is expected to be processed.
- The size (B) of the virtual buffer.
- The initial occupancy (F) of the virtual buffer.

In this model the virtual buffer is used to smooth out fluctuations in the bitrate of the encoded bitstream over time without experiencing buffer overflow or underflow, as long as the bitrate of the encoded stream does not diverge from the target bitrate for extended periods of time.

This buffering **may** inherently impose a processing delay, as the goal of the model is to enable decoders maintain a consistent processing rate of an encoded bitstream with varying data rate.

The initial or start-up delay (D) is computed as:

$$D = F / R$$

| | |
|---|---|
| **NOTE** | Applications need to configure the virtual buffer with sufficient size to avoid or minimize buffer overflows and underflows while also keeping it small enough to meet their latency goals. |

### 42.16.3. Rate Control Layers

Some video compression standards and video profiles allow associating encoded pictures with specific *video coding layers*. The name, identification, and semantics associated with such video coding layers are defined by the corresponding video compression standards.

Analogously, stream-level rate control **can** be configured to use one or more *rate control layers*:

- When a single rate control layer is configured, it is applied to all encoded pictures, regardless of the picture's video coding layer. In this case the distribution of the available bitrate budget across video coding layers is implementation-dependent.
- When multiple rate control layers are configured, each rate control layer is applied to the corresponding video coding layer, i.e. only across encoded pictures pertaining to the corresponding video coding layer.

Individual rate control layers are identified using *layer indices* between zero and N-1, where N is the number of active rate control layers.

Rate control layers are only applicable when using stream-level rate control modes.

### 42.16.4. Rate Control State

Rate control state is maintained by the implementation in the video session objects and its parameters are specified using an instance of the VkVideoEncodeRateControlInfoKHR structure. The complete rate control state of a video session is defined by the following set of parameters:

- The values of the members of the VkVideoEncodeRateControlInfoKHR structure used to configure the rate control state.

- The values of the members of any VkVideoEncodeRateControlLayerInfoKHR structures specified in VkVideoEncodeRateControlInfoKHR::pLayers used to configure the state of individual rate control layers.
- If the video session was created with an H.264 encode profile:
  - The values of the members of the VkVideoEncodeH264RateControlInfoKHR structure, if one is specified in the pNext chain of the VkVideoEncodeRateControlInfoKHR used to configure the rate control state.
  - The values of the members of any VkVideoEncodeH264RateControlLayerInfoKHR structures included in the pNext chain of a VkVideoEncodeRateControlLayerInfoKHR structure used to configure the state of a rate control layer.
- If the video session was created with an H.265 encode profile:
  - The values of the members of the VkVideoEncodeH265RateControlInfoKHR structure, if one is specified in the pNext chain of the VkVideoEncodeRateControlInfoKHR used to configure the rate control state.
  - The values of the members of any VkVideoEncodeH265RateControlLayerInfoKHR structures included in the pNext chain of a VkVideoEncodeRateControlLayerInfoKHR structure used to configure the state of a rate control layer.

Two rate control states match if all the parameters listed above match between them.

The VkVideoEncodeRateControlInfoKHR structure is defined as:

```
// Provided by VK_KHR_video_encode_queue
typedef struct VkVideoEncodeRateControlInfoKHR {
    VkStructureType                                 sType;
    const void*                                     pNext;
    VkVideoEncodeRateControlFlagsKHR                flags;
    VkVideoEncodeRateControlModeFlagBitsKHR         rateControlMode;
    uint32_t                                        layerCount;
    const VkVideoEncodeRateControlLayerInfoKHR*     pLayers;
    uint32_t                                        virtualBufferSizeInMs;
    uint32_t                                        initialVirtualBufferSizeInMs;
} VkVideoEncodeRateControlInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- flags is reserved for future use.
- rateControlMode is a VkVideoEncodeRateControlModeFlagBitsKHR value specifying the rate control mode.
- layerCount specifies the number of rate control layers to use.
- pLayers is a pointer to an array of layerCount VkVideoEncodeRateControlLayerInfoKHR structures, each specifying the rate control configuration of the corresponding rate control layer.

- virtualBufferSizeInMs is the size in milliseconds of the virtual buffer used by the implementation's rate control algorithm for the leaky bucket model, with respect to the average bitrate of the stream calculated by summing the values of the averageBitrate members of the elements of the pLayers array.
- initialVirtualBufferSizeInMs is the initial occupancy in milliseconds of the virtual buffer used by the implementation's rate control algorithm for the leaky bucket model.

If layerCount is zero then the values of virtualBufferSizeInMs and initialVirtualBufferSizeInMs are ignored.

This structure **can** be specified in the following places:

- In the pNext chain of VkVideoBeginCodingInfoKHR to specify the current rate control state expected to be configured when beginning a video coding scope.
- In the pNext chain of VkVideoCodingControlInfoKHR to change the rate control configuration of the bound video session.

Including this structure in the pNext chain of VkVideoCodingControlInfoKHR and including VK_VIDEO_CODING_CONTROL_ENCODE_RATE_CONTROL_BIT_KHR in VkVideoCodingControlInfoKHR::flags enables updating the rate control configuration of the bound video session. This replaces the entire rate control configuration of the bound video session and **may** reset the state of all enabled rate control layers to an initial state according to the codec-specific rate control semantics defined in the corresponding sections listed below.

When layerCount is greater than one, multiple rate control layers are configured, and each rate control layer is applied to the corresponding video coding layer identified by the index of the corresponding element of pLayer.

- If the video session was created with the video codec operation VK_VIDEO_CODEC_OPERATION_ENCODE_H264_BIT_KHR, then this index specifies the H.264 temporal layer ID of the video coding layer the rate control layer is applied to.
- If the video session was created with the video codec operation VK_VIDEO_CODEC_OPERATION_ENCODE_H265_BIT_KHR, then this index specifies the H.265 temporal ID of the video coding layer the rate control layer is applied to.

Additional structures providing codec-specific rate control parameters **can** be included in the pNext chain of VkVideoCodingControlInfoKHR depending on the video profile the bound video session was created. For further details see:

- Video Coding Control
- H.264 Encode Rate Control
- H.265 Encode Rate Control

The new rate control configuration takes effect when the corresponding vkCmdControlVideoCodingKHR is executed on the device, and only impacts video encode operations that follow in execution order.

## Valid Usage

- VUID-VkVideoEncodeRateControlInfoKHR-rateControlMode-08248
  If `rateControlMode` is `VK_VIDEO_ENCODE_RATE_CONTROL_MODE_DEFAULT_KHR` or `VK_VIDEO_ENCODE_RATE_CONTROL_MODE_DISABLED_BIT_KHR`, then `layerCount` **must** be `0`
- VUID-VkVideoEncodeRateControlInfoKHR-rateControlMode-08275
  If `rateControlMode` is `VK_VIDEO_ENCODE_RATE_CONTROL_MODE_CBR_BIT_KHR` or `VK_VIDEO_ENCODE_RATE_CONTROL_MODE_VBR_BIT_KHR`, then `layerCount` **must** be greater than `0`
- VUID-VkVideoEncodeRateControlInfoKHR-rateControlMode-08244
  If `rateControlMode` is not `VK_VIDEO_ENCODE_RATE_CONTROL_MODE_DEFAULT_KHR`, then it **must** specify one of the bits included in VkVideoEncodeCapabilitiesKHR::`rateControlModes`, as returned by vkGetPhysicalDeviceVideoCapabilitiesKHR for the used video profile
- VUID-VkVideoEncodeRateControlInfoKHR-layerCount-08245
  `layerCount` member **must** be less than or equal to VkVideoEncodeCapabilitiesKHR::`maxRateControlLayers`, as returned by vkGetPhysicalDeviceVideoCapabilitiesKHR for the used video profile
- VUID-VkVideoEncodeRateControlInfoKHR-pLayers-08276
  For each element of `pLayers`, its `averageBitrate` member **must** be between `1` and VkVideoEncodeCapabilitiesKHR::`maxBitrate`, as returned by vkGetPhysicalDeviceVideoCapabilitiesKHR for the used video profile
- VUID-VkVideoEncodeRateControlInfoKHR-pLayers-08277
  For each element of `pLayers`, its `maxBitrate` member **must** be between `1` and VkVideoEncodeCapabilitiesKHR::`maxBitrate`, as returned by vkGetPhysicalDeviceVideoCapabilitiesKHR for the used video profile
- VUID-VkVideoEncodeRateControlInfoKHR-rateControlMode-08356
  If `rateControlMode` is `VK_VIDEO_ENCODE_RATE_CONTROL_MODE_CBR_BIT_KHR`, then for each element of `pLayers`, its `averageBitrate` member **must** equal its `maxBitrate` member
- VUID-VkVideoEncodeRateControlInfoKHR-rateControlMode-08278
  If `rateControlMode` is `VK_VIDEO_ENCODE_RATE_CONTROL_MODE_VBR_BIT_KHR`, then for each element of `pLayers`, its `averageBitrate` member **must** be less than or equal to its `maxBitrate` member
- VUID-VkVideoEncodeRateControlInfoKHR-layerCount-08357
  If `layerCount` is not zero, then `virtualBufferSizeInMs` **must** be greater than zero
- VUID-VkVideoEncodeRateControlInfoKHR-layerCount-08358
  If `layerCount` is not zero, then `initialVirtualBufferSizeInMs` **must** be less than `virtualBufferSizeInMs`
- VUID-VkVideoEncodeRateControlInfoKHR-videoCodecOperation-07022
  If the `videoCodecOperation` of the used video profile is `VK_VIDEO_CODEC_OPERATION_ENCODE_H264_BIT_KHR`, the `pNext` chain this structure is included in also includes an instance of the VkVideoEncodeH264RateControlInfoKHR structure, and `layerCount` is greater than `1`, then `layerCount` **must** equal VkVideoEncodeH264RateControlInfoKHR::`temporalLayerCount`
- VUID-VkVideoEncodeRateControlInfoKHR-videoCodecOperation-07025

If the videoCodecOperation of the used video profile is VK_VIDEO_CODEC_OPERATION_ENCODE_H265_BIT_KHR, the pNext chain this structure is included in also includes an instance of the VkVideoEncodeH265RateControlInfoKHR structure, and layerCount is greater than 1, then layerCount **must** equal VkVideoEncodeH265RateControlInfoKHR::subLayerCount

### Valid Usage (Implicit)

- VUID-VkVideoEncodeRateControlInfoKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_VIDEO_ENCODE_RATE_CONTROL_INFO_KHR
- VUID-VkVideoEncodeRateControlInfoKHR-flags-zerobitmask
  flags **must** be 0
- VUID-VkVideoEncodeRateControlInfoKHR-rateControlMode-parameter
  If rateControlMode is not 0, rateControlMode **must** be a valid VkVideoEncodeRateControlModeFlagBitsKHR value
- VUID-VkVideoEncodeRateControlInfoKHR-pLayers-parameter
  If layerCount is not 0, pLayers **must** be a valid pointer to an array of layerCount valid VkVideoEncodeRateControlLayerInfoKHR structures

```
// Provided by VK_KHR_video_encode_queue
typedef VkFlags VkVideoEncodeRateControlFlagsKHR;
```

VkVideoEncodeRateControlFlagsKHR is a bitmask type for setting a mask, but currently reserved for future use.

#### Rate Control Layer State

The configuration of individual rate control layers is specified using an instance of the VkVideoEncodeRateControlLayerInfoKHR structure.

The VkVideoEncodeRateControlLayerInfoKHR structure is defined as:

```
// Provided by VK_KHR_video_encode_queue
typedef struct VkVideoEncodeRateControlLayerInfoKHR {
    VkStructureType    sType;
    const void*        pNext;
    uint64_t           averageBitrate;
    uint64_t           maxBitrate;
    uint32_t           frameRateNumerator;
    uint32_t           frameRateDenominator;
} VkVideoEncodeRateControlLayerInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is a pointer to a structure extending this structure.

- `averageBitrate` is the average bitrate to be targeted by the implementation's rate control algorithm.
- `maxBitrate` is the peak bitrate to be targeted by the implementation's rate control algorithm.
- `frameRateNumerator` is the numerator of the frame rate assumed by the implementation's rate control algorithm.
- `frameRateDenominator` is the denominator of the frame rate assumed by the implementation's rate control algorithm.

| | |
|---|---|
| **NOTE** | The ability of the implementation's rate control algorithm to be able to match the requested average and/or peak bitrates **may** be limited by the set of other codec-independent and codec-specific rate control parameters specified by the application, the input content, as well as the application conforming to the rate control guidance provided to the implementation, as described earlier. |

Additional structures providing codec-specific rate control parameters **can** be included in the `pNext` chain of `VkVideoEncodeRateControlLayerInfoKHR` depending on the video profile the bound video session was created with. For further details see:

- Video Coding Control
- H.264 Encode Rate Control
- H.265 Encode Rate Control

### Valid Usage

- VUID-VkVideoEncodeRateControlLayerInfoKHR-frameRateNumerator-08350
  `frameRateNumerator` **must** be greater than zero
- VUID-VkVideoEncodeRateControlLayerInfoKHR-frameRateDenominator-08351
  `frameRateDenominator` **must** be greater than zero

### Valid Usage (Implicit)

- VUID-VkVideoEncodeRateControlLayerInfoKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_VIDEO_ENCODE_RATE_CONTROL_LAYER_INFO_KHR`
- VUID-VkVideoEncodeRateControlLayerInfoKHR-pNext-pNext
  Each `pNext` member of any structure (including this one) in the `pNext` chain **must** be either `NULL` or a pointer to a valid instance of VkVideoEncodeH264RateControlLayerInfoKHR or VkVideoEncodeH265RateControlLayerInfoKHR
- VUID-VkVideoEncodeRateControlLayerInfoKHR-sType-unique
  The `sType` value of each struct in the `pNext` chain **must** be unique

# 42.17. H.264 Encode Operations

Video encode operations using an H.264 encode profile **can** be used to encode elementary video stream sequences compliant to the ITU-T H.264 Specification.

| | |
|---|---|
| **NOTE** | Refer to the Preamble for information on how the Khronos Intellectual Property Rights Policy relates to normative references to external materials not created by Khronos. |

This process is performed according to the video encode operation steps with the codec-specific semantics defined in section 8 of the ITU-T H.264 Specification as follows:

- Syntax elements, derived values, and other parameters are applied from the following structures:
  - The `StdVideoH264SequenceParameterSet` structure corresponding to the active SPS specifying the H.264 sequence parameter set.
  - The `StdVideoH264PictureParameterSet` structure corresponding to the active PPS specifying the H.264 picture parameter set.
  - The `StdVideoEncodeH264PictureInfo` structure specifying the H.264 picture information.
  - The `StdVideoEncodeH264SliceHeader` structures specifying the H.264 slice header parameters for each encoded H.264 slice.
  - The `StdVideoEncodeH264ReferenceInfo` structures specifying the H.264 reference information corresponding to the optional reconstructed picture and any active reference pictures.
- The encoded bitstream data is written to the destination video bitstream buffer range as defined in the H.264 Encode Bitstream Data Access section.
- Picture data in the video picture resources corresponding to the used encode input picture, active reference pictures, and optional reconstructed picture is accessed as defined in the H.264 Encode Picture Data Access section.
- The decision on reference picture setup is made according to the parameters specified in the H.264 picture information.

If the parameters adhere to the syntactic and semantic requirements defined in the corresponding sections of the ITU-T H.264 Specification, as described above, and the DPB slots associated with the active reference pictures all refer to valid picture references, then the video encode operation will complete successfully. Otherwise, the video encode operation **may** complete unsuccessfully.

## 42.17.1. H.264 Encode Parameter Overrides

Implementations **may** override, unless otherwise specified, any of the H.264 encode parameters specified in the following Video Std structures:

- `StdVideoH264SequenceParameterSet`
- `StdVideoH264PictureParameterSet`
- `StdVideoEncodeH264PictureInfo`

- StdVideoEncodeH264SliceHeader
- StdVideoEncodeH264ReferenceInfo

All such H.264 encode parameter overrides **must** fulfill the conditions defined in the Video Encode Parameter Overrides section.

In addition, implementations **must** not override any of the following H.264 encode parameters:

- StdVideoEncodeH264PictureInfo::primary_pic_type
- StdVideoEncodeH264SliceHeader::slice_type

In case of H.264 encode parameters stored in video session parameters objects, applications need to use the vkGetEncodedVideoSessionParametersKHR command to determine whether any implementation overrides happened. If the query indicates that implementation overrides were applied, then the application needs to retrieve and use the encoded H.264 parameter sets in the bitstream in order to be able to produce a compliant H.264 video bitstream using the H.264 encode parameters stored in the video session parameters object.

In case of any H.264 encode parameters stored in the encoded bitstream produced by video encode operations, if the implementation supports the VK_VIDEO_ENCODE_FEEDBACK_BITSTREAM_HAS_OVERRIDES_BIT_KHR video encode feedback query flag, the application **can** use such queries to retrieve feedback about whether any implementation overrides have been applied to those H.264 encode parameters.

### 42.17.2. H.264 Encode Bitstream Data Access

Each video encode operation writes one or more VCL NAL units comprising of slice headers and data of the encoded picture, in the format defined in sections 7.3.3 and 7.3.4, according to the semantics defined in sections 7.4.3 and 7.4.4 of the ITU-T H.264 Specification, respectively. The number of VCL NAL units written is specified by VkVideoEncodeH264PictureInfoKHR::naluSliceEntryCount.

In addition, if VkVideoEncodeH264PictureInfoKHR::generatePrefixNalu is set to VK_TRUE for the video encode operation, then an additional prefix NAL unit is written before each VCL NAL unit corresponding to individual slices in the format defined in section 7.3.2.12, according to the semantics defined in section 7.4.2.12 of the ITU-T H.264 Specification, respectively.

### 42.17.3. H.264 Encode Picture Data Access

Accesses to image data within a video picture resource happen at the granularity indicated by VkVideoCapabilitiesKHR::pictureAccessGranularity, as returned by vkGetPhysicalDeviceVideoCapabilitiesKHR for the used video profile. Accordingly, the complete image subregion of a encode input picture, reference picture, or reconstructed picture accessed by video coding operations using an H.264 encode profile is defined as the set of texels within the coordinate range:

([0,endX), [0,endY))

Where:

- `endX` equals `codedExtent.width` rounded up to the nearest integer multiple of `pictureAccessGranularity.width` and clamped to the width of the image subresource referred to by the corresponding VkVideoPictureResourceInfoKHR structure;
- endY equals `codedExtent.height` rounded up to the nearest integer multiple of `pictureAccessGranularity.height` and clamped to the height of the image subresource referred to by the corresponding VkVideoPictureResourceInfoKHR structure;

Where `codedExtent` is the member of the VkVideoPictureResourceInfoKHR structure corresponding to the picture.

In case of video encode operations using an H.264 encode profile, any access to a picture at the coordinates (`x`,`y`), as defined by the ITU-T H.264 Specification, is an access to the image subresource referred to by the corresponding VkVideoPictureResourceInfoKHR structure at the texel coordinates (`x`,`y`).

Implementations **may** choose not to access some or all texels within particular reference pictures available to a video encode operation (e.g. due to video encode parameter overrides restricting the effective set of used reference pictures, or if the encoding algorithm chooses not to use certain subregions of the reference picture data for sample prediction).

### 42.17.4. H.264 Frame, Picture, and Slice

H.264 pictures are partitioned into slices, as defined in section 6.3 of the ITU-T H.264 Specification.

For the purposes of this specification, the H.264 slices comprising a picture are referred to as the *picture partitions* of the picture.

Video encode operations using an H.264 encode profile **can** encode slices of different types, as defined in section 7.4.3 of the ITU-T H.264 Specification, by specifying the corresponding enumeration constant value in `StdVideoEncodeH264SliceHeader::slice_type` in the H.264 slice header parameters from the Video Std enumeration type `StdVideoH264SliceType`:

- `STD_VIDEO_H264_SLICE_TYPE_P` indicates that the slice is a *P slice* as defined in section 3.109 of the ITU-T H.264 Specification.
- `STD_VIDEO_H264_SLICE_TYPE_B` indicates that the slice is a *B slice* as defined in section 3.9 of the ITU-T H.264 Specification.
- `STD_VIDEO_H264_SLICE_TYPE_I` indicates that the slice is an *I slice* as defined in section 3.66 of the ITU-T H.264 Specification.

Pictures constructed from such slices **can** be of different types, as defined in section 7.4.2.4 of the ITU-T H.264 Specification. Video encode operations using an H.264 encode profile **can** encode pictures of a specific type by specifying the corresponding enumeration constant value in `StdVideoEncodeH264PictureInfo::primary_pic_type` in the H.264 picture information from the Video Std enumeration type `StdVideoH264PictureType`:

- `STD_VIDEO_H264_PICTURE_TYPE_P` indicates that the picture is a *P picture*. A frame consisting of a P picture is also referred to as a *P frame*.

- STD_VIDEO_H264_PICTURE_TYPE_B indicates that the picture is a *B picture*. A frame consisting of a B picture is also referred to as a *B frame*.
- STD_VIDEO_H264_PICTURE_TYPE_I indicates that the picture is an *I picture*. A frame consisting of an I picture is also referred to as an *I frame*.
- STD_VIDEO_H264_PICTURE_TYPE_IDR indicates that the picture is a special type of I picture called an *IDR picture* as defined in section 3.69 of the ITU-T H.264 Specification. A frame consisting of an IDR picture is also referred to as an *IDR frame*.

### 42.17.5. H.264 Coding Blocks

H.264 encode supports a single type of coding block called a *macroblock*, as defined in section 3.84 of the ITU-T H.264 Specification.

### 42.17.6. H.264 Encode Profile

A video profile supporting H.264 video encode operations is specified by setting VkVideoProfileInfoKHR::videoCodecOperation to VK_VIDEO_CODEC_OPERATION_ENCODE_H264_BIT_KHR and adding a VkVideoEncodeH264ProfileInfoKHR structure to the VkVideoProfileInfoKHR::pNext chain.

The VkVideoEncodeH264ProfileInfoKHR structure is defined as:

```
// Provided by VK_KHR_video_encode_h264
typedef struct VkVideoEncodeH264ProfileInfoKHR {
    VkStructureType           sType;
    const void*               pNext;
    StdVideoH264ProfileIdc    stdProfileIdc;
} VkVideoEncodeH264ProfileInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- stdProfileIdc is a StdVideoH264ProfileIdc value specifying the H.264 codec profile IDC, as defined in section A.2 of the ITU-T H.264 Specification.

**Valid Usage (Implicit)**

- VUID-VkVideoEncodeH264ProfileInfoKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_VIDEO_ENCODE_H264_PROFILE_INFO_KHR

### 42.17.7. H.264 Encode Capabilities

When calling vkGetPhysicalDeviceVideoCapabilitiesKHR to query the capabilities for an H.264 encode profile, the VkVideoCapabilitiesKHR::pNext chain **must** include a VkVideoEncodeH264CapabilitiesKHR structure that will be filled with the profile-specific capabilities.

The VkVideoEncodeH264CapabilitiesKHR structure is defined as:

```
// Provided by VK_KHR_video_encode_h264
typedef struct VkVideoEncodeH264CapabilitiesKHR {
    VkStructureType                         sType;
    void*                                   pNext;
    VkVideoEncodeH264CapabilityFlagsKHR     flags;
    StdVideoH264LevelIdc                    maxLevelIdc;
    uint32_t                                maxSliceCount;
    uint32_t                                maxPPictureL0ReferenceCount;
    uint32_t                                maxBPictureL0ReferenceCount;
    uint32_t                                maxL1ReferenceCount;
    uint32_t                                maxTemporalLayerCount;
    VkBool32                                expectDyadicTemporalLayerPattern;
    int32_t                                 minQp;
    int32_t                                 maxQp;
    VkBool32                                prefersGopRemainingFrames;
    VkBool32                                requiresGopRemainingFrames;
    VkVideoEncodeH264StdFlagsKHR            stdSyntaxFlags;
} VkVideoEncodeH264CapabilitiesKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is a bitmask of VkVideoEncodeH264CapabilityFlagBitsKHR indicating supported H.264 encoding capabilities.
- `maxLevelIdc` is a `StdVideoH264LevelIdc` value indicating the maximum H.264 level supported by the profile, where enum constant `STD_VIDEO_H264_LEVEL_IDC_<major>_<minor>` identifies H.264 level `<major>.<minor>` as defined in section A.3 of the ITU-T H.264 Specification.
- `maxSliceCount` indicates the maximum number of slices that **can** be encoded for a single picture. Further restrictions **may** apply to the number of slices that **can** be encoded for a single picture depending on other capabilities and codec-specific rules.
- `maxPPictureL0ReferenceCount` indicates the maximum number of reference pictures the implementation supports in the reference list L0 for P pictures.

| | |
|---|---|
| **NOTE** | As implementations **may** override the reference lists, `maxPPictureL0ReferenceCount` does not limit the number of elements that the application **can** specify in the L0 reference list for P pictures. However, if `maxPPictureL0ReferenceCount` is zero, then the use of P pictures is not allowed. |

- `maxBPictureL0ReferenceCount` indicates the maximum number of reference pictures the implementation supports in the reference list L0 for B pictures.
- `maxL1ReferenceCount` indicates the maximum number of reference pictures the implementation supports in the reference list L1 if encoding of B pictures is supported.

| | |
|---|---|
| **NOTE** | As implementations **may** override the reference lists, `maxBPictureL0ReferenceCount` and `maxL1ReferenceCount` does not limit the number of elements that the application **can** specify in the L0 and L1 reference lists for B |

pictures. However, if maxBPictureL0ReferenceCount and maxL1ReferenceCount are both zero, then the use of B pictures is not allowed.

- maxTemporalLayerCount indicates the maximum number of H.264 temporal layers supported by the implementation.
- expectDyadicTemporalLayerPattern indicates that the implementation's rate control algorithms expect the application to use a dyadic temporal layer pattern when encoding multiple temporal layers.
- minQp indicates the minimum QP value supported.
- maxQp indicates the maximum QP value supported.
- prefersGopRemainingFrames indicates that the implementation's rate control algorithm prefers the application to specify the number of frames of each type remaining in the current group of pictures when beginning a video coding scope.
- requiresGopRemainingFrames indicates that the implementation's rate control algorithm requires the application to specify the number of frames of each type remaining in the current group of pictures when beginning a video coding scope.
- stdSyntaxFlags is a bitmask of VkVideoEncodeH264StdFlagBitsKHR indicating capabilities related to H.264 syntax elements.

### Valid Usage (Implicit)

- VUID-VkVideoEncodeH264CapabilitiesKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_VIDEO_ENCODE_H264_CAPABILITIES_KHR

Bits which **may** be set in VkVideoEncodeH264CapabilitiesKHR::flags, indicating the H.264 encoding capabilities supported, are:

```
// Provided by VK_KHR_video_encode_h264
typedef enum VkVideoEncodeH264CapabilityFlagBitsKHR {
    VK_VIDEO_ENCODE_H264_CAPABILITY_HRD_COMPLIANCE_BIT_KHR = 0x00000001,
    VK_VIDEO_ENCODE_H264_CAPABILITY_PREDICTION_WEIGHT_TABLE_GENERATED_BIT_KHR =
0x00000002,
    VK_VIDEO_ENCODE_H264_CAPABILITY_ROW_UNALIGNED_SLICE_BIT_KHR = 0x00000004,
    VK_VIDEO_ENCODE_H264_CAPABILITY_DIFFERENT_SLICE_TYPE_BIT_KHR = 0x00000008,
    VK_VIDEO_ENCODE_H264_CAPABILITY_B_FRAME_IN_L0_LIST_BIT_KHR = 0x00000010,
    VK_VIDEO_ENCODE_H264_CAPABILITY_B_FRAME_IN_L1_LIST_BIT_KHR = 0x00000020,
    VK_VIDEO_ENCODE_H264_CAPABILITY_PER_PICTURE_TYPE_MIN_MAX_QP_BIT_KHR = 0x00000040,
    VK_VIDEO_ENCODE_H264_CAPABILITY_PER_SLICE_CONSTANT_QP_BIT_KHR = 0x00000080,
    VK_VIDEO_ENCODE_H264_CAPABILITY_GENERATE_PREFIX_NALU_BIT_KHR = 0x00000100,
} VkVideoEncodeH264CapabilityFlagBitsKHR;
```

- VK_VIDEO_ENCODE_H264_CAPABILITY_HRD_COMPLIANCE_BIT_KHR indicates whether the implementation **may** be able to generate HRD compliant bitstreams if any of the nal_hrd_parameters_present_flag or vcl_hrd_parameters_present_flag members of

StdVideoH264SpsVuiFlags are set to 1 in the active SPS.

- VK_VIDEO_ENCODE_H264_CAPABILITY_PREDICTION_WEIGHT_TABLE_GENERATED_BIT_KHR indicates that if StdVideoH264PpsFlags::weighted_pred_flag is set to 1 or StdVideoH264PictureParameterSet::weighted_bipred_idc is set to STD_VIDEO_H264_WEIGHTED_BIPRED_IDC_EXPLICIT in the active PPS when encoding a P picture or B picture, respectively, then the implementation is able to internally decide syntax for pred_weight_table, as defined in section 7.4.3.2 of the ITU-T H.264 Specification, and the application is not **required** to provide a weight table in the H.264 slice header parameters.
- VK_VIDEO_ENCODE_H264_CAPABILITY_ROW_UNALIGNED_SLICE_BIT_KHR indicates that each slice in a frame with multiple slices may begin or finish at any offset in a macroblock row. If not supported, all slices in the frame **must** begin at the start of a macroblock row (and hence each slice **must** finish at the end of a macroblock row).
- VK_VIDEO_ENCODE_H264_CAPABILITY_DIFFERENT_SLICE_TYPE_BIT_KHR indicates that when a frame is encoded with multiple slices, the implementation allows encoding each slice with a different StdVideoEncodeH264SliceHeader::slice_type specified in the H.264 slice header parameters. If not supported, all slices of the frame **must** be encoded with the same slice_type which corresponds to the picture type of the frame.
- VK_VIDEO_ENCODE_H264_CAPABILITY_B_FRAME_IN_L0_LIST_BIT_KHR indicates support for using a B frame as L0 reference, as specified in StdVideoEncodeH264ReferenceListsInfo::RefPicList0 in the H.264 picture information.
- VK_VIDEO_ENCODE_H264_CAPABILITY_B_FRAME_IN_L1_LIST_BIT_KHR indicates support for using a B frame as L1 reference, as specified in StdVideoEncodeH264ReferenceListsInfo::RefPicList1 in the H.264 picture information.
- VK_VIDEO_ENCODE_H264_CAPABILITY_PER_PICTURE_TYPE_MIN_MAX_QP_BIT_KHR indicates support for specifying different QP values in the members of VkVideoEncodeH264QpKHR.
- VK_VIDEO_ENCODE_H264_CAPABILITY_PER_SLICE_CONSTANT_QP_BIT_KHR indicates support for specifying different constant QP values for each slice.
- VK_VIDEO_ENCODE_H264_CAPABILITY_GENERATE_PREFIX_NALU_BIT_KHR indicates support for generating prefix NAL units by setting VkVideoEncodeH264PictureInfoKHR::generatePrefixNalu to VK_TRUE.

```
// Provided by VK_KHR_video_encode_h264
typedef VkFlags VkVideoEncodeH264CapabilityFlagsKHR;
```

VkVideoEncodeH264CapabilityFlagsKHR is a bitmask type for setting a mask of zero or more VkVideoEncodeH264CapabilityFlagBitsKHR.

Bits which **may** be set in VkVideoEncodeH264CapabilitiesKHR::stdSyntaxFlags, indicating the capabilities related to the H.264 syntax elements, are:

```
// Provided by VK_KHR_video_encode_h264
typedef enum VkVideoEncodeH264StdFlagBitsKHR {
    VK_VIDEO_ENCODE_H264_STD_SEPARATE_COLOR_PLANE_FLAG_SET_BIT_KHR = 0x00000001,
    VK_VIDEO_ENCODE_H264_STD_QPPRIME_Y_ZERO_TRANSFORM_BYPASS_FLAG_SET_BIT_KHR =
0x00000002,
    VK_VIDEO_ENCODE_H264_STD_SCALING_MATRIX_PRESENT_FLAG_SET_BIT_KHR = 0x00000004,
    VK_VIDEO_ENCODE_H264_STD_CHROMA_QP_INDEX_OFFSET_BIT_KHR = 0x00000008,
    VK_VIDEO_ENCODE_H264_STD_SECOND_CHROMA_QP_INDEX_OFFSET_BIT_KHR = 0x00000010,
    VK_VIDEO_ENCODE_H264_STD_PIC_INIT_QP_MINUS26_BIT_KHR = 0x00000020,
    VK_VIDEO_ENCODE_H264_STD_WEIGHTED_PRED_FLAG_SET_BIT_KHR = 0x00000040,
    VK_VIDEO_ENCODE_H264_STD_WEIGHTED_BIPRED_IDC_EXPLICIT_BIT_KHR = 0x00000080,
    VK_VIDEO_ENCODE_H264_STD_WEIGHTED_BIPRED_IDC_IMPLICIT_BIT_KHR = 0x00000100,
    VK_VIDEO_ENCODE_H264_STD_TRANSFORM_8X8_MODE_FLAG_SET_BIT_KHR = 0x00000200,
    VK_VIDEO_ENCODE_H264_STD_DIRECT_SPATIAL_MV_PRED_FLAG_UNSET_BIT_KHR = 0x00000400,
    VK_VIDEO_ENCODE_H264_STD_ENTROPY_CODING_MODE_FLAG_UNSET_BIT_KHR = 0x00000800,
    VK_VIDEO_ENCODE_H264_STD_ENTROPY_CODING_MODE_FLAG_SET_BIT_KHR = 0x00001000,
    VK_VIDEO_ENCODE_H264_STD_DIRECT_8X8_INFERENCE_FLAG_UNSET_BIT_KHR = 0x00002000,
    VK_VIDEO_ENCODE_H264_STD_CONSTRAINED_INTRA_PRED_FLAG_SET_BIT_KHR = 0x00004000,
    VK_VIDEO_ENCODE_H264_STD_DEBLOCKING_FILTER_DISABLED_BIT_KHR = 0x00008000,
    VK_VIDEO_ENCODE_H264_STD_DEBLOCKING_FILTER_ENABLED_BIT_KHR = 0x00010000,
    VK_VIDEO_ENCODE_H264_STD_DEBLOCKING_FILTER_PARTIAL_BIT_KHR = 0x00020000,
    VK_VIDEO_ENCODE_H264_STD_SLICE_QP_DELTA_BIT_KHR = 0x00080000,
    VK_VIDEO_ENCODE_H264_STD_DIFFERENT_SLICE_QP_DELTA_BIT_KHR = 0x00100000,
} VkVideoEncodeH264StdFlagBitsKHR;
```

- `VK_VIDEO_ENCODE_H264_STD_SEPARATE_COLOR_PLANE_FLAG_SET_BIT_KHR` indicates whether the implementation supports using the application-provided value for `StdVideoH264SpsFlags::separate_colour_plane_flag` in the SPS when that value is `1`.
- `VK_VIDEO_ENCODE_H264_STD_QPPRIME_Y_ZERO_TRANSFORM_BYPASS_FLAG_SET_BIT_KHR` indicates whether the implementation supports using the application-provided value for `StdVideoH264SpsFlags::qpprime_y_zero_transform_bypass_flag` in the SPS when that value is `1`.
- `VK_VIDEO_ENCODE_H264_STD_SCALING_MATRIX_PRESENT_FLAG_SET_BIT_KHR` indicates whether the implementation supports using the application-provided values for `StdVideoH264SpsFlags::seq_scaling_matrix_present_flag` in the SPS and `StdVideoH264PpsFlags::pic_scaling_matrix_present_flag` in the PPS when any of those values are `1`.
- `VK_VIDEO_ENCODE_H264_STD_CHROMA_QP_INDEX_OFFSET_BIT_KHR` indicates whether the implementation supports using the application-provided value for `StdVideoH264PictureParameterSet::chroma_qp_index_offset` in the PPS when that value is non-zero.
- `VK_VIDEO_ENCODE_H264_STD_SECOND_CHROMA_QP_INDEX_OFFSET_BIT_KHR` indicates whether the implementation supports using the application-provided value for `StdVideoH264PictureParameterSet::second_chroma_qp_index_offset` in the PPS when that value is non-zero.
- `VK_VIDEO_ENCODE_H264_STD_PIC_INIT_QP_MINUS26_BIT_KHR` indicates whether the implementation supports using the application-provided value for `StdVideoH264PictureParameterSet::pic_init_qp_minus26` in the PPS when that value is non-zero.

- VK_VIDEO_ENCODE_H264_STD_WEIGHTED_PRED_FLAG_SET_BIT_KHR indicates whether the implementation supports using the application-provided value for StdVideoH264PpsFlags::weighted_pred_flag in the PPS when that value is 1.
- VK_VIDEO_ENCODE_H264_STD_WEIGHTED_BIPRED_IDC_EXPLICIT_BIT_KHR indicates whether the implementation supports using the application-provided value for StdVideoH264PictureParameterSet::weighted_bipred_idc in the PPS when that value is STD_VIDEO_H264_WEIGHTED_BIPRED_IDC_EXPLICIT.
- VK_VIDEO_ENCODE_H264_STD_WEIGHTED_BIPRED_IDC_IMPLICIT_BIT_KHR indicates whether the implementation supports using the application-provided value for StdVideoH264PictureParameterSet::weighted_bipred_idc in the PPS when that value is STD_VIDEO_H264_WEIGHTED_BIPRED_IDC_IMPLICIT.
- VK_VIDEO_ENCODE_H264_STD_TRANSFORM_8X8_MODE_FLAG_SET_BIT_KHR indicates whether the implementation supports using the application-provided value for StdVideoH264PpsFlags::transform_8x8_mode_flag in the PPS when that value is 1.
- VK_VIDEO_ENCODE_H264_STD_DIRECT_SPATIAL_MV_PRED_FLAG_UNSET_BIT_KHR indicates whether the implementation supports using the application-provided value for StdVideoEncodeH264SliceHeaderFlags::direct_spatial_mv_pred_flag in the H.264 slice header parameters when that value is 0.
- VK_VIDEO_ENCODE_H264_STD_ENTROPY_CODING_MODE_FLAG_UNSET_BIT_KHR indicates whether the implementation supports CAVLC entropy coding, as defined in section 9.2 of the ITU-T H.264 Specification, and thus supports using the application-provided value for StdVideoH264PpsFlags::entropy_coding_mode_flag in the PPS when that value is 0.
- VK_VIDEO_ENCODE_H264_STD_ENTROPY_CODING_MODE_FLAG_SET_BIT_KHR indicates whether the implementation supports CABAC entropy coding, as defined in section 9.3 of the ITU-T H.264 Specification, and thus supports using the application-provided value for StdVideoH264PpsFlags::entropy_coding_mode_flag in the PPS when that value is 1.
- VK_VIDEO_ENCODE_H264_STD_DIRECT_8X8_INFERENCE_FLAG_UNSET_BIT_KHR indicates whether the implementation supports using the application-provided value for StdVideoH264SpsFlags::direct_8x8_inference_flag in the SPS when that value is 0.
- VK_VIDEO_ENCODE_H264_STD_CONSTRAINED_INTRA_PRED_FLAG_SET_BIT_KHR indicates whether the implementation supports using the application-provided value for StdVideoH264PpsFlags::constrained_intra_pred_flag in the PPS when that value is 1.
- VK_VIDEO_ENCODE_H264_STD_DEBLOCKING_FILTER_DISABLED_BIT_KHR indicates whether the implementation supports using the application-provided value for StdVideoEncodeH264SliceHeader::disable_deblocking_filter_idc in the H.264 slice header parameters when that value is STD_VIDEO_H264_DISABLE_DEBLOCKING_FILTER_IDC_DISABLED.
- VK_VIDEO_ENCODE_H264_STD_DEBLOCKING_FILTER_ENABLED_BIT_KHR indicates whether the implementation supports using the application-provided value for StdVideoEncodeH264SliceHeader::disable_deblocking_filter_idc in the H.264 slice header parameters when that value is STD_VIDEO_H264_DISABLE_DEBLOCKING_FILTER_IDC_ENABLED.
- VK_VIDEO_ENCODE_H264_STD_DEBLOCKING_FILTER_PARTIAL_BIT_KHR indicates whether the implementation supports using the application-provided value for StdVideoEncodeH264SliceHeader::disable_deblocking_filter_idc in the H.264 slice header

parameters when that value is STD_VIDEO_H264_DISABLE_DEBLOCKING_FILTER_IDC_PARTIAL.

- VK_VIDEO_ENCODE_H264_STD_SLICE_QP_DELTA_BIT_KHR indicates whether the implementation supports using the application-provided value for StdVideoEncodeH264SliceHeader::slice_qp_delta in the H.264 slice header parameters when that value is identical across the slices of the encoded frame.
- VK_VIDEO_ENCODE_H264_STD_DIFFERENT_SLICE_QP_DELTA_BIT_KHR indicates whether the implementation supports using the application-provided value for StdVideoEncodeH264SliceHeader::slice_qp_delta in the H.264 slice header parameters when that value is different across the slices of the encoded frame.

These capability flags provide information to the application about specific H.264 syntax element values that the implementation supports without having to override them and do not otherwise restrict the values that the application **can** specify for any of the mentioned H.264 syntax elements.

```
// Provided by VK_KHR_video_encode_h264
typedef VkFlags VkVideoEncodeH264StdFlagsKHR;
```

VkVideoEncodeH264StdFlagsKHR is a bitmask type for setting a mask of zero or more VkVideoEncodeH264StdFlagBitsKHR.

### 42.17.8. H.264 Encode Quality Level Properties

When calling vkGetPhysicalDeviceVideoEncodeQualityLevelPropertiesKHR with pVideoProfile->videoCodecOperation specified as VK_VIDEO_CODEC_OPERATION_ENCODE_H264_BIT_KHR, the VkVideoEncodeH264QualityLevelPropertiesKHR structure **must** be included in the pNext chain of the VkVideoEncodeQualityLevelPropertiesKHR structure to retrieve additional video encode quality level properties specific to H.264 encoding.

The VkVideoEncodeH264QualityLevelPropertiesKHR structure is defined as:

```
// Provided by VK_KHR_video_encode_h264
typedef struct VkVideoEncodeH264QualityLevelPropertiesKHR {
    VkStructureType                         sType;
    void*                                   pNext;
    VkVideoEncodeH264RateControlFlagsKHR    preferredRateControlFlags;
    uint32_t                                preferredGopFrameCount;
    uint32_t                                preferredIdrPeriod;
    uint32_t                                preferredConsecutiveBFrameCount;
    uint32_t                                preferredTemporalLayerCount;
    VkVideoEncodeH264QpKHR                  preferredConstantQp;
    uint32_t                                preferredMaxL0ReferenceCount;
    uint32_t                                preferredMaxL1ReferenceCount;
    VkBool32                                preferredStdEntropyCodingModeFlag;
} VkVideoEncodeH264QualityLevelPropertiesKHR;
```

- sType is a VkStructureType value identifying this structure.

- pNext is NULL or a pointer to a structure extending this structure.
- preferredRateControlFlags is a bitmask of VkVideoEncodeH264RateControlFlagBitsKHR values indicating the preferred flags to use for VkVideoEncodeH264RateControlInfoKHR::flags.
- preferredGopFrameCount indicates the preferred value to use for VkVideoEncodeH264RateControlInfoKHR::gopFrameCount.
- preferredIdrPeriod indicates the preferred value to use for VkVideoEncodeH264RateControlInfoKHR::idrPeriod.
- preferredConsecutiveBFrameCount indicates the preferred value to use for VkVideoEncodeH264RateControlInfoKHR::consecutiveBFrameCount.
- preferredTemporalLayerCount indicates the preferred value to use for VkVideoEncodeH264RateControlInfoKHR::temporalLayerCount.
- preferredConstantQp indicates the preferred values to use for VkVideoEncodeH264NaluSliceInfoKHR::constantQp for each picture type when using rate control mode VK_VIDEO_ENCODE_RATE_CONTROL_MODE_DISABLED_BIT_KHR.
- preferredMaxL0ReferenceCount indicates the preferred maximum number of reference pictures to use in the reference list L0.
- preferredMaxL1ReferenceCount indicates the preferred maximum number of reference pictures to use in the reference list L1.
- preferredStdEntropyCodingModeFlag indicates the preferred value to use for entropy_coding_mode_flag in StdVideoH264PpsFlags.

### Valid Usage (Implicit)

- VUID-VkVideoEncodeH264QualityLevelPropertiesKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_VIDEO_ENCODE_H264_QUALITY_LEVEL_PROPERTIES_KHR

## 42.17.9. H.264 Encode Session

Additional parameters can be specified when creating a video session with an H.264 encode profile by including an instance of the VkVideoEncodeH264SessionCreateInfoKHR structure in the pNext chain of VkVideoSessionCreateInfoKHR.

The VkVideoEncodeH264SessionCreateInfoKHR structure is defined as:

```
// Provided by VK_KHR_video_encode_h264
typedef struct VkVideoEncodeH264SessionCreateInfoKHR {
    VkStructureType         sType;
    const void*             pNext;
    VkBool32                useMaxLevelIdc;
    StdVideoH264LevelIdc    maxLevelIdc;
} VkVideoEncodeH264SessionCreateInfoKHR;
```

- sType is a VkStructureType value identifying this structure.

- pNext is NULL or a pointer to a structure extending this structure.
- useMaxLevelIdc indicates whether the value of maxLevelIdc should be used by the implementation. When it is set to VK_FALSE, the implementation ignores the value of maxLevelIdc and uses the value of VkVideoEncodeH264CapabilitiesKHR::maxLevelIdc, as reported by vkGetPhysicalDeviceVideoCapabilitiesKHR for the video profile.
- maxLevelIdc is a StdVideoH264LevelIdc value specifying the upper bound on the H.264 level for the video bitstreams produced by the created video session, where enum constant STD_VIDEO_H264_LEVEL_IDC_<major>_<minor> identifies H.264 level <major>.<minor> as defined in section A.3 of the ITU-T H.264 Specification.

### Valid Usage (Implicit)

- VUID-VkVideoEncodeH264SessionCreateInfoKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_VIDEO_ENCODE_H264_SESSION_CREATE_INFO_KHR

## 42.17.10. H.264 Encode Parameter Sets

Video session parameters objects created with the video codec operation VK_VIDEO_CODEC_OPERATION_ENCODE_H264_BIT_KHR **can** contain the following types of parameters:

**H.264 Sequence Parameter Sets (SPS)**

Represented by StdVideoH264SequenceParameterSet structures and interpreted as follows:

- reserved1 and reserved2 are used only for padding purposes and are otherwise ignored;
- seq_parameter_set_id is used as the key of the SPS entry;
- level_idc is one of the enum constants STD_VIDEO_H264_LEVEL_IDC_<major>_<minor> identifying the H.264 level <major>.<minor> as defined in section A.3 of the ITU-T H.264 Specification;
- if flags.seq_scaling_matrix_present_flag is set, then the StdVideoH264ScalingLists structure pointed to by pScalingLists is interpreted as follows:
  - scaling_list_present_mask is a bitmask where bit index i corresponds to seq_scaling_list_present_flag[i] as defined in section 7.4.2.1 of the ITU-T H.264 Specification;
  - use_default_scaling_matrix_mask is a bitmask where bit index i corresponds to UseDefaultScalingMatrix4x4Flag[i], when i < 6, or corresponds to UseDefaultScalingMatrix8x8Flag[i-6], otherwise, as defined in section 7.3.2.1 of the ITU-T H.264 Specification;
  - ScalingList4x4 and ScalingList8x8 correspond to the identically named syntax elements defined in section 7.3.2.1 of the ITU-T H.264 Specification;
- if flags.vui_parameters_present_flag is set, then pSequenceParameterSetVui is a pointer to a StdVideoH264SequenceParameterSetVui structure that is interpreted as follows:
  - reserved1 is used only for padding purposes and is otherwise ignored;

    - if `flags.nal_hrd_parameters_present_flag` or `flags.vcl_hrd_parameters_present_flag` is set, then the `StdVideoH264HrdParameters` structure pointed to by `pHrdParameters` is interpreted as follows:
      - `reserved1` is used only for padding purposes and is otherwise ignored;
      - all other members of `StdVideoH264HrdParameters` are interpreted as defined in section E.2.2 of the ITU-T H.264 Specification;
    - all other members of `StdVideoH264SequenceParameterSetVui` are interpreted as defined in section E.2.1 of the ITU-T H.264 Specification;
  - all other members of `StdVideoH264SequenceParameterSet` are interpreted as defined in section 7.4.2.1 of the ITU-T H.264 Specification.

**H.264 Picture Parameter Sets (PPS)**

Represented by `StdVideoH264PictureParameterSet` structures and interpreted as follows:

- the pair constructed from `seq_parameter_set_id` and `pic_parameter_set_id` is used as the key of the PPS entry;
- if `flags.pic_scaling_matrix_present_flag` is set, then the `StdVideoH264ScalingLists` structure pointed to by `pScalingLists` is interpreted as follows:
  - `scaling_list_present_mask` is a bitmask where bit index i corresponds to `pic_scaling_list_present_flag[i]` as defined in section 7.4.2.2 of the ITU-T H.264 Specification;
  - `use_default_scaling_matrix_mask` is a bitmask where bit index i corresponds to `UseDefaultScalingMatrix4x4Flag[i]`, when i < 6, or corresponds to `UseDefaultScalingMatrix8x8Flag[i-6]`, otherwise, as defined in section 7.3.2.2 of the ITU-T H.264 Specification;
  - `ScalingList4x4` and `ScalingList8x8` correspond to the identically named syntax elements defined in section 7.3.2.2 of the ITU-T H.264 Specification;
- all other members of `StdVideoH264PictureParameterSet` are interpreted as defined in section 7.4.2.2 of the ITU-T H.264 Specification.

Implementations **may** override any of these parameters according to the semantics defined in the Video Encode Parameter Overrides section before storing the resulting H.264 parameter sets into the video session parameters object. Applications need to use the vkGetEncodedVideoSessionParametersKHR command to determine whether any implementation overrides happened and to retrieve the encoded H.264 parameter sets in order to be able to produce a compliant H.264 video bitstream.

Such H.264 parameter set overrides **may** also have cascading effects on the implementation overrides applied to the encoded bitstream produced by video encode operations. If the implementation supports the `VK_VIDEO_ENCODE_FEEDBACK_BITSTREAM_HAS_OVERRIDES_BIT_KHR` video encode feedback query flag, then the application **can** use such queries to retrieve feedback about whether any implementation overrides have been applied to the encoded bitstream.

When a video session parameters object is created with the codec operation VK_VIDEO_CODEC_OPERATION_ENCODE_H264_BIT_KHR, the VkVideoSessionParametersCreateInfoKHR::pNext chain **must** include a VkVideoEncodeH264SessionParametersCreateInfoKHR structure specifying the capacity and initial contents of the object.

The VkVideoEncodeH264SessionParametersCreateInfoKHR structure is defined as:

```
// Provided by VK_KHR_video_encode_h264
typedef struct VkVideoEncodeH264SessionParametersCreateInfoKHR {
    VkStructureType                                         sType;
    const void*                                             pNext;
    uint32_t                                                maxStdSPSCount;
    uint32_t                                                maxStdPPSCount;
    const VkVideoEncodeH264SessionParametersAddInfoKHR*     pParametersAddInfo;
} VkVideoEncodeH264SessionParametersCreateInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- maxStdSPSCount is the maximum number of H.264 SPS entries the created VkVideoSessionParametersKHR **can** contain.
- maxStdPPSCount is the maximum number of H.264 PPS entries the created VkVideoSessionParametersKHR **can** contain.
- pParametersAddInfo is NULL or a pointer to a VkVideoEncodeH264SessionParametersAddInfoKHR structure specifying H.264 parameters to add upon object creation.

### Valid Usage (Implicit)

- VUID-VkVideoEncodeH264SessionParametersCreateInfoKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_VIDEO_ENCODE_H264_SESSION_PARAMETERS_CREATE_INFO_KHR
- VUID-VkVideoEncodeH264SessionParametersCreateInfoKHR-pParametersAddInfo-parameter
  If pParametersAddInfo is not NULL, pParametersAddInfo **must** be a valid pointer to a valid VkVideoEncodeH264SessionParametersAddInfoKHR structure

The VkVideoEncodeH264SessionParametersAddInfoKHR structure is defined as:

```
// Provided by VK_KHR_video_encode_h264
typedef struct VkVideoEncodeH264SessionParametersAddInfoKHR {
    VkStructureType                                 sType;
    const void*                                     pNext;
    uint32_t                                        stdSPSCount;
    const StdVideoH264SequenceParameterSet*         pStdSPSs;
    uint32_t                                        stdPPSCount;
    const StdVideoH264PictureParameterSet*          pStdPPSs;
} VkVideoEncodeH264SessionParametersAddInfoKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `stdSPSCount` is the number of elements in the `pStdSPSs` array.
- `pStdSPSs` is a pointer to an array of `StdVideoH264SequenceParameterSet` structures describing the H.264 SPS entries to add.
- `stdPPSCount` is the number of elements in the `pStdPPSs` array.
- `pStdPPSs` is a pointer to an array of `StdVideoH264PictureParameterSet` structures describing the H.264 PPS entries to add.

This structure **can** be specified in the following places:

- In the `pParametersAddInfo` member of the VkVideoEncodeH264SessionParametersCreateInfoKHR structure specified in the `pNext` chain of VkVideoSessionParametersCreateInfoKHR used to create a video session parameters object. In this case, if the video codec operation the video session parameters object is created with is `VK_VIDEO_CODEC_OPERATION_ENCODE_H264_BIT_KHR`, then it defines the set of initial parameters to add to the created object (see Creating Video Session Parameters).
- In the `pNext` chain of VkVideoSessionParametersUpdateInfoKHR. In this case, if the video codec operation the video session parameters object to be updated was created with is `VK_VIDEO_CODEC_OPERATION_ENCODE_H264_BIT_KHR`, then it defines the set of parameters to add to it (see Updating Video Session Parameters).

## Valid Usage

- VUID-VkVideoEncodeH264SessionParametersAddInfoKHR-None-04837
  The `seq_parameter_set_id` member of each `StdVideoH264SequenceParameterSet` structure specified in the elements of `pStdSPSs` **must** be unique within `pStdSPSs`
- VUID-VkVideoEncodeH264SessionParametersAddInfoKHR-None-04838
  The pair constructed from the `seq_parameter_set_id` and `pic_parameter_set_id` members of each `StdVideoH264PictureParameterSet` structure specified in the elements of `pStdPPSs` **must** be unique within `pStdPPSs`

### Valid Usage (Implicit)

- VUID-VkVideoEncodeH264SessionParametersAddInfoKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_VIDEO_ENCODE_H264_SESSION_PARAMETERS_ADD_INFO_KHR`
- VUID-VkVideoEncodeH264SessionParametersAddInfoKHR-pStdSPSs-parameter
  If `stdSPSCount` is not `0`, and `pStdSPSs` is not `NULL`, `pStdSPSs` **must** be a valid pointer to an array of `stdSPSCount` `StdVideoH264SequenceParameterSet` values
- VUID-VkVideoEncodeH264SessionParametersAddInfoKHR-pStdPPSs-parameter
  If `stdPPSCount` is not `0`, and `pStdPPSs` is not `NULL`, `pStdPPSs` **must** be a valid pointer to an array of `stdPPSCount` `StdVideoH264PictureParameterSet` values

The `VkVideoEncodeH264SessionParametersGetInfoKHR` structure is defined as:

```
// Provided by VK_KHR_video_encode_h264
typedef struct VkVideoEncodeH264SessionParametersGetInfoKHR {
    VkStructureType    sType;
    const void*        pNext;
    VkBool32           writeStdSPS;
    VkBool32           writeStdPPS;
    uint32_t           stdSPSId;
    uint32_t           stdPPSId;
} VkVideoEncodeH264SessionParametersGetInfoKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `writeStdSPS` indicates whether the encoded H.264 sequence parameter set identified by `stdSPSId` is requested to be retrieved.
- `writeStdPPS` indicates whether the encoded H.264 picture parameter set identified by the pair constructed from `stdSPSId` and `stdPPSId` is requested to be retrieved.
- `stdSPSId` specifies the H.264 sequence parameter set ID used to identify the retrieved H.264 sequence and/or picture parameter set(s).
- `stdPPSId` specifies the H.264 picture parameter set ID used to identify the retrieved H.264 picture parameter set when `writeStdPPS` is set to `VK_TRUE`.

When this structure is specified in the `pNext` chain of the VkVideoEncodeSessionParametersGetInfoKHR structure passed to vkGetEncodedVideoSessionParametersKHR, the command will write encoded parameter data to the output buffer in the following order:

1. The H.264 sequence parameter set identified by `stdSPSId`, if `writeStdSPS` is set to `VK_TRUE`.
2. The H.264 picture parameter set identified by the pair constructed from `stdSPSId` and `stdPPSId`, if `writeStdPPS` is set to `VK_TRUE`.

### Valid Usage

- VUID-VkVideoEncodeH264SessionParametersGetInfoKHR-writeStdSPS-08279
  At least one of `writeStdSPS` and `writeStdPPS` **must** be `VK_TRUE`

### Valid Usage (Implicit)

- VUID-VkVideoEncodeH264SessionParametersGetInfoKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_VIDEO_ENCODE_H264_SESSION_PARAMETERS_GET_INFO_KHR`

The `VkVideoEncodeH264SessionParametersFeedbackInfoKHR` structure is defined as:

```
// Provided by VK_KHR_video_encode_h264
typedef struct VkVideoEncodeH264SessionParametersFeedbackInfoKHR {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           hasStdSPSOverrides;
    VkBool32           hasStdPPSOverrides;
} VkVideoEncodeH264SessionParametersFeedbackInfoKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `hasStdSPSOverrides` indicates whether any of the parameters of the requested H.264 sequence parameter set, if one was requested via VkVideoEncodeH264SessionParametersGetInfoKHR::`writeStdSPS`, were overridden by the implementation.
- `hasStdPPSOverrides` indicates whether any of the parameters of the requested H.264 picture parameter set, if one was requested via VkVideoEncodeH264SessionParametersGetInfoKHR::`writeStdPPS`, were overridden by the implementation.

### Valid Usage (Implicit)

- VUID-VkVideoEncodeH264SessionParametersFeedbackInfoKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_VIDEO_ENCODE_H264_SESSION_PARAMETERS_FEEDBACK_INFO_KHR`

## 42.17.11. H.264 Encoding Parameters

The VkVideoEncodeH264PictureInfoKHR structure is defined as:

```
// Provided by VK_KHR_video_encode_h264
typedef struct VkVideoEncodeH264PictureInfoKHR {
    VkStructureType                                 sType;
    const void*                                     pNext;
    uint32_t                                        naluSliceEntryCount;
    const VkVideoEncodeH264NaluSliceInfoKHR*        pNaluSliceEntries;
    const StdVideoEncodeH264PictureInfo*            pStdPictureInfo;
    VkBool32                                        generatePrefixNalu;
} VkVideoEncodeH264PictureInfoKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `naluSliceEntryCount` is the number of elements in `pNaluSliceEntries`.
- `pNaluSliceEntries` is a pointer to an array of `naluSliceEntryCount` VkVideoEncodeH264NaluSliceInfoKHR structures specifying the parameters of the individual H.264 slices to encode for the input picture.
- `pStdPictureInfo` is a pointer to a `StdVideoEncodeH264PictureInfo` structure specifying H.264 picture information.
- `generatePrefixNalu` controls whether prefix NALUs are generated before slice NALUs into the target bitstream, as defined in sections 7.3.2.12 and 7.4.2.12 of the ITU-T H.264 Specification.

This structure is specified in the `pNext` chain of the VkVideoEncodeInfoKHR structure passed to vkCmdEncodeVideoKHR to specify the codec-specific picture information for an H.264 encode operation.

**Encode Input Picture Information**

When this structure is specified in the `pNext` chain of the VkVideoEncodeInfoKHR structure passed to vkCmdEncodeVideoKHR, the information related to the encode input picture is defined as follows:

- The image subregion used is determined according to the H.264 Encode Picture Data Access section.
- The encode input picture is associated with the H.264 picture information provided in `pStdPictureInfo`.

**Std Picture Information**

The members of the `StdVideoEncodeH264PictureInfo` structure pointed to by `pStdPictureInfo` are interpreted as follows:

- `flags.reserved` and `reserved1` are used only for padding purposes and are otherwise ignored;
- `flags.IdrPicFlag` as defined in section 7.4.1 of the ITU-T H.264 Specification;
- `flags.is_reference` as defined in section 3.136 of the ITU-T H.264 Specification;
- `seq_parameter_set_id` and `pic_parameter_set_id` are used to identify the active parameter sets, as described below;

- primary_pic_type as defined in section 7.4.2 of the ITU-T H.264 Specification;
- PicOrderCnt as defined in section 8.2 of the ITU-T H.264 Specification;
- temporal_id as defined in section G.7.4.1.1 of the ITU-T H.264 Specification;
- if pRefLists is not NULL, then it is a pointer to a StdVideoEncodeH264ReferenceListsInfo structure that is interpreted as follows:
  - flags.reserved is used only for padding purposes and is otherwise ignored;
  - ref_pic_list_modification_flag_l0 and ref_pic_list_modification_flag_l1 as defined in section 7.4.3.1 of the ITU-T H.264 Specification;
  - num_ref_idx_l0_active_minus1 and num_ref_idx_l1_active_minus1 as defined in section 7.4.3 of the ITU-T H.264 Specification;
  - RefPicList0 and RefPicList1 as defined in section 8.2.4 of the ITU-T H.264 Specification where each element of these arrays either identifies an active reference picture using its DPB slot index or contains the value STD_VIDEO_H264_NO_REFERENCE_PICTURE to indicate "no reference picture";
  - if refList0ModOpCount is not zero, then pRefList0ModOperations is a pointer to an array of refList0ModOpCount number of StdVideoEncodeH264RefListModEntry structures specifying the modification parameters for the reference list L0 as defined in section 7.4.3.1 of the ITU-T H.264 Specification;
  - if refList1ModOpCount is not zero, then pRefList1ModOperations is a pointer to an array of refList1ModOpCount number of StdVideoEncodeH264RefListModEntry structures specifying the modification parameters for the reference list L1 as defined in section 7.4.3.1 of the ITU-T H.264 Specification;
  - if refPicMarkingOpCount is not zero, then refPicMarkingOperations is a pointer to an array of refPicMarkingOpCount number of StdVideoEncodeH264RefPicMarkingEntry structures specifying the reference picture marking parameters as defined in section 7.4.3.3 of the ITU-T H.264 Specification;
- all other members are interpreted as defined in section 7.4.3 of the ITU-T H.264 Specification.

Reference picture setup is controlled by the value of StdVideoEncodeH264PictureInfo::flags.is_reference. If it is set and a reconstructed picture is specified, then the latter is used as the target of picture reconstruction to activate the DPB slot specified in pEncodeInfo->pSetupReferenceSlot->slotIndex. If StdVideoEncodeH264PictureInfo::flags.is_reference is not set, but a reconstructed picture is specified, then the corresponding picture reference associated with the DPB slot is invalidated, as described in the DPB Slot States section.

#### Active Parameter Sets

The members of the StdVideoEncodeH264PictureInfo structure pointed to by pStdPictureInfo are used to select the active parameter sets to use from the bound video session parameters object, as follows:

- The *active SPS* is the SPS identified by the key specified in StdVideoEncodeH264PictureInfo::seq_parameter_set_id.
- The *active PPS* is the PPS identified by the key specified by the pair constructed from StdVideoEncodeH264PictureInfo::seq_parameter_set_id and StdVideoEncodeH264PictureInfo

::pic_parameter_set_id.

H.264 encoding uses *explicit weighted sample prediction* for a slice, as defined in section 8.4.2.3 of the ITU-T H.264 Specification, if any of the following conditions are true for the active PPS and the pStdSliceHeader member of the corresponding element of pNaluSliceEntries:

- pStdSliceHeader->slice_type is STD_VIDEO_H264_SLICE_TYPE_P and weighted_pred_flag is enabled in the active PPS.
- pStdSliceHeader->slice_type is STD_VIDEO_H264_SLICE_TYPE_B and weighted_bipred_idc in the active PPS equals STD_VIDEO_H264_WEIGHTED_BIPRED_IDC_EXPLICIT.

## Valid Usage

- VUID-VkVideoEncodeH264PictureInfoKHR-naluSliceEntryCount-08301
  naluSliceEntryCount **must** be between 1 and VkVideoEncodeH264CapabilitiesKHR::maxSliceCount, inclusive, as returned by vkGetPhysicalDeviceVideoCapabilitiesKHR for the used video profile
- VUID-VkVideoEncodeH264PictureInfoKHR-flags-08304
  If VkVideoEncodeH264CapabilitiesKHR::flags, as returned by vkGetPhysicalDeviceVideoCapabilitiesKHR for the used video profile, does not include VK_VIDEO_ENCODE_H264_CAPABILITY_GENERATE_PREFIX_NALU_BIT_KHR, then generatePrefixNalu **must** be VK_FALSE
- VUID-VkVideoEncodeH264PictureInfoKHR-flags-08314
  If VkVideoEncodeH264CapabilitiesKHR::flags, as returned by vkGetPhysicalDeviceVideoCapabilitiesKHR for the used video profile, does not include VK_VIDEO_ENCODE_H264_CAPABILITY_PREDICTION_WEIGHT_TABLE_GENERATED_BIT_KHR and the slice corresponding to any element of pNaluSliceEntries uses explicit weighted sample prediction, then VkVideoEncodeH264NaluSliceInfoKHR::pStdSliceHeader->pWeightTable **must** not be NULL for that element of pNaluSliceEntries
- VUID-VkVideoEncodeH264PictureInfoKHR-flags-08315
  If VkVideoEncodeH264CapabilitiesKHR::flags, as returned by vkGetPhysicalDeviceVideoCapabilitiesKHR for the used video profile, does not include VK_VIDEO_ENCODE_H264_CAPABILITY_DIFFERENT_SLICE_TYPE_BIT_KHR, then VkVideoEncodeH264NaluSliceInfoKHR::pStdSliceHeader->slice_type **must** be identical for all elements of pNaluSliceEntries

## Valid Usage (Implicit)

- VUID-VkVideoEncodeH264PictureInfoKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_VIDEO_ENCODE_H264_PICTURE_INFO_KHR
- VUID-VkVideoEncodeH264PictureInfoKHR-pNaluSliceEntries-parameter
  pNaluSliceEntries **must** be a valid pointer to an array of naluSliceEntryCount valid VkVideoEncodeH264NaluSliceInfoKHR structures
- VUID-VkVideoEncodeH264PictureInfoKHR-pStdPictureInfo-parameter

pStdPictureInfo **must** be a valid pointer to a valid StdVideoEncodeH264PictureInfo value

- VUID-VkVideoEncodeH264PictureInfoKHR-naluSliceEntryCount-arraylength
  naluSliceEntryCount **must** be greater than 0

The VkVideoEncodeH264NaluSliceInfoKHR structure is defined as:

```
// Provided by VK_KHR_video_encode_h264
typedef struct VkVideoEncodeH264NaluSliceInfoKHR {
    VkStructureType                         sType;
    const void*                             pNext;
    int32_t                                 constantQp;
    const StdVideoEncodeH264SliceHeader*    pStdSliceHeader;
} VkVideoEncodeH264NaluSliceInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- constantQp is the QP to use for the slice if the current rate control mode configured for the video session is VK_VIDEO_ENCODE_RATE_CONTROL_MODE_DISABLED_BIT_KHR.
- pStdSliceHeader is a pointer to a StdVideoEncodeH264SliceHeader structure specifying H.264 slice header parameters for the slice.

**Std Slice Header Parameters**

The members of the StdVideoEncodeH264SliceHeader structure pointed to by pStdSliceHeader are interpreted as follows:

- flags.reserved and reserved1 are used only for padding purposes and are otherwise ignored;
- if pWeightTable is not NULL, then it is a pointer to a StdVideoEncodeH264WeightTable that is interpreted as follows:
  - flags.reserved is used only for padding purposes and is otherwise ignored;
  - all other members of StdVideoEncodeH264WeightTable are interpreted as defined in section 7.4.3.2 of the ITU-T H.264 Specification;
- all other members are interpreted as defined in section 7.4.3 of the ITU-T H.264 Specification.

### Valid Usage (Implicit)

- VUID-VkVideoEncodeH264NaluSliceInfoKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_VIDEO_ENCODE_H264_NALU_SLICE_INFO_KHR
- VUID-VkVideoEncodeH264NaluSliceInfoKHR-pNext-pNext
  pNext **must** be NULL
- VUID-VkVideoEncodeH264NaluSliceInfoKHR-pStdSliceHeader-parameter
  pStdSliceHeader **must** be a valid pointer to a valid StdVideoEncodeH264SliceHeader value

The VkVideoEncodeH264DpbSlotInfoKHR structure is defined as:

```
// Provided by VK_KHR_video_encode_h264
typedef struct VkVideoEncodeH264DpbSlotInfoKHR {
    VkStructureType                             sType;
    const void*                                 pNext;
    const StdVideoEncodeH264ReferenceInfo*      pStdReferenceInfo;
} VkVideoEncodeH264DpbSlotInfoKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `pStdReferenceInfo` is a pointer to a `StdVideoEncodeH264ReferenceInfo` structure specifying H.264 reference information.

This structure is specified in the `pNext` chain of VkVideoEncodeInfoKHR::`pSetupReferenceSlot`, if not `NULL`, and the `pNext` chain of the elements of VkVideoEncodeInfoKHR::`pReferenceSlots` to specify the codec-specific reference picture information for an H.264 encode operation.

**Active Reference Picture Information**

When this structure is specified in the `pNext` chain of the elements of VkVideoEncodeInfoKHR::`pReferenceSlots`, one element is added to the list of active reference pictures used by the video encode operation for each element of VkVideoEncodeInfoKHR::`pReferenceSlots` as follows:

- The image subregion used is determined according to the H.264 Encode Picture Data Access section.
- The reference picture is associated with the DPB slot index specified in the `slotIndex` member of the corresponding element of VkVideoEncodeInfoKHR::`pReferenceSlots`.
- The reference picture is associated with the H.264 reference information provided in `pStdReferenceInfo`.

**Reconstructed Picture Information**

When this structure is specified in the `pNext` chain of VkVideoEncodeInfoKHR::`pSetupReferenceSlot`, the information related to the reconstructed picture is defined as follows:

- The image subregion used is determined according to the H.264 Encode Picture Data Access section.
- If reference picture setup is requested, then the reconstructed picture is used to activate the DPB slot with the index specified in VkVideoEncodeInfoKHR::`pSetupReferenceSlot->slotIndex`.
- The reconstructed picture is associated with the H.264 reference information provided in `pStdReferenceInfo`.

**Std Reference Information**

The members of the `StdVideoEncodeH264ReferenceInfo` structure pointed to by `pStdReferenceInfo` are interpreted as follows:

- `flags.reserved` is used only for padding purposes and is otherwise ignored;
- `flags.used_for_long_term_reference` is used to indicate whether the picture is marked as "used for long-term reference" as defined in section 8.2.5.1 of the ITU-T H.264 Specification;
- `primary_pic_type` as defined in section 7.4.2 of the ITU-T H.264 Specification;
- `long_term_pic_num` and `long_term_frame_idx` as defined in section 7.4.3 of the ITU-T H.264 Specification;
- `temporal_id` as defined in section G.7.4.1.1 of the ITU-T H.264 Specification;
- all other members are interpreted as defined in section 8.2 of the ITU-T H.264 Specification.

**Valid Usage (Implicit)**

- VUID-VkVideoEncodeH264DpbSlotInfoKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_VIDEO_ENCODE_H264_DPB_SLOT_INFO_KHR`
- VUID-VkVideoEncodeH264DpbSlotInfoKHR-pStdReferenceInfo-parameter
  `pStdReferenceInfo` **must** be a valid pointer to a valid `StdVideoEncodeH264ReferenceInfo` value

## 42.17.12. H.264 Encode Rate Control

**Group of Pictures**

In case of H.264 encoding it is common practice to follow a regular pattern of different picture types in display order when encoding subsequent frames. This pattern is referred to as the *group of pictures* (GOP).

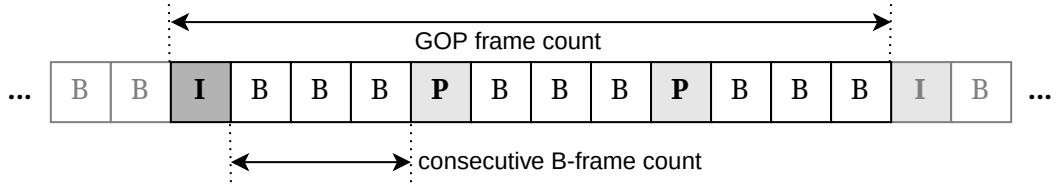
A regular GOP is defined by the following parameters:

- The number of frames in the GOP;
- The number of consecutive B frames between I and/or P frames in display order.

GOPs are further classified as *open* and *closed* GOPs.

Frame types in an open GOP follow each other in display order according to the following algorithm:

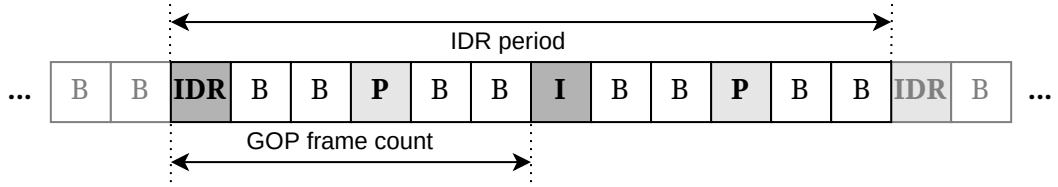
1. The first frame is always an I frame.
2. This is followed by a number of consecutive B frames, as defined above.
3. If the number of frames in the GOP is not reached yet, then the next frame is a P frame and the algorithm continues from step 2.




Text is not SVG - cannot display

*Figure 31. H.264 open GOP*

In case of a closed GOP, an IDR frame is used at a certain period.



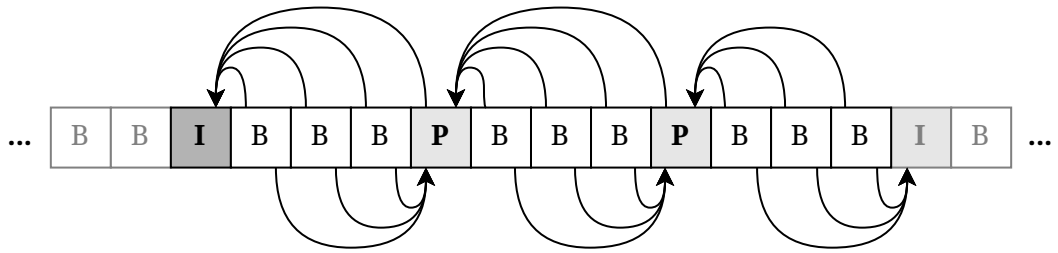

Text is not SVG - cannot display

*Figure 32. H.264 closed GOP*

It is also typical for H.264 encoding to use specific reference picture usage patterns across the frames of the GOP. The two most common reference patterns used are as follows:

**Flat Reference Pattern**

- Each P frame uses the last non-B frame, in display order, as reference.
- Each B frame uses the last non-B frame, in display order, as its forward reference, and uses the next non-B frame, in display order, as its backward reference.



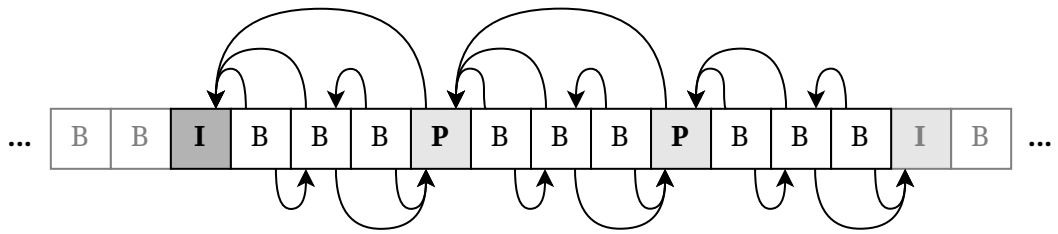

Text is not SVG - cannot display

*Figure 33. H.264 flat reference pattern*

**Dyadic Reference Pattern**

- Each P frame uses the last non-B frame, in display order, as reference.

- The following algorithm is applied to the sequence of consecutive B frames between I and/or P frames in display order:
  1. The B frame in the middle of this sequence uses the frame preceding the sequence as its forward reference, and uses the frame following the sequence as its backward reference.
  2. The algorithm is executed recursively for the following frame sequences:
     - The B frames of the original sequence preceding the frame in the middle, if any.
     - The B frames of the original sequence following the frame in the middle, if any.




Text is not SVG - cannot display

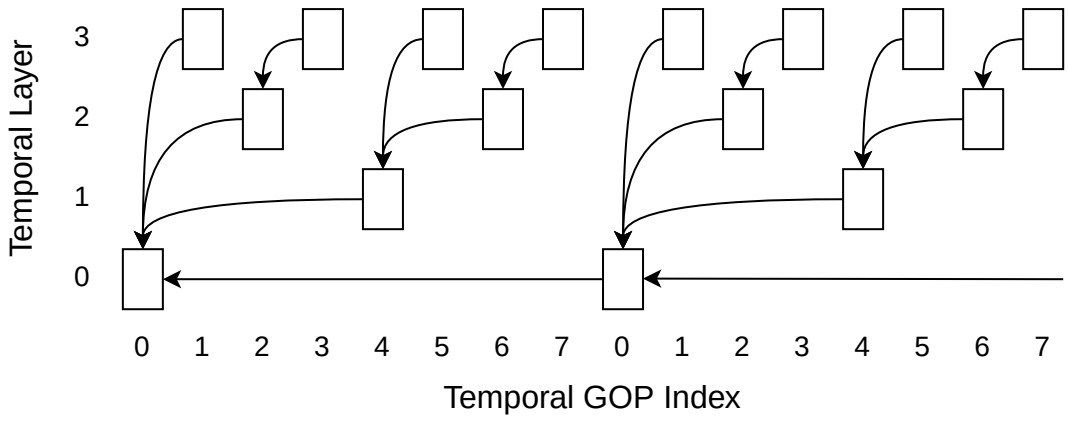
*Figure 34. H.264 dyadic reference pattern*

The application **can** provide guidance to the implementation's rate control algorithm about the structure of the GOP used by the application. Any such guidance about the GOP and its structure does not mandate that specific GOP structure to be used by the application, as the picture type of individual encoded pictures is still application-controlled, however, any deviation from the provided guidance **may** result in undesired rate control behavior including, but not limited, to the implementation not being able to conform to the expected average or target bitrates, or other rate control parameters specified by the application.

When an H.264 encode session is used to encode multiple temporal layers, it is also common practice to follow a regular pattern for the H.264 temporal ID for the encoded pictures in display order when encoding subsequent frames. This pattern is referred to as the *temporal GOP*. The most common temporal layer pattern used is as follows:

**Dyadic Temporal Layer Pattern**

- The number of frames in the temporal GOP is $2^{n-1}$, where n is the number of temporal layers.
- The $i^{th}$ frame in the temporal GOP uses temporal ID t, if and only if the index of the least significant bit set in i equals n-t-1, except for the first frame, which is the only frame in the temporal GOP using temporal ID zero.
- The $i^{th}$ frame in the temporal GOP uses the $r^{th}$ frame as reference, where r is calculated from i by clearing the least significant bit set in it, except for the first frame in the temporal GOP, which uses the first frame of the previous temporal GOP, if any, as reference.




Text is not SVG - cannot display

*Figure 35. H.264 dyadic temporal layer pattern*

**NOTE** Multi-layer rate control and multi-layer coding are typically used for streaming cases where low latency is expected, hence B pictures with backward prediction are usually not used.

The `VkVideoEncodeH264RateControlInfoKHR` structure is defined as:

```
// Provided by VK_KHR_video_encode_h264
typedef struct VkVideoEncodeH264RateControlInfoKHR {
    VkStructureType                         sType;
    const void*                             pNext;
    VkVideoEncodeH264RateControlFlagsKHR    flags;
    uint32_t                                gopFrameCount;
    uint32_t                                idrPeriod;
    uint32_t                                consecutiveBFrameCount;
    uint32_t                                temporalLayerCount;
} VkVideoEncodeH264RateControlInfoKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is a bitmask of VkVideoEncodeH264RateControlFlagBitsKHR specifying H.264 rate control flags.
- `gopFrameCount` is the number of frames within a group of pictures (GOP) intended to be used by the application. If it is set to 0, the rate control algorithm **may** assume an implementation-dependent GOP length. If it is set to `UINT32_MAX`, the GOP length is treated as infinite.
- `idrPeriod` is the interval, in terms of number of frames, between two IDR frames (see IDR period). If it is set to 0, the rate control algorithm **may** assume an implementation-dependent IDR period. If it is set to `UINT32_MAX`, the IDR period is treated as infinite.
- `consecutiveBFrameCount` is the number of consecutive B frames between I and/or P frames within

the GOP.

- `temporalLayerCount` specifies the number of H.264 temporal layers that the application intends to use.

When an instance of this structure is included in the `pNext` chain of the VkVideoCodingControlInfoKHR structure passed to the vkCmdControlVideoCodingKHR command, and VkVideoCodingControlInfoKHR::`flags` includes `VK_VIDEO_CODING_CONTROL_ENCODE_RATE_CONTROL_BIT_KHR`, the parameters in this structure are used as guidance for the implementation's rate control algorithm (see Video Coding Control).

If `flags` includes `VK_VIDEO_ENCODE_H264_RATE_CONTROL_ATTEMPT_HRD_COMPLIANCE_BIT_KHR`, then the rate control state is reset to an initial state to meet HRD compliance requirements. Otherwise the new rate control state **may** be applied without a reset depending on the implementation and the specified rate control parameters.

**NOTE** It would be possible to infer the picture type to be used when encoding a frame, on the basis of the values provided for `consecutiveBFrameCount`, `idrPeriod`, and `gopFrameCount`, but this inferred picture type will not be used by implementations to override the picture type provided to the video encode operation.

## Valid Usage

- VUID-VkVideoEncodeH264RateControlInfoKHR-flags-08280
  If VkVideoEncodeH264CapabilitiesKHR::`flags`, as returned by vkGetPhysicalDeviceVideoCapabilitiesKHR for the used video profile, does not include `VK_VIDEO_ENCODE_H264_CAPABILITY_HRD_COMPLIANCE_BIT_KHR`, then `flags` **must** not contain `VK_VIDEO_ENCODE_H264_RATE_CONTROL_ATTEMPT_HRD_COMPLIANCE_BIT_KHR`
- VUID-VkVideoEncodeH264RateControlInfoKHR-flags-08281
  If `flags` contains `VK_VIDEO_ENCODE_H264_RATE_CONTROL_REFERENCE_PATTERN_FLAT_BIT_KHR` or `VK_VIDEO_ENCODE_H264_RATE_CONTROL_REFERENCE_PATTERN_DYADIC_BIT_KHR`, then it **must** also contain `VK_VIDEO_ENCODE_H264_RATE_CONTROL_REGULAR_GOP_BIT_KHR`
- VUID-VkVideoEncodeH264RateControlInfoKHR-flags-08282
  If `flags` contains `VK_VIDEO_ENCODE_H264_RATE_CONTROL_REFERENCE_PATTERN_FLAT_BIT_KHR`, then it **must** not also contain `VK_VIDEO_ENCODE_H264_RATE_CONTROL_REFERENCE_PATTERN_DYADIC_BIT_KHR`
- VUID-VkVideoEncodeH264RateControlInfoKHR-flags-08283
  If `flags` contains `VK_VIDEO_ENCODE_H264_RATE_CONTROL_REGULAR_GOP_BIT_KHR`, then `gopFrameCount` **must** be greater than `0`
- VUID-VkVideoEncodeH264RateControlInfoKHR-idrPeriod-08284
  If `idrPeriod` is not `0`, then it **must** be greater than or equal to `gopFrameCount`
- VUID-VkVideoEncodeH264RateControlInfoKHR-consecutiveBFrameCount-08285
  If `consecutiveBFrameCount` is not `0`, then it **must** be less than `gopFrameCount`

### Valid Usage (Implicit)

- VUID-VkVideoEncodeH264RateControlInfoKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_VIDEO_ENCODE_H264_RATE_CONTROL_INFO_KHR`
- VUID-VkVideoEncodeH264RateControlInfoKHR-flags-parameter
  `flags` **must** be a valid combination of VkVideoEncodeH264RateControlFlagBitsKHR values

Bits which **can** be set in VkVideoEncodeH264RateControlInfoKHR::`flags`, specifying H.264 rate control flags, are:

```
// Provided by VK_KHR_video_encode_h264
typedef enum VkVideoEncodeH264RateControlFlagBitsKHR {
    VK_VIDEO_ENCODE_H264_RATE_CONTROL_ATTEMPT_HRD_COMPLIANCE_BIT_KHR = 0x00000001,
    VK_VIDEO_ENCODE_H264_RATE_CONTROL_REGULAR_GOP_BIT_KHR = 0x00000002,
    VK_VIDEO_ENCODE_H264_RATE_CONTROL_REFERENCE_PATTERN_FLAT_BIT_KHR = 0x00000004,
    VK_VIDEO_ENCODE_H264_RATE_CONTROL_REFERENCE_PATTERN_DYADIC_BIT_KHR = 0x00000008,
    VK_VIDEO_ENCODE_H264_RATE_CONTROL_TEMPORAL_LAYER_PATTERN_DYADIC_BIT_KHR =
0x00000010,
} VkVideoEncodeH264RateControlFlagBitsKHR;
```

- `VK_VIDEO_ENCODE_H264_RATE_CONTROL_ATTEMPT_HRD_COMPLIANCE_BIT_KHR` specifies that rate control **should** attempt to produce an HRD compliant bitstream, as defined in annex C of the ITU-T H.264 Specification.
- `VK_VIDEO_ENCODE_H264_RATE_CONTROL_REGULAR_GOP_BIT_KHR` specifies that the application intends to use a regular GOP structure according to the parameters specified in the `gopFrameCount`, `idrPeriod`, and `consecutiveBFrameCount` members of the VkVideoEncodeH264RateControlInfoKHR structure.
- `VK_VIDEO_ENCODE_H264_RATE_CONTROL_REFERENCE_PATTERN_FLAT_BIT_KHR` specifies that the application intends to follow a flat reference pattern in the GOP.
- `VK_VIDEO_ENCODE_H264_RATE_CONTROL_REFERENCE_PATTERN_DYADIC_BIT_KHR` specifies that the application intends to follow a dyadic reference pattern in the GOP.
- `VK_VIDEO_ENCODE_H264_RATE_CONTROL_TEMPORAL_LAYER_PATTERN_DYADIC_BIT_KHR` specifies that the application intends to follow a dyadic temporal layer pattern.

```
// Provided by VK_KHR_video_encode_h264
typedef VkFlags VkVideoEncodeH264RateControlFlagsKHR;
```

`VkVideoEncodeH264RateControlFlagsKHR` is a bitmask type for setting a mask of zero or more VkVideoEncodeH264RateControlFlagBitsKHR.

#### Rate Control Layers

The `VkVideoEncodeH264RateControlLayerInfoKHR` structure is defined as:

```
// Provided by VK_KHR_video_encode_h264
typedef struct VkVideoEncodeH264RateControlLayerInfoKHR {
    VkStructureType                  sType;
    const void*                      pNext;
    VkBool32                         useMinQp;
    VkVideoEncodeH264QpKHR           minQp;
    VkBool32                         useMaxQp;
    VkVideoEncodeH264QpKHR           maxQp;
    VkBool32                         useMaxFrameSize;
    VkVideoEncodeH264FrameSizeKHR    maxFrameSize;
} VkVideoEncodeH264RateControlLayerInfoKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `useMinQp` indicates whether the QP values determined by rate control will be clamped to the lower bounds on the QP values specified in `minQp`.
- `minQp` specifies the lower bounds on the QP values, for each picture type, that the implementation's rate control algorithm will use when `useMinQp` is set to `VK_TRUE`.
- `useMaxQp` indicates whether the QP values determined by rate control will be clamped to the upper bounds on the QP values specified in `maxQp`.
- `maxQp` specifies the upper bounds on the QP values, for each picture type, that the implementation's rate control algorithm will use when `useMaxQp` is set to `VK_TRUE`.
- `useMaxFrameSize` indicates whether the implementation's rate control algorithm **should** use the values specified in `maxFrameSize` as the upper bounds on the encoded frame size for each picture type.
- `maxFrameSize` specifies the upper bounds on the encoded frame size, for each picture type, when `useMaxFrameSize` is set to `VK_TRUE`.

When used, the values in `minQp` and `maxQp` guarantee that the effective QP values used by the implementation will respect those lower and upper bounds, respectively. However, limiting the range of QP values that the implementation is able to use will also limit the capabilities of the implementation's rate control algorithm to comply to other constraints. In particular, the implementation **may** not be able to comply to the following:

- The average and/or peak bitrate values to be used for the encoded bitstream specified in the `averageBitrate` and `maxBitrate` members of the VkVideoEncodeRateControlLayerInfoKHR structure.
- The upper bounds on the encoded frame size, for each picture type, specified in the `maxFrameSize` member of `VkVideoEncodeH264RateControlLayerInfoKHR`.

**NOTE** In general, applications need to configure rate control parameters appropriately in order to be able to get the desired rate control behavior, as described in the Video Encode Rate Control section.

When an instance of this structure is included in the `pNext` chain of a

VkVideoEncodeRateControlLayerInfoKHR structure specified in one of the elements of the pLayers array member of the VkVideoEncodeRateControlInfoKHR structure passed to the vkCmdControlVideoCodingKHR command, VkVideoCodingControlInfoKHR::flags includes VK_VIDEO_CODING_CONTROL_ENCODE_RATE_CONTROL_BIT_KHR, and the bound video session was created with the video codec operation VK_VIDEO_CODEC_OPERATION_ENCODE_H264_BIT_KHR, it specifies the H.264-specific rate control parameters of the rate control layer corresponding to that element of pLayers.

## Valid Usage

- VUID-VkVideoEncodeH264RateControlLayerInfoKHR-useMinQp-08286
  If useMinQp is VK_TRUE, then the qpI, qpP, and qpB members of minQp **must** all be between VkVideoEncodeH264CapabilitiesKHR::minQp and VkVideoEncodeH264CapabilitiesKHR::maxQp, as returned by vkGetPhysicalDeviceVideoCapabilitiesKHR for the used video profile
- VUID-VkVideoEncodeH264RateControlLayerInfoKHR-useMaxQp-08287
  If useMaxQp is VK_TRUE, then the qpI, qpP, and qpB members of maxQp **must** all be between VkVideoEncodeH264CapabilitiesKHR::minQp and VkVideoEncodeH264CapabilitiesKHR::maxQp, as returned by vkGetPhysicalDeviceVideoCapabilitiesKHR for the used video profile
- VUID-VkVideoEncodeH264RateControlLayerInfoKHR-useMinQp-08288
  If useMinQp is VK_TRUE and VkVideoEncodeH264CapabilitiesKHR::flags, as returned by vkGetPhysicalDeviceVideoCapabilitiesKHR for the used video profile, does not include VK_VIDEO_ENCODE_H264_CAPABILITY_PER_PICTURE_TYPE_MIN_MAX_QP_BIT_KHR, then the qpI, qpP, and qpB members of minQp **must** all specify the same value
- VUID-VkVideoEncodeH264RateControlLayerInfoKHR-useMaxQp-08289
  If useMaxQp is VK_TRUE and VkVideoEncodeH264CapabilitiesKHR::flags, as returned by vkGetPhysicalDeviceVideoCapabilitiesKHR for the used video profile, does not include VK_VIDEO_ENCODE_H264_CAPABILITY_PER_PICTURE_TYPE_MIN_MAX_QP_BIT_KHR, then the qpI, qpP, and qpB members of maxQp **must** all specify the same value
- VUID-VkVideoEncodeH264RateControlLayerInfoKHR-useMinQp-08374
  If useMinQp and useMaxQp are both VK_TRUE, then the qpI, qpP, and qpB members of minQp **must** all be less than or equal to the respective members of maxQp

## Valid Usage (Implicit)

- VUID-VkVideoEncodeH264RateControlLayerInfoKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_VIDEO_ENCODE_H264_RATE_CONTROL_LAYER_INFO_KHR
- VUID-VkVideoEncodeH264RateControlLayerInfoKHR-minQp-parameter
  minQp **must** be a valid VkVideoEncodeH264QpKHR structure
- VUID-VkVideoEncodeH264RateControlLayerInfoKHR-maxQp-parameter
  maxQp **must** be a valid VkVideoEncodeH264QpKHR structure
- VUID-VkVideoEncodeH264RateControlLayerInfoKHR-maxFrameSize-parameter

maxFrameSize **must** be a valid VkVideoEncodeH264FrameSizeKHR structure

The VkVideoEncodeH264QpKHR structure is defined as:

```
// Provided by VK_KHR_video_encode_h264
typedef struct VkVideoEncodeH264QpKHR {
    int32_t    qpI;
    int32_t    qpP;
    int32_t    qpB;
} VkVideoEncodeH264QpKHR;
```

- qpI is the QP to be used for I pictures.
- qpP is the QP to be used for P pictures.
- qpB is the QP to be used for B pictures.

The VkVideoEncodeH264FrameSizeKHR structure is defined as:

```
// Provided by VK_KHR_video_encode_h264
typedef struct VkVideoEncodeH264FrameSizeKHR {
    uint32_t    frameISize;
    uint32_t    framePSize;
    uint32_t    frameBSize;
} VkVideoEncodeH264FrameSizeKHR;
```

- frameISize is the size in bytes to be used for I pictures.
- framePSize is the size in bytes to be used for P pictures.
- frameBSize is the size in bytes to be used for B pictures.

#### GOP Remaining Frames

Besides session level rate control configuration, the application **can** specify the number of frames per frame type remaining in the group of pictures (GOP).

The VkVideoEncodeH264GopRemainingFrameInfoKHR structure is defined as:

```
// Provided by VK_KHR_video_encode_h264
typedef struct VkVideoEncodeH264GopRemainingFrameInfoKHR {
    VkStructureType    sType;
    const void*        pNext;
    VkBool32           useGopRemainingFrames;
    uint32_t           gopRemainingI;
    uint32_t           gopRemainingP;
    uint32_t           gopRemainingB;
} VkVideoEncodeH264GopRemainingFrameInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- useGopRemainingFrames indicates whether the implementation's rate control algorithm **should** use the values specified in gopRemainingI, gopRemainingP, and gopRemainingB. If useGopRemainingFrames is VK_FALSE, then the values of gopRemainingI, gopRemainingP, and gopRemainingB are ignored.
- gopRemainingI specifies the number of I frames the implementation's rate control algorithm **should** assume to be remaining in the GOP prior to executing the video encode operation.
- gopRemainingP specifies the number of P frames the implementation's rate control algorithm **should** assume to be remaining in the GOP prior to executing the video encode operation.
- gopRemainingB specifies the number of B frames the implementation's rate control algorithm **should** assume to be remaining in the GOP prior to executing the video encode operation.

Setting useGopRemainingFrames to VK_TRUE and including this structure in the pNext chain of VkVideoBeginCodingInfoKHR is only mandatory if the VkVideoEncodeH264CapabilitiesKHR::requiresGopRemainingFrames reported for the used video profile is VK_TRUE. However, implementations **may** use these remaining frame counts, when specified, even when it is not required. In particular, when the application does not use a regular GOP structure, these values **may** provide additional guidance for the implementation's rate control algorithm.

The VkVideoEncodeH264CapabilitiesKHR::prefersGopRemainingFrames capability is also used to indicate that the implementation's rate control algorithm **may** operate more accurately if the application specifies the remaining frame counts using this structure.

As with other rate control guidance values, if the effective order and number of frames encoded by the application are not in line with the remaining frame counts specified in this structure at any given point, then the behavior of the implementation's rate control algorithm **may** deviate from the one expected by the application.

### Valid Usage (Implicit)

- VUID-VkVideoEncodeH264GopRemainingFrameInfoKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_VIDEO_ENCODE_H264_GOP_REMAINING_FRAME_INFO_KHR

## 42.17.13. H.264 Encode Requirements

This section described the **required** H.264 encoding capabilities for physical devices that have at least one queue family that supports the video codec operation VK_VIDEO_CODEC_OPERATION_ENCODE_H264_BIT_KHR, as returned by vkGetPhysicalDeviceQueueFamilyProperties2 in VkQueueFamilyVideoPropertiesKHR::videoCodecOperations.

*Table 57. Required Video Std Header Versions*

| Video Std Header Name | Version |
|---|---|
| vulkan_video_codec_h264std_encode | 1.0.0 |

*Table 58. Required Video Capabilities*

| Video Capability | Requirement | Requirement Type[1] |
|---|---|---|
| **VkVideoCapabilitiesKHR** | | |
| flags | - | min |
| minBitstreamBufferOffsetAlignment | 4096 | max |
| minBitstreamBufferSizeAlignment | 4096 | max |
| pictureAccessGranularity | (64,64) | max |
| minCodedExtent | - | max |
| maxCodedExtent | - | min |
| maxDpbSlots | 0 | min |
| maxActiveReferencePictures | 0 | min |
| **VkVideoEncodeCapabilitiesKHR** | | |
| flags | - | min |
| rateControlModes | - | min |
| maxBitrate | 64000 | min |
| maxQualityLevels | 1 | min |
| encodeInputPictureGranularity | (64,64) | max |
| supportedEncodeFeedbackFlags | VK_VIDEO_ENCODE_FEEDBACK_BITSTREAM_BUFFER_OFFSET_BIT_KHR<br>VK_VIDEO_ENCODE_FEEDBACK_BITSTREAM_BYTES_WRITTEN_BIT_KHR | min |
| **VkVideoEncodeH264CapabilitiesKHR** | | |
| flags | - | min |
| maxLevelIdc | STD_VIDEO_H264_LEVEL_IDC_1_0 | min |
| maxSliceCount | 1 | min |
| maxPPictureL0ReferenceCount | 0 | min |
| maxBPictureL0ReferenceCount | 0 | min |
| maxL1ReferenceCount | 0 | min |
| maxTemporalLayerCount | 1 | min |
| expectDyadicTemporalLayerPattern | - | implementation-dependent |
| minQp | - | max |
| maxQp | - | min |

| Video Capability | Requirement | Requirement Type[1] |
|---|---|---|
| `prefersGopRemainingFrames` | - | implementation-dependent |
| `requiresGopRemainingFrames` | - | implementation-dependent |
| `stdSyntaxFlags` | - | min |

1

The **Requirement Type** column specifies the requirement is either the minimum value all implementations **must** support, the maximum value all implementations **must** support, or the exact value all implementations **must** support. For bitmasks a minimum value is the least bits all implementations **must** set, but they **may** have additional bits set beyond this minimum.

# 42.18. H.265 Encode Operations

Video encode operations using an H.265 encode profile **can** be used to encode elementary video stream sequences compliant to the ITU-T H.265 Specification.

| | |
|---|---|
| **NOTE** | Refer to the Preamble for information on how the Khronos Intellectual Property Rights Policy relates to normative references to external materials not created by Khronos. |

This process is performed according to the video encode operation steps with the codec-specific semantics defined in section 8 of the ITU-T H.265 Specification as follows:

- Syntax elements, derived values, and other parameters are applied from the following structures:
  - The `StdVideoH265VideoParameterSet` structure corresponding to the active VPS specifying the H.265 video parameter set.
  - The `StdVideoH265SequenceParameterSet` structure corresponding to the active SPS specifying the H.265 sequence parameter set.
  - The `StdVideoH265PictureParameterSet` structure corresponding to the active PPS specifying the H.265 picture parameter set.
  - The `StdVideoEncodeH265PictureInfo` structure specifying the H.265 picture information.
  - The `StdVideoEncodeH265SliceSegmentHeader` structures specifying the H.265 slice segment header parameters for each encoded H.265 slice segment.
  - The `StdVideoEncodeH265ReferenceInfo` structures specifying the H.265 reference information corresponding to the optional reconstructed picture and any active reference pictures.
- The encoded bitstream data is written to the destination video bitstream buffer range as defined in the H.265 Encode Bitstream Data Access section.
- Picture data in the video picture resources corresponding to the used encode input picture, active reference pictures, and optional reconstructed picture is accessed as defined in the H.265

Encode Picture Data Access section.

- The decision on reference picture setup is made according to the parameters specified in the H.265 picture information.

If the parameters adhere to the syntactic and semantic requirements defined in the corresponding sections of the ITU-T H.265 Specification, as described above, and the DPB slots associated with the active reference pictures all refer to valid picture references, then the video encode operation will complete successfully. Otherwise, the video encode operation **may** complete unsuccessfully.

### 42.18.1. H.265 Encode Parameter Overrides

Implementations **may** override, unless otherwise specified, any of the H.265 encode parameters specified in the following Video Std structures:

- `StdVideoH265VideoParameterSet`
- `StdVideoH265SequenceParameterSet`
- `StdVideoH265PictureParameterSet`
- `StdVideoEncodeH265PictureInfo`
- `StdVideoEncodeH265SliceSegmentHeader`
- `StdVideoEncodeH265ReferenceInfo`

All such H.265 encode parameter overrides **must** fulfill the conditions defined in the Video Encode Parameter Overrides section.

In addition, implementations **must** not override any of the following H.265 encode parameters:

- `StdVideoEncodeH265PictureInfo`::`pic_type`
- `StdVideoEncodeH265SliceSegmentHeader`::`slice_type`

In case of H.265 encode parameters stored in video session parameters objects, applications need to use the vkGetEncodedVideoSessionParametersKHR command to determine whether any implementation overrides happened. If the query indicates that implementation overrides were applied, then the application needs to retrieve and use the encoded H.265 parameter sets in the bitstream in order to be able to produce a compliant H.265 video bitstream using the H.265 encode parameters stored in the video session parameters object.

In case of any H.265 encode parameters stored in the encoded bitstream produced by video encode operations, if the implementation supports the `VK_VIDEO_ENCODE_FEEDBACK_BITSTREAM_HAS_OVERRIDES_BIT_KHR` video encode feedback query flag, the application **can** use such queries to retrieve feedback about whether any implementation overrides have been applied to those H.265 encode parameters.

### 42.18.2. H.265 Encode Bitstream Data Access

Each video encode operation writes one or more VCL NAL units comprising of slice segment headers and data of the encoded picture, in the format defined in sections 7.3.6 and 7.3.8, according to the semantics defined in sections 7.4.7 and 7.4.9 of the ITU-T H.265 Specification, respectively.

The number of VCL NAL units written is specified by VkVideoEncodeH265PictureInfoKHR::naluSliceSegmentEntryCount.

### 42.18.3. H.265 Encode Picture Data Access

Accesses to image data within a video picture resource happen at the granularity indicated by VkVideoCapabilitiesKHR::pictureAccessGranularity, as returned by vkGetPhysicalDeviceVideoCapabilitiesKHR for the used video profile. Accordingly, the complete image subregion of a encode input picture, reference picture, or reconstructed picture accessed by video coding operations using an H.265 encode profile is defined as the set of texels within the coordinate range:

([0,endX), [0,endY))

Where:

- endX equals codedExtent.width rounded up to the nearest integer multiple of pictureAccessGranularity.width and clamped to the width of the image subresource referred to by the corresponding VkVideoPictureResourceInfoKHR structure;
- endY equals codedExtent.height rounded up to the nearest integer multiple of pictureAccessGranularity.height and clamped to the height of the image subresource referred to by the corresponding VkVideoPictureResourceInfoKHR structure;

Where codedExtent is the member of the VkVideoPictureResourceInfoKHR structure corresponding to the picture.

In case of video encode operations using an H.265 encode profile, any access to a picture at the coordinates (x,y), as defined by the ITU-T H.265 Specification, is an access to the image subresource referred to by the corresponding VkVideoPictureResourceInfoKHR structure at the texel coordinates (x,y).

Implementations **may** choose not to access some or all texels within particular reference pictures available to a video encode operation (e.g. due to video encode parameter overrides restricting the effective set of used reference pictures, or if the encoding algorithm chooses not to use certain subregions of the reference picture data for sample prediction).

### 42.18.4. H.265 Frame, Picture, Slice Segments, and Tiles

H.265 pictures consist of one or more slices, slice segments, and tiles, as defined in section 6.3.1 of the ITU-T H.265 Specification.

For the purposes of this specification, the H.265 slice segments comprising a picture are referred to as the *picture partitions* of the picture.

Video encode operations using an H.265 encode profile **can** encode slice segments of different types, as defined in section 7.4.7.1 of the ITU-T H.265 Specification, by specifying the corresponding enumeration constant value in StdVideoEncodeH265SliceSegmentHeader::slice_type in the H.265 slice segment header parameters from the Video Std enumeration type StdVideoH265SliceType:

- STD_VIDEO_H265_SLICE_TYPE_B indicates that the slice segment is part of a *B slice* as defined in section 3.12 of the ITU-T H.265 Specification.
- STD_VIDEO_H265_SLICE_TYPE_P indicates that the slice segment is part of a *P slice* as defined in section 3.111 of the ITU-T H.265 Specification.
- STD_VIDEO_H265_SLICE_TYPE_I indicates that the slice segment is part of an *I slice* as defined in section 3.74 of the ITU-T H.265 Specification.

Pictures constructed from such slice segments **can** be of different types, as defined in section 7.4.3.5 of the ITU-T H.265 Specification. Video encode operations using an H.265 encode profile **can** encode pictures of a specific type by specifying the corresponding enumeration constant value in StdVideoEncodeH265PictureInfo::pic_type in the H.265 picture information from the Video Std enumeration type StdVideoH265PictureType:

- STD_VIDEO_H265_PICTURE_TYPE_P indicates that the picture is a *P picture*. A frame consisting of a P picture is also referred to as a *P frame*.
- STD_VIDEO_H265_PICTURE_TYPE_B indicates that the picture is a *B picture*. A frame consisting of a B picture is also referred to as a *B frame*.
- STD_VIDEO_H265_PICTURE_TYPE_I indicates that the picture is an *I picture*. A frame consisting of an I picture is also referred to as an *I frame*.
- STD_VIDEO_H265_PICTURE_TYPE_IDR indicates that the picture is a special type of I picture called an *IDR picture* as defined in section 3.67 of the ITU-T H.265 Specification. A frame consisting of an IDR picture is also referred to as an *IDR frame*.

### 42.18.5. H.265 Coding Blocks

H.265 encode supports two types of coding blocks:

- *Coding tree unit*, as defined in section 3.35 of the ITU-T H.265 Specification.
- *Coding unit*, as defined in section 3.36 of the ITU-T H.265 Specification.

### 42.18.6. H.265 Encode Profile

A video profile supporting H.265 video encode operations is specified by setting VkVideoProfileInfoKHR::videoCodecOperation to VK_VIDEO_CODEC_OPERATION_ENCODE_H265_BIT_KHR and adding a VkVideoEncodeH265ProfileInfoKHR structure to the VkVideoProfileInfoKHR::pNext chain.

The VkVideoEncodeH265ProfileInfoKHR structure is defined as:

```
// Provided by VK_KHR_video_encode_h265
typedef struct VkVideoEncodeH265ProfileInfoKHR {
    VkStructureType           sType;
    const void*               pNext;
    StdVideoH265ProfileIdc    stdProfileIdc;
} VkVideoEncodeH265ProfileInfoKHR;
```

- sType is a VkStructureType value identifying this structure.

- pNext is NULL or a pointer to a structure extending this structure.
- stdProfileIdc is a StdVideoH265ProfileIdc value specifying the H.265 codec profile IDC, as defined in section A.3 of the ITU-T H.265 Specification.

**Valid Usage (Implicit)**

- VUID-VkVideoEncodeH265ProfileInfoKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_VIDEO_ENCODE_H265_PROFILE_INFO_KHR

### 42.18.7. H.265 Encode Capabilities

When calling vkGetPhysicalDeviceVideoCapabilitiesKHR to query the capabilities for an H.265 encode profile, the VkVideoCapabilitiesKHR::pNext chain **must** include a VkVideoEncodeH265CapabilitiesKHR structure that will be filled with the profile-specific capabilities.

The VkVideoEncodeH265CapabilitiesKHR structure is defined as:

```
// Provided by VK_KHR_video_encode_h265
typedef struct VkVideoEncodeH265CapabilitiesKHR {
    VkStructureType                                 sType;
    void*                                           pNext;
    VkVideoEncodeH265CapabilityFlagsKHR             flags;
    StdVideoH265LevelIdc                            maxLevelIdc;
    uint32_t                                        maxSliceSegmentCount;
    VkExtent2D                                      maxTiles;
    VkVideoEncodeH265CtbSizeFlagsKHR                ctbSizes;
    VkVideoEncodeH265TransformBlockSizeFlagsKHR     transformBlockSizes;
    uint32_t                                        maxPPictureL0ReferenceCount;
    uint32_t                                        maxBPictureL0ReferenceCount;
    uint32_t                                        maxL1ReferenceCount;
    uint32_t                                        maxSubLayerCount;
    VkBool32
expectDyadicTemporalSubLayerPattern;
    int32_t                                         minQp;
    int32_t                                         maxQp;
    VkBool32                                        prefersGopRemainingFrames;
    VkBool32                                        requiresGopRemainingFrames;
    VkVideoEncodeH265StdFlagsKHR                    stdSyntaxFlags;
} VkVideoEncodeH265CapabilitiesKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- flags is a bitmask of VkVideoEncodeH265CapabilityFlagBitsKHR indicating supported H.265 encoding capabilities.
- maxLevelIdc is a StdVideoH265LevelIdc value indicating the maximum H.265 level supported by the profile, where enum constant STD_VIDEO_H265_LEVEL_IDC_<major>_<minor> identifies H.265

level `<major>.<minor>` as defined in section A.4 of the ITU-T H.265 Specification.

- `maxSliceSegmentCount` indicates the maximum number of slice segments that **can** be encoded for a single picture. Further restrictions **may** apply to the number of slice segments that **can** be encoded for a single picture depending on other capabilities and codec-specific rules.
- `maxTiles` indicates the maximum number of H.265 tile columns and rows, as defined in sections 3.175 and 3.176 of the ITU-T H.265 Specification that **can** be encoded for a single picture. Further restrictions **may** apply to the number of H.265 tiles that **can** be encoded for a single picture depending on other capabilities and codec-specific rules.
- `ctbSizes` is a bitmask of VkVideoEncodeH265CtbSizeFlagBitsKHR describing the supported CTB sizes.
- `transformBlockSizes` is a bitmask of VkVideoEncodeH265TransformBlockSizeFlagBitsKHR describing the supported transform block sizes.
- `maxPPictureL0ReferenceCount` indicates the maximum number of reference pictures the implementation supports in the reference list L0 for P pictures.

> **NOTE** As implementations **may** override the reference lists, `maxPPictureL0ReferenceCount` does not limit the number of elements that the application **can** specify in the L0 reference list for P pictures. However, if `maxPPictureL0ReferenceCount` is zero, then the use of P pictures is not allowed. In case of H.265 encoding, pictures **can** be encoded using only forward prediction even if P pictures are not supported, as the ITU-T H.265 Specification supports *generalized P & B frames* (also known as low delay B frames) whereas B frames **can** refer to past frames through both the L0 and L1 reference lists.

- `maxBPictureL0ReferenceCount` indicates the maximum number of reference pictures the implementation supports in the reference list L0 for B pictures.
- `maxL1ReferenceCount` indicates the maximum number of reference pictures the implementation supports in the reference list L1 if encoding of B pictures is supported.

> **NOTE** As implementations **may** override the reference lists, `maxBPictureL0ReferenceCount` and `maxL1ReferenceCount` does not limit the number of elements that the application **can** specify in the L0 and L1 reference lists for B pictures. However, if `maxBPictureL0ReferenceCount` and `maxL1ReferenceCount` are both zero, then the use of B pictures is not allowed.

- `maxSubLayerCount` indicates the maximum number of H.265 sub-layers supported by the implementation.
- `expectDyadicTemporalSubLayerPattern` indicates that the implementation's rate control algorithms expect the application to use a dyadic temporal sub-layer pattern when encoding multiple temporal sub-layers.
- `minQp` indicates the minimum QP value supported.
- `maxQp` indicates the maximum QP value supported.
- `prefersGopRemainingFrames` indicates that the implementation's rate control algorithm prefers the application to specify the number of frames of each type remaining in the current group of

pictures when beginning a video coding scope.

- requiresGopRemainingFrames indicates that the implementation's rate control algorithm requires the application to specify the number of frames of each type remaining in the current group of pictures when beginning a video coding scope.
- stdSyntaxFlags is a bitmask of VkVideoEncodeH265StdFlagBitsKHR indicating capabilities related to H.265 syntax elements.

### Valid Usage (Implicit)

- VUID-VkVideoEncodeH265CapabilitiesKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_VIDEO_ENCODE_H265_CAPABILITIES_KHR

Bits which **may** be set in VkVideoEncodeH265CapabilitiesKHR::flags, indicating the H.265 encoding capabilities supported, are:

```
// Provided by VK_KHR_video_encode_h265
typedef enum VkVideoEncodeH265CapabilityFlagBitsKHR {
    VK_VIDEO_ENCODE_H265_CAPABILITY_HRD_COMPLIANCE_BIT_KHR = 0x00000001,
    VK_VIDEO_ENCODE_H265_CAPABILITY_PREDICTION_WEIGHT_TABLE_GENERATED_BIT_KHR =
0x00000002,
    VK_VIDEO_ENCODE_H265_CAPABILITY_ROW_UNALIGNED_SLICE_SEGMENT_BIT_KHR = 0x00000004,
    VK_VIDEO_ENCODE_H265_CAPABILITY_DIFFERENT_SLICE_SEGMENT_TYPE_BIT_KHR = 0x00000008,
    VK_VIDEO_ENCODE_H265_CAPABILITY_B_FRAME_IN_L0_LIST_BIT_KHR = 0x00000010,
    VK_VIDEO_ENCODE_H265_CAPABILITY_B_FRAME_IN_L1_LIST_BIT_KHR = 0x00000020,
    VK_VIDEO_ENCODE_H265_CAPABILITY_PER_PICTURE_TYPE_MIN_MAX_QP_BIT_KHR = 0x00000040,
    VK_VIDEO_ENCODE_H265_CAPABILITY_PER_SLICE_SEGMENT_CONSTANT_QP_BIT_KHR =
0x00000080,
    VK_VIDEO_ENCODE_H265_CAPABILITY_MULTIPLE_TILES_PER_SLICE_SEGMENT_BIT_KHR =
0x00000100,
    VK_VIDEO_ENCODE_H265_CAPABILITY_MULTIPLE_SLICE_SEGMENTS_PER_TILE_BIT_KHR =
0x00000200,
} VkVideoEncodeH265CapabilityFlagBitsKHR;
```

- VK_VIDEO_ENCODE_H265_CAPABILITY_HRD_COMPLIANCE_BIT_KHR indicates if the implementation **may** be able to generate HRD compliant bitstreams if any of the nal_hrd_parameters_present_flag, vcl_hrd_parameters_present_flag, or sub_pic_hrd_params_present_flag members of StdVideoH265HrdFlags are set to 1 in the HRD parameters of the active VPS or active SPS, or if StdVideoH265SpsVuiFlags::vui_hrd_parameters_present_flag is set to 1 in the active SPS.
- VK_VIDEO_ENCODE_H265_CAPABILITY_PREDICTION_WEIGHT_TABLE_GENERATED_BIT_KHR indicates that if the weighted_pred_flag or the weighted_bipred_flag member of StdVideoH265PpsFlags is set to 1 in the active PPS when encoding a P picture or B picture, respectively, then the implementation is able to internally decide syntax for pred_weight_table, as defined in section 7.4.7.3 of the ITU-T H.265 Specification, and the application is not **required** to provide a weight table in the H.265 slice segment header parameters.
- VK_VIDEO_ENCODE_H265_CAPABILITY_ROW_UNALIGNED_SLICE_SEGMENT_BIT_KHR indicates that each slice

segment in a frame with a single or multiple tiles per slice may begin or finish at any offset in a CTB row. If not supported, all slice segments in such a frame **must** begin at the start of a CTB row (and hence each slice segment **must** finish at the end of a CTB row). Also indicates that each slice segment in a frame with multiple slices per tile may begin or finish at any offset within the enclosing tile's CTB row. If not supported, slice segments in such a frame **must** begin at the start of the enclosing tile's CTB row (and hence each slice segment **must** finish at the end of the enclosing tile's CTB row).

- `VK_VIDEO_ENCODE_H265_CAPABILITY_DIFFERENT_SLICE_SEGMENT_TYPE_BIT_KHR` indicates that when a frame is encoded with multiple slice segments, the implementation allows encoding each slice segment with a different `StdVideoEncodeH265SliceSegmentHeader`::`slice_type` specified in the H.265 slice segment header parameters. If not supported, all slice segments of the frame **must** be encoded with the same `slice_type` which corresponds to the picture type of the frame.
- `VK_VIDEO_ENCODE_H265_CAPABILITY_B_FRAME_IN_L0_LIST_BIT_KHR` indicates support for using a B frame as L0 reference, as specified in `StdVideoEncodeH265ReferenceListsInfo`::`RefPicList0` in the H.265 picture information.
- `VK_VIDEO_ENCODE_H265_CAPABILITY_B_FRAME_IN_L1_LIST_BIT_KHR` indicates support for using a B frame as L1 reference, as specified in `StdVideoEncodeH265ReferenceListsInfo`::`RefPicList1` in the H.265 picture information.
- `VK_VIDEO_ENCODE_H265_CAPABILITY_PER_PICTURE_TYPE_MIN_MAX_QP_BIT_KHR` indicates support for specifying different QP values in the members of VkVideoEncodeH265QpKHR.
- `VK_VIDEO_ENCODE_H265_CAPABILITY_PER_SLICE_SEGMENT_CONSTANT_QP_BIT_KHR` indicates support for specifying different constant QP values for each slice segment.
- `VK_VIDEO_ENCODE_H265_CAPABILITY_MULTIPLE_TILES_PER_SLICE_SEGMENT_BIT_KHR` indicates if encoding multiple tiles per slice segment, as defined in section 6.3.1 of the ITU-T H.265 Specification, is supported. If this capability flag is not present, then the implementation is only able to encode a single tile for each slice segment.
- `VK_VIDEO_ENCODE_H265_CAPABILITY_MULTIPLE_SLICE_SEGMENTS_PER_TILE_BIT_KHR` indicates if encoding multiple slice segments per tile, as defined in section 6.3.1 of the ITU-T H.265 Specification, is supported. If this capability flag is not present, then the implementation is only able to encode a single slice segment for each tile.

```
// Provided by VK_KHR_video_encode_h265
typedef VkFlags VkVideoEncodeH265CapabilityFlagsKHR;
```

`VkVideoEncodeH265CapabilityFlagsKHR` is a bitmask type for setting a mask of zero or more VkVideoEncodeH265CapabilityFlagBitsKHR.

Bits which **may** be set in VkVideoEncodeH265CapabilitiesKHR::`stdSyntaxFlags`, indicating the capabilities related to the H.265 syntax elements, are:

```
// Provided by VK_KHR_video_encode_h265
typedef enum VkVideoEncodeH265StdFlagBitsKHR {
    VK_VIDEO_ENCODE_H265_STD_SEPARATE_COLOR_PLANE_FLAG_SET_BIT_KHR = 0x00000001,
    VK_VIDEO_ENCODE_H265_STD_SAMPLE_ADAPTIVE_OFFSET_ENABLED_FLAG_SET_BIT_KHR =
0x00000002,
    VK_VIDEO_ENCODE_H265_STD_SCALING_LIST_DATA_PRESENT_FLAG_SET_BIT_KHR = 0x00000004,
    VK_VIDEO_ENCODE_H265_STD_PCM_ENABLED_FLAG_SET_BIT_KHR = 0x00000008,
    VK_VIDEO_ENCODE_H265_STD_SPS_TEMPORAL_MVP_ENABLED_FLAG_SET_BIT_KHR = 0x00000010,
    VK_VIDEO_ENCODE_H265_STD_INIT_QP_MINUS26_BIT_KHR = 0x00000020,
    VK_VIDEO_ENCODE_H265_STD_WEIGHTED_PRED_FLAG_SET_BIT_KHR = 0x00000040,
    VK_VIDEO_ENCODE_H265_STD_WEIGHTED_BIPRED_FLAG_SET_BIT_KHR = 0x00000080,
    VK_VIDEO_ENCODE_H265_STD_LOG2_PARALLEL_MERGE_LEVEL_MINUS2_BIT_KHR = 0x00000100,
    VK_VIDEO_ENCODE_H265_STD_SIGN_DATA_HIDING_ENABLED_FLAG_SET_BIT_KHR = 0x00000200,
    VK_VIDEO_ENCODE_H265_STD_TRANSFORM_SKIP_ENABLED_FLAG_SET_BIT_KHR = 0x00000400,
    VK_VIDEO_ENCODE_H265_STD_TRANSFORM_SKIP_ENABLED_FLAG_UNSET_BIT_KHR = 0x00000800,
    VK_VIDEO_ENCODE_H265_STD_PPS_SLICE_CHROMA_QP_OFFSETS_PRESENT_FLAG_SET_BIT_KHR =
0x00001000,
    VK_VIDEO_ENCODE_H265_STD_TRANSQUANT_BYPASS_ENABLED_FLAG_SET_BIT_KHR = 0x00002000,
    VK_VIDEO_ENCODE_H265_STD_CONSTRAINED_INTRA_PRED_FLAG_SET_BIT_KHR = 0x00004000,
    VK_VIDEO_ENCODE_H265_STD_ENTROPY_CODING_SYNC_ENABLED_FLAG_SET_BIT_KHR =
0x00008000,
    VK_VIDEO_ENCODE_H265_STD_DEBLOCKING_FILTER_OVERRIDE_ENABLED_FLAG_SET_BIT_KHR =
0x00010000,
    VK_VIDEO_ENCODE_H265_STD_DEPENDENT_SLICE_SEGMENTS_ENABLED_FLAG_SET_BIT_KHR =
0x00020000,
    VK_VIDEO_ENCODE_H265_STD_DEPENDENT_SLICE_SEGMENT_FLAG_SET_BIT_KHR = 0x00040000,
    VK_VIDEO_ENCODE_H265_STD_SLICE_QP_DELTA_BIT_KHR = 0x00080000,
    VK_VIDEO_ENCODE_H265_STD_DIFFERENT_SLICE_QP_DELTA_BIT_KHR = 0x00100000,
} VkVideoEncodeH265StdFlagBitsKHR;
```

- `VK_VIDEO_ENCODE_H265_STD_SEPARATE_COLOR_PLANE_FLAG_SET_BIT_KHR` indicates whether the implementation supports using the application-provided value for `StdVideoH265SpsFlags::separate_colour_plane_flag` in the SPS when that value is `1`.
- `VK_VIDEO_ENCODE_H265_STD_SAMPLE_ADAPTIVE_OFFSET_ENABLED_FLAG_SET_BIT_KHR` indicates whether the implementation supports using the application-provided value for `StdVideoH265SpsFlags::sample_adaptive_offset_enabled_flag` in the SPS when that value is `1`.
- `VK_VIDEO_ENCODE_H265_STD_SCALING_LIST_DATA_PRESENT_FLAG_SET_BIT_KHR` indicates whether the implementation supports using the application-provided value for the `scaling_list_enabled_flag` and `sps_scaling_list_data_present_flag` members of `StdVideoH265SpsFlags` in the SPS, and the application-provided value for `StdVideoH265PpsFlags::pps_scaling_list_data_present_flag` in the PPS when those values are `1`.
- `VK_VIDEO_ENCODE_H265_STD_PCM_ENABLED_FLAG_SET_BIT_KHR` indicates whether the implementation supports using the application-provided value for `StdVideoH265SpsFlags::pcm_enable_flag` in the SPS when that value is `1`.
- `VK_VIDEO_ENCODE_H265_STD_SPS_TEMPORAL_MVP_ENABLED_FLAG_SET_BIT_KHR` indicates whether the implementation supports using the application-provided value for `StdVideoH265SpsFlags::sps_temporal_mvp_enabled_flag` in the SPS when that value is `1`.

- VK_VIDEO_ENCODE_H265_STD_INIT_QP_MINUS26_BIT_KHR indicates whether the implementation supports using the application-provided value for StdVideoH265PictureParameterSet::init_qp_minus26 in the PPS when that value is non-zero.
- VK_VIDEO_ENCODE_H265_STD_WEIGHTED_PRED_FLAG_SET_BIT_KHR indicates whether the implementation supports using the application-provided value for StdVideoH265PpsFlags::weighted_pred_flag in the PPS when that value is 1.
- VK_VIDEO_ENCODE_H265_STD_WEIGHTED_BIPRED_FLAG_SET_BIT_KHR indicates whether the implementation supports using the application-provided value for StdVideoH265PpsFlags::weighted_bipred_flag in the PPS when that value is 1.
- VK_VIDEO_ENCODE_H265_STD_LOG2_PARALLEL_MERGE_LEVEL_MINUS2_BIT_KHR indicates whether the implementation supports using the application-provided value for StdVideoH265PictureParameterSet::log2_parallel_merge_level_minus2 in the PPS when that value is non-zero.
- VK_VIDEO_ENCODE_H265_STD_SIGN_DATA_HIDING_ENABLED_FLAG_SET_BIT_KHR indicates whether the implementation supports using the application-provided value for StdVideoH265PpsFlags::sign_data_hiding_enabled_flag in the PPS when that value is 1.
- VK_VIDEO_ENCODE_H265_STD_TRANSFORM_SKIP_ENABLED_FLAG_SET_BIT_KHR indicates whether the implementation supports using the application-provided value for StdVideoH265PpsFlags::transform_skip_enabled_flag in the PPS when that value is 1.
- VK_VIDEO_ENCODE_H265_STD_TRANSFORM_SKIP_ENABLED_FLAG_UNSET_BIT_KHR indicates whether the implementation supports using the application-provided value for StdVideoH265PpsFlags::transform_skip_enabled_flag in the PPS when that value is 0.
- VK_VIDEO_ENCODE_H265_STD_PPS_SLICE_CHROMA_QP_OFFSETS_PRESENT_FLAG_SET_BIT_KHR indicates whether the implementation supports using the application-provided value for StdVideoH265PpsFlags::pps_slice_chroma_qp_offsets_present_flag in the PPS when that value is 1.
- VK_VIDEO_ENCODE_H265_STD_TRANSQUANT_BYPASS_ENABLED_FLAG_SET_BIT_KHR indicates whether the implementation supports using the application-provided value for StdVideoH265PpsFlags::transquant_bypass_enabled_flag in the PPS when that value is 1.
- VK_VIDEO_ENCODE_H265_STD_CONSTRAINED_INTRA_PRED_FLAG_SET_BIT_KHR indicates whether the implementation supports using the application-provided value for StdVideoH265PpsFlags::constrained_intra_pred_flag in the PPS when that value is 1.
- VK_VIDEO_ENCODE_H265_STD_ENTROPY_CODING_SYNC_ENABLED_FLAG_SET_BIT_KHR indicates whether the implementation supports using the application-provided value for StdVideoH265PpsFlags::entropy_coding_sync_enabled_flag in the PPS when that value is 1.
- VK_VIDEO_ENCODE_H265_STD_DEBLOCKING_FILTER_OVERRIDE_ENABLED_FLAG_SET_BIT_KHR indicates whether the implementation supports using the application-provided value for StdVideoH265PpsFlags::deblocking_filter_override_enabled_flag in the PPS when that value is 1.
- VK_VIDEO_ENCODE_H265_STD_DEPENDENT_SLICE_SEGMENTS_ENABLED_FLAG_SET_BIT_KHR indicates whether the implementation supports using the application-provided value for StdVideoH265PpsFlags::dependent_slice_segments_enabled_flag in the PPS when that value is 1.
- VK_VIDEO_ENCODE_H265_STD_DEPENDENT_SLICE_SEGMENT_FLAG_SET_BIT_KHR indicates whether the implementation supports using the application-provided value for StdVideoEncodeH265SliceSegmentHeader::dependent_slice_segment_flag in the H.265 slice segment

header parameters when that value is 1.

- VK_VIDEO_ENCODE_H265_STD_SLICE_QP_DELTA_BIT_KHR indicates whether the implementation supports using the application-provided value for StdVideoEncodeH265SliceSegmentHeader::slice_qp_delta in the H.265 slice segment header parameters when that value is identical across the slice segments of the encoded frame.
- VK_VIDEO_ENCODE_H265_STD_DIFFERENT_SLICE_QP_DELTA_BIT_KHR indicates whether the implementation supports using the application-provided value for StdVideoEncodeH265SliceSegmentHeader::slice_qp_delta in the H.265 slice segment header parameters when that value is different across the slice segments of the encoded frame.

These capability flags provide information to the application about specific H.265 syntax element values that the implementation supports without having to override them and do not otherwise restrict the values that the application **can** specify for any of the mentioned H.265 syntax elements.

```
// Provided by VK_KHR_video_encode_h265
typedef VkFlags VkVideoEncodeH265StdFlagsKHR;
```

VkVideoEncodeH265StdFlagsKHR is a bitmask type for setting a mask of zero or more VkVideoEncodeH265StdFlagBitsKHR.

Bits which **may** be set in VkVideoEncodeH265CapabilitiesKHR::ctbSizes, indicating the CTB sizes supported by the implementation, are:

```
// Provided by VK_KHR_video_encode_h265
typedef enum VkVideoEncodeH265CtbSizeFlagBitsKHR {
    VK_VIDEO_ENCODE_H265_CTB_SIZE_16_BIT_KHR = 0x00000001,
    VK_VIDEO_ENCODE_H265_CTB_SIZE_32_BIT_KHR = 0x00000002,
    VK_VIDEO_ENCODE_H265_CTB_SIZE_64_BIT_KHR = 0x00000004,
} VkVideoEncodeH265CtbSizeFlagBitsKHR;
```

- VK_VIDEO_ENCODE_H265_CTB_SIZE_16_BIT_KHR specifies that a CTB size of 16x16 is supported.
- VK_VIDEO_ENCODE_H265_CTB_SIZE_32_BIT_KHR specifies that a CTB size of 32x32 is supported.
- VK_VIDEO_ENCODE_H265_CTB_SIZE_64_BIT_KHR specifies that a CTB size of 64x64 is supported.

```
// Provided by VK_KHR_video_encode_h265
typedef VkFlags VkVideoEncodeH265CtbSizeFlagsKHR;
```

VkVideoEncodeH265CtbSizeFlagsKHR is a bitmask type for setting a mask of zero or more VkVideoEncodeH265CtbSizeFlagBitsKHR.

Implementations **must** support at least one of VkVideoEncodeH265CtbSizeFlagBitsKHR.

Bits which **may** be set in VkVideoEncodeH265CapabilitiesKHR::transformBlockSizes, indicating the transform block sizes supported by the implementation, are:

```
// Provided by VK_KHR_video_encode_h265
typedef enum VkVideoEncodeH265TransformBlockSizeFlagBitsKHR {
    VK_VIDEO_ENCODE_H265_TRANSFORM_BLOCK_SIZE_4_BIT_KHR = 0x00000001,
    VK_VIDEO_ENCODE_H265_TRANSFORM_BLOCK_SIZE_8_BIT_KHR = 0x00000002,
    VK_VIDEO_ENCODE_H265_TRANSFORM_BLOCK_SIZE_16_BIT_KHR = 0x00000004,
    VK_VIDEO_ENCODE_H265_TRANSFORM_BLOCK_SIZE_32_BIT_KHR = 0x00000008,
} VkVideoEncodeH265TransformBlockSizeFlagBitsKHR;
```

- `VK_VIDEO_ENCODE_H265_TRANSFORM_BLOCK_SIZE_4_BIT_KHR` specifies that a transform block size of 4x4 is supported.
- `VK_VIDEO_ENCODE_H265_TRANSFORM_BLOCK_SIZE_8_BIT_KHR` specifies that a transform block size of 8x8 is supported.
- `VK_VIDEO_ENCODE_H265_TRANSFORM_BLOCK_SIZE_16_BIT_KHR` specifies that a transform block size of 16x16 is supported.
- `VK_VIDEO_ENCODE_H265_TRANSFORM_BLOCK_SIZE_32_BIT_KHR` specifies that a transform block size of 32x32 is supported.

```
// Provided by VK_KHR_video_encode_h265
typedef VkFlags VkVideoEncodeH265TransformBlockSizeFlagsKHR;
```

`VkVideoEncodeH265TransformBlockSizeFlagsKHR` is a bitmask type for setting a mask of zero or more VkVideoEncodeH265TransformBlockSizeFlagBitsKHR.

Implementations **must** support at least one of `VkVideoEncodeH265TransformBlockSizeFlagBitsKHR`.

### 42.18.8. H.265 Encode Quality Level Properties

When calling vkGetPhysicalDeviceVideoEncodeQualityLevelPropertiesKHR with `pVideoProfile->videoCodecOperation` specified as `VK_VIDEO_CODEC_OPERATION_ENCODE_H265_BIT_KHR`, the VkVideoEncodeH265QualityLevelPropertiesKHR structure **must** be included in the `pNext` chain of the VkVideoEncodeQualityLevelPropertiesKHR structure to retrieve additional video encode quality level properties specific to H.265 encoding.

The VkVideoEncodeH265QualityLevelPropertiesKHR structure is defined as:

```
// Provided by VK_KHR_video_encode_h265
typedef struct VkVideoEncodeH265QualityLevelPropertiesKHR {
    VkStructureType                          sType;
    void*                                    pNext;
    VkVideoEncodeH265RateControlFlagsKHR     preferredRateControlFlags;
    uint32_t                                 preferredGopFrameCount;
    uint32_t                                 preferredIdrPeriod;
    uint32_t                                 preferredConsecutiveBFrameCount;
    uint32_t                                 preferredSubLayerCount;
    VkVideoEncodeH265QpKHR                   preferredConstantQp;
    uint32_t                                 preferredMaxL0ReferenceCount;
    uint32_t                                 preferredMaxL1ReferenceCount;
} VkVideoEncodeH265QualityLevelPropertiesKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `preferredRateControlFlags` is a bitmask of VkVideoEncodeH265RateControlFlagBitsKHR values indicating the preferred flags to use for VkVideoEncodeH265RateControlInfoKHR::`flags`.
- `preferredGopFrameCount` indicates the preferred value to use for VkVideoEncodeH265RateControlInfoKHR::`gopFrameCount`.
- `preferredIdrPeriod` indicates the preferred value to use for VkVideoEncodeH265RateControlInfoKHR::`idrPeriod`.
- `preferredConsecutiveBFrameCount` indicates the preferred value to use for VkVideoEncodeH265RateControlInfoKHR::`consecutiveBFrameCount`.
- `preferredSubLayerCount` indicates the preferred value to use for VkVideoEncodeH265RateControlInfoKHR::`subLayerCount`.
- `preferredConstantQp` indicates the preferred values to use for VkVideoEncodeH265NaluSliceSegmentInfoKHR::`constantQp` for each picture type when using rate control mode `VK_VIDEO_ENCODE_RATE_CONTROL_MODE_DISABLED_BIT_KHR`.
- `preferredMaxL0ReferenceCount` indicates the preferred maximum number of reference pictures to use in the reference list L0.
- `preferredMaxL1ReferenceCount` indicates the preferred maximum number of reference pictures to use in the reference list L1.

### Valid Usage (Implicit)

- VUID-VkVideoEncodeH265QualityLevelPropertiesKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_VIDEO_ENCODE_H265_QUALITY_LEVEL_PROPERTIES_KHR`

## 42.18.9. H.265 Encode Session

Additional parameters can be specified when creating a video session with an H.265 encode profile by including an instance of the VkVideoEncodeH265SessionCreateInfoKHR structure in the `pNext`

chain of VkVideoSessionCreateInfoKHR.

The VkVideoEncodeH265SessionCreateInfoKHR structure is defined as:

```
// Provided by VK_KHR_video_encode_h265
typedef struct VkVideoEncodeH265SessionCreateInfoKHR {
    VkStructureType         sType;
    const void*             pNext;
    VkBool32                useMaxLevelIdc;
    StdVideoH265LevelIdc    maxLevelIdc;
} VkVideoEncodeH265SessionCreateInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- useMaxLevelIdc indicates whether the value of maxLevelIdc should be used by the implementation. When it is set to VK_FALSE, the implementation ignores the value of maxLevelIdc and uses the value of VkVideoEncodeH265CapabilitiesKHR::maxLevelIdc, as reported by vkGetPhysicalDeviceVideoCapabilitiesKHR for the video profile.
- maxLevelIdc is a StdVideoH265LevelIdc value specifying the upper bound on the H.265 level for the video bitstreams produced by the created video session, where enum constant STD_VIDEO_H265_LEVEL_IDC_<major>_<minor> identifies H.265 level <major>.<minor> as defined in section A.4 of the ITU-T H.265 Specification.

**Valid Usage (Implicit)**

- VUID-VkVideoEncodeH265SessionCreateInfoKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_VIDEO_ENCODE_H265_SESSION_CREATE_INFO_KHR

## 42.18.10. H.265 Encode Parameter Sets

Video session parameters objects created with the video codec operation VK_VIDEO_CODEC_OPERATION_ENCODE_H265_BIT_KHR **can** contain the following types of parameters:

**H.265 Video Parameter Sets (VPS)**

Represented by StdVideoH265VideoParameterSet structures and interpreted as follows:

- reserved1, reserved2, and reserved3 are used only for padding purposes and are otherwise ignored;
- vps_video_parameter_set_id is used as the key of the VPS entry;
- the max_latency_increase_plus1, max_dec_pic_buffering_minus1, and max_num_reorder_pics members of the StdVideoH265DecPicBufMgr structure pointed to by pDecPicBufMgr correspond to vps_max_latency_increase_plus1, vps_max_dec_pic_buffering_minus1, and vps_max_num_reorder_pics, respectively, as defined in section 7.4.3.1 of the ITU-T H.265

Specification;

- the `StdVideoH265HrdParameters` structure pointed to by `pHrdParameters` is interpreted as follows:
  - `reserved` is used only for padding purposes and is otherwise ignored;
  - `flags.fixed_pic_rate_general_flag` is a bitmask where bit index i corresponds to `fixed_pic_rate_general_flag[i]` as defined in section E.3.2 of the ITU-T H.265 Specification;
  - `flags.fixed_pic_rate_within_cvs_flag` is a bitmask where bit index i corresponds to `fixed_pic_rate_within_cvs_flag[i]` as defined in section E.3.2 of the ITU-T H.265 Specification;
  - `flags.low_delay_hrd_flag` is a bitmask where bit index i corresponds to `low_delay_hrd_flag[i]` as defined in section E.3.2 of the ITU-T H.265 Specification;
  - if `flags.nal_hrd_parameters_present_flag` is set, then `pSubLayerHrdParametersNal` is a pointer to an array of `vps_max_sub_layers_minus1` + 1 number of `StdVideoH265SubLayerHrdParameters` structures where `vps_max_sub_layers_minus1` is the corresponding member of the encompassing `StdVideoH265VideoParameterSet` structure and each element is interpreted as follows:
    - `cbr_flag` is a bitmask where bit index i corresponds to `cbr_flag[i]` as defined in section E.3.3 of the ITU-T H.265 Specification;
    - all other members of the `StdVideoH265SubLayerHrdParameters` structure are interpreted as defined in section E.3.3 of the ITU-T H.265 Specification;
  - if `flags.vcl_hrd_parameters_present_flag` is set, then `pSubLayerHrdParametersVcl` is a pointer to an array of `vps_max_sub_layers_minus1` + 1 number of `StdVideoH265SubLayerHrdParameters` structures where `vps_max_sub_layers_minus1` is the corresponding member of the encompassing `StdVideoH265VideoParameterSet` structure and each element is interpreted as follows:
    - `cbr_flag` is a bitmask where bit index i corresponds to `cbr_flag[i]` as defined in section E.3.3 of the ITU-T H.265 Specification;
    - all other members of the `StdVideoH265SubLayerHrdParameters` structure are interpreted as defined in section E.3.3 of the ITU-T H.265 Specification;
  - all other members of `StdVideoH265HrdParameters` are interpreted as defined in section E.3.2 of the ITU-T H.265 Specification;
- the `StdVideoH265ProfileTierLevel` structure pointed to by `pProfileTierLevel` are interpreted as follows:
  - `general_level_idc` is one of the enum constants `STD_VIDEO_H265_LEVEL_IDC_<major>_<minor>` identifying the H.265 level `<major>.<minor>` as defined in section A.4 of the ITU-T H.265 Specification;
  - all other members of `StdVideoH265ProfileTierLevel` are interpreted as defined in section 7.4.4 of the ITU-T H.265 Specification;
- all other members of `StdVideoH265VideoParameterSet` are interpreted as defined in section 7.4.3.1 of the ITU-T H.265 Specification.

**H.265 Sequence Parameter Sets (SPS)**

Represented by `StdVideoH265SequenceParameterSet` structures and interpreted as follows:

- `reserved1` and `reserved2` are used only for padding purposes and are otherwise ignored;
- the pair constructed from `sps_video_parameter_set_id` and `sps_seq_parameter_set_id` is used as the key of the SPS entry;
- the `StdVideoH265ProfileTierLevel` structure pointed to by `pProfileTierLevel` are interpreted as follows:
  - `general_level_idc` is one of the enum constants `STD_VIDEO_H265_LEVEL_IDC_<major>_<minor>` identifying the H.265 level `<major>.<minor>` as defined in section A.4 of the ITU-T H.265 Specification;
  - all other members of `StdVideoH265ProfileTierLevel` are interpreted as defined in section 7.4.4 of the ITU-T H.265 Specification;
- the `max_latency_increase_plus1`, `max_dec_pic_buffering_minus1`, and `max_num_reorder_pics` members of the `StdVideoH265DecPicBufMgr` structure pointed to by `pDecPicBufMgr` correspond to `sps_max_latency_increase_plus1`, `sps_max_dec_pic_buffering_minus1`, and `sps_max_num_reorder_pics`, respectively, as defined in section 7.4.3.2 of the ITU-T H.265 Specification;
- if `flags.sps_scaling_list_data_present_flag` is set, then the `StdVideoH265ScalingLists` structure pointed to by `pScalingLists` is interpreted as follows:
  - `ScalingList4x4`, `ScalingList8x8`, `ScalingList16x16`, and `ScalingList32x32` correspond to `ScalingList[0]`, `ScalingList[1]`, `ScalingList[2]`, and `ScalingList[3]`, respectively, as defined in section 7.3.4 of the ITU-T H.265 Specification;
  - `ScalingListDCCoef16x16` and `ScalingListDCCoef32x32` correspond to `scaling_list_dc_coef_minus8[0]` and `scaling_list_dc_coef_minus8[1]`, respectively, as defined in section 7.3.4 of the ITU-T H.265 Specification;
- `pShortTermRefPicSet` is a pointer to an array of `num_short_term_ref_pic_sets` number of `StdVideoH265ShortTermRefPicSet` structures where each element is interpreted as follows:
  - `reserved1`, `reserved2`, and `reserved3` are used only for padding purposes and are otherwise ignored;
  - `used_by_curr_pic_flag` is a bitmask where bit index i corresponds to `used_by_curr_pic_flag[i]` as defined in section 7.4.8 of the ITU-T H.265 Specification;
  - `use_delta_flag` is a bitmask where bit index i corresponds to `use_delta_flag[i]` as defined in section 7.4.8 of the ITU-T H.265 Specification;
  - `used_by_curr_pic_s0_flag` is a bitmask where bit index i corresponds to `used_by_curr_pic_s0_flag[i]` as defined in section 7.4.8 of the ITU-T H.265 Specification;
  - `used_by_curr_pic_s1_flag` is a bitmask where bit index i corresponds to `used_by_curr_pic_s1_flag[i]` as defined in section 7.4.8 of the ITU-T H.265 Specification;
  - all other members of `StdVideoH265ShortTermRefPicSet` are interpreted as defined in section 7.4.8 of the ITU-T H.265 Specification;

- if `flags.long_term_ref_pics_present_flag` is set then the `StdVideoH265LongTermRefPicsSps` structure pointed to by `pLongTermRefPicsSps` is interpreted as follows:
  - `used_by_curr_pic_lt_sps_flag` is a bitmask where bit index i corresponds to `used_by_curr_pic_lt_sps_flag[i]` as defined in section 7.4.3.2 of the ITU-T H.265 Specification;
  - all other members of `StdVideoH265LongTermRefPicsSps` are interpreted as defined in section 7.4.3.2 of the ITU-T H.265 Specification;
- if `flags.vui_parameters_present_flag` is set, then the `StdVideoH265SequenceParameterSetVui` structure pointed to by `pSequenceParameterSetVui` is interpreted as follows:
  - `reserved1`, `reserved2`, and `reserved3` are used only for padding purposes and are otherwise ignored;
  - the `StdVideoH265HrdParameters` structure pointed to by `pHrdParameters` is interpreted as follows:
    - `flags.fixed_pic_rate_general_flag` is a bitmask where bit index i corresponds to `fixed_pic_rate_general_flag[i]` as defined in section E.3.2 of the ITU-T H.265 Specification;
    - `flags.fixed_pic_rate_within_cvs_flag` is a bitmask where bit index i corresponds to `fixed_pic_rate_within_cvs_flag[i]` as defined in section E.3.2 of the ITU-T H.265 Specification;
    - `flags.low_delay_hrd_flag` is a bitmask where bit index i corresponds to `low_delay_hrd_flag[i]` as defined in section E.3.2 of the ITU-T H.265 Specification;
    - if `flags.nal_hrd_parameters_present_flag` is set, then `pSubLayerHrdParametersNal` is a pointer to an array of `sps_max_sub_layers_minus1` + 1 number of `StdVideoH265SubLayerHrdParameters` structures where `sps_max_sub_layers_minus1` is the corresponding member of the encompassing `StdVideoH265SequenceParameterSet` structure and each element is interpreted as follows:
      - `cbr_flag` is a bitmask where bit index i corresponds to `cbr_flag[i]` as defined in section E.3.3 of the ITU-T H.265 Specification;
      - all other members of the `StdVideoH265SubLayerHrdParameters` structure are interpreted as defined in section E.3.3 of the ITU-T H.265 Specification;
    - if `flags.vcl_hrd_parameters_present_flag` is set, then `pSubLayerHrdParametersVcl` is a pointer to an array of `sps_max_sub_layers_minus1` + 1 number of `StdVideoH265SubLayerHrdParameters` structures where `sps_max_sub_layers_minus1` is the corresponding member of the encompassing `StdVideoH265SequenceParameterSet` structure and each element is interpreted as follows:
      - `cbr_flag` is a bitmask where bit index i corresponds to `cbr_flag[i]` as defined in section E.3.3 of the ITU-T H.265 Specification;
      - all other members of the `StdVideoH265SubLayerHrdParameters` structure are interpreted as defined in section E.3.3 of the ITU-T H.265 Specification;
    - all other members of `StdVideoH265HrdParameters` are interpreted as defined in section E.3.2 of the ITU-T H.265 Specification;

    - all other members of `pSequenceParameterSetVui` are interpreted as defined in section E.3.1 of the ITU-T H.265 Specification;
- if `flags.sps_palette_predictor_initializer_present_flag` is set, then the `PredictorPaletteEntries` member of the `StdVideoH265PredictorPaletteEntries` structure pointed to by `pPredictorPaletteEntries` is interpreted as defined in section 7.4.9.13 of the ITU-T H.265 Specification;
- all other members of `StdVideoH265SequenceParameterSet` are interpreted as defined in section 7.4.3.1 of the ITU-T H.265 Specification.

**H.265 Picture Parameter Sets (PPS)**

Represented by `StdVideoH265PictureParameterSet` structures and interpreted as follows:

- `reserved1`, `reserved2`, and `reserved3` are used only for padding purposes and are otherwise ignored;
- the triplet constructed from `sps_video_parameter_set_id`, `pps_seq_parameter_set_id`, and `pps_pic_parameter_set_id` is used as the key of the PPS entry;
- if `flags.pps_scaling_list_data_present_flag` is set, then the `StdVideoH265ScalingLists` structure pointed to by `pScalingLists` is interpreted as follows:
    - `ScalingList4x4`, `ScalingList8x8`, `ScalingList16x16`, and `ScalingList32x32` correspond to `ScalingList[0]`, `ScalingList[1]`, `ScalingList[2]`, and `ScalingList[3]`, respectively, as defined in section 7.3.4 of the ITU-T H.265 Specification;
    - `ScalingListDCCoef16x16` and `ScalingListDCCoef32x32` correspond to `scaling_list_dc_coef_minus8[0]` and `scaling_list_dc_coef_minus8[1]`, respectively, as defined in section 7.3.4 of the ITU-T H.265 Specification;
- if `flags.pps_palette_predictor_initializer_present_flag` is set, then the `PredictorPaletteEntries` member of the `StdVideoH265PredictorPaletteEntries` structure pointed to by `pPredictorPaletteEntries` is interpreted as defined in section 7.4.9.13 of the ITU-T H.265 Specification;
- all other members of `StdVideoH265PictureParameterSet` are interpreted as defined in section 7.4.3.3 of the ITU-T H.265 Specification.

Implementations **may** override any of these parameters according to the semantics defined in the Video Encode Parameter Overrides section before storing the resulting H.265 parameter sets into the video session parameters object. Applications need to use the vkGetEncodedVideoSessionParametersKHR command to determine whether any implementation overrides happened and to retrieve the encoded H.265 parameter sets in order to be able to produce a compliant H.265 video bitstream.

Such H.265 parameter set overrides **may** also have cascading effects on the implementation overrides applied to the encoded bitstream produced by video encode operations. If the implementation supports the `VK_VIDEO_ENCODE_FEEDBACK_BITSTREAM_HAS_OVERRIDES_BIT_KHR` video encode feedback query flag, then the application **can** use such queries to retrieve feedback about whether any implementation overrides have been applied to the encoded bitstream.

When a video session parameters object is created with the codec operation VK_VIDEO_CODEC_OPERATION_ENCODE_H265_BIT_KHR, the VkVideoSessionParametersCreateInfoKHR::pNext chain **must** include a VkVideoEncodeH265SessionParametersCreateInfoKHR structure specifying the capacity and initial contents of the object.

The VkVideoEncodeH265SessionParametersCreateInfoKHR structure is defined as:

```
// Provided by VK_KHR_video_encode_h265
typedef struct VkVideoEncodeH265SessionParametersCreateInfoKHR {
    VkStructureType                                         sType;
    const void*                                             pNext;
    uint32_t                                                maxStdVPSCount;
    uint32_t                                                maxStdSPSCount;
    uint32_t                                                maxStdPPSCount;
    const VkVideoEncodeH265SessionParametersAddInfoKHR*     pParametersAddInfo;
} VkVideoEncodeH265SessionParametersCreateInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- maxStdVPSCount is the maximum number of H.265 VPS entries the created VkVideoSessionParametersKHR **can** contain.
- maxStdSPSCount is the maximum number of H.265 SPS entries the created VkVideoSessionParametersKHR **can** contain.
- maxStdPPSCount is the maximum number of H.265 PPS entries the created VkVideoSessionParametersKHR **can** contain.
- pParametersAddInfo is NULL or a pointer to a VkVideoEncodeH265SessionParametersAddInfoKHR structure specifying H.265 parameters to add upon object creation.

### Valid Usage (Implicit)

- VUID-VkVideoEncodeH265SessionParametersCreateInfoKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_VIDEO_ENCODE_H265_SESSION_PARAMETERS_CREATE_INFO_KHR
- VUID-VkVideoEncodeH265SessionParametersCreateInfoKHR-pParametersAddInfo-parameter
  If pParametersAddInfo is not NULL, pParametersAddInfo **must** be a valid pointer to a valid VkVideoEncodeH265SessionParametersAddInfoKHR structure

The VkVideoEncodeH265SessionParametersAddInfoKHR structure is defined as:

```
// Provided by VK_KHR_video_encode_h265
typedef struct VkVideoEncodeH265SessionParametersAddInfoKHR {
    VkStructureType                             sType;
    const void*                                 pNext;
    uint32_t                                    stdVPSCount;
    const StdVideoH265VideoParameterSet*        pStdVPSs;
    uint32_t                                    stdSPSCount;
    const StdVideoH265SequenceParameterSet*     pStdSPSs;
    uint32_t                                    stdPPSCount;
    const StdVideoH265PictureParameterSet*      pStdPPSs;
} VkVideoEncodeH265SessionParametersAddInfoKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `stdVPSCount` is the number of elements in the `pStdVPSs` array.
- `pStdVPSs` is a pointer to an array of `StdVideoH265VideoParameterSet` structures describing the H.265 VPS entries to add.
- `stdSPSCount` is the number of elements in the `pStdSPSs` array.
- `pStdSPSs` is a pointer to an array of `StdVideoH265SequenceParameterSet` structures describing the H.265 SPS entries to add.
- `stdPPSCount` is the number of elements in the `pStdPPSs` array.
- `pStdPPSs` is a pointer to an array of `StdVideoH265PictureParameterSet` structures describing the H.265 PPS entries to add.

This structure **can** be specified in the following places:

- In the `pParametersAddInfo` member of the VkVideoEncodeH265SessionParametersCreateInfoKHR structure specified in the `pNext` chain of VkVideoSessionParametersCreateInfoKHR used to create a video session parameters object. In this case, if the video codec operation the video session parameters object is created with is `VK_VIDEO_CODEC_OPERATION_ENCODE_H265_BIT_KHR`, then it defines the set of initial parameters to add to the created object (see Creating Video Session Parameters).
- In the `pNext` chain of VkVideoSessionParametersUpdateInfoKHR. In this case, if the video codec operation the video session parameters object to be updated was created with is `VK_VIDEO_CODEC_OPERATION_ENCODE_H265_BIT_KHR`, then it defines the set of parameters to add to it (see Updating Video Session Parameters).

### Valid Usage (Implicit)

- VUID-VkVideoEncodeH265SessionParametersAddInfoKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_VIDEO_ENCODE_H265_SESSION_PARAMETERS_ADD_INFO_KHR`
- VUID-VkVideoEncodeH265SessionParametersAddInfoKHR-pStdVPSs-parameter
  If `stdVPSCount` is not `0`, and `pStdVPSs` is not `NULL`, `pStdVPSs` **must** be a valid pointer to an array of `stdVPSCount` `StdVideoH265VideoParameterSet` values

- VUID-VkVideoEncodeH265SessionParametersAddInfoKHR-pStdSPSs-parameter
  If `stdSPSCount` is not `0`, and `pStdSPSs` is not `NULL`, `pStdSPSs` **must** be a valid pointer to an array of `stdSPSCount` `StdVideoH265SequenceParameterSet` values
- VUID-VkVideoEncodeH265SessionParametersAddInfoKHR-pStdPPSs-parameter
  If `stdPPSCount` is not `0`, and `pStdPPSs` is not `NULL`, `pStdPPSs` **must** be a valid pointer to an array of `stdPPSCount` `StdVideoH265PictureParameterSet` values

### Valid Usage

- VUID-VkVideoEncodeH265SessionParametersAddInfoKHR-None-06438
  The `vps_video_parameter_set_id` member of each `StdVideoH265VideoParameterSet` structure specified in the elements of `pStdVPSs` **must** be unique within `pStdVPSs`
- VUID-VkVideoEncodeH265SessionParametersAddInfoKHR-None-06439
  The pair constructed from the `sps_video_parameter_set_id` and `sps_seq_parameter_set_id` members of each `StdVideoH265SequenceParameterSet` structure specified in the elements of `pStdSPSs` **must** be unique within `pStdSPSs`
- VUID-VkVideoEncodeH265SessionParametersAddInfoKHR-None-06440
  The triplet constructed from the `sps_video_parameter_set_id`, `pps_seq_parameter_set_id`, and `pps_pic_parameter_set_id` members of each `StdVideoH265PictureParameterSet` structure specified in the elements of `pStdPPSs` **must** be unique within `pStdPPSs`

The `VkVideoEncodeH265SessionParametersGetInfoKHR` structure is defined as:

```
// Provided by VK_KHR_video_encode_h265
typedef struct VkVideoEncodeH265SessionParametersGetInfoKHR {
    VkStructureType    sType;
    const void*        pNext;
    VkBool32           writeStdVPS;
    VkBool32           writeStdSPS;
    VkBool32           writeStdPPS;
    uint32_t           stdVPSId;
    uint32_t           stdSPSId;
    uint32_t           stdPPSId;
} VkVideoEncodeH265SessionParametersGetInfoKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `writeStdVPS` indicates whether the encoded H.265 video parameter set identified by `stdVPSId` is requested to be retrieved.
- `writeStdSPS` indicates whether the encoded H.265 sequence parameter set identified by the pair constructed from `stdVPSId` and `stdSPSId` is requested to be retrieved.
- `writeStdPPS` indicates whether the encoded H.265 picture parameter set identified by the triplet constructed from `stdVPSId`, `stdSPSId`, and `stdPPSId` is requested to be retrieved.

- stdVPSId specifies the H.265 video parameter set ID used to identify the retrieved H.265 video, sequence, and/or picture parameter set(s).
- stdSPSId specifies the H.265 sequence parameter set ID used to identify the retrieved H.265 sequence and/or picture parameter set(s) when writeStdSPS and/or writeStdPPS is set to VK_TRUE.
- stdPPSId specifies the H.265 picture parameter set ID used to identify the retrieved H.265 picture parameter set when writeStdPPS is set to VK_TRUE.

When this structure is specified in the pNext chain of the VkVideoEncodeSessionParametersGetInfoKHR structure passed to vkGetEncodedVideoSessionParametersKHR, the command will write encoded parameter data to the output buffer in the following order:

1. The H.265 video parameter set identified by stdVPSId, if writeStdVPS is set to VK_TRUE.
2. The H.265 sequence parameter set identified by the pair constructed from stdVPSId and stdSPSId, if writeStdSPS is set to VK_TRUE.
3. The H.265 picture parameter set identified by the triplet constructed from stdVPSId, stdSPSId, and stdPPSId, if writeStdPPS is set to VK_TRUE.

### Valid Usage

- VUID-VkVideoEncodeH265SessionParametersGetInfoKHR-writeStdVPS-08290
  At least one of writeStdVPS, writeStdSPS, and writeStdPPS **must** be VK_TRUE

### Valid Usage (Implicit)

- VUID-VkVideoEncodeH265SessionParametersGetInfoKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_VIDEO_ENCODE_H265_SESSION_PARAMETERS_GET_INFO_KHR

The VkVideoEncodeH265SessionParametersFeedbackInfoKHR structure is defined as:

```
// Provided by VK_KHR_video_encode_h265
typedef struct VkVideoEncodeH265SessionParametersFeedbackInfoKHR {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           hasStdVPSOverrides;
    VkBool32           hasStdSPSOverrides;
    VkBool32           hasStdPPSOverrides;
} VkVideoEncodeH265SessionParametersFeedbackInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- hasStdVPSOverrides indicates whether any of the parameters of the requested H.265 video parameter set, if one was requested via VkVideoEncodeH265SessionParametersGetInfoKHR

::writeStdVPS, were overridden by the implementation.

- hasStdSPSOverrides indicates whether any of the parameters of the requested H.265 sequence parameter set, if one was requested via VkVideoEncodeH265SessionParametersGetInfoKHR ::writeStdSPS, were overridden by the implementation.
- hasStdPPSOverrides indicates whether any of the parameters of the requested H.265 picture parameter set, if one was requested via VkVideoEncodeH265SessionParametersGetInfoKHR ::writeStdPPS, were overridden by the implementation.

**Valid Usage (Implicit)**

- VUID-VkVideoEncodeH265SessionParametersFeedbackInfoKHR-sType-sType sType **must** be VK_STRUCTURE_TYPE_VIDEO_ENCODE_H265_SESSION_PARAMETERS_FEEDBACK_INFO_KHR

### 42.18.11. H.265 Encoding Parameters

The VkVideoEncodeH265PictureInfoKHR structure is defined as:

```
// Provided by VK_KHR_video_encode_h265
typedef struct VkVideoEncodeH265PictureInfoKHR {
    VkStructureType                                     sType;
    const void*                                         pNext;
    uint32_t                                            naluSliceSegmentEntryCount;
    const VkVideoEncodeH265NaluSliceSegmentInfoKHR*     pNaluSliceSegmentEntries;
    const StdVideoEncodeH265PictureInfo*                pStdPictureInfo;
} VkVideoEncodeH265PictureInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- naluSliceSegmentEntryCount is the number of elements in pNaluSliceSegmentEntries.
- pNaluSliceSegmentEntries is a pointer to an array of naluSliceSegmentEntryCount VkVideoEncodeH265NaluSliceSegmentInfoKHR structures specifying the parameters of the individual H.265 slice segments to encode for the input picture.
- pStdPictureInfo is a pointer to a StdVideoEncodeH265PictureInfo structure specifying H.265 picture information.

This structure is specified in the pNext chain of the VkVideoEncodeInfoKHR structure passed to vkCmdEncodeVideoKHR to specify the codec-specific picture information for an H.265 encode operation.

**Encode Input Picture Information**

When this structure is specified in the pNext chain of the VkVideoEncodeInfoKHR structure passed to vkCmdEncodeVideoKHR, the information related to the encode input picture is defined as follows:

- The image subregion used is determined according to the H.265 Encode Picture Data Access section.
- The encode input picture is associated with the H.265 picture information provided in `pStdPictureInfo`.

**Std Picture Information**

The members of the `StdVideoEncodeH265PictureInfo` structure pointed to by `pStdPictureInfo` are interpreted as follows:

- `flags.reserved` and `reserved1` are used only for padding purposes and are otherwise ignored;
- `flags.is_reference` as defined in section 3.132 of the ITU-T H.265 Specification;
- `flags.IrapPicFlag` as defined in section 3.73 of the ITU-T H.265 Specification;
- `flags.used_for_long_term_reference` is used to indicate whether the picture is marked as "used for long-term reference" as defined in section 8.3.2 of the ITU-T H.265 Specification;
- `flags.discardable_flag` and `cross_layer_bla_flag` as defined in section F.7.4.7.1 of the ITU-T H.265 Specification;
- `pic_type` as defined in section 7.4.3.5 of the ITU-T H.265 Specification;
- `sps_video_parameter_set_id`, `pps_seq_parameter_set_id`, and `pps_pic_parameter_set_id` are used to identify the active parameter sets, as described below;
- `PicOrderCntVal` as defined in section 8.3.1 of the ITU-T H.265 Specification;
- `TemporalId` as defined in section 7.4.2.2 of the ITU-T H.265 Specification;
- if `pRefLists` is not `NULL`, then it is a pointer to a `StdVideoEncodeH265ReferenceListsInfo` structure that is interpreted as follows:
  - `flags.reserved` is used only for padding purposes and is otherwise ignored;
  - `ref_pic_list_modification_flag_l0` and `ref_pic_list_modification_flag_l1` as defined in section 7.4.7.2 of the ITU-T H.265 Specification;
  - `num_ref_idx_l0_active_minus1` and `num_ref_idx_l1_active_minus1` as defined in section 7.4.7.1 of the ITU-T H.265 Specification;
  - `RefPicList0` and `RefPicList1` as defined in section 8.3.4 of the ITU-T H.265 Specification where each element of these arrays either identifies an active reference picture using its DPB slot index or contains the value `STD_VIDEO_H265_NO_REFERENCE_PICTURE` to indicate "no reference picture";
  - `list_entry_l0` and `list_entry_l1` as defined in section 7.4.7.2 of the ITU-T H.265 Specification;
- if `flags.short_term_ref_pic_set_sps_flag` is set, then the `StdVideoH265ShortTermRefPicSet` structure pointed to by `pShortTermRefPicSet` is interpreted as defined for the elements of the `pShortTermRefPicSet` array specified in H.265 sequence parameter sets.
- if `flags.long_term_ref_pics_present_flag` is set in the active SPS, then the `StdVideoEncodeH265LongTermRefPics` structure pointed to by `pLongTermRefPics` is interpreted as follows:
  - `used_by_curr_pic_lt_flag` is a bitmask where bit index i corresponds to

used_by_curr_pic_lt_flag[i] as defined in section 7.4.7.1 of the ITU-T H.265 Specification;

  - all other members of StdVideoEncodeH265LongTermRefPics are interpreted as defined in section 7.4.7.1 of the ITU-T H.265 Specification;

- all other members are interpreted as defined in section 7.4.7.1 of the ITU-T H.265 Specification.

Reference picture setup is controlled by the value of StdVideoEncodeH265PictureInfo::flags.is_reference. If it is set and a reconstructed picture is specified, then the latter is used as the target of picture reconstruction to activate the DPB slot specified in pEncodeInfo->pSetupReferenceSlot->slotIndex. If StdVideoEncodeH265PictureInfo::flags.is_reference is not set, but a reconstructed picture is specified, then the corresponding picture reference associated with the DPB slot is invalidated, as described in the DPB Slot States section.

**Active Parameter Sets**

The members of the StdVideoEncodeH265PictureInfo structure pointed to by pStdPictureInfo are used to select the active parameter sets to use from the bound video session parameters object, as follows:

- The *active VPS* is the VPS identified by the key specified in StdVideoEncodeH265PictureInfo::sps_video_parameter_set_id.
- The *active SPS* is the SPS identified by the key specified by the pair constructed from StdVideoEncodeH265PictureInfo::sps_video_parameter_set_id and StdVideoEncodeH265PictureInfo::pps_seq_parameter_set_id.
- The *active PPS* is the PPS identified by the key specified by the triplet constructed from StdVideoEncodeH265PictureInfo::sps_video_parameter_set_id, StdVideoEncodeH265PictureInfo::pps_seq_parameter_set_id, and StdVideoEncodeH265PictureInfo::pps_pic_parameter_set_id.

H.265 encoding uses *explicit weighted sample prediction* for a slice segment, as defined in section 8.5.3.3.4 of the ITU-T H.265 Specification, if any of the following conditions are true for the active PPS and the pStdSliceSegmentHeader member of the corresponding element of pNaluSliceSegmentEntries:

- pStdSliceSegmentHeader->slice_type is STD_VIDEO_H265_SLICE_TYPE_P and weighted_pred_flag is enabled in the active PPS.
- pStdSliceSegmentHeader->slice_type is STD_VIDEO_H265_SLICE_TYPE_B and weighted_bipred_flag is enabled in the active PPS.

The number of H.265 tiles, as defined in section 3.174 of the ITU-T H.265 Specification, is derived from the num_tile_columns_minus1 and num_tile_rows_minus1 members of the active PPS as follows:

(num_tile_columns_minus1 + 1) × (num_tile_rows_minus1 + 1)

## Valid Usage

- VUID-VkVideoEncodeH265PictureInfoKHR-naluSliceSegmentEntryCount-08306
  naluSliceSegmentEntryCount **must** be between 1 and VkVideoEncodeH265CapabilitiesKHR

::maxSliceSegmentCount, inclusive, as returned by vkGetPhysicalDeviceVideoCapabilitiesKHR for the used video profile

- VUID-VkVideoEncodeH265PictureInfoKHR-flags-08323
  If VkVideoEncodeH265CapabilitiesKHR::flags, as returned by vkGetPhysicalDeviceVideoCapabilitiesKHR for the used video profile, does not include VK_VIDEO_ENCODE_H265_CAPABILITY_MULTIPLE_TILES_PER_SLICE_SEGMENT_BIT_KHR, then naluSliceSegmentEntryCount **must** be greater than or equal to the number of H.265 tiles in the picture
- VUID-VkVideoEncodeH265PictureInfoKHR-flags-08324
  If VkVideoEncodeH265CapabilitiesKHR::flags, as returned by vkGetPhysicalDeviceVideoCapabilitiesKHR for the used video profile, does not include VK_VIDEO_ENCODE_H265_CAPABILITY_MULTIPLE_SLICE_SEGMENTS_PER_TILE_BIT_KHR, then naluSliceSegmentEntryCount **must** be less than or equal to the number of H.265 tiles in the picture
- VUID-VkVideoEncodeH265PictureInfoKHR-flags-08316
  If VkVideoEncodeH265CapabilitiesKHR::flags, as returned by vkGetPhysicalDeviceVideoCapabilitiesKHR for the used video profile, does not include VK_VIDEO_ENCODE_H265_CAPABILITY_PREDICTION_WEIGHT_TABLE_GENERATED_BIT_KHR and the slice segment corresponding to any element of pNaluSliceSegmentEntries uses explicit weighted sample prediction, then VkVideoEncodeH265NaluSliceSegmentInfoKHR::pStdSliceSegmentHeader->pWeightTable **must** not be NULL for that element of pNaluSliceSegmentEntries
- VUID-VkVideoEncodeH265PictureInfoKHR-flags-08317
  If VkVideoEncodeH265CapabilitiesKHR::flags, as returned by vkGetPhysicalDeviceVideoCapabilitiesKHR for the used video profile, does not include VK_VIDEO_ENCODE_H265_CAPABILITY_DIFFERENT_SLICE_SEGMENT_TYPE_BIT_KHR, then VkVideoEncodeH265NaluSliceSegmentInfoKHR::pStdSliceSegmentHeader->slice_type **must** be identical for all elements of pNaluSliceSegmentEntries

## Valid Usage (Implicit)

- VUID-VkVideoEncodeH265PictureInfoKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_VIDEO_ENCODE_H265_PICTURE_INFO_KHR
- VUID-VkVideoEncodeH265PictureInfoKHR-pNaluSliceSegmentEntries-parameter
  pNaluSliceSegmentEntries **must** be a valid pointer to an array of naluSliceSegmentEntryCount valid VkVideoEncodeH265NaluSliceSegmentInfoKHR structures
- VUID-VkVideoEncodeH265PictureInfoKHR-pStdPictureInfo-parameter
  pStdPictureInfo **must** be a valid pointer to a valid StdVideoEncodeH265PictureInfo value
- VUID-VkVideoEncodeH265PictureInfoKHR-naluSliceSegmentEntryCount-arraylength
  naluSliceSegmentEntryCount **must** be greater than 0

The VkVideoEncodeH265NaluSliceSegmentInfoKHR structure is defined as:

```
// Provided by VK_KHR_video_encode_h265
typedef struct VkVideoEncodeH265NaluSliceSegmentInfoKHR {
    VkStructureType                                 sType;
    const void*                                     pNext;
    int32_t                                         constantQp;
    const StdVideoEncodeH265SliceSegmentHeader*     pStdSliceSegmentHeader;
} VkVideoEncodeH265NaluSliceSegmentInfoKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `constantQp` is the QP to use for the slice segment if the current rate control mode configured for the video session is `VK_VIDEO_ENCODE_RATE_CONTROL_MODE_DISABLED_BIT_KHR`.
- `pStdSliceSegmentHeader` is a pointer to a `StdVideoEncodeH265SliceSegmentHeader` structure specifying H.265 slice segment header parameters for the slice segment.

**Std Slice Segment Header Parameters**

The members of the `StdVideoEncodeH265SliceSegmentHeader` structure pointed to by `pStdSliceSegmentHeader` are interpreted as follows:

- `flags.reserved` and `reserved1` are used only for padding purposes and are otherwise ignored;
- if `pWeightTable` is not `NULL`, then it is a pointer to a `StdVideoEncodeH265WeightTable` that is interpreted as follows:
  - `flags.luma_weight_l0_flag`, `flags.chroma_weight_l0_flag`, `flags.luma_weight_l1_flag`, and `flags.chroma_weight_l1_flag` are bitmasks where bit index i corresponds to `luma_weight_l0_flag[i]`, `chroma_weight_l0_flag[i]`, `luma_weight_l1_flag[i]`, and `chroma_weight_l1_flag[i]`, respectively, as defined in section 7.4.7.3 of the ITU-T H.265 Specification;
  - all other members of `StdVideoEncodeH265WeightTable` are interpreted as defined in section 7.4.7.3 of the ITU-T H.265 Specification;
- all other members are interpreted as defined in section 7.4.7.1 of the ITU-T H.265 Specification.

## Valid Usage (Implicit)

- VUID-VkVideoEncodeH265NaluSliceSegmentInfoKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_VIDEO_ENCODE_H265_NALU_SLICE_SEGMENT_INFO_KHR`
- VUID-VkVideoEncodeH265NaluSliceSegmentInfoKHR-pNext-pNext
  `pNext` **must** be `NULL`
- VUID-VkVideoEncodeH265NaluSliceSegmentInfoKHR-pStdSliceSegmentHeader-parameter
  `pStdSliceSegmentHeader` **must** be a valid pointer to a valid `StdVideoEncodeH265SliceSegmentHeader` value

The VkVideoEncodeH265DpbSlotInfoKHR structure is defined as:

```
// Provided by VK_KHR_video_encode_h265
typedef struct VkVideoEncodeH265DpbSlotInfoKHR {
    VkStructureType                             sType;
    const void*                                 pNext;
    const StdVideoEncodeH265ReferenceInfo*      pStdReferenceInfo;
} VkVideoEncodeH265DpbSlotInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- pStdReferenceInfo is a pointer to a StdVideoEncodeH265ReferenceInfo structure specifying H.265 reference information.

This structure is specified in the pNext chain of VkVideoEncodeInfoKHR::pSetupReferenceSlot, if not NULL, and the pNext chain of the elements of VkVideoEncodeInfoKHR::pReferenceSlots to specify the codec-specific reference picture information for an H.265 encode operation.

**Active Reference Picture Information**

When this structure is specified in the pNext chain of the elements of VkVideoEncodeInfoKHR::pReferenceSlots, one element is added to the list of active reference pictures used by the video encode operation for each element of VkVideoEncodeInfoKHR::pReferenceSlots as follows:

- The image subregion used is determined according to the H.265 Encode Picture Data Access section.
- The reference picture is associated with the DPB slot index specified in the slotIndex member of the corresponding element of VkVideoEncodeInfoKHR::pReferenceSlots.
- The reference picture is associated with the H.265 reference information provided in pStdReferenceInfo.

**Reconstructed Picture Information**

When this structure is specified in the pNext chain of VkVideoEncodeInfoKHR::pSetupReferenceSlot, the information related to the reconstructed picture is defined as follows:

- The image subregion used is determined according to the H.265 Encode Picture Data Access section.
- If reference picture setup is requested, then the reconstructed picture is used to activate the DPB slot with the index specified in VkVideoEncodeInfoKHR::pSetupReferenceSlot->slotIndex.
- The reconstructed picture is associated with the H.265 reference information provided in pStdReferenceInfo.

**Std Reference Information**

The members of the StdVideoEncodeH265ReferenceInfo structure pointed to by pStdReferenceInfo are interpreted as follows:

- `flags.reserved` is used only for padding purposes and is otherwise ignored;
- `flags.used_for_long_term_reference` is used to indicate whether the picture is marked as "used for long-term reference" as defined in section 8.3.2 of the ITU-T H.265 Specification;
- `flags.unused_for_reference` is used to indicate whether the picture is marked as "unused for reference" as defined in section 8.3.2 of the ITU-T H.265 Specification;
- `pic_type` as defined in section 7.4.3.5 of the ITU-T H.265 Specification;
- `PicOrderCntVal` as defined in section 8.3.1 of the ITU-T H.265 Specification;
- `TemporalId` as defined in section 7.4.2.2 of the ITU-T H.265 Specification.

**Valid Usage (Implicit)**

- VUID-VkVideoEncodeH265DpbSlotInfoKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_VIDEO_ENCODE_H265_DPB_SLOT_INFO_KHR`
- VUID-VkVideoEncodeH265DpbSlotInfoKHR-pStdReferenceInfo-parameter
  `pStdReferenceInfo` **must** be a valid pointer to a valid `StdVideoEncodeH265ReferenceInfo` value

## 42.18.12. H.265 Encode Rate Control

**Group of Pictures**

In case of H.265 encoding it is common practice to follow a regular pattern of different picture types in display order when encoding subsequent frames. This pattern is referred to as the *group of pictures* (GOP).

A regular GOP is defined by the following parameters:

- The number of frames in the GOP;
- The number of consecutive B frames between I and/or P frames in display order.

GOPs are further classified as *open* and *closed* GOPs.

Frame types in an open GOP follow each other in display order according to the following algorithm:

1. The first frame is always an I frame.
2. This is followed by a number of consecutive B frames, as defined above.
3. If the number of frames in the GOP is not reached yet, then the next frame is a P frame and the algorithm continues from step 2.

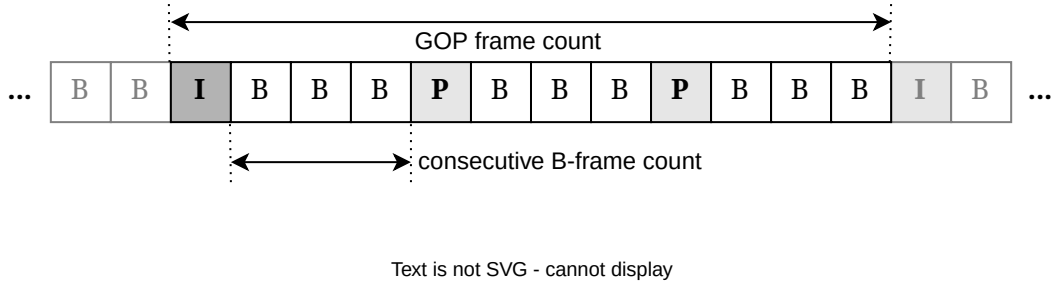



*Figure 36. H.265 open GOP*

In case of a closed GOP, an IDR frame is used at a certain period.

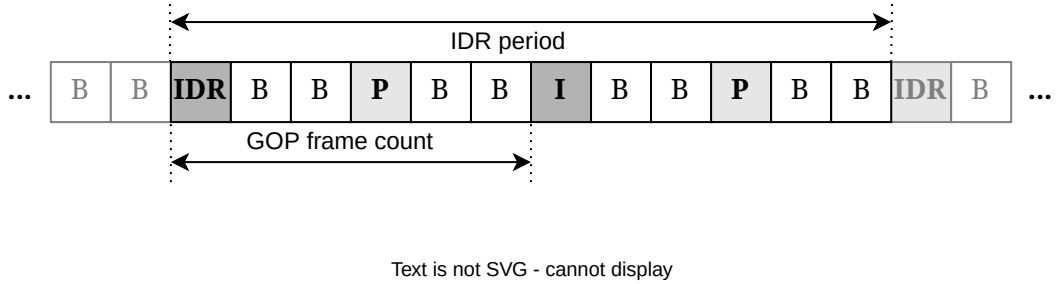



*Figure 37. H.265 closed GOP*

It is also typical for H.265 encoding to use specific reference picture usage patterns across the frames of the GOP. The two most common reference patterns used are as follows:

**Flat Reference Pattern**

- Each P frame uses the last non-B frame, in display order, as reference.
- Each B frame uses the last non-B frame, in display order, as its forward reference, and uses the next non-B frame, in display order, as its backward reference.

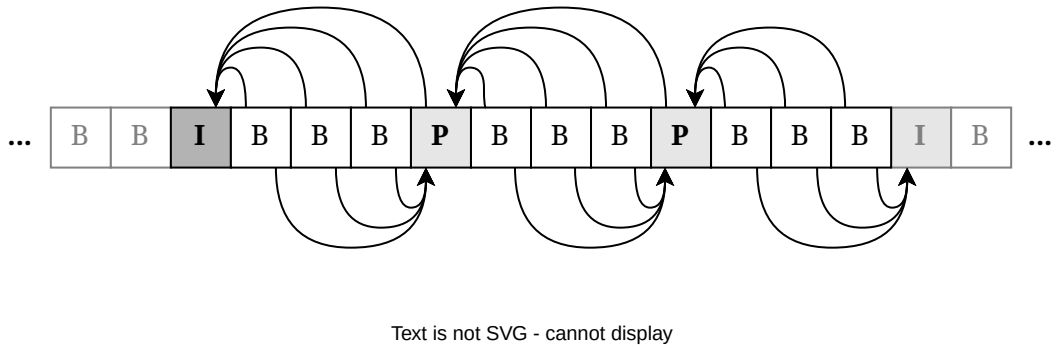
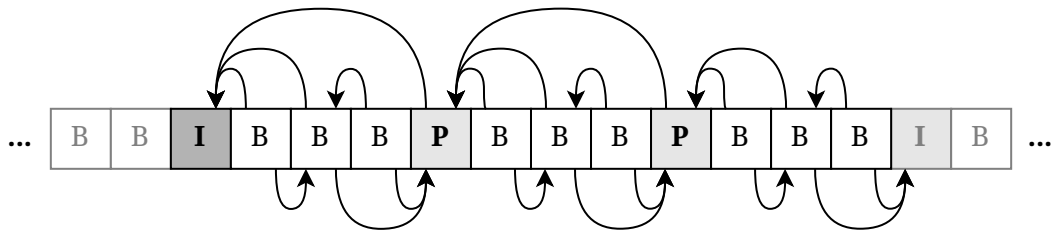



*Figure 38. H.265 flat reference pattern*

**Dyadic Reference Pattern**

- Each P frame uses the last non-B frame, in display order, as reference.

- The following algorithm is applied to the sequence of consecutive B frames between I and/or P frames in display order:
  1. The B frame in the middle of this sequence uses the frame preceding the sequence as its forward reference, and uses the frame following the sequence as its backward reference.
  2. The algorithm is executed recursively for the following frame sequences:
     - The B frames of the original sequence preceding the frame in the middle, if any.
     - The B frames of the original sequence following the frame in the middle, if any.




Text is not SVG - cannot display

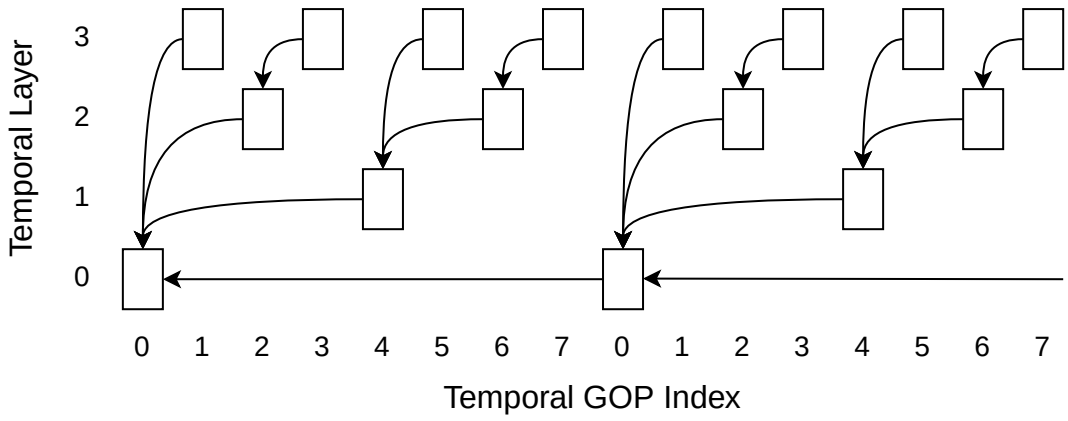
*Figure 39. H.265 dyadic reference pattern*

The application **can** provide guidance to the implementation's rate control algorithm about the structure of the GOP used by the application. Any such guidance about the GOP and its structure does not mandate that specific GOP structure to be used by the application, as the picture type of individual encoded pictures is still application-controlled, however, any deviation from the provided guidance **may** result in undesired rate control behavior including, but not limited, to the implementation not being able to conform to the expected average or target bitrates, or other rate control parameters specified by the application.

When an H.265 encode session is used to encode multiple temporal sub-layers, it is also common practice to follow a regular pattern for the H.265 temporal ID for the encoded pictures in display order when encoding subsequent frames. This pattern is referred to as the *temporal GOP*. The most common temporal layer pattern used is as follows:

**Dyadic Temporal Sub-Layer Pattern**

- The number of frames in the temporal GOP is $2^{n-1}$, where n is the number of temporal sub-layers.
- The $i^{th}$ frame in the temporal GOP uses temporal ID t, if and only if the index of the least significant bit set in i equals n-t-1, except for the first frame, which is the only frame in the temporal GOP using temporal ID zero.
- The $i^{th}$ frame in the temporal GOP uses the $r^{th}$ frame as reference, where r is calculated from i by clearing the least significant bit set in it, except for the first frame in the temporal GOP, which uses the first frame of the previous temporal GOP, if any, as reference.




Text is not SVG - cannot display

*Figure 40. H.265 dyadic temporal sub-layer pattern*

| | |
|---|---|
| **NOTE** | Multi-layer rate control and multi-layer coding are typically used for streaming cases where low latency is expected, hence B pictures with backward prediction are usually not used. |

The `VkVideoEncodeH265RateControlInfoKHR` structure is defined as:

```
// Provided by VK_KHR_video_encode_h265
typedef struct VkVideoEncodeH265RateControlInfoKHR {
    VkStructureType                         sType;
    const void*                             pNext;
    VkVideoEncodeH265RateControlFlagsKHR    flags;
    uint32_t                                gopFrameCount;
    uint32_t                                idrPeriod;
    uint32_t                                consecutiveBFrameCount;
    uint32_t                                subLayerCount;
} VkVideoEncodeH265RateControlInfoKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is a bitmask of VkVideoEncodeH265RateControlFlagBitsKHR specifying H.265 rate control flags.
- `gopFrameCount` is the number of frames within a group of pictures (GOP) intended to be used by the application. If it is set to 0, the rate control algorithm **may** assume an implementation-dependent GOP length. If it is set to `UINT32_MAX`, the GOP length is treated as infinite.
- `idrPeriod` is the interval, in terms of number of frames, between two IDR frames (see IDR period). If it is set to 0, the rate control algorithm **may** assume an implementation-dependent IDR period. If it is set to `UINT32_MAX`, the IDR period is treated as infinite.
- `consecutiveBFrameCount` is the number of consecutive B frames between I and/or P frames within

the GOP.

- subLayerCount specifies the number of H.265 sub-layers that the application intends to use.

When an instance of this structure is included in the pNext chain of the VkVideoCodingControlInfoKHR structure passed to the vkCmdControlVideoCodingKHR command, and VkVideoCodingControlInfoKHR::flags includes VK_VIDEO_CODING_CONTROL_ENCODE_RATE_CONTROL_BIT_KHR, the parameters in this structure are used as guidance for the implementation's rate control algorithm (see Video Coding Control).

If flags includes VK_VIDEO_ENCODE_H265_RATE_CONTROL_ATTEMPT_HRD_COMPLIANCE_BIT_KHR, then the rate control state is reset to an initial state to meet HRD compliance requirements. Otherwise the new rate control state **may** be applied without a reset depending on the implementation and the specified rate control parameters.

**NOTE** It would be possible to infer the picture type to be used when encoding a frame, on the basis of the values provided for consecutiveBFrameCount, idrPeriod, and gopFrameCount, but this inferred picture type will not be used by implementations to override the picture type provided to the video encode operation.

## Valid Usage

- VUID-VkVideoEncodeH265RateControlInfoKHR-flags-08291
  If VkVideoEncodeH265CapabilitiesKHR::flags, as returned by vkGetPhysicalDeviceVideoCapabilitiesKHR for the used video profile, does not include VK_VIDEO_ENCODE_H265_CAPABILITY_HRD_COMPLIANCE_BIT_KHR, then flags **must** not contain VK_VIDEO_ENCODE_H265_RATE_CONTROL_ATTEMPT_HRD_COMPLIANCE_BIT_KHR
- VUID-VkVideoEncodeH265RateControlInfoKHR-flags-08292
  If flags contains VK_VIDEO_ENCODE_H265_RATE_CONTROL_REFERENCE_PATTERN_FLAT_BIT_KHR or VK_VIDEO_ENCODE_H265_RATE_CONTROL_REFERENCE_PATTERN_DYADIC_BIT_KHR, then it **must** also contain VK_VIDEO_ENCODE_H265_RATE_CONTROL_REGULAR_GOP_BIT_KHR
- VUID-VkVideoEncodeH265RateControlInfoKHR-flags-08293
  If flags contains VK_VIDEO_ENCODE_H265_RATE_CONTROL_REFERENCE_PATTERN_FLAT_BIT_KHR, then it **must** not also contain VK_VIDEO_ENCODE_H265_RATE_CONTROL_REFERENCE_PATTERN_DYADIC_BIT_KHR
- VUID-VkVideoEncodeH265RateControlInfoKHR-flags-08294
  If flags contains VK_VIDEO_ENCODE_H265_RATE_CONTROL_REGULAR_GOP_BIT_KHR, then gopFrameCount **must** be greater than 0
- VUID-VkVideoEncodeH265RateControlInfoKHR-idrPeriod-08295
  If idrPeriod is not 0, then it **must** be greater than or equal to gopFrameCount
- VUID-VkVideoEncodeH265RateControlInfoKHR-consecutiveBFrameCount-08296
  If consecutiveBFrameCount is not 0, then it **must** be less than gopFrameCount

### Valid Usage (Implicit)

- VUID-VkVideoEncodeH265RateControlInfoKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_VIDEO_ENCODE_H265_RATE_CONTROL_INFO_KHR`
- VUID-VkVideoEncodeH265RateControlInfoKHR-flags-parameter
  `flags` **must** be a valid combination of VkVideoEncodeH265RateControlFlagBitsKHR values

Bits which **can** be set in VkVideoEncodeH265RateControlInfoKHR::`flags`, specifying H.265 rate control flags, are:

```
// Provided by VK_KHR_video_encode_h265
typedef enum VkVideoEncodeH265RateControlFlagBitsKHR {
    VK_VIDEO_ENCODE_H265_RATE_CONTROL_ATTEMPT_HRD_COMPLIANCE_BIT_KHR = 0x00000001,
    VK_VIDEO_ENCODE_H265_RATE_CONTROL_REGULAR_GOP_BIT_KHR = 0x00000002,
    VK_VIDEO_ENCODE_H265_RATE_CONTROL_REFERENCE_PATTERN_FLAT_BIT_KHR = 0x00000004,
    VK_VIDEO_ENCODE_H265_RATE_CONTROL_REFERENCE_PATTERN_DYADIC_BIT_KHR = 0x00000008,
    VK_VIDEO_ENCODE_H265_RATE_CONTROL_TEMPORAL_SUB_LAYER_PATTERN_DYADIC_BIT_KHR =
0x00000010,
} VkVideoEncodeH265RateControlFlagBitsKHR;
```

- `VK_VIDEO_ENCODE_H265_RATE_CONTROL_ATTEMPT_HRD_COMPLIANCE_BIT_KHR` specifies that rate control **should** attempt to produce an HRD compliant bitstream, as defined in annex C of the ITU-T H.265 Specification.
- `VK_VIDEO_ENCODE_H265_RATE_CONTROL_REGULAR_GOP_BIT_KHR` specifies that the application intends to use a regular GOP structure according to the parameters specified in the `gopFrameCount`, `idrPeriod`, and `consecutiveBFrameCount` members of the VkVideoEncodeH265RateControlInfoKHR structure.
- `VK_VIDEO_ENCODE_H265_RATE_CONTROL_REFERENCE_PATTERN_FLAT_BIT_KHR` specifies that the application intends to follow a flat reference pattern in the GOP.
- `VK_VIDEO_ENCODE_H265_RATE_CONTROL_REFERENCE_PATTERN_DYADIC_BIT_KHR` specifies that the application intends to follow a dyadic reference pattern in the GOP.
- `VK_VIDEO_ENCODE_H265_RATE_CONTROL_TEMPORAL_SUB_LAYER_PATTERN_DYADIC_BIT_KHR` specifies that the application intends to follow a dyadic temporal sub-layer pattern.

```
// Provided by VK_KHR_video_encode_h265
typedef VkFlags VkVideoEncodeH265RateControlFlagsKHR;
```

`VkVideoEncodeH265RateControlFlagsKHR` is a bitmask type for setting a mask of zero or more VkVideoEncodeH265RateControlFlagBitsKHR.

#### Rate Control Layers

The `VkVideoEncodeH265RateControlLayerInfoKHR` structure is defined as:

```
// Provided by VK_KHR_video_encode_h265
typedef struct VkVideoEncodeH265RateControlLayerInfoKHR {
    VkStructureType                  sType;
    const void*                      pNext;
    VkBool32                         useMinQp;
    VkVideoEncodeH265QpKHR           minQp;
    VkBool32                         useMaxQp;
    VkVideoEncodeH265QpKHR           maxQp;
    VkBool32                         useMaxFrameSize;
    VkVideoEncodeH265FrameSizeKHR    maxFrameSize;
} VkVideoEncodeH265RateControlLayerInfoKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `useMinQp` indicates whether the QP values determined by rate control will be clamped to the lower bounds on the QP values specified in `minQp`.
- `minQp` specifies the lower bounds on the QP values, for each picture type, that the implementation's rate control algorithm will use when `useMinQp` is set to `VK_TRUE`.
- `useMaxQp` indicates whether the QP values determined by rate control will be clamped to the upper bounds on the QP values specified in `maxQp`.
- `maxQp` specifies the upper bounds on the QP values, for each picture type, that the implementation's rate control algorithm will use when `useMaxQp` is set to `VK_TRUE`.
- `useMaxFrameSize` indicates whether the implementation's rate control algorithm **should** use the values specified in `maxFrameSize` as the upper bounds on the encoded frame size for each picture type.
- `maxFrameSize` specifies the upper bounds on the encoded frame size, for each picture type, when `useMaxFrameSize` is set to `VK_TRUE`.

When used, the values in `minQp` and `maxQp` guarantee that the effective QP values used by the implementation will respect those lower and upper bounds, respectively. However, limiting the range of QP values that the implementation is able to use will also limit the capabilities of the implementation's rate control algorithm to comply to other constraints. In particular, the implementation **may** not be able to comply to the following:

- The average and/or peak bitrate values to be used for the encoded bitstream specified in the `averageBitrate` and `maxBitrate` members of the VkVideoEncodeRateControlLayerInfoKHR structure.
- The upper bounds on the encoded frame size, for each picture type, specified in the `maxFrameSize` member of `VkVideoEncodeH265RateControlLayerInfoKHR`.

**NOTE** In general, applications need to configure rate control parameters appropriately in order to be able to get the desired rate control behavior, as described in the Video Encode Rate Control section.

When an instance of this structure is included in the `pNext` chain of a

VkVideoEncodeRateControlLayerInfoKHR structure specified in one of the elements of the pLayers array member of the VkVideoEncodeRateControlInfoKHR structure passed to the vkCmdControlVideoCodingKHR command, VkVideoCodingControlInfoKHR::flags includes VK_VIDEO_CODING_CONTROL_ENCODE_RATE_CONTROL_BIT_KHR, and the bound video session was created with the video codec operation VK_VIDEO_CODEC_OPERATION_ENCODE_H265_BIT_KHR, it specifies the H.265-specific rate control parameters of the rate control layer corresponding to that element of pLayers.

## Valid Usage

- VUID-VkVideoEncodeH265RateControlLayerInfoKHR-useMinQp-08297
  If useMinQp is VK_TRUE, then the qpI, qpP, and qpB members of minQp **must** all be between VkVideoEncodeH265CapabilitiesKHR::minQp and VkVideoEncodeH265CapabilitiesKHR::maxQp, as returned by vkGetPhysicalDeviceVideoCapabilitiesKHR for the used video profile
- VUID-VkVideoEncodeH265RateControlLayerInfoKHR-useMaxQp-08298
  If useMaxQp is VK_TRUE, then the qpI, qpP, and qpB members of maxQp **must** all be between VkVideoEncodeH265CapabilitiesKHR::minQp and VkVideoEncodeH265CapabilitiesKHR::maxQp, as returned by vkGetPhysicalDeviceVideoCapabilitiesKHR for the used video profile
- VUID-VkVideoEncodeH265RateControlLayerInfoKHR-useMinQp-08299
  If useMinQp is VK_TRUE and VkVideoEncodeH265CapabilitiesKHR::flags, as returned by vkGetPhysicalDeviceVideoCapabilitiesKHR for the used video profile, does not include VK_VIDEO_ENCODE_H265_CAPABILITY_PER_PICTURE_TYPE_MIN_MAX_QP_BIT_KHR, then the qpI, qpP, and qpB members of minQp **must** all specify the same value
- VUID-VkVideoEncodeH265RateControlLayerInfoKHR-useMaxQp-08300
  If useMaxQp is VK_TRUE and VkVideoEncodeH265CapabilitiesKHR::flags, as returned by vkGetPhysicalDeviceVideoCapabilitiesKHR for the used video profile, does not include VK_VIDEO_ENCODE_H265_CAPABILITY_PER_PICTURE_TYPE_MIN_MAX_QP_BIT_KHR, then the qpI, qpP, and qpB members of maxQp **must** all specify the same value
- VUID-VkVideoEncodeH265RateControlLayerInfoKHR-useMinQp-08375
  If useMinQp and useMaxQp are both VK_TRUE, then the qpI, qpP, and qpB members of minQp **must** all be less than or equal to the respective members of maxQp

## Valid Usage (Implicit)

- VUID-VkVideoEncodeH265RateControlLayerInfoKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_VIDEO_ENCODE_H265_RATE_CONTROL_LAYER_INFO_KHR
- VUID-VkVideoEncodeH265RateControlLayerInfoKHR-minQp-parameter
  minQp **must** be a valid VkVideoEncodeH265QpKHR structure
- VUID-VkVideoEncodeH265RateControlLayerInfoKHR-maxQp-parameter
  maxQp **must** be a valid VkVideoEncodeH265QpKHR structure
- VUID-VkVideoEncodeH265RateControlLayerInfoKHR-maxFrameSize-parameter

maxFrameSize **must** be a valid VkVideoEncodeH265FrameSizeKHR structure

The VkVideoEncodeH265QpKHR structure is defined as:

```
// Provided by VK_KHR_video_encode_h265
typedef struct VkVideoEncodeH265QpKHR {
    int32_t    qpI;
    int32_t    qpP;
    int32_t    qpB;
} VkVideoEncodeH265QpKHR;
```

- qpI is the QP to be used for I pictures.
- qpP is the QP to be used for P pictures.
- qpB is the QP to be used for B pictures.

The VkVideoEncodeH265FrameSizeKHR structure is defined as:

```
// Provided by VK_KHR_video_encode_h265
typedef struct VkVideoEncodeH265FrameSizeKHR {
    uint32_t    frameISize;
    uint32_t    framePSize;
    uint32_t    frameBSize;
} VkVideoEncodeH265FrameSizeKHR;
```

- frameISize is the size in bytes to be used for I frames.
- framePSize is the size in bytes to be used for P frames.
- frameBSize is the size in bytes to be used for B frames.

#### GOP Remaining Frames

Besides session level rate control configuration, the application **can** specify the number of frames per frame type remaining in the group of pictures (GOP).

The VkVideoEncodeH265GopRemainingFrameInfoKHR structure is defined as:

```
// Provided by VK_KHR_video_encode_h265
typedef struct VkVideoEncodeH265GopRemainingFrameInfoKHR {
    VkStructureType    sType;
    const void*        pNext;
    VkBool32           useGopRemainingFrames;
    uint32_t           gopRemainingI;
    uint32_t           gopRemainingP;
    uint32_t           gopRemainingB;
} VkVideoEncodeH265GopRemainingFrameInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- useGopRemainingFrames indicates whether the implementation's rate control algorithm **should** use the values specified in gopRemainingI, gopRemainingP, and gopRemainingB. If useGopRemainingFrames is VK_FALSE, then the values of gopRemainingI, gopRemainingP, and gopRemainingB are ignored.
- gopRemainingI specifies the number of I frames the implementation's rate control algorithm **should** assume to be remaining in the GOP prior to executing the video encode operation.
- gopRemainingP specifies the number of P frames the implementation's rate control algorithm **should** assume to be remaining in the GOP prior to executing the video encode operation.
- gopRemainingB specifies the number of B frames the implementation's rate control algorithm **should** assume to be remaining in the GOP prior to executing the video encode operation.

Setting useGopRemainingFrames to VK_TRUE and including this structure in the pNext chain of VkVideoBeginCodingInfoKHR is only mandatory if the VkVideoEncodeH265CapabilitiesKHR::requiresGopRemainingFrames reported for the used video profile is VK_TRUE. However, implementations **may** use these remaining frame counts, when specified, even when it is not required. In particular, when the application does not use a regular GOP structure, these values **may** provide additional guidance for the implementation's rate control algorithm.

The VkVideoEncodeH265CapabilitiesKHR::prefersGopRemainingFrames capability is also used to indicate that the implementation's rate control algorithm **may** operate more accurately if the application specifies the remaining frame counts using this structure.

As with other rate control guidance values, if the effective order and number of frames encoded by the application are not in line with the remaining frame counts specified in this structure at any given point, then the behavior of the implementation's rate control algorithm **may** deviate from the one expected by the application.

### Valid Usage (Implicit)

- VUID-VkVideoEncodeH265GopRemainingFrameInfoKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_VIDEO_ENCODE_H265_GOP_REMAINING_FRAME_INFO_KHR

## 42.18.13. H.265 Encode Requirements

This section described the **required** H.265 encoding capabilities for physical devices that have at least one queue family that supports the video codec operation VK_VIDEO_CODEC_OPERATION_ENCODE_H265_BIT_KHR, as returned by vkGetPhysicalDeviceQueueFamilyProperties2 in VkQueueFamilyVideoPropertiesKHR::videoCodecOperations.

*Table 59. Required Video Std Header Versions*

| Video Std Header Name | Version |
|---|---|
| vulkan_video_codec_h265std_encode | 1.0.0 |

*Table 60. Required Video Capabilities*

| Video Capability | Requirement | Requirement Type[1] |
|---|---|---|
| **VkVideoCapabilitiesKHR** | | |
| flags | - | min |
| minBitstreamBufferOffsetAlignment | 4096 | max |
| minBitstreamBufferSizeAlignment | 4096 | max |
| pictureAccessGranularity | (64,64) | max |
| minCodedExtent | - | max |
| maxCodedExtent | - | min |
| maxDpbSlots | 0 | min |
| maxActiveReferencePictures | 0 | min |
| **VkVideoEncodeCapabilitiesKHR** | | |
| flags | - | min |
| rateControlModes | - | min |
| maxBitrate | 128000 | min |
| maxQualityLevels | 1 | min |
| encodeInputPictureGranularity | (64,64) | max |
| supportedEncodeFeedbackFlags | VK_VIDEO_ENCODE_FEEDBACK_BITSTREAM_BUFFER_OFFSET_BIT_KHR<br>VK_VIDEO_ENCODE_FEEDBACK_BITSTREAM_BYTES_WRITTEN_BIT_KHR | min |
| **VkVideoEncodeH265CapabilitiesKHR** | | |
| flags | - | min |
| maxLevelIdc | STD_VIDEO_H265_LEVEL_IDC_1_0 | min |
| maxSliceSegmentCount | 1 | min |
| maxTiles | (1,1) | min |
| ctbSizes | at least one bit set | implementation-dependent |
| transformBlockSizes | at least one bit set | implementation-dependent |
| maxPPictureL0ReferenceCount | 0 | min |
| maxBPictureL0ReferenceCount | 0 | min |
| maxL1ReferenceCount | 0 | min |
| maxSubLayerCount | 1 | min |

| Video Capability | Requirement | Requirement Type[1] |
|---|---|---|
| `expectDyadicTemporalSubLayerPattern` | - | implementation-dependent |
| `minQp` | - | max |
| `maxQp` | - | min |
| `prefersGopRemainingFrames` | - | implementation-dependent |
| `requiresGopRemainingFrames` | - | implementation-dependent |
| `stdSyntaxFlags` | - | min |

**1**

The **Requirement Type** column specifies the requirement is either the minimum value all implementations **must** support, the maximum value all implementations **must** support, or the exact value all implementations **must** support. For bitmasks a minimum value is the least bits all implementations **must** set, but they **may** have additional bits set beyond this minimum.

# Chapter 43. Optical Flow

## 43.1. Optical Flow Queues

VK_NV_optical_flow adds an optical flow queue type bit VK_QUEUE_OPTICAL_FLOW_BIT_NV to VkQueueFlagBits. Optical flow operations are supported by queues with an advertised queue capability of VK_QUEUE_OPTICAL_FLOW_BIT_NV. As in the case of other queue types, an application **must** use vkGetPhysicalDeviceQueueFamilyProperties to query whether the physical device has support for the Optical Flow Queue. When the implementation reports the VK_QUEUE_OPTICAL_FLOW_BIT_NV bit for a queue family, it advertises general support for Vulkan queue operations described in Devices and Queues.

## 43.2. Optical Flow Image Formats

To enumerate the supported image formats for a specific optical flow usage, call:

```
// Provided by VK_NV_optical_flow
VkResult vkGetPhysicalDeviceOpticalFlowImageFormatsNV(
    VkPhysicalDevice                            physicalDevice,
    const VkOpticalFlowImageFormatInfoNV*       pOpticalFlowImageFormatInfo,
    uint32_t*                                   pFormatCount,
    VkOpticalFlowImageFormatPropertiesNV*       pImageFormatProperties);
```

- physicalDevice is the physical device being queried.
- pOpticalFlowImageFormatInfo is a pointer to a VkOpticalFlowImageFormatInfoNV structure specifying the optical flow usage for which information is returned.
- pFormatCount is a pointer to an integer related to the number of optical flow properties available or queried, as described below.
- pImageFormatProperties is a pointer to an array of VkOpticalFlowImageFormatPropertiesNV structures in which supported formats and image parameters are returned.

If pImageFormatProperties is NULL, then the number of optical flow properties supported for the given physicalDevice is returned in pFormatCount. Otherwise, pFormatCount **must** point to a variable set by the application to the number of elements in the pImageFormatProperties array, and on return the variable is overwritten with the number of values actually written to pImageFormatProperties. If the value of pFormatCount is less than the number of optical flow properties supported, at most pFormatCount values will be written to pImageFormatProperties, and VK_INCOMPLETE will be returned instead of VK_SUCCESS, to indicate that not all the available values were returned.

Before creating an image to be used as an optical flow frame, obtain the supported image creation parameters by querying with vkGetPhysicalDeviceFormatProperties2 and vkGetPhysicalDeviceImageFormatProperties2 using one of the reported formats and adding VkOpticalFlowImageFormatInfoNV to the pNext chain of VkPhysicalDeviceImageFormatInfo2.

When querying the parameters with vkGetPhysicalDeviceImageFormatProperties2 for images used

for optical flow operations, the VkOpticalFlowImageFormatInfoNV::usage field **must** contain one or more of the bits defined in VkOpticalFlowUsageFlagBitsNV.

### Valid Usage (Implicit)

- VUID-vkGetPhysicalDeviceOpticalFlowImageFormatsNV-physicalDevice-parameter
  physicalDevice **must** be a valid VkPhysicalDevice handle
- VUID-vkGetPhysicalDeviceOpticalFlowImageFormatsNV-pOpticalFlowImageFormatInfo-parameter
  pOpticalFlowImageFormatInfo **must** be a valid pointer to a valid VkOpticalFlowImageFormatInfoNV structure
- VUID-vkGetPhysicalDeviceOpticalFlowImageFormatsNV-pFormatCount-parameter
  pFormatCount **must** be a valid pointer to a uint32_t value
- VUID-vkGetPhysicalDeviceOpticalFlowImageFormatsNV-pImageFormatProperties-parameter
  If the value referenced by pFormatCount is not 0, and pImageFormatProperties is not NULL, pImageFormatProperties **must** be a valid pointer to an array of pFormatCount VkOpticalFlowImageFormatPropertiesNV structures

### Return Codes

**Success**

- VK_SUCCESS
- VK_INCOMPLETE

**Failure**

- VK_ERROR_EXTENSION_NOT_PRESENT
- VK_ERROR_INITIALIZATION_FAILED
- VK_ERROR_FORMAT_NOT_SUPPORTED

**NOTE**

VK_FORMAT_B8G8R8A8_UNORM, VK_FORMAT_R8_UNORM and VK_FORMAT_G8_B8R8_2PLANE_420_UNORM are initially supported for images with optical usage VK_OPTICAL_FLOW_USAGE_INPUT_BIT_NV.

VK_FORMAT_R16G16_SFIXED5_NV is initially supported for images with optical flow usage VK_OPTICAL_FLOW_USAGE_OUTPUT_BIT_NV, VK_OPTICAL_FLOW_USAGE_HINT_BIT_NV and VK_OPTICAL_FLOW_USAGE_GLOBAL_FLOW_BIT_NV.

VK_FORMAT_R8_UINT and VK_FORMAT_R32_UINT are initially supported for images with optical flow usage VK_OPTICAL_FLOW_USAGE_COST_BIT_NV. It is recommended to use VK_FORMAT_R8_UINT because of the lower bandwidth.

The VkOpticalFlowImageFormatInfoNV structure is defined as:

```
// Provided by VK_NV_optical_flow
typedef struct VkOpticalFlowImageFormatInfoNV {
    VkStructureType              sType;
    const void*                  pNext;
    VkOpticalFlowUsageFlagsNV    usage;
} VkOpticalFlowImageFormatInfoNV;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `usage` is a bitmask of VkOpticalFlowUsageFlagBitsNV describing the intended optical flow usage of the image.

### Valid Usage (Implicit)

- VUID-VkOpticalFlowImageFormatInfoNV-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_OPTICAL_FLOW_IMAGE_FORMAT_INFO_NV`
- VUID-VkOpticalFlowImageFormatInfoNV-usage-parameter
  `usage` **must** be a valid combination of VkOpticalFlowUsageFlagBitsNV values
- VUID-VkOpticalFlowImageFormatInfoNV-usage-requiredbitmask
  `usage` **must** not be `0`

Bits which **can** be set in VkOpticalFlowImageFormatInfoNV::`usage`, controlling optical flow usage, are:

```
// Provided by VK_NV_optical_flow
typedef enum VkOpticalFlowUsageFlagBitsNV {
    VK_OPTICAL_FLOW_USAGE_UNKNOWN_NV = 0,
    VK_OPTICAL_FLOW_USAGE_INPUT_BIT_NV = 0x00000001,
    VK_OPTICAL_FLOW_USAGE_OUTPUT_BIT_NV = 0x00000002,
    VK_OPTICAL_FLOW_USAGE_HINT_BIT_NV = 0x00000004,
    VK_OPTICAL_FLOW_USAGE_COST_BIT_NV = 0x00000008,
    VK_OPTICAL_FLOW_USAGE_GLOBAL_FLOW_BIT_NV = 0x00000010,
} VkOpticalFlowUsageFlagBitsNV;
```

- `VK_OPTICAL_FLOW_USAGE_INPUT_BIT_NV` specifies that the image **can** be used as input or reference frame for an optical flow operation.
- `VK_OPTICAL_FLOW_USAGE_OUTPUT_BIT_NV` specifies that the image **can** be used as output flow vector map for an optical flow operation.
- `VK_OPTICAL_FLOW_USAGE_HINT_BIT_NV` specifies that the image **can** be used as hint flow vector map for an optical flow operation.
- `VK_OPTICAL_FLOW_USAGE_COST_BIT_NV` specifies that the image **can** be used as output cost map for an optical flow operation.
- `VK_OPTICAL_FLOW_USAGE_GLOBAL_FLOW_BIT_NV` specifies that the image **can** be used as global flow

vector for an optical flow operation.

```
// Provided by VK_NV_optical_flow
typedef VkFlags   VkOpticalFlowUsageFlagsNV;
```

VkOpticalFlowUsageFlagsNV is a bitmask type for setting a mask of zero or more VkOpticalFlowUsageFlagBitsNV.

The VkOpticalFlowImageFormatPropertiesNV structure is defined as:

```
// Provided by VK_NV_optical_flow
typedef struct VkOpticalFlowImageFormatPropertiesNV {
    VkStructureType    sType;
    const void*        pNext;
    VkFormat           format;
} VkOpticalFlowImageFormatPropertiesNV;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- format is a VkFormat that specifies the format that can be used with the specified optical flow image usages.

**Valid Usage (Implicit)**

- VUID-VkOpticalFlowImageFormatPropertiesNV-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_OPTICAL_FLOW_IMAGE_FORMAT_PROPERTIES_NV
- VUID-VkOpticalFlowImageFormatPropertiesNV-pNext-pNext
  pNext **must** be NULL

# 43.3. Optical Flow Session

## 43.3.1. Optical Flow Session Object

Optical flow session objects are abstracted and represented by VkOpticalFlowSessionNV handles:

```
// Provided by VK_NV_optical_flow
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkOpticalFlowSessionNV)
```

### Creating an Optical Flow Session

To create an optical flow session object, call:

```
// Provided by VK_NV_optical_flow
VkResult vkCreateOpticalFlowSessionNV(
    VkDevice                                    device,
    const VkOpticalFlowSessionCreateInfoNV*     pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkOpticalFlowSessionNV*                     pSession);
```

- `device` is the logical device that creates the optical flow session object.
- `pCreateInfo` is a pointer to a VkOpticalFlowSessionCreateInfoNV structure containing parameters specifying the creation of the optical flow session.
- `pAllocator` controls host memory allocation as described in the Memory Allocation chapter.
- `pSession` is a pointer to a VkOpticalFlowSessionNV handle specifying the optical flow session object which will be created by this function when it returns `VK_SUCCESS`

## Valid Usage (Implicit)

- VUID-vkCreateOpticalFlowSessionNV-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkCreateOpticalFlowSessionNV-pCreateInfo-parameter
  `pCreateInfo` **must** be a valid pointer to a valid VkOpticalFlowSessionCreateInfoNV structure
- VUID-vkCreateOpticalFlowSessionNV-pAllocator-parameter
  If `pAllocator` is not `NULL`, `pAllocator` **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkCreateOpticalFlowSessionNV-pSession-parameter
  `pSession` **must** be a valid pointer to a VkOpticalFlowSessionNV handle

## Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_INITIALIZATION_FAILED`

The VkOpticalFlowSessionCreateInfoNV structure is defined as:

```
// Provided by VK_NV_optical_flow
typedef struct VkOpticalFlowSessionCreateInfoNV {
    VkStructureType                     sType;
    void*                               pNext;
    uint32_t                            width;
    uint32_t                            height;
    VkFormat                            imageFormat;
    VkFormat                            flowVectorFormat;
    VkFormat                            costFormat;
    VkOpticalFlowGridSizeFlagsNV        outputGridSize;
    VkOpticalFlowGridSizeFlagsNV        hintGridSize;
    VkOpticalFlowPerformanceLevelNV     performanceLevel;
    VkOpticalFlowSessionCreateFlagsNV   flags;
} VkOpticalFlowSessionCreateInfoNV;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `width` is the width in pixels of the input or reference frame to be bound to this optical flow session.
- `height` is the height in pixels of the input or reference frame to be bound to this optical flow session.
- `imageFormat` is the VkFormat of the input and reference frame to be bound to this optical flow session.
- `flowVectorFormat` is the VkFormat of the flow vector maps (output or hint) to be bound to this optical flow session.
- `costFormat` is the VkFormat of the cost maps to be bound to this optical flow session.
- `outputGridSize` is exactly one bit of VkOpticalFlowGridSizeFlagsNV specifying the grid size of the output flow and cost maps to be bound to this optical flow session. The size of the output flow and cost maps is determined by `VkOpticalFlowSessionCreateInfoNV::width` and `VkOpticalFlowSessionCreateInfoNV::height` divided by `VkOpticalFlowSessionCreateInfoNV::outputGridSize`.
- `hintGridSize` is one exactly bit of VkOpticalFlowGridSizeFlagsNV specifying the grid size of the hint flow vector maps to be bound to this optical flow session. The size of the hint maps is determined by `VkOpticalFlowSessionCreateInfoNV::width` and `VkOpticalFlowSessionCreateInfoNV::height` divided by `VkOpticalFlowSessionCreateInfoNV::hintGridSize`.
- `performanceLevel` is the VkOpticalFlowPerformanceLevelNV used for this optical flow session.
- `flags` are the VkOpticalFlowSessionCreateFlagsNV used for this optical flow session.

## Valid Usage

- VUID-VkOpticalFlowSessionCreateInfoNV-width-07581
  `width` **must** be greater than or equal to `VkPhysicalDeviceOpticalFlowPropertiesNV::minWidth` and less than or equal to `VkPhysicalDeviceOpticalFlowPropertiesNV::maxWidth`

- VUID-VkOpticalFlowSessionCreateInfoNV-height-07582
  height **must** be greater than or equal to VkPhysicalDeviceOpticalFlowPropertiesNV::minHeight and less than or equal to VkPhysicalDeviceOpticalFlowPropertiesNV::maxHeight
- VUID-VkOpticalFlowSessionCreateInfoNV-imageFormat-07583
  imageFormat **must** be one of the formats returned by vkGetPhysicalDeviceOpticalFlowImageFormatsNV for VK_OPTICAL_FLOW_USAGE_INPUT_BIT_NV
- VUID-VkOpticalFlowSessionCreateInfoNV-flowVectorFormat-07584
  flowVectorFormat **must** be one of the formats returned by vkGetPhysicalDeviceOpticalFlowImageFormatsNV for VK_OPTICAL_FLOW_USAGE_OUTPUT_BIT_NV
- VUID-VkOpticalFlowSessionCreateInfoNV-costFormat-07585
  costFormat **must** be one of the formats returned by vkGetPhysicalDeviceOpticalFlowImageFormatsNV for VK_OPTICAL_FLOW_USAGE_COST_BIT_NV if VK_OPTICAL_FLOW_SESSION_CREATE_ENABLE_COST_BIT_NV is set in flags
- VUID-VkOpticalFlowSessionCreateInfoNV-outputGridSize-07586
  outputGridSize **must** be exactly one of the bits reported in VkPhysicalDeviceOpticalFlowPropertiesNV::supportedOutputGridSizes
- VUID-VkOpticalFlowSessionCreateInfoNV-hintGridSize-07587
  hintGridSize **must** be exactly one of the bits reported in VkPhysicalDeviceOpticalFlowPropertiesNV::supportedHintGridSizes if VK_OPTICAL_FLOW_SESSION_CREATE_ENABLE_HINT_BIT_NV is set in flags
- VUID-VkOpticalFlowSessionCreateInfoNV-flags-07588
  VK_OPTICAL_FLOW_SESSION_CREATE_ENABLE_HINT_BIT_NV **must** not be set in flags if VkPhysicalDeviceOpticalFlowPropertiesNV::hintSupported is VK_FALSE
- VUID-VkOpticalFlowSessionCreateInfoNV-flags-07589
  VK_OPTICAL_FLOW_SESSION_CREATE_ENABLE_COST_BIT_NV **must** not be set in flags if VkPhysicalDeviceOpticalFlowPropertiesNV::costSupported is VK_FALSE
- VUID-VkOpticalFlowSessionCreateInfoNV-flags-07590
  VK_OPTICAL_FLOW_SESSION_CREATE_ENABLE_GLOBAL_FLOW_BIT_NV **must** not be set in flags if VkPhysicalDeviceOpticalFlowPropertiesNV::globalFlowSupported is VK_FALSE
- VUID-VkOpticalFlowSessionCreateInfoNV-flags-07591
  VK_OPTICAL_FLOW_SESSION_CREATE_ALLOW_REGIONS_BIT_NV **must** not be set in flags if VkPhysicalDeviceOpticalFlowPropertiesNV::maxNumRegionsOfInterest is 0
- VUID-VkOpticalFlowSessionCreateInfoNV-flags-07592
  VK_OPTICAL_FLOW_SESSION_CREATE_BOTH_DIRECTIONS_BIT_NV **must** not be set in flags if VkPhysicalDeviceOpticalFlowPropertiesNV::bidirectionalFlowSupported is VK_FALSE

## Valid Usage (Implicit)

- VUID-VkOpticalFlowSessionCreateInfoNV-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_OPTICAL_FLOW_SESSION_CREATE_INFO_NV
- VUID-VkOpticalFlowSessionCreateInfoNV-pNext-pNext

pNext **must** be NULL or a pointer to a valid instance of VkOpticalFlowSessionCreatePrivateDataInfoNV

- VUID-VkOpticalFlowSessionCreateInfoNV-sType-unique
  The sType value of each struct in the pNext chain **must** be unique
- VUID-VkOpticalFlowSessionCreateInfoNV-imageFormat-parameter
  imageFormat **must** be a valid VkFormat value
- VUID-VkOpticalFlowSessionCreateInfoNV-flowVectorFormat-parameter
  flowVectorFormat **must** be a valid VkFormat value
- VUID-VkOpticalFlowSessionCreateInfoNV-costFormat-parameter
  If costFormat is not 0, costFormat **must** be a valid VkFormat value
- VUID-VkOpticalFlowSessionCreateInfoNV-outputGridSize-parameter
  outputGridSize **must** be a valid combination of VkOpticalFlowGridSizeFlagBitsNV values
- VUID-VkOpticalFlowSessionCreateInfoNV-outputGridSize-requiredbitmask
  outputGridSize **must** not be 0
- VUID-VkOpticalFlowSessionCreateInfoNV-hintGridSize-parameter
  hintGridSize **must** be a valid combination of VkOpticalFlowGridSizeFlagBitsNV values
- VUID-VkOpticalFlowSessionCreateInfoNV-performanceLevel-parameter
  If performanceLevel is not 0, performanceLevel **must** be a valid VkOpticalFlowPerformanceLevelNV value
- VUID-VkOpticalFlowSessionCreateInfoNV-flags-parameter
  flags **must** be a valid combination of VkOpticalFlowSessionCreateFlagBitsNV values

The VkOpticalFlowSessionCreatePrivateDataInfoNV structure is for NV internal use only and is defined as:

```
// Provided by VK_NV_optical_flow
typedef struct VkOpticalFlowSessionCreatePrivateDataInfoNV {
    VkStructureType    sType;
    void*              pNext;
    uint32_t           id;
    uint32_t           size;
    const void*        pPrivateData;
} VkOpticalFlowSessionCreatePrivateDataInfoNV;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- id is an identifier for data which is passed at a memory location specified in VkOpticalFlowSessionCreatePrivateDataInfoNV::pPrivateData.
- size is the size of data in bytes which is passed at a memory location specified in VkOpticalFlowSessionCreatePrivateDataInfoNV::pPrivateData.
- pPrivateData is a pointer to NV internal data.

### Valid Usage (Implicit)

- VUID-VkOpticalFlowSessionCreatePrivateDataInfoNV-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_OPTICAL_FLOW_SESSION_CREATE_PRIVATE_DATA_INFO_NV`
- VUID-VkOpticalFlowSessionCreatePrivateDataInfoNV-pPrivateData-parameter
  `pPrivateData` **must** be a pointer value

Optical flow vectors are generated block-wise, one vector for each block of NxN pixels (referred to as grid).

Bits which **can** be set in VkOpticalFlowSessionCreateInfoNV::`outputGridSize` and VkOpticalFlowSessionCreateInfoNV::`hintGridSize`, or which are returned in VkPhysicalDeviceOpticalFlowPropertiesNV::`supportedOutputGridSizes` and VkPhysicalDeviceOpticalFlowPropertiesNV::`supportedHintGridSizes` controlling optical flow grid sizes, are:

```
// Provided by VK_NV_optical_flow
typedef enum VkOpticalFlowGridSizeFlagBitsNV {
    VK_OPTICAL_FLOW_GRID_SIZE_UNKNOWN_NV = 0,
    VK_OPTICAL_FLOW_GRID_SIZE_1X1_BIT_NV = 0x00000001,
    VK_OPTICAL_FLOW_GRID_SIZE_2X2_BIT_NV = 0x00000002,
    VK_OPTICAL_FLOW_GRID_SIZE_4X4_BIT_NV = 0x00000004,
    VK_OPTICAL_FLOW_GRID_SIZE_8X8_BIT_NV = 0x00000008,
} VkOpticalFlowGridSizeFlagBitsNV;
```

- `VK_OPTICAL_FLOW_GRID_SIZE_1X1_BIT_NV` specifies that grid is 1x1 pixel.
- `VK_OPTICAL_FLOW_GRID_SIZE_2X2_BIT_NV` specifies that grid is 2x2 pixel.
- `VK_OPTICAL_FLOW_GRID_SIZE_4X4_BIT_NV` specifies that grid is 4x4 pixel.
- `VK_OPTICAL_FLOW_GRID_SIZE_8X8_BIT_NV` specifies that grid is 8x8 pixel.

```
// Provided by VK_NV_optical_flow
typedef VkFlags   VkOpticalFlowGridSizeFlagsNV;
```

`VkOpticalFlowGridSizeFlagsNV` is a bitmask type for setting a mask of zero or more VkOpticalFlowGridSizeFlagBitsNV.

Optical flow exposes performance levels which the application **can** choose based on the desired performance and quality requirement.

The optical flow performance level types are defined with the following:

```
// Provided by VK_NV_optical_flow
typedef enum VkOpticalFlowPerformanceLevelNV {
    VK_OPTICAL_FLOW_PERFORMANCE_LEVEL_UNKNOWN_NV = 0,
    VK_OPTICAL_FLOW_PERFORMANCE_LEVEL_SLOW_NV = 1,
    VK_OPTICAL_FLOW_PERFORMANCE_LEVEL_MEDIUM_NV = 2,
    VK_OPTICAL_FLOW_PERFORMANCE_LEVEL_FAST_NV = 3,
} VkOpticalFlowPerformanceLevelNV;
```

- VK_OPTICAL_FLOW_PERFORMANCE_LEVEL_SLOW_NV is a level with slower performance but higher quality.
- VK_OPTICAL_FLOW_PERFORMANCE_LEVEL_MEDIUM_NV is a level with medium performance and medium quality.
- VK_OPTICAL_FLOW_PERFORMANCE_LEVEL_FAST_NV is a preset with higher performance but lower quality.

Bits which **can** be set in VkOpticalFlowSessionCreateInfoNV::flags, controlling optical flow session operations, are:

```
// Provided by VK_NV_optical_flow
typedef enum VkOpticalFlowSessionCreateFlagBitsNV {
    VK_OPTICAL_FLOW_SESSION_CREATE_ENABLE_HINT_BIT_NV = 0x00000001,
    VK_OPTICAL_FLOW_SESSION_CREATE_ENABLE_COST_BIT_NV = 0x00000002,
    VK_OPTICAL_FLOW_SESSION_CREATE_ENABLE_GLOBAL_FLOW_BIT_NV = 0x00000004,
    VK_OPTICAL_FLOW_SESSION_CREATE_ALLOW_REGIONS_BIT_NV = 0x00000008,
    VK_OPTICAL_FLOW_SESSION_CREATE_BOTH_DIRECTIONS_BIT_NV = 0x00000010,
} VkOpticalFlowSessionCreateFlagBitsNV;
```

- VK_OPTICAL_FLOW_SESSION_CREATE_ENABLE_HINT_BIT_NV specifies that a VkImageView with external flow vectors will be used as hints in performing the motion search and **must** be bound to VK_OPTICAL_FLOW_SESSION_BINDING_POINT_HINT_NV.
- VK_OPTICAL_FLOW_SESSION_CREATE_ENABLE_COST_BIT_NV specifies that the cost for the forward flow is generated in a VkImageView which **must** be bound to VK_OPTICAL_FLOW_SESSION_BINDING_POINT_COST_NV. Additionally, if VK_OPTICAL_FLOW_SESSION_CREATE_BOTH_DIRECTIONS_BIT_NV is also set, the cost for backward flow is generated in a VkImageView which **must** be bound to VK_OPTICAL_FLOW_SESSION_BINDING_POINT_BACKWARD_COST_NV. The cost is the confidence level of the flow vector for each grid in the frame. The Cost implies how (in)accurate the flow vector is. Higher cost value implies the flow vector to be less accurate and vice-versa.
- VK_OPTICAL_FLOW_SESSION_CREATE_ENABLE_GLOBAL_FLOW_BIT_NV specifies that a global flow vector is estimated from forward flow in a single pixel VkImageView which **must** be bound to VK_OPTICAL_FLOW_SESSION_BINDING_POINT_GLOBAL_FLOW_NV.
- VK_OPTICAL_FLOW_SESSION_CREATE_ALLOW_REGIONS_BIT_NV specifies that regions of interest **can** be specified in VkOpticalFlowExecuteInfoNV.
- VK_OPTICAL_FLOW_SESSION_CREATE_BOTH_DIRECTIONS_BIT_NV specifies that backward flow is

generated in addition to forward flow which is always generated.

VkOpticalFlowSessionCreateFlagsNV is a bitmask type for setting a mask of zero or more VkOpticalFlowSessionCreateFlagBitsNV.

```
// Provided by VK_NV_optical_flow
typedef VkFlags   VkOpticalFlowSessionCreateFlagsNV;
```

**Destroying an Optical Flow Session**

To destroy an optical flow session object, call:

```
// Provided by VK_NV_optical_flow
void vkDestroyOpticalFlowSessionNV(
    VkDevice                                    device,
    VkOpticalFlowSessionNV                      session,
    const VkAllocationCallbacks*                pAllocator);
```

- device is the device that was used for the creation of the optical flow session.
- session is the optical flow session to be destroyed.
- pAllocator controls host memory allocation as described in the Memory Allocation chapter.

**Valid Usage (Implicit)**

- VUID-vkDestroyOpticalFlowSessionNV-device-parameter
  device **must** be a valid VkDevice handle
- VUID-vkDestroyOpticalFlowSessionNV-session-parameter
  session **must** be a valid VkOpticalFlowSessionNV handle
- VUID-vkDestroyOpticalFlowSessionNV-pAllocator-parameter
  If pAllocator is not NULL, pAllocator **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkDestroyOpticalFlowSessionNV-session-parent
  session **must** have been created, allocated, or retrieved from device

## 43.3.2. Binding Vulkan Image Views to an Optical Flow Session

To bind a vulkan image to an optical flow session object, call:

```
// Provided by VK_NV_optical_flow
VkResult vkBindOpticalFlowSessionImageNV(
    VkDevice                                    device,
    VkOpticalFlowSessionNV                      session,
    VkOpticalFlowSessionBindingPointNV          bindingPoint,
    VkImageView                                 view,
    VkImageLayout                               layout);
```

- `device` is the device which owns the optical flow session object `session`.
- `session` is the optical flow session object to which the image view is to be bound.
- `bindingPoint` specifies the binding point VkOpticalFlowSessionBindingPointNV to which the image view is bound.
- `view` is a VkImageView to be bound.
- layout **must** specify the layout that the image subresources accessible from `view` will be in at the time the optical flow vectors are calculated with vkCmdOpticalFlowExecuteNV on a `VkDevice`.

## Valid Usage (Implicit)

- VUID-vkBindOpticalFlowSessionImageNV-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkBindOpticalFlowSessionImageNV-session-parameter
  `session` **must** be a valid VkOpticalFlowSessionNV handle
- VUID-vkBindOpticalFlowSessionImageNV-bindingPoint-parameter
  `bindingPoint` **must** be a valid VkOpticalFlowSessionBindingPointNV value
- VUID-vkBindOpticalFlowSessionImageNV-view-parameter
  If `view` is not VK_NULL_HANDLE, `view` **must** be a valid VkImageView handle
- VUID-vkBindOpticalFlowSessionImageNV-layout-parameter
  `layout` **must** be a valid VkImageLayout value
- VUID-vkBindOpticalFlowSessionImageNV-session-parent
  `session` **must** have been created, allocated, or retrieved from `device`
- VUID-vkBindOpticalFlowSessionImageNV-view-parent
  If `view` is a valid handle, it **must** have been created, allocated, or retrieved from `device`

## Return Codes

### Success

- `VK_SUCCESS`

### Failure

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_INITIALIZATION_FAILED`

The optical flow session binding points are defined with the following:

```
// Provided by VK_NV_optical_flow
typedef enum VkOpticalFlowSessionBindingPointNV {
    VK_OPTICAL_FLOW_SESSION_BINDING_POINT_UNKNOWN_NV = 0,
    VK_OPTICAL_FLOW_SESSION_BINDING_POINT_INPUT_NV = 1,
    VK_OPTICAL_FLOW_SESSION_BINDING_POINT_REFERENCE_NV = 2,
    VK_OPTICAL_FLOW_SESSION_BINDING_POINT_HINT_NV = 3,
    VK_OPTICAL_FLOW_SESSION_BINDING_POINT_FLOW_VECTOR_NV = 4,
    VK_OPTICAL_FLOW_SESSION_BINDING_POINT_BACKWARD_FLOW_VECTOR_NV = 5,
    VK_OPTICAL_FLOW_SESSION_BINDING_POINT_COST_NV = 6,
    VK_OPTICAL_FLOW_SESSION_BINDING_POINT_BACKWARD_COST_NV = 7,
    VK_OPTICAL_FLOW_SESSION_BINDING_POINT_GLOBAL_FLOW_NV = 8,
} VkOpticalFlowSessionBindingPointNV;
```

- `VK_OPTICAL_FLOW_SESSION_BINDING_POINT_INPUT_NV` specifies the binding point for the input frame.
- `VK_OPTICAL_FLOW_SESSION_BINDING_POINT_REFERENCE_NV` specifies the binding point for the input reference frame.
- `VK_OPTICAL_FLOW_SESSION_BINDING_POINT_HINT_NV` specifies the binding point for the optional external hint flow vectors.
- `VK_OPTICAL_FLOW_SESSION_BINDING_POINT_FLOW_VECTOR_NV` specifies the binding point for output flow vectors of default forward flow calculation.
- `VK_OPTICAL_FLOW_SESSION_BINDING_POINT_BACKWARD_FLOW_VECTOR_NV` specifies the binding point for the optional output flow vector map of optional backward flow calculation.
- `VK_OPTICAL_FLOW_SESSION_BINDING_POINT_COST_NV` specifies the binding point for the optional output cost map of default forward flow calculation.
- `VK_OPTICAL_FLOW_SESSION_BINDING_POINT_BACKWARD_COST_NV` specifies the binding point for the optional output cost map of optional backward flow calculation.
- `VK_OPTICAL_FLOW_SESSION_BINDING_POINT_GLOBAL_FLOW_NV` specifies the binding point for the optional global flow value of default forward flow calculation.

### 43.3.3. Optical Flow Execution

Default direction of flow estimation is forward which calculates the optical flow from input frame to reference frame. Optionally backward flow estimation can be additionally calculated. An output flow vector (Vx, Vy) means that current pixel (x, y) of input frame can be found at location (x+Vx, y+Vy) in reference frame. A backward flow vector (Vx, Vy) means that current pixel (x, y) of reference frame can be found at location (x+Vx, y+Vy) in input frame.

To calculate optical flow vectors from two input frames, call:

```
// Provided by VK_NV_optical_flow
void vkCmdOpticalFlowExecuteNV(
    VkCommandBuffer                             commandBuffer,
    VkOpticalFlowSessionNV                      session,
    const VkOpticalFlowExecuteInfoNV*           pExecuteInfo);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `session` is the optical flow session object on which this command is operating.
- `pExecuteInfo` Info is a pointer to a VkOpticalFlowExecuteInfoNV.

## Valid Usage (Implicit)

- VUID-vkCmdOpticalFlowExecuteNV-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdOpticalFlowExecuteNV-session-parameter
  `session` **must** be a valid VkOpticalFlowSessionNV handle
- VUID-vkCmdOpticalFlowExecuteNV-pExecuteInfo-parameter
  `pExecuteInfo` **must** be a valid pointer to a valid VkOpticalFlowExecuteInfoNV structure
- VUID-vkCmdOpticalFlowExecuteNV-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdOpticalFlowExecuteNV-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support optical flow operations
- VUID-vkCmdOpticalFlowExecuteNV-renderpass
  This command **must** only be called outside of a render pass instance
- VUID-vkCmdOpticalFlowExecuteNV-videocoding
  This command **must** only be called outside of a video coding scope
- VUID-vkCmdOpticalFlowExecuteNV-commonparent
  Both of `commandBuffer`, and `session` **must** have been created, allocated, or retrieved from the same VkDevice

## Host Synchronization

- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Outside | Outside | Opticalflow | Action |

The VkOpticalFlowExecuteInfoNV structure is defined as:

```
// Provided by VK_NV_optical_flow
typedef struct VkOpticalFlowExecuteInfoNV {
    VkStructureType                 sType;
    void*                           pNext;
    VkOpticalFlowExecuteFlagsNV     flags;
    uint32_t                        regionCount;
    const VkRect2D*                 pRegions;
} VkOpticalFlowExecuteInfoNV;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` are the VkOpticalFlowExecuteFlagsNV used for this command.
- `regionCount` is the number of regions of interest specified in `pRegions`.
- `pRegions` is a pointer to `regionCount` `VkRect2D` regions of interest.

## Valid Usage

- VUID-VkOpticalFlowExecuteInfoNV-regionCount-07593
  `regionCount` **must** be 0 if `VK_OPTICAL_FLOW_SESSION_CREATE_ALLOW_REGIONS_BIT_NV` was not set for `VkOpticalFlowSessionNV` on which this command is operating

## Valid Usage (Implicit)

- VUID-VkOpticalFlowExecuteInfoNV-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_OPTICAL_FLOW_EXECUTE_INFO_NV`
- VUID-VkOpticalFlowExecuteInfoNV-pNext-pNext
  `pNext` **must** be `NULL`
- VUID-VkOpticalFlowExecuteInfoNV-flags-parameter
  `flags` **must** be a valid combination of VkOpticalFlowExecuteFlagBitsNV values
- VUID-VkOpticalFlowExecuteInfoNV-pRegions-parameter
  If `regionCount` is not `0`, `pRegions` **must** be a valid pointer to an array of `regionCount` VkRect2D structures

Bits which **can** be set in VkOpticalFlowExecuteInfoNV::flags, controlling optical flow execution, are:

```
// Provided by VK_NV_optical_flow
typedef enum VkOpticalFlowExecuteFlagBitsNV {
    VK_OPTICAL_FLOW_EXECUTE_DISABLE_TEMPORAL_HINTS_BIT_NV = 0x00000001,
} VkOpticalFlowExecuteFlagBitsNV;
```

- VK_OPTICAL_FLOW_EXECUTE_DISABLE_TEMPORAL_HINTS_BIT_NV specifies that temporal hints from previously generated flow vectors are not used. If temporal hints are enabled, optical flow vectors from previous vkCmdOpticalFlowExecuteNV call are automatically used as hints for the current vkCmdOpticalFlowExecuteNV call, to take advantage of temporal correlation in a video sequence. Temporal hints should be disabled if there is a-priori knowledge of no temporal correlation (e.g. a scene change, independent successive frame pairs).

VkOpticalFlowExecuteFlagsNV is a bitmask type for setting a mask of zero or more VkOpticalFlowExecuteFlagBitsNV.

```
// Provided by VK_NV_optical_flow
typedef VkFlags   VkOpticalFlowExecuteFlagsNV;
```

# Chapter 44. Execution Graphs

*Execution graphs* provide a way for applications to dispatch multiple operations dynamically from a single initial command on the host. To achieve this, a new execution graph pipeline is provided, that links together multiple shaders or pipelines which each describe one or more operations that can be dispatched within the execution graph. Each linked pipeline or shader describes an *execution node* within the graph, which **can** be dispatched dynamically from another shader within the same graph. This allows applications to describe much richer execution topologies at a finer granularity than would typically be possible with API commands alone.

## 44.1. Pipeline Creation

To create execution graph pipelines, call:

```
// Provided by VK_AMDX_shader_enqueue
VkResult vkCreateExecutionGraphPipelinesAMDX(
    VkDevice                                    device,
    VkPipelineCache                             pipelineCache,
    uint32_t                                    createInfoCount,
    const VkExecutionGraphPipelineCreateInfoAMDX* pCreateInfos,
    const VkAllocationCallbacks*                pAllocator,
    VkPipeline*                                 pPipelines);
```

- `device` is the logical device that creates the execution graph pipelines.
- `pipelineCache` is either VK_NULL_HANDLE, indicating that pipeline caching is disabled; or the handle of a valid pipeline cache object, in which case use of that cache is enabled for the duration of the command.
- `createInfoCount` is the length of the `pCreateInfos` and `pPipelines` arrays.
- `pCreateInfos` is a pointer to an array of VkExecutionGraphPipelineCreateInfoAMDX structures.
- `pAllocator` controls host memory allocation as described in the Memory Allocation chapter.
- `pPipelines` is a pointer to an array of VkPipeline handles in which the resulting execution graph pipeline objects are returned.

Pipelines are created and returned as described for Multiple Pipeline Creation.

### Valid Usage

- VUID-vkCreateExecutionGraphPipelinesAMDX-shaderEnqueue-09124
  The `shaderEnqueue` feature **must** be enabled
- VUID-vkCreateExecutionGraphPipelinesAMDX-flags-09125
  If the `flags` member of any element of `pCreateInfos` contains the `VK_PIPELINE_CREATE_DERIVATIVE_BIT` flag, and the `basePipelineIndex` member of that same element is not `-1`, `basePipelineIndex` **must** be less than the index into `pCreateInfos` that corresponds to that element

- VUID-vkCreateExecutionGraphPipelinesAMDX-flags-09126
  If the `flags` member of any element of `pCreateInfos` contains the `VK_PIPELINE_CREATE_DERIVATIVE_BIT` flag, the base pipeline **must** have been created with the `VK_PIPELINE_CREATE_ALLOW_DERIVATIVES_BIT` flag set
- VUID-vkCreateExecutionGraphPipelinesAMDX-pipelineCache-09127
  If `pipelineCache` was created with `VK_PIPELINE_CACHE_CREATE_EXTERNALLY_SYNCHRONIZED_BIT`, host access to `pipelineCache` **must** be externally synchronized
- VUID-vkCreateExecutionGraphPipelinesAMDX-pNext-09616
  If VkPipelineBinaryInfoKHR::`binaryCount` is not `0` for any element of `pCreateInfos`, `pipelineCache` **must** be VK_NULL_HANDLE
- VUID-vkCreateExecutionGraphPipelinesAMDX-pNext-09617
  If a VkPipelineCreateFlags2CreateInfoKHR structure with the `VK_PIPELINE_CREATE_2_CAPTURE_DATA_BIT_KHR` flag set is included in the `pNext` chain of any element of `pCreateInfos`, `pipelineCache` **must** be VK_NULL_HANDLE
- VUID-vkCreateExecutionGraphPipelinesAMDX-binaryCount-09620
  If VkPipelineBinaryInfoKHR::`binaryCount` is not `0` for any element of `pCreateInfos`, `VK_PIPELINE_CREATION_FEEDBACK_APPLICATION_PIPELINE_CACHE_HIT_BIT` **must** not be set in the `flags` of that element
- VUID-vkCreateExecutionGraphPipelinesAMDX-binaryCount-09621
  If VkPipelineBinaryInfoKHR::`binaryCount` is not `0` for any element of `pCreateInfos`, `VK_PIPELINE_CREATION_FEEDBACK_BASE_PIPELINE_ACCELERATION_BIT` **must** not be set in the `flags` of that element
- VUID-vkCreateExecutionGraphPipelinesAMDX-binaryCount-09622
  If VkPipelineBinaryInfoKHR::`binaryCount` is not `0` for any element of `pCreateInfos`, `VK_PIPELINE_CREATE_FAIL_ON_PIPELINE_COMPILE_REQUIRED_BIT_EXT` **must** not be set in the `flags` of that element

## Valid Usage (Implicit)

- VUID-vkCreateExecutionGraphPipelinesAMDX-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkCreateExecutionGraphPipelinesAMDX-pipelineCache-parameter
  If `pipelineCache` is not VK_NULL_HANDLE, `pipelineCache` **must** be a valid VkPipelineCache handle
- VUID-vkCreateExecutionGraphPipelinesAMDX-pCreateInfos-parameter
  `pCreateInfos` **must** be a valid pointer to an array of `createInfoCount` valid VkExecutionGraphPipelineCreateInfoAMDX structures
- VUID-vkCreateExecutionGraphPipelinesAMDX-pAllocator-parameter
  If `pAllocator` is not `NULL`, `pAllocator` **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkCreateExecutionGraphPipelinesAMDX-pPipelines-parameter
  `pPipelines` **must** be a valid pointer to an array of `createInfoCount` VkPipeline handles

- VUID-vkCreateExecutionGraphPipelinesAMDX-createInfoCount-arraylength
  `createInfoCount` **must** be greater than `0`
- VUID-vkCreateExecutionGraphPipelinesAMDX-pipelineCache-parent
  If `pipelineCache` is a valid handle, it **must** have been created, allocated, or retrieved from `device`

## Return Codes

**Success**

- `VK_SUCCESS`
- `VK_PIPELINE_COMPILE_REQUIRED_EXT`

**Failure**

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`

The `VkExecutionGraphPipelineCreateInfoAMDX` structure is defined as:

```
// Provided by VK_AMDX_shader_enqueue
typedef struct VkExecutionGraphPipelineCreateInfoAMDX {
    VkStructureType                             sType;
    const void*                                 pNext;
    VkPipelineCreateFlags                       flags;
    uint32_t                                    stageCount;
    const VkPipelineShaderStageCreateInfo*      pStages;
    const VkPipelineLibraryCreateInfoKHR*       pLibraryInfo;
    VkPipelineLayout                            layout;
    VkPipeline                                  basePipelineHandle;
    int32_t                                     basePipelineIndex;
} VkExecutionGraphPipelineCreateInfoAMDX;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is a bitmask of VkPipelineCreateFlagBits specifying how the pipeline will be generated.
- `stageCount` is the number of entries in the `pStages` array.
- `pStages` is a pointer to an array of `stageCount` VkPipelineShaderStageCreateInfo structures describing the set of the shader stages to be included in the execution graph pipeline.
- `pLibraryInfo` is a pointer to a VkPipelineLibraryCreateInfoKHR structure defining pipeline libraries to include.
- `layout` is the description of binding locations used by both the pipeline and descriptor sets used with the pipeline.
- `basePipelineHandle` is a pipeline to derive from

- basePipelineIndex is an index into the pCreateInfos parameter to use as a pipeline to derive from

The parameters basePipelineHandle and basePipelineIndex are described in more detail in Pipeline Derivatives.

Each shader stage provided when creating an execution graph pipeline (including those in libraries) is associated with a name and an index, determined by the inclusion or omission of a VkPipelineShaderStageNodeCreateInfoAMDX structure in its pNext chain.

In addition to the shader name and index, an internal "node index" is also generated for each node, which can be queried with vkGetExecutionGraphPipelineNodeIndexAMDX, and is used exclusively for initial dispatch of an execution graph.

## Valid Usage

- VUID-VkExecutionGraphPipelineCreateInfoAMDX-None-09497
  If the pNext chain does not include a VkPipelineCreateFlags2CreateInfoKHR structure, flags **must** be a valid combination of VkPipelineCreateFlagBits values
- VUID-VkExecutionGraphPipelineCreateInfoAMDX-flags-07984
  If flags contains the VK_PIPELINE_CREATE_DERIVATIVE_BIT flag, and basePipelineIndex is -1, basePipelineHandle **must** be a valid execution graph VkPipeline handle
- VUID-VkExecutionGraphPipelineCreateInfoAMDX-flags-07985
  If flags contains the VK_PIPELINE_CREATE_DERIVATIVE_BIT flag, and basePipelineHandle is VK_NULL_HANDLE, basePipelineIndex **must** be a valid index into the calling command's pCreateInfos parameter
- VUID-VkExecutionGraphPipelineCreateInfoAMDX-flags-07986
  If flags contains the VK_PIPELINE_CREATE_DERIVATIVE_BIT flag, basePipelineIndex **must** be -1 or basePipelineHandle **must** be VK_NULL_HANDLE
- VUID-VkExecutionGraphPipelineCreateInfoAMDX-layout-07987
  If a push constant block is declared in a shader, a push constant range in layout **must** match the shader stage
- VUID-VkExecutionGraphPipelineCreateInfoAMDX-layout-10069
  If a push constant block is declared in a shader, the block must be contained inside the push constant range in layout that matches the stage
- VUID-VkExecutionGraphPipelineCreateInfoAMDX-layout-07988
  If a resource variables is declared in a shader, a descriptor slot in layout **must** match the shader stage
- VUID-VkExecutionGraphPipelineCreateInfoAMDX-layout-07990
  If a resource variables is declared in a shader, and the descriptor type is not VK_DESCRIPTOR_TYPE_MUTABLE_EXT, a descriptor slot in layout **must** match the descriptor type
- VUID-VkExecutionGraphPipelineCreateInfoAMDX-layout-07991
  If a resource variables is declared in a shader as an array, a descriptor slot in layout **must** match the descriptor count

- VUID-VkExecutionGraphPipelineCreateInfoAMDX-flags-03365
  `flags` **must** not include `VK_PIPELINE_CREATE_RAY_TRACING_NO_NULL_ANY_HIT_SHADERS_BIT_KHR`
- VUID-VkExecutionGraphPipelineCreateInfoAMDX-flags-03366
  `flags` **must** not include `VK_PIPELINE_CREATE_RAY_TRACING_NO_NULL_CLOSEST_HIT_SHADERS_BIT_KHR`
- VUID-VkExecutionGraphPipelineCreateInfoAMDX-flags-03367
  `flags` **must** not include `VK_PIPELINE_CREATE_RAY_TRACING_NO_NULL_MISS_SHADERS_BIT_KHR`
- VUID-VkExecutionGraphPipelineCreateInfoAMDX-flags-03368
  `flags` **must** not include `VK_PIPELINE_CREATE_RAY_TRACING_NO_NULL_INTERSECTION_SHADERS_BIT_KHR`
- VUID-VkExecutionGraphPipelineCreateInfoAMDX-flags-03369
  `flags` **must** not include `VK_PIPELINE_CREATE_RAY_TRACING_SKIP_TRIANGLES_BIT_KHR`
- VUID-VkExecutionGraphPipelineCreateInfoAMDX-flags-03370
  `flags` **must** not include `VK_PIPELINE_CREATE_RAY_TRACING_SKIP_AABBS_BIT_KHR`
- VUID-VkExecutionGraphPipelineCreateInfoAMDX-flags-03576
  `flags` **must** not include `VK_PIPELINE_CREATE_RAY_TRACING_SHADER_GROUP_HANDLE_CAPTURE_REPLAY_BIT_KHR`
- VUID-VkExecutionGraphPipelineCreateInfoAMDX-flags-04945
  `flags` **must** not include `VK_PIPELINE_CREATE_RAY_TRACING_ALLOW_MOTION_BIT_NV`
- VUID-VkExecutionGraphPipelineCreateInfoAMDX-flags-09007
  If the `VkPhysicalDeviceDeviceGeneratedCommandsComputeFeaturesNV::deviceGeneratedComputePipelines` is not enabled, `flags` **must** not include `VK_PIPELINE_CREATE_INDIRECT_BINDABLE_BIT_NV`
- VUID-VkExecutionGraphPipelineCreateInfoAMDX-flags-09008
  If `flags` includes `VK_PIPELINE_CREATE_INDIRECT_BINDABLE_BIT_NV`, then the `pNext` chain **must** include a pointer to a valid instance of VkComputePipelineIndirectBufferInfoNV specifying the address where the pipeline's metadata will be saved
- VUID-VkExecutionGraphPipelineCreateInfoAMDX-pipelineCreationCacheControl-02875
  If the `pipelineCreationCacheControl` feature is not enabled, `flags` **must** not include `VK_PIPELINE_CREATE_FAIL_ON_PIPELINE_COMPILE_REQUIRED_BIT` or `VK_PIPELINE_CREATE_EARLY_RETURN_ON_FAILURE_BIT`
- VUID-VkExecutionGraphPipelineCreateInfoAMDX-stage-09128
  The `stage` member of any element of `pStages` **must** be `VK_SHADER_STAGE_COMPUTE_BIT`
- VUID-VkExecutionGraphPipelineCreateInfoAMDX-pStages-09129
  The shader code for the entry point identified by each element of `pStages` and the rest of the state identified by this structure **must** adhere to the pipeline linking rules described in the Shader Interfaces chapter
- VUID-VkExecutionGraphPipelineCreateInfoAMDX-layout-09130
  `layout` **must** be consistent with the layout of the shaders specified in `pStages`
- VUID-VkExecutionGraphPipelineCreateInfoAMDX-pLibraryInfo-09131
  If `pLibraryInfo` is not `NULL`, each element of its `pLibraries` member **must** have been created with a `layout` that is compatible with the `layout` in this pipeline

- VUID-VkExecutionGraphPipelineCreateInfoAMDX-layout-09132
  The number of resources in `layout` accessible to each shader stage that is used by the pipeline **must** be less than or equal to `VkPhysicalDeviceLimits::maxPerStageResources`
- VUID-VkExecutionGraphPipelineCreateInfoAMDX-pLibraryInfo-09133
  If `pLibraryInfo` is not `NULL`, each element of `pLibraryInfo->libraries` **must** be either a compute pipeline or an execution graph pipeline
- VUID-VkExecutionGraphPipelineCreateInfoAMDX-None-09134
  There **must** be no two nodes in the pipeline that share both the same shader name and index, as specified by VkPipelineShaderStageNodeCreateInfoAMDX
- VUID-VkExecutionGraphPipelineCreateInfoAMDX-None-09135
  There **must** be no two nodes in the pipeline that share the same shader name and have input payload declarations with different sizes
- VUID-VkExecutionGraphPipelineCreateInfoAMDX-None-09136
  There **must** be no two nodes in the pipeline that share the same name but have different execution models
- VUID-VkExecutionGraphPipelineCreateInfoAMDX-CoalescedInputCountAMDX-09137
  There **must** be no two nodes in the pipeline that share the same name where one includes `CoalescedInputCountAMDX` and the other does not
- VUID-VkExecutionGraphPipelineCreateInfoAMDX-StaticNumWorkgroupsAMDX-09138
  There **must** be no two nodes in the pipeline that share the same name where one includes `StaticNumWorkgroupsAMDX` and the other does not
- VUID-VkExecutionGraphPipelineCreateInfoAMDX-PayloadNodeNameAMDX-09139
  If an output payload declared in any shader in the pipeline has a `PayloadNodeNameAMDX` decoration with a `Node Name` that matches the shader name of any other node in the graph, the size of the output payload **must** match the size of the input payload in the matching node

## Valid Usage (Implicit)

- VUID-VkExecutionGraphPipelineCreateInfoAMDX-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_EXECUTION_GRAPH_PIPELINE_CREATE_INFO_AMDX`
- VUID-VkExecutionGraphPipelineCreateInfoAMDX-pNext-pNext
  Each `pNext` member of any structure (including this one) in the `pNext` chain **must** be either `NULL` or a pointer to a valid instance of VkPipelineCompilerControlCreateInfoAMD or VkPipelineCreationFeedbackCreateInfo
- VUID-VkExecutionGraphPipelineCreateInfoAMDX-sType-unique
  The `sType` value of each struct in the `pNext` chain **must** be unique
- VUID-VkExecutionGraphPipelineCreateInfoAMDX-pStages-parameter
  If `stageCount` is not `0`, and `pStages` is not `NULL`, `pStages` **must** be a valid pointer to an array of `stageCount` valid VkPipelineShaderStageCreateInfo structures
- VUID-VkExecutionGraphPipelineCreateInfoAMDX-pLibraryInfo-parameter
  If `pLibraryInfo` is not `NULL`, `pLibraryInfo` **must** be a valid pointer to a valid

VkPipelineLibraryCreateInfoKHR structure

- VUID-VkExecutionGraphPipelineCreateInfoAMDX-layout-parameter
  `layout` **must** be a valid VkPipelineLayout handle
- VUID-VkExecutionGraphPipelineCreateInfoAMDX-commonparent
  Both of `basePipelineHandle`, and `layout` that are valid handles of non-ignored parameters **must** have been created, allocated, or retrieved from the same VkDevice

`VK_SHADER_INDEX_UNUSED_AMDX` is a special shader index used to indicate that the created node does not override the index. In this case, the shader index is determined through other means. It is defined as:

```
#define VK_SHADER_INDEX_UNUSED_AMDX       (~0U)
```

The `VkPipelineShaderStageNodeCreateInfoAMDX` structure is defined as:

```
// Provided by VK_AMDX_shader_enqueue
typedef struct VkPipelineShaderStageNodeCreateInfoAMDX {
      VkStructureType    sType;
    const void*          pNext;
    const char*          pName;
    uint32_t             index;
} VkPipelineShaderStageNodeCreateInfoAMDX;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `pName` is the shader name to use when creating a node in an execution graph. If `pName` is `NULL`, the name of the entry point specified in SPIR-V is used as the shader name.
- `index` is the shader index to use when creating a node in an execution graph. If `index` is `VK_SHADER_INDEX_UNUSED_AMDX` then the original index is used, either as specified by the `ShaderIndexAMDX` execution mode, or `0` if that too is not specified.

When included in the `pNext` chain of a VkPipelineShaderStageCreateInfo structure, this structure specifies the shader name and shader index of a node when creating an execution graph pipeline. If this structure is omitted, the shader name is set to the name of the entry point in SPIR-V and the shader index is set to `0`.

When dispatching a node from another shader, the name is fixed at pipeline creation, but the index **can** be set dynamically. By associating multiple shaders with the same name but different indexes, applications can dynamically select different nodes to execute. Applications **must** ensure each node has a unique name and index.

### Valid Usage (Implicit)

- VUID-VkPipelineShaderStageNodeCreateInfoAMDX-sType-sType

sType **must** be VK_STRUCTURE_TYPE_PIPELINE_SHADER_STAGE_NODE_CREATE_INFO_AMDX

- VUID-VkPipelineShaderStageNodeCreateInfoAMDX-pName-parameter
  If pName is not NULL, pName **must** be a null-terminated UTF-8 string

To query the internal node index for a particular node in an execution graph, call:

```
// Provided by VK_AMDX_shader_enqueue
VkResult vkGetExecutionGraphPipelineNodeIndexAMDX(
    VkDevice                                    device,
    VkPipeline                                  executionGraph,
    const VkPipelineShaderStageNodeCreateInfoAMDX* pNodeInfo,
    uint32_t*                                   pNodeIndex);
```

- device is the that executionGraph was created on.
- executionGraph is the execution graph pipeline to query the internal node index for.
- pNodeInfo is a pointer to a VkPipelineShaderStageNodeCreateInfoAMDX structure identifying the name and index of the node to query.
- pNodeIndex is the returned internal node index of the identified node.

Once this function returns, the contents of pNodeIndex contain the internal node index of the identified node.

## Valid Usage

- VUID-vkGetExecutionGraphPipelineNodeIndexAMDX-pNodeInfo-09140
  pNodeInfo->pName **must** not be NULL
- VUID-vkGetExecutionGraphPipelineNodeIndexAMDX-pNodeInfo-09141
  pNodeInfo->index **must** not be VK_SHADER_INDEX_UNUSED_AMDX
- VUID-vkGetExecutionGraphPipelineNodeIndexAMDX-executionGraph-09142
  There **must** be a node in executionGraph with a shader name and index equal to pNodeInfo->pName and pNodeInfo->index

## Valid Usage (Implicit)

- VUID-vkGetExecutionGraphPipelineNodeIndexAMDX-device-parameter
  device **must** be a valid VkDevice handle
- VUID-vkGetExecutionGraphPipelineNodeIndexAMDX-executionGraph-parameter
  executionGraph **must** be a valid VkPipeline handle
- VUID-vkGetExecutionGraphPipelineNodeIndexAMDX-pNodeInfo-parameter
  pNodeInfo **must** be a valid pointer to a valid VkPipelineShaderStageNodeCreateInfoAMDX structure
- VUID-vkGetExecutionGraphPipelineNodeIndexAMDX-pNodeIndex-parameter

pNodeIndex **must** be a valid pointer to a uint32_t value

- VUID-vkGetExecutionGraphPipelineNodeIndexAMDX-executionGraph-parent executionGraph **must** have been created, allocated, or retrieved from device

### Return Codes

**Success**

- VK_SUCCESS

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY

## 44.2. Initializing Scratch Memory

Implementations **may** need scratch memory to manage dispatch queues or similar when executing a pipeline graph, and this is explicitly managed by the application.

To query the scratch space required to dispatch an execution graph, call:

```
// Provided by VK_AMDX_shader_enqueue
VkResult vkGetExecutionGraphPipelineScratchSizeAMDX(
    VkDevice                                    device,
    VkPipeline                                  executionGraph,
    VkExecutionGraphPipelineScratchSizeAMDX*    pSizeInfo);
```

- device is the that executionGraph was created on.
- executionGraph is the execution graph pipeline to query the scratch space for.
- pSizeInfo is a pointer to a VkExecutionGraphPipelineScratchSizeAMDX structure that will contain the required scratch size.

After this function returns, information about the scratch space required will be returned in pSizeInfo.

### Valid Usage (Implicit)

- VUID-vkGetExecutionGraphPipelineScratchSizeAMDX-device-parameter device **must** be a valid VkDevice handle
- VUID-vkGetExecutionGraphPipelineScratchSizeAMDX-executionGraph-parameter executionGraph **must** be a valid VkPipeline handle
- VUID-vkGetExecutionGraphPipelineScratchSizeAMDX-pSizeInfo-parameter pSizeInfo **must** be a valid pointer to a VkExecutionGraphPipelineScratchSizeAMDX structure
- VUID-vkGetExecutionGraphPipelineScratchSizeAMDX-executionGraph-parent

executionGraph **must** have been created, allocated, or retrieved from device

### Return Codes

**Success**

- VK_SUCCESS

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY

The VkExecutionGraphPipelineScratchSizeAMDX structure is defined as:

```
// Provided by VK_AMDX_shader_enqueue
typedef struct VkExecutionGraphPipelineScratchSizeAMDX {
    VkStructureType    sType;
    void*              pNext;
    VkDeviceSize       size;
} VkExecutionGraphPipelineScratchSizeAMDX;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- size indicates the scratch space required for dispatch the queried execution graph.

### Valid Usage (Implicit)

- VUID-VkExecutionGraphPipelineScratchSizeAMDX-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_EXECUTION_GRAPH_PIPELINE_SCRATCH_SIZE_AMDX

To initialize scratch memory for a particular execution graph, call:

```
// Provided by VK_AMDX_shader_enqueue
void vkCmdInitializeGraphScratchMemoryAMDX(
    VkCommandBuffer                             commandBuffer,
    VkDeviceAddress                             scratch);
```

- commandBuffer is the command buffer into which the command will be recorded.
- scratch is a pointer to the scratch memory to be initialized.

This command **must** be called before using scratch to dispatch the currently bound execution graph pipeline.

Execution of this command **may** modify any memory locations in the range [scratch,scratch + size), where size is the value returned in VkExecutionGraphPipelineScratchSizeAMDX::size by VkExecutionGraphPipelineScratchSizeAMDX for the currently bound execution graph pipeline.

Accesses to this memory range are performed in the VK_PIPELINE_STAGE_2_COMPUTE_SHADER_BIT pipeline stage with the VK_ACCESS_2_SHADER_STORAGE_READ_BIT and VK_ACCESS_2_SHADER_STORAGE_WRITE_BIT access flags.

If any portion of scratch is modified by any command other than vkCmdDispatchGraphAMDX, vkCmdDispatchGraphIndirectAMDX, vkCmdDispatchGraphIndirectCountAMDX, or vkCmdInitializeGraphScratchMemoryAMDX with the same execution graph, it **must** be reinitialized for the execution graph again before dispatching against it.

## Valid Usage

- VUID-vkCmdInitializeGraphScratchMemoryAMDX-scratch-09143
  scratch **must** be the device address of an allocated memory range at least as large as the value of VkExecutionGraphPipelineScratchSizeAMDX::size returned by VkExecutionGraphPipelineScratchSizeAMDX for the currently bound execution graph pipeline.
- VUID-vkCmdInitializeGraphScratchMemoryAMDX-scratch-09144
  scratch **must** be a multiple of 64

## Valid Usage (Implicit)

- VUID-vkCmdInitializeGraphScratchMemoryAMDX-commandBuffer-parameter
  commandBuffer **must** be a valid VkCommandBuffer handle
- VUID-vkCmdInitializeGraphScratchMemoryAMDX-commandBuffer-recording
  commandBuffer **must** be in the recording state
- VUID-vkCmdInitializeGraphScratchMemoryAMDX-commandBuffer-cmdpool
  The VkCommandPool that commandBuffer was allocated from **must** support graphics, or compute operations
- VUID-vkCmdInitializeGraphScratchMemoryAMDX-renderpass
  This command **must** only be called outside of a render pass instance
- VUID-vkCmdInitializeGraphScratchMemoryAMDX-videocoding
  This command **must** only be called outside of a video coding scope
- VUID-vkCmdInitializeGraphScratchMemoryAMDX-bufferlevel
  commandBuffer **must** be a primary VkCommandBuffer

## Host Synchronization

- Host access to the VkCommandPool that commandBuffer was allocated from **must** be externally synchronized

**Command Properties**

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary | Outside | Outside | Graphics<br>Compute | Action |

## 44.3. Dispatching a Graph

Initial dispatch of an execution graph is done from the host in the same way as any other command, and **can** be used in a similar way to compute dispatch commands, with indirect variants available.

To record an execution graph dispatch, call:

```
// Provided by VK_AMDX_shader_enqueue
void vkCmdDispatchGraphAMDX(
    VkCommandBuffer                             commandBuffer,
    VkDeviceAddress                             scratch,
    const VkDispatchGraphCountInfoAMDX*         pCountInfo);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `scratch` is a pointer to the scratch memory to be used.
- `pCountInfo` is a host pointer to a VkDispatchGraphCountInfoAMDX structure defining the nodes which will be initially executed.

When this command is executed, the nodes specified in `pCountInfo` are executed. Nodes executed as part of this command are not implicitly synchronized in any way against each other once they are dispatched.

For this command, all device/host pointers in substructures are treated as host pointers and read only during host execution of this command. Once this command returns, no reference to the original pointers is retained.

Execution of this command **may** modify any memory locations in the range [`scratch`,`scratch` + `size`), where `size` is the value returned in VkExecutionGraphPipelineScratchSizeAMDX::`size` by VkExecutionGraphPipelineScratchSizeAMDX for the currently bound execution graph pipeline Accesses to this memory range are performed in the `VK_PIPELINE_STAGE_2_COMPUTE_SHADER_BIT` pipeline stage with the `VK_ACCESS_2_SHADER_STORAGE_READ_BIT` and `VK_ACCESS_2_SHADER_STORAGE_WRITE_BIT` access flags.

**Valid Usage**

- VUID-vkCmdDispatchGraphAMDX-magFilter-04553

If a VkSampler created with magFilter or minFilter equal to VK_FILTER_LINEAR, reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE, and compareEnable equal to VK_FALSE is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT

- VUID-vkCmdDispatchGraphAMDX-magFilter-09598
  If a VkSampler created with magFilter or minFilter equal to VK_FILTER_LINEAR and reductionMode equal to either VK_SAMPLER_REDUCTION_MODE_MIN or VK_SAMPLER_REDUCTION_MODE_MAX is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT
- VUID-vkCmdDispatchGraphAMDX-mipmapMode-04770
  If a VkSampler created with mipmapMode equal to VK_SAMPLER_MIPMAP_MODE_LINEAR, reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE, and compareEnable equal to VK_FALSE is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT
- VUID-vkCmdDispatchGraphAMDX-mipmapMode-09599
  If a VkSampler created with mipmapMode equal to VK_SAMPLER_MIPMAP_MODE_LINEAR and reductionMode equal to either VK_SAMPLER_REDUCTION_MODE_MIN or VK_SAMPLER_REDUCTION_MODE_MAX is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT
- VUID-vkCmdDispatchGraphAMDX-unnormalizedCoordinates-09635
  If a VkSampler created with unnormalizedCoordinates equal to VK_TRUE is used to sample a VkImageView as a result of this command, then the image view's levelCount and layerCount **must** be 1
- VUID-vkCmdDispatchGraphAMDX-unnormalizedCoordinates-09636
  If a VkSampler created with unnormalizedCoordinates equal to VK_TRUE is used to sample a VkImageView as a result of this command, then the image view's viewType **must** be VK_IMAGE_VIEW_TYPE_1D or VK_IMAGE_VIEW_TYPE_2D
- VUID-vkCmdDispatchGraphAMDX-None-06479
  If a VkImageView is sampled with depth comparison, the image view's format features **must** contain VK_FORMAT_FEATURE_2_SAMPLED_IMAGE_DEPTH_COMPARISON_BIT
- VUID-vkCmdDispatchGraphAMDX-None-02691
  If a VkImageView is accessed using atomic operations as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_STORAGE_IMAGE_ATOMIC_BIT
- VUID-vkCmdDispatchGraphAMDX-None-07888
  If a VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER descriptor is accessed using atomic operations as a result of this command, then the storage texel buffer's format features **must** contain VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_ATOMIC_BIT
- VUID-vkCmdDispatchGraphAMDX-None-02692
  If a VkImageView is sampled with VK_FILTER_CUBIC_EXT as a result of this command, then

the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_CUBIC_BIT_EXT

- VUID-vkCmdDispatchGraphAMDX-None-02693
  If the VK_EXT_filter_cubic extension is not enabled and any VkImageView is sampled with VK_FILTER_CUBIC_EXT as a result of this command, it **must** not have a VkImageViewType of VK_IMAGE_VIEW_TYPE_3D, VK_IMAGE_VIEW_TYPE_CUBE, or VK_IMAGE_VIEW_TYPE_CUBE_ARRAY
- VUID-vkCmdDispatchGraphAMDX-filterCubic-02694
  Any VkImageView being sampled with VK_FILTER_CUBIC_EXT as a result of this command **must** have a VkImageViewType and format that supports cubic filtering, as specified by VkFilterCubicImageViewImageFormatPropertiesEXT::filterCubic returned by vkGetPhysicalDeviceImageFormatProperties2
- VUID-vkCmdDispatchGraphAMDX-filterCubicMinmax-02695
  Any VkImageView being sampled with VK_FILTER_CUBIC_EXT with a reduction mode of either VK_SAMPLER_REDUCTION_MODE_MIN or VK_SAMPLER_REDUCTION_MODE_MAX as a result of this command **must** have a VkImageViewType and format that supports cubic filtering together with minmax filtering, as specified by VkFilterCubicImageViewImageFormatPropertiesEXT::filterCubicMinmax returned by vkGetPhysicalDeviceImageFormatProperties2
- VUID-vkCmdDispatchGraphAMDX-cubicRangeClamp-09212
  If the cubicRangeClamp feature is not enabled, then any VkImageView being sampled with VK_FILTER_CUBIC_EXT as a result of this command **must** not have a VkSamplerReductionModeCreateInfo::reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE_RANGECLAMP_QCOM
- VUID-vkCmdDispatchGraphAMDX-reductionMode-09213
  Any VkImageView being sampled with a VkSamplerReductionModeCreateInfo::reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE_RANGECLAMP_QCOM as a result of this command **must** sample with VK_FILTER_CUBIC_EXT
- VUID-vkCmdDispatchGraphAMDX-selectableCubicWeights-09214
  If the selectableCubicWeights feature is not enabled, then any VkImageView being sampled with VK_FILTER_CUBIC_EXT as a result of this command **must** have VkSamplerCubicWeightsCreateInfoQCOM::cubicWeights equal to VK_CUBIC_FILTER_WEIGHTS_CATMULL_ROM_QCOM
- VUID-vkCmdDispatchGraphAMDX-flags-02696
  Any VkImage created with a VkImageCreateInfo::flags containing VK_IMAGE_CREATE_CORNER_SAMPLED_BIT_NV sampled as a result of this command **must** only be sampled using a VkSamplerAddressMode of VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_EDGE
- VUID-vkCmdDispatchGraphAMDX-OpTypeImage-07027
  For any VkImageView being written as a storage image where the image format field of the OpTypeImage is Unknown, the view's format features **must** contain VK_FORMAT_FEATURE_2_STORAGE_WRITE_WITHOUT_FORMAT_BIT
- VUID-vkCmdDispatchGraphAMDX-OpTypeImage-07028
  For any VkImageView being read as a storage image where the image format field of the OpTypeImage is Unknown, the view's format features **must** contain VK_FORMAT_FEATURE_2_STORAGE_READ_WITHOUT_FORMAT_BIT

- VUID-vkCmdDispatchGraphAMDX-OpTypeImage-07029
  For any VkBufferView being written as a storage texel buffer where the image format field of the `OpTypeImage` is `Unknown`, the view's buffer features **must** contain `VK_FORMAT_FEATURE_2_STORAGE_WRITE_WITHOUT_FORMAT_BIT`
- VUID-vkCmdDispatchGraphAMDX-OpTypeImage-07030
  Any VkBufferView being read as a storage texel buffer where the image format field of the `OpTypeImage` is `Unknown` then the view's buffer features **must** contain `VK_FORMAT_FEATURE_2_STORAGE_READ_WITHOUT_FORMAT_BIT`
- VUID-vkCmdDispatchGraphAMDX-None-08600
  For each set *n* that is statically used by a bound shader, a descriptor set **must** have been bound to *n* at the same pipeline bind point, with a VkPipelineLayout that is compatible for set *n*, with the VkPipelineLayout used to create the current VkPipeline or the VkDescriptorSetLayout array used to create the current VkShaderEXT , as described in Pipeline Layout Compatibility
- VUID-vkCmdDispatchGraphAMDX-None-08601
  For each push constant that is statically used by a bound shader, a push constant value **must** have been set for the same pipeline bind point, with a VkPipelineLayout that is compatible for push constants, with the VkPipelineLayout used to create the current VkPipeline or the VkDescriptorSetLayout array used to create the current VkShaderEXT , as described in Pipeline Layout Compatibility
- VUID-vkCmdDispatchGraphAMDX-None-10068
  For each array of resources that is used by a bound shader, the indices used to access members of the array **must** be less than the descriptor count for the identified binding in the descriptor sets used by this command
- VUID-vkCmdDispatchGraphAMDX-maintenance4-08602
  If the `maintenance4` feature is not enabled, then for each push constant that is statically used by a bound shader, a push constant value **must** have been set for the same pipeline bind point, with a VkPipelineLayout that is compatible for push constants, with the VkPipelineLayout used to create the current VkPipeline or the VkDescriptorSetLayout and VkPushConstantRange arrays used to create the current VkShaderEXT , as described in Pipeline Layout Compatibility
- VUID-vkCmdDispatchGraphAMDX-None-08114
  Descriptors in each bound descriptor set, specified via vkCmdBindDescriptorSets, **must** be valid as described by descriptor validity if they are statically used by the VkPipeline bound to the pipeline bind point used by this command and the bound VkPipeline was not created with `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`
- VUID-vkCmdDispatchGraphAMDX-None-08115
  If the descriptors used by the VkPipeline bound to the pipeline bind point were specified via vkCmdBindDescriptorSets, the bound VkPipeline **must** have been created without `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`
- VUID-vkCmdDispatchGraphAMDX-None-08116
  Descriptors in bound descriptor buffers, specified via vkCmdSetDescriptorBufferOffsetsEXT, **must** be valid if they are dynamically used by the VkPipeline bound to the pipeline bind point used by this command and the bound VkPipeline was created with `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`

- VUID-vkCmdDispatchGraphAMDX-None-08604
  Descriptors in bound descriptor buffers, specified via vkCmdSetDescriptorBufferOffsetsEXT, **must** be valid if they are dynamically used by any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command
- VUID-vkCmdDispatchGraphAMDX-None-08117
  If the descriptors used by the VkPipeline bound to the pipeline bind point were specified via vkCmdSetDescriptorBufferOffsetsEXT, the bound VkPipeline **must** have been created with `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`
- VUID-vkCmdDispatchGraphAMDX-None-08119
  If a descriptor is dynamically used with a VkPipeline created with `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`, the descriptor memory **must** be resident
- VUID-vkCmdDispatchGraphAMDX-None-08605
  If a descriptor is dynamically used with a VkShaderEXT created with a `VkDescriptorSetLayout` that was created with `VK_DESCRIPTOR_SET_LAYOUT_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`, the descriptor memory **must** be resident
- VUID-vkCmdDispatchGraphAMDX-None-08606
  If the `shaderObject` feature is not enabled, a valid pipeline **must** be bound to the pipeline bind point used by this command
- VUID-vkCmdDispatchGraphAMDX-None-08608
  If a pipeline is bound to the pipeline bind point used by this command, there **must** not have been any calls to dynamic state setting commands for any state specified statically in the VkPipeline object bound to the pipeline bind point used by this command, since that pipeline was bound
- VUID-vkCmdDispatchGraphAMDX-None-08609
  If the VkPipeline object bound to the pipeline bind point used by this command or any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a VkSampler object that uses unnormalized coordinates, that sampler **must** not be used to sample from any VkImage with a VkImageView of the type `VK_IMAGE_VIEW_TYPE_3D`, `VK_IMAGE_VIEW_TYPE_CUBE`, `VK_IMAGE_VIEW_TYPE_1D_ARRAY`, `VK_IMAGE_VIEW_TYPE_2D_ARRAY` or `VK_IMAGE_VIEW_TYPE_CUBE_ARRAY`, in any shader stage
- VUID-vkCmdDispatchGraphAMDX-None-08610
  If the VkPipeline object bound to the pipeline bind point used by this command or any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a VkSampler object that uses unnormalized coordinates, that sampler **must** not be used with any of the SPIR-V `OpImageSample*` or `OpImageSparseSample*` instructions with `ImplicitLod`, `Dref` or `Proj` in their name, in any shader stage
- VUID-vkCmdDispatchGraphAMDX-None-08611
  If the VkPipeline object bound to the pipeline bind point used by this command or any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a VkSampler object that uses unnormalized coordinates, that sampler **must** not be used with any of the SPIR-V `OpImageSample*` or `OpImageSparseSample*` instructions that includes a LOD bias or any offset values, in any shader stage

- VUID-vkCmdDispatchGraphAMDX-None-08607
  If the `shaderObject` is enabled, either a valid pipeline **must** be bound to the pipeline bind point used by this command, or a valid combination of valid and VK_NULL_HANDLE shader objects **must** be bound to every supported shader stage corresponding to the pipeline bind point used by this command
- VUID-vkCmdDispatchGraphAMDX-uniformBuffers-06935
  If any stage of the VkPipeline object bound to the pipeline bind point used by this command accesses a uniform buffer, and that stage was created without enabling either `VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_EXT` or `VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_2_EXT` for `uniformBuffers`, and the `robustBufferAccess` feature is not enabled, that stage **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point
- VUID-vkCmdDispatchGraphAMDX-None-08612
  If the `robustBufferAccess` feature is not enabled, and any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a uniform buffer, it **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point
- VUID-vkCmdDispatchGraphAMDX-storageBuffers-06936
  If any stage of the VkPipeline object bound to the pipeline bind point used by this command accesses a storage buffer, and that stage was created without enabling either `VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_EXT` or `VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_2_EXT` for `storageBuffers`, and the `robustBufferAccess` feature is not enabled, that stage **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point
- VUID-vkCmdDispatchGraphAMDX-None-08613
  If the `robustBufferAccess` feature is not enabled, and any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a storage buffer, it **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point
- VUID-vkCmdDispatchGraphAMDX-commandBuffer-02707
  If `commandBuffer` is an unprotected command buffer and `protectedNoFault` is not supported, any resource accessed by bound shaders **must** not be a protected resource
- VUID-vkCmdDispatchGraphAMDX-None-06550
  If a bound shader accesses a VkSampler or VkImageView object that enables sampler $Y'C_BC_R$ conversion, that object **must** only be used with `OpImageSample*` or `OpImageSparseSample*` instructions
- VUID-vkCmdDispatchGraphAMDX-ConstOffset-06551
  If a bound shader accesses a VkSampler or VkImageView object that enables sampler $Y'C_BC_R$ conversion, that object **must** not use the `ConstOffset` and `Offset` operands
- VUID-vkCmdDispatchGraphAMDX-viewType-07752
  If a VkImageView is accessed as a result of this command, then the image view's `viewType` **must** match the `Dim` operand of the `OpTypeImage` as described in Compatibility Between SPIR-V Image Dimensions and Vulkan ImageView Types

- VUID-vkCmdDispatchGraphAMDX-format-07753
  If a VkImageView is accessed as a result of this command, then the numeric type of the image view's `format` and the `Sampled Type` operand of the `OpTypeImage` **must** match
- VUID-vkCmdDispatchGraphAMDX-OpImageWrite-08795
  If a VkImageView created with a format other than `VK_FORMAT_A8_UNORM_KHR` is accessed using `OpImageWrite` as a result of this command, then the `Type` of the `Texel` operand of that instruction **must** have at least as many components as the image view's format
- VUID-vkCmdDispatchGraphAMDX-OpImageWrite-08796
  If a VkImageView created with the format `VK_FORMAT_A8_UNORM_KHR` is accessed using `OpImageWrite` as a result of this command, then the `Type` of the `Texel` operand of that instruction **must** have four components
- VUID-vkCmdDispatchGraphAMDX-OpImageWrite-04469
  If a VkBufferView is accessed using `OpImageWrite` as a result of this command, then the `Type` of the `Texel` operand of that instruction **must** have at least as many components as the buffer view's format
- VUID-vkCmdDispatchGraphAMDX-SampledType-04470
  If a VkImageView with a VkFormat that has a 64-bit component width is accessed as a result of this command, the `SampledType` of the `OpTypeImage` operand of that instruction **must** have a `Width` of 64
- VUID-vkCmdDispatchGraphAMDX-SampledType-04471
  If a VkImageView with a VkFormat that has a component width less than 64-bit is accessed as a result of this command, the `SampledType` of the `OpTypeImage` operand of that instruction **must** have a `Width` of 32
- VUID-vkCmdDispatchGraphAMDX-SampledType-04472
  If a VkBufferView with a VkFormat that has a 64-bit component width is accessed as a result of this command, the `SampledType` of the `OpTypeImage` operand of that instruction **must** have a `Width` of 64
- VUID-vkCmdDispatchGraphAMDX-SampledType-04473
  If a VkBufferView with a VkFormat that has a component width less than 64-bit is accessed as a result of this command, the `SampledType` of the `OpTypeImage` operand of that instruction **must** have a `Width` of 32
- VUID-vkCmdDispatchGraphAMDX-sparseImageInt64Atomics-04474
  If the `sparseImageInt64Atomics` feature is not enabled, VkImage objects created with the `VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT` flag **must** not be accessed by atomic instructions through an `OpTypeImage` with a `SampledType` with a `Width` of 64 by this command
- VUID-vkCmdDispatchGraphAMDX-sparseImageInt64Atomics-04475
  If the `sparseImageInt64Atomics` feature is not enabled, VkBuffer objects created with the `VK_BUFFER_CREATE_SPARSE_RESIDENCY_BIT` flag **must** not be accessed by atomic instructions through an `OpTypeImage` with a `SampledType` with a `Width` of 64 by this command
- VUID-vkCmdDispatchGraphAMDX-OpImageWeightedSampleQCOM-06971
  If `OpImageWeightedSampleQCOM` is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_2_WEIGHT_SAMPLED_IMAGE_BIT_QCOM`

- VUID-vkCmdDispatchGraphAMDX-OpImageWeightedSampleQCOM-06972
  If `OpImageWeightedSampleQCOM` uses a VkImageView as a sample weight image as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_2_WEIGHT_IMAGE_BIT_QCOM`
- VUID-vkCmdDispatchGraphAMDX-OpImageBoxFilterQCOM-06973
  If `OpImageBoxFilterQCOM` is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_2_BOX_FILTER_SAMPLED_BIT_QCOM`
- VUID-vkCmdDispatchGraphAMDX-OpImageBlockMatchSSDQCOM-06974
  If `OpImageBlockMatchSSDQCOM` is used to read from an VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM`
- VUID-vkCmdDispatchGraphAMDX-OpImageBlockMatchSADQCOM-06975
  If `OpImageBlockMatchSADQCOM` is used to read from an VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM`
- VUID-vkCmdDispatchGraphAMDX-OpImageBlockMatchSADQCOM-06976
  If `OpImageBlockMatchSADQCOM` or OpImageBlockMatchSSDQCOM is used to read from a reference image as result of this command, then the specified reference coordinates **must** not fail integer texel coordinate validation
- VUID-vkCmdDispatchGraphAMDX-OpImageWeightedSampleQCOM-06977
  If `OpImageWeightedSampleQCOM`, `OpImageBoxFilterQCOM`, `OpImageBlockMatchWindowSSDQCOM`, `OpImageBlockMatchWindowSADQCOM`, `OpImageBlockMatchGatherSSDQCOM`, `OpImageBlockMatchGatherSADQCOM`, `OpImageBlockMatchSSDQCOM`, or `OpImageBlockMatchSADQCOM` uses a VkSampler as a result of this command, then the sampler **must** have been created with `VK_SAMPLER_CREATE_IMAGE_PROCESSING_BIT_QCOM`
- VUID-vkCmdDispatchGraphAMDX-OpImageWeightedSampleQCOM-06978
  If any command other than `OpImageWeightedSampleQCOM`, `OpImageBoxFilterQCOM`, `OpImageBlockMatchWindowSSDQCOM`, `OpImageBlockMatchWindowSADQCOM`, `OpImageBlockMatchGatherSSDQCOM`, `OpImageBlockMatchGatherSADQCOM`, `OpImageBlockMatchSSDQCOM`, or `OpImageBlockMatchSADQCOM` uses a VkSampler as a result of this command, then the sampler **must** not have been created with `VK_SAMPLER_CREATE_IMAGE_PROCESSING_BIT_QCOM`
- VUID-vkCmdDispatchGraphAMDX-OpImageBlockMatchWindow-09215
  If a `OpImageBlockMatchWindow*QCOM` or `OpImageBlockMatchGather*QCOM` instruction is used to read from an VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM`
- VUID-vkCmdDispatchGraphAMDX-OpImageBlockMatchWindow-09216
  If a `OpImageBlockMatchWindow*QCOM` or `OpImageBlockMatchGather*QCOM` instruction is used to read from an VkImageView as a result of this command, then the image view's format **must** be a single-component format
- VUID-vkCmdDispatchGraphAMDX-OpImageBlockMatchWindow-09217
  If a `OpImageBlockMatchWindow*QCOM` or `OpImageBlockMatchGather*QCOM` read from a reference image as result of this command, then the specified reference coordinates **must** not fail

integer texel coordinate validation

- VUID-vkCmdDispatchGraphAMDX-None-07288
  Any shader invocation executed by this command **must** terminate
- VUID-vkCmdDispatchGraphAMDX-None-09600
  If a descriptor with type equal to any of `VK_DESCRIPTOR_TYPE_SAMPLE_WEIGHT_IMAGE_QCOM`, `VK_DESCRIPTOR_TYPE_BLOCK_MATCH_IMAGE_QCOM`, `VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE`, `VK_DESCRIPTOR_TYPE_STORAGE_IMAGE`, or `VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT` is accessed as a result of this command, the image subresource identified by that descriptor **must** be in the image layout identified when the descriptor was written
- VUID-vkCmdDispatchGraphAMDX-commandBuffer-09181
  `commandBuffer` **must** not be a protected command buffer
- VUID-vkCmdDispatchGraphAMDX-commandBuffer-09182
  `commandBuffer` **must** be a primary command buffer
- VUID-vkCmdDispatchGraphAMDX-scratch-09183
  `scratch` **must** be the device address of an allocated memory range at least as large as the value of VkExecutionGraphPipelineScratchSizeAMDX::`size` returned by VkExecutionGraphPipelineScratchSizeAMDX for the currently bound execution graph pipeline
- VUID-vkCmdDispatchGraphAMDX-scratch-09184
  `scratch` **must** be a device address within a VkBuffer created with the `VK_BUFFER_USAGE_EXECUTION_GRAPH_SCRATCH_BIT_AMDX` or `VK_BUFFER_USAGE_2_EXECUTION_GRAPH_SCRATCH_BIT_AMDX` flag
- VUID-vkCmdDispatchGraphAMDX-scratch-09185
  Device memory in the range [`scratch`,`scratch` VkExecutionGraphPipelineScratchSizeAMDX::`size`) **must** have been initialized with vkCmdInitializeGraphScratchMemoryAMDX using the currently bound execution graph pipeline, and not modified after that by anything other than another execution graph dispatch command
- VUID-vkCmdDispatchGraphAMDX-maxComputeWorkGroupCount-09186
  Execution of this command **must** not cause a node to be dispatched with a larger number of workgroups than that specified by either a `MaxNumWorkgroupsAMDX` decoration in the dispatched node or `maxComputeWorkGroupCount`
- VUID-vkCmdDispatchGraphAMDX-maxExecutionGraphShaderPayloadCount-09187
  Execution of this command **must** not cause any shader to initialize more than `maxExecutionGraphShaderPayloadCount` output payloads
- VUID-vkCmdDispatchGraphAMDX-NodeMaxPayloadsAMDX-09188
  Execution of this command **must** not cause any shader that declares `NodeMaxPayloadsAMDX` to initialize more output payloads than specified by the max number of payloads for that decoration. This requirement applies to each `NodeMaxPayloadsAMDX` decoration separately
- VUID-vkCmdDispatchGraphAMDX-pCountInfo-09145
  `pCountInfo->infos` **must** be a host pointer to a memory allocation at least as large as the product of `count` and `stride`
- VUID-vkCmdDispatchGraphAMDX-infos-09146

Host memory locations at indexes in the range [infos, infos + (count*stride)), at a granularity of stride **must** contain valid VkDispatchGraphInfoAMDX structures in the first 24 bytes

- VUID-vkCmdDispatchGraphAMDX-pCountInfo-09147
  For each VkDispatchGraphInfoAMDX structure in pCountInfo->infos, payloads **must** be a host pointer to a memory allocation at least as large as the product of payloadCount and payloadStride
- VUID-vkCmdDispatchGraphAMDX-pCountInfo-09148
  For each VkDispatchGraphInfoAMDX structure in pCountInfo->infos, nodeIndex **must** be a valid node index in the currently bound execution graph pipeline, as returned by vkGetExecutionGraphPipelineNodeIndexAMDX
- VUID-vkCmdDispatchGraphAMDX-pCountInfo-09149
  For each VkDispatchGraphInfoAMDX structure in pCountInfo->infos, host memory locations at indexes in the range [payloads, payloads + (payloadCount * payloadStride)), at a granularity of payloadStride **must** contain a payload matching the size of the input payload expected by the node in nodeIndex in the first bytes

## Valid Usage (Implicit)

- VUID-vkCmdDispatchGraphAMDX-commandBuffer-parameter
  commandBuffer **must** be a valid VkCommandBuffer handle
- VUID-vkCmdDispatchGraphAMDX-pCountInfo-parameter
  pCountInfo **must** be a valid pointer to a valid VkDispatchGraphCountInfoAMDX structure
- VUID-vkCmdDispatchGraphAMDX-commandBuffer-recording
  commandBuffer **must** be in the recording state
- VUID-vkCmdDispatchGraphAMDX-commandBuffer-cmdpool
  The VkCommandPool that commandBuffer was allocated from **must** support graphics, or compute operations
- VUID-vkCmdDispatchGraphAMDX-renderpass
  This command **must** only be called outside of a render pass instance
- VUID-vkCmdDispatchGraphAMDX-videocoding
  This command **must** only be called outside of a video coding scope
- VUID-vkCmdDispatchGraphAMDX-bufferlevel
  commandBuffer **must** be a primary VkCommandBuffer

## Host Synchronization

- Host access to the VkCommandPool that commandBuffer was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary | Outside | Outside | Graphics<br>Compute | Action |

To record an execution graph dispatch with node and payload parameters read on device, call:

```
// Provided by VK_AMDX_shader_enqueue
void vkCmdDispatchGraphIndirectAMDX(
    VkCommandBuffer                             commandBuffer,
    VkDeviceAddress                             scratch,
    const VkDispatchGraphCountInfoAMDX*         pCountInfo);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `scratch` is a pointer to the scratch memory to be used.
- `pCountInfo` is a host pointer to a VkDispatchGraphCountInfoAMDX structure defining the nodes which will be initially executed.

When this command is executed, the nodes specified in `pCountInfo` are executed. Nodes executed as part of this command are not implicitly synchronized in any way against each other once they are dispatched.

For this command, all device/host pointers in substructures are treated as device pointers and read during device execution of this command. The allocation and contents of these pointers only needs to be valid during device execution. All of these addresses will be read in the `VK_PIPELINE_STAGE_2_COMPUTE_SHADER_BIT` pipeline stage with the `VK_ACCESS_2_SHADER_STORAGE_READ_BIT` access flag.

Execution of this command **may** modify any memory locations in the range [`scratch`,`scratch` + `size`), where `size` is the value returned in VkExecutionGraphPipelineScratchSizeAMDX::`size` by VkExecutionGraphPipelineScratchSizeAMDX for the currently bound execution graph pipeline. Accesses to this memory range are performed in the `VK_PIPELINE_STAGE_2_COMPUTE_SHADER_BIT` pipeline stage with the `VK_ACCESS_2_SHADER_STORAGE_READ_BIT` and `VK_ACCESS_2_SHADER_STORAGE_WRITE_BIT` access flags.

## Valid Usage

- VUID-vkCmdDispatchGraphIndirectAMDX-magFilter-04553
  If a VkSampler created with `magFilter` or `minFilter` equal to `VK_FILTER_LINEAR`, `reductionMode` equal to `VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE`, and `compareEnable` equal to `VK_FALSE` is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT`

- VUID-vkCmdDispatchGraphIndirectAMDX-magFilter-09598
  If a VkSampler created with magFilter or minFilter equal to VK_FILTER_LINEAR and reductionMode equal to either VK_SAMPLER_REDUCTION_MODE_MIN or VK_SAMPLER_REDUCTION_MODE_MAX is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT
- VUID-vkCmdDispatchGraphIndirectAMDX-mipmapMode-04770
  If a VkSampler created with mipmapMode equal to VK_SAMPLER_MIPMAP_MODE_LINEAR, reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE, and compareEnable equal to VK_FALSE is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT
- VUID-vkCmdDispatchGraphIndirectAMDX-mipmapMode-09599
  If a VkSampler created with mipmapMode equal to VK_SAMPLER_MIPMAP_MODE_LINEAR and reductionMode equal to either VK_SAMPLER_REDUCTION_MODE_MIN or VK_SAMPLER_REDUCTION_MODE_MAX is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT
- VUID-vkCmdDispatchGraphIndirectAMDX-unnormalizedCoordinates-09635
  If a VkSampler created with unnormalizedCoordinates equal to VK_TRUE is used to sample a VkImageView as a result of this command, then the image view's levelCount and layerCount **must** be 1
- VUID-vkCmdDispatchGraphIndirectAMDX-unnormalizedCoordinates-09636
  If a VkSampler created with unnormalizedCoordinates equal to VK_TRUE is used to sample a VkImageView as a result of this command, then the image view's viewType **must** be VK_IMAGE_VIEW_TYPE_1D or VK_IMAGE_VIEW_TYPE_2D
- VUID-vkCmdDispatchGraphIndirectAMDX-None-06479
  If a VkImageView is sampled with depth comparison, the image view's format features **must** contain VK_FORMAT_FEATURE_2_SAMPLED_IMAGE_DEPTH_COMPARISON_BIT
- VUID-vkCmdDispatchGraphIndirectAMDX-None-02691
  If a VkImageView is accessed using atomic operations as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_STORAGE_IMAGE_ATOMIC_BIT
- VUID-vkCmdDispatchGraphIndirectAMDX-None-07888
  If a VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER descriptor is accessed using atomic operations as a result of this command, then the storage texel buffer's format features **must** contain VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_ATOMIC_BIT
- VUID-vkCmdDispatchGraphIndirectAMDX-None-02692
  If a VkImageView is sampled with VK_FILTER_CUBIC_EXT as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_CUBIC_BIT_EXT
- VUID-vkCmdDispatchGraphIndirectAMDX-None-02693
  If the VK_EXT_filter_cubic extension is not enabled and any VkImageView is sampled with VK_FILTER_CUBIC_EXT as a result of this command, it **must** not have a VkImageViewType of

VK_IMAGE_VIEW_TYPE_3D, VK_IMAGE_VIEW_TYPE_CUBE, or VK_IMAGE_VIEW_TYPE_CUBE_ARRAY

- VUID-vkCmdDispatchGraphIndirectAMDX-filterCubic-02694
  Any VkImageView being sampled with VK_FILTER_CUBIC_EXT as a result of this command **must** have a VkImageViewType and format that supports cubic filtering, as specified by VkFilterCubicImageViewImageFormatPropertiesEXT::filterCubic returned by vkGetPhysicalDeviceImageFormatProperties2
- VUID-vkCmdDispatchGraphIndirectAMDX-filterCubicMinmax-02695
  Any VkImageView being sampled with VK_FILTER_CUBIC_EXT with a reduction mode of either VK_SAMPLER_REDUCTION_MODE_MIN or VK_SAMPLER_REDUCTION_MODE_MAX as a result of this command **must** have a VkImageViewType and format that supports cubic filtering together with minmax filtering, as specified by VkFilterCubicImageViewImageFormatPropertiesEXT::filterCubicMinmax returned by vkGetPhysicalDeviceImageFormatProperties2
- VUID-vkCmdDispatchGraphIndirectAMDX-cubicRangeClamp-09212
  If the cubicRangeClamp feature is not enabled, then any VkImageView being sampled with VK_FILTER_CUBIC_EXT as a result of this command **must** not have a VkSamplerReductionModeCreateInfo::reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE_RANGECLAMP_QCOM
- VUID-vkCmdDispatchGraphIndirectAMDX-reductionMode-09213
  Any VkImageView being sampled with a VkSamplerReductionModeCreateInfo::reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE_RANGECLAMP_QCOM as a result of this command **must** sample with VK_FILTER_CUBIC_EXT
- VUID-vkCmdDispatchGraphIndirectAMDX-selectableCubicWeights-09214
  If the selectableCubicWeights feature is not enabled, then any VkImageView being sampled with VK_FILTER_CUBIC_EXT as a result of this command **must** have VkSamplerCubicWeightsCreateInfoQCOM::cubicWeights equal to VK_CUBIC_FILTER_WEIGHTS_CATMULL_ROM_QCOM
- VUID-vkCmdDispatchGraphIndirectAMDX-flags-02696
  Any VkImage created with a VkImageCreateInfo::flags containing VK_IMAGE_CREATE_CORNER_SAMPLED_BIT_NV sampled as a result of this command **must** only be sampled using a VkSamplerAddressMode of VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_EDGE
- VUID-vkCmdDispatchGraphIndirectAMDX-OpTypeImage-07027
  For any VkImageView being written as a storage image where the image format field of the OpTypeImage is Unknown, the view's format features **must** contain VK_FORMAT_FEATURE_2_STORAGE_WRITE_WITHOUT_FORMAT_BIT
- VUID-vkCmdDispatchGraphIndirectAMDX-OpTypeImage-07028
  For any VkImageView being read as a storage image where the image format field of the OpTypeImage is Unknown, the view's format features **must** contain VK_FORMAT_FEATURE_2_STORAGE_READ_WITHOUT_FORMAT_BIT
- VUID-vkCmdDispatchGraphIndirectAMDX-OpTypeImage-07029
  For any VkBufferView being written as a storage texel buffer where the image format field of the OpTypeImage is Unknown, the view's buffer features **must** contain VK_FORMAT_FEATURE_2_STORAGE_WRITE_WITHOUT_FORMAT_BIT
- VUID-vkCmdDispatchGraphIndirectAMDX-OpTypeImage-07030

Any VkBufferView being read as a storage texel buffer where the image format field of the `OpTypeImage` is `Unknown` then the view's buffer features **must** contain `VK_FORMAT_FEATURE_2_STORAGE_READ_WITHOUT_FORMAT_BIT`

- VUID-vkCmdDispatchGraphIndirectAMDX-None-08600
  For each set *n* that is statically used by a bound shader, a descriptor set **must** have been bound to *n* at the same pipeline bind point, with a VkPipelineLayout that is compatible for set *n*, with the VkPipelineLayout used to create the current VkPipeline or the VkDescriptorSetLayout array used to create the current VkShaderEXT , as described in Pipeline Layout Compatibility

- VUID-vkCmdDispatchGraphIndirectAMDX-None-08601
  For each push constant that is statically used by a bound shader, a push constant value **must** have been set for the same pipeline bind point, with a VkPipelineLayout that is compatible for push constants, with the VkPipelineLayout used to create the current VkPipeline or the VkDescriptorSetLayout array used to create the current VkShaderEXT , as described in Pipeline Layout Compatibility

- VUID-vkCmdDispatchGraphIndirectAMDX-None-10068
  For each array of resources that is used by a bound shader, the indices used to access members of the array **must** be less than the descriptor count for the identified binding in the descriptor sets used by this command

- VUID-vkCmdDispatchGraphIndirectAMDX-maintenance4-08602
  If the `maintenance4` feature is not enabled, then for each push constant that is statically used by a bound shader, a push constant value **must** have been set for the same pipeline bind point, with a VkPipelineLayout that is compatible for push constants, with the VkPipelineLayout used to create the current VkPipeline or the VkDescriptorSetLayout and VkPushConstantRange arrays used to create the current VkShaderEXT , as described in Pipeline Layout Compatibility

- VUID-vkCmdDispatchGraphIndirectAMDX-None-08114
  Descriptors in each bound descriptor set, specified via vkCmdBindDescriptorSets, **must** be valid as described by descriptor validity if they are statically used by the VkPipeline bound to the pipeline bind point used by this command and the bound VkPipeline was not created with `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`

- VUID-vkCmdDispatchGraphIndirectAMDX-None-08115
  If the descriptors used by the VkPipeline bound to the pipeline bind point were specified via vkCmdBindDescriptorSets, the bound VkPipeline **must** have been created without `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`

- VUID-vkCmdDispatchGraphIndirectAMDX-None-08116
  Descriptors in bound descriptor buffers, specified via vkCmdSetDescriptorBufferOffsetsEXT, **must** be valid if they are dynamically used by the VkPipeline bound to the pipeline bind point used by this command and the bound VkPipeline was created with `VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT`

- VUID-vkCmdDispatchGraphIndirectAMDX-None-08604
  Descriptors in bound descriptor buffers, specified via vkCmdSetDescriptorBufferOffsetsEXT, **must** be valid if they are dynamically used by any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command

- VUID-vkCmdDispatchGraphIndirectAMDX-None-08117
  If the descriptors used by the VkPipeline bound to the pipeline bind point were specified via vkCmdSetDescriptorBufferOffsetsEXT, the bound VkPipeline **must** have been created with VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT
- VUID-vkCmdDispatchGraphIndirectAMDX-None-08119
  If a descriptor is dynamically used with a VkPipeline created with VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT, the descriptor memory **must** be resident
- VUID-vkCmdDispatchGraphIndirectAMDX-None-08605
  If a descriptor is dynamically used with a VkShaderEXT created with a VkDescriptorSetLayout that was created with VK_DESCRIPTOR_SET_LAYOUT_CREATE_DESCRIPTOR_BUFFER_BIT_EXT, the descriptor memory **must** be resident
- VUID-vkCmdDispatchGraphIndirectAMDX-None-08606
  If the shaderObject feature is not enabled, a valid pipeline **must** be bound to the pipeline bind point used by this command
- VUID-vkCmdDispatchGraphIndirectAMDX-None-08608
  If a pipeline is bound to the pipeline bind point used by this command, there **must** not have been any calls to dynamic state setting commands for any state specified statically in the VkPipeline object bound to the pipeline bind point used by this command, since that pipeline was bound
- VUID-vkCmdDispatchGraphIndirectAMDX-None-08609
  If the VkPipeline object bound to the pipeline bind point used by this command or any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a VkSampler object that uses unnormalized coordinates, that sampler **must** not be used to sample from any VkImage with a VkImageView of the type VK_IMAGE_VIEW_TYPE_3D, VK_IMAGE_VIEW_TYPE_CUBE, VK_IMAGE_VIEW_TYPE_1D_ARRAY, VK_IMAGE_VIEW_TYPE_2D_ARRAY or VK_IMAGE_VIEW_TYPE_CUBE_ARRAY, in any shader stage
- VUID-vkCmdDispatchGraphIndirectAMDX-None-08610
  If the VkPipeline object bound to the pipeline bind point used by this command or any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a VkSampler object that uses unnormalized coordinates, that sampler **must** not be used with any of the SPIR-V OpImageSample* or OpImageSparseSample* instructions with ImplicitLod, Dref or Proj in their name, in any shader stage
- VUID-vkCmdDispatchGraphIndirectAMDX-None-08611
  If the VkPipeline object bound to the pipeline bind point used by this command or any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a VkSampler object that uses unnormalized coordinates, that sampler **must** not be used with any of the SPIR-V OpImageSample* or OpImageSparseSample* instructions that includes a LOD bias or any offset values, in any shader stage
- VUID-vkCmdDispatchGraphIndirectAMDX-None-08607
  If the shaderObject is enabled, either a valid pipeline **must** be bound to the pipeline bind point used by this command, or a valid combination of valid and VK_NULL_HANDLE shader objects **must** be bound to every supported shader stage corresponding to the pipeline bind point used by this command

- VUID-vkCmdDispatchGraphIndirectAMDX-uniformBuffers-06935
  If any stage of the VkPipeline object bound to the pipeline bind point used by this command accesses a uniform buffer, and that stage was created without enabling either `VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_EXT` or `VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_2_EXT` for `uniformBuffers`, and the `robustBufferAccess` feature is not enabled, that stage **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point
- VUID-vkCmdDispatchGraphIndirectAMDX-None-08612
  If the `robustBufferAccess` feature is not enabled, and any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a uniform buffer, it **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point
- VUID-vkCmdDispatchGraphIndirectAMDX-storageBuffers-06936
  If any stage of the VkPipeline object bound to the pipeline bind point used by this command accesses a storage buffer, and that stage was created without enabling either `VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_EXT` or `VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_2_EXT` for `storageBuffers`, and the `robustBufferAccess` feature is not enabled, that stage **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point
- VUID-vkCmdDispatchGraphIndirectAMDX-None-08613
  If the `robustBufferAccess` feature is not enabled, and any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a storage buffer, it **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point
- VUID-vkCmdDispatchGraphIndirectAMDX-commandBuffer-02707
  If `commandBuffer` is an unprotected command buffer and `protectedNoFault` is not supported, any resource accessed by bound shaders **must** not be a protected resource
- VUID-vkCmdDispatchGraphIndirectAMDX-None-06550
  If a bound shader accesses a VkSampler or VkImageView object that enables sampler $Y'C_BC_R$ conversion, that object **must** only be used with `OpImageSample*` or `OpImageSparseSample*` instructions
- VUID-vkCmdDispatchGraphIndirectAMDX-ConstOffset-06551
  If a bound shader accesses a VkSampler or VkImageView object that enables sampler $Y'C_BC_R$ conversion, that object **must** not use the `ConstOffset` and `Offset` operands
- VUID-vkCmdDispatchGraphIndirectAMDX-viewType-07752
  If a VkImageView is accessed as a result of this command, then the image view's `viewType` **must** match the `Dim` operand of the `OpTypeImage` as described in Compatibility Between SPIR-V Image Dimensions and Vulkan ImageView Types
- VUID-vkCmdDispatchGraphIndirectAMDX-format-07753
  If a VkImageView is accessed as a result of this command, then the numeric type of the image view's `format` and the `Sampled Type` operand of the `OpTypeImage` **must** match
- VUID-vkCmdDispatchGraphIndirectAMDX-OpImageWrite-08795

If a VkImageView created with a format other than VK_FORMAT_A8_UNORM_KHR is accessed using OpImageWrite as a result of this command, then the Type of the Texel operand of that instruction **must** have at least as many components as the image view's format

- VUID-vkCmdDispatchGraphIndirectAMDX-OpImageWrite-08796
  If a VkImageView created with the format VK_FORMAT_A8_UNORM_KHR is accessed using OpImageWrite as a result of this command, then the Type of the Texel operand of that instruction **must** have four components
- VUID-vkCmdDispatchGraphIndirectAMDX-OpImageWrite-04469
  If a VkBufferView is accessed using OpImageWrite as a result of this command, then the Type of the Texel operand of that instruction **must** have at least as many components as the buffer view's format
- VUID-vkCmdDispatchGraphIndirectAMDX-SampledType-04470
  If a VkImageView with a VkFormat that has a 64-bit component width is accessed as a result of this command, the SampledType of the OpTypeImage operand of that instruction **must** have a Width of 64
- VUID-vkCmdDispatchGraphIndirectAMDX-SampledType-04471
  If a VkImageView with a VkFormat that has a component width less than 64-bit is accessed as a result of this command, the SampledType of the OpTypeImage operand of that instruction **must** have a Width of 32
- VUID-vkCmdDispatchGraphIndirectAMDX-SampledType-04472
  If a VkBufferView with a VkFormat that has a 64-bit component width is accessed as a result of this command, the SampledType of the OpTypeImage operand of that instruction **must** have a Width of 64
- VUID-vkCmdDispatchGraphIndirectAMDX-SampledType-04473
  If a VkBufferView with a VkFormat that has a component width less than 64-bit is accessed as a result of this command, the SampledType of the OpTypeImage operand of that instruction **must** have a Width of 32
- VUID-vkCmdDispatchGraphIndirectAMDX-sparseImageInt64Atomics-04474
  If the sparseImageInt64Atomics feature is not enabled, VkImage objects created with the VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT flag **must** not be accessed by atomic instructions through an OpTypeImage with a SampledType with a Width of 64 by this command
- VUID-vkCmdDispatchGraphIndirectAMDX-sparseImageInt64Atomics-04475
  If the sparseImageInt64Atomics feature is not enabled, VkBuffer objects created with the VK_BUFFER_CREATE_SPARSE_RESIDENCY_BIT flag **must** not be accessed by atomic instructions through an OpTypeImage with a SampledType with a Width of 64 by this command
- VUID-vkCmdDispatchGraphIndirectAMDX-OpImageWeightedSampleQCOM-06971
  If OpImageWeightedSampleQCOM is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_WEIGHT_SAMPLED_IMAGE_BIT_QCOM
- VUID-vkCmdDispatchGraphIndirectAMDX-OpImageWeightedSampleQCOM-06972
  If OpImageWeightedSampleQCOM uses a VkImageView as a sample weight image as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_WEIGHT_IMAGE_BIT_QCOM

- VUID-vkCmdDispatchGraphIndirectAMDX-OpImageBoxFilterQCOM-06973
  If `OpImageBoxFilterQCOM` is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_2_BOX_FILTER_SAMPLED_BIT_QCOM`
- VUID-vkCmdDispatchGraphIndirectAMDX-OpImageBlockMatchSSDQCOM-06974
  If `OpImageBlockMatchSSDQCOM` is used to read from an VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM`
- VUID-vkCmdDispatchGraphIndirectAMDX-OpImageBlockMatchSADQCOM-06975
  If `OpImageBlockMatchSADQCOM` is used to read from an VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM`
- VUID-vkCmdDispatchGraphIndirectAMDX-OpImageBlockMatchSADQCOM-06976
  If `OpImageBlockMatchSADQCOM` or OpImageBlockMatchSSDQCOM is used to read from a reference image as result of this command, then the specified reference coordinates **must** not fail integer texel coordinate validation
- VUID-vkCmdDispatchGraphIndirectAMDX-OpImageWeightedSampleQCOM-06977
  If `OpImageWeightedSampleQCOM`, `OpImageBoxFilterQCOM`, `OpImageBlockMatchWindowSSDQCOM`, `OpImageBlockMatchWindowSADQCOM`, `OpImageBlockMatchGatherSSDQCOM`, `OpImageBlockMatchGatherSADQCOM`, `OpImageBlockMatchSSDQCOM`, or `OpImageBlockMatchSADQCOM` uses a VkSampler as a result of this command, then the sampler **must** have been created with `VK_SAMPLER_CREATE_IMAGE_PROCESSING_BIT_QCOM`
- VUID-vkCmdDispatchGraphIndirectAMDX-OpImageWeightedSampleQCOM-06978
  If any command other than `OpImageWeightedSampleQCOM`, `OpImageBoxFilterQCOM`, `OpImageBlockMatchWindowSSDQCOM`, `OpImageBlockMatchWindowSADQCOM`, `OpImageBlockMatchGatherSSDQCOM`, `OpImageBlockMatchGatherSADQCOM`, `OpImageBlockMatchSSDQCOM`, or `OpImageBlockMatchSADQCOM` uses a VkSampler as a result of this command, then the sampler **must** not have been created with `VK_SAMPLER_CREATE_IMAGE_PROCESSING_BIT_QCOM`
- VUID-vkCmdDispatchGraphIndirectAMDX-OpImageBlockMatchWindow-09215
  If a `OpImageBlockMatchWindow*QCOM` or `OpImageBlockMatchGather*QCOM` instruction is used to read from an VkImageView as a result of this command, then the image view's format features **must** contain `VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM`
- VUID-vkCmdDispatchGraphIndirectAMDX-OpImageBlockMatchWindow-09216
  If a `OpImageBlockMatchWindow*QCOM` or `OpImageBlockMatchGather*QCOM` instruction is used to read from an VkImageView as a result of this command, then the image view's format **must** be a single-component format
- VUID-vkCmdDispatchGraphIndirectAMDX-OpImageBlockMatchWindow-09217
  If a `OpImageBlockMatchWindow*QCOM` or `OpImageBlockMatchGather*QCOM` read from a reference image as result of this command, then the specified reference coordinates **must** not fail integer texel coordinate validation
- VUID-vkCmdDispatchGraphIndirectAMDX-None-07288
  Any shader invocation executed by this command **must** terminate
- VUID-vkCmdDispatchGraphIndirectAMDX-None-09600

If a descriptor with type equal to any of VK_DESCRIPTOR_TYPE_SAMPLE_WEIGHT_IMAGE_QCOM, VK_DESCRIPTOR_TYPE_BLOCK_MATCH_IMAGE_QCOM, VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE, VK_DESCRIPTOR_TYPE_STORAGE_IMAGE, or VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT is accessed as a result of this command, the image subresource identified by that descriptor **must** be in the image layout identified when the descriptor was written

- VUID-vkCmdDispatchGraphIndirectAMDX-commandBuffer-09181
  commandBuffer **must** not be a protected command buffer
- VUID-vkCmdDispatchGraphIndirectAMDX-commandBuffer-09182
  commandBuffer **must** be a primary command buffer
- VUID-vkCmdDispatchGraphIndirectAMDX-scratch-09183
  scratch **must** be the device address of an allocated memory range at least as large as the value of VkExecutionGraphPipelineScratchSizeAMDX::size returned by VkExecutionGraphPipelineScratchSizeAMDX for the currently bound execution graph pipeline
- VUID-vkCmdDispatchGraphIndirectAMDX-scratch-09184
  scratch **must** be a device address within a VkBuffer created with the VK_BUFFER_USAGE_EXECUTION_GRAPH_SCRATCH_BIT_AMDX or VK_BUFFER_USAGE_2_EXECUTION_GRAPH_SCRATCH_BIT_AMDX flag
- VUID-vkCmdDispatchGraphIndirectAMDX-scratch-09185
  Device memory in the range [scratch,scratch VkExecutionGraphPipelineScratchSizeAMDX::size) **must** have been initialized with vkCmdInitializeGraphScratchMemoryAMDX using the currently bound execution graph pipeline, and not modified after that by anything other than another execution graph dispatch command
- VUID-vkCmdDispatchGraphIndirectAMDX-maxComputeWorkGroupCount-09186
  Execution of this command **must** not cause a node to be dispatched with a larger number of workgroups than that specified by either a MaxNumWorkgroupsAMDX decoration in the dispatched node or maxComputeWorkGroupCount
- VUID-vkCmdDispatchGraphIndirectAMDX-maxExecutionGraphShaderPayloadCount-09187
  Execution of this command **must** not cause any shader to initialize more than maxExecutionGraphShaderPayloadCount output payloads
- VUID-vkCmdDispatchGraphIndirectAMDX-NodeMaxPayloadsAMDX-09188
  Execution of this command **must** not cause any shader that declares NodeMaxPayloadsAMDX to initialize more output payloads than specified by the max number of payloads for that decoration. This requirement applies to each NodeMaxPayloadsAMDX decoration separately
- VUID-vkCmdDispatchGraphIndirectAMDX-pCountInfo-09150
  pCountInfo->infos **must** be a device pointer to a memory allocation at least as large as the product of count and stride when this command is executed on the device
- VUID-vkCmdDispatchGraphIndirectAMDX-pCountInfo-09151
  pCountInfo->infos **must** be a device address within a VkBuffer created with the VK_BUFFER_USAGE_INDIRECT_BUFFER_BIT flag
- VUID-vkCmdDispatchGraphIndirectAMDX-pCountInfo-09152

pCountInfo->infos **must** be a multiple of executionGraphDispatchAddressAlignment

- VUID-vkCmdDispatchGraphIndirectAMDX-infos-09153
  Device memory locations at indexes in the range [infos, infos + (count*stride)), at a granularity of stride **must** contain valid VkDispatchGraphInfoAMDX structures in the first 24 bytes when this command is executed on the device
- VUID-vkCmdDispatchGraphIndirectAMDX-pCountInfo-09154
  For each VkDispatchGraphInfoAMDX structure in pCountInfo->infos, payloads **must** be a device pointer to a memory allocation at least as large as the product of payloadCount and payloadStride when this command is executed on the device
- VUID-vkCmdDispatchGraphIndirectAMDX-pCountInfo-09155
  For each VkDispatchGraphInfoAMDX structure in pCountInfo->infos, payloads **must** be a device address within a VkBuffer created with the VK_BUFFER_USAGE_INDIRECT_BUFFER_BIT flag
- VUID-vkCmdDispatchGraphIndirectAMDX-pCountInfo-09156
  For each VkDispatchGraphInfoAMDX structure in pCountInfo->infos, payloads **must** be a multiple of executionGraphDispatchAddressAlignment
- VUID-vkCmdDispatchGraphIndirectAMDX-pCountInfo-09157
  For each VkDispatchGraphInfoAMDX structure in pCountInfo->infos, nodeIndex **must** be a valid node index in the currently bound execution graph pipeline, as returned by vkGetExecutionGraphPipelineNodeIndexAMDX when this command is executed on the device
- VUID-vkCmdDispatchGraphIndirectAMDX-pCountInfo-09158
  For each VkDispatchGraphInfoAMDX structure in pCountInfo->infos, device memory locations at indexes in the range [payloads, payloads + (payloadCount * payloadStride)), at a granularity of payloadStride **must** contain a payload matching the size of the input payload expected by the node in nodeIndex in the first bytes when this command is executed on the device

## Valid Usage (Implicit)

- VUID-vkCmdDispatchGraphIndirectAMDX-commandBuffer-parameter
  commandBuffer **must** be a valid VkCommandBuffer handle
- VUID-vkCmdDispatchGraphIndirectAMDX-pCountInfo-parameter
  pCountInfo **must** be a valid pointer to a valid VkDispatchGraphCountInfoAMDX structure
- VUID-vkCmdDispatchGraphIndirectAMDX-commandBuffer-recording
  commandBuffer **must** be in the recording state
- VUID-vkCmdDispatchGraphIndirectAMDX-commandBuffer-cmdpool
  The VkCommandPool that commandBuffer was allocated from **must** support graphics, or compute operations
- VUID-vkCmdDispatchGraphIndirectAMDX-renderpass
  This command **must** only be called outside of a render pass instance
- VUID-vkCmdDispatchGraphIndirectAMDX-videocoding

This command **must** only be called outside of a video coding scope

- VUID-vkCmdDispatchGraphIndirectAMDX-bufferlevel
  `commandBuffer` **must** be a primary `VkCommandBuffer`

## Host Synchronization

- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary | Outside | Outside | Graphics<br>Compute | Action |

To record an execution graph dispatch with all parameters read on device, call:

```
// Provided by VK_AMDX_shader_enqueue
void vkCmdDispatchGraphIndirectCountAMDX(
    VkCommandBuffer                             commandBuffer,
    VkDeviceAddress                             scratch,
    VkDeviceAddress                             countInfo);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `scratch` is a pointer to the scratch memory to be used.
- `countInfo` is a device address of a VkDispatchGraphCountInfoAMDX structure defining the nodes which will be initially executed.

When this command is executed, the nodes specified in `countInfo` are executed. Nodes executed as part of this command are not implicitly synchronized in any way against each other once they are dispatched.

For this command, all pointers in substructures are treated as device pointers and read during device execution of this command. The allocation and contents of these pointers only needs to be valid during device execution. All of these addresses will be read in the `VK_PIPELINE_STAGE_2_COMPUTE_SHADER_BIT` pipeline stage with the `VK_ACCESS_2_SHADER_STORAGE_READ_BIT` access flag.

Execution of this command **may** modify any memory locations in the range [`scratch`,`scratch` + `size`), where `size` is the value returned in VkExecutionGraphPipelineScratchSizeAMDX::`size` by VkExecutionGraphPipelineScratchSizeAMDX for the currently bound execution graph pipeline. Accesses to this memory range are performed in the `VK_PIPELINE_STAGE_2_COMPUTE_SHADER_BIT`

pipeline stage with the VK_ACCESS_2_SHADER_STORAGE_READ_BIT and VK_ACCESS_2_SHADER_STORAGE_WRITE_BIT access flags.

## Valid Usage

- VUID-vkCmdDispatchGraphIndirectCountAMDX-magFilter-04553
  If a VkSampler created with magFilter or minFilter equal to VK_FILTER_LINEAR, reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE, and compareEnable equal to VK_FALSE is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT
- VUID-vkCmdDispatchGraphIndirectCountAMDX-magFilter-09598
  If a VkSampler created with magFilter or minFilter equal to VK_FILTER_LINEAR and reductionMode equal to either VK_SAMPLER_REDUCTION_MODE_MIN or VK_SAMPLER_REDUCTION_MODE_MAX is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT
- VUID-vkCmdDispatchGraphIndirectCountAMDX-mipmapMode-04770
  If a VkSampler created with mipmapMode equal to VK_SAMPLER_MIPMAP_MODE_LINEAR, reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE, and compareEnable equal to VK_FALSE is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT
- VUID-vkCmdDispatchGraphIndirectCountAMDX-mipmapMode-09599
  If a VkSampler created with mipmapMode equal to VK_SAMPLER_MIPMAP_MODE_LINEAR and reductionMode equal to either VK_SAMPLER_REDUCTION_MODE_MIN or VK_SAMPLER_REDUCTION_MODE_MAX is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT
- VUID-vkCmdDispatchGraphIndirectCountAMDX-unnormalizedCoordinates-09635
  If a VkSampler created with unnormalizedCoordinates equal to VK_TRUE is used to sample a VkImageView as a result of this command, then the image view's levelCount and layerCount **must** be 1
- VUID-vkCmdDispatchGraphIndirectCountAMDX-unnormalizedCoordinates-09636
  If a VkSampler created with unnormalizedCoordinates equal to VK_TRUE is used to sample a VkImageView as a result of this command, then the image view's viewType **must** be VK_IMAGE_VIEW_TYPE_1D or VK_IMAGE_VIEW_TYPE_2D
- VUID-vkCmdDispatchGraphIndirectCountAMDX-None-06479
  If a VkImageView is sampled with depth comparison, the image view's format features **must** contain VK_FORMAT_FEATURE_2_SAMPLED_IMAGE_DEPTH_COMPARISON_BIT
- VUID-vkCmdDispatchGraphIndirectCountAMDX-None-02691
  If a VkImageView is accessed using atomic operations as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_STORAGE_IMAGE_ATOMIC_BIT

- VUID-vkCmdDispatchGraphIndirectCountAMDX-None-07888
  If a VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER descriptor is accessed using atomic operations as a result of this command, then the storage texel buffer's format features **must** contain VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_ATOMIC_BIT
- VUID-vkCmdDispatchGraphIndirectCountAMDX-None-02692
  If a VkImageView is sampled with VK_FILTER_CUBIC_EXT as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_CUBIC_BIT_EXT
- VUID-vkCmdDispatchGraphIndirectCountAMDX-None-02693
  If the VK_EXT_filter_cubic extension is not enabled and any VkImageView is sampled with VK_FILTER_CUBIC_EXT as a result of this command, it **must** not have a VkImageViewType of VK_IMAGE_VIEW_TYPE_3D, VK_IMAGE_VIEW_TYPE_CUBE, or VK_IMAGE_VIEW_TYPE_CUBE_ARRAY
- VUID-vkCmdDispatchGraphIndirectCountAMDX-filterCubic-02694
  Any VkImageView being sampled with VK_FILTER_CUBIC_EXT as a result of this command **must** have a VkImageViewType and format that supports cubic filtering, as specified by VkFilterCubicImageViewImageFormatPropertiesEXT::filterCubic returned by vkGetPhysicalDeviceImageFormatProperties2
- VUID-vkCmdDispatchGraphIndirectCountAMDX-filterCubicMinmax-02695
  Any VkImageView being sampled with VK_FILTER_CUBIC_EXT with a reduction mode of either VK_SAMPLER_REDUCTION_MODE_MIN or VK_SAMPLER_REDUCTION_MODE_MAX as a result of this command **must** have a VkImageViewType and format that supports cubic filtering together with minmax filtering, as specified by VkFilterCubicImageViewImageFormatPropertiesEXT::filterCubicMinmax returned by vkGetPhysicalDeviceImageFormatProperties2
- VUID-vkCmdDispatchGraphIndirectCountAMDX-cubicRangeClamp-09212
  If the cubicRangeClamp feature is not enabled, then any VkImageView being sampled with VK_FILTER_CUBIC_EXT as a result of this command **must** not have a VkSamplerReductionModeCreateInfo::reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE_RANGECLAMP_QCOM
- VUID-vkCmdDispatchGraphIndirectCountAMDX-reductionMode-09213
  Any VkImageView being sampled with a VkSamplerReductionModeCreateInfo::reductionMode equal to VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE_RANGECLAMP_QCOM as a result of this command **must** sample with VK_FILTER_CUBIC_EXT
- VUID-vkCmdDispatchGraphIndirectCountAMDX-selectableCubicWeights-09214
  If the selectableCubicWeights feature is not enabled, then any VkImageView being sampled with VK_FILTER_CUBIC_EXT as a result of this command **must** have VkSamplerCubicWeightsCreateInfoQCOM::cubicWeights equal to VK_CUBIC_FILTER_WEIGHTS_CATMULL_ROM_QCOM
- VUID-vkCmdDispatchGraphIndirectCountAMDX-flags-02696
  Any VkImage created with a VkImageCreateInfo::flags containing VK_IMAGE_CREATE_CORNER_SAMPLED_BIT_NV sampled as a result of this command **must** only be sampled using a VkSamplerAddressMode of VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_EDGE
- VUID-vkCmdDispatchGraphIndirectCountAMDX-OpTypeImage-07027
  For any VkImageView being written as a storage image where the image format field of

the OpTypeImage is Unknown, the view's format features **must** contain VK_FORMAT_FEATURE_2_STORAGE_WRITE_WITHOUT_FORMAT_BIT

- VUID-vkCmdDispatchGraphIndirectCountAMDX-OpTypeImage-07028
  For any VkImageView being read as a storage image where the image format field of the OpTypeImage is Unknown, the view's format features **must** contain VK_FORMAT_FEATURE_2_STORAGE_READ_WITHOUT_FORMAT_BIT
- VUID-vkCmdDispatchGraphIndirectCountAMDX-OpTypeImage-07029
  For any VkBufferView being written as a storage texel buffer where the image format field of the OpTypeImage is Unknown, the view's buffer features **must** contain VK_FORMAT_FEATURE_2_STORAGE_WRITE_WITHOUT_FORMAT_BIT
- VUID-vkCmdDispatchGraphIndirectCountAMDX-OpTypeImage-07030
  Any VkBufferView being read as a storage texel buffer where the image format field of the OpTypeImage is Unknown then the view's buffer features **must** contain VK_FORMAT_FEATURE_2_STORAGE_READ_WITHOUT_FORMAT_BIT
- VUID-vkCmdDispatchGraphIndirectCountAMDX-None-08600
  For each set *n* that is statically used by a bound shader, a descriptor set **must** have been bound to *n* at the same pipeline bind point, with a VkPipelineLayout that is compatible for set *n*, with the VkPipelineLayout used to create the current VkPipeline or the VkDescriptorSetLayout array used to create the current VkShaderEXT , as described in Pipeline Layout Compatibility
- VUID-vkCmdDispatchGraphIndirectCountAMDX-None-08601
  For each push constant that is statically used by a bound shader, a push constant value **must** have been set for the same pipeline bind point, with a VkPipelineLayout that is compatible for push constants, with the VkPipelineLayout used to create the current VkPipeline or the VkDescriptorSetLayout array used to create the current VkShaderEXT , as described in Pipeline Layout Compatibility
- VUID-vkCmdDispatchGraphIndirectCountAMDX-None-10068
  For each array of resources that is used by a bound shader, the indices used to access members of the array **must** be less than the descriptor count for the identified binding in the descriptor sets used by this command
- VUID-vkCmdDispatchGraphIndirectCountAMDX-maintenance4-08602
  If the maintenance4 feature is not enabled, then for each push constant that is statically used by a bound shader, a push constant value **must** have been set for the same pipeline bind point, with a VkPipelineLayout that is compatible for push constants, with the VkPipelineLayout used to create the current VkPipeline or the VkDescriptorSetLayout and VkPushConstantRange arrays used to create the current VkShaderEXT , as described in Pipeline Layout Compatibility
- VUID-vkCmdDispatchGraphIndirectCountAMDX-None-08114
  Descriptors in each bound descriptor set, specified via vkCmdBindDescriptorSets, **must** be valid as described by descriptor validity if they are statically used by the VkPipeline bound to the pipeline bind point used by this command and the bound VkPipeline was not created with VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT
- VUID-vkCmdDispatchGraphIndirectCountAMDX-None-08115
  If the descriptors used by the VkPipeline bound to the pipeline bind point were specified

via vkCmdBindDescriptorSets, the bound VkPipeline **must** have been created without VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT

- VUID-vkCmdDispatchGraphIndirectCountAMDX-None-08116
  Descriptors in bound descriptor buffers, specified via vkCmdSetDescriptorBufferOffsetsEXT, **must** be valid if they are dynamically used by the VkPipeline bound to the pipeline bind point used by this command and the bound VkPipeline was created with VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT
- VUID-vkCmdDispatchGraphIndirectCountAMDX-None-08604
  Descriptors in bound descriptor buffers, specified via vkCmdSetDescriptorBufferOffsetsEXT, **must** be valid if they are dynamically used by any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command
- VUID-vkCmdDispatchGraphIndirectCountAMDX-None-08117
  If the descriptors used by the VkPipeline bound to the pipeline bind point were specified via vkCmdSetDescriptorBufferOffsetsEXT, the bound VkPipeline **must** have been created with VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT
- VUID-vkCmdDispatchGraphIndirectCountAMDX-None-08119
  If a descriptor is dynamically used with a VkPipeline created with VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT, the descriptor memory **must** be resident
- VUID-vkCmdDispatchGraphIndirectCountAMDX-None-08605
  If a descriptor is dynamically used with a VkShaderEXT created with a VkDescriptorSetLayout that was created with VK_DESCRIPTOR_SET_LAYOUT_CREATE_DESCRIPTOR_BUFFER_BIT_EXT, the descriptor memory **must** be resident
- VUID-vkCmdDispatchGraphIndirectCountAMDX-None-08606
  If the shaderObject feature is not enabled, a valid pipeline **must** be bound to the pipeline bind point used by this command
- VUID-vkCmdDispatchGraphIndirectCountAMDX-None-08608
  If a pipeline is bound to the pipeline bind point used by this command, there **must** not have been any calls to dynamic state setting commands for any state specified statically in the VkPipeline object bound to the pipeline bind point used by this command, since that pipeline was bound
- VUID-vkCmdDispatchGraphIndirectCountAMDX-None-08609
  If the VkPipeline object bound to the pipeline bind point used by this command or any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a VkSampler object that uses unnormalized coordinates, that sampler **must** not be used to sample from any VkImage with a VkImageView of the type VK_IMAGE_VIEW_TYPE_3D, VK_IMAGE_VIEW_TYPE_CUBE, VK_IMAGE_VIEW_TYPE_1D_ARRAY, VK_IMAGE_VIEW_TYPE_2D_ARRAY or VK_IMAGE_VIEW_TYPE_CUBE_ARRAY, in any shader stage
- VUID-vkCmdDispatchGraphIndirectCountAMDX-None-08610
  If the VkPipeline object bound to the pipeline bind point used by this command or any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a VkSampler object that uses unnormalized coordinates, that sampler **must** not be used with any of the SPIR-V OpImageSample* or OpImageSparseSample*

instructions with `ImplicitLod`, `Dref` or `Proj` in their name, in any shader stage

- VUID-vkCmdDispatchGraphIndirectCountAMDX-None-08611
  If the VkPipeline object bound to the pipeline bind point used by this command or any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a VkSampler object that uses unnormalized coordinates, that sampler **must** not be used with any of the SPIR-V `OpImageSample*` or `OpImageSparseSample*` instructions that includes a LOD bias or any offset values, in any shader stage

- VUID-vkCmdDispatchGraphIndirectCountAMDX-None-08607
  If the `shaderObject` is enabled, either a valid pipeline **must** be bound to the pipeline bind point used by this command, or a valid combination of valid and VK_NULL_HANDLE shader objects **must** be bound to every supported shader stage corresponding to the pipeline bind point used by this command

- VUID-vkCmdDispatchGraphIndirectCountAMDX-uniformBuffers-06935
  If any stage of the VkPipeline object bound to the pipeline bind point used by this command accesses a uniform buffer, and that stage was created without enabling either `VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_EXT` or `VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_2_EXT` for `uniformBuffers`, and the `robustBufferAccess` feature is not enabled, that stage **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point

- VUID-vkCmdDispatchGraphIndirectCountAMDX-None-08612
  If the `robustBufferAccess` feature is not enabled, and any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a uniform buffer, it **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point

- VUID-vkCmdDispatchGraphIndirectCountAMDX-storageBuffers-06936
  If any stage of the VkPipeline object bound to the pipeline bind point used by this command accesses a storage buffer, and that stage was created without enabling either `VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_EXT` or `VK_PIPELINE_ROBUSTNESS_BUFFER_BEHAVIOR_ROBUST_BUFFER_ACCESS_2_EXT` for `storageBuffers`, and the `robustBufferAccess` feature is not enabled, that stage **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point

- VUID-vkCmdDispatchGraphIndirectCountAMDX-None-08613
  If the `robustBufferAccess` feature is not enabled, and any VkShaderEXT bound to a stage corresponding to the pipeline bind point used by this command accesses a storage buffer, it **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point

- VUID-vkCmdDispatchGraphIndirectCountAMDX-commandBuffer-02707
  If `commandBuffer` is an unprotected command buffer and `protectedNoFault` is not supported, any resource accessed by bound shaders **must** not be a protected resource

- VUID-vkCmdDispatchGraphIndirectCountAMDX-None-06550
  If a bound shader accesses a VkSampler or VkImageView object that enables sampler $Y'C_BC_R$ conversion, that object **must** only be used with `OpImageSample*` or `OpImageSparseSample*` instructions

- VUID-vkCmdDispatchGraphIndirectCountAMDX-ConstOffset-06551
  If a bound shader accesses a VkSampler or VkImageView object that enables sampler $Y'C_BC_R$ conversion, that object **must** not use the `ConstOffset` and `Offset` operands
- VUID-vkCmdDispatchGraphIndirectCountAMDX-viewType-07752
  If a VkImageView is accessed as a result of this command, then the image view's `viewType` **must** match the `Dim` operand of the `OpTypeImage` as described in Compatibility Between SPIR-V Image Dimensions and Vulkan ImageView Types
- VUID-vkCmdDispatchGraphIndirectCountAMDX-format-07753
  If a VkImageView is accessed as a result of this command, then the numeric type of the image view's `format` and the `Sampled Type` operand of the `OpTypeImage` **must** match
- VUID-vkCmdDispatchGraphIndirectCountAMDX-OpImageWrite-08795
  If a VkImageView created with a format other than `VK_FORMAT_A8_UNORM_KHR` is accessed using `OpImageWrite` as a result of this command, then the `Type` of the `Texel` operand of that instruction **must** have at least as many components as the image view's format
- VUID-vkCmdDispatchGraphIndirectCountAMDX-OpImageWrite-08796
  If a VkImageView created with the format `VK_FORMAT_A8_UNORM_KHR` is accessed using `OpImageWrite` as a result of this command, then the `Type` of the `Texel` operand of that instruction **must** have four components
- VUID-vkCmdDispatchGraphIndirectCountAMDX-OpImageWrite-04469
  If a VkBufferView is accessed using `OpImageWrite` as a result of this command, then the `Type` of the `Texel` operand of that instruction **must** have at least as many components as the buffer view's format
- VUID-vkCmdDispatchGraphIndirectCountAMDX-SampledType-04470
  If a VkImageView with a VkFormat that has a 64-bit component width is accessed as a result of this command, the `SampledType` of the `OpTypeImage` operand of that instruction **must** have a `Width` of 64
- VUID-vkCmdDispatchGraphIndirectCountAMDX-SampledType-04471
  If a VkImageView with a VkFormat that has a component width less than 64-bit is accessed as a result of this command, the `SampledType` of the `OpTypeImage` operand of that instruction **must** have a `Width` of 32
- VUID-vkCmdDispatchGraphIndirectCountAMDX-SampledType-04472
  If a VkBufferView with a VkFormat that has a 64-bit component width is accessed as a result of this command, the `SampledType` of the `OpTypeImage` operand of that instruction **must** have a `Width` of 64
- VUID-vkCmdDispatchGraphIndirectCountAMDX-SampledType-04473
  If a VkBufferView with a VkFormat that has a component width less than 64-bit is accessed as a result of this command, the `SampledType` of the `OpTypeImage` operand of that instruction **must** have a `Width` of 32
- VUID-vkCmdDispatchGraphIndirectCountAMDX-sparseImageInt64Atomics-04474
  If the `sparseImageInt64Atomics` feature is not enabled, VkImage objects created with the `VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT` flag **must** not be accessed by atomic instructions through an `OpTypeImage` with a `SampledType` with a `Width` of 64 by this command
- VUID-vkCmdDispatchGraphIndirectCountAMDX-sparseImageInt64Atomics-04475

If the sparseImageInt64Atomics feature is not enabled, VkBuffer objects created with the VK_BUFFER_CREATE_SPARSE_RESIDENCY_BIT flag **must** not be accessed by atomic instructions through an OpTypeImage with a SampledType with a Width of 64 by this command

- VUID-vkCmdDispatchGraphIndirectCountAMDX-OpImageWeightedSampleQCOM-06971
  If OpImageWeightedSampleQCOM is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_WEIGHT_SAMPLED_IMAGE_BIT_QCOM
- VUID-vkCmdDispatchGraphIndirectCountAMDX-OpImageWeightedSampleQCOM-06972
  If OpImageWeightedSampleQCOM uses a VkImageView as a sample weight image as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_WEIGHT_IMAGE_BIT_QCOM
- VUID-vkCmdDispatchGraphIndirectCountAMDX-OpImageBoxFilterQCOM-06973
  If OpImageBoxFilterQCOM is used to sample a VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_BOX_FILTER_SAMPLED_BIT_QCOM
- VUID-vkCmdDispatchGraphIndirectCountAMDX-OpImageBlockMatchSSDQCOM-06974
  If OpImageBlockMatchSSDQCOM is used to read from an VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM
- VUID-vkCmdDispatchGraphIndirectCountAMDX-OpImageBlockMatchSADQCOM-06975
  If OpImageBlockMatchSADQCOM is used to read from an VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM
- VUID-vkCmdDispatchGraphIndirectCountAMDX-OpImageBlockMatchSADQCOM-06976
  If OpImageBlockMatchSADQCOM or OpImageBlockMatchSSDQCOM is used to read from a reference image as result of this command, then the specified reference coordinates **must** not fail integer texel coordinate validation
- VUID-vkCmdDispatchGraphIndirectCountAMDX-OpImageWeightedSampleQCOM-06977
  If OpImageWeightedSampleQCOM, OpImageBoxFilterQCOM, OpImageBlockMatchWindowSSDQCOM, OpImageBlockMatchWindowSADQCOM, OpImageBlockMatchGatherSSDQCOM, OpImageBlockMatchGatherSADQCOM, OpImageBlockMatchSSDQCOM, or OpImageBlockMatchSADQCOM uses a VkSampler as a result of this command, then the sampler **must** have been created with VK_SAMPLER_CREATE_IMAGE_PROCESSING_BIT_QCOM
- VUID-vkCmdDispatchGraphIndirectCountAMDX-OpImageWeightedSampleQCOM-06978
  If any command other than OpImageWeightedSampleQCOM, OpImageBoxFilterQCOM, OpImageBlockMatchWindowSSDQCOM, OpImageBlockMatchWindowSADQCOM, OpImageBlockMatchGatherSSDQCOM, OpImageBlockMatchGatherSADQCOM, OpImageBlockMatchSSDQCOM, or OpImageBlockMatchSADQCOM uses a VkSampler as a result of this command, then the sampler **must** not have been created with VK_SAMPLER_CREATE_IMAGE_PROCESSING_BIT_QCOM
- VUID-vkCmdDispatchGraphIndirectCountAMDX-OpImageBlockMatchWindow-09215
  If a OpImageBlockMatchWindow*QCOM or OpImageBlockMatchGather*QCOM instruction is used to read from an VkImageView as a result of this command, then the image view's format features **must** contain VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM

- VUID-vkCmdDispatchGraphIndirectCountAMDX-OpImageBlockMatchWindow-09216
  If a `OpImageBlockMatchWindow*QCOM` or `OpImageBlockMatchGather*QCOM` instruction is used to read from an VkImageView as a result of this command, then the image view's format **must** be a single-component format
- VUID-vkCmdDispatchGraphIndirectCountAMDX-OpImageBlockMatchWindow-09217
  If a `OpImageBlockMatchWindow*QCOM` or `OpImageBlockMatchGather*QCOM` read from a reference image as result of this command, then the specified reference coordinates **must** not fail integer texel coordinate validation
- VUID-vkCmdDispatchGraphIndirectCountAMDX-None-07288
  Any shader invocation executed by this command **must** terminate
- VUID-vkCmdDispatchGraphIndirectCountAMDX-None-09600
  If a descriptor with type equal to any of `VK_DESCRIPTOR_TYPE_SAMPLE_WEIGHT_IMAGE_QCOM`, `VK_DESCRIPTOR_TYPE_BLOCK_MATCH_IMAGE_QCOM`, `VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE`, `VK_DESCRIPTOR_TYPE_STORAGE_IMAGE`, or `VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT` is accessed as a result of this command, the image subresource identified by that descriptor **must** be in the image layout identified when the descriptor was written
- VUID-vkCmdDispatchGraphIndirectCountAMDX-commandBuffer-09181
  `commandBuffer` **must** not be a protected command buffer
- VUID-vkCmdDispatchGraphIndirectCountAMDX-commandBuffer-09182
  `commandBuffer` **must** be a primary command buffer
- VUID-vkCmdDispatchGraphIndirectCountAMDX-scratch-09183
  `scratch` **must** be the device address of an allocated memory range at least as large as the value of VkExecutionGraphPipelineScratchSizeAMDX::`size` returned by VkExecutionGraphPipelineScratchSizeAMDX for the currently bound execution graph pipeline
- VUID-vkCmdDispatchGraphIndirectCountAMDX-scratch-09184
  `scratch` **must** be a device address within a VkBuffer created with the `VK_BUFFER_USAGE_EXECUTION_GRAPH_SCRATCH_BIT_AMDX` or `VK_BUFFER_USAGE_2_EXECUTION_GRAPH_SCRATCH_BIT_AMDX` flag
- VUID-vkCmdDispatchGraphIndirectCountAMDX-scratch-09185
  Device memory in the range [`scratch`,`scratch` VkExecutionGraphPipelineScratchSizeAMDX::`size`) **must** have been initialized with vkCmdInitializeGraphScratchMemoryAMDX using the currently bound execution graph pipeline, and not modified after that by anything other than another execution graph dispatch command
- VUID-vkCmdDispatchGraphIndirectCountAMDX-maxComputeWorkGroupCount-09186
  Execution of this command **must** not cause a node to be dispatched with a larger number of workgroups than that specified by either a `MaxNumWorkgroupsAMDX` decoration in the dispatched node or `maxComputeWorkGroupCount`
- VUID-vkCmdDispatchGraphIndirectCountAMDX-maxExecutionGraphShaderPayloadCount-09187
  Execution of this command **must** not cause any shader to initialize more than `maxExecutionGraphShaderPayloadCount` output payloads

- VUID-vkCmdDispatchGraphIndirectCountAMDX-NodeMaxPayloadsAMDX-09188
  Execution of this command **must** not cause any shader that declares `NodeMaxPayloadsAMDX` to initialize more output payloads than specified by the max number of payloads for that decoration. This requirement applies to each `NodeMaxPayloadsAMDX` decoration separately
- VUID-vkCmdDispatchGraphIndirectCountAMDX-countInfo-09159
  `countInfo` **must** be a device pointer to a memory allocation containing a valid VkDispatchGraphCountInfoAMDX structure when this command is executed on the device
- VUID-vkCmdDispatchGraphIndirectCountAMDX-countInfo-09160
  `countInfo` **must** be a device address within a VkBuffer created with the `VK_BUFFER_USAGE_INDIRECT_BUFFER_BIT` flag
- VUID-vkCmdDispatchGraphIndirectCountAMDX-countInfo-09161
  `countInfo` **must** be a multiple of `executionGraphDispatchAddressAlignment`
- VUID-vkCmdDispatchGraphIndirectCountAMDX-countInfo-09162
  `countInfo->infos` **must** be a device pointer to a memory allocation at least as large as the product of `count` and `stride` when this command is executed on the device
- VUID-vkCmdDispatchGraphIndirectCountAMDX-countInfo-09163
  `countInfo->infos` **must** be a device address within a VkBuffer created with the `VK_BUFFER_USAGE_INDIRECT_BUFFER_BIT` flag
- VUID-vkCmdDispatchGraphIndirectCountAMDX-countInfo-09164
  `countInfo->infos` **must** be a multiple of `executionGraphDispatchAddressAlignment`
- VUID-vkCmdDispatchGraphIndirectCountAMDX-infos-09165
  Device memory locations at indexes in the range [`infos`, `infos` + (`count`*`stride`)), at a granularity of `stride` **must** contain valid VkDispatchGraphInfoAMDX structures in the first 24 bytes when this command is executed on the device
- VUID-vkCmdDispatchGraphIndirectCountAMDX-countInfo-09166
  For each VkDispatchGraphInfoAMDX structure in `countInfo->infos`, `payloads` **must** be a device pointer to a memory allocation at least as large as the product of `payloadCount` and `payloadStride` when this command is executed on the device
- VUID-vkCmdDispatchGraphIndirectCountAMDX-countInfo-09167
  For each VkDispatchGraphInfoAMDX structure in `countInfo->infos`, `payloads` **must** be a device address within a VkBuffer created with the `VK_BUFFER_USAGE_INDIRECT_BUFFER_BIT` flag
- VUID-vkCmdDispatchGraphIndirectCountAMDX-countInfo-09168
  For each VkDispatchGraphInfoAMDX structure in `countInfo->infos`, `payloads` **must** be a multiple of `executionGraphDispatchAddressAlignment`
- VUID-vkCmdDispatchGraphIndirectCountAMDX-countInfo-09169
  For each VkDispatchGraphInfoAMDX structure in `countInfo->infos`, `nodeIndex` **must** be a valid node index in the currently bound execution graph pipeline, as returned by vkGetExecutionGraphPipelineNodeIndexAMDX when this command is executed on the device
- VUID-vkCmdDispatchGraphIndirectCountAMDX-countInfo-09170
  For each VkDispatchGraphInfoAMDX structure in `countInfo->infos`, device memory

locations at indexes in the range [payloads, payloads + (payloadCount * payloadStride)), at a granularity of payloadStride **must** contain a payload matching the size of the input payload expected by the node in nodeIndex in the first bytes when this command is executed on the device

## Valid Usage (Implicit)

- VUID-vkCmdDispatchGraphIndirectCountAMDX-commandBuffer-parameter
  commandBuffer **must** be a valid VkCommandBuffer handle
- VUID-vkCmdDispatchGraphIndirectCountAMDX-commandBuffer-recording
  commandBuffer **must** be in the recording state
- VUID-vkCmdDispatchGraphIndirectCountAMDX-commandBuffer-cmdpool
  The VkCommandPool that commandBuffer was allocated from **must** support graphics, or compute operations
- VUID-vkCmdDispatchGraphIndirectCountAMDX-renderpass
  This command **must** only be called outside of a render pass instance
- VUID-vkCmdDispatchGraphIndirectCountAMDX-videocoding
  This command **must** only be called outside of a video coding scope
- VUID-vkCmdDispatchGraphIndirectCountAMDX-bufferlevel
  commandBuffer **must** be a primary VkCommandBuffer

## Host Synchronization

- Host access to the VkCommandPool that commandBuffer was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary | Outside | Outside | Graphics<br>Compute | Action |

The VkDeviceOrHostAddressConstAMDX union is defined as:

```
// Provided by VK_AMDX_shader_enqueue
typedef union VkDeviceOrHostAddressConstAMDX {
    VkDeviceAddress    deviceAddress;
    const void*        hostAddress;
} VkDeviceOrHostAddressConstAMDX;
```

- deviceAddress is a buffer device address as returned by the vkGetBufferDeviceAddressKHR command.
- hostAddress is a const host memory address.

The VkDispatchGraphCountInfoAMDX structure is defined as:

```
// Provided by VK_AMDX_shader_enqueue
typedef struct VkDispatchGraphCountInfoAMDX {
    uint32_t                        count;
    VkDeviceOrHostAddressConstAMDX  infos;
    uint64_t                        stride;
} VkDispatchGraphCountInfoAMDX;
```

- count is the number of dispatches to perform.
- infos is the device or host address of a flat array of VkDispatchGraphInfoAMDX structures
- stride is the byte stride between successive VkDispatchGraphInfoAMDX structures in infos

Whether infos is consumed as a device or host pointer is defined by the command this structure is used in.

The VkDispatchGraphInfoAMDX structure is defined as:

```
// Provided by VK_AMDX_shader_enqueue
typedef struct VkDispatchGraphInfoAMDX {
    uint32_t                        nodeIndex;
    uint32_t                        payloadCount;
    VkDeviceOrHostAddressConstAMDX  payloads;
    uint64_t                        payloadStride;
} VkDispatchGraphInfoAMDX;
```

- nodeIndex is the index of a node in an execution graph to be dispatched.
- payloadCount is the number of payloads to dispatch for the specified node.
- payloads is a device or host address pointer to a flat array of payloads with size equal to the product of payloadCount and payloadStride
- payloadStride is the byte stride between successive payloads in payloads

Whether payloads is consumed as a device or host pointer is defined by the command this structure is used in.

### Valid Usage

- VUID-VkDispatchGraphInfoAMDX-payloadCount-09171
  payloadCount **must** be no greater than maxExecutionGraphShaderPayloadCount

## 44.4. Shader Enqueue

Compute shaders in an execution graph **can** use the `OpInitializeNodePayloadsAMDX` to initialize nodes for dispatch. Any node payload initialized in this way will be enqueued for dispatch once the shader is done writing to the payload. As compilers **may** be conservative when making this determination, shaders **can** further call `OpFinalizeNodePayloadsAMDX` to guarantee that the payload is no longer being written.

The `Node Name` operand of the `PayloadNodeNameAMDX` decoration on a payload identifies the shader name of the node to be enqueued, and the `Shader Index` operand of `OpInitializeNodePayloadsAMDX` identifies the shader index. A node identified in this way is dispatched as described in the following sections.

### 44.4.1. Compute Nodes

Compute shaders added as nodes to an execution graph are executed differently based on the presence or absence of the `StaticNumWorkgroupsAMDX` or `CoalescingAMDX` execution modes.

Dispatching a compute shader node that does not declare either the `StaticNumWorkgroupsAMDX` or `CoalescingAMDX` execution mode will execute a number of workgroups in each dimension specified by the first 12 bytes of the payload, interpreted as a VkDispatchIndirectCommand. The same payload will be broadcast to each workgroup in the same dispatch. Additional values in the payload are have no effect on execution.

Dispatching a compute shader node with the `StaticNumWorkgroupsAMDX` execution mode will execute workgroups in each dimension according to the `x`, `y`, and `z size` operands to the `StaticNumWorkgroupsAMDX` execution mode. The same payload will be broadcast to each workgroup in the same dispatch. Any values in the payload have no effect on execution.

Dispatching a compute shader node with the `CoalescingAMDX` execution mode will enqueue a single invocation for execution. Implementations **may** combine multiple such dispatches into the same workgroup, up to the size of the workgroup. The number of invocations coalesced into a given workgroup in this way **can** be queried via the `CoalescedInputCountAMDX` built-in. Any values in the payload have no effect on execution.

# Chapter 45. Extending Vulkan

New functionality **may** be added to Vulkan via either new extensions or new versions of the core, or new versions of an extension in some cases.

This chapter describes how Vulkan is versioned, how compatibility is affected between different versions, and compatibility rules that are followed by the Vulkan Working Group.

## 45.1. Instance and Device Functionality

Commands that enumerate instance properties, or that accept a VkInstance object as a parameter, are considered instance-level functionality.

Commands that dispatch from a VkDevice object or a child object of a VkDevice, or take any of them as a parameter, are considered device-level functionality. Types defined by a device extension are also considered device-level functionality.

Commands that dispatch from VkPhysicalDevice, or accept a VkPhysicalDevice object as a parameter, are considered either instance-level or device-level functionality depending if the functionality is specified by an instance extension or device extension respectively.

Additionally, commands that enumerate physical device properties are considered device-level functionality.

**NOTE** Applications usually interface to Vulkan using a loader that implements only instance-level functionality, passing device-level functionality to implementations of the full Vulkan API on the system. In some circumstances, as these may be implemented independently, it is possible that the loader and device implementations on a given installation will support different versions. To allow for this and call out when it happens, the Vulkan specification enumerates device and instance level functionality separately - they have independent version queries.

**NOTE** Vulkan 1.0 initially specified new physical device enumeration functionality as instance-level, requiring it to be included in an instance extension. As the capabilities of device-level functionality require discovery via physical device enumeration, this led to the situation where many device extensions required an instance extension as well. To alleviate this extra work, `VK_KHR_get_physical_device_properties2` (and subsequently Vulkan 1.1) redefined device-level functionality to include physical device enumeration.

## 45.2. Core Versions

The Vulkan Specification is regularly updated with bug fixes and clarifications. Occasionally new functionality is added to the core and at some point it is expected that there will be a desire to perform a large, breaking change to the API. In order to indicate to developers how and when these changes are made to the specification, and to provide a way to identify each set of changes, the Vulkan API maintains a version number.

### 45.2.1. Version Numbers

The Vulkan version number comprises four parts indicating the variant, major, minor and patch version of the Vulkan API Specification.

The *variant* indicates the variant of the Vulkan API supported by the implementation. This is always 0 for the Vulkan API.

**NOTE** A non-zero variant indicates the API is a variant of the Vulkan API and applications will typically need to be modified to run against it. The variant field was a later addition to the version number, added in version 1.2.175 of the Specification. As Vulkan uses variant 0, this change is fully backwards compatible with the previous version number format for Vulkan implementations. New version number macros have been added for this change and the old macros deprecated. For existing applications using the older format and macros, an implementation with non-zero variant will decode as a very high Vulkan version. The high version number should be detectable by applications performing suitable version checking.

The *major version* indicates a significant change in the API, which will encompass a wholly new version of the specification.

The *minor version* indicates the incorporation of new functionality into the core specification.

The *patch version* indicates bug fixes, clarifications, and language improvements have been incorporated into the specification.

Compatibility guarantees made about versions of the API sharing any of the same version numbers are documented in Core Versions

The version number is used in several places in the API. In each such use, the version numbers are packed into a 32-bit integer as follows:

- The variant is a 3-bit integer packed into bits 31-29.
- The major version is a 7-bit integer packed into bits 28-22.
- The minor version number is a 10-bit integer packed into bits 21-12.
- The patch version number is a 12-bit integer packed into bits 11-0.

`VK_API_VERSION_VARIANT` extracts the API variant number from a packed version number:

```
// Provided by VK_VERSION_1_0
#define VK_API_VERSION_VARIANT(version) ((uint32_t)(version) >> 29U)
```

`VK_API_VERSION_MAJOR` extracts the API major version number from a packed version number:

```
// Provided by VK_VERSION_1_0
#define VK_API_VERSION_MAJOR(version) (((uint32_t)(version) >> 22U) & 0x7FU)
```

VK_VERSION_MAJOR extracts the API major version number from a packed version number:

```
// Provided by VK_VERSION_1_0
// VK_VERSION_MAJOR is deprecated, but no reason was given in the API XML
// DEPRECATED: This define is deprecated. VK_API_VERSION_MAJOR should be used instead.
#define VK_VERSION_MAJOR(version) ((uint32_t)(version) >> 22U)
```

VK_API_VERSION_MINOR extracts the API minor version number from a packed version number:

```
// Provided by VK_VERSION_1_0
#define VK_API_VERSION_MINOR(version) (((uint32_t)(version) >> 12U) & 0x3FFU)
```

VK_VERSION_MINOR extracts the API minor version number from a packed version number:

```
// Provided by VK_VERSION_1_0
// VK_VERSION_MINOR is deprecated, but no reason was given in the API XML
// DEPRECATED: This define is deprecated. VK_API_VERSION_MINOR should be used instead.
#define VK_VERSION_MINOR(version) (((uint32_t)(version) >> 12U) & 0x3FFU)
```

VK_API_VERSION_PATCH extracts the API patch version number from a packed version number:

```
// Provided by VK_VERSION_1_0
#define VK_API_VERSION_PATCH(version) ((uint32_t)(version) & 0xFFFU)
```

VK_VERSION_PATCH extracts the API patch version number from a packed version number:

```
// Provided by VK_VERSION_1_0
// VK_VERSION_PATCH is deprecated, but no reason was given in the API XML
// DEPRECATED: This define is deprecated. VK_API_VERSION_PATCH should be used instead.
#define VK_VERSION_PATCH(version) ((uint32_t)(version) & 0xFFFU)
```

VK_MAKE_API_VERSION constructs an API version number.

```
// Provided by VK_VERSION_1_0
#define VK_MAKE_API_VERSION(variant, major, minor, patch) \
    ((((uint32_t)(variant)) << 29U) | (((uint32_t)(major)) << 22U) |
(((uint32_t)(minor)) << 12U) | ((uint32_t)(patch)))
```

- variant is the variant number.
- major is the major version number.
- minor is the minor version number.
- patch is the patch version number.

VK_MAKE_VERSION constructs an API version number.

```
// Provided by VK_VERSION_1_0
// VK_MAKE_VERSION is deprecated, but no reason was given in the API XML
// DEPRECATED: This define is deprecated. VK_MAKE_API_VERSION should be used instead.
#define VK_MAKE_VERSION(major, minor, patch) \
    ((((uint32_t)(major)) << 22U) | (((uint32_t)(minor)) << 12U) |
((uint32_t)(patch)))
```

- major is the major version number.
- minor is the minor version number.
- patch is the patch version number.

VK_API_VERSION_1_0 returns the API version number for Vulkan 1.0.0.

```
// Provided by VK_VERSION_1_0
// Vulkan 1.0 version number
#define VK_API_VERSION_1_0 VK_MAKE_API_VERSION(0, 1, 0, 0)// Patch version should
always be set to 0
```

VK_API_VERSION_1_1 returns the API version number for Vulkan 1.1.0.

```
// Provided by VK_VERSION_1_1
// Vulkan 1.1 version number
#define VK_API_VERSION_1_1 VK_MAKE_API_VERSION(0, 1, 1, 0)// Patch version should
always be set to 0
```

### 45.2.2. Querying Version Support

The version of instance-level functionality can be queried by calling vkEnumerateInstanceVersion.

The version of device-level functionality can be queried by calling vkGetPhysicalDeviceProperties or vkGetPhysicalDeviceProperties2, and is returned in VkPhysicalDeviceProperties::apiVersion, encoded as described in Version Numbers.

## 45.3. Layers

When a layer is enabled, it inserts itself into the call chain for Vulkan commands the layer is interested in. Layers **can** be used for a variety of tasks that extend the base behavior of Vulkan beyond what is required by the specification - such as call logging, tracing, validation, or providing additional extensions.

| | |
|---|---|
| **NOTE** | For example, an implementation is not expected to check that the value of enums used by the application fall within allowed ranges. Instead, a validation layer would do those checks and flag issues. This avoids a performance penalty during |

production use of the application because those layers would not be enabled in production.

**NOTE** Vulkan layers **may** wrap object handles (i.e. return a different handle value to the application than that generated by the implementation). This is generally discouraged, as it increases the probability of incompatibilities with new extensions. The validation layers wrap handles in order to track the proper use and destruction of each object. See the "Architecture of the Vulkan Loader Interfaces" document for additional information.

To query the available layers, call:

```
// Provided by VK_VERSION_1_0
VkResult vkEnumerateInstanceLayerProperties(
    uint32_t*                                   pPropertyCount,
    VkLayerProperties*                          pProperties);
```

- `pPropertyCount` is a pointer to an integer related to the number of layer properties available or queried, as described below.
- `pProperties` is either `NULL` or a pointer to an array of VkLayerProperties structures.

If `pProperties` is `NULL`, then the number of layer properties available is returned in `pPropertyCount`. Otherwise, `pPropertyCount` **must** point to a variable set by the application to the number of elements in the `pProperties` array, and on return the variable is overwritten with the number of structures actually written to `pProperties`. If `pPropertyCount` is less than the number of layer properties available, at most `pPropertyCount` structures will be written, and `VK_INCOMPLETE` will be returned instead of `VK_SUCCESS`, to indicate that not all the available properties were returned.

The list of available layers may change at any time due to actions outside of the Vulkan implementation, so two calls to `vkEnumerateInstanceLayerProperties` with the same parameters **may** return different results, or retrieve different `pPropertyCount` values or `pProperties` contents. Once an instance has been created, the layers enabled for that instance will continue to be enabled and valid for the lifetime of that instance, even if some of them become unavailable for future instances.

### Valid Usage (Implicit)

- VUID-vkEnumerateInstanceLayerProperties-pPropertyCount-parameter
  `pPropertyCount` **must** be a valid pointer to a `uint32_t` value
- VUID-vkEnumerateInstanceLayerProperties-pProperties-parameter
  If the value referenced by `pPropertyCount` is not `0`, and `pProperties` is not `NULL`, `pProperties` **must** be a valid pointer to an array of `pPropertyCount` VkLayerProperties structures

### Return Codes

**Success**

- VK_SUCCESS
- VK_INCOMPLETE

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY

The VkLayerProperties structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkLayerProperties {
    char        layerName[VK_MAX_EXTENSION_NAME_SIZE];
    uint32_t    specVersion;
    uint32_t    implementationVersion;
    char        description[VK_MAX_DESCRIPTION_SIZE];
} VkLayerProperties;
```

- layerName is an array of VK_MAX_EXTENSION_NAME_SIZE char containing a null-terminated UTF-8 string which is the name of the layer. Use this name in the ppEnabledLayerNames array passed in the VkInstanceCreateInfo structure to enable this layer for an instance.
- specVersion is the Vulkan version the layer was written to, encoded as described in Version Numbers.
- implementationVersion is the version of this layer. It is an integer, increasing with backward compatible changes.
- description is an array of VK_MAX_DESCRIPTION_SIZE char containing a null-terminated UTF-8 string which provides additional details that **can** be used by the application to identify the layer.

VK_MAX_EXTENSION_NAME_SIZE is the length in char values of an array containing a layer or extension name string, as returned in VkLayerProperties::layerName, VkExtensionProperties::extensionName, and other queries.

```
#define VK_MAX_EXTENSION_NAME_SIZE        256U
```

VK_MAX_DESCRIPTION_SIZE is the length in char values of an array containing a string with additional descriptive information about a query, as returned in VkLayerProperties::description and other queries.

```
#define VK_MAX_DESCRIPTION_SIZE           256U
```

To enable a layer, the name of the layer **should** be added to the ppEnabledLayerNames member of

VkInstanceCreateInfo when creating a VkInstance.

Loader implementations **may** provide mechanisms outside the Vulkan API for enabling specific layers. Layers enabled through such a mechanism are *implicitly enabled*, while layers enabled by including the layer name in the ppEnabledLayerNames member of VkInstanceCreateInfo are *explicitly enabled*. Implicitly enabled layers are loaded before explicitly enabled layers, such that implicitly enabled layers are closer to the application, and explicitly enabled layers are closer to the driver. Except where otherwise specified, implicitly enabled and explicitly enabled layers differ only in the way they are enabled, and the order in which they are loaded. Explicitly enabling a layer that is implicitly enabled results in this layer being loaded as an implicitly enabled layer; it has no additional effect.

### 45.3.1. Device Layer Deprecation

Previous versions of this specification distinguished between instance and device layers. Instance layers were only able to intercept commands that operate on VkInstance and VkPhysicalDevice, except they were not able to intercept vkCreateDevice. Device layers were enabled for individual devices when they were created, and could only intercept commands operating on that device or its child objects.

Device-only layers are now deprecated, and this specification no longer distinguishes between instance and device layers. Layers are enabled during instance creation, and are able to intercept all commands operating on that instance or any of its child objects. At the time of deprecation there were no known device-only layers and no compelling reason to create one.

In order to maintain compatibility with implementations released prior to device-layer deprecation, applications **should** still enumerate and enable device layers. The behavior of vkEnumerateDeviceLayerProperties and valid usage of the ppEnabledLayerNames member of VkDeviceCreateInfo maximizes compatibility with applications written to work with the previous requirements.

To enumerate device layers, call:

```
// Provided by VK_VERSION_1_0
VkResult vkEnumerateDeviceLayerProperties(
    VkPhysicalDevice                            physicalDevice,
    uint32_t*                                   pPropertyCount,
    VkLayerProperties*                          pProperties);
```

- physicalDevice is the physical device that will be queried.
- pPropertyCount is a pointer to an integer related to the number of layer properties available or queried.
- pProperties is either NULL or a pointer to an array of VkLayerProperties structures.

If pProperties is NULL, then the number of layer properties available is returned in pPropertyCount. Otherwise, pPropertyCount **must** point to a variable set by the application to the number of elements in the pProperties array, and on return the variable is overwritten with the number of structures actually written to pProperties. If pPropertyCount is less than the number of layer properties

available, at most `pPropertyCount` structures will be written, and `VK_INCOMPLETE` will be returned instead of `VK_SUCCESS`, to indicate that not all the available properties were returned.

The list of layers enumerated by `vkEnumerateDeviceLayerProperties` **must** be exactly the sequence of layers enabled for the instance. The members of `VkLayerProperties` for each enumerated layer **must** be the same as the properties when the layer was enumerated by `vkEnumerateInstanceLayerProperties`.

| | |
|---|---|
| **NOTE** | Due to platform details on Android, `vkEnumerateDeviceLayerProperties` may be called with `physicalDevice` equal to `NULL` during layer discovery. This behavior will only be observed by layer implementations, and not the underlying Vulkan driver. |

### Valid Usage (Implicit)

- VUID-vkEnumerateDeviceLayerProperties-physicalDevice-parameter
  `physicalDevice` **must** be a valid VkPhysicalDevice handle
- VUID-vkEnumerateDeviceLayerProperties-pPropertyCount-parameter
  `pPropertyCount` **must** be a valid pointer to a `uint32_t` value
- VUID-vkEnumerateDeviceLayerProperties-pProperties-parameter
  If the value referenced by `pPropertyCount` is not `0`, and `pProperties` is not `NULL`, `pProperties` **must** be a valid pointer to an array of `pPropertyCount` VkLayerProperties structures

### Return Codes

**Success**

- `VK_SUCCESS`
- `VK_INCOMPLETE`

**Failure**

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`

The `ppEnabledLayerNames` and `enabledLayerCount` members of VkDeviceCreateInfo are deprecated and their values **must** be ignored by implementations. However, for compatibility, only an empty list of layers or a list that exactly matches the sequence enabled at instance creation time are valid, and validation layers **should** issue diagnostics for other cases.

Regardless of the enabled layer list provided in VkDeviceCreateInfo, the sequence of layers active for a device will be exactly the sequence of layers enabled when the parent instance was created.

## 45.4. Extensions

Extensions **may** define new Vulkan commands, structures, and enumerants. For compilation purposes, the interfaces defined by registered extensions, including new structures and

enumerants as well as function pointer types for new commands, are defined in the Khronos-supplied vulkan_core.h together with the core API. However, commands defined by extensions **may** not be available for static linking - in which case function pointers to these commands **should** be queried at runtime as described in Command Function Pointers. Extensions **may** be provided by layers as well as by a Vulkan implementation.

Because extensions **may** extend or change the behavior of the Vulkan API, extension authors **should** add support for their extensions to the Khronos validation layers. This is especially important for new commands whose parameters have been wrapped by the validation layers. See the "Architecture of the Vulkan Loader Interfaces" document for additional information.

**NOTE**

To enable an instance extension, the name of the extension **can** be added to the ppEnabledExtensionNames member of VkInstanceCreateInfo when creating a VkInstance.

To enable a device extension, the name of the extension **can** be added to the ppEnabledExtensionNames member of VkDeviceCreateInfo when creating a VkDevice.

Physical-Device-Level functionality does not have any enabling mechanism and **can** be used as long as the VkPhysicalDevice supports the device extension as determined by vkEnumerateDeviceExtensionProperties.

Enabling an extension (with no further use of that extension) does not change the behavior of functionality exposed by the core Vulkan API or any other extension, other than making valid the use of the commands, enums and structures defined by that extension.

Valid Usage sections for individual commands and structures do not currently contain which extensions have to be enabled in order to make their use valid, although they might do so in the future. It is defined only in the Valid Usage for Extensions section.

### 45.4.1. Instance Extensions

Instance extensions add new instance-level functionality to the API, outside of the core specification.

To query the available instance extensions, call:

```
// Provided by VK_VERSION_1_0
VkResult vkEnumerateInstanceExtensionProperties(
    const char*                                 pLayerName,
    uint32_t*                                   pPropertyCount,
    VkExtensionProperties*                      pProperties);
```

- pLayerName is either NULL or a pointer to a null-terminated UTF-8 string naming the layer to retrieve extensions from.
- pPropertyCount is a pointer to an integer related to the number of extension properties available

or queried, as described below.

- `pProperties` is either `NULL` or a pointer to an array of VkExtensionProperties structures.

When `pLayerName` parameter is `NULL`, only extensions provided by the Vulkan implementation or by implicitly enabled layers are returned. When `pLayerName` is the name of a layer, the instance extensions provided by that layer are returned.

If `pProperties` is `NULL`, then the number of extensions properties available is returned in `pPropertyCount`. Otherwise, `pPropertyCount` **must** point to a variable set by the application to the number of elements in the `pProperties` array, and on return the variable is overwritten with the number of structures actually written to `pProperties`. If `pPropertyCount` is less than the number of extension properties available, at most `pPropertyCount` structures will be written, and `VK_INCOMPLETE` will be returned instead of `VK_SUCCESS`, to indicate that not all the available properties were returned.

Because the list of available layers may change externally between calls to vkEnumerateInstanceExtensionProperties, two calls may retrieve different results if a `pLayerName` is available in one call but not in another. The extensions supported by a layer may also change between two calls, e.g. if the layer implementation is replaced by a different version between those calls.

Implementations **must** not advertise any pair of extensions that cannot be enabled together due to behavioral differences, or any extension that cannot be enabled against the advertised version.

## Valid Usage (Implicit)

- VUID-vkEnumerateInstanceExtensionProperties-pLayerName-parameter
  If `pLayerName` is not `NULL`, `pLayerName` **must** be a null-terminated UTF-8 string
- VUID-vkEnumerateInstanceExtensionProperties-pPropertyCount-parameter
  `pPropertyCount` **must** be a valid pointer to a `uint32_t` value
- VUID-vkEnumerateInstanceExtensionProperties-pProperties-parameter
  If the value referenced by `pPropertyCount` is not `0`, and `pProperties` is not `NULL`, `pProperties` **must** be a valid pointer to an array of `pPropertyCount` VkExtensionProperties structures

## Return Codes

### Success

- `VK_SUCCESS`
- `VK_INCOMPLETE`

### Failure

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_LAYER_NOT_PRESENT`

### 45.4.2. Device Extensions

Device extensions add new device-level functionality to the API, outside of the core specification.

To query the extensions available to a given physical device, call:

```
// Provided by VK_VERSION_1_0
VkResult vkEnumerateDeviceExtensionProperties(
    VkPhysicalDevice                            physicalDevice,
    const char*                                 pLayerName,
    uint32_t*                                   pPropertyCount,
    VkExtensionProperties*                      pProperties);
```

- `physicalDevice` is the physical device that will be queried.
- `pLayerName` is either `NULL` or a pointer to a null-terminated UTF-8 string naming the layer to retrieve extensions from.
- `pPropertyCount` is a pointer to an integer related to the number of extension properties available or queried, and is treated in the same fashion as the vkEnumerateInstanceExtensionProperties ::`pPropertyCount` parameter.
- `pProperties` is either `NULL` or a pointer to an array of VkExtensionProperties structures.

When `pLayerName` parameter is `NULL`, only extensions provided by the Vulkan implementation or by implicitly enabled layers are returned. When `pLayerName` is the name of a layer, the device extensions provided by that layer are returned.

Implementations **must** not advertise any pair of extensions that cannot be enabled together due to behavioral differences, or any extension that cannot be enabled against the advertised version.

If `VK_KHR_ray_tracing_pipeline` is advertised as supported by this query, `VK_KHR_pipeline_library` **must** also be supported.

| | |
|---|---|
| **NOTE** | Due to platform details on Android, `vkEnumerateDeviceExtensionProperties` may be called with `physicalDevice` equal to `NULL` during layer discovery. This behavior will only be observed by layer implementations, and not the underlying Vulkan driver. |

#### Valid Usage (Implicit)

- VUID-vkEnumerateDeviceExtensionProperties-physicalDevice-parameter
  `physicalDevice` **must** be a valid VkPhysicalDevice handle
- VUID-vkEnumerateDeviceExtensionProperties-pLayerName-parameter
  If `pLayerName` is not `NULL`, `pLayerName` **must** be a null-terminated UTF-8 string
- VUID-vkEnumerateDeviceExtensionProperties-pPropertyCount-parameter
  `pPropertyCount` **must** be a valid pointer to a `uint32_t` value
- VUID-vkEnumerateDeviceExtensionProperties-pProperties-parameter
  If the value referenced by `pPropertyCount` is not `0`, and `pProperties` is not `NULL`, `pProperties`

**must** be a valid pointer to an array of `pPropertyCount` VkExtensionProperties structures

### Return Codes

**Success**

- `VK_SUCCESS`
- `VK_INCOMPLETE`

**Failure**

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_LAYER_NOT_PRESENT`

The `VkExtensionProperties` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkExtensionProperties {
    char        extensionName[VK_MAX_EXTENSION_NAME_SIZE];
    uint32_t    specVersion;
} VkExtensionProperties;
```

- `extensionName` is an array of `VK_MAX_EXTENSION_NAME_SIZE` `char` containing a null-terminated UTF-8 string which is the name of the extension.
- `specVersion` is the version of this extension. It is an integer, incremented with backward compatible changes.

#### Accessing Device-Level Functionality From a VkPhysicalDevice

Some device extensions also add support for physical-device-level functionality. Physical-device-level functionality **can** be used, if the required extension is supported as advertised by vkEnumerateDeviceExtensionProperties for a given VkPhysicalDevice.

#### Accessing Device-Level Functionality From a VkDevice

For commands that are dispatched from a VkDevice, or from a child object of a VkDevice, device extensions **must** be enabled in vkCreateDevice.

## 45.5. Extension Dependencies

Some extensions are dependent on other extensions, or on specific core API versions, to function. To enable extensions with dependencies, any *required extensions* **must** also be enabled through the same API mechanisms when creating an instance with vkCreateInstance or a device with vkCreateDevice. Each extension which has such dependencies documents them in the appendix summarizing that extension.

If an extension is supported (as queried by vkEnumerateInstanceExtensionProperties or vkEnumerateDeviceExtensionProperties), then *required extensions* of that extension **must** also be supported for the same instance or physical device.

Any device extension that has an instance extension dependency that is not enabled by vkCreateInstance is considered to be unsupported, hence it **must** not be returned by vkEnumerateDeviceExtensionProperties for any VkPhysicalDevice child of the instance. Instance extensions do not have dependencies on device extensions.

If a required extension has been promoted to another extension or to a core API version, then as a *general* rule, the dependency is also satisfied by the promoted extension or core version. This will be true so long as any features required by the original extension are also required or enabled by the promoted extension or core version. However, in some cases an extension is promoted while making some of its features optional in the promoted extension or core version. In this case, the dependency **may** not be satisfied. The only way to be certain is to look at the descriptions of the original dependency and the promoted version in the Layers & Extensions and Core Revisions appendices.

**NOTE** There is metadata in `vk.xml` describing some aspects of promotion, especially `requires`, `promotedto` and `deprecatedby` attributes of `<extension>` tags. However, the metadata does not yet fully describe this scenario. In the future, we may extend the XML schema to describe the full set of extensions and versions satisfying a dependency. As discussed in more detail for Promotion below, when an extension is promoted it does not mean that a mechanical substitution of an extension API by the corresponding promoted API will work in exactly the same fashion; be supported at runtime; or even exist.

# 45.6. Compatibility Guarantees (Informative)

This section is marked as informal as there is no binding responsibility on implementations of the Vulkan API - these guarantees are however a contract between the Vulkan Working Group and developers using this Specification.

## 45.6.1. Core Versions

Each of the major, minor, and patch versions of the Vulkan specification provide different compatibility guarantees.

### Patch Versions

A difference in the patch version indicates that a set of bug fixes or clarifications have been made to the Specification. Informative enums returned by Vulkan commands that will not affect the runtime behavior of a valid application may be added in a patch version (e.g. VkVendorId).

The specification’s patch version is strictly increasing for a given major version of the specification; any change to a specification as described above will result in the patch version being increased by 1. Patch versions are applied to all minor versions, even if a given minor version is not affected by the provoking change.

Specifications with different patch versions but the same major and minor version are *fully compatible* with each other - such that a valid application written against one will work with an implementation of another.

| | |
|---|---|
| **NOTE** | If a patch version includes a bug fix or clarification that could have a significant impact on developer expectations, these will be highlighted in the change log. Generally the Vulkan Working Group tries to avoid these kinds of changes, instead fixing them in either an extension or core version. |

**Minor Versions**

Changes in the minor version of the specification indicate that new functionality has been added to the core specification. This will usually include new interfaces in the header, and **may** also include behavior changes and bug fixes. Core functionality **may** be deprecated in a minor version, but will not be obsoleted or removed.

The specification's minor version is strictly increasing for a given major version of the specification; any change to a specification as described above will result in the minor version being increased by 1. Changes that can be accommodated in a patch version will not increase the minor version.

Specifications with a lower minor version are *backwards compatible* with an implementation of a specification with a higher minor version for core functionality and extensions issued with the KHR vendor tag. Vendor and multi-vendor extensions are not guaranteed to remain functional across minor versions, though in general they are with few exceptions - see Obsoletion for more information.

**Major Versions**

A difference in the major version of specifications indicates a large set of changes which will likely include interface changes, behavioral changes, removal of deprecated functionality, and the modification, addition, or replacement of other functionality.

The specification's major version is monotonically increasing; any change to the specification as described above will result in the major version being increased. Changes that can be accommodated in a patch or minor version will not increase the major version.

The Vulkan Working Group intends to only issue a new major version of the Specification in order to realize significant improvements to the Vulkan API that will necessarily require breaking compatibility.

A new major version will likely include a wholly new version of the specification to be issued - which could include an overhaul of the versioning semantics for the minor and patch versions. The patch and minor versions of a specification are therefore not meaningful across major versions. If a major version of the specification includes similar versioning semantics, it is expected that the patch and the minor version will be reset to 0 for that major version.

### 45.6.2. Extensions

A KHR extension **must** be able to be enabled alongside any other KHR extension, and for any minor or patch version of the core Specification beyond the minimum version it requires. A multi-vendor extension **should** be able to be enabled alongside any KHR extension or other multi-vendor extension, and for any minor or patch version of the core Specification beyond the minimum version it requires. A vendor extension **should** be able to be enabled alongside any KHR extension, multi-vendor extension, or other vendor extension from the same vendor, and for any minor or patch version of the core Specification beyond the minimum version it requires. A vendor extension **may** be able to be enabled alongside vendor extensions from another vendor.

The one other exception to this is if a vendor or multi-vendor extension is made obsolete by either a core version or another extension, which will be highlighted in the extension appendix.

#### Promotion

Extensions, or features of an extension, **may** be promoted to a new core version of the API, or a newer extension which an equal or greater number of implementors are in favor of.

When extension functionality is promoted, minor changes **may** be introduced, limited to the following:

- Naming
- Non-intrusive parameter changes
- Feature advertisement/enablement
- Combining structure parameters into larger structures
- Author ID suffixes changed or removed

**NOTE**

If extension functionality is promoted, there is no guarantee of direct compatibility, however it should require little effort to port code from the original feature to the promoted one.

The Vulkan Working Group endeavors to ensure that larger changes are marked as either deprecated or obsoleted as appropriate, and can do so retroactively if necessary.

Extensions that are promoted are listed as being promoted in their extension appendices, with reference to where they were promoted to.

When an extension is promoted, any backwards compatibility aliases which exist in the extension will **not** be promoted.

**NOTE**

As a hypothetical example, if the `VK_KHR_surface` extension were promoted to part of a future core version, the `VK_COLOR_SPACE_SRGB_NONLINEAR_KHR` token defined by that extension would be promoted to `VK_COLOR_SPACE_SRGB_NONLINEAR`. However, the `VK_COLORSPACE_SRGB_NONLINEAR_KHR` token aliases `VK_COLOR_SPACE_SRGB_NONLINEAR_KHR`. The `VK_COLORSPACE_SRGB_NONLINEAR_KHR` would not be promoted, because it is a backwards compatibility alias that exists only due to a naming mistake when the

extension was initially published.

### Deprecation

Extensions **may** be marked as deprecated when the intended use cases either become irrelevant or can be solved in other ways. Generally, a new feature will become available to solve the use case in another extension or core version of the API, but it is not guaranteed.

**NOTE** Features that are intended to replace deprecated functionality have no guarantees of compatibility, and applications may require drastic modification in order to make use of the new features.

Extensions that are deprecated are listed as being deprecated in their extension appendices, with an explanation of the deprecation and any features that are relevant.

### Obsoletion

Occasionally, an extension will be marked as obsolete if a new version of the core API or a new extension is fundamentally incompatible with it. An obsoleted extension **must** not be used with the extension or core version that obsoleted it.

Extensions that are obsoleted are listed as being obsoleted in their extension appendices, with reference to what they were obsoleted by.

### Aliases

When an extension is promoted or deprecated by a newer feature, some or all of its functionality **may** be replicated into the newer feature. Rather than duplication of all the documentation and definitions, the specification instead identifies the identical commands and types as *aliases* of one another. Each alias is mentioned together with the definition it aliases, with the older aliases marked as “equivalents”. Each alias of the same command has identical behavior, and each alias of the same type has identical meaning - they can be used interchangeably in an application with no compatibility issues.

**NOTE** For promoted types, the aliased extension type is semantically identical to the new core type. The C99 headers simply `typedef` the older aliases to the promoted types.

For promoted command aliases, however, there are two separate command definitions, due to the fact that the C99 ABI has no way to alias command definitions without resorting to macros. Calling either command will produce identical behavior within the bounds of the specification, and should still invoke the same path in the implementation. Debug tools may use separate commands with different debug behavior; to write the appropriate command name to an output log, for instance.

### Special Use Extensions

Some extensions exist only to support a specific purpose or specific class of application. These are referred to as “special use extensions”. Use of these extensions in applications not meeting the

special use criteria is not recommended.

Special use cases are restricted, and only those defined below are used to describe extensions:

*Table 61. Extension Special Use Cases*

| Special Use | XML Tag | Full Description |
|---|---|---|
| CAD support | cadsupport | Extension is intended to support specialized functionality used by CAD/CAM applications. |
| D3D support | d3demulation | Extension is intended to support D3D emulation layers, and applications ported from D3D, by adding functionality specific to D3D. |
| Developer tools | devtools | Extension is intended to support developer tools such as capture-replay libraries. |
| Debugging tools | debugging | Extension is intended for use by applications when debugging. |
| OpenGL / ES support | glemulation | Extension is intended to support OpenGL and/or OpenGL ES emulation layers, and applications ported from those APIs, by adding functionality specific to those APIs. |

Special use extensions are identified in the metadata for each such extension in the Layers & Extensions appendix, using the name in the “Special Use” column above.

Special use extensions are also identified in `vk.xml` with the short name in “XML Tag” column above, as described in the “API Extensions (`extension` tag)” section of the registry schema documentation.

# Chapter 46. Features

*Features* describe functionality which is not supported on all implementations. Features are properties of the physical device. Features are **optional**, and **must** be explicitly enabled before use. Support for features is reported and enabled on a per-feature basis.

**NOTE** Features are reported via the basic VkPhysicalDeviceFeatures structure, as well as the extensible structure `VkPhysicalDeviceFeatures2`, which was added in the `VK_KHR_get_physical_device_properties2` extension and included in Vulkan 1.1. When new features are added in future Vulkan versions or extensions, each extension **should** introduce one new feature structure, if needed. This structure **can** be added to the `pNext` chain of the `VkPhysicalDeviceFeatures2` structure.

For convenience, new core versions of Vulkan **may** introduce new unified feature structures for features promoted from extensions. At the same time, the extension's original feature structure (if any) is also promoted to the core API, and is an alias of the extension's structure. This results in multiple names for the same feature: in the original extension's feature structure and the promoted structure alias, in the unified feature structure. When a feature was implicitly supported and enabled in the extension, but an explicit name was added during promotion, then the extension itself acts as an alias for the feature as listed in the table below.

All aliases of the same feature in the core API **must** be reported consistently: either all **must** be reported as supported, or none of them. When a promoted extension is available, any corresponding feature aliases **must** be supported.

*Table 62. Extension Feature Aliases*

| Extension | Feature(s) |
|---|---|

To query supported features, call:

```
// Provided by VK_VERSION_1_0
void vkGetPhysicalDeviceFeatures(
    VkPhysicalDevice                            physicalDevice,
    VkPhysicalDeviceFeatures*                   pFeatures);
```

- `physicalDevice` is the physical device from which to query the supported features.
- `pFeatures` is a pointer to a VkPhysicalDeviceFeatures structure in which the physical device features are returned. For each feature, a value of `VK_TRUE` specifies that the feature is supported on this physical device, and `VK_FALSE` specifies that the feature is not supported.

### Valid Usage (Implicit)

- VUID-vkGetPhysicalDeviceFeatures-physicalDevice-parameter
  `physicalDevice` **must** be a valid VkPhysicalDevice handle
- VUID-vkGetPhysicalDeviceFeatures-pFeatures-parameter

pFeatures **must** be a valid pointer to a VkPhysicalDeviceFeatures structure

Fine-grained features used by a logical device **must** be enabled at VkDevice creation time. If a feature is enabled that the physical device does not support, VkDevice creation will fail and return VK_ERROR_FEATURE_NOT_PRESENT.

The fine-grained features are enabled by passing a pointer to the VkPhysicalDeviceFeatures structure via the pEnabledFeatures member of the VkDeviceCreateInfo structure that is passed into the vkCreateDevice call. If a member of pEnabledFeatures is set to VK_TRUE or VK_FALSE, then the device will be created with the indicated feature enabled or disabled, respectively. Features **can** also be enabled by using the VkPhysicalDeviceFeatures2 structure.

If an application wishes to enable all features supported by a device, it **can** simply pass in the VkPhysicalDeviceFeatures structure that was previously returned by vkGetPhysicalDeviceFeatures. To disable an individual feature, the application **can** set the desired member to VK_FALSE in the same structure. Setting pEnabledFeatures to NULL and not including a VkPhysicalDeviceFeatures2 in the pNext chain of VkDeviceCreateInfo is equivalent to setting all members of the structure to VK_FALSE.

| | |
|---|---|
| **NOTE** | Some features, such as robustBufferAccess, **may** incur a runtime performance cost. Application writers **should** carefully consider the implications of enabling all supported features. |

To query supported features defined by the core or extensions, call:

```
// Provided by VK_VERSION_1_1
void vkGetPhysicalDeviceFeatures2(
    VkPhysicalDevice                            physicalDevice,
    VkPhysicalDeviceFeatures2*                  pFeatures);
```

or the equivalent command

```
// Provided by VK_KHR_get_physical_device_properties2
void vkGetPhysicalDeviceFeatures2KHR(
    VkPhysicalDevice                            physicalDevice,
    VkPhysicalDeviceFeatures2*                  pFeatures);
```

- physicalDevice is the physical device from which to query the supported features.
- pFeatures is a pointer to a VkPhysicalDeviceFeatures2 structure in which the physical device features are returned.

Each structure in pFeatures and its pNext chain contains members corresponding to fine-grained features. vkGetPhysicalDeviceFeatures2 writes each member to a boolean value indicating whether that feature is supported.

### Valid Usage (Implicit)

- VUID-vkGetPhysicalDeviceFeatures2-physicalDevice-parameter
  `physicalDevice` **must** be a valid VkPhysicalDevice handle
- VUID-vkGetPhysicalDeviceFeatures2-pFeatures-parameter
  `pFeatures` **must** be a valid pointer to a VkPhysicalDeviceFeatures2 structure

The `VkPhysicalDeviceFeatures2` structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkPhysicalDeviceFeatures2 {
    VkStructureType             sType;
    void*                       pNext;
    VkPhysicalDeviceFeatures    features;
} VkPhysicalDeviceFeatures2;
```

or the equivalent

```
// Provided by VK_KHR_get_physical_device_properties2
typedef VkPhysicalDeviceFeatures2 VkPhysicalDeviceFeatures2KHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `features` is a VkPhysicalDeviceFeatures structure describing the fine-grained features of the Vulkan 1.0 API.

The `pNext` chain of this structure is used to extend the structure with features defined by extensions. This structure **can** be used in vkGetPhysicalDeviceFeatures2 or **can** be included in the `pNext` chain of a VkDeviceCreateInfo structure, in which case it controls which features are enabled on the device in lieu of `pEnabledFeatures`.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceFeatures2-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_FEATURES_2`

The `VkPhysicalDeviceFeatures` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkPhysicalDeviceFeatures {
    VkBool32    robustBufferAccess;
    VkBool32    fullDrawIndexUint32;
    VkBool32    imageCubeArray;
```

```
    VkBool32    independentBlend;
    VkBool32    geometryShader;
    VkBool32    tessellationShader;
    VkBool32    sampleRateShading;
    VkBool32    dualSrcBlend;
    VkBool32    logicOp;
    VkBool32    multiDrawIndirect;
    VkBool32    drawIndirectFirstInstance;
    VkBool32    depthClamp;
    VkBool32    depthBiasClamp;
    VkBool32    fillModeNonSolid;
    VkBool32    depthBounds;
    VkBool32    wideLines;
    VkBool32    largePoints;
    VkBool32    alphaToOne;
    VkBool32    multiViewport;
    VkBool32    samplerAnisotropy;
    VkBool32    textureCompressionETC2;
    VkBool32    textureCompressionASTC_LDR;
    VkBool32    textureCompressionBC;
    VkBool32    occlusionQueryPrecise;
    VkBool32    pipelineStatisticsQuery;
    VkBool32    vertexPipelineStoresAndAtomics;
    VkBool32    fragmentStoresAndAtomics;
    VkBool32    shaderTessellationAndGeometryPointSize;
    VkBool32    shaderImageGatherExtended;
    VkBool32    shaderStorageImageExtendedFormats;
    VkBool32    shaderStorageImageMultisample;
    VkBool32    shaderStorageImageReadWithoutFormat;
    VkBool32    shaderStorageImageWriteWithoutFormat;
    VkBool32    shaderUniformBufferArrayDynamicIndexing;
    VkBool32    shaderSampledImageArrayDynamicIndexing;
    VkBool32    shaderStorageBufferArrayDynamicIndexing;
    VkBool32    shaderStorageImageArrayDynamicIndexing;
    VkBool32    shaderClipDistance;
    VkBool32    shaderCullDistance;
    VkBool32    shaderFloat64;
    VkBool32    shaderInt64;
    VkBool32    shaderInt16;
    VkBool32    shaderResourceResidency;
    VkBool32    shaderResourceMinLod;
    VkBool32    sparseBinding;
    VkBool32    sparseResidencyBuffer;
    VkBool32    sparseResidencyImage2D;
    VkBool32    sparseResidencyImage3D;
    VkBool32    sparseResidency2Samples;
    VkBool32    sparseResidency4Samples;
    VkBool32    sparseResidency8Samples;
    VkBool32    sparseResidency16Samples;
    VkBool32    sparseResidencyAliased;
    VkBool32    variableMultisampleRate;
```

```
    VkBool32    inheritedQueries;
} VkPhysicalDeviceFeatures;
```

This structure describes the following features:

- `robustBufferAccess` specifies that accesses to buffers are bounds-checked against the range of the buffer descriptor (as determined by `VkDescriptorBufferInfo::range`, VkBufferViewCreateInfo::`range`, or the size of the buffer). Out of bounds accesses **must** not cause application termination, and the effects of shader loads, stores, and atomics **must** conform to an implementation-dependent behavior as described below.
  - A buffer access is considered to be out of bounds if any of the following are true:
    - The pointer was formed by `OpImageTexelPointer` and the coordinate is less than zero or greater than or equal to the number of whole elements in the bound range.
    - The pointer was not formed by `OpImageTexelPointer` and the object pointed to is not wholly contained within the bound range. This includes accesses performed via *variable pointers* where the buffer descriptor being accessed cannot be statically determined. Uninitialized pointers and pointers equal to `OpConstantNull` are treated as pointing to a zero-sized object, so all accesses through such pointers are considered to be out of bounds. Buffer accesses through buffer device addresses are not bounds-checked.
    - If the `VkPhysicalDeviceCooperativeMatrixFeaturesNV::cooperativeMatrixRobustBufferAccess` feature is not enabled, then accesses using `OpCooperativeMatrixLoadNV` and `OpCooperativeMatrixStoreNV` **may** not be bounds-checked.
    - If the `VkPhysicalDeviceCooperativeMatrixFeaturesKHR::cooperativeMatrixRobustBufferAccess` feature is not enabled, then accesses using `OpCooperativeMatrixLoadKHR` and `OpCooperativeMatrixStoreKHR` **may** not be bounds-checked.

      **NOTE** If a SPIR-V `OpLoad` instruction loads a structure and the tail end of the structure is out of bounds, then all members of the structure are considered out of bounds even if the members at the end are not statically used.

    - If `robustBufferAccess2` is not enabled and any buffer access is determined to be out of bounds, then any other access of the same type (load, store, or atomic) to the same buffer that accesses an address less than 16 bytes away from the out of bounds address **may** also be considered out of bounds.
    - If the access is a load that reads from the same memory locations as a prior store in the same shader invocation, with no other intervening accesses to the same memory locations in that shader invocation, then the result of the load **may** be the value stored by the store instruction, even if the access is out of bounds. If the load is `Volatile`, then an out of bounds load **must** return the appropriate out of bounds value.
  - Accesses to descriptors written with a VK_NULL_HANDLE resource or view are not considered to be out of bounds. Instead, each type of descriptor access defines a specific behavior for accesses to a null descriptor.

- Out-of-bounds buffer loads will return any of the following values:
  - If the access is to a uniform buffer and `robustBufferAccess2` is enabled, loads of offsets between the end of the descriptor range and the end of the descriptor range rounded up to a multiple of `robustUniformBufferAccessSizeAlignment` bytes **must** return either zero values or the contents of the memory at the offset being loaded. Loads of offsets past the descriptor range rounded up to a multiple of `robustUniformBufferAccessSizeAlignment` bytes **must** return zero values.
  - If the access is to a storage buffer and `robustBufferAccess2` is enabled, loads of offsets between the end of the descriptor range and the end of the descriptor range rounded up to a multiple of `robustStorageBufferAccessSizeAlignment` bytes **must** return either zero values or the contents of the memory at the offset being loaded. Loads of offsets past the descriptor range rounded up to a multiple of `robustStorageBufferAccessSizeAlignment` bytes **must** return zero values. Similarly, stores to addresses between the end of the descriptor range and the end of the descriptor range rounded up to a multiple of `robustStorageBufferAccessSizeAlignment` bytes **may** be discarded.
  - Non-atomic accesses to storage buffers that are a multiple of 32 bits **may** be decomposed into 32-bit accesses that are individually bounds-checked.
  - If the access is to an index buffer and `robustBufferAccess2` is enabled, zero values **must** be returned.
  - If the access is to a uniform texel buffer or storage texel buffer and `robustBufferAccess2` is enabled, zero values **must** be returned, and then Conversion to RGBA is applied based on the buffer view's format.
  - Values from anywhere within the memory range(s) bound to the buffer (possibly including bytes of memory past the end of the buffer, up to the end of the bound range).
  - Zero values, or (0,0,0,x) vectors for vector reads where x is a valid value represented in the type of the vector components and **may** be any of:
    - 0, 1, or the maximum representable positive integer value, for signed or unsigned integer components
    - 0.0 or 1.0, for floating-point components
- Out-of-bounds writes **may** modify values within the memory range(s) bound to the buffer, but **must** not modify any other memory.
  - If `robustBufferAccess2` is enabled, out of bounds writes **must** not modify any memory.
- Out-of-bounds atomics **may** modify values within the memory range(s) bound to the buffer, but **must** not modify any other memory, and return an undefined value.
  - If `robustBufferAccess2` is enabled, out of bounds atomics **must** not modify any memory, and return an undefined value.
- If `robustBufferAccess2` is disabled, vertex input attributes are considered out of bounds if the offset of the attribute in the bound vertex buffer range plus the size of the attribute is greater than either:
  - `vertexBufferRangeSize`, if `bindingStride` == 0; or
  - (`vertexBufferRangeSize` - (`vertexBufferRangeSize` % `bindingStride`))

where vertexBufferRangeSize is the byte size of the memory range bound to the vertex buffer binding and bindingStride is the byte stride of the corresponding vertex input binding. Further, if any vertex input attribute using a specific vertex input binding is out of bounds, then all vertex input attributes using that vertex input binding for that vertex shader invocation are considered out of bounds.

    - If a vertex input attribute is out of bounds, it will be assigned one of the following values:
      - Values from anywhere within the memory range(s) bound to the buffer, converted according to the format of the attribute.
      - Zero values, format converted according to the format of the attribute.
      - Zero values, or (0,0,0,x) vectors, as described above.
  - If robustBufferAccess2 is enabled, vertex input attributes are considered out of bounds if the offset of the attribute in the bound vertex buffer range plus the size of the attribute is greater than the byte size of the memory range bound to the vertex buffer binding.
    - If a vertex input attribute is out of bounds, the raw data extracted are zero values, and missing G, B, or A components are filled with (0,0,1).
  - If robustBufferAccess is not enabled, applications **must** not perform out of bounds accesses except under the conditions enabled by the pipelineRobustness feature .

- fullDrawIndexUint32 specifies the full 32-bit range of indices is supported for indexed draw calls when using a VkIndexType of VK_INDEX_TYPE_UINT32. maxDrawIndexedIndexValue is the maximum index value that **may** be used (aside from the primitive restart index, which is always $2^{32}$-1 when the VkIndexType is VK_INDEX_TYPE_UINT32). If this feature is supported, maxDrawIndexedIndexValue **must** be $2^{32}$-1; otherwise it **must** be no smaller than $2^{24}$-1. See maxDrawIndexedIndexValue.
- imageCubeArray specifies whether image views with a VkImageViewType of VK_IMAGE_VIEW_TYPE_CUBE_ARRAY **can** be created, and that the corresponding SampledCubeArray and ImageCubeArray SPIR-V capabilities **can** be used in shader code.
- independentBlend specifies whether the VkPipelineColorBlendAttachmentState settings are controlled independently per-attachment. If this feature is not enabled, the VkPipelineColorBlendAttachmentState settings for all color attachments **must** be identical. Otherwise, a different VkPipelineColorBlendAttachmentState **can** be provided for each bound color attachment.
- geometryShader specifies whether geometry shaders are supported. If this feature is not enabled, the VK_SHADER_STAGE_GEOMETRY_BIT and VK_PIPELINE_STAGE_GEOMETRY_SHADER_BIT enum values **must** not be used. This also specifies whether shader modules **can** declare the Geometry capability.
- tessellationShader specifies whether tessellation control and evaluation shaders are supported. If this feature is not enabled, the VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT, VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT, VK_PIPELINE_STAGE_TESSELLATION_CONTROL_SHADER_BIT, VK_PIPELINE_STAGE_TESSELLATION_EVALUATION_SHADER_BIT, and VK_STRUCTURE_TYPE_PIPELINE_TESSELLATION_STATE_CREATE_INFO enum values **must** not be used. This also specifies whether shader modules **can** declare the Tessellation capability.

- sampleRateShading specifies whether Sample Shading and multisample interpolation are supported. If this feature is not enabled, the sampleShadingEnable member of the VkPipelineMultisampleStateCreateInfo structure **must** be set to VK_FALSE and the minSampleShading member is ignored. This also specifies whether shader modules **can** declare the SampleRateShading capability.
- dualSrcBlend specifies whether blend operations which take two sources are supported. If this feature is not enabled, the VK_BLEND_FACTOR_SRC1_COLOR, VK_BLEND_FACTOR_ONE_MINUS_SRC1_COLOR, VK_BLEND_FACTOR_SRC1_ALPHA, and VK_BLEND_FACTOR_ONE_MINUS_SRC1_ALPHA enum values **must** not be used as source or destination blending factors. See Dual-Source Blending.
- logicOp specifies whether logic operations are supported. If this feature is not enabled, the logicOpEnable member of the VkPipelineColorBlendStateCreateInfo structure **must** be set to VK_FALSE, and the logicOp member is ignored.
- multiDrawIndirect specifies whether multiple draw indirect is supported. If this feature is not enabled, the drawCount parameter to the vkCmdDrawIndirect and vkCmdDrawIndexedIndirect commands **must** be 0 or 1. The maxDrawIndirectCount member of the VkPhysicalDeviceLimits structure **must** also be 1 if this feature is not supported. See maxDrawIndirectCount.
- drawIndirectFirstInstance specifies whether indirect drawing calls support the firstInstance parameter. If this feature is not enabled, the firstInstance member of all VkDrawIndirectCommand and VkDrawIndexedIndirectCommand structures that are provided to the vkCmdDrawIndirect and vkCmdDrawIndexedIndirect commands **must** be 0.
- depthClamp specifies whether depth clamping is supported. If this feature is not enabled, the depthClampEnable member of the VkPipelineRasterizationStateCreateInfo structure **must** be VK_FALSE. Otherwise, setting depthClampEnable to VK_TRUE will enable depth clamping.
- depthBiasClamp specifies whether depth bias clamping is supported. If this feature is not enabled, the depthBiasClamp member of the VkPipelineRasterizationStateCreateInfo structure **must** be set to 0.0 unless the VK_DYNAMIC_STATE_DEPTH_BIAS dynamic state is enabled, in which case the depthBiasClamp parameter to vkCmdSetDepthBias **must** be set to 0.0.
- fillModeNonSolid specifies whether point and wireframe fill modes are supported. If this feature is not enabled, the VK_POLYGON_MODE_POINT and VK_POLYGON_MODE_LINE enum values **must** not be used.
- depthBounds specifies whether depth bounds tests are supported. If this feature is not enabled, the depthBoundsTestEnable member of the VkPipelineDepthStencilStateCreateInfo structure **must** be VK_FALSE unless the VK_DYNAMIC_STATE_DEPTH_BOUNDS_TEST_ENABLE dynamic state is enabled, in which case the depthBoundsTestEnable parameter to vkCmdSetDepthBoundsTestEnableEXT **must** be VK_FALSE. When depthBoundsTestEnable is VK_FALSE, the minDepthBounds and maxDepthBounds members of the VkPipelineDepthStencilStateCreateInfo structure are ignored.
- wideLines specifies whether lines with width other than 1.0 are supported. If this feature is not enabled, the lineWidth member of the VkPipelineRasterizationStateCreateInfo structure **must** be set to 1.0 unless the VK_DYNAMIC_STATE_LINE_WIDTH dynamic state is enabled, in which case the lineWidth parameter to vkCmdSetLineWidth **must** be set to 1.0. When this feature is supported, the range and granularity of supported line widths are indicated by the lineWidthRange and lineWidthGranularity members of the VkPhysicalDeviceLimits structure, respectively.
- largePoints specifies whether points with size greater than 1.0 are supported. If this feature is

not enabled, only a point size of 1.0 written by a shader is supported. The range and granularity of supported point sizes are indicated by the `pointSizeRange` and `pointSizeGranularity` members of the `VkPhysicalDeviceLimits` structure, respectively.

- `alphaToOne` specifies whether the implementation is able to replace the alpha value of the fragment shader color output in the Multisample Coverage fragment operation. If this feature is not enabled, then the `alphaToOneEnable` member of the VkPipelineMultisampleStateCreateInfo structure **must** be `VK_FALSE`. Otherwise setting `alphaToOneEnable` to `VK_TRUE` will enable alpha-to-one behavior.
- `multiViewport` specifies whether more than one viewport is supported. If this feature is not enabled:
  - The `viewportCount` and `scissorCount` members of the VkPipelineViewportStateCreateInfo structure **must** be set to 1.
  - The `firstViewport` and `viewportCount` parameters to the vkCmdSetViewport command **must** be set to 0 and 1, respectively.
  - The `firstScissor` and `scissorCount` parameters to the vkCmdSetScissor command **must** be set to 0 and 1, respectively.
  - The `exclusiveScissorCount` member of the VkPipelineViewportExclusiveScissorStateCreateInfoNV structure **must** be set to 0 or 1.
  - The `firstExclusiveScissor` and `exclusiveScissorCount` parameters to the vkCmdSetExclusiveScissorNV command **must** be set to 0 and 1, respectively.
- `samplerAnisotropy` specifies whether anisotropic filtering is supported. If this feature is not enabled, the `anisotropyEnable` member of the VkSamplerCreateInfo structure **must** be `VK_FALSE`.
- `textureCompressionETC2` specifies whether all of the ETC2 and EAC compressed texture formats are supported. If this feature is enabled, then the `VK_FORMAT_FEATURE_SAMPLED_IMAGE_BIT`, `VK_FORMAT_FEATURE_BLIT_SRC_BIT` and `VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT` features **must** be supported in `optimalTilingFeatures` for the following formats:
  - `VK_FORMAT_ETC2_R8G8B8_UNORM_BLOCK`
  - `VK_FORMAT_ETC2_R8G8B8_SRGB_BLOCK`
  - `VK_FORMAT_ETC2_R8G8B8A1_UNORM_BLOCK`
  - `VK_FORMAT_ETC2_R8G8B8A1_SRGB_BLOCK`
  - `VK_FORMAT_ETC2_R8G8B8A8_UNORM_BLOCK`
  - `VK_FORMAT_ETC2_R8G8B8A8_SRGB_BLOCK`
  - `VK_FORMAT_EAC_R11_UNORM_BLOCK`
  - `VK_FORMAT_EAC_R11_SNORM_BLOCK`
  - `VK_FORMAT_EAC_R11G11_UNORM_BLOCK`
  - `VK_FORMAT_EAC_R11G11_SNORM_BLOCK`

  To query for additional properties, or if the feature is not enabled, vkGetPhysicalDeviceFormatProperties and vkGetPhysicalDeviceImageFormatProperties **can** be used to check for supported properties of individual formats as normal.

- `textureCompressionASTC_LDR` specifies whether all of the ASTC LDR compressed texture formats are supported. If this feature is enabled, then the `VK_FORMAT_FEATURE_SAMPLED_IMAGE_BIT`, `VK_FORMAT_FEATURE_BLIT_SRC_BIT` and `VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT` features **must** be supported in `optimalTilingFeatures` for the following formats:
  - `VK_FORMAT_ASTC_4x4_UNORM_BLOCK`
  - `VK_FORMAT_ASTC_4x4_SRGB_BLOCK`
  - `VK_FORMAT_ASTC_5x4_UNORM_BLOCK`
  - `VK_FORMAT_ASTC_5x4_SRGB_BLOCK`
  - `VK_FORMAT_ASTC_5x5_UNORM_BLOCK`
  - `VK_FORMAT_ASTC_5x5_SRGB_BLOCK`
  - `VK_FORMAT_ASTC_6x5_UNORM_BLOCK`
  - `VK_FORMAT_ASTC_6x5_SRGB_BLOCK`
  - `VK_FORMAT_ASTC_6x6_UNORM_BLOCK`
  - `VK_FORMAT_ASTC_6x6_SRGB_BLOCK`
  - `VK_FORMAT_ASTC_8x5_UNORM_BLOCK`
  - `VK_FORMAT_ASTC_8x5_SRGB_BLOCK`
  - `VK_FORMAT_ASTC_8x6_UNORM_BLOCK`
  - `VK_FORMAT_ASTC_8x6_SRGB_BLOCK`
  - `VK_FORMAT_ASTC_8x8_UNORM_BLOCK`
  - `VK_FORMAT_ASTC_8x8_SRGB_BLOCK`
  - `VK_FORMAT_ASTC_10x5_UNORM_BLOCK`
  - `VK_FORMAT_ASTC_10x5_SRGB_BLOCK`
  - `VK_FORMAT_ASTC_10x6_UNORM_BLOCK`
  - `VK_FORMAT_ASTC_10x6_SRGB_BLOCK`
  - `VK_FORMAT_ASTC_10x8_UNORM_BLOCK`
  - `VK_FORMAT_ASTC_10x8_SRGB_BLOCK`
  - `VK_FORMAT_ASTC_10x10_UNORM_BLOCK`
  - `VK_FORMAT_ASTC_10x10_SRGB_BLOCK`
  - `VK_FORMAT_ASTC_12x10_UNORM_BLOCK`
  - `VK_FORMAT_ASTC_12x10_SRGB_BLOCK`
  - `VK_FORMAT_ASTC_12x12_UNORM_BLOCK`
  - `VK_FORMAT_ASTC_12x12_SRGB_BLOCK`

  To query for additional properties, or if the feature is not enabled, vkGetPhysicalDeviceFormatProperties and vkGetPhysicalDeviceImageFormatProperties **can** be used to check for supported properties of individual formats as normal.

- textureCompressionBC specifies whether all of the BC compressed texture formats are supported. If this feature is enabled, then the VK_FORMAT_FEATURE_SAMPLED_IMAGE_BIT, VK_FORMAT_FEATURE_BLIT_SRC_BIT and VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT features **must** be supported in optimalTilingFeatures for the following formats:
  - VK_FORMAT_BC1_RGB_UNORM_BLOCK
  - VK_FORMAT_BC1_RGB_SRGB_BLOCK
  - VK_FORMAT_BC1_RGBA_UNORM_BLOCK
  - VK_FORMAT_BC1_RGBA_SRGB_BLOCK
  - VK_FORMAT_BC2_UNORM_BLOCK
  - VK_FORMAT_BC2_SRGB_BLOCK
  - VK_FORMAT_BC3_UNORM_BLOCK
  - VK_FORMAT_BC3_SRGB_BLOCK
  - VK_FORMAT_BC4_UNORM_BLOCK
  - VK_FORMAT_BC4_SNORM_BLOCK
  - VK_FORMAT_BC5_UNORM_BLOCK
  - VK_FORMAT_BC5_SNORM_BLOCK
  - VK_FORMAT_BC6H_UFLOAT_BLOCK
  - VK_FORMAT_BC6H_SFLOAT_BLOCK
  - VK_FORMAT_BC7_UNORM_BLOCK
  - VK_FORMAT_BC7_SRGB_BLOCK

  To query for additional properties, or if the feature is not enabled, vkGetPhysicalDeviceFormatProperties and vkGetPhysicalDeviceImageFormatProperties **can** be used to check for supported properties of individual formats as normal.

- occlusionQueryPrecise specifies whether occlusion queries returning actual sample counts are supported. Occlusion queries are created in a VkQueryPool by specifying the queryType of VK_QUERY_TYPE_OCCLUSION in the VkQueryPoolCreateInfo structure which is passed to vkCreateQueryPool. If this feature is enabled, queries of this type **can** enable VK_QUERY_CONTROL_PRECISE_BIT in the flags parameter to vkCmdBeginQuery. If this feature is not supported, the implementation supports only boolean occlusion queries. When any samples are passed, boolean queries will return a non-zero result value, otherwise a result value of zero is returned. When this feature is enabled and VK_QUERY_CONTROL_PRECISE_BIT is set, occlusion queries will report the actual number of samples passed.
- pipelineStatisticsQuery specifies whether the pipeline statistics queries are supported. If this feature is not enabled, queries of type VK_QUERY_TYPE_PIPELINE_STATISTICS **cannot** be created, and none of the VkQueryPipelineStatisticFlagBits bits **can** be set in the pipelineStatistics member of the VkQueryPoolCreateInfo structure.
- vertexPipelineStoresAndAtomics specifies whether storage buffers and images support stores and atomic operations in the vertex, tessellation, and geometry shader stages. If this feature is not enabled, all storage image, storage texel buffer, and storage buffer variables used by these

stages in shader modules **must** be decorated with the `NonWritable` decoration (or the `readonly` memory qualifier in GLSL).

- `fragmentStoresAndAtomics` specifies whether storage buffers and images support stores and atomic operations in the fragment shader stage. If this feature is not enabled, all storage image, storage texel buffer, and storage buffer variables used by the fragment stage in shader modules **must** be decorated with the `NonWritable` decoration (or the `readonly` memory qualifier in GLSL).
- `shaderTessellationAndGeometryPointSize` specifies whether the `PointSize` built-in decoration is available in the tessellation control, tessellation evaluation, and geometry shader stages. If this feature is not enabled, members decorated with the `PointSize` built-in decoration **must** not be read from or written to and all points written from a tessellation or geometry shader will have a size of 1.0. This also specifies whether shader modules **can** declare the `TessellationPointSize` capability for tessellation control and evaluation shaders, or if the shader modules **can** declare the `GeometryPointSize` capability for geometry shaders. An implementation supporting this feature **must** also support one or both of the `tessellationShader` or `geometryShader` features.
- `shaderImageGatherExtended` specifies whether the extended set of image gather instructions are available in shader code. If this feature is not enabled, the `OpImage*Gather` instructions do not support the `Offset` and `ConstOffsets` operands. This also specifies whether shader modules **can** declare the `ImageGatherExtended` capability.
- `shaderStorageImageExtendedFormats` specifies whether all the “storage image extended formats” below are supported; if this feature is supported, then the `VK_FORMAT_FEATURE_STORAGE_IMAGE_BIT` **must** be supported in `optimalTilingFeatures` for the following formats:
  - `VK_FORMAT_R16G16_SFLOAT`
  - `VK_FORMAT_B10G11R11_UFLOAT_PACK32`
  - `VK_FORMAT_R16_SFLOAT`
  - `VK_FORMAT_R16G16B16A16_UNORM`
  - `VK_FORMAT_A2B10G10R10_UNORM_PACK32`
  - `VK_FORMAT_R16G16_UNORM`
  - `VK_FORMAT_R8G8_UNORM`
  - `VK_FORMAT_R16_UNORM`
  - `VK_FORMAT_R8_UNORM`
  - `VK_FORMAT_R16G16B16A16_SNORM`
  - `VK_FORMAT_R16G16_SNORM`
  - `VK_FORMAT_R8G8_SNORM`
  - `VK_FORMAT_R16_SNORM`
  - `VK_FORMAT_R8_SNORM`
  - `VK_FORMAT_R16G16_SINT`
  - `VK_FORMAT_R8G8_SINT`
  - `VK_FORMAT_R16_SINT`
  - `VK_FORMAT_R8_SINT`

- VK_FORMAT_A2B10G10R10_UINT_PACK32
- VK_FORMAT_R16G16_UINT
- VK_FORMAT_R8G8_UINT
- VK_FORMAT_R16_UINT
- VK_FORMAT_R8_UINT

**NOTE**

shaderStorageImageExtendedFormats feature only adds a guarantee of format support, which is specified for the whole physical device. Therefore enabling or disabling the feature via vkCreateDevice has no practical effect.

To query for additional properties, or if the feature is not supported, vkGetPhysicalDeviceFormatProperties and vkGetPhysicalDeviceImageFormatProperties **can** be used to check for supported properties of individual formats, as usual rules allow.

VK_FORMAT_R32G32_UINT, VK_FORMAT_R32G32_SINT, and VK_FORMAT_R32G32_SFLOAT from StorageImageExtendedFormats SPIR-V capability, are already covered by core Vulkan mandatory format support.

- shaderStorageImageMultisample specifies whether multisampled storage images are supported. If this feature is not enabled, images that are created with a usage that includes VK_IMAGE_USAGE_STORAGE_BIT **must** be created with samples equal to VK_SAMPLE_COUNT_1_BIT. This also specifies whether shader modules **can** declare the StorageImageMultisample and ImageMSArray capabilities.
- shaderStorageImageReadWithoutFormat specifies whether storage images and storage texel buffers require a format qualifier to be specified when reading. shaderStorageImageReadWithoutFormat applies only to formats listed in the storage without format list.
- shaderStorageImageWriteWithoutFormat specifies whether storage images and storage texel buffers require a format qualifier to be specified when writing. shaderStorageImageWriteWithoutFormat applies only to formats listed in the storage without format list.
- shaderUniformBufferArrayDynamicIndexing specifies whether arrays of uniform buffers **can** be indexed by integer expressions that are dynamically uniform within either the subgroup or the invocation group in shader code. If this feature is not enabled, resources with a descriptor type of VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER or VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC **must** be indexed only by constant integral expressions when aggregated into arrays in shader code. This also specifies whether shader modules **can** declare the UniformBufferArrayDynamicIndexing capability.
- shaderSampledImageArrayDynamicIndexing specifies whether arrays of samplers or sampled images **can** be indexed by integer expressions that are dynamically uniform within either the subgroup or the invocation group in shader code. If this feature is not enabled, resources with a descriptor type of VK_DESCRIPTOR_TYPE_SAMPLER, VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER, or VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE **must** be indexed only by constant integral expressions when aggregated into arrays in shader code. This also specifies whether shader modules **can** declare the SampledImageArrayDynamicIndexing capability.

- shaderStorageBufferArrayDynamicIndexing specifies whether arrays of storage buffers **can** be indexed by integer expressions that are dynamically uniform within either the subgroup or the invocation group in shader code. If this feature is not enabled, resources with a descriptor type of VK_DESCRIPTOR_TYPE_STORAGE_BUFFER or VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC **must** be indexed only by constant integral expressions when aggregated into arrays in shader code. This also specifies whether shader modules **can** declare the StorageBufferArrayDynamicIndexing capability.
- shaderStorageImageArrayDynamicIndexing specifies whether arrays of storage images **can** be indexed by integer expressions that are dynamically uniform within either the subgroup or the invocation group in shader code. If this feature is not enabled, resources with a descriptor type of VK_DESCRIPTOR_TYPE_STORAGE_IMAGE **must** be indexed only by constant integral expressions when aggregated into arrays in shader code. This also specifies whether shader modules **can** declare the StorageImageArrayDynamicIndexing capability.
- shaderClipDistance specifies whether clip distances are supported in shader code. If this feature is not enabled, any members decorated with the ClipDistance built-in decoration **must** not be read from or written to in shader modules. This also specifies whether shader modules **can** declare the ClipDistance capability.
- shaderCullDistance specifies whether cull distances are supported in shader code. If this feature is not enabled, any members decorated with the CullDistance built-in decoration **must** not be read from or written to in shader modules. This also specifies whether shader modules **can** declare the CullDistance capability.
- shaderFloat64 specifies whether 64-bit floats (doubles) are supported in shader code. If this feature is not enabled, 64-bit floating-point types **must** not be used in shader code. This also specifies whether shader modules **can** declare the Float64 capability. Declaring and using 64-bit floats is enabled for all storage classes that SPIR-V allows with the Float64 capability.
- shaderInt64 specifies whether 64-bit integers (signed and unsigned) are supported in shader code. If this feature is not enabled, 64-bit integer types **must** not be used in shader code. This also specifies whether shader modules **can** declare the Int64 capability. Declaring and using 64-bit integers is enabled for all storage classes that SPIR-V allows with the Int64 capability.
- shaderInt16 specifies whether 16-bit integers (signed and unsigned) are supported in shader code. If this feature is not enabled, 16-bit integer types **must** not be used in shader code. This also specifies whether shader modules **can** declare the Int16 capability. However, this only enables a subset of the storage classes that SPIR-V allows for the Int16 SPIR-V capability: Declaring and using 16-bit integers in the Private, Workgroup (for non-Block variables), and Function storage classes is enabled, while declaring them in the interface storage classes (e.g., UniformConstant, Uniform, StorageBuffer, Input, Output, and PushConstant) is not enabled.
- shaderResourceResidency specifies whether image operations that return resource residency information are supported in shader code. If this feature is not enabled, the OpImageSparse* instructions **must** not be used in shader code. This also specifies whether shader modules **can** declare the SparseResidency capability. The feature requires at least one of the sparseResidency* features to be supported.
- shaderResourceMinLod specifies whether image operations specifying the minimum resource LOD are supported in shader code. If this feature is not enabled, the MinLod image operand **must** not be used in shader code. This also specifies whether shader modules **can** declare the MinLod capability.

- sparseBinding specifies whether resource memory **can** be managed at opaque sparse block level instead of at the object level. If this feature is not enabled, resource memory **must** be bound only on a per-object basis using the vkBindBufferMemory and vkBindImageMemory commands. In this case, buffers and images **must** not be created with VK_BUFFER_CREATE_SPARSE_BINDING_BIT and VK_IMAGE_CREATE_SPARSE_BINDING_BIT set in the flags member of the VkBufferCreateInfo and VkImageCreateInfo structures, respectively. Otherwise resource memory **can** be managed as described in Sparse Resource Features.
- sparseResidencyBuffer specifies whether the device **can** access partially resident buffers. If this feature is not enabled, buffers **must** not be created with VK_BUFFER_CREATE_SPARSE_RESIDENCY_BIT set in the flags member of the VkBufferCreateInfo structure.
- sparseResidencyImage2D specifies whether the device **can** access partially resident 2D images with 1 sample per pixel. If this feature is not enabled, images with an imageType of VK_IMAGE_TYPE_2D and samples set to VK_SAMPLE_COUNT_1_BIT **must** not be created with VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT set in the flags member of the VkImageCreateInfo structure.
- sparseResidencyImage3D specifies whether the device **can** access partially resident 3D images. If this feature is not enabled, images with an imageType of VK_IMAGE_TYPE_3D **must** not be created with VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT set in the flags member of the VkImageCreateInfo structure.
- sparseResidency2Samples specifies whether the physical device **can** access partially resident 2D images with 2 samples per pixel. If this feature is not enabled, images with an imageType of VK_IMAGE_TYPE_2D and samples set to VK_SAMPLE_COUNT_2_BIT **must** not be created with VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT set in the flags member of the VkImageCreateInfo structure.
- sparseResidency4Samples specifies whether the physical device **can** access partially resident 2D images with 4 samples per pixel. If this feature is not enabled, images with an imageType of VK_IMAGE_TYPE_2D and samples set to VK_SAMPLE_COUNT_4_BIT **must** not be created with VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT set in the flags member of the VkImageCreateInfo structure.
- sparseResidency8Samples specifies whether the physical device **can** access partially resident 2D images with 8 samples per pixel. If this feature is not enabled, images with an imageType of VK_IMAGE_TYPE_2D and samples set to VK_SAMPLE_COUNT_8_BIT **must** not be created with VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT set in the flags member of the VkImageCreateInfo structure.
- sparseResidency16Samples specifies whether the physical device **can** access partially resident 2D images with 16 samples per pixel. If this feature is not enabled, images with an imageType of VK_IMAGE_TYPE_2D and samples set to VK_SAMPLE_COUNT_16_BIT **must** not be created with VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT set in the flags member of the VkImageCreateInfo structure.
- sparseResidencyAliased specifies whether the physical device **can** correctly access data aliased into multiple locations. If this feature is not enabled, the VK_BUFFER_CREATE_SPARSE_ALIASED_BIT and VK_IMAGE_CREATE_SPARSE_ALIASED_BIT enum values **must** not be used in flags members of the VkBufferCreateInfo and VkImageCreateInfo structures, respectively.
- variableMultisampleRate specifies whether all pipelines that will be bound to a command buffer

during a subpass which uses no attachments **must** have the same value for VkPipelineMultisampleStateCreateInfo::rasterizationSamples. If set to VK_TRUE, the implementation supports variable multisample rates in a subpass which uses no attachments. If set to VK_FALSE, then all pipelines bound in such a subpass **must** have the same multisample rate. This has no effect in situations where a subpass uses any attachments.

- inheritedQueries specifies whether a secondary command buffer **may** be executed while a query is active.

The VkPhysicalDeviceVariablePointersFeatures structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkPhysicalDeviceVariablePointersFeatures {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           variablePointersStorageBuffer;
    VkBool32           variablePointers;
} VkPhysicalDeviceVariablePointersFeatures;
```

```
// Provided by VK_VERSION_1_1
typedef VkPhysicalDeviceVariablePointersFeatures
VkPhysicalDeviceVariablePointerFeatures;
```

or the equivalent

```
// Provided by VK_KHR_variable_pointers
typedef VkPhysicalDeviceVariablePointersFeatures
VkPhysicalDeviceVariablePointersFeaturesKHR;
```

```
// Provided by VK_KHR_variable_pointers
typedef VkPhysicalDeviceVariablePointersFeatures
VkPhysicalDeviceVariablePointerFeaturesKHR;
```

This structure describes the following features:

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- variablePointersStorageBuffer specifies whether the implementation supports the SPIR-V VariablePointersStorageBuffer capability. When this feature is not enabled, shader modules **must** not declare the SPV_KHR_variable_pointers extension or the VariablePointersStorageBuffer capability.
- variablePointers specifies whether the implementation supports the SPIR-V VariablePointers capability. When this feature is not enabled, shader modules **must** not declare the VariablePointers capability.

If the VkPhysicalDeviceVariablePointersFeatures structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceVariablePointersFeatures **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage

- VUID-VkPhysicalDeviceVariablePointersFeatures-variablePointers-01431
  If variablePointers is enabled then variablePointersStorageBuffer **must** also be enabled

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceVariablePointersFeatures-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_VARIABLE_POINTERS_FEATURES

The VkPhysicalDeviceMultiviewFeatures structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkPhysicalDeviceMultiviewFeatures {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           multiview;
    VkBool32           multiviewGeometryShader;
    VkBool32           multiviewTessellationShader;
} VkPhysicalDeviceMultiviewFeatures;
```

or the equivalent

```
// Provided by VK_KHR_multiview
typedef VkPhysicalDeviceMultiviewFeatures VkPhysicalDeviceMultiviewFeaturesKHR;
```

This structure describes the following features:

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- multiview specifies whether the implementation supports multiview rendering within a render pass. If this feature is not enabled, the view mask of each subpass **must** always be zero.
- multiviewGeometryShader specifies whether the implementation supports multiview rendering within a render pass, with geometry shaders. If this feature is not enabled, then a pipeline compiled against a subpass with a non-zero view mask **must** not include a geometry shader.
- multiviewTessellationShader specifies whether the implementation supports multiview rendering within a render pass, with tessellation shaders. If this feature is not enabled, then a

pipeline compiled against a subpass with a non-zero view mask **must** not include any tessellation shaders.

If the VkPhysicalDeviceMultiviewFeatures structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceMultiviewFeatures **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage

- VUID-VkPhysicalDeviceMultiviewFeatures-multiviewGeometryShader-00580
  If multiviewGeometryShader is enabled then multiview **must** also be enabled
- VUID-VkPhysicalDeviceMultiviewFeatures-multiviewTessellationShader-00581
  If multiviewTessellationShader is enabled then multiview **must** also be enabled

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceMultiviewFeatures-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MULTIVIEW_FEATURES

The VkPhysicalDeviceShaderAtomicFloatFeaturesEXT structure is defined as:

```
// Provided by VK_EXT_shader_atomic_float
typedef struct VkPhysicalDeviceShaderAtomicFloatFeaturesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           shaderBufferFloat32Atomics;
    VkBool32           shaderBufferFloat32AtomicAdd;
    VkBool32           shaderBufferFloat64Atomics;
    VkBool32           shaderBufferFloat64AtomicAdd;
    VkBool32           shaderSharedFloat32Atomics;
    VkBool32           shaderSharedFloat32AtomicAdd;
    VkBool32           shaderSharedFloat64Atomics;
    VkBool32           shaderSharedFloat64AtomicAdd;
    VkBool32           shaderImageFloat32Atomics;
    VkBool32           shaderImageFloat32AtomicAdd;
    VkBool32           sparseImageFloat32Atomics;
    VkBool32           sparseImageFloat32AtomicAdd;
} VkPhysicalDeviceShaderAtomicFloatFeaturesEXT;
```

This structure describes the following features:

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- shaderBufferFloat32Atomics indicates whether shaders **can** perform 32-bit floating-point load,

store and exchange atomic operations on storage buffers.

- `shaderBufferFloat32AtomicAdd` indicates whether shaders **can** perform 32-bit floating-point add atomic operations on storage buffers.
- `shaderBufferFloat64Atomics` indicates whether shaders **can** perform 64-bit floating-point load, store and exchange atomic operations on storage buffers.
- `shaderBufferFloat64AtomicAdd` indicates whether shaders **can** perform 64-bit floating-point add atomic operations on storage buffers.
- `shaderSharedFloat32Atomics` indicates whether shaders **can** perform 32-bit floating-point load, store and exchange atomic operations on shared and payload memory.
- `shaderSharedFloat32AtomicAdd` indicates whether shaders **can** perform 32-bit floating-point add atomic operations on shared and payload memory.
- `shaderSharedFloat64Atomics` indicates whether shaders **can** perform 64-bit floating-point load, store and exchange atomic operations on shared and payload memory.
- `shaderSharedFloat64AtomicAdd` indicates whether shaders **can** perform 64-bit floating-point add atomic operations on shared and payload memory.
- `shaderImageFloat32Atomics` indicates whether shaders **can** perform 32-bit floating-point load, store and exchange atomic image operations.
- `shaderImageFloat32AtomicAdd` indicates whether shaders **can** perform 32-bit floating-point add atomic image operations.
- `sparseImageFloat32Atomics` indicates whether 32-bit floating-point load, store and exchange atomic operations **can** be used on sparse images.
- `sparseImageFloat32AtomicAdd` indicates whether 32-bit floating-point add atomic operations **can** be used on sparse images.

If the `VkPhysicalDeviceShaderAtomicFloatFeaturesEXT` structure is included in the `pNext` chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. `VkPhysicalDeviceShaderAtomicFloatFeaturesEXT` **can** also be used in the `pNext` chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceShaderAtomicFloatFeaturesEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_ATOMIC_FLOAT_FEATURES_EXT`

The VkPhysicalDeviceShaderAtomicFloat2FeaturesEXT structure is defined as:

```
// Provided by VK_EXT_shader_atomic_float2
typedef struct VkPhysicalDeviceShaderAtomicFloat2FeaturesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           shaderBufferFloat16Atomics;
    VkBool32           shaderBufferFloat16AtomicAdd;
    VkBool32           shaderBufferFloat16AtomicMinMax;
    VkBool32           shaderBufferFloat32AtomicMinMax;
    VkBool32           shaderBufferFloat64AtomicMinMax;
    VkBool32           shaderSharedFloat16Atomics;
    VkBool32           shaderSharedFloat16AtomicAdd;
    VkBool32           shaderSharedFloat16AtomicMinMax;
    VkBool32           shaderSharedFloat32AtomicMinMax;
    VkBool32           shaderSharedFloat64AtomicMinMax;
    VkBool32           shaderImageFloat32AtomicMinMax;
    VkBool32           sparseImageFloat32AtomicMinMax;
} VkPhysicalDeviceShaderAtomicFloat2FeaturesEXT;
```

This structure describes the following features:

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `shaderBufferFloat16Atomics` indicates whether shaders **can** perform 16-bit floating-point load, store, and exchange atomic operations on storage buffers.
- `shaderBufferFloat16AtomicAdd` indicates whether shaders **can** perform 16-bit floating-point add atomic operations on storage buffers.
- `shaderBufferFloat16AtomicMinMax` indicates whether shaders **can** perform 16-bit floating-point min and max atomic operations on storage buffers.
- `shaderBufferFloat32AtomicMinMax` indicates whether shaders **can** perform 32-bit floating-point min and max atomic operations on storage buffers.
- `shaderBufferFloat64AtomicMinMax` indicates whether shaders **can** perform 64-bit floating-point min and max atomic operations on storage buffers.
- `shaderSharedFloat16Atomics` indicates whether shaders **can** perform 16-bit floating-point load, store and exchange atomic operations on shared and payload memory.
- `shaderSharedFloat16AtomicAdd` indicates whether shaders **can** perform 16-bit floating-point add atomic operations on shared and payload memory.
- `shaderSharedFloat16AtomicMinMax` indicates whether shaders **can** perform 16-bit floating-point min and max atomic operations on shared and payload memory.
- `shaderSharedFloat32AtomicMinMax` indicates whether shaders **can** perform 32-bit floating-point min and max atomic operations on shared and payload memory.
- `shaderSharedFloat64AtomicMinMax` indicates whether shaders **can** perform 64-bit floating-point min and max atomic operations on shared and payload memory.
- `shaderImageFloat32AtomicMinMax` indicates whether shaders **can** perform 32-bit floating-point

min and max atomic image operations.

- sparseImageFloat32AtomicMinMax indicates whether 32-bit floating-point min and max atomic operations **can** be used on sparse images.

If the VkPhysicalDeviceShaderAtomicFloat2FeaturesEXT structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceShaderAtomicFloat2FeaturesEXT **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceShaderAtomicFloat2FeaturesEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_ATOMIC_FLOAT_2_FEATURES_EXT

The VkPhysicalDeviceShaderAtomicInt64Features structure is defined as:

```
typedef struct VkPhysicalDeviceShaderAtomicInt64Features {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           shaderBufferInt64Atomics;
    VkBool32           shaderSharedInt64Atomics;
} VkPhysicalDeviceShaderAtomicInt64Features;
```

or the equivalent

```
// Provided by VK_KHR_shader_atomic_int64
typedef VkPhysicalDeviceShaderAtomicInt64Features
VkPhysicalDeviceShaderAtomicInt64FeaturesKHR;
```

This structure describes the following features:

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- shaderBufferInt64Atomics indicates whether shaders **can** perform 64-bit unsigned and signed integer atomic operations on buffers.
- shaderSharedInt64Atomics indicates whether shaders **can** perform 64-bit unsigned and signed integer atomic operations on shared and payload memory.

If the VkPhysicalDeviceShaderAtomicInt64Features structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceShaderAtomicInt64Features **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceShaderAtomicInt64Features-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_ATOMIC_INT64_FEATURES`

The VkPhysicalDeviceShaderImageAtomicInt64FeaturesEXT structure is defined as:

```
// Provided by VK_EXT_shader_image_atomic_int64
typedef struct VkPhysicalDeviceShaderImageAtomicInt64FeaturesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           shaderImageInt64Atomics;
    VkBool32           sparseImageInt64Atomics;
} VkPhysicalDeviceShaderImageAtomicInt64FeaturesEXT;
```

This structure describes the following features:

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `shaderImageInt64Atomics` indicates whether shaders **can** support 64-bit unsigned and signed integer atomic operations on images.
- `sparseImageInt64Atomics` indicates whether 64-bit integer atomics **can** be used on sparse images.

If the `VkPhysicalDeviceShaderAtomicInt64FeaturesEXT` structure is included in the `pNext` chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. `VkPhysicalDeviceShaderAtomicInt64FeaturesEXT` **can** also be used in the `pNext` chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceShaderImageAtomicInt64FeaturesEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_IMAGE_ATOMIC_INT64_FEATURES_EXT`

The VkPhysicalDevice8BitStorageFeatures structure is defined as:

```
typedef struct VkPhysicalDevice8BitStorageFeatures {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           storageBuffer8BitAccess;
    VkBool32           uniformAndStorageBuffer8BitAccess;
    VkBool32           storagePushConstant8;
} VkPhysicalDevice8BitStorageFeatures;
```

or the equivalent

```
// Provided by VK_KHR_8bit_storage
typedef VkPhysicalDevice8BitStorageFeatures VkPhysicalDevice8BitStorageFeaturesKHR;
```

This structure describes the following features:

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `storageBuffer8BitAccess` indicates whether objects in the `StorageBuffer`, `ShaderRecordBufferKHR`, or `PhysicalStorageBuffer` storage class with the `Block` decoration **can** have 8-bit integer members. If this feature is not enabled, 8-bit integer members **must** not be used in such objects. This also indicates whether shader modules **can** declare the `StorageBuffer8BitAccess` capability.
- `uniformAndStorageBuffer8BitAccess` indicates whether objects in the `Uniform` storage class with the `Block` decoration **can** have 8-bit integer members. If this feature is not enabled, 8-bit integer members **must** not be used in such objects. This also indicates whether shader modules **can** declare the `UniformAndStorageBuffer8BitAccess` capability.
- `storagePushConstant8` indicates whether objects in the `PushConstant` storage class **can** have 8-bit integer members. If this feature is not enabled, 8-bit integer members **must** not be used in such objects. This also indicates whether shader modules **can** declare the `StoragePushConstant8` capability.

If the `VkPhysicalDevice8BitStorageFeatures` structure is included in the `pNext` chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. `VkPhysicalDevice8BitStorageFeatures` **can** also be used in the `pNext` chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDevice8BitStorageFeatures-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_8BIT_STORAGE_FEATURES`

The VkPhysicalDevice16BitStorageFeatures structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkPhysicalDevice16BitStorageFeatures {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           storageBuffer16BitAccess;
    VkBool32           uniformAndStorageBuffer16BitAccess;
    VkBool32           storagePushConstant16;
    VkBool32           storageInputOutput16;
} VkPhysicalDevice16BitStorageFeatures;
```

or the equivalent

```
// Provided by VK_KHR_16bit_storage
typedef VkPhysicalDevice16BitStorageFeatures VkPhysicalDevice16BitStorageFeaturesKHR;
```

This structure describes the following features:

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `storageBuffer16BitAccess` specifies whether objects in the `StorageBuffer`, `ShaderRecordBufferKHR`, or `PhysicalStorageBuffer` storage class with the `Block` decoration **can** have 16-bit integer and 16-bit floating-point members. If this feature is not enabled, 16-bit integer or 16-bit floating-point members **must** not be used in such objects. This also specifies whether shader modules **can** declare the `StorageBuffer16BitAccess` capability.
- `uniformAndStorageBuffer16BitAccess` specifies whether objects in the `Uniform` storage class with the `Block` decoration **can** have 16-bit integer and 16-bit floating-point members. If this feature is not enabled, 16-bit integer or 16-bit floating-point members **must** not be used in such objects. This also specifies whether shader modules **can** declare the `UniformAndStorageBuffer16BitAccess` capability.
- `storagePushConstant16` specifies whether objects in the `PushConstant` storage class **can** have 16-bit integer and 16-bit floating-point members. If this feature is not enabled, 16-bit integer or floating-point members **must** not be used in such objects. This also specifies whether shader modules **can** declare the `StoragePushConstant16` capability.
- `storageInputOutput16` specifies whether objects in the `Input` and `Output` storage classes **can** have 16-bit integer and 16-bit floating-point members. If this feature is not enabled, 16-bit integer or 16-bit floating-point members **must** not be used in such objects. This also specifies whether shader modules **can** declare the `StorageInputOutput16` capability.

If the `VkPhysicalDevice16BitStorageFeatures` structure is included in the `pNext` chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. `VkPhysicalDevice16BitStorageFeatures` **can** also be used in the `pNext` chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDevice16BitStorageFeatures-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_16BIT_STORAGE_FEATURES`

The `VkPhysicalDeviceShaderFloat16Int8Features` structure is defined as:

```
typedef struct VkPhysicalDeviceShaderFloat16Int8Features {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           shaderFloat16;
    VkBool32           shaderInt8;
} VkPhysicalDeviceShaderFloat16Int8Features;
```

or the equivalent

```
// Provided by VK_KHR_shader_float16_int8
typedef VkPhysicalDeviceShaderFloat16Int8Features
VkPhysicalDeviceShaderFloat16Int8FeaturesKHR;
```

```
// Provided by VK_KHR_shader_float16_int8
typedef VkPhysicalDeviceShaderFloat16Int8Features
VkPhysicalDeviceFloat16Int8FeaturesKHR;
```

This structure describes the following features:

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `shaderFloat16` indicates whether 16-bit floats (halfs) are supported in shader code. This also indicates whether shader modules **can** declare the `Float16` capability. However, this only enables a subset of the storage classes that SPIR-V allows for the `Float16` SPIR-V capability: Declaring and using 16-bit floats in the `Private`, `Workgroup` (for non-Block variables), and `Function` storage classes is enabled, while declaring them in the interface storage classes (e.g., `UniformConstant`, `Uniform`, `StorageBuffer`, `Input`, `Output`, and `PushConstant`) is not enabled.
- `shaderInt8` indicates whether 8-bit integers (signed and unsigned) are supported in shader code. This also indicates whether shader modules **can** declare the `Int8` capability. However, this only enables a subset of the storage classes that SPIR-V allows for the `Int8` SPIR-V capability: Declaring and using 8-bit integers in the `Private`, `Workgroup` (for non-Block variables), and `Function` storage classes is enabled, while declaring them in the interface storage classes (e.g., `UniformConstant`, `Uniform`, `StorageBuffer`, `Input`, `Output`, and `PushConstant`) is not enabled.

If the `VkPhysicalDeviceShaderFloat16Int8Features` structure is included in the `pNext` chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. `VkPhysicalDeviceShaderFloat16Int8Features` **can** also be used in the `pNext` chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceShaderFloat16Int8Features-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_FLOAT16_INT8_FEATURES`

The VkPhysicalDeviceShaderClockFeaturesKHR structure is defined as:

```
// Provided by VK_KHR_shader_clock
typedef struct VkPhysicalDeviceShaderClockFeaturesKHR {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           shaderSubgroupClock;
    VkBool32           shaderDeviceClock;
} VkPhysicalDeviceShaderClockFeaturesKHR;
```

This structure describes the following features:

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `shaderSubgroupClock` indicates whether shaders **can** perform `Subgroup` scoped clock reads.
- `shaderDeviceClock` indicates whether shaders **can** perform `Device` scoped clock reads.

If the `VkPhysicalDeviceShaderClockFeaturesKHR` structure is included in the `pNext` chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. `VkPhysicalDeviceShaderClockFeaturesKHR` **can** also be used in the `pNext` chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceShaderClockFeaturesKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_CLOCK_FEATURES_KHR`

The `VkPhysicalDeviceSamplerYcbcrConversionFeatures` structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkPhysicalDeviceSamplerYcbcrConversionFeatures {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           samplerYcbcrConversion;
} VkPhysicalDeviceSamplerYcbcrConversionFeatures;
```

or the equivalent

```
// Provided by VK_KHR_sampler_ycbcr_conversion
typedef VkPhysicalDeviceSamplerYcbcrConversionFeatures
VkPhysicalDeviceSamplerYcbcrConversionFeaturesKHR;
```

This structure describes the following feature:

- `sType` is a VkStructureType value identifying this structure.

- pNext is NULL or a pointer to a structure extending this structure.
- samplerYcbcrConversion specifies whether the implementation supports sampler $Y'C_BC_R$ conversion. If samplerYcbcrConversion is VK_FALSE, sampler $Y'C_BC_R$ conversion is not supported, and samplers using sampler $Y'C_BC_R$ conversion **must** not be used.

If the VkPhysicalDeviceSamplerYcbcrConversionFeatures structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceSamplerYcbcrConversionFeatures **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceSamplerYcbcrConversionFeatures-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SAMPLER_YCBCR_CONVERSION_FEATURES

The VkPhysicalDeviceProtectedMemoryFeatures structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkPhysicalDeviceProtectedMemoryFeatures {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           protectedMemory;
} VkPhysicalDeviceProtectedMemoryFeatures;
```

This structure describes the following feature:

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- protectedMemory specifies whether protected memory is supported.

If the VkPhysicalDeviceProtectedMemoryFeatures structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceProtectedMemoryFeatures **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceProtectedMemoryFeatures-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PROTECTED_MEMORY_FEATURES

The VkPhysicalDeviceBlendOperationAdvancedFeaturesEXT structure is defined as:

```
// Provided by VK_EXT_blend_operation_advanced
typedef struct VkPhysicalDeviceBlendOperationAdvancedFeaturesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           advancedBlendCoherentOperations;
} VkPhysicalDeviceBlendOperationAdvancedFeaturesEXT;
```

This structure describes the following feature:

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `advancedBlendCoherentOperations` specifies whether blending using advanced blend operations is guaranteed to execute atomically and in primitive order. If this is `VK_TRUE`, `VK_ACCESS_COLOR_ATTACHMENT_READ_NONCOHERENT_BIT_EXT` is treated the same as `VK_ACCESS_COLOR_ATTACHMENT_READ_BIT`, and advanced blending needs no additional synchronization over basic blending. If this is `VK_FALSE`, then memory dependencies are required to guarantee order between two advanced blending operations that occur on the same sample.

If the `VkPhysicalDeviceBlendOperationAdvancedFeaturesEXT` structure is included in the `pNext` chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. `VkPhysicalDeviceBlendOperationAdvancedFeaturesEXT` **can** also be used in the `pNext` chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceBlendOperationAdvancedFeaturesEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_BLEND_OPERATION_ADVANCED_FEATURES_EXT`

The `VkPhysicalDeviceConditionalRenderingFeaturesEXT` structure is defined as:

```
// Provided by VK_EXT_conditional_rendering
typedef struct VkPhysicalDeviceConditionalRenderingFeaturesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           conditionalRendering;
    VkBool32           inheritedConditionalRendering;
} VkPhysicalDeviceConditionalRenderingFeaturesEXT;
```

This structure describes the following features:

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `conditionalRendering` specifies whether conditional rendering is supported.

- inheritedConditionalRendering specifies whether a secondary command buffer **can** be executed while conditional rendering is active in the primary command buffer.

If the VkPhysicalDeviceConditionalRenderingFeaturesEXT structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceConditionalRenderingFeaturesEXT **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceConditionalRenderingFeaturesEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_CONDITIONAL_RENDERING_FEATURES_EXT

The VkPhysicalDeviceShaderDrawParametersFeatures structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkPhysicalDeviceShaderDrawParametersFeatures {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           shaderDrawParameters;
} VkPhysicalDeviceShaderDrawParametersFeatures;
```

```
// Provided by VK_VERSION_1_1
typedef VkPhysicalDeviceShaderDrawParametersFeatures
VkPhysicalDeviceShaderDrawParameterFeatures;
```

This structure describes the following feature:

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- shaderDrawParameters specifies whether the implementation supports the SPIR-V DrawParameters capability. When this feature is not enabled, shader modules **must** not declare the SPV_KHR_shader_draw_parameters extension or the DrawParameters capability.

If the VkPhysicalDeviceShaderDrawParametersFeatures structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceShaderDrawParametersFeatures **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceShaderDrawParametersFeatures-sType-sType

sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_DRAW_PARAMETERS_FEATURES

The VkPhysicalDeviceMeshShaderFeaturesNV structure is defined as:

```
// Provided by VK_NV_mesh_shader
typedef struct VkPhysicalDeviceMeshShaderFeaturesNV {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           taskShader;
    VkBool32           meshShader;
} VkPhysicalDeviceMeshShaderFeaturesNV;
```

This structure describes the following features:

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- taskShader specifies whether task shaders are supported. If this feature is not enabled, the VK_SHADER_STAGE_TASK_BIT_NV and VK_PIPELINE_STAGE_TASK_SHADER_BIT_NV enum values **must** not be used.
- meshShader specifies whether mesh shaders are supported. If this feature is not enabled, the VK_SHADER_STAGE_MESH_BIT_NV and VK_PIPELINE_STAGE_MESH_SHADER_BIT_NV enum values **must** not be used.

If the VkPhysicalDeviceMeshShaderFeaturesNV structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceMeshShaderFeaturesNV **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceMeshShaderFeaturesNV-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MESH_SHADER_FEATURES_NV

The VkPhysicalDeviceMeshShaderFeaturesEXT structure is defined as:

```
// Provided by VK_EXT_mesh_shader
typedef struct VkPhysicalDeviceMeshShaderFeaturesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           taskShader;
    VkBool32           meshShader;
    VkBool32           multiviewMeshShader;
    VkBool32           primitiveFragmentShadingRateMeshShader;
    VkBool32           meshShaderQueries;
} VkPhysicalDeviceMeshShaderFeaturesEXT;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- taskShader specifies whether task shaders are supported. If this feature is not enabled, the VK_SHADER_STAGE_TASK_BIT_EXT and VK_PIPELINE_STAGE_TASK_SHADER_BIT_EXT enum values **must** not be used.
- meshShader specifies whether mesh shaders are supported. If this feature is not enabled, the VK_SHADER_STAGE_MESH_BIT_EXT and VK_PIPELINE_STAGE_MESH_SHADER_BIT_EXT enum values **must** not be used.
- multiviewMeshShader specifies whether the implementation supports multiview rendering within a render pass, with mesh shaders. If this feature is not enabled, then a pipeline compiled against a subpass with a non-zero view mask **must** not include a mesh shader.
- primitiveFragmentShadingRateMeshShader indicates that the implementation supports the primitive fragment shading rate in mesh shaders.
- meshShaderQueries indicates that the implementation supports creating query pools using the VK_QUERY_TYPE_MESH_PRIMITIVES_GENERATED_EXT query type and statistic queries containing the VK_QUERY_PIPELINE_STATISTIC_TASK_SHADER_INVOCATIONS_BIT_EXT and VK_QUERY_PIPELINE_STATISTIC_MESH_SHADER_INVOCATIONS_BIT_EXT flags

If the VkPhysicalDeviceMeshShaderFeaturesEXT structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceMeshShaderFeaturesEXT **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

The corresponding features of the VkPhysicalDeviceMeshShaderFeaturesNV structure **must** match those in VkPhysicalDeviceMeshShaderFeaturesEXT.

### Valid Usage

- VUID-VkPhysicalDeviceMeshShaderFeaturesEXT-multiviewMeshShader-07032
  If multiviewMeshShader is enabled then VkPhysicalDeviceMultiviewFeaturesKHR::multiview **must** also be enabled
- VUID-VkPhysicalDeviceMeshShaderFeaturesEXT-primitiveFragmentShadingRateMeshShader-07033
  If primitiveFragmentShadingRateMeshShader is enabled then VkPhysicalDeviceFragmentShadingRateFeaturesKHR::primitiveFragmentShadingRate **must** also be enabled

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceMeshShaderFeaturesEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MESH_SHADER_FEATURES_EXT

The VkPhysicalDeviceMemoryDecompressionFeaturesNV structure is defined as:

```
// Provided by VK_NV_memory_decompression
typedef struct VkPhysicalDeviceMemoryDecompressionFeaturesNV {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           memoryDecompression;
} VkPhysicalDeviceMemoryDecompressionFeaturesNV;
```

This structure describes the following features:

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `memoryDecompression` indicates whether memory decompression is supported.

If the `VkPhysicalDeviceMemoryDecompressionFeaturesNV` structure is included in the `pNext` chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. `VkPhysicalDeviceMemoryDecompressionFeaturesNV` **can** also be used in the `pNext` chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceMemoryDecompressionFeaturesNV-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MEMORY_DECOMPRESSION_FEATURES_NV`

The `VkPhysicalDeviceDescriptorIndexingFeatures` structure is defined as:

```
typedef struct VkPhysicalDeviceDescriptorIndexingFeatures {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           shaderInputAttachmentArrayDynamicIndexing;
    VkBool32           shaderUniformTexelBufferArrayDynamicIndexing;
    VkBool32           shaderStorageTexelBufferArrayDynamicIndexing;
    VkBool32           shaderUniformBufferArrayNonUniformIndexing;
    VkBool32           shaderSampledImageArrayNonUniformIndexing;
    VkBool32           shaderStorageBufferArrayNonUniformIndexing;
    VkBool32           shaderStorageImageArrayNonUniformIndexing;
    VkBool32           shaderInputAttachmentArrayNonUniformIndexing;
    VkBool32           shaderUniformTexelBufferArrayNonUniformIndexing;
    VkBool32           shaderStorageTexelBufferArrayNonUniformIndexing;
    VkBool32           descriptorBindingUniformBufferUpdateAfterBind;
    VkBool32           descriptorBindingSampledImageUpdateAfterBind;
    VkBool32           descriptorBindingStorageImageUpdateAfterBind;
    VkBool32           descriptorBindingStorageBufferUpdateAfterBind;
    VkBool32           descriptorBindingUniformTexelBufferUpdateAfterBind;
    VkBool32           descriptorBindingStorageTexelBufferUpdateAfterBind;
    VkBool32           descriptorBindingUpdateUnusedWhilePending;
    VkBool32           descriptorBindingPartiallyBound;
    VkBool32           descriptorBindingVariableDescriptorCount;
    VkBool32           runtimeDescriptorArray;
} VkPhysicalDeviceDescriptorIndexingFeatures;
```

or the equivalent

```
// Provided by VK_EXT_descriptor_indexing
typedef VkPhysicalDeviceDescriptorIndexingFeatures
VkPhysicalDeviceDescriptorIndexingFeaturesEXT;
```

This structure describes the following features:

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `shaderInputAttachmentArrayDynamicIndexing` indicates whether arrays of input attachments **can** be indexed by integer expressions that are dynamically uniform within either the subgroup or the invocation group in shader code. If this feature is not enabled, resources with a descriptor type of `VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT` **must** be indexed only by constant integral expressions when aggregated into arrays in shader code. This also indicates whether shader modules **can** declare the `InputAttachmentArrayDynamicIndexing` capability.
- `shaderUniformTexelBufferArrayDynamicIndexing` indicates whether arrays of uniform texel buffers **can** be indexed by integer expressions that are dynamically uniform within either the subgroup or the invocation group in shader code. If this feature is not enabled, resources with a descriptor type of `VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER` **must** be indexed only by constant integral expressions when aggregated into arrays in shader code. This also indicates whether

shader modules **can** declare the UniformTexelBufferArrayDynamicIndexing capability.

- shaderStorageTexelBufferArrayDynamicIndexing indicates whether arrays of storage texel buffers **can** be indexed by integer expressions that are dynamically uniform within either the subgroup or the invocation group in shader code. If this feature is not enabled, resources with a descriptor type of VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER **must** be indexed only by constant integral expressions when aggregated into arrays in shader code. This also indicates whether shader modules **can** declare the StorageTexelBufferArrayDynamicIndexing capability.
- shaderUniformBufferArrayNonUniformIndexing indicates whether arrays of uniform buffers **can** be indexed by non-uniform integer expressions in shader code. If this feature is not enabled, resources with a descriptor type of VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER or VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC **must** not be indexed by non-uniform integer expressions when aggregated into arrays in shader code. This also indicates whether shader modules **can** declare the UniformBufferArrayNonUniformIndexing capability.
- shaderSampledImageArrayNonUniformIndexing indicates whether arrays of samplers or sampled images **can** be indexed by non-uniform integer expressions in shader code. If this feature is not enabled, resources with a descriptor type of VK_DESCRIPTOR_TYPE_SAMPLER, VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER, or VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE **must** not be indexed by non-uniform integer expressions when aggregated into arrays in shader code. This also indicates whether shader modules **can** declare the SampledImageArrayNonUniformIndexing capability.
- shaderStorageBufferArrayNonUniformIndexing indicates whether arrays of storage buffers **can** be indexed by non-uniform integer expressions in shader code. If this feature is not enabled, resources with a descriptor type of VK_DESCRIPTOR_TYPE_STORAGE_BUFFER or VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC **must** not be indexed by non-uniform integer expressions when aggregated into arrays in shader code. This also indicates whether shader modules **can** declare the StorageBufferArrayNonUniformIndexing capability.
- shaderStorageImageArrayNonUniformIndexing indicates whether arrays of storage images **can** be indexed by non-uniform integer expressions in shader code. If this feature is not enabled, resources with a descriptor type of VK_DESCRIPTOR_TYPE_STORAGE_IMAGE **must** not be indexed by non-uniform integer expressions when aggregated into arrays in shader code. This also indicates whether shader modules **can** declare the StorageImageArrayNonUniformIndexing capability.
- shaderInputAttachmentArrayNonUniformIndexing indicates whether arrays of input attachments **can** be indexed by non-uniform integer expressions in shader code. If this feature is not enabled, resources with a descriptor type of VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT **must** not be indexed by non-uniform integer expressions when aggregated into arrays in shader code. This also indicates whether shader modules **can** declare the InputAttachmentArrayNonUniformIndexing capability.
- shaderUniformTexelBufferArrayNonUniformIndexing indicates whether arrays of uniform texel buffers **can** be indexed by non-uniform integer expressions in shader code. If this feature is not enabled, resources with a descriptor type of VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER **must** not be indexed by non-uniform integer expressions when aggregated into arrays in shader code. This also indicates whether shader modules **can** declare the UniformTexelBufferArrayNonUniformIndexing capability.
- shaderStorageTexelBufferArrayNonUniformIndexing indicates whether arrays of storage texel

buffers **can** be indexed by non-uniform integer expressions in shader code. If this feature is not enabled, resources with a descriptor type of VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER **must** not be indexed by non-uniform integer expressions when aggregated into arrays in shader code. This also indicates whether shader modules **can** declare the StorageTexelBufferArrayNonUniformIndexing capability.

- descriptorBindingUniformBufferUpdateAfterBind indicates whether the implementation supports updating uniform buffer descriptors after a set is bound. If this feature is not enabled, VK_DESCRIPTOR_BINDING_UPDATE_AFTER_BIND_BIT **must** not be used with VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER.
- descriptorBindingSampledImageUpdateAfterBind indicates whether the implementation supports updating sampled image descriptors after a set is bound. If this feature is not enabled, VK_DESCRIPTOR_BINDING_UPDATE_AFTER_BIND_BIT **must** not be used with VK_DESCRIPTOR_TYPE_SAMPLER, VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER, or VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE.
- descriptorBindingStorageImageUpdateAfterBind indicates whether the implementation supports updating storage image descriptors after a set is bound. If this feature is not enabled, VK_DESCRIPTOR_BINDING_UPDATE_AFTER_BIND_BIT **must** not be used with VK_DESCRIPTOR_TYPE_STORAGE_IMAGE.
- descriptorBindingStorageBufferUpdateAfterBind indicates whether the implementation supports updating storage buffer descriptors after a set is bound. If this feature is not enabled, VK_DESCRIPTOR_BINDING_UPDATE_AFTER_BIND_BIT **must** not be used with VK_DESCRIPTOR_TYPE_STORAGE_BUFFER.
- descriptorBindingUniformTexelBufferUpdateAfterBind indicates whether the implementation supports updating uniform texel buffer descriptors after a set is bound. If this feature is not enabled, VK_DESCRIPTOR_BINDING_UPDATE_AFTER_BIND_BIT **must** not be used with VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER.
- descriptorBindingStorageTexelBufferUpdateAfterBind indicates whether the implementation supports updating storage texel buffer descriptors after a set is bound. If this feature is not enabled, VK_DESCRIPTOR_BINDING_UPDATE_AFTER_BIND_BIT **must** not be used with VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER.
- descriptorBindingUpdateUnusedWhilePending indicates whether the implementation supports updating descriptors while the set is in use. If this feature is not enabled, VK_DESCRIPTOR_BINDING_UPDATE_UNUSED_WHILE_PENDING_BIT **must** not be used.
- descriptorBindingPartiallyBound indicates whether the implementation supports statically using a descriptor set binding in which some descriptors are not valid. If this feature is not enabled, VK_DESCRIPTOR_BINDING_PARTIALLY_BOUND_BIT **must** not be used.
- descriptorBindingVariableDescriptorCount indicates whether the implementation supports descriptor sets with a variable-sized last binding. If this feature is not enabled, VK_DESCRIPTOR_BINDING_VARIABLE_DESCRIPTOR_COUNT_BIT **must** not be used.
- runtimeDescriptorArray indicates whether the implementation supports the SPIR-V RuntimeDescriptorArray capability. If this feature is not enabled, descriptors **must** not be declared in runtime arrays.

If the VkPhysicalDeviceDescriptorIndexingFeatures structure is included in the pNext chain of the

VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceDescriptorIndexingFeatures **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceDescriptorIndexingFeatures-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DESCRIPTOR_INDEXING_FEATURES

The VkPhysicalDeviceCopyMemoryIndirectFeaturesNV structure is defined as:

```
// Provided by VK_NV_copy_memory_indirect
typedef struct VkPhysicalDeviceCopyMemoryIndirectFeaturesNV {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           indirectCopy;
} VkPhysicalDeviceCopyMemoryIndirectFeaturesNV;
```

This structure describes the following features:

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- indirectCopy indicates whether indirect copies are supported.

If the VkPhysicalDeviceCopyMemoryIndirectFeaturesNV structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceCopyMemoryIndirectFeaturesNV **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceCopyMemoryIndirectFeaturesNV-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_COPY_MEMORY_INDIRECT_FEATURES_NV

The VkPhysicalDeviceVertexAttributeDivisorFeaturesKHR structure is defined as:

```
// Provided by VK_KHR_vertex_attribute_divisor
typedef struct VkPhysicalDeviceVertexAttributeDivisorFeaturesKHR {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           vertexAttributeInstanceRateDivisor;
    VkBool32           vertexAttributeInstanceRateZeroDivisor;
} VkPhysicalDeviceVertexAttributeDivisorFeaturesKHR;
```

or the equivalent

```
// Provided by VK_EXT_vertex_attribute_divisor
typedef VkPhysicalDeviceVertexAttributeDivisorFeaturesKHR
VkPhysicalDeviceVertexAttributeDivisorFeaturesEXT;
```

This structure describes the following features:

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `vertexAttributeInstanceRateDivisor` specifies whether vertex attribute fetching may be repeated in the case of instanced rendering.
- `vertexAttributeInstanceRateZeroDivisor` specifies whether a zero value for VkVertexInputBindingDivisorDescriptionEXT::`divisor` is supported.

If the `VkPhysicalDeviceVertexAttributeDivisorFeaturesKHR` structure is included in the `pNext` chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. `VkPhysicalDeviceVertexAttributeDivisorFeaturesKHR` **can** also be used in the `pNext` chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceVertexAttributeDivisorFeaturesKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_VERTEX_ATTRIBUTE_DIVISOR_FEATURES_KHR`

The `VkPhysicalDeviceASTCDecodeFeaturesEXT` structure is defined as:

```
// Provided by VK_EXT_astc_decode_mode
typedef struct VkPhysicalDeviceASTCDecodeFeaturesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           decodeModeSharedExponent;
} VkPhysicalDeviceASTCDecodeFeaturesEXT;
```

This structure describes the following feature:

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- decodeModeSharedExponent indicates whether the implementation supports decoding ASTC compressed formats to VK_FORMAT_E5B9G9R9_UFLOAT_PACK32 internal precision.

If the VkPhysicalDeviceASTCDecodeFeaturesEXT structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceASTCDecodeFeaturesEXT **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceASTCDecodeFeaturesEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_ASTC_DECODE_FEATURES_EXT

The VkPhysicalDeviceTransformFeedbackFeaturesEXT structure is defined as:

```
// Provided by VK_EXT_transform_feedback
typedef struct VkPhysicalDeviceTransformFeedbackFeaturesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           transformFeedback;
    VkBool32           geometryStreams;
} VkPhysicalDeviceTransformFeedbackFeaturesEXT;
```

This structure describes the following features:

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- transformFeedback indicates whether the implementation supports transform feedback and shader modules **can** declare the TransformFeedback capability.
- geometryStreams indicates whether the implementation supports the GeometryStreams SPIR-V capability.

If the VkPhysicalDeviceTransformFeedbackFeaturesEXT structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceTransformFeedbackFeaturesEXT **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceTransformFeedbackFeaturesEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_TRANSFORM_FEEDBACK_FEATURES_EXT

The VkPhysicalDeviceVulkanMemoryModelFeatures structure is defined as:

```
typedef struct VkPhysicalDeviceVulkanMemoryModelFeatures {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           vulkanMemoryModel;
    VkBool32           vulkanMemoryModelDeviceScope;
    VkBool32           vulkanMemoryModelAvailabilityVisibilityChains;
} VkPhysicalDeviceVulkanMemoryModelFeatures;
```

or the equivalent

```
// Provided by VK_KHR_vulkan_memory_model
typedef VkPhysicalDeviceVulkanMemoryModelFeatures
VkPhysicalDeviceVulkanMemoryModelFeaturesKHR;
```

This structure describes the following features:

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- vulkanMemoryModel indicates whether shader modules **can** declare the VulkanMemoryModel capability.
- vulkanMemoryModelDeviceScope indicates whether the Vulkan Memory Model can use Device scope synchronization. This also indicates whether shader modules **can** declare the VulkanMemoryModelDeviceScope capability.
- vulkanMemoryModelAvailabilityVisibilityChains indicates whether the Vulkan Memory Model can use availability and visibility chains with more than one element.

If the VkPhysicalDeviceVulkanMemoryModelFeaturesKHR structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceVulkanMemoryModelFeaturesKHR **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceVulkanMemoryModelFeatures-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_VULKAN_MEMORY_MODEL_FEATURES

The VkPhysicalDeviceInlineUniformBlockFeatures structure is defined as:

```
typedef struct VkPhysicalDeviceInlineUniformBlockFeatures {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           inlineUniformBlock;
    VkBool32           descriptorBindingInlineUniformBlockUpdateAfterBind;
} VkPhysicalDeviceInlineUniformBlockFeatures;
```

or the equivalent

```
// Provided by VK_EXT_inline_uniform_block
typedef VkPhysicalDeviceInlineUniformBlockFeatures
VkPhysicalDeviceInlineUniformBlockFeaturesEXT;
```

This structure describes the following features:

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `inlineUniformBlock` indicates whether the implementation supports inline uniform block descriptors. If this feature is not enabled, `VK_DESCRIPTOR_TYPE_INLINE_UNIFORM_BLOCK` **must** not be used.
- `descriptorBindingInlineUniformBlockUpdateAfterBind` indicates whether the implementation supports updating inline uniform block descriptors after a set is bound. If this feature is not enabled, `VK_DESCRIPTOR_BINDING_UPDATE_AFTER_BIND_BIT` **must** not be used with `VK_DESCRIPTOR_TYPE_INLINE_UNIFORM_BLOCK`.

If the `VkPhysicalDeviceInlineUniformBlockFeatures` structure is included in the `pNext` chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. `VkPhysicalDeviceInlineUniformBlockFeatures` **can** also be used in the `pNext` chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceInlineUniformBlockFeatures-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_INLINE_UNIFORM_BLOCK_FEATURES`

The `VkPhysicalDeviceRepresentativeFragmentTestFeaturesNV` structure is defined as:

```
// Provided by VK_NV_representative_fragment_test
typedef struct VkPhysicalDeviceRepresentativeFragmentTestFeaturesNV {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           representativeFragmentTest;
} VkPhysicalDeviceRepresentativeFragmentTestFeaturesNV;
```

This structure describes the following feature:

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `representativeFragmentTest` indicates whether the implementation supports the representative fragment test. See Representative Fragment Test.

If the `VkPhysicalDeviceRepresentativeFragmentTestFeaturesNV` structure is included in the `pNext` chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. `VkPhysicalDeviceRepresentativeFragmentTestFeaturesNV` **can** also be used in the `pNext` chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceRepresentativeFragmentTestFeaturesNV-sType-sType `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_REPRESENTATIVE_FRAGMENT_TEST_FEATURES_NV`

The `VkPhysicalDeviceExclusiveScissorFeaturesNV` structure is defined as:

```
// Provided by VK_NV_scissor_exclusive
typedef struct VkPhysicalDeviceExclusiveScissorFeaturesNV {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           exclusiveScissor;
} VkPhysicalDeviceExclusiveScissorFeaturesNV;
```

This structure describes the following feature:

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `exclusiveScissor` indicates that the implementation supports the exclusive scissor test.

See Exclusive Scissor Test for more information.

If the `VkPhysicalDeviceExclusiveScissorFeaturesNV` structure is included in the `pNext` chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to

indicate whether each corresponding feature is supported. VkPhysicalDeviceExclusiveScissorFeaturesNV **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceExclusiveScissorFeaturesNV-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXCLUSIVE_SCISSOR_FEATURES_NV

The VkPhysicalDeviceCornerSampledImageFeaturesNV structure is defined as:

```
// Provided by VK_NV_corner_sampled_image
typedef struct VkPhysicalDeviceCornerSampledImageFeaturesNV {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           cornerSampledImage;
} VkPhysicalDeviceCornerSampledImageFeaturesNV;
```

This structure describes the following feature:

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- cornerSampledImage specifies whether images can be created with a VkImageCreateInfo::flags containing VK_IMAGE_CREATE_CORNER_SAMPLED_BIT_NV. See Corner-Sampled Images.

If the VkPhysicalDeviceCornerSampledImageFeaturesNV structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceCornerSampledImageFeaturesNV **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceCornerSampledImageFeaturesNV-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_CORNER_SAMPLED_IMAGE_FEATURES_NV

The VkPhysicalDeviceComputeShaderDerivativesFeaturesKHR structure is defined as:

```
// Provided by VK_KHR_compute_shader_derivatives
typedef struct VkPhysicalDeviceComputeShaderDerivativesFeaturesKHR {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           computeDerivativeGroupQuads;
    VkBool32           computeDerivativeGroupLinear;
} VkPhysicalDeviceComputeShaderDerivativesFeaturesKHR;
```

or the equivalent

```
// Provided by VK_NV_compute_shader_derivatives
typedef VkPhysicalDeviceComputeShaderDerivativesFeaturesKHR
VkPhysicalDeviceComputeShaderDerivativesFeaturesNV;
```

This structure describes the following features:

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `computeDerivativeGroupQuads` indicates that the implementation supports the `ComputeDerivativeGroupQuadsKHR` SPIR-V capability.
- `computeDerivativeGroupLinear` indicates that the implementation supports the `ComputeDerivativeGroupLinearKHR` SPIR-V capability.

See Quad shader scope for more information.

If the `VkPhysicalDeviceComputeShaderDerivativesFeaturesKHR`. structure is included in the `pNext` chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. `VkPhysicalDeviceComputeShaderDerivativesFeaturesKHR`. **can** also be used in the `pNext` chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceComputeShaderDerivativesFeaturesKHR-sType-sType `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_COMPUTE_SHADER_DERIVATIVES_FEATURES_KHR`

The `VkPhysicalDeviceFragmentShaderBarycentricFeaturesKHR` structure is defined as:

```
// Provided by VK_KHR_fragment_shader_barycentric
typedef struct VkPhysicalDeviceFragmentShaderBarycentricFeaturesKHR {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           fragmentShaderBarycentric;
} VkPhysicalDeviceFragmentShaderBarycentricFeaturesKHR;
```

or the equivalent

```
// Provided by VK_NV_fragment_shader_barycentric
typedef VkPhysicalDeviceFragmentShaderBarycentricFeaturesKHR
VkPhysicalDeviceFragmentShaderBarycentricFeaturesNV;
```

This structure describes the following feature:

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `fragmentShaderBarycentric` indicates that the implementation supports the `BaryCoordKHR` and `BaryCoordNoPerspKHR` SPIR-V fragment shader built-ins and supports the `PerVertexKHR` SPIR-V decoration on fragment shader input variables.

See Barycentric Interpolation for more information.

If the `VkPhysicalDeviceFragmentShaderBarycentricFeaturesKHR` structure is included in the `pNext` chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. `VkPhysicalDeviceFragmentShaderBarycentricFeaturesKHR` **can** also be used in the `pNext` chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceFragmentShaderBarycentricFeaturesKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_FRAGMENT_SHADER_BARYCENTRIC_FEATURES_KHR`

The `VkPhysicalDeviceShaderImageFootprintFeaturesNV` structure is defined as:

```
// Provided by VK_NV_shader_image_footprint
typedef struct VkPhysicalDeviceShaderImageFootprintFeaturesNV {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           imageFootprint;
} VkPhysicalDeviceShaderImageFootprintFeaturesNV;
```

This structure describes the following feature:

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- imageFootprint specifies whether the implementation supports the ImageFootprintNV SPIR-V capability.

See Texel Footprint Evaluation for more information.

If the VkPhysicalDeviceShaderImageFootprintFeaturesNV structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceShaderImageFootprintFeaturesNV **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceShaderImageFootprintFeaturesNV-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_IMAGE_FOOTPRINT_FEATURES_NV

The VkPhysicalDeviceShadingRateImageFeaturesNV structure is defined as:

```
// Provided by VK_NV_shading_rate_image
typedef struct VkPhysicalDeviceShadingRateImageFeaturesNV {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           shadingRateImage;
    VkBool32           shadingRateCoarseSampleOrder;
} VkPhysicalDeviceShadingRateImageFeaturesNV;
```

This structure describes the following features:

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- shadingRateImage indicates that the implementation supports the use of a shading rate image to derive an effective shading rate for fragment processing. It also indicates that the implementation supports the ShadingRateNV SPIR-V execution mode.
- shadingRateCoarseSampleOrder indicates that the implementation supports an application-configurable ordering of coverage samples in fragments larger than one pixel.

See Shading Rate Image for more information.

If the VkPhysicalDeviceShadingRateImageFeaturesNV structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceShadingRateImageFeaturesNV **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceShadingRateImageFeaturesNV-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADING_RATE_IMAGE_FEATURES_NV`

The `VkPhysicalDeviceFragmentDensityMapFeaturesEXT` structure is defined as:

```
// Provided by VK_EXT_fragment_density_map
typedef struct VkPhysicalDeviceFragmentDensityMapFeaturesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           fragmentDensityMap;
    VkBool32           fragmentDensityMapDynamic;
    VkBool32           fragmentDensityMapNonSubsampledImages;
} VkPhysicalDeviceFragmentDensityMapFeaturesEXT;
```

This structure describes the following features:

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `fragmentDensityMap` specifies whether the implementation supports render passes with a fragment density map attachment. If this feature is not enabled and the `pNext` chain of VkRenderPassCreateInfo includes a VkRenderPassFragmentDensityMapCreateInfoEXT structure, `fragmentDensityMapAttachment` **must** be `VK_ATTACHMENT_UNUSED`.
- `fragmentDensityMapDynamic` specifies whether the implementation supports dynamic fragment density map image views. If this feature is not enabled, `VK_IMAGE_VIEW_CREATE_FRAGMENT_DENSITY_MAP_DYNAMIC_BIT_EXT` **must** not be included in VkImageViewCreateInfo::`flags`.
- `fragmentDensityMapNonSubsampledImages` specifies whether the implementation supports regular non-subsampled image attachments with fragment density map render passes. If this feature is not enabled, render passes with a fragment density map attachment **must** only have subsampled attachments bound.

If the `VkPhysicalDeviceFragmentDensityMapFeaturesEXT` structure is included in the `pNext` chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. `VkPhysicalDeviceFragmentDensityMapFeaturesEXT` **can** also be used in the `pNext` chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceFragmentDensityMapFeaturesEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_FRAGMENT_DENSITY_MAP_FEATURES_EXT`

The VkPhysicalDeviceFragmentDensityMap2FeaturesEXT structure is defined as:

```
// Provided by VK_EXT_fragment_density_map2
typedef struct VkPhysicalDeviceFragmentDensityMap2FeaturesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           fragmentDensityMapDeferred;
} VkPhysicalDeviceFragmentDensityMap2FeaturesEXT;
```

This structure describes the following feature:

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- fragmentDensityMapDeferred specifies whether the implementation supports deferred reads of fragment density map image views. If this feature is not enabled, VK_IMAGE_VIEW_CREATE_FRAGMENT_DENSITY_MAP_DEFERRED_BIT_EXT **must** not be included in VkImageViewCreateInfo::flags.

If the VkPhysicalDeviceFragmentDensityMap2FeaturesEXT structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceFragmentDensityMap2FeaturesEXT **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceFragmentDensityMap2FeaturesEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_FRAGMENT_DENSITY_MAP_2_FEATURES_EXT

The VkPhysicalDeviceFragmentDensityMapOffsetFeaturesQCOM structure is defined as:

```
// Provided by VK_QCOM_fragment_density_map_offset
typedef struct VkPhysicalDeviceFragmentDensityMapOffsetFeaturesQCOM {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           fragmentDensityMapOffset;
} VkPhysicalDeviceFragmentDensityMapOffsetFeaturesQCOM;
```

This structure describes the following feature:

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- fragmentDensityMapOffset specifies whether the implementation supports fragment density map offsets

If the VkPhysicalDeviceFragmentDensityMapOffsetFeaturesQCOM structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceFragmentDensityMapOffsetFeaturesQCOM **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceFragmentDensityMapOffsetFeaturesQCOM-sType-sType sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_FRAGMENT_DENSITY_MAP_OFFSET_FEATURES_QCOM

The VkPhysicalDeviceInvocationMaskFeaturesHUAWEI structure is defined as:

```
// Provided by VK_HUAWEI_invocation_mask
typedef struct VkPhysicalDeviceInvocationMaskFeaturesHUAWEI {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           invocationMask;
} VkPhysicalDeviceInvocationMaskFeaturesHUAWEI;
```

This structure describes the following feature:

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- invocationMask indicates that the implementation supports the use of an invocation mask image to optimize the ray dispatch.

If the VkPhysicalDeviceInvocationMaskFeaturesHUAWEI structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceInvocationMaskFeaturesHUAWEI **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceInvocationMaskFeaturesHUAWEI-sType-sType sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_INVOCATION_MASK_FEATURES_HUAWEI

The VkPhysicalDeviceScalarBlockLayoutFeatures structure is defined as:

```
typedef struct VkPhysicalDeviceScalarBlockLayoutFeatures {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           scalarBlockLayout;
} VkPhysicalDeviceScalarBlockLayoutFeatures;
```

or the equivalent

```
// Provided by VK_EXT_scalar_block_layout
typedef VkPhysicalDeviceScalarBlockLayoutFeatures
VkPhysicalDeviceScalarBlockLayoutFeaturesEXT;
```

This structure describes the following feature:

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `scalarBlockLayout` indicates that the implementation supports the layout of resource blocks in shaders using scalar alignment.

If the `VkPhysicalDeviceScalarBlockLayoutFeatures` structure is included in the `pNext` chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. `VkPhysicalDeviceScalarBlockLayoutFeatures` **can** also be used in the `pNext` chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceScalarBlockLayoutFeatures-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SCALAR_BLOCK_LAYOUT_FEATURES`

The `VkPhysicalDeviceUniformBufferStandardLayoutFeatures` structure is defined as:

```
typedef struct VkPhysicalDeviceUniformBufferStandardLayoutFeatures {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           uniformBufferStandardLayout;
} VkPhysicalDeviceUniformBufferStandardLayoutFeatures;
```

or the equivalent

```
// Provided by VK_KHR_uniform_buffer_standard_layout
typedef VkPhysicalDeviceUniformBufferStandardLayoutFeatures
VkPhysicalDeviceUniformBufferStandardLayoutFeaturesKHR;
```

This structure describes the following feature:

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- uniformBufferStandardLayout indicates that the implementation supports the same layouts for uniform buffers as for storage and other kinds of buffers. See Standard Buffer Layout.

If the VkPhysicalDeviceUniformBufferStandardLayoutFeatures structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceUniformBufferStandardLayoutFeatures **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceUniformBufferStandardLayoutFeatures-sType-sType sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_UNIFORM_BUFFER_STANDARD_LAYOUT_FEATURES

The VkPhysicalDeviceDepthClipEnableFeaturesEXT structure is defined as:

```
// Provided by VK_EXT_depth_clip_enable
typedef struct VkPhysicalDeviceDepthClipEnableFeaturesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           depthClipEnable;
} VkPhysicalDeviceDepthClipEnableFeaturesEXT;
```

This structure describes the following feature:

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- depthClipEnable indicates that the implementation supports setting the depth clipping operation explicitly via the VkPipelineRasterizationDepthClipStateCreateInfoEXT pipeline state. Otherwise depth clipping is only enabled when VkPipelineRasterizationStateCreateInfo::depthClampEnable is set to VK_FALSE.

If the VkPhysicalDeviceDepthClipEnableFeaturesEXT structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceDepthClipEnableFeaturesEXT **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceDepthClipEnableFeaturesEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DEPTH_CLIP_ENABLE_FEATURES_EXT`

The `VkPhysicalDeviceMemoryPriorityFeaturesEXT` structure is defined as:

```
// Provided by VK_EXT_memory_priority
typedef struct VkPhysicalDeviceMemoryPriorityFeaturesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           memoryPriority;
} VkPhysicalDeviceMemoryPriorityFeaturesEXT;
```

This structure describes the following feature:

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `memoryPriority` indicates that the implementation supports memory priorities specified at memory allocation time via VkMemoryPriorityAllocateInfoEXT.

If the `VkPhysicalDeviceMemoryPriorityFeaturesEXT` structure is included in the `pNext` chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. `VkPhysicalDeviceMemoryPriorityFeaturesEXT` **can** also be used in the `pNext` chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceMemoryPriorityFeaturesEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MEMORY_PRIORITY_FEATURES_EXT`

The `VkPhysicalDeviceBufferDeviceAddressFeatures` structure is defined as:

```
typedef struct VkPhysicalDeviceBufferDeviceAddressFeatures {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           bufferDeviceAddress;
    VkBool32           bufferDeviceAddressCaptureReplay;
    VkBool32           bufferDeviceAddressMultiDevice;
} VkPhysicalDeviceBufferDeviceAddressFeatures;
```

or the equivalent

```
// Provided by VK_KHR_buffer_device_address
typedef VkPhysicalDeviceBufferDeviceAddressFeatures
VkPhysicalDeviceBufferDeviceAddressFeaturesKHR;
```

This structure describes the following features:

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `bufferDeviceAddress` indicates that the implementation supports accessing buffer memory in shaders as storage buffers via an address queried from vkGetBufferDeviceAddressKHR.
- `bufferDeviceAddressCaptureReplay` indicates that the implementation supports saving and reusing buffer and device addresses, e.g. for trace capture and replay.
- `bufferDeviceAddressMultiDevice` indicates that the implementation supports the `bufferDeviceAddress` , `rayTracingPipeline` and `rayQuery` features for logical devices created with multiple physical devices. If this feature is not supported, buffer and acceleration structure addresses **must** not be queried on a logical device created with more than one physical device.

**NOTE** `bufferDeviceAddressMultiDevice` exists to allow certain legacy platforms to be able to support `bufferDeviceAddress` without needing to support shared GPU virtual addresses for multi-device configurations.

See vkGetBufferDeviceAddressKHR for more information.

If the `VkPhysicalDeviceBufferDeviceAddressFeatures` structure is included in the `pNext` chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. `VkPhysicalDeviceBufferDeviceAddressFeatures` **can** also be used in the `pNext` chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceBufferDeviceAddressFeatures-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_BUFFER_DEVICE_ADDRESS_FEATURES`

The `VkPhysicalDeviceBufferDeviceAddressFeaturesEXT` structure is defined as:

```
// Provided by VK_EXT_buffer_device_address
typedef struct VkPhysicalDeviceBufferDeviceAddressFeaturesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           bufferDeviceAddress;
    VkBool32           bufferDeviceAddressCaptureReplay;
    VkBool32           bufferDeviceAddressMultiDevice;
} VkPhysicalDeviceBufferDeviceAddressFeaturesEXT;
```

```
// Provided by VK_EXT_buffer_device_address
typedef VkPhysicalDeviceBufferDeviceAddressFeaturesEXT
VkPhysicalDeviceBufferAddressFeaturesEXT;
```

This structure describes the following features:

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `bufferDeviceAddress` indicates that the implementation supports accessing buffer memory in shaders as storage buffers via an address queried from vkGetBufferDeviceAddressEXT.
- `bufferDeviceAddressCaptureReplay` indicates that the implementation supports saving and reusing buffer addresses, e.g. for trace capture and replay.
- `bufferDeviceAddressMultiDevice` indicates that the implementation supports the `bufferDeviceAddress` feature for logical devices created with multiple physical devices. If this feature is not supported, buffer addresses **must** not be queried on a logical device created with more than one physical device.

If the `VkPhysicalDeviceBufferDeviceAddressFeaturesEXT` structure is included in the `pNext` chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. `VkPhysicalDeviceBufferDeviceAddressFeaturesEXT` **can** also be used in the `pNext` chain of VkDeviceCreateInfo to selectively enable these features.

| | |
|---|---|
| **NOTE** | The `VkPhysicalDeviceBufferDeviceAddressFeaturesEXT` structure has the same members as the `VkPhysicalDeviceBufferDeviceAddressFeatures` structure, but the functionality indicated by the members is expressed differently. The features indicated by the `VkPhysicalDeviceBufferDeviceAddressFeatures` structure requires additional flags to be passed at memory allocation time, and the capture and replay mechanism is built around opaque capture addresses for buffer and memory objects. |

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceBufferDeviceAddressFeaturesEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_BUFFER_DEVICE_ADDRESS_FEATURES_EXT`

The `VkPhysicalDeviceDedicatedAllocationImageAliasingFeaturesNV` structure is defined as:

```
// Provided by VK_NV_dedicated_allocation_image_aliasing
typedef struct VkPhysicalDeviceDedicatedAllocationImageAliasingFeaturesNV {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           dedicatedAllocationImageAliasing;
} VkPhysicalDeviceDedicatedAllocationImageAliasingFeaturesNV;
```

This structure describes the following feature:

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- dedicatedAllocationImageAliasing indicates that the implementation supports aliasing of compatible image objects on a dedicated allocation.

If the VkPhysicalDeviceDedicatedAllocationImageAliasingFeaturesNV structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceDedicatedAllocationImageAliasingFeaturesNV **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceDedicatedAllocationImageAliasingFeaturesNV-sType-sType sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DEDICATED_ALLOCATION_IMAGE_ALIASING_FEATURES_NV

The VkPhysicalDeviceImagelessFramebufferFeatures structure is defined as:

```
typedef struct VkPhysicalDeviceImagelessFramebufferFeatures {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           imagelessFramebuffer;
} VkPhysicalDeviceImagelessFramebufferFeatures;
```

or the equivalent

```
// Provided by VK_KHR_imageless_framebuffer
typedef VkPhysicalDeviceImagelessFramebufferFeatures
VkPhysicalDeviceImagelessFramebufferFeaturesKHR;
```

This structure describes the following feature:

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- imagelessFramebuffer indicates that the implementation supports specifying the image view for attachments at render pass begin time via VkRenderPassAttachmentBeginInfo.

If the VkPhysicalDeviceImagelessFramebufferFeatures structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceImagelessFramebufferFeatures **can** also be used in the pNext chain of

VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceImagelessFramebufferFeatures-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_IMAGELESS_FRAMEBUFFER_FEATURES

The VkPhysicalDeviceFragmentShaderInterlockFeaturesEXT structure is defined as:

```
// Provided by VK_EXT_fragment_shader_interlock
typedef struct VkPhysicalDeviceFragmentShaderInterlockFeaturesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           fragmentShaderSampleInterlock;
    VkBool32           fragmentShaderPixelInterlock;
    VkBool32           fragmentShaderShadingRateInterlock;
} VkPhysicalDeviceFragmentShaderInterlockFeaturesEXT;
```

This structure describes the following features:

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- fragmentShaderSampleInterlock indicates that the implementation supports the FragmentShaderSampleInterlockEXT SPIR-V capability.
- fragmentShaderPixelInterlock indicates that the implementation supports the FragmentShaderPixelInterlockEXT SPIR-V capability.
- fragmentShaderShadingRateInterlock indicates that the implementation supports the FragmentShaderShadingRateInterlockEXT SPIR-V capability.

If the VkPhysicalDeviceFragmentShaderInterlockFeaturesEXT structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceFragmentShaderInterlockFeaturesEXT **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceFragmentShaderInterlockFeaturesEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_FRAGMENT_SHADER_INTERLOCK_FEATURES_EXT

The VkPhysicalDeviceCooperativeMatrixFeaturesNV structure is defined as:

```
// Provided by VK_NV_cooperative_matrix
typedef struct VkPhysicalDeviceCooperativeMatrixFeaturesNV {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           cooperativeMatrix;
    VkBool32           cooperativeMatrixRobustBufferAccess;
} VkPhysicalDeviceCooperativeMatrixFeaturesNV;
```

This structure describes the following features:

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `cooperativeMatrix` indicates that the implementation supports the `CooperativeMatrixNV` SPIR-V capability.
- `cooperativeMatrixRobustBufferAccess` indicates that the implementation supports robust buffer access for SPIR-V `OpCooperativeMatrixLoadNV` and `OpCooperativeMatrixStoreNV` instructions.

If the `VkPhysicalDeviceCooperativeMatrixFeaturesNV` structure is included in the `pNext` chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. `VkPhysicalDeviceCooperativeMatrixFeaturesNV` **can** also be used in the `pNext` chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceCooperativeMatrixFeaturesNV-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_COOPERATIVE_MATRIX_FEATURES_NV`

The `VkPhysicalDeviceCooperativeMatrixFeaturesKHR` structure is defined as:

```
// Provided by VK_KHR_cooperative_matrix
typedef struct VkPhysicalDeviceCooperativeMatrixFeaturesKHR {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           cooperativeMatrix;
    VkBool32           cooperativeMatrixRobustBufferAccess;
} VkPhysicalDeviceCooperativeMatrixFeaturesKHR;
```

This structure describes the following features:

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `cooperativeMatrix` indicates that the implementation supports the `CooperativeMatrixKHR` SPIR-V capability.

- cooperativeMatrixRobustBufferAccess indicates that the implementation supports robust buffer access for SPIR-V OpCooperativeMatrixLoadKHR and OpCooperativeMatrixStoreKHR instructions.

If the VkPhysicalDeviceCooperativeMatrixFeaturesKHR structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceCooperativeMatrixFeaturesKHR **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceCooperativeMatrixFeaturesKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_COOPERATIVE_MATRIX_FEATURES_KHR

The VkPhysicalDeviceYcbcrImageArraysFeaturesEXT structure is defined as:

```
// Provided by VK_EXT_ycbcr_image_arrays
typedef struct VkPhysicalDeviceYcbcrImageArraysFeaturesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           ycbcrImageArrays;
} VkPhysicalDeviceYcbcrImageArraysFeaturesEXT;
```

This structure describes the following feature:

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- ycbcrImageArrays indicates that the implementation supports creating images with a format that requires $Y'C_BC_R$ conversion and has multiple array layers.

If the VkPhysicalDeviceYcbcrImageArraysFeaturesEXT structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceYcbcrImageArraysFeaturesEXT **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceYcbcrImageArraysFeaturesEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_YCBCR_IMAGE_ARRAYS_FEATURES_EXT

The VkPhysicalDeviceShaderSubgroupExtendedTypesFeatures structure is defined as:

```
typedef struct VkPhysicalDeviceShaderSubgroupExtendedTypesFeatures {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           shaderSubgroupExtendedTypes;
} VkPhysicalDeviceShaderSubgroupExtendedTypesFeatures;
```

or the equivalent

```
// Provided by VK_KHR_shader_subgroup_extended_types
typedef VkPhysicalDeviceShaderSubgroupExtendedTypesFeatures
VkPhysicalDeviceShaderSubgroupExtendedTypesFeaturesKHR;
```

This structure describes the following feature:

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `shaderSubgroupExtendedTypes` is a boolean specifying whether subgroup operations can use 8-bit integer, 16-bit integer, 64-bit integer, 16-bit floating-point, and vectors of these types in group operations with subgroup scope, if the implementation supports the types.

If the `VkPhysicalDeviceShaderSubgroupExtendedTypesFeatures` structure is included in the `pNext` chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. `VkPhysicalDeviceShaderSubgroupExtendedTypesFeatures` **can** also be used in the `pNext` chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceShaderSubgroupExtendedTypesFeatures-sType-sType `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_SUBGROUP_EXTENDED_TYPES_FEATURES`

The `VkPhysicalDeviceHostQueryResetFeatures` structure is defined as:

```
typedef struct VkPhysicalDeviceHostQueryResetFeatures {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           hostQueryReset;
} VkPhysicalDeviceHostQueryResetFeatures;
```

or the equivalent

```
// Provided by VK_EXT_host_query_reset
typedef VkPhysicalDeviceHostQueryResetFeatures
VkPhysicalDeviceHostQueryResetFeaturesEXT;
```

This structure describes the following feature:

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `hostQueryReset` indicates that the implementation supports resetting queries from the host with vkResetQueryPoolEXT.

If the `VkPhysicalDeviceHostQueryResetFeatures` structure is included in the `pNext` chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. `VkPhysicalDeviceHostQueryResetFeatures` **can** also be used in the `pNext` chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceHostQueryResetFeatures-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_HOST_QUERY_RESET_FEATURES`

The `VkPhysicalDeviceShaderIntegerFunctions2FeaturesINTEL` structure is defined as:

```
// Provided by VK_INTEL_shader_integer_functions2
typedef struct VkPhysicalDeviceShaderIntegerFunctions2FeaturesINTEL {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           shaderIntegerFunctions2;
} VkPhysicalDeviceShaderIntegerFunctions2FeaturesINTEL;
```

This structure describes the following feature:

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `shaderIntegerFunctions2` indicates that the implementation supports the `IntegerFunctions2INTEL` SPIR-V capability.

If the `VkPhysicalDeviceShaderIntegerFunctions2FeaturesINTELfeatures`. structure is included in the `pNext` chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. `VkPhysicalDeviceShaderIntegerFunctions2FeaturesINTELfeatures`. **can** also be used in the `pNext` chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceShaderIntegerFunctions2FeaturesINTEL-sType-sType `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_INTEGER_FUNCTIONS_2_FEATURES_INTEL`

The `VkPhysicalDeviceCoverageReductionModeFeaturesNV` structure is defined as:

```
// Provided by VK_NV_coverage_reduction_mode
typedef struct VkPhysicalDeviceCoverageReductionModeFeaturesNV {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           coverageReductionMode;
} VkPhysicalDeviceCoverageReductionModeFeaturesNV;
```

This structure describes the following feature:

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `coverageReductionMode` indicates whether the implementation supports coverage reduction modes. See Coverage Reduction.

If the `VkPhysicalDeviceCoverageReductionModeFeaturesNV` structure is included in the `pNext` chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. `VkPhysicalDeviceCoverageReductionModeFeaturesNV` **can** also be used in the `pNext` chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceCoverageReductionModeFeaturesNV-sType-sType `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_COVERAGE_REDUCTION_MODE_FEATURES_NV`

The `VkPhysicalDeviceTimelineSemaphoreFeatures` structure is defined as:

```
typedef struct VkPhysicalDeviceTimelineSemaphoreFeatures {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           timelineSemaphore;
} VkPhysicalDeviceTimelineSemaphoreFeatures;
```

or the equivalent

```
// Provided by VK_KHR_timeline_semaphore
typedef VkPhysicalDeviceTimelineSemaphoreFeatures
VkPhysicalDeviceTimelineSemaphoreFeaturesKHR;
```

This structure describes the following feature:

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `timelineSemaphore` indicates whether semaphores created with a VkSemaphoreType of `VK_SEMAPHORE_TYPE_TIMELINE` are supported.

If the `VkPhysicalDeviceTimelineSemaphoreFeatures` structure is included in the `pNext` chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. `VkPhysicalDeviceTimelineSemaphoreFeatures` **can** also be used in the `pNext` chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceTimelineSemaphoreFeatures-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_TIMELINE_SEMAPHORE_FEATURES`

The `VkPhysicalDeviceExternalMemoryScreenBufferFeaturesQNX` structure is defined as:

```
// Provided by VK_QNX_external_memory_screen_buffer
typedef struct VkPhysicalDeviceExternalMemoryScreenBufferFeaturesQNX {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           screenBufferImport;
} VkPhysicalDeviceExternalMemoryScreenBufferFeaturesQNX;
```

The members of the `VkPhysicalDeviceExternalMemoryScreenBufferFeaturesQNX` structure describe the following features:

- `screenBufferImport` indicates whether QNX Screen buffer import functionality is supported. If `screenBufferImport` is set to `VK_TRUE`, VkDeviceMemory supports importing `_screen_buffer` from applications. In this case, the application is responsible for the resource management of the `_screen_buffer`.

*Table 63. Functionality supported for QNX Screen Buffer features*

| Features | Functionality |
|---|---|
| `screenBufferImport` | VkImportScreenBufferInfoQNX |

| | |
|---|---|
| Always supported[1] | vkGetScreenBufferPropertiesQNX, VkScreenBufferPropertiesQNX, VkScreenBufferFormatPropertiesQNX, VkExternalFormatQNX |

**1**

Functionality in this row is always available.

The Functionality supported for QNX Screen buffer features table summarizes the functionality enabled by the `VkPhysicalDeviceExternalMemoryScreenBufferFeaturesQNX` structure. Each entry in the body of the table summarizes the functionality that **can** be used when the given features are supported and enabled. This summarizes Valid Usage statements that are added elsewhere in this specification.

If the `VkPhysicalDeviceExternalMemoryScreenBufferFeaturesQNX` structure is included in the `pNext` chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. `VkPhysicalDeviceExternalMemoryScreenBufferFeaturesQNX` **can** also be used in the `pNext` chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceExternalMemoryScreenBufferFeaturesQNX-sType-sType `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTERNAL_MEMORY_SCREEN_BUFFER_FEATURES_QNX`

The `VkPhysicalDeviceIndexTypeUint8FeaturesKHR` structure is defined as:

```
// Provided by VK_KHR_index_type_uint8
typedef struct VkPhysicalDeviceIndexTypeUint8FeaturesKHR {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           indexTypeUint8;
} VkPhysicalDeviceIndexTypeUint8FeaturesKHR;
```

or the equivalent

```
// Provided by VK_EXT_index_type_uint8
typedef VkPhysicalDeviceIndexTypeUint8FeaturesKHR
VkPhysicalDeviceIndexTypeUint8FeaturesEXT;
```

This structure describes the following feature:

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.

- indexTypeUint8 indicates that VK_INDEX_TYPE_UINT8_KHR can be used with vkCmdBindIndexBuffer2KHR and vkCmdBindIndexBuffer.

If the VkPhysicalDeviceIndexTypeUint8FeaturesKHR structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceIndexTypeUint8FeaturesKHR **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceIndexTypeUint8FeaturesKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_INDEX_TYPE_UINT8_FEATURES_KHR

The VkPhysicalDevicePrimitiveTopologyListRestartFeaturesEXT structure is defined as:

```
// Provided by VK_EXT_primitive_topology_list_restart
typedef struct VkPhysicalDevicePrimitiveTopologyListRestartFeaturesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           primitiveTopologyListRestart;
    VkBool32           primitiveTopologyPatchListRestart;
} VkPhysicalDevicePrimitiveTopologyListRestartFeaturesEXT;
```

This structure describes the following features:

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- primitiveTopologyListRestart indicates that list type primitives, VK_PRIMITIVE_TOPOLOGY_POINT_LIST, VK_PRIMITIVE_TOPOLOGY_LINE_LIST, VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST, VK_PRIMITIVE_TOPOLOGY_LINE_LIST_WITH_ADJACENCY and VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST_WITH_ADJACENCY, **can** use the primitive restart index value in index buffers.
- primitiveTopologyPatchListRestart indicates that the VK_PRIMITIVE_TOPOLOGY_PATCH_LIST topology **can** use the primitive restart index value in index buffers.

If the VkPhysicalDevicePrimitiveTopologyListRestartFeaturesEXT structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDevicePrimitiveTopologyListRestartFeaturesEXT **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDevicePrimitiveTopologyListRestartFeaturesEXT-sType-sType
  sType **must** be

```
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PRIMITIVE_TOPOLOGY_LIST_RESTART_FEATURES_EXT
```

The VkPhysicalDeviceShaderSMBuiltinsFeaturesNV structure is defined as:

```
// Provided by VK_NV_shader_sm_builtins
typedef struct VkPhysicalDeviceShaderSMBuiltinsFeaturesNV {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           shaderSMBuiltins;
} VkPhysicalDeviceShaderSMBuiltinsFeaturesNV;
```

This structure describes the following feature:

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- shaderSMBuiltins indicates whether the implementation supports the SPIR-V ShaderSMBuiltinsNV capability.

If the VkPhysicalDeviceShaderSMBuiltinsFeaturesNV structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceShaderSMBuiltinsFeaturesNV **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceShaderSMBuiltinsFeaturesNV-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_SM_BUILTINS_FEATURES_NV

The VkPhysicalDeviceSeparateDepthStencilLayoutsFeatures structure is defined as:

```
typedef struct VkPhysicalDeviceSeparateDepthStencilLayoutsFeatures {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           separateDepthStencilLayouts;
} VkPhysicalDeviceSeparateDepthStencilLayoutsFeatures;
```

or the equivalent

```
// Provided by VK_KHR_separate_depth_stencil_layouts
typedef VkPhysicalDeviceSeparateDepthStencilLayoutsFeatures
VkPhysicalDeviceSeparateDepthStencilLayoutsFeaturesKHR;
```

This structure describes the following feature:

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- separateDepthStencilLayouts indicates whether the implementation supports a VkImageMemoryBarrier for a depth/stencil image with only one of VK_IMAGE_ASPECT_DEPTH_BIT or VK_IMAGE_ASPECT_STENCIL_BIT set, and whether VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL, VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL, VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL, or VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL can be used.

If the VkPhysicalDeviceSeparateDepthStencilLayoutsFeatures structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceSeparateDepthStencilLayoutsFeatures **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceSeparateDepthStencilLayoutsFeatures-sType-sType sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SEPARATE_DEPTH_STENCIL_LAYOUTS_FEATURES

The VkPhysicalDevicePipelineExecutablePropertiesFeaturesKHR structure is defined as:

```
// Provided by VK_KHR_pipeline_executable_properties
typedef struct VkPhysicalDevicePipelineExecutablePropertiesFeaturesKHR {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           pipelineExecutableInfo;
} VkPhysicalDevicePipelineExecutablePropertiesFeaturesKHR;
```

This structure describes the following feature:

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- pipelineExecutableInfo indicates that the implementation supports reporting properties and statistics about the pipeline executables associated with a compiled pipeline.

If the VkPhysicalDevicePipelineExecutablePropertiesFeaturesKHR structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDevicePipelineExecutablePropertiesFeaturesKHR **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDevicePipelineExecutablePropertiesFeaturesKHR-sType-sType

sType must be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PIPELINE_EXECUTABLE_PROPERTIES_FEATURES_KHR

The VkPhysicalDeviceShaderDemoteToHelperInvocationFeatures structure is defined as:

```
typedef struct VkPhysicalDeviceShaderDemoteToHelperInvocationFeatures {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           shaderDemoteToHelperInvocation;
} VkPhysicalDeviceShaderDemoteToHelperInvocationFeatures;
```

or the equivalent

```
// Provided by VK_EXT_shader_demote_to_helper_invocation
typedef VkPhysicalDeviceShaderDemoteToHelperInvocationFeatures
VkPhysicalDeviceShaderDemoteToHelperInvocationFeaturesEXT;
```

This structure describes the following feature:

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- shaderDemoteToHelperInvocation indicates whether the implementation supports the SPIR-V DemoteToHelperInvocationEXT capability.

If the VkPhysicalDeviceShaderDemoteToHelperInvocationFeatures structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceShaderDemoteToHelperInvocationFeatures **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceShaderDemoteToHelperInvocationFeatures-sType-sType
  sType must be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_DEMOTE_TO_HELPER_INVOCATION_FEATURES

The VkPhysicalDeviceTexelBufferAlignmentFeaturesEXT structure is defined as:

```
// Provided by VK_EXT_texel_buffer_alignment
typedef struct VkPhysicalDeviceTexelBufferAlignmentFeaturesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           texelBufferAlignment;
} VkPhysicalDeviceTexelBufferAlignmentFeaturesEXT;
```

This structure describes the following feature:

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `texelBufferAlignment` indicates whether the implementation uses more specific alignment requirements advertised in VkPhysicalDeviceTexelBufferAlignmentProperties rather than VkPhysicalDeviceLimits::`minTexelBufferOffsetAlignment`.

If the `VkPhysicalDeviceTexelBufferAlignmentFeaturesEXT` structure is included in the `pNext` chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. `VkPhysicalDeviceTexelBufferAlignmentFeaturesEXT` **can** also be used in the `pNext` chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceTexelBufferAlignmentFeaturesEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_TEXEL_BUFFER_ALIGNMENT_FEATURES_EXT`

The `VkPhysicalDeviceAttachmentFeedbackLoopDynamicStateFeaturesEXT` structure is defined as:

```
// Provided by VK_EXT_attachment_feedback_loop_dynamic_state
typedef struct VkPhysicalDeviceAttachmentFeedbackLoopDynamicStateFeaturesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           attachmentFeedbackLoopDynamicState;
} VkPhysicalDeviceAttachmentFeedbackLoopDynamicStateFeaturesEXT;
```

This structure describes the following features:

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `attachmentFeedbackLoopDynamicState` specifies whether dynamic feedback loops are supported.

If the `VkPhysicalDeviceAttachmentFeedbackLoopDynamicStateFeaturesEXT` structure is included in the `pNext` chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. `VkPhysicalDeviceAttachmentFeedbackLoopDynamicStateFeaturesEXT` **can** also be used in the `pNext` chain

of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceAttachmentFeedbackLoopDynamicStateFeaturesEXT-sType-sType `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_ATTACHMENT_FEEDBACK_LOOP_DYNAMIC_STATE_FEATURES_EXT`

The `VkPhysicalDeviceLegacyVertexAttributesFeaturesEXT` structure is defined as:

```
// Provided by VK_EXT_legacy_vertex_attributes
typedef struct VkPhysicalDeviceLegacyVertexAttributesFeaturesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           legacyVertexAttributes;
} VkPhysicalDeviceLegacyVertexAttributesFeaturesEXT;
```

This structure describes the following features:

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `legacyVertexAttributes` specifies whether compatibility features for vertex attributes are supported when using dynamic vertex input state.

If the `VkPhysicalDeviceLegacyVertexAttributesFeaturesEXT` structure is included in the `pNext` chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. `VkPhysicalDeviceLegacyVertexAttributesFeaturesEXT` **can** also be used in the `pNext` chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceLegacyVertexAttributesFeaturesEXT-sType-sType `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_LEGACY_VERTEX_ATTRIBUTES_FEATURES_EXT`

The `VkPhysicalDeviceTextureCompressionASTCHDRFeatures` structure is defined as:

```
typedef struct VkPhysicalDeviceTextureCompressionASTCHDRFeatures {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           textureCompressionASTC_HDR;
} VkPhysicalDeviceTextureCompressionASTCHDRFeatures;
```

or the equivalent

```
// Provided by VK_EXT_texture_compression_astc_hdr
typedef VkPhysicalDeviceTextureCompressionASTCHDRFeatures
VkPhysicalDeviceTextureCompressionASTCHDRFeaturesEXT;
```

This structure describes the following feature:

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `textureCompressionASTC_HDR` indicates whether all of the ASTC HDR compressed texture formats are supported. If this feature is enabled, then the `VK_FORMAT_FEATURE_SAMPLED_IMAGE_BIT`, `VK_FORMAT_FEATURE_BLIT_SRC_BIT` and `VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT` features **must** be supported in `optimalTilingFeatures` for the following formats:
  - `VK_FORMAT_ASTC_4x4_SFLOAT_BLOCK`
  - `VK_FORMAT_ASTC_5x4_SFLOAT_BLOCK`
  - `VK_FORMAT_ASTC_5x5_SFLOAT_BLOCK`
  - `VK_FORMAT_ASTC_6x5_SFLOAT_BLOCK`
  - `VK_FORMAT_ASTC_6x6_SFLOAT_BLOCK`
  - `VK_FORMAT_ASTC_8x5_SFLOAT_BLOCK`
  - `VK_FORMAT_ASTC_8x6_SFLOAT_BLOCK`
  - `VK_FORMAT_ASTC_8x8_SFLOAT_BLOCK`
  - `VK_FORMAT_ASTC_10x5_SFLOAT_BLOCK`
  - `VK_FORMAT_ASTC_10x6_SFLOAT_BLOCK`
  - `VK_FORMAT_ASTC_10x8_SFLOAT_BLOCK`
  - `VK_FORMAT_ASTC_10x10_SFLOAT_BLOCK`
  - `VK_FORMAT_ASTC_12x10_SFLOAT_BLOCK`
  - `VK_FORMAT_ASTC_12x12_SFLOAT_BLOCK`

  To query for additional properties, or if the feature is not enabled, vkGetPhysicalDeviceFormatProperties and vkGetPhysicalDeviceImageFormatProperties **can** be used to check for supported properties of individual formats as normal.

If the `VkPhysicalDeviceTextureCompressionASTCHDRFeatures` structure is included in the `pNext` chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. `VkPhysicalDeviceTextureCompressionASTCHDRFeatures` **can** also be used in the `pNext` chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceTextureCompressionASTCHDRFeatures-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_TEXTURE_COMPRESSION_ASTC_HDR_FEATURES`

The VkPhysicalDeviceLineRasterizationFeaturesKHR structure is defined as:

```
// Provided by VK_KHR_line_rasterization
typedef struct VkPhysicalDeviceLineRasterizationFeaturesKHR {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           rectangularLines;
    VkBool32           bresenhamLines;
    VkBool32           smoothLines;
    VkBool32           stippledRectangularLines;
    VkBool32           stippledBresenhamLines;
    VkBool32           stippledSmoothLines;
} VkPhysicalDeviceLineRasterizationFeaturesKHR;
```

or the equivalent

```
// Provided by VK_EXT_line_rasterization
typedef VkPhysicalDeviceLineRasterizationFeaturesKHR
VkPhysicalDeviceLineRasterizationFeaturesEXT;
```

This structure describes the following features:

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- rectangularLines indicates whether the implementation supports rectangular line rasterization.
- bresenhamLines indicates whether the implementation supports Bresenham-style line rasterization.
- smoothLines indicates whether the implementation supports smooth line rasterization.
- stippledRectangularLines indicates whether the implementation supports stippled line rasterization with VK_LINE_RASTERIZATION_MODE_RECTANGULAR_KHR lines.
- stippledBresenhamLines indicates whether the implementation supports stippled line rasterization with VK_LINE_RASTERIZATION_MODE_BRESENHAM_KHR lines.
- stippledSmoothLines indicates whether the implementation supports stippled line rasterization with VK_LINE_RASTERIZATION_MODE_RECTANGULAR_SMOOTH_KHR lines.

If the VkPhysicalDeviceLineRasterizationFeaturesKHR structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceLineRasterizationFeaturesKHR **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceLineRasterizationFeaturesKHR-sType-sType

sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_LINE_RASTERIZATION_FEATURES_KHR

The VkPhysicalDeviceSubgroupSizeControlFeatures structure is defined as:

```
typedef struct VkPhysicalDeviceSubgroupSizeControlFeatures {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           subgroupSizeControl;
    VkBool32           computeFullSubgroups;
} VkPhysicalDeviceSubgroupSizeControlFeatures;
```

or the equivalent

```
// Provided by VK_EXT_subgroup_size_control
typedef VkPhysicalDeviceSubgroupSizeControlFeatures
VkPhysicalDeviceSubgroupSizeControlFeaturesEXT;
```

This structure describes the following features:

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- subgroupSizeControl indicates whether the implementation supports controlling shader subgroup sizes via the VK_PIPELINE_SHADER_STAGE_CREATE_ALLOW_VARYING_SUBGROUP_SIZE_BIT flag and the VkPipelineShaderStageRequiredSubgroupSizeCreateInfo structure.
- computeFullSubgroups indicates whether the implementation supports requiring full subgroups in compute , mesh, or task shaders via the VK_PIPELINE_SHADER_STAGE_CREATE_REQUIRE_FULL_SUBGROUPS_BIT flag.

If the VkPhysicalDeviceSubgroupSizeControlFeatures structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceSubgroupSizeControlFeatures **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

**NOTE**

The VkPhysicalDeviceSubgroupSizeControlFeaturesEXT structure was added in version 2 of the VK_EXT_subgroup_size_control extension. Version 1 implementations of this extension will not fill out the features structure but applications may assume that both subgroupSizeControl and computeFullSubgroups are supported if the extension is supported. (See also the Feature Requirements section.) Applications are advised to add a VkPhysicalDeviceSubgroupSizeControlFeaturesEXT structure to the pNext chain of VkDeviceCreateInfo to enable the features regardless of the version of the extension supported by the implementation. If the implementation only supports version 1, it will safely ignore the VkPhysicalDeviceSubgroupSizeControlFeaturesEXT structure.

> Vulkan 1.3 implementations always support the features structure.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceSubgroupSizeControlFeatures-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SUBGROUP_SIZE_CONTROL_FEATURES`

The `VkPhysicalDeviceCoherentMemoryFeaturesAMD` structure is defined as:

```
// Provided by VK_AMD_device_coherent_memory
typedef struct VkPhysicalDeviceCoherentMemoryFeaturesAMD {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           deviceCoherentMemory;
} VkPhysicalDeviceCoherentMemoryFeaturesAMD;
```

This structure describes the following feature:

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `deviceCoherentMemory` indicates that the implementation supports device coherent memory.

If the `VkPhysicalDeviceCoherentMemoryFeaturesAMD` structure is included in the `pNext` chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. `VkPhysicalDeviceCoherentMemoryFeaturesAMD` **can** also be used in the `pNext` chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceCoherentMemoryFeaturesAMD-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_COHERENT_MEMORY_FEATURES_AMD`

The `VkPhysicalDeviceAccelerationStructureFeaturesKHR` structure is defined as:

```
// Provided by VK_KHR_acceleration_structure
typedef struct VkPhysicalDeviceAccelerationStructureFeaturesKHR {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           accelerationStructure;
    VkBool32           accelerationStructureCaptureReplay;
    VkBool32           accelerationStructureIndirectBuild;
    VkBool32           accelerationStructureHostCommands;
    VkBool32           descriptorBindingAccelerationStructureUpdateAfterBind;
} VkPhysicalDeviceAccelerationStructureFeaturesKHR;
```

This structure describes the following features:

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `accelerationStructure` indicates whether the implementation supports the acceleration structure functionality. See Acceleration Structures.
- `accelerationStructureCaptureReplay` indicates whether the implementation supports saving and reusing acceleration structure device addresses, e.g. for trace capture and replay.
- `accelerationStructureIndirectBuild` indicates whether the implementation supports indirect acceleration structure build commands, e.g. vkCmdBuildAccelerationStructuresIndirectKHR.
- `accelerationStructureHostCommands` indicates whether the implementation supports host side acceleration structure commands, e.g. vkBuildAccelerationStructuresKHR, vkCopyAccelerationStructureKHR, vkCopyAccelerationStructureToMemoryKHR, vkCopyMemoryToAccelerationStructureKHR, vkWriteAccelerationStructuresPropertiesKHR.
- `descriptorBindingAccelerationStructureUpdateAfterBind` indicates whether the implementation supports updating acceleration structure descriptors after a set is bound. If this feature is not enabled, `VK_DESCRIPTOR_BINDING_UPDATE_AFTER_BIND_BIT` **must** not be used with `VK_DESCRIPTOR_TYPE_ACCELERATION_STRUCTURE_KHR`.

If the `VkPhysicalDeviceAccelerationStructureFeaturesKHR` structure is included in the `pNext` chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. `VkPhysicalDeviceAccelerationStructureFeaturesKHR` **can** also be used in the `pNext` chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceAccelerationStructureFeaturesKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_ACCELERATION_STRUCTURE_FEATURES_KHR`

The `VkPhysicalDeviceRayTracingPipelineFeaturesKHR` structure is defined as:

```
// Provided by VK_KHR_ray_tracing_pipeline
typedef struct VkPhysicalDeviceRayTracingPipelineFeaturesKHR {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           rayTracingPipeline;
    VkBool32           rayTracingPipelineShaderGroupHandleCaptureReplay;
    VkBool32           rayTracingPipelineShaderGroupHandleCaptureReplayMixed;
    VkBool32           rayTracingPipelineTraceRaysIndirect;
    VkBool32           rayTraversalPrimitiveCulling;
} VkPhysicalDeviceRayTracingPipelineFeaturesKHR;
```

This structure describes the following features:

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `rayTracingPipeline` indicates whether the implementation supports the ray tracing pipeline functionality. See Ray Tracing.
- `rayTracingPipelineShaderGroupHandleCaptureReplay` indicates whether the implementation supports saving and reusing shader group handles, e.g. for trace capture and replay.
- `rayTracingPipelineShaderGroupHandleCaptureReplayMixed` indicates whether the implementation supports reuse of shader group handles being arbitrarily mixed with creation of non-reused shader group handles. If this is `VK_FALSE`, all reused shader group handles **must** be specified before any non-reused handles **may** be created.
- `rayTracingPipelineTraceRaysIndirect` indicates whether the implementation supports indirect ray tracing commands, e.g. vkCmdTraceRaysIndirectKHR.
- `rayTraversalPrimitiveCulling` indicates whether the implementation supports primitive culling during ray traversal.

If the `VkPhysicalDeviceRayTracingPipelineFeaturesKHR` structure is included in the `pNext` chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. `VkPhysicalDeviceRayTracingPipelineFeaturesKHR` **can** also be used in the `pNext` chain of VkDeviceCreateInfo to selectively enable these features.

## Valid Usage

- VUID-VkPhysicalDeviceRayTracingPipelineFeaturesKHR-rayTracingPipelineShaderGroupHandleCaptureReplayMixed-03575
  If `rayTracingPipelineShaderGroupHandleCaptureReplayMixed` is `VK_TRUE`, `rayTracingPipelineShaderGroupHandleCaptureReplay` **must** also be `VK_TRUE`

## Valid Usage (Implicit)

- VUID-VkPhysicalDeviceRayTracingPipelineFeaturesKHR-sType-sType

sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_RAY_TRACING_PIPELINE_FEATURES_KHR

The VkPhysicalDeviceRayQueryFeaturesKHR structure is defined as:

```
// Provided by VK_KHR_ray_query
typedef struct VkPhysicalDeviceRayQueryFeaturesKHR {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           rayQuery;
} VkPhysicalDeviceRayQueryFeaturesKHR;
```

This structure describes the following feature:

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- rayQuery indicates whether the implementation supports ray query (OpRayQueryProceedKHR) functionality.

If the VkPhysicalDeviceRayQueryFeaturesKHR structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceRayQueryFeaturesKHR **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceRayQueryFeaturesKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_RAY_QUERY_FEATURES_KHR

The VkPhysicalDeviceRayTracingMaintenance1FeaturesKHR structure is defined as:

```
// Provided by VK_KHR_ray_tracing_maintenance1
typedef struct VkPhysicalDeviceRayTracingMaintenance1FeaturesKHR {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           rayTracingMaintenance1;
    VkBool32           rayTracingPipelineTraceRaysIndirect2;
} VkPhysicalDeviceRayTracingMaintenance1FeaturesKHR;
```

This structure describes the following features:

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- rayTracingMaintenance1 indicates that the implementation supports the following:
  - The CullMaskKHR SPIR-V builtin using the SPV_KHR_ray_cull_mask SPIR-V extension.

- Additional acceleration structure property queries: VK_QUERY_TYPE_ACCELERATION_STRUCTURE_SERIALIZATION_BOTTOM_LEVEL_POINTERS_KHR and VK_QUERY_TYPE_ACCELERATION_STRUCTURE_SIZE_KHR.
- A new access flag VK_ACCESS_2_SHADER_BINDING_TABLE_READ_BIT_KHR.
- A new pipeline stage flag bit VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_COPY_BIT_KHR

- rayTracingPipelineTraceRaysIndirect2 indicates whether the implementation supports the extended indirect ray tracing command vkCmdTraceRaysIndirect2KHR.

If the VkPhysicalDeviceRayTracingMaintenance1FeaturesKHR structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceRayTracingMaintenance1FeaturesKHR **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceRayTracingMaintenance1FeaturesKHR-sType-sType sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_RAY_TRACING_MAINTENANCE_1_FEATURES_KHR

The VkPhysicalDeviceVideoMaintenance1FeaturesKHR structure is defined as:

```
// Provided by VK_KHR_video_maintenance1
typedef struct VkPhysicalDeviceVideoMaintenance1FeaturesKHR {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           videoMaintenance1;
} VkPhysicalDeviceVideoMaintenance1FeaturesKHR;
```

This structure describes the following features:

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- videoMaintenance1 indicates that the implementation supports the following:
  - The new buffer creation flag VK_BUFFER_CREATE_VIDEO_PROFILE_INDEPENDENT_BIT_KHR.
  - The new image creation flag VK_IMAGE_CREATE_VIDEO_PROFILE_INDEPENDENT_BIT_KHR.
  - The new video session creation flag VK_VIDEO_SESSION_CREATE_INLINE_QUERIES_BIT_KHR.

If the VkPhysicalDeviceVideoMaintenance1FeaturesKHR structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceVideoMaintenance1FeaturesKHR **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceVideoMaintenance1FeaturesKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_VIDEO_MAINTENANCE_1_FEATURES_KHR`

The `VkPhysicalDeviceExtendedDynamicStateFeaturesEXT` structure is defined as:

```
// Provided by VK_EXT_extended_dynamic_state
typedef struct VkPhysicalDeviceExtendedDynamicStateFeaturesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           extendedDynamicState;
} VkPhysicalDeviceExtendedDynamicStateFeaturesEXT;
```

This structure describes the following feature:

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `extendedDynamicState` indicates that the implementation supports the following dynamic states:
  - `VK_DYNAMIC_STATE_CULL_MODE`
  - `VK_DYNAMIC_STATE_FRONT_FACE`
  - `VK_DYNAMIC_STATE_PRIMITIVE_TOPOLOGY`
  - `VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT`
  - `VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT`
  - `VK_DYNAMIC_STATE_VERTEX_INPUT_BINDING_STRIDE`
  - `VK_DYNAMIC_STATE_DEPTH_TEST_ENABLE`
  - `VK_DYNAMIC_STATE_DEPTH_WRITE_ENABLE`
  - `VK_DYNAMIC_STATE_DEPTH_COMPARE_OP`
  - `VK_DYNAMIC_STATE_DEPTH_BOUNDS_TEST_ENABLE`
  - `VK_DYNAMIC_STATE_STENCIL_TEST_ENABLE`
  - `VK_DYNAMIC_STATE_STENCIL_OP`

If the `VkPhysicalDeviceExtendedDynamicStateFeaturesEXT` structure is included in the `pNext` chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. `VkPhysicalDeviceExtendedDynamicStateFeaturesEXT` **can** also be used in the `pNext` chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceExtendedDynamicStateFeaturesEXT-sType-sType

sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTENDED_DYNAMIC_STATE_FEATURES_EXT

The VkPhysicalDeviceExtendedDynamicState2FeaturesEXT structure is defined as:

```
// Provided by VK_EXT_extended_dynamic_state2
typedef struct VkPhysicalDeviceExtendedDynamicState2FeaturesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           extendedDynamicState2;
    VkBool32           extendedDynamicState2LogicOp;
    VkBool32           extendedDynamicState2PatchControlPoints;
} VkPhysicalDeviceExtendedDynamicState2FeaturesEXT;
```

This structure describes the following features:

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- extendedDynamicState2 indicates that the implementation supports the following dynamic states:
  - VK_DYNAMIC_STATE_DEPTH_BIAS_ENABLE
  - VK_DYNAMIC_STATE_PRIMITIVE_RESTART_ENABLE
  - VK_DYNAMIC_STATE_RASTERIZER_DISCARD_ENABLE
- extendedDynamicState2LogicOp indicates that the implementation supports the following dynamic state:
  - VK_DYNAMIC_STATE_LOGIC_OP_EXT
- extendedDynamicState2PatchControlPoints indicates that the implementation supports the following dynamic state:
  - VK_DYNAMIC_STATE_PATCH_CONTROL_POINTS_EXT

If the VkPhysicalDeviceExtendedDynamicState2FeaturesEXT structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceExtendedDynamicState2FeaturesEXT **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceExtendedDynamicState2FeaturesEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTENDED_DYNAMIC_STATE_2_FEATURES_EXT

The VkPhysicalDeviceExtendedDynamicState3FeaturesEXT structure is defined as:

```
// Provided by VK_EXT_extended_dynamic_state3
typedef struct VkPhysicalDeviceExtendedDynamicState3FeaturesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           extendedDynamicState3TessellationDomainOrigin;
    VkBool32           extendedDynamicState3DepthClampEnable;
    VkBool32           extendedDynamicState3PolygonMode;
    VkBool32           extendedDynamicState3RasterizationSamples;
    VkBool32           extendedDynamicState3SampleMask;
    VkBool32           extendedDynamicState3AlphaToCoverageEnable;
    VkBool32           extendedDynamicState3AlphaToOneEnable;
    VkBool32           extendedDynamicState3LogicOpEnable;
    VkBool32           extendedDynamicState3ColorBlendEnable;
    VkBool32           extendedDynamicState3ColorBlendEquation;
    VkBool32           extendedDynamicState3ColorWriteMask;
    VkBool32           extendedDynamicState3RasterizationStream;
    VkBool32           extendedDynamicState3ConservativeRasterizationMode;
    VkBool32           extendedDynamicState3ExtraPrimitiveOverestimationSize;
    VkBool32           extendedDynamicState3DepthClipEnable;
    VkBool32           extendedDynamicState3SampleLocationsEnable;
    VkBool32           extendedDynamicState3ColorBlendAdvanced;
    VkBool32           extendedDynamicState3ProvokingVertexMode;
    VkBool32           extendedDynamicState3LineRasterizationMode;
    VkBool32           extendedDynamicState3LineStippleEnable;
    VkBool32           extendedDynamicState3DepthClipNegativeOneToOne;
    VkBool32           extendedDynamicState3ViewportWScalingEnable;
    VkBool32           extendedDynamicState3ViewportSwizzle;
    VkBool32           extendedDynamicState3CoverageToColorEnable;
    VkBool32           extendedDynamicState3CoverageToColorLocation;
    VkBool32           extendedDynamicState3CoverageModulationMode;
    VkBool32           extendedDynamicState3CoverageModulationTableEnable;
    VkBool32           extendedDynamicState3CoverageModulationTable;
    VkBool32           extendedDynamicState3CoverageReductionMode;
    VkBool32           extendedDynamicState3RepresentativeFragmentTestEnable;
    VkBool32           extendedDynamicState3ShadingRateImageEnable;
} VkPhysicalDeviceExtendedDynamicState3FeaturesEXT;
```

This structure describes the following features:

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `extendedDynamicState3TessellationDomainOrigin` indicates that the implementation supports the following dynamic state:
  - `VK_DYNAMIC_STATE_TESSELLATION_DOMAIN_ORIGIN_EXT`
- `extendedDynamicState3DepthClampEnable` indicates that the implementation supports the following dynamic state:
  - `VK_DYNAMIC_STATE_DEPTH_CLAMP_ENABLE_EXT`

- `extendedDynamicState3PolygonMode` indicates that the implementation supports the following dynamic state:
  - `VK_DYNAMIC_STATE_POLYGON_MODE_EXT`
- `extendedDynamicState3RasterizationSamples` indicates that the implementation supports the following dynamic state:
  - `VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT`
- `extendedDynamicState3SampleMask` indicates that the implementation supports the following dynamic state:
  - `VK_DYNAMIC_STATE_SAMPLE_MASK_EXT`
- `extendedDynamicState3AlphaToCoverageEnable` indicates that the implementation supports the following dynamic state:
  - `VK_DYNAMIC_STATE_ALPHA_TO_COVERAGE_ENABLE_EXT`
- `extendedDynamicState3AlphaToOneEnable` indicates that the implementation supports the following dynamic state:
  - `VK_DYNAMIC_STATE_ALPHA_TO_ONE_ENABLE_EXT`
- `extendedDynamicState3LogicOpEnable` indicates that the implementation supports the following dynamic state:
  - `VK_DYNAMIC_STATE_LOGIC_OP_ENABLE_EXT`
- `extendedDynamicState3ColorBlendEnable` indicates that the implementation supports the following dynamic state:
  - `VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT`
- `extendedDynamicState3ColorBlendEquation` indicates that the implementation supports the following dynamic state:
  - `VK_DYNAMIC_STATE_COLOR_BLEND_EQUATION_EXT`
- `extendedDynamicState3ColorWriteMask` indicates that the implementation supports the following dynamic state:
  - `VK_DYNAMIC_STATE_COLOR_WRITE_MASK_EXT`
- `extendedDynamicState3RasterizationStream` indicates that the implementation supports the following dynamic state:
  - `VK_DYNAMIC_STATE_RASTERIZATION_STREAM_EXT`
- `extendedDynamicState3ConservativeRasterizationMode` indicates that the implementation supports the following dynamic state:
  - `VK_DYNAMIC_STATE_CONSERVATIVE_RASTERIZATION_MODE_EXT`
- `extendedDynamicState3ExtraPrimitiveOverestimationSize` indicates that the implementation supports the following dynamic state:
  - `VK_DYNAMIC_STATE_EXTRA_PRIMITIVE_OVERESTIMATION_SIZE_EXT`
- `extendedDynamicState3DepthClipEnable` indicates that the implementation supports the following dynamic state:

  - VK_DYNAMIC_STATE_DEPTH_CLIP_ENABLE_EXT
- extendedDynamicState3SampleLocationsEnable indicates that the implementation supports the following dynamic state:
  - VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT
- extendedDynamicState3ColorBlendAdvanced indicates that the implementation supports the following dynamic state:
  - VK_DYNAMIC_STATE_COLOR_BLEND_ADVANCED_EXT
- extendedDynamicState3ProvokingVertexMode indicates that the implementation supports the following dynamic state:
  - VK_DYNAMIC_STATE_PROVOKING_VERTEX_MODE_EXT
- extendedDynamicState3LineRasterizationMode indicates that the implementation supports the following dynamic state:
  - VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT
- extendedDynamicState3LineStippleEnable indicates that the implementation supports the following dynamic state:
  - VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT
- extendedDynamicState3DepthClipNegativeOneToOne indicates that the implementation supports the following dynamic state:
  - VK_DYNAMIC_STATE_DEPTH_CLIP_NEGATIVE_ONE_TO_ONE_EXT
- extendedDynamicState3ViewportWScalingEnable indicates that the implementation supports the following dynamic state:
  - VK_DYNAMIC_STATE_VIEWPORT_W_SCALING_ENABLE_NV
- extendedDynamicState3ViewportSwizzle indicates that the implementation supports the following dynamic state:
  - VK_DYNAMIC_STATE_VIEWPORT_SWIZZLE_NV
- extendedDynamicState3CoverageToColorEnable indicates that the implementation supports the following dynamic state:
  - VK_DYNAMIC_STATE_COVERAGE_TO_COLOR_ENABLE_NV
- extendedDynamicState3CoverageToColorLocation indicates that the implementation supports the following dynamic state:
  - VK_DYNAMIC_STATE_COVERAGE_TO_COLOR_LOCATION_NV
- extendedDynamicState3CoverageModulationMode indicates that the implementation supports the following dynamic state:
  - VK_DYNAMIC_STATE_COVERAGE_MODULATION_MODE_NV
- extendedDynamicState3CoverageModulationTableEnable indicates that the implementation supports the following dynamic state:
  - VK_DYNAMIC_STATE_COVERAGE_MODULATION_TABLE_ENABLE_NV
- extendedDynamicState3CoverageModulationTable indicates that the implementation supports the

following dynamic state:

  - VK_DYNAMIC_STATE_COVERAGE_MODULATION_TABLE_NV

- extendedDynamicState3CoverageReductionMode indicates that the implementation supports the following dynamic state:

  - VK_DYNAMIC_STATE_COVERAGE_REDUCTION_MODE_NV

- extendedDynamicState3RepresentativeFragmentTestEnable indicates that the implementation supports the following dynamic state:

  - VK_DYNAMIC_STATE_REPRESENTATIVE_FRAGMENT_TEST_ENABLE_NV

- extendedDynamicState3ShadingRateImageEnable indicates that the implementation supports the following dynamic state:

  - VK_DYNAMIC_STATE_SHADING_RATE_IMAGE_ENABLE_NV

If the VkPhysicalDeviceExtendedDynamicState3FeaturesEXT structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceExtendedDynamicState3FeaturesEXT **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceExtendedDynamicState3FeaturesEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTENDED_DYNAMIC_STATE_3_FEATURES_EXT

The VkPhysicalDeviceDeviceGeneratedCommandsFeaturesNV structure is defined as:

```
// Provided by VK_NV_device_generated_commands
typedef struct VkPhysicalDeviceDeviceGeneratedCommandsFeaturesNV {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           deviceGeneratedCommands;
} VkPhysicalDeviceDeviceGeneratedCommandsFeaturesNV;
```

This structure describes the following feature:

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- deviceGeneratedCommands indicates whether the implementation supports functionality to generate commands on the device. See Device-Generated Commands.

If the VkPhysicalDeviceDeviceGeneratedCommandsFeaturesNV structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceDeviceGeneratedCommandsFeaturesNV **can** also be used in the pNext chain of

VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceDeviceGeneratedCommandsFeaturesNV-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DEVICE_GENERATED_COMMANDS_FEATURES_NV

The VkPhysicalDeviceDeviceGeneratedCommandsComputeFeaturesNV structure is defined as:

```
// Provided by VK_NV_device_generated_commands_compute
typedef struct VkPhysicalDeviceDeviceGeneratedCommandsComputeFeaturesNV {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           deviceGeneratedCompute;
    VkBool32           deviceGeneratedComputePipelines;
    VkBool32           deviceGeneratedComputeCaptureReplay;
} VkPhysicalDeviceDeviceGeneratedCommandsComputeFeaturesNV;
```

This structure describes the following features:

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- deviceGeneratedCompute indicates whether the implementation supports functionality to generate dispatch commands and push constants for the compute pipeline on the device. See Device-Generated Commands.
- deviceGeneratedComputePipelines indicates whether the implementation supports functionality to generate commands to bind compute pipelines on the device. See Device-Generated Commands.
- deviceGeneratedComputeCaptureReplay indicates whether the implementation supports functionality to capture compute pipeline address and reuse later for replay in Device-Generated Commands.

If the VkPhysicalDeviceDeviceGeneratedCommandsComputeFeaturesNV structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceDeviceGeneratedCommandsComputeFeaturesNV **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceDeviceGeneratedCommandsComputeFeaturesNV-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DEVICE_GENERATED_COMMANDS_COMPUTE_FEATURES_NV

The VkPhysicalDeviceDiagnosticsConfigFeaturesNV structure is defined as:

```
// Provided by VK_NV_device_diagnostics_config
typedef struct VkPhysicalDeviceDiagnosticsConfigFeaturesNV {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           diagnosticsConfig;
} VkPhysicalDeviceDiagnosticsConfigFeaturesNV;
```

This structure describes the following feature:

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- diagnosticsConfig indicates whether the implementation supports the ability to configure diagnostic tools.

If the VkPhysicalDeviceDiagnosticsConfigFeaturesNV structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceDiagnosticsConfigFeaturesNV **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceDiagnosticsConfigFeaturesNV-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DIAGNOSTICS_CONFIG_FEATURES_NV

The VkPhysicalDeviceDeviceMemoryReportFeaturesEXT structure is defined as:

```
// Provided by VK_EXT_device_memory_report
typedef struct VkPhysicalDeviceDeviceMemoryReportFeaturesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           deviceMemoryReport;
} VkPhysicalDeviceDeviceMemoryReportFeaturesEXT;
```

This structure describes the following feature:

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- deviceMemoryReport indicates whether the implementation supports the ability to register device memory report callbacks.

If the VkPhysicalDeviceDeviceMemoryReportFeaturesEXT structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to

indicate whether each corresponding feature is supported. VkPhysicalDeviceDeviceMemoryReportFeaturesEXT **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceDeviceMemoryReportFeaturesEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DEVICE_MEMORY_REPORT_FEATURES_EXT

The VkPhysicalDeviceGlobalPriorityQueryFeaturesKHR structure is defined as:

```
// Provided by VK_KHR_global_priority
typedef struct VkPhysicalDeviceGlobalPriorityQueryFeaturesKHR {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           globalPriorityQuery;
} VkPhysicalDeviceGlobalPriorityQueryFeaturesKHR;
```

or the equivalent

```
// Provided by VK_EXT_global_priority_query
typedef VkPhysicalDeviceGlobalPriorityQueryFeaturesKHR
VkPhysicalDeviceGlobalPriorityQueryFeaturesEXT;
```

This structure describes the following feature:

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- globalPriorityQuery indicates whether the implementation supports the ability to query global queue priorities.

If the VkPhysicalDeviceGlobalPriorityQueryFeaturesKHR structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceGlobalPriorityQueryFeaturesKHR **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceGlobalPriorityQueryFeaturesKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_GLOBAL_PRIORITY_QUERY_FEATURES_KHR

The VkPhysicalDevicePipelineCreationCacheControlFeatures structure is defined as:

```
typedef struct VkPhysicalDevicePipelineCreationCacheControlFeatures {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           pipelineCreationCacheControl;
} VkPhysicalDevicePipelineCreationCacheControlFeatures;
```

or the equivalent

```
// Provided by VK_EXT_pipeline_creation_cache_control
typedef VkPhysicalDevicePipelineCreationCacheControlFeatures
VkPhysicalDevicePipelineCreationCacheControlFeaturesEXT;
```

This structure describes the following feature:

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `pipelineCreationCacheControl` indicates that the implementation supports:
  - The following **can** be used in `Vk*PipelineCreateInfo::flags`:
    - `VK_PIPELINE_CREATE_FAIL_ON_PIPELINE_COMPILE_REQUIRED_BIT`
    - `VK_PIPELINE_CREATE_EARLY_RETURN_ON_FAILURE_BIT`
  - The following **can** be used in VkPipelineCacheCreateInfo::`flags`:
    - `VK_PIPELINE_CACHE_CREATE_EXTERNALLY_SYNCHRONIZED_BIT`

If the `VkPhysicalDevicePipelineCreationCacheControlFeatures` structure is included in the `pNext` chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. `VkPhysicalDevicePipelineCreationCacheControlFeatures` **can** also be used in the `pNext` chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDevicePipelineCreationCacheControlFeatures-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PIPELINE_CREATION_CACHE_CONTROL_FEATURES`

The `VkPhysicalDeviceZeroInitializeWorkgroupMemoryFeatures` structure is defined as:

```
typedef struct VkPhysicalDeviceZeroInitializeWorkgroupMemoryFeatures {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           shaderZeroInitializeWorkgroupMemory;
} VkPhysicalDeviceZeroInitializeWorkgroupMemoryFeatures;
```

or the equivalent

```
// Provided by VK_KHR_zero_initialize_workgroup_memory
typedef VkPhysicalDeviceZeroInitializeWorkgroupMemoryFeatures
VkPhysicalDeviceZeroInitializeWorkgroupMemoryFeaturesKHR;
```

This structure describes the following feature:

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `shaderZeroInitializeWorkgroupMemory` specifies whether the implementation supports initializing a variable in Workgroup storage class.

If the `VkPhysicalDeviceZeroInitializeWorkgroupMemoryFeatures` structure is included in the `pNext` chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. `VkPhysicalDeviceZeroInitializeWorkgroupMemoryFeatures` **can** also be used in the `pNext` chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceZeroInitializeWorkgroupMemoryFeatures-sType-sType `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_ZERO_INITIALIZE_WORKGROUP_MEMORY_FEATURES`

The `VkPhysicalDevicePrivateDataFeatures` structure is defined as:

```
typedef struct VkPhysicalDevicePrivateDataFeatures {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           privateData;
} VkPhysicalDevicePrivateDataFeatures;
```

or the equivalent

```
// Provided by VK_EXT_private_data
typedef VkPhysicalDevicePrivateDataFeatures VkPhysicalDevicePrivateDataFeaturesEXT;
```

This structure describes the following feature:

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `privateData` indicates whether the implementation supports private data. See Private Data.

If the VkPhysicalDevicePrivateDataFeatures structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDevicePrivateDataFeatures **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDevicePrivateDataFeatures-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PRIVATE_DATA_FEATURES

The VkPhysicalDeviceShaderSubgroupUniformControlFlowFeaturesKHR structure is defined as:

```
// Provided by VK_KHR_shader_subgroup_uniform_control_flow
typedef struct VkPhysicalDeviceShaderSubgroupUniformControlFlowFeaturesKHR {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           shaderSubgroupUniformControlFlow;
} VkPhysicalDeviceShaderSubgroupUniformControlFlowFeaturesKHR;
```

This structure describes the following feature:

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- shaderSubgroupUniformControlFlow specifies whether the implementation supports the shader execution mode SubgroupUniformControlFlowKHR

If the VkPhysicalDeviceShaderSubgroupUniformControlFlowFeaturesKHR structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceShaderSubgroupUniformControlFlowFeaturesKHR **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceShaderSubgroupUniformControlFlowFeaturesKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_SUBGROUP_UNIFORM_CONTROL_FLOW_FEATURES_KHR

The VkPhysicalDeviceRobustness2FeaturesEXT structure is defined as:

```
// Provided by VK_EXT_robustness2
typedef struct VkPhysicalDeviceRobustness2FeaturesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           robustBufferAccess2;
    VkBool32           robustImageAccess2;
    VkBool32           nullDescriptor;
} VkPhysicalDeviceRobustness2FeaturesEXT;
```

This structure describes the following features:

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `robustBufferAccess2` indicates whether buffer accesses are tightly bounds-checked against the range of the descriptor. Uniform buffers **must** be bounds-checked to the range of the descriptor, where the range is rounded up to a multiple of `robustUniformBufferAccessSizeAlignment`. Storage buffers **must** be bounds-checked to the range of the descriptor, where the range is rounded up to a multiple of `robustStorageBufferAccessSizeAlignment`. Out of bounds buffer loads will return zero values, and image load, sample, and atomic operations from texel buffers will have (0,0,1) values inserted for missing G, B, or A components based on the format.
- `robustImageAccess2` indicates whether image accesses are tightly bounds-checked against the dimensions of the image view. Out of bounds image load, sample, and atomic operations from images will return zero values, with (0,0,1) values inserted for missing G, B, or A components based on the format.
- `nullDescriptor` indicates whether descriptors **can** be written with a VK_NULL_HANDLE resource or view, which are considered valid to access and act as if the descriptor were bound to nothing.

If the `VkPhysicalDeviceRobustness2FeaturesEXT` structure is included in the `pNext` chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. `VkPhysicalDeviceRobustness2FeaturesEXT` **can** also be used in the `pNext` chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage

- VUID-VkPhysicalDeviceRobustness2FeaturesEXT-robustBufferAccess2-04000
  If `robustBufferAccess2` is enabled then `robustBufferAccess` **must** also be enabled

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceRobustness2FeaturesEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_ROBUSTNESS_2_FEATURES_EXT`

The `VkPhysicalDeviceImageRobustnessFeatures` structure is defined as:

```
typedef struct VkPhysicalDeviceImageRobustnessFeatures {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           robustImageAccess;
} VkPhysicalDeviceImageRobustnessFeatures;
```

or the equivalent

```
// Provided by VK_EXT_image_robustness
typedef VkPhysicalDeviceImageRobustnessFeatures
VkPhysicalDeviceImageRobustnessFeaturesEXT;
```

This structure describes the following feature:

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `robustImageAccess` indicates whether image accesses are tightly bounds-checked against the dimensions of the image view. Invalid texels resulting from out of bounds image loads will be replaced as described in Texel Replacement, with either (0,0,1) or (0,0,0) values inserted for missing G, B, or A components based on the format.

If the `VkPhysicalDeviceImageRobustnessFeatures` structure is included in the `pNext` chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. `VkPhysicalDeviceImageRobustnessFeatures` **can** also be used in the `pNext` chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceImageRobustnessFeatures-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_IMAGE_ROBUSTNESS_FEATURES`

The `VkPhysicalDeviceShaderTerminateInvocationFeatures` structure is defined as:

```
typedef struct VkPhysicalDeviceShaderTerminateInvocationFeatures {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           shaderTerminateInvocation;
} VkPhysicalDeviceShaderTerminateInvocationFeatures;
```

or the equivalent

```
// Provided by VK_KHR_shader_terminate_invocation
typedef VkPhysicalDeviceShaderTerminateInvocationFeatures
VkPhysicalDeviceShaderTerminateInvocationFeaturesKHR;
```

This structure describes the following feature:

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `shaderTerminateInvocation` specifies whether the implementation supports SPIR-V modules that use the `SPV_KHR_terminate_invocation` extension.

If the `VkPhysicalDeviceShaderTerminateInvocationFeatures` structure is included in the `pNext` chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. `VkPhysicalDeviceShaderTerminateInvocationFeatures` **can** also be used in the `pNext` chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceShaderTerminateInvocationFeatures-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_TERMINATE_INVOCATION_FEATURES`

The `VkPhysicalDeviceCustomBorderColorFeaturesEXT` structure is defined as:

```
// Provided by VK_EXT_custom_border_color
typedef struct VkPhysicalDeviceCustomBorderColorFeaturesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           customBorderColors;
    VkBool32           customBorderColorWithoutFormat;
} VkPhysicalDeviceCustomBorderColorFeaturesEXT;
```

This structure describes the following features:

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `customBorderColors` indicates that the implementation supports providing a `borderColor` value with one of the following values at sampler creation time:
  - `VK_BORDER_COLOR_FLOAT_CUSTOM_EXT`
  - `VK_BORDER_COLOR_INT_CUSTOM_EXT`
- `customBorderColorWithoutFormat` indicates that explicit formats are not required for custom border colors and the value of the `format` member of the VkSamplerCustomBorderColorCreateInfoEXT structure **may** be `VK_FORMAT_UNDEFINED`. If this

feature bit is not set, applications **must** provide the VkFormat of the image view(s) being sampled by this sampler in the `format` member of the VkSamplerCustomBorderColorCreateInfoEXT structure.

If the `VkPhysicalDeviceCustomBorderColorFeaturesEXT` structure is included in the `pNext` chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. `VkPhysicalDeviceCustomBorderColorFeaturesEXT` **can** also be used in the `pNext` chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceCustomBorderColorFeaturesEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_CUSTOM_BORDER_COLOR_FEATURES_EXT`

The `VkPhysicalDeviceBorderColorSwizzleFeaturesEXT` structure is defined as:

```
// Provided by VK_EXT_border_color_swizzle
typedef struct VkPhysicalDeviceBorderColorSwizzleFeaturesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           borderColorSwizzle;
    VkBool32           borderColorSwizzleFromImage;
} VkPhysicalDeviceBorderColorSwizzleFeaturesEXT;
```

This structure describes the following features:

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `borderColorSwizzle` indicates that defined values are returned by sampled image operations when used with a sampler that uses a `VK_BORDER_COLOR_FLOAT_OPAQUE_BLACK`, `VK_BORDER_COLOR_INT_OPAQUE_BLACK`, `VK_BORDER_COLOR_FLOAT_CUSTOM_EXT`, or `VK_BORDER_COLOR_INT_CUSTOM_EXT` `borderColor` and an image view that uses a non-identity component mapping, when either `borderColorSwizzleFromImage` is enabled or the VkSamplerBorderColorComponentMappingCreateInfoEXT is specified.
- `borderColorSwizzleFromImage` indicates that the implementation will return the correct border color values from sampled image operations under the conditions expressed above, without the application having to specify the border color component mapping when creating the sampler object. If this feature bit is not set, applications **can** chain a VkSamplerBorderColorComponentMappingCreateInfoEXT structure when creating samplers for use with image views that do not have an identity swizzle and, when those samplers are combined with image views using the same component mapping, sampled image operations that use opaque black or custom border colors will return the correct border color values.

If the `VkPhysicalDeviceBorderColorSwizzleFeaturesEXT` structure is included in the `pNext` chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to

indicate whether each corresponding feature is supported. VkPhysicalDeviceBorderColorSwizzleFeaturesEXT **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceBorderColorSwizzleFeaturesEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_BORDER_COLOR_SWIZZLE_FEATURES_EXT

The VkPhysicalDevicePortabilitySubsetFeaturesKHR structure is defined as:

```
// Provided by VK_KHR_portability_subset
typedef struct VkPhysicalDevicePortabilitySubsetFeaturesKHR {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           constantAlphaColorBlendFactors;
    VkBool32           events;
    VkBool32           imageViewFormatReinterpretation;
    VkBool32           imageViewFormatSwizzle;
    VkBool32           imageView2DOn3DImage;
    VkBool32           multisampleArrayImage;
    VkBool32           mutableComparisonSamplers;
    VkBool32           pointPolygons;
    VkBool32           samplerMipLodBias;
    VkBool32           separateStencilMaskRef;
    VkBool32           shaderSampleRateInterpolationFunctions;
    VkBool32           tessellationIsolines;
    VkBool32           tessellationPointMode;
    VkBool32           triangleFans;
    VkBool32           vertexAttributeAccessBeyondStride;
} VkPhysicalDevicePortabilitySubsetFeaturesKHR;
```

This structure describes the following features:

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- constantAlphaColorBlendFactors indicates whether this implementation supports constant *alpha* Blend Factors used as source or destination *color* Blending.
- events indicates whether this implementation supports synchronization using Events.
- imageViewFormatReinterpretation indicates whether this implementation supports a VkImageView being created with a texel format containing a different number of components, or a different number of bits in each component, than the texel format of the underlying VkImage.
- imageViewFormatSwizzle indicates whether this implementation supports remapping format components using VkImageViewCreateInfo::components.
- imageView2DOn3DImage indicates whether this implementation supports a VkImage being created

with the VK_IMAGE_CREATE_2D_ARRAY_COMPATIBLE_BIT flag set, permitting a 2D or 2D array image view to be created on a 3D VkImage.

- multisampleArrayImage indicates whether this implementation supports a VkImage being created as a 2D array with multiple samples per texel.
- mutableComparisonSamplers indicates whether this implementation allows descriptors with comparison samplers to be updated.
- pointPolygons indicates whether this implementation supports Rasterization using a *point* Polygon Mode.
- samplerMipLodBias indicates whether this implementation supports setting a mipmap LOD bias value when creating a sampler.
- separateStencilMaskRef indicates whether this implementation supports separate front and back Stencil Test reference values.
- shaderSampleRateInterpolationFunctions indicates whether this implementation supports fragment shaders which use the InterpolationFunction capability and the extended instructions InterpolateAtCentroid, InterpolateAtOffset, and InterpolateAtSample from the GLSL.std.450 extended instruction set. This member is only meaningful if the sampleRateShading feature is supported.
- tessellationIsolines indicates whether this implementation supports isoline output from the Tessellation stage of a graphics pipeline. This member is only meaningful if tessellationShader are supported.
- tessellationPointMode indicates whether this implementation supports point output from the Tessellation stage of a graphics pipeline. This member is only meaningful if tessellationShader are supported.
- triangleFans indicates whether this implementation supports Triangle Fans primitive topology.
- vertexAttributeAccessBeyondStride indicates whether this implementation supports accessing a vertex input attribute beyond the stride of the corresponding vertex input binding.

If the VkPhysicalDevicePortabilitySubsetFeaturesKHR structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDevicePortabilitySubsetFeaturesKHR **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDevicePortabilitySubsetFeaturesKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PORTABILITY_SUBSET_FEATURES_KHR

The VkPhysicalDevicePerformanceQueryFeaturesKHR structure is defined as:

```
// Provided by VK_KHR_performance_query
typedef struct VkPhysicalDevicePerformanceQueryFeaturesKHR {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           performanceCounterQueryPools;
    VkBool32           performanceCounterMultipleQueryPools;
} VkPhysicalDevicePerformanceQueryFeaturesKHR;
```

This structure describes the following features:

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `performanceCounterQueryPools` indicates whether the implementation supports performance counter query pools.
- `performanceCounterMultipleQueryPools` indicates whether the implementation supports using multiple performance query pools in a primary command buffer and secondary command buffers executed within it.

If the `VkPhysicalDevicePerformanceQueryFeaturesKHR` structure is included in the `pNext` chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. `VkPhysicalDevicePerformanceQueryFeaturesKHR` **can** also be used in the `pNext` chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDevicePerformanceQueryFeaturesKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PERFORMANCE_QUERY_FEATURES_KHR`

The `VkPhysicalDevice4444FormatsFeaturesEXT` structure is defined as:

```
// Provided by VK_EXT_4444_formats
typedef struct VkPhysicalDevice4444FormatsFeaturesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           formatA4R4G4B4;
    VkBool32           formatA4B4G4R4;
} VkPhysicalDevice4444FormatsFeaturesEXT;
```

This structure describes the following features:

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `formatA4R4G4B4` indicates that the implementation **must** support using a VkFormat of

VK_FORMAT_A4R4G4B4_UNORM_PACK16_EXT with at least the following VkFormatFeatureFlagBits:

  - VK_FORMAT_FEATURE_SAMPLED_IMAGE_BIT
  - VK_FORMAT_FEATURE_BLIT_SRC_BIT
  - VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT

- formatA4B4G4R4 indicates that the implementation **must** support using a VkFormat of VK_FORMAT_A4B4G4R4_UNORM_PACK16_EXT with at least the following VkFormatFeatureFlagBits:

  - VK_FORMAT_FEATURE_SAMPLED_IMAGE_BIT
  - VK_FORMAT_FEATURE_BLIT_SRC_BIT
  - VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT

If the VkPhysicalDevice4444FormatsFeaturesEXT structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDevice4444FormatsFeaturesEXT **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDevice4444FormatsFeaturesEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_4444_FORMATS_FEATURES_EXT

The VkPhysicalDeviceMutableDescriptorTypeFeaturesEXT structure is defined as:

```
// Provided by VK_EXT_mutable_descriptor_type
typedef struct VkPhysicalDeviceMutableDescriptorTypeFeaturesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           mutableDescriptorType;
} VkPhysicalDeviceMutableDescriptorTypeFeaturesEXT;
```

or the equivalent

```
// Provided by VK_VALVE_mutable_descriptor_type
typedef VkPhysicalDeviceMutableDescriptorTypeFeaturesEXT
VkPhysicalDeviceMutableDescriptorTypeFeaturesVALVE;
```

This structure describes the following feature:

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- mutableDescriptorType indicates that the implementation **must** support using the VkDescriptorType of VK_DESCRIPTOR_TYPE_MUTABLE_EXT with at least the following descriptor types, where any combination of the types **must** be supported:

    - VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE
    - VK_DESCRIPTOR_TYPE_STORAGE_IMAGE
    - VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER
    - VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER
    - VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER
    - VK_DESCRIPTOR_TYPE_STORAGE_BUFFER
- Additionally, mutableDescriptorType indicates that:
    - Non-uniform descriptor indexing **must** be supported if all descriptor types in a VkMutableDescriptorTypeListEXT for VK_DESCRIPTOR_TYPE_MUTABLE_EXT have the corresponding non-uniform indexing features enabled in VkPhysicalDeviceDescriptorIndexingFeatures.
    - VK_DESCRIPTOR_BINDING_UPDATE_AFTER_BIND_BIT with descriptorType of VK_DESCRIPTOR_TYPE_MUTABLE_EXT relaxes the list of required descriptor types to the descriptor types which have the corresponding update-after-bind feature enabled in VkPhysicalDeviceDescriptorIndexingFeatures.
    - Dynamically uniform descriptor indexing **must** be supported if all descriptor types in a VkMutableDescriptorTypeListEXT for VK_DESCRIPTOR_TYPE_MUTABLE_EXT have the corresponding dynamic indexing features enabled.
    - VK_DESCRIPTOR_SET_LAYOUT_CREATE_HOST_ONLY_POOL_BIT_EXT **must** be supported.
    - VK_DESCRIPTOR_POOL_CREATE_HOST_ONLY_BIT_EXT **must** be supported.

If the VkPhysicalDeviceMutableDescriptorTypeFeaturesEXT structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceMutableDescriptorTypeFeaturesEXT **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceMutableDescriptorTypeFeaturesEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MUTABLE_DESCRIPTOR_TYPE_FEATURES_EXT

The VkPhysicalDeviceDepthClipControlFeaturesEXT structure is defined as:

```
// Provided by VK_EXT_depth_clip_control
typedef struct VkPhysicalDeviceDepthClipControlFeaturesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           depthClipControl;
} VkPhysicalDeviceDepthClipControlFeaturesEXT;
```

This structure describes the following feature:

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- depthClipControl indicates that the implementation supports setting VkPipelineViewportDepthClipControlCreateInfoEXT::negativeOneToOne to VK_TRUE.

If the VkPhysicalDeviceDepthClipControlFeaturesEXT structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceDepthClipControlFeaturesEXT **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceDepthClipControlFeaturesEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DEPTH_CLIP_CONTROL_FEATURES_EXT

The VkPhysicalDeviceWorkgroupMemoryExplicitLayoutFeaturesKHR structure is defined as:

```
// Provided by VK_KHR_workgroup_memory_explicit_layout
typedef struct VkPhysicalDeviceWorkgroupMemoryExplicitLayoutFeaturesKHR {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           workgroupMemoryExplicitLayout;
    VkBool32           workgroupMemoryExplicitLayoutScalarBlockLayout;
    VkBool32           workgroupMemoryExplicitLayout8BitAccess;
    VkBool32           workgroupMemoryExplicitLayout16BitAccess;
} VkPhysicalDeviceWorkgroupMemoryExplicitLayoutFeaturesKHR;
```

This structure describes the following features:

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- workgroupMemoryExplicitLayout indicates whether the implementation supports the SPIR-V WorkgroupMemoryExplicitLayoutKHR capability.
- workgroupMemoryExplicitLayoutScalarBlockLayout indicates whether the implementation supports scalar alignment for laying out Workgroup Blocks.
- workgroupMemoryExplicitLayout8BitAccess indicates whether objects in the Workgroup storage class with the Block decoration **can** have 8-bit integer members. If this feature is not enabled, 8-bit integer members **must** not be used in such objects. This also indicates whether shader modules **can** declare the WorkgroupMemoryExplicitLayout8BitAccessKHR capability.
- workgroupMemoryExplicitLayout16BitAccess indicates whether objects in the Workgroup storage class with the Block decoration **can** have 16-bit integer and 16-bit floating-point members. If this feature is not enabled, 16-bit integer or 16-bit floating-point members **must** not be used in such objects. This also indicates whether shader modules **can** declare the

WorkgroupMemoryExplicitLayout16BitAccessKHR capability.

If the VkPhysicalDeviceWorkgroupMemoryExplicitLayoutFeaturesKHR structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceWorkgroupMemoryExplicitLayoutFeaturesKHR **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceWorkgroupMemoryExplicitLayoutFeaturesKHR-sType-sType sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_WORKGROUP_MEMORY_EXPLICIT_LAYOUT_FEATURES_KHR

The VkPhysicalDeviceSynchronization2Features structure is defined as:

```
typedef struct VkPhysicalDeviceSynchronization2Features {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           synchronization2;
} VkPhysicalDeviceSynchronization2Features;
```

or the equivalent

```
// Provided by VK_KHR_synchronization2
typedef VkPhysicalDeviceSynchronization2Features
VkPhysicalDeviceSynchronization2FeaturesKHR;
```

This structure describes the following feature:

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- synchronization2 indicates whether the implementation supports the new set of synchronization commands introduced in VK_KHR_synchronization2.

If the VkPhysicalDeviceSynchronization2Features structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceSynchronization2Features **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceSynchronization2Features-sType-sType sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SYNCHRONIZATION_2_FEATURES

The VkPhysicalDeviceVertexInputDynamicStateFeaturesEXT structure is defined as:

```
// Provided by VK_EXT_vertex_input_dynamic_state
typedef struct VkPhysicalDeviceVertexInputDynamicStateFeaturesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           vertexInputDynamicState;
} VkPhysicalDeviceVertexInputDynamicStateFeaturesEXT;
```

This structure describes the following feature:

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- vertexInputDynamicState indicates that the implementation supports the following dynamic states:
  - VK_DYNAMIC_STATE_VERTEX_INPUT_EXT

If the VkPhysicalDeviceVertexInputDynamicStateFeaturesEXT structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceVertexInputDynamicStateFeaturesEXT **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceVertexInputDynamicStateFeaturesEXT-sType-sType sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_VERTEX_INPUT_DYNAMIC_STATE_FEATURES_EXT

The VkPhysicalDevicePrimitivesGeneratedQueryFeaturesEXT structure is defined as:

```
// Provided by VK_EXT_primitives_generated_query
typedef struct VkPhysicalDevicePrimitivesGeneratedQueryFeaturesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           primitivesGeneratedQuery;
    VkBool32           primitivesGeneratedQueryWithRasterizerDiscard;
    VkBool32           primitivesGeneratedQueryWithNonZeroStreams;
} VkPhysicalDevicePrimitivesGeneratedQueryFeaturesEXT;
```

This structure describes the following features:

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- primitivesGeneratedQuery indicates whether the implementation supports the

VK_QUERY_TYPE_PRIMITIVES_GENERATED_EXT query type.

- primitivesGeneratedQueryWithRasterizerDiscard indicates whether the implementation supports this query when rasterization discard is enabled.
- primitivesGeneratedQueryWithNonZeroStreams indicates whether the implementation supports this query with a non-zero index in vkCmdBeginQueryIndexedEXT.

If the VkPhysicalDevicePrimitivesGeneratedQueryFeaturesEXT structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDevicePrimitivesGeneratedQueryFeaturesEXT **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDevicePrimitivesGeneratedQueryFeaturesEXT-sType-sType sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PRIMITIVES_GENERATED_QUERY_FEATURES_EXT

The VkPhysicalDeviceFragmentShadingRateFeaturesKHR structure is defined as:

```
// Provided by VK_KHR_fragment_shading_rate
typedef struct VkPhysicalDeviceFragmentShadingRateFeaturesKHR {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           pipelineFragmentShadingRate;
    VkBool32           primitiveFragmentShadingRate;
    VkBool32           attachmentFragmentShadingRate;
} VkPhysicalDeviceFragmentShadingRateFeaturesKHR;
```

This structure describes the following features:

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- pipelineFragmentShadingRate indicates that the implementation supports the pipeline fragment shading rate.
- primitiveFragmentShadingRate indicates that the implementation supports the primitive fragment shading rate.
- attachmentFragmentShadingRate indicates that the implementation supports the attachment fragment shading rate.

If the VkPhysicalDeviceFragmentShadingRateFeaturesKHR structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceFragmentShadingRateFeaturesKHR **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceFragmentShadingRateFeaturesKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_FRAGMENT_SHADING_RATE_FEATURES_KHR`

The `VkPhysicalDeviceLegacyDitheringFeaturesEXT` structure is defined as:

```
// Provided by VK_EXT_legacy_dithering
typedef struct VkPhysicalDeviceLegacyDitheringFeaturesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           legacyDithering;
} VkPhysicalDeviceLegacyDitheringFeaturesEXT;
```

This structure describes the following feature:

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `legacyDithering` indicates whether the implementation supports Legacy Dithering.

If the `VkPhysicalDeviceLegacyDitheringFeaturesEXT` structure is included in the `pNext` chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. `VkPhysicalDeviceLegacyDitheringFeaturesEXT` **can** also be used in the `pNext` chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceLegacyDitheringFeaturesEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_LEGACY_DITHERING_FEATURES_EXT`

The `VkPhysicalDeviceFragmentShadingRateEnumsFeaturesNV` structure is defined as:

```
// Provided by VK_NV_fragment_shading_rate_enums
typedef struct VkPhysicalDeviceFragmentShadingRateEnumsFeaturesNV {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           fragmentShadingRateEnums;
    VkBool32           supersampleFragmentShadingRates;
    VkBool32           noInvocationFragmentShadingRates;
} VkPhysicalDeviceFragmentShadingRateEnumsFeaturesNV;
```

This structure describes the following features:

- `sType` is a VkStructureType value identifying this structure.

- pNext is NULL or a pointer to a structure extending this structure.
- fragmentShadingRateEnums indicates that the implementation supports specifying fragment shading rates using the VkFragmentShadingRateNV enumerated type.
- supersampleFragmentShadingRates indicates that the implementation supports fragment shading rate enum values indicating more than one invocation per fragment.
- noInvocationFragmentShadingRates indicates that the implementation supports a fragment shading rate enum value indicating that no fragment shaders should be invoked when that shading rate is used.

If the VkPhysicalDeviceFragmentShadingRateEnumsFeaturesNV structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceFragmentShadingRateEnumsFeaturesNV **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceFragmentShadingRateEnumsFeaturesNV-sType-sType sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_FRAGMENT_SHADING_RATE_ENUMS_FEATURES_NV

The VkPhysicalDeviceInheritedViewportScissorFeaturesNV structure is defined as:

```
// Provided by VK_NV_inherited_viewport_scissor
typedef struct VkPhysicalDeviceInheritedViewportScissorFeaturesNV {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           inheritedViewportScissor2D;
} VkPhysicalDeviceInheritedViewportScissorFeaturesNV;
```

This structure describes the following feature:

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- inheritedViewportScissor2D indicates whether secondary command buffers can inherit most of the dynamic state affected by VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT, VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT, VK_DYNAMIC_STATE_DISCARD_RECTANGLE_EXT, VK_DYNAMIC_STATE_DISCARD_RECTANGLE_ENABLE_EXT, VK_DYNAMIC_STATE_DISCARD_RECTANGLE_MODE_EXT, VK_DYNAMIC_STATE_VIEWPORT or VK_DYNAMIC_STATE_SCISSOR, from a primary command buffer.

If the VkPhysicalDeviceInheritedViewportScissorFeaturesNV structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceInheritedViewportScissorFeaturesNV **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceInheritedViewportScissorFeaturesNV-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_INHERITED_VIEWPORT_SCISSOR_FEATURES_NV

The VkPhysicalDevicePipelineProtectedAccessFeaturesEXT structure is defined as:

```
// Provided by VK_EXT_pipeline_protected_access
typedef struct VkPhysicalDevicePipelineProtectedAccessFeaturesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           pipelineProtectedAccess;
} VkPhysicalDevicePipelineProtectedAccessFeaturesEXT;
```

This structure describes the following feature:

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- pipelineProtectedAccess indicates whether the implementation supports specifying protected access on individual pipelines.

If the VkPhysicalDevicePipelineProtectedAccessFeaturesEXT structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDevicePipelineProtectedAccessFeaturesEXT **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDevicePipelineProtectedAccessFeaturesEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PIPELINE_PROTECTED_ACCESS_FEATURES_EXT

The VkPhysicalDeviceYcbcr2Plane444FormatsFeaturesEXT structure is defined as:

```
// Provided by VK_EXT_ycbcr_2plane_444_formats
typedef struct VkPhysicalDeviceYcbcr2Plane444FormatsFeaturesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           ycbcr2plane444Formats;
} VkPhysicalDeviceYcbcr2Plane444FormatsFeaturesEXT;
```

This structure describes the following feature:

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- ycbcr2plane444Formats indicates that the implementation supports the following 2-plane 444 Y'C$_B$C$_R$ formats:
  - VK_FORMAT_G8_B8R8_2PLANE_444_UNORM
  - VK_FORMAT_G10X6_B10X6R10X6_2PLANE_444_UNORM_3PACK16
  - VK_FORMAT_G12X4_B12X4R12X4_2PLANE_444_UNORM_3PACK16
  - VK_FORMAT_G16_B16R16_2PLANE_444_UNORM

If the VkPhysicalDeviceYcbcr2Plane444FormatsFeaturesEXT structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceYcbcr2Plane444FormatsFeaturesEXT **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceYcbcr2Plane444FormatsFeaturesEXT-sType-sType sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_YCBCR_2_PLANE_444_FORMATS_FEATURES_EXT

The VkPhysicalDeviceColorWriteEnableFeaturesEXT structure is defined as:

```
// Provided by VK_EXT_color_write_enable
typedef struct VkPhysicalDeviceColorWriteEnableFeaturesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           colorWriteEnable;
} VkPhysicalDeviceColorWriteEnableFeaturesEXT;
```

This structure describes the following feature:

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- colorWriteEnable indicates that the implementation supports the dynamic state VK_DYNAMIC_STATE_COLOR_WRITE_ENABLE_EXT.

If the VkPhysicalDeviceColorWriteEnableFeaturesEXT structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceColorWriteEnableFeaturesEXT **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceColorWriteEnableFeaturesEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_COLOR_WRITE_ENABLE_FEATURES_EXT`

The `VkPhysicalDevicePipelinePropertiesFeaturesEXT` structure is defined as:

```
// Provided by VK_EXT_pipeline_properties
typedef struct VkPhysicalDevicePipelinePropertiesFeaturesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           pipelinePropertiesIdentifier;
} VkPhysicalDevicePipelinePropertiesFeaturesEXT;
```

This structure describes the following feature:

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `pipelinePropertiesIdentifier` indicates that the implementation supports querying a unique pipeline identifier.

If the `VkPhysicalDevicePipelinePropertiesFeaturesEXT` structure is included in the `pNext` chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. `VkPhysicalDevicePipelinePropertiesFeaturesEXT` **can** also be used in the `pNext` chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDevicePipelinePropertiesFeaturesEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PIPELINE_PROPERTIES_FEATURES_EXT`

The `VkPhysicalDeviceProvokingVertexFeaturesEXT` structure is defined as:

```
// Provided by VK_EXT_provoking_vertex
typedef struct VkPhysicalDeviceProvokingVertexFeaturesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           provokingVertexLast;
    VkBool32           transformFeedbackPreservesProvokingVertex;
} VkPhysicalDeviceProvokingVertexFeaturesEXT;
```

This structure describes the following features:

- `sType` is a VkStructureType value identifying this structure.

- pNext is NULL or a pointer to a structure extending this structure.
- provokingVertexLast indicates whether the implementation supports the VK_PROVOKING_VERTEX_MODE_LAST_VERTEX_EXT provoking vertex mode for flat shading.
- transformFeedbackPreservesProvokingVertex indicates that the order of vertices within each primitive written by transform feedback will preserve the provoking vertex. This does not apply to triangle fan primitives when transformFeedbackPreservesTriangleFanProvokingVertex is VK_FALSE. transformFeedbackPreservesProvokingVertex **must** be VK_FALSE when the VK_EXT_transform_feedback extension is not supported.

If the VkPhysicalDeviceProvokingVertexFeaturesEXT structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceProvokingVertexFeaturesEXT **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

When VkPhysicalDeviceProvokingVertexFeaturesEXT is in the pNext chain of VkDeviceCreateInfo but the transformFeedback feature is not enabled, the value of transformFeedbackPreservesProvokingVertex is ignored.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceProvokingVertexFeaturesEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PROVOKING_VERTEX_FEATURES_EXT

The VkPhysicalDeviceDescriptorBufferFeaturesEXT structure is defined as:

```
// Provided by VK_EXT_descriptor_buffer
typedef struct VkPhysicalDeviceDescriptorBufferFeaturesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           descriptorBuffer;
    VkBool32           descriptorBufferCaptureReplay;
    VkBool32           descriptorBufferImageLayoutIgnored;
    VkBool32           descriptorBufferPushDescriptors;
} VkPhysicalDeviceDescriptorBufferFeaturesEXT;
```

This structure describes the following feature:

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- descriptorBuffer indicates that the implementation supports putting shader-accessible descriptors directly in memory.
- descriptorBufferCaptureReplay indicates that the implementation supports capture and replay when using descriptor buffers. If this is VK_TRUE, all resources created with VK_BUFFER_CREATE_DESCRIPTOR_BUFFER_CAPTURE_REPLAY_BIT_EXT,

VK_IMAGE_CREATE_DESCRIPTOR_BUFFER_CAPTURE_REPLAY_BIT_EXT, VK_IMAGE_VIEW_CREATE_DESCRIPTOR_BUFFER_CAPTURE_REPLAY_BIT_EXT, VK_SAMPLER_CREATE_DESCRIPTOR_BUFFER_CAPTURE_REPLAY_BIT_EXT, or VK_ACCELERATION_STRUCTURE_CREATE_DESCRIPTOR_BUFFER_CAPTURE_REPLAY_BIT_EXT **must** be created before resources of the same types without those flags.

- descriptorBufferImageLayoutIgnored indicates that the implementation will ignore imageLayout in VkDescriptorImageInfo when calling vkGetDescriptorEXT.
- descriptorBufferPushDescriptors indicates that the implementation supports using push descriptors with descriptor buffers.

If the VkPhysicalDeviceDescriptorBufferFeaturesEXT structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceDescriptorBufferFeaturesEXT **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceDescriptorBufferFeaturesEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DESCRIPTOR_BUFFER_FEATURES_EXT

The VkPhysicalDevicePageableDeviceLocalMemoryFeaturesEXT structure is defined as:

```
// Provided by VK_EXT_pageable_device_local_memory
typedef struct VkPhysicalDevicePageableDeviceLocalMemoryFeaturesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           pageableDeviceLocalMemory;
} VkPhysicalDevicePageableDeviceLocalMemoryFeaturesEXT;
```

This structure describes the following feature:

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- pageableDeviceLocalMemory indicates that the implementation supports pageable device-local memory and **may** transparently move device-local memory allocations to host-local memory to better share device-local memory with other applications.

If the VkPhysicalDevicePageableDeviceLocalMemoryFeaturesEXT structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDevicePageableDeviceLocalMemoryFeaturesEXT **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDevicePageableDeviceLocalMemoryFeaturesEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PAGEABLE_DEVICE_LOCAL_MEMORY_FEATURES_EXT

The VkPhysicalDeviceMultiDrawFeaturesEXT structure is defined as:

```
// Provided by VK_EXT_multi_draw
typedef struct VkPhysicalDeviceMultiDrawFeaturesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           multiDraw;
} VkPhysicalDeviceMultiDrawFeaturesEXT;
```

This structure describes the following features:

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- multiDraw indicates that the implementation supports vkCmdDrawMultiEXT and vkCmdDrawMultiIndexedEXT.

If the VkPhysicalDeviceMultiDrawFeaturesEXT structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceMultiDrawFeaturesEXT **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceMultiDrawFeaturesEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MULTI_DRAW_FEATURES_EXT

The VkPhysicalDeviceRayTracingMotionBlurFeaturesNV structure is defined as:

```
// Provided by VK_NV_ray_tracing_motion_blur
typedef struct VkPhysicalDeviceRayTracingMotionBlurFeaturesNV {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           rayTracingMotionBlur;
    VkBool32           rayTracingMotionBlurPipelineTraceRaysIndirect;
} VkPhysicalDeviceRayTracingMotionBlurFeaturesNV;
```

This structure describes the following features:

- sType is a VkStructureType value identifying this structure.

- pNext is NULL or a pointer to a structure extending this structure.
- rayTracingMotionBlur indicates whether the implementation supports the motion blur feature.
- rayTracingMotionBlurPipelineTraceRaysIndirect indicates whether the implementation supports indirect ray tracing commands with the motion blur feature enabled.

If the VkPhysicalDeviceRayTracingMotionBlurFeaturesNV structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceRayTracingMotionBlurFeaturesNV **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceRayTracingMotionBlurFeaturesNV-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_RAY_TRACING_MOTION_BLUR_FEATURES_NV

The VkPhysicalDeviceOpacityMicromapFeaturesEXT structure is defined as:

```
// Provided by VK_EXT_opacity_micromap
typedef struct VkPhysicalDeviceOpacityMicromapFeaturesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           micromap;
    VkBool32           micromapCaptureReplay;
    VkBool32           micromapHostCommands;
} VkPhysicalDeviceOpacityMicromapFeaturesEXT;
```

This structure describes the following feature:

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- micromap indicates whether the implementation supports the micromap array feature.
- micromapCaptureReplay indicates whether the implementation supports capture and replay of addresses for micromap arrays.
- micromapHostCommands indicates whether the implementation supports host side micromap array commands.

If the VkPhysicalDeviceOpacityMicromapFeaturesEXT structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceOpacityMicromapFeaturesEXT **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceOpacityMicromapFeaturesEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_OPACITY_MICROMAP_FEATURES_EXT`

The `VkPhysicalDeviceDisplacementMicromapFeaturesNV` structure is defined as:

```
// Provided by VK_NV_displacement_micromap
typedef struct VkPhysicalDeviceDisplacementMicromapFeaturesNV {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           displacementMicromap;
} VkPhysicalDeviceDisplacementMicromapFeaturesNV;
```

This structure describes the following feature:

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `displacementMicromap` indicates whether the implementation supports the displacement micromap feature.

If the `VkPhysicalDeviceDisplacementMicromapFeaturesNV` structure is included in the `pNext` chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. `VkPhysicalDeviceDisplacementMicromapFeaturesNV` **can** also be used in the `pNext` chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceDisplacementMicromapFeaturesNV-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DISPLACEMENT_MICROMAP_FEATURES_NV`

The `VkPhysicalDeviceRayTracingValidationFeaturesNV` structure is defined as:

```
// Provided by VK_NV_ray_tracing_validation
typedef struct VkPhysicalDeviceRayTracingValidationFeaturesNV {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           rayTracingValidation;
} VkPhysicalDeviceRayTracingValidationFeaturesNV;
```

This structure describes the following feature:

- `sType` is a VkStructureType value identifying this structure.

- pNext is NULL or a pointer to a structure extending this structure.
- rayTracingValidation indicates whether the implementation supports the ray tracing validation feature.

If the VkPhysicalDeviceRayTracingValidationFeaturesNV structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceRayTracingValidationFeaturesNV **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceRayTracingValidationFeaturesNV-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_RAY_TRACING_VALIDATION_FEATURES_NV

The VkPhysicalDeviceSubpassShadingFeaturesHUAWEI structure is defined as:

```
// Provided by VK_HUAWEI_subpass_shading
typedef struct VkPhysicalDeviceSubpassShadingFeaturesHUAWEI {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           subpassShading;
} VkPhysicalDeviceSubpassShadingFeaturesHUAWEI;
```

This structure describes the following feature:

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- subpassShading specifies whether subpass shading is supported.

If the VkPhysicalDeviceSubpassShadingFeaturesHUAWEI structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceSubpassShadingFeaturesHUAWEI **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceSubpassShadingFeaturesHUAWEI-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SUBPASS_SHADING_FEATURES_HUAWEI

The VkPhysicalDeviceExternalMemoryRDMAFeaturesNV structure is defined as:

```
// Provided by VK_NV_external_memory_rdma
typedef struct VkPhysicalDeviceExternalMemoryRDMAFeaturesNV {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           externalMemoryRDMA;
} VkPhysicalDeviceExternalMemoryRDMAFeaturesNV;
```

This structure describes the following feature:

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `externalMemoryRDMA` indicates whether the implementation has support for the `VK_MEMORY_PROPERTY_RDMA_CAPABLE_BIT_NV` memory property and the `VK_EXTERNAL_MEMORY_HANDLE_TYPE_RDMA_ADDRESS_BIT_NV` external memory handle type.

If the `VkPhysicalDeviceExternalMemoryRDMAFeaturesNV` structure is included in the `pNext` chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. `VkPhysicalDeviceExternalMemoryRDMAFeaturesNV` **can** also be used in the `pNext` chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceExternalMemoryRDMAFeaturesNV-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTERNAL_MEMORY_RDMA_FEATURES_NV`

The `VkPhysicalDevicePresentIdFeaturesKHR` structure is defined as:

```
// Provided by VK_KHR_present_id
typedef struct VkPhysicalDevicePresentIdFeaturesKHR {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           presentId;
} VkPhysicalDevicePresentIdFeaturesKHR;
```

This structure describes the following feature:

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `presentId` indicates that the implementation supports specifying present ID values in the `VkPresentIdKHR` extension to the `VkPresentInfoKHR` struct.

If the `VkPhysicalDevicePresentIdFeaturesKHR` structure is included in the `pNext` chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. `VkPhysicalDevicePresentIdFeaturesKHR`

**can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDevicePresentIdFeaturesKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PRESENT_ID_FEATURES_KHR

The VkPhysicalDevicePresentWaitFeaturesKHR structure is defined as:

```
// Provided by VK_KHR_present_wait
typedef struct VkPhysicalDevicePresentWaitFeaturesKHR {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           presentWait;
} VkPhysicalDevicePresentWaitFeaturesKHR;
```

This structure describes the following feature:

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- presentWait indicates that the implementation supports vkWaitForPresentKHR.

If the VkPhysicalDevicePresentWaitFeaturesKHR structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDevicePresentWaitFeaturesKHR **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDevicePresentWaitFeaturesKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PRESENT_WAIT_FEATURES_KHR

The VkPhysicalDeviceHostImageCopyFeaturesEXT structure is defined as:

```
// Provided by VK_EXT_host_image_copy
typedef struct VkPhysicalDeviceHostImageCopyFeaturesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           hostImageCopy;
} VkPhysicalDeviceHostImageCopyFeaturesEXT;
```

This structure describes the following feature:

- sType is a VkStructureType value identifying this structure.

- pNext is NULL or a pointer to a structure extending this structure.
- hostImageCopy indicates that the implementation supports copying from host memory to images using the vkCopyMemoryToImageEXT command, copying from images to host memory using the vkCopyImageToMemoryEXT command, and copying between images using the vkCopyImageToImageEXT command.

If the VkPhysicalDeviceHostImageCopyFeaturesEXT structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceHostImageCopyFeaturesEXT **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceHostImageCopyFeaturesEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_HOST_IMAGE_COPY_FEATURES_EXT

The VkPhysicalDevicePresentBarrierFeaturesNV structure is defined as:

```
// Provided by VK_NV_present_barrier
typedef struct VkPhysicalDevicePresentBarrierFeaturesNV {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           presentBarrier;
} VkPhysicalDevicePresentBarrierFeaturesNV;
```

This structure describes the following feature:

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- presentBarrier indicates that the implementation supports the present barrier feature.

If the VkPhysicalDevicePresentBarrierFeaturesNV structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDevicePresentBarrierFeaturesNV **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDevicePresentBarrierFeaturesNV-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PRESENT_BARRIER_FEATURES_NV

The VkPhysicalDeviceShaderIntegerDotProductFeatures structure is defined as:

```
typedef struct VkPhysicalDeviceShaderIntegerDotProductFeatures {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           shaderIntegerDotProduct;
} VkPhysicalDeviceShaderIntegerDotProductFeatures;
```

or the equivalent

```
// Provided by VK_KHR_shader_integer_dot_product
typedef VkPhysicalDeviceShaderIntegerDotProductFeatures
VkPhysicalDeviceShaderIntegerDotProductFeaturesKHR;
```

This structure describes the following feature:

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `shaderIntegerDotProduct` specifies whether shader modules **can** declare the `DotProductInputAllKHR`, `DotProductInput4x8BitKHR`, `DotProductInput4x8BitPackedKHR` and `DotProductKHR` capabilities.

If the `VkPhysicalDeviceShaderIntegerDotProductFeatures` structure is included in the `pNext` chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. `VkPhysicalDeviceShaderIntegerDotProductFeatures` **can** also be used in the `pNext` chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceShaderIntegerDotProductFeatures-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_INTEGER_DOT_PRODUCT_FEATURES`

The `VkPhysicalDeviceMaintenance4Features` structure is defined as:

```
typedef struct VkPhysicalDeviceMaintenance4Features {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           maintenance4;
} VkPhysicalDeviceMaintenance4Features;
```

or the equivalent

```
// Provided by VK_KHR_maintenance4
typedef VkPhysicalDeviceMaintenance4Features VkPhysicalDeviceMaintenance4FeaturesKHR;
```

This structure describes the following feature:

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- maintenance4 indicates that the implementation supports the following:
  - The application **may** destroy a VkPipelineLayout object immediately after using it to create another object.
  - LocalSizeId **can** be used as an alternative to LocalSize to specify the local workgroup size with specialization constants.
  - Images created with identical creation parameters will always have the same alignment requirements.
  - The size memory requirement of a buffer or image is never greater than that of another buffer or image created with a greater or equal size.
  - Push constants do not have to be initialized before they are dynamically accessed.
  - The interface matching rules allow a larger output vector to match with a smaller input vector, with additional values being discarded.

If the VkPhysicalDeviceMaintenance4Features structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceMaintenance4Features **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceMaintenance4Features-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MAINTENANCE_4_FEATURES

The VkPhysicalDeviceMaintenance5FeaturesKHR structure is defined as:

```
// Provided by VK_KHR_maintenance5
typedef struct VkPhysicalDeviceMaintenance5FeaturesKHR {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           maintenance5;
} VkPhysicalDeviceMaintenance5FeaturesKHR;
```

This structure describes the following feature:

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- maintenance5 indicates that the implementation supports the following:
  - The ability to expose support for the optional format VK_FORMAT_A1B5G5R5_UNORM_PACK16_KHR.

- The ability to expose support for the optional format VK_FORMAT_A8_UNORM_KHR.
- A property to indicate that multisample coverage operations are performed after sample counting in EarlyFragmentTests mode.
- Creating a VkBufferView with a subset of the associated VkBuffer usage using VkBufferUsageFlags2CreateInfoKHR.
- A new function vkCmdBindIndexBuffer2KHR, allowing a range of memory to be bound as an index buffer.
- vkGetDeviceProcAddr will return NULL for function pointers of core functions for versions higher than the version requested by the application.
- vkCmdBindVertexBuffers2EXT supports using VK_WHOLE_SIZE in the pSizes parameter.
- If PointSize is not written, a default value of 1.0 is used for the size of points.
- VkShaderModuleCreateInfo **can** be added as a chained structure to pipeline creation via VkPipelineShaderStageCreateInfo, rather than having to create a shader module.
- A function vkGetRenderingAreaGranularityKHR to query the optimal render area for a dynamic rendering instance.
- A property to indicate that depth/stencil texturing operations with VK_COMPONENT_SWIZZLE_ONE have defined behavior.
- vkGetDeviceImageSubresourceLayoutKHR allows an application to perform a vkGetImageSubresourceLayout query without having to create an image.
- VK_REMAINING_ARRAY_LAYERS as the layerCount member of VkImageSubresourceLayers.
- A property to indicate whether PointSize controls the final rasterization of polygons if polygon mode is VK_POLYGON_MODE_POINT.
- Two properties to indicate the non-strict line rasterization algorithm used.
- Two new flags words VkPipelineCreateFlagBits2KHR and VkBufferUsageFlagBits2KHR.
- Physical-device-level functions **can** now be called with any value in the valid range for a type beyond the defined enumerants, such that applications can avoid checking individual features, extensions, or versions before querying supported properties of a particular enumerant.
- Copies between images of any type are allowed, with 1D images treated as 2D images with a height of 1.

If the VkPhysicalDeviceMaintenance5FeaturesKHR structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceMaintenance5FeaturesKHR **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceMaintenance5FeaturesKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MAINTENANCE_5_FEATURES_KHR

The VkPhysicalDeviceMaintenance6FeaturesKHR structure is defined as:

```
// Provided by VK_KHR_maintenance6
typedef struct VkPhysicalDeviceMaintenance6FeaturesKHR {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           maintenance6;
} VkPhysicalDeviceMaintenance6FeaturesKHR;
```

This structure describes the following feature:

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- maintenance6 indicates that the implementation supports the following:
  - VK_NULL_HANDLE **can** be used when binding an index buffer
  - VkBindMemoryStatusKHR **can** be included in the pNext chain of the VkBindBufferMemoryInfo and VkBindImageMemoryInfo structures, enabling applications to retrieve VkResult values for individual memory binding operations.
  - VkPhysicalDeviceMaintenance6PropertiesKHR::blockTexelViewCompatibleMultipleLayers property to indicate that the implementation supports creating image views with VK_IMAGE_CREATE_BLOCK_TEXEL_VIEW_COMPATIBLE_BIT where the layerCount member of subresourceRange is greater than 1.
  - VkPhysicalDeviceMaintenance6PropertiesKHR::maxCombinedImageSamplerDescriptorCount property which indicates the maximum descriptor size required for any format that requires a sampler $Y'C_BC_R$ conversion supported by the implementation.
  - A VkPhysicalDeviceMaintenance6PropertiesKHR::fragmentShadingRateClampCombinerInputs property which indicates whether the implementation clamps the inputs to fragment shading rate combiner operations.

If the VkPhysicalDeviceMaintenance6FeaturesKHR structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceMaintenance6FeaturesKHR **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceMaintenance6FeaturesKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MAINTENANCE_6_FEATURES_KHR

The VkPhysicalDeviceMaintenance7FeaturesKHR structure is defined as:

```
// Provided by VK_KHR_maintenance7
typedef struct VkPhysicalDeviceMaintenance7FeaturesKHR {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           maintenance7;
} VkPhysicalDeviceMaintenance7FeaturesKHR;
```

This structure describes the following feature:

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `maintenance7` indicates that the implementation supports the following:
  - The `VK_RENDERING_CONTENTS_INLINE_BIT_KHR` and `VK_SUBPASS_CONTENTS_INLINE_AND_SECONDARY_COMMAND_BUFFERS_KHR` flags **can** be used to record commands in render pass instances both inline and in secondary command buffers executed with vkCmdExecuteCommands for dynamic rendering and legacy render passes respectively.
  - Querying information regarding the underlying devices in environments where the Vulkan implementation is provided through layered implementations. This is done by chaining VkPhysicalDeviceLayeredApiPropertiesListKHR to VkPhysicalDeviceProperties2.
  - New limits which indicate the maximum total count of dynamic uniform buffers and dynamic storage buffers that **can** be included in a pipeline layout.
  - 32-bit timestamp queries **must** wrap on overflow
  - A property that indicates whether a fragment shading rate attachment can have a size that is too small to cover a specified render area.
  - A property that indicates support for writing to one aspect of a depth/stencil attachment without performing a read-modify-write operation on the other aspect

If the `VkPhysicalDeviceMaintenance7FeaturesKHR` structure is included in the `pNext` chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. `VkPhysicalDeviceMaintenance7FeaturesKHR` **can** also be used in the `pNext` chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceMaintenance7FeaturesKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MAINTENANCE_7_FEATURES_KHR`

The `VkPhysicalDeviceDynamicRenderingFeatures` structure is defined as:

```
typedef struct VkPhysicalDeviceDynamicRenderingFeatures {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           dynamicRendering;
} VkPhysicalDeviceDynamicRenderingFeatures;
```

or the equivalent

```
// Provided by VK_KHR_dynamic_rendering
typedef VkPhysicalDeviceDynamicRenderingFeatures
VkPhysicalDeviceDynamicRenderingFeaturesKHR;
```

This structure describes the following feature:

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `dynamicRendering` specifies that the implementation supports dynamic render pass instances using the vkCmdBeginRenderingKHR command.

If the `VkPhysicalDeviceDynamicRenderingFeatures` structure is included in the `pNext` chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. `VkPhysicalDeviceDynamicRenderingFeatures` **can** also be used in the `pNext` chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceDynamicRenderingFeatures-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DYNAMIC_RENDERING_FEATURES`

The `VkPhysicalDeviceRGBA10X6FormatsFeaturesEXT` structure is defined as:

```
// Provided by VK_EXT_rgba10x6_formats
typedef struct VkPhysicalDeviceRGBA10X6FormatsFeaturesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           formatRgba10x6WithoutYCbCrSampler;
} VkPhysicalDeviceRGBA10X6FormatsFeaturesEXT;
```

This structure describes the following feature:

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.

- formatRgba10x6WithoutYCbCrSampler indicates that VK_FORMAT_R10X6G10X6B10X6A10X6_UNORM_4PACK16 **can** be used with a VkImageView with subresourceRange.aspectMask equal to VK_IMAGE_ASPECT_COLOR_BIT without a sampler $Y'C_BC_R$ conversion enabled.

If the VkPhysicalDeviceRGBA10X6FormatsFeaturesEXT structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceRGBA10X6FormatsFeaturesEXT **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceRGBA10X6FormatsFeaturesEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_RGBA10X6_FORMATS_FEATURES_EXT

The VkPhysicalDevicePipelineRobustnessFeaturesEXT structure is defined as:

```
// Provided by VK_EXT_pipeline_robustness
typedef struct VkPhysicalDevicePipelineRobustnessFeaturesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           pipelineRobustness;
} VkPhysicalDevicePipelineRobustnessFeaturesEXT;
```

This structure describes the following feature:

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- pipelineRobustness indicates that robustness **can** be requested on a per-pipeline-stage granularity.

| | |
|---|---|
| **NOTE** | Enabling pipelineRobustness may, on some platforms, incur a minor performance cost when robustBufferAccess is disabled, even for pipelines which do not make use of any robustness features. If robustness is not needed, pipelineRobustness should not be enabled by an application. |

If the VkPhysicalDevicePipelineRobustnessFeaturesEXT structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDevicePipelineRobustnessFeaturesEXT **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDevicePipelineRobustnessFeaturesEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PIPELINE_ROBUSTNESS_FEATURES_EXT

The VkPhysicalDeviceImageViewMinLodFeaturesEXT structure is defined as:

```
// Provided by VK_EXT_image_view_min_lod
typedef struct VkPhysicalDeviceImageViewMinLodFeaturesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           minLod;
} VkPhysicalDeviceImageViewMinLodFeaturesEXT;
```

This structure describes the following feature:

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- minLod indicates whether the implementation supports clamping the minimum LOD value during Image Level(s) Selection, Texel Gathering and Integer Texel Coordinate Operations with a given VkImageView by VkImageViewMinLodCreateInfoEXT::minLod.

If the VkPhysicalDeviceImageViewMinLodFeaturesEXT structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceImageViewMinLodFeaturesEXT **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceImageViewMinLodFeaturesEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_IMAGE_VIEW_MIN_LOD_FEATURES_EXT

The VkPhysicalDeviceRasterizationOrderAttachmentAccessFeaturesEXT structure is defined as:

```
// Provided by VK_EXT_rasterization_order_attachment_access
typedef struct VkPhysicalDeviceRasterizationOrderAttachmentAccessFeaturesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           rasterizationOrderColorAttachmentAccess;
    VkBool32           rasterizationOrderDepthAttachmentAccess;
    VkBool32           rasterizationOrderStencilAttachmentAccess;
} VkPhysicalDeviceRasterizationOrderAttachmentAccessFeaturesEXT;
```

or the equivalent

```
// Provided by VK_ARM_rasterization_order_attachment_access
typedef VkPhysicalDeviceRasterizationOrderAttachmentAccessFeaturesEXT
VkPhysicalDeviceRasterizationOrderAttachmentAccessFeaturesARM;
```

This structure describes the following features:

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- rasterizationOrderColorAttachmentAccess indicates that rasterization order access to color and input attachments is supported by the implementation.
- rasterizationOrderDepthAttachmentAccess indicates that rasterization order access to the depth aspect of depth/stencil and input attachments is supported by the implementation.
- rasterizationOrderStencilAttachmentAccess indicates that rasterization order access to the stencil aspect of depth/stencil and input attachments is supported by the implementation.

If the VkPhysicalDeviceRasterizationOrderAttachmentAccessFeaturesEXT structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceRasterizationOrderAttachmentAccessFeaturesEXT **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceRasterizationOrderAttachmentAccessFeaturesEXT-sType-sType sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_RASTERIZATION_ORDER_ATTACHMENT_ACCESS_FEATURES_EXT

The VkPhysicalDeviceSubpassMergeFeedbackFeaturesEXT structure is defined as:

```
// Provided by VK_EXT_subpass_merge_feedback
typedef struct VkPhysicalDeviceSubpassMergeFeedbackFeaturesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           subpassMergeFeedback;
} VkPhysicalDeviceSubpassMergeFeedbackFeaturesEXT;
```

This structure describes the following feature:

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- subpassMergeFeedback indicates whether the implementation supports feedback of subpass merging.

If the VkPhysicalDeviceSubpassMergeFeedbackFeaturesEXT structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceSubpassMergeFeedbackFeaturesEXT **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceSubpassMergeFeedbackFeaturesEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SUBPASS_MERGE_FEEDBACK_FEATURES_EXT`

The `VkPhysicalDeviceLinearColorAttachmentFeaturesNV` structure is defined as:

```
// Provided by VK_NV_linear_color_attachment
typedef struct VkPhysicalDeviceLinearColorAttachmentFeaturesNV {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           linearColorAttachment;
} VkPhysicalDeviceLinearColorAttachmentFeaturesNV;
```

This structure describes the following feature:

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `linearColorAttachment` indicates whether the implementation supports renderable Linear Color Attachment

If the `VkPhysicalDeviceLinearColorAttachmentFeaturesNV` structure is included in the `pNext` chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. `VkPhysicalDeviceLinearColorAttachmentFeaturesNV` **can** also be used in the `pNext` chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceLinearColorAttachmentFeaturesNV-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_LINEAR_COLOR_ATTACHMENT_FEATURES_NV`

The `VkPhysicalDeviceAttachmentFeedbackLoopLayoutFeaturesEXT` structure is defined as:

```
// Provided by VK_EXT_attachment_feedback_loop_layout
typedef struct VkPhysicalDeviceAttachmentFeedbackLoopLayoutFeaturesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           attachmentFeedbackLoopLayout;
} VkPhysicalDeviceAttachmentFeedbackLoopLayoutFeaturesEXT;
```

This structure describes the following feature:

- `sType` is a VkStructureType value identifying this structure.

- pNext is NULL or a pointer to a structure extending this structure.
- attachmentFeedbackLoopLayout indicates whether the implementation supports using VK_IMAGE_LAYOUT_ATTACHMENT_FEEDBACK_LOOP_OPTIMAL_EXT image layout for images created with VK_IMAGE_USAGE_ATTACHMENT_FEEDBACK_LOOP_BIT_EXT.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceAttachmentFeedbackLoopLayoutFeaturesEXT-sType-sType sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_ATTACHMENT_FEEDBACK_LOOP_LAYOUT_FEATURES_EXT

The VkPhysicalDeviceNestedCommandBufferFeaturesEXT structure is defined as:

```
// Provided by VK_EXT_nested_command_buffer
typedef struct VkPhysicalDeviceNestedCommandBufferFeaturesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           nestedCommandBuffer;
    VkBool32           nestedCommandBufferRendering;
    VkBool32           nestedCommandBufferSimultaneousUse;
} VkPhysicalDeviceNestedCommandBufferFeaturesEXT;
```

This structure describes the following features:

- nestedCommandBuffer indicates the implementation supports nested command buffers, which allows Secondary Command Buffers to execute other Secondary Command Buffers.
- nestedCommandBufferRendering indicates that it is valid to call vkCmdExecuteCommands inside a Secondary Command Buffer recorded with VK_COMMAND_BUFFER_USAGE_RENDER_PASS_CONTINUE_BIT.
- nestedCommandBufferSimultaneousUse indicates that the implementation supports nested command buffers with command buffers that are recorded with VK_COMMAND_BUFFER_USAGE_SIMULTANEOUS_USE_BIT.

If the VkPhysicalDeviceNestedCommandBufferFeaturesEXT structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceNestedCommandBufferFeaturesEXT **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceNestedCommandBufferFeaturesEXT-sType-sType sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_NESTED_COMMAND_BUFFER_FEATURES_EXT

The VkPhysicalDeviceGraphicsPipelineLibraryFeaturesEXT structure is defined as:

```
// Provided by VK_EXT_graphics_pipeline_library
typedef struct VkPhysicalDeviceGraphicsPipelineLibraryFeaturesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           graphicsPipelineLibrary;
} VkPhysicalDeviceGraphicsPipelineLibraryFeaturesEXT;
```

This structure describes the following feature:

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `graphicsPipelineLibrary` indicates that the implementation supports graphics pipeline libraries.

If the `VkPhysicalDeviceGraphicsPipelineLibraryFeaturesEXT` structure is included in the `pNext` chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. `VkPhysicalDeviceGraphicsPipelineLibraryFeaturesEXT` **can** also be used in the `pNext` chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceGraphicsPipelineLibraryFeaturesEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_GRAPHICS_PIPELINE_LIBRARY_FEATURES_EXT`

The `VkPhysicalDevicePipelineBinaryFeaturesKHR` structure is defined as:

```
// Provided by VK_KHR_pipeline_binary
typedef struct VkPhysicalDevicePipelineBinaryFeaturesKHR {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           pipelineBinaries;
} VkPhysicalDevicePipelineBinaryFeaturesKHR;
```

This structure describes the following feature:

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `pipelineBinaries` indicates that the implementation supports Pipeline Binaries.

If the `VkPhysicalDevicePipelineBinaryFeaturesKHR` structure is included in the `pNext` chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. `VkPhysicalDevicePipelineBinaryFeaturesKHR` **can** also be used in the `pNext` chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDevicePipelineBinaryFeaturesKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PIPELINE_BINARY_FEATURES_KHR`

The `VkPhysicalDeviceMultisampledRenderToSingleSampledFeaturesEXT` structure is defined as:

```
// Provided by VK_EXT_multisampled_render_to_single_sampled
typedef struct VkPhysicalDeviceMultisampledRenderToSingleSampledFeaturesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           multisampledRenderToSingleSampled;
} VkPhysicalDeviceMultisampledRenderToSingleSampledFeaturesEXT;
```

This structure describes the following features:

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `multisampledRenderToSingleSampled` indicates that the implementation supports multisampled rendering to single-sampled render pass attachments.

If the `VkPhysicalDeviceMultisampledRenderToSingleSampledFeaturesEXT` structure is included in the `pNext` chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. `VkPhysicalDeviceMultisampledRenderToSingleSampledFeaturesEXT` **can** also be used in the `pNext` chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceMultisampledRenderToSingleSampledFeaturesEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MULTISAMPLED_RENDER_TO_SINGLE_SAMPLED_FEATURES_EXT`

The `VkPhysicalDeviceImage2DViewOf3DFeaturesEXT` structure is defined as:

```
// Provided by VK_EXT_image_2d_view_of_3d
typedef struct VkPhysicalDeviceImage2DViewOf3DFeaturesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           image2DViewOf3D;
    VkBool32           sampler2DViewOf3D;
} VkPhysicalDeviceImage2DViewOf3DFeaturesEXT;
```

This structure describes the following features:

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- image2DViewOf3D indicates that the implementation supports using a 2D view of a 3D image in a descriptor of type VK_DESCRIPTOR_TYPE_STORAGE_IMAGE if the image is created using VK_IMAGE_CREATE_2D_VIEW_COMPATIBLE_BIT_EXT.
- sampler2DViewOf3D indicates that the implementation supports using a 2D view of a 3D image in a descriptor of type VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE or VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER if the image is created using VK_IMAGE_CREATE_2D_VIEW_COMPATIBLE_BIT_EXT.

If the VkPhysicalDeviceImage2DViewOf3DFeaturesEXT structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceImage2DViewOf3DFeaturesEXT **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceImage2DViewOf3DFeaturesEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_IMAGE_2D_VIEW_OF_3D_FEATURES_EXT

The VkPhysicalDeviceImageSlicedViewOf3DFeaturesEXT structure is defined as:

```
// Provided by VK_EXT_image_sliced_view_of_3d
typedef struct VkPhysicalDeviceImageSlicedViewOf3DFeaturesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           imageSlicedViewOf3D;
} VkPhysicalDeviceImageSlicedViewOf3DFeaturesEXT;
```

The members of the VkPhysicalDeviceImageSlicedViewOf3DFeaturesEXT structure describe the following features:

- imageSlicedViewOf3D indicates that the implementation supports using a sliced view of a 3D image in a descriptor of type VK_DESCRIPTOR_TYPE_STORAGE_IMAGE by using a VkImageViewSlicedCreateInfoEXT structure when creating the view.

If the VkPhysicalDeviceImageSlicedViewOf3DFeaturesEXT structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceImageSlicedViewOf3DFeaturesEXT **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceImageSlicedViewOf3DFeaturesEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_IMAGE_SLICED_VIEW_OF_3D_FEATURES_EXT

The VkPhysicalDeviceImageCompressionControlFeaturesEXT structure is defined as:

```
// Provided by VK_EXT_image_compression_control
typedef struct VkPhysicalDeviceImageCompressionControlFeaturesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           imageCompressionControl;
} VkPhysicalDeviceImageCompressionControlFeaturesEXT;
```

This structure describes the following feature:

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- imageCompressionControl indicates that the implementation supports providing controls for image compression at image creation time.

If the VkPhysicalDeviceImageCompressionControlFeaturesEXT structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceImageCompressionControlFeaturesEXT **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceImageCompressionControlFeaturesEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_IMAGE_COMPRESSION_CONTROL_FEATURES_EXT

The VkPhysicalDeviceImageCompressionControlSwapchainFeaturesEXT structure is defined as:

```
// Provided by VK_EXT_image_compression_control_swapchain
typedef struct VkPhysicalDeviceImageCompressionControlSwapchainFeaturesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           imageCompressionControlSwapchain;
} VkPhysicalDeviceImageCompressionControlSwapchainFeaturesEXT;
```

This structure describes the following feature:

- sType is a VkStructureType value identifying this structure.

- pNext is NULL or a pointer to a structure extending this structure.
- imageCompressionControlSwapchain indicates that the implementation supports controlling image controls per swapchain and querying image compression properties per surface.

If the VkPhysicalDeviceImageCompressionControlSwapchainFeaturesEXT structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceImageCompressionControlSwapchainFeaturesEXT **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceImageCompressionControlSwapchainFeaturesEXT-sType-sType sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_IMAGE_COMPRESSION_CONTROL_SWAPCHAIN_FEATURES_EXT

The VkPhysicalDeviceRayTracingPositionFetchFeaturesKHR structure is defined as:

```
// Provided by VK_KHR_ray_tracing_position_fetch
typedef struct VkPhysicalDeviceRayTracingPositionFetchFeaturesKHR {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           rayTracingPositionFetch;
} VkPhysicalDeviceRayTracingPositionFetchFeaturesKHR;
```

This structure describes the following feature:

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- rayTracingPositionFetch indicates that the implementation supports fetching the object space vertex positions of a hit triangle.

If the VkPhysicalDeviceRayTracingPositionFetchFeaturesKHR structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceRayTracingPositionFetchFeaturesKHR **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceRayTracingPositionFetchFeaturesKHR-sType-sType sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_RAY_TRACING_POSITION_FETCH_FEATURES_KHR

The VkPhysicalDeviceShaderFloatControls2FeaturesKHR structure is defined as:

```
// Provided by VK_KHR_shader_float_controls2
typedef struct VkPhysicalDeviceShaderFloatControls2FeaturesKHR {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           shaderFloatControls2;
} VkPhysicalDeviceShaderFloatControls2FeaturesKHR;
```

This structure describes the following feature:

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `shaderFloatControls2` specifies whether shader modules **can** declare the `FloatControls2` capability.

If the `VkPhysicalDeviceShaderFloatControls2FeaturesKHR` structure is included in the `pNext` chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. `VkPhysicalDeviceShaderFloatControls2FeaturesKHR` **can** also be used in the `pNext` chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceShaderFloatControls2FeaturesKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_FLOAT_CONTROLS_2_FEATURES_KHR`

The `VkPhysicalDeviceShaderEarlyAndLateFragmentTestsFeaturesAMD` structure is defined as:

```
// Provided by VK_AMD_shader_early_and_late_fragment_tests
typedef struct VkPhysicalDeviceShaderEarlyAndLateFragmentTestsFeaturesAMD {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           shaderEarlyAndLateFragmentTests;
} VkPhysicalDeviceShaderEarlyAndLateFragmentTestsFeaturesAMD;
```

This structure describes the following feature:

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `shaderEarlyAndLateFragmentTests` indicates whether the implementation supports the `EarlyAndLateFragmentTestsAMD Execution Mode`.

If the `VkPhysicalDeviceShaderEarlyAndLateFragmentTestsFeaturesAMD` structure is included in the `pNext` chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. `VkPhysicalDeviceShaderEarlyAndLateFragmentTestsFeaturesAMD` **can** also be used in the `pNext` chain of

VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceShaderEarlyAndLateFragmentTestsFeaturesAMD-sType-sType sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_EARLY_AND_LATE_FRAGMENT_TESTS_FEATURES_AMD

The VkPhysicalDeviceAntiLagFeaturesAMD structure is defined as:

```
// Provided by VK_AMD_anti_lag
typedef struct VkPhysicalDeviceAntiLagFeaturesAMD {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           antiLag;
} VkPhysicalDeviceAntiLagFeaturesAMD;
```

This structure describes the following feature:

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- antiLag indicates whether the implementation supports AMD Radeon™ Anti-Lag functionality. The antiLag feature only supports a single GPU and **must** not be enabled if VkDeviceGroupDeviceCreateInfo::physicalDeviceCount is greater than 1.

If the VkPhysicalDeviceAntiLagFeaturesAMD structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceAntiLagFeaturesAMD **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceAntiLagFeaturesAMD-sType-sType sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_ANTI_LAG_FEATURES_AMD

The VkPhysicalDeviceNonSeamlessCubeMapFeaturesEXT structure is defined as:

```
// Provided by VK_EXT_non_seamless_cube_map
typedef struct VkPhysicalDeviceNonSeamlessCubeMapFeaturesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           nonSeamlessCubeMap;
} VkPhysicalDeviceNonSeamlessCubeMapFeaturesEXT;
```

This structure describes the following feature:

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- nonSeamlessCubeMap indicates that the implementation supports VK_SAMPLER_CREATE_NON_SEAMLESS_CUBE_MAP_BIT_EXT.

If the VkPhysicalDeviceNonSeamlessCubeMapFeaturesEXT structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceNonSeamlessCubeMapFeaturesEXT **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceNonSeamlessCubeMapFeaturesEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_NON_SEAMLESS_CUBE_MAP_FEATURES_EXT

The VkPhysicalDeviceShaderModuleIdentifierFeaturesEXT structure is defined as:

```
// Provided by VK_EXT_shader_module_identifier
typedef struct VkPhysicalDeviceShaderModuleIdentifierFeaturesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           shaderModuleIdentifier;
} VkPhysicalDeviceShaderModuleIdentifierFeaturesEXT;
```

This structure describes the following feature:

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- shaderModuleIdentifier indicates whether the implementation supports querying an identifier of a VkShaderModule or VkShaderModuleCreateInfo structure, and creating pipelines from identifiers only.

If the VkPhysicalDeviceShaderModuleIdentifierFeaturesEXT structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceShaderModuleIdentifierFeaturesEXT **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceShaderModuleIdentifierFeaturesEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_MODULE_IDENTIFIER_FEATURES_EXT

The VkPhysicalDeviceTilePropertiesFeaturesQCOM structure is defined as:

```
// Provided by VK_QCOM_tile_properties
typedef struct VkPhysicalDeviceTilePropertiesFeaturesQCOM {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           tileProperties;
} VkPhysicalDeviceTilePropertiesFeaturesQCOM;
```

This structure describes the following features:

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `tileProperties` indicates that the implementation supports queries for returning tile properties.

If the `VkPhysicalDeviceTilePropertiesFeaturesQCOM` structure is included in the `pNext` chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. `VkPhysicalDeviceTilePropertiesFeaturesQCOM` **can** also be used in the `pNext` chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceTilePropertiesFeaturesQCOM-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_TILE_PROPERTIES_FEATURES_QCOM`

The `VkPhysicalDeviceImageProcessingFeaturesQCOM` structure is defined as:

```
// Provided by VK_QCOM_image_processing
typedef struct VkPhysicalDeviceImageProcessingFeaturesQCOM {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           textureSampleWeighted;
    VkBool32           textureBoxFilter;
    VkBool32           textureBlockMatch;
} VkPhysicalDeviceImageProcessingFeaturesQCOM;
```

This structure describes the following features:

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `textureSampleWeighted` indicates that the implementation supports shader modules that declare the `TextureSampleWeightedQCOM` capability.
- `textureBoxFilter` indicates that the implementation supports shader modules that declare the `TextureBoxFilterQCOM` capability.
- `textureBlockMatch` indicates that the implementation supports shader modules that declare the

TextureBlockMatchQCOM capability.

If the VkPhysicalDeviceImageProcessingFeaturesQCOM structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceImageProcessingFeaturesQCOM **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceImageProcessingFeaturesQCOM-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_IMAGE_PROCESSING_FEATURES_QCOM

The VkPhysicalDeviceImageProcessing2FeaturesQCOM structure is defined as:

```
// Provided by VK_QCOM_image_processing2
typedef struct VkPhysicalDeviceImageProcessing2FeaturesQCOM {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           textureBlockMatch2;
} VkPhysicalDeviceImageProcessing2FeaturesQCOM;
```

This structure describes the following features:

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- textureBlockMatch2 indicates that the implementation supports shader modules that declare the TextureBlockMatch2QCOM capability.

If the VkPhysicalDeviceImageProcessing2FeaturesQCOM structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceImageProcessing2FeaturesQCOM **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceImageProcessing2FeaturesQCOM-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_IMAGE_PROCESSING_2_FEATURES_QCOM

The VkPhysicalDeviceDepthClampZeroOneFeaturesEXT structure is defined as:

```
// Provided by VK_EXT_depth_clamp_zero_one
typedef struct VkPhysicalDeviceDepthClampZeroOneFeaturesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           depthClampZeroOne;
} VkPhysicalDeviceDepthClampZeroOneFeaturesEXT;
```

This structure describes the following feature:

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `depthClampZeroOne` indicates that the implementation supports clamping the depth to a range of `0` to `1`.

If the `VkPhysicalDeviceDepthClampZeroOneFeaturesEXT` structure is included in the `pNext` chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. `VkPhysicalDeviceDepthClampZeroOneFeaturesEXT` **can** also be used in the `pNext` chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceDepthClampZeroOneFeaturesEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DEPTH_CLAMP_ZERO_ONE_FEATURES_EXT`

The `VkPhysicalDeviceShaderTileImageFeaturesEXT` structure is defined as:

```
// Provided by VK_EXT_shader_tile_image
typedef struct VkPhysicalDeviceShaderTileImageFeaturesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           shaderTileImageColorReadAccess;
    VkBool32           shaderTileImageDepthReadAccess;
    VkBool32           shaderTileImageStencilReadAccess;
} VkPhysicalDeviceShaderTileImageFeaturesEXT;
```

The members of the `VkPhysicalDeviceShaderTileImageFeaturesEXT` structure describe the following features:

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `shaderTileImageColorReadAccess` indicates that the implementation supports the `TileImageColorReadAccessEXT` SPIR-V capability.
- `shaderTileImageDepthReadAccess` indicates that the implementation supports the

TileImageDepthReadAccessEXT SPIR-V capability.

- shaderTileImageStencilReadAccess indicates that the implementation supports the TileImageStencilReadAccessEXT SPIR-V capability.

If the VkPhysicalDeviceShaderTileImageFeaturesEXT structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceShaderTileImageFeaturesEXT **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceShaderTileImageFeaturesEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_TILE_IMAGE_FEATURES_EXT

The VkPhysicalDeviceDepthBiasControlFeaturesEXT structure is defined as:

```
// Provided by VK_EXT_depth_bias_control
typedef struct VkPhysicalDeviceDepthBiasControlFeaturesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           depthBiasControl;
    VkBool32           leastRepresentableValueForceUnormRepresentation;
    VkBool32           floatRepresentation;
    VkBool32           depthBiasExact;
} VkPhysicalDeviceDepthBiasControlFeaturesEXT;
```

This structure describes the following feature:

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- depthBiasControl indicates whether the implementation supports the vkCmdSetDepthBias2EXT command and the VkDepthBiasRepresentationInfoEXT structure.
- leastRepresentableValueForceUnormRepresentation indicates whether the implementation supports using the VK_DEPTH_BIAS_REPRESENTATION_LEAST_REPRESENTABLE_VALUE_FORCE_UNORM_EXT depth bias representation.
- floatRepresentation indicates whether the implementation supports using the VK_DEPTH_BIAS_REPRESENTATION_FLOAT_EXT depth bias representation.
- depthBiasExact indicates whether the implementation supports forcing depth bias to not be scaled to ensure a minimum resolvable difference using VkDepthBiasRepresentationInfoEXT::depthBiasExact.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceDepthBiasControlFeaturesEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DEPTH_BIAS_CONTROL_FEATURES_EXT`

The `VkPhysicalDeviceAddressBindingReportFeaturesEXT` structure is defined as:

```
// Provided by VK_EXT_device_address_binding_report
typedef struct VkPhysicalDeviceAddressBindingReportFeaturesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           reportAddressBinding;
} VkPhysicalDeviceAddressBindingReportFeaturesEXT;
```

This structure describes the following feature:

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `reportAddressBinding` indicates whether this implementation supports reporting the binding of GPU virtual address ranges to Vulkan objects.

If the `VkPhysicalDeviceAddressBindingReportFeaturesEXT` structure is included in the `pNext` chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. `VkPhysicalDeviceAddressBindingReportFeaturesEXT` **can** also be used in the `pNext` chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceAddressBindingReportFeaturesEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_ADDRESS_BINDING_REPORT_FEATURES_EXT`

The `VkPhysicalDeviceOpticalFlowFeaturesNV` structure is defined as:

```
// Provided by VK_NV_optical_flow
typedef struct VkPhysicalDeviceOpticalFlowFeaturesNV {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           opticalFlow;
} VkPhysicalDeviceOpticalFlowFeaturesNV;
```

This structure describes the following feature:

- `sType` is a VkStructureType value identifying this structure.

- pNext is NULL or a pointer to a structure extending this structure.
- opticalFlow indicates whether the implementation supports optical flow.

If the VkPhysicalDeviceOpticalFlowFeaturesNV structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceOpticalFlowFeaturesNV **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceOpticalFlowFeaturesNV-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_OPTICAL_FLOW_FEATURES_NV

The VkPhysicalDeviceFaultFeaturesEXT structure is defined as:

```
// Provided by VK_EXT_device_fault
typedef struct VkPhysicalDeviceFaultFeaturesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           deviceFault;
    VkBool32           deviceFaultVendorBinary;
} VkPhysicalDeviceFaultFeaturesEXT;
```

The members of the VkPhysicalDeviceFaultFeaturesEXT structure describe the following features:

- deviceFault indicates that the implementation supports the reporting of device fault information.
- deviceFaultVendorBinary indicates that the implementation supports the generation of vendor-specific binary crash dumps. These may provide additional information when imported into vendor-specific external tools.

If the VkPhysicalDeviceFaultFeaturesEXT structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceFaultFeaturesEXT **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceFaultFeaturesEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_FAULT_FEATURES_EXT

The VkPhysicalDevicePipelineLibraryGroupHandlesFeaturesEXT structure is defined as:

```
// Provided by VK_EXT_pipeline_library_group_handles
typedef struct VkPhysicalDevicePipelineLibraryGroupHandlesFeaturesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           pipelineLibraryGroupHandles;
} VkPhysicalDevicePipelineLibraryGroupHandlesFeaturesEXT;
```

This structure describes the following features:

- `pipelineLibraryGroupHandles` indicates whether the implementation supports querying group handles directly from a ray tracing pipeline library, and guarantees bitwise identical group handles for such libraries when linked into other pipelines.

If the `VkPhysicalDevicePipelineLibraryGroupHandlesFeaturesEXT` structure is included in the `pNext` chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. `VkPhysicalDevicePipelineLibraryGroupHandlesFeaturesEXT` **can** also be used in the `pNext` chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDevicePipelineLibraryGroupHandlesFeaturesEXT-sType-sType `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PIPELINE_LIBRARY_GROUP_HANDLES_FEATURES_EXT`

The `VkPhysicalDeviceShaderObjectFeaturesEXT` structure is defined as:

```
// Provided by VK_EXT_shader_object
typedef struct VkPhysicalDeviceShaderObjectFeaturesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           shaderObject;
} VkPhysicalDeviceShaderObjectFeaturesEXT;
```

This structure describes the following feature:

- `shaderObject` indicates whether the implementation supports shader objects.

If the `VkPhysicalDeviceShaderObjectFeaturesEXT` structure is included in the `pNext` chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. `VkPhysicalDeviceShaderObjectFeaturesEXT` **can** also be used in the `pNext` chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceShaderObjectFeaturesEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_OBJECT_FEATURES_EXT`

The `VkPhysicalDeviceShaderCoreBuiltinsFeaturesARM` structure is defined as:

```
// Provided by VK_ARM_shader_core_builtins
typedef struct VkPhysicalDeviceShaderCoreBuiltinsFeaturesARM {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           shaderCoreBuiltins;
} VkPhysicalDeviceShaderCoreBuiltinsFeaturesARM;
```

This structure describes the following feature:

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `shaderCoreBuiltins` indicates whether the implementation supports the SPIR-V `CoreBuiltinsARM` capability.

If the `VkPhysicalDeviceShaderCoreBuiltinsFeaturesARM` structure is included in the `pNext` chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. `VkPhysicalDeviceShaderCoreBuiltinsFeaturesARM` **can** also be used in the `pNext` chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceShaderCoreBuiltinsFeaturesARM-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_CORE_BUILTINS_FEATURES_ARM`

The `VkPhysicalDeviceFrameBoundaryFeaturesEXT` structure is defined as:

```
// Provided by VK_EXT_frame_boundary
typedef struct VkPhysicalDeviceFrameBoundaryFeaturesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           frameBoundary;
} VkPhysicalDeviceFrameBoundaryFeaturesEXT;
```

This structure describes the following feature:

- `sType` is a VkStructureType value identifying this structure.

- pNext is NULL or a pointer to a structure extending this structure.
- frameBoundary indicates whether the implementation supports frame boundary information.

If the VkPhysicalDeviceFrameBoundaryFeaturesEXT structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceFrameBoundaryFeaturesEXT **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceFrameBoundaryFeaturesEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_FRAME_BOUNDARY_FEATURES_EXT

The VkPhysicalDeviceSwapchainMaintenance1FeaturesEXT structure is defined as:

```
// Provided by VK_EXT_swapchain_maintenance1
typedef struct VkPhysicalDeviceSwapchainMaintenance1FeaturesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           swapchainMaintenance1;
} VkPhysicalDeviceSwapchainMaintenance1FeaturesEXT;
```

This structure describes the following feature:

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- swapchainMaintenance1 indicates that the implementation supports the following:
  - VkSwapchainPresentFenceInfoEXT, specifying a fence that is signaled when the resources associated with a present operation **can** be safely destroyed.
  - VkSwapchainPresentModesCreateInfoEXT and VkSwapchainPresentModeInfoEXT, allowing the swapchain to switch present modes without a need for recreation.
  - VkSwapchainPresentScalingCreateInfoEXT, specifying the scaling behavior of the swapchain in presence of window resizing.
  - The VK_SWAPCHAIN_CREATE_DEFERRED_MEMORY_ALLOCATION_BIT_EXT flag, allowing the implementation to defer the allocation of swapchain image memory until first acquisition.
  - vkReleaseSwapchainImagesEXT, allowing acquired swapchain images to be released without presenting them.

If the VkPhysicalDeviceSwapchainMaintenance1FeaturesEXT structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceSwapchainMaintenance1FeaturesEXT **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceSwapchainMaintenance1FeaturesEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SWAPCHAIN_MAINTENANCE_1_FEATURES_EXT`

The `VkPhysicalDeviceDynamicRenderingUnusedAttachmentsFeaturesEXT` structure is defined as:

```
// Provided by VK_EXT_dynamic_rendering_unused_attachments
typedef struct VkPhysicalDeviceDynamicRenderingUnusedAttachmentsFeaturesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           dynamicRenderingUnusedAttachments;
} VkPhysicalDeviceDynamicRenderingUnusedAttachmentsFeaturesEXT;
```

This structure describes the following feature:

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `dynamicRenderingUnusedAttachments` indicates that the implementation supports binding graphics pipelines within a render pass instance where any pipeline VkPipelineRenderingCreateInfo::`pColorAttachmentFormats` element with a format other than `VK_FORMAT_UNDEFINED` is allowed with a corresponding VkRenderingInfo::`pColorAttachments` element with an `imageView` equal to VK_NULL_HANDLE, or any pipeline VkPipelineRenderingCreateInfo::`pColorAttachmentFormats` element with a `VK_FORMAT_UNDEFINED` format is allowed with a corresponding VkRenderingInfo::`pColorAttachments` element with a non-VK_NULL_HANDLE `imageView`. Also a VkPipelineRenderingCreateInfo::`depthAttachmentFormat` other than `VK_FORMAT_UNDEFINED` is allowed with a VK_NULL_HANDLE VkRenderingInfo::`pDepthAttachment`, or a VkPipelineRenderingCreateInfo::`depthAttachmentFormat` of `VK_FORMAT_UNDEFINED` is allowed with a non-VK_NULL_HANDLE VkRenderingInfo::`pDepthAttachment`. Also a VkPipelineRenderingCreateInfo::`stencilAttachmentFormat` other than `VK_FORMAT_UNDEFINED` is allowed with a VK_NULL_HANDLE VkRenderingInfo::`pStencilAttachment`, or a VkPipelineRenderingCreateInfo::`stencilAttachmentFormat` of `VK_FORMAT_UNDEFINED` is allowed with a non-VK_NULL_HANDLE VkRenderingInfo::`pStencilAttachment`. Any writes to a VkRenderingInfo::`pColorAttachments`, VkRenderingInfo::`pDepthAttachment`, or VkRenderingInfo::`pStencilAttachment` with VK_NULL_HANDLE are discarded.

If the `VkPhysicalDeviceDynamicRenderingUnusedAttachmentsFeaturesEXT` structure is included in the `pNext` chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. `VkPhysicalDeviceDynamicRenderingUnusedAttachmentsFeaturesEXT` **can** also be used in the `pNext` chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceDynamicRenderingUnusedAttachmentsFeaturesEXT-sType-sType

sType must be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DYNAMIC_RENDERING_UNUSED_ATTACHMENTS_FEATURES_EXT

The VkPhysicalDeviceRayTracingInvocationReorderFeaturesNV structure is defined as:

```
// Provided by VK_NV_ray_tracing_invocation_reorder
typedef struct VkPhysicalDeviceRayTracingInvocationReorderFeaturesNV {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           rayTracingInvocationReorder;
} VkPhysicalDeviceRayTracingInvocationReorderFeaturesNV;
```

This structure describes the following feature:

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- rayTracingInvocationReorder indicates that the implementation supports SPV_NV_shader_invocation_reorder.

If the VkPhysicalDeviceRayTracingInvocationReorderFeaturesNV structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceRayTracingInvocationReorderFeaturesNV **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceRayTracingInvocationReorderFeaturesNV-sType-sType sType must be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_RAY_TRACING_INVOCATION_REORDER_FEATURES_NV

The VkPhysicalDeviceExtendedSparseAddressSpaceFeaturesNV structure is defined as:

```
// Provided by VK_NV_extended_sparse_address_space
typedef struct VkPhysicalDeviceExtendedSparseAddressSpaceFeaturesNV {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           extendedSparseAddressSpace;
} VkPhysicalDeviceExtendedSparseAddressSpaceFeaturesNV;
```

This structure describes the following feature:

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.

- extendedSparseAddressSpace indicates that the implementation supports allowing certain usages of sparse memory resources to exceed VkPhysicalDeviceLimits::sparseAddressSpaceSize. See VkPhysicalDeviceExtendedSparseAddressSpacePropertiesNV.

If the VkPhysicalDeviceExtendedSparseAddressSpaceFeaturesNV structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceExtendedSparseAddressSpaceFeaturesNV **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceExtendedSparseAddressSpaceFeaturesNV-sType-sType sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTENDED_SPARSE_ADDRESS_SPACE_FEATURES_NV

The VkPhysicalDeviceMultiviewPerViewViewportsFeaturesQCOM structure is defined as:

```
// Provided by VK_QCOM_multiview_per_view_viewports
typedef struct VkPhysicalDeviceMultiviewPerViewViewportsFeaturesQCOM {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           multiviewPerViewViewports;
} VkPhysicalDeviceMultiviewPerViewViewportsFeaturesQCOM;
```

This structure describes the following features:

- multiviewPerViewViewports indicates that the implementation supports multiview per-view viewports.

If the VkPhysicalDeviceMultiviewPerViewViewportsFeaturesQCOM structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceMultiviewPerViewViewportsFeaturesQCOM **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceMultiviewPerViewViewportsFeaturesQCOM-sType-sType sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MULTIVIEW_PER_VIEW_VIEWPORTS_FEATURES_QCOM

The VkPhysicalDeviceMultiviewPerViewRenderAreasFeaturesQCOM structure is defined as:

```
// Provided by VK_QCOM_multiview_per_view_render_areas
typedef struct VkPhysicalDeviceMultiviewPerViewRenderAreasFeaturesQCOM {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           multiviewPerViewRenderAreas;
} VkPhysicalDeviceMultiviewPerViewRenderAreasFeaturesQCOM;
```

This structure describes the following features:

- `multiviewPerViewRenderAreas` indicates that the implementation supports multiview per-view render areas.

If the `VkPhysicalDeviceMultiviewPerViewRenderAreasFeaturesQCOM` structure is included in the `pNext` chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. `VkPhysicalDeviceMultiviewPerViewRenderAreasFeaturesQCOM` **can** also be used in the `pNext` chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceMultiviewPerViewRenderAreasFeaturesQCOM-sType-sType `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MULTIVIEW_PER_VIEW_RENDER_AREAS_FEATURES_QCOM`

The `VkPhysicalDeviceClusterCullingShaderFeaturesHUAWEI` structure is defined as:

```
// Provided by VK_HUAWEI_cluster_culling_shader
typedef struct VkPhysicalDeviceClusterCullingShaderFeaturesHUAWEI {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           clustercullingShader;
    VkBool32           multiviewClusterCullingShader;
} VkPhysicalDeviceClusterCullingShaderFeaturesHUAWEI;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `clustercullingShader` specifies whether cluster culling shader is supported.
- `multiviewClusterCullingShader` specifies whether multiview is supported.

If the `VkPhysicalDeviceClusterCullingShaderFeaturesHUAWEI` structure is included in the `pNext` chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. `VkPhysicalDeviceClusterCullingShaderFeaturesHUAWEI` **can** also be used in the `pNext` chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceClusterCullingShaderFeaturesHUAWEI-sType-sType sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_CLUSTER_CULLING_SHADER_FEATURES_HUAWEI

To query whether a Cluster Culling Shader supports the per-cluster shading rate feature, include a VkPhysicalDeviceClusterCullingShaderVrsFeaturesHUAWEI structure in the pNext chain of the VkPhysicalDeviceClusterCullingShaderFeaturesHUAWEI structure. This structure is defined as:

```
// Provided by VK_HUAWEI_cluster_culling_shader
typedef struct VkPhysicalDeviceClusterCullingShaderVrsFeaturesHUAWEI {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           clusterShadingRate;
} VkPhysicalDeviceClusterCullingShaderVrsFeaturesHUAWEI;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- clusterShadingRate specifies whether per-cluster shading rates is supported.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceClusterCullingShaderVrsFeaturesHUAWEI-sType-sType sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_CLUSTER_CULLING_SHADER_VRS_FEATURES_HUAWEI

The VkPhysicalDeviceShaderEnqueueFeaturesAMDX structure is defined as:

```
// Provided by VK_AMDX_shader_enqueue
typedef struct VkPhysicalDeviceShaderEnqueueFeaturesAMDX {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           shaderEnqueue;
} VkPhysicalDeviceShaderEnqueueFeaturesAMDX;
```

This structure describes the following feature:

- shaderEnqueue indicates whether the implementation supports execution graphs.

If the VkPhysicalDeviceShaderEnqueueFeaturesAMDX structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceShaderEnqueueFeaturesAMDX **can** also be used in the pNext chain of

VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceShaderEnqueueFeaturesAMDX-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_ENQUEUE_FEATURES_AMDX`

The `VkPhysicalDeviceCubicClampFeaturesQCOM` structure is defined as:

```
// Provided by VK_QCOM_filter_cubic_clamp
typedef struct VkPhysicalDeviceCubicClampFeaturesQCOM {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           cubicRangeClamp;
} VkPhysicalDeviceCubicClampFeaturesQCOM;
```

This structure describes the following features:

- `cubicRangeClamp` indicates that the implementation supports cubic filtering in combination with a texel range clamp.

If the `VkPhysicalDeviceCubicClampFeaturesQCOM` structure is included in the `pNext` chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. `VkPhysicalDeviceCubicClampFeaturesQCOM` **can** also be used in the `pNext` chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceCubicClampFeaturesQCOM-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_CUBIC_CLAMP_FEATURES_QCOM`

The `VkPhysicalDeviceYcbcrDegammaFeaturesQCOM` structure is defined as:

```
// Provided by VK_QCOM_ycbcr_degamma
typedef struct VkPhysicalDeviceYcbcrDegammaFeaturesQCOM {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           ycbcrDegamma;
} VkPhysicalDeviceYcbcrDegammaFeaturesQCOM;
```

This structure describes the following features:

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `ycbcrDegamma` indicates whether the implementation supports $Y'C_BC_R$ degamma.

If the VkPhysicalDeviceYcbcrDegammaFeaturesQCOM structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceYcbcrDegammaFeaturesQCOM **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceYcbcrDegammaFeaturesQCOM-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_YCBCR_DEGAMMA_FEATURES_QCOM

The VkPhysicalDeviceCubicWeightsFeaturesQCOM structure is defined as:

```
// Provided by VK_QCOM_filter_cubic_weights
typedef struct VkPhysicalDeviceCubicWeightsFeaturesQCOM {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           selectableCubicWeights;
} VkPhysicalDeviceCubicWeightsFeaturesQCOM;
```

This structure describes the following feature:

- selectableCubicWeights indicates that the implementation supports the selection of filter cubic weights.

If the VkPhysicalDeviceCubicWeightsFeaturesQCOM structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceCubicWeightsFeaturesQCOM **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceCubicWeightsFeaturesQCOM-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_CUBIC_WEIGHTS_FEATURES_QCOM

The VkPhysicalDeviceDescriptorPoolOverallocationFeaturesNV structure is defined as:

```
// Provided by VK_NV_descriptor_pool_overallocation
typedef struct VkPhysicalDeviceDescriptorPoolOverallocationFeaturesNV {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           descriptorPoolOverallocation;
} VkPhysicalDeviceDescriptorPoolOverallocationFeaturesNV;
```

This structure describes the following feature:

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- descriptorPoolOverallocation indicates that the implementation allows the application to opt into descriptor pool overallocation by creating the descriptor pool with VK_DESCRIPTOR_POOL_CREATE_ALLOW_OVERALLOCATION_SETS_BIT_NV and/or VK_DESCRIPTOR_POOL_CREATE_ALLOW_OVERALLOCATION_POOLS_BIT_NV flags.

If the VkPhysicalDeviceDescriptorPoolOverallocationFeaturesNV structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceDescriptorPoolOverallocationFeaturesNV **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceDescriptorPoolOverallocationFeaturesNV-sType-sType sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DESCRIPTOR_POOL_OVERALLOCATION_FEATURES_NV

The VkPhysicalDevicePerStageDescriptorSetFeaturesNV structure is defined as:

```
// Provided by VK_NV_per_stage_descriptor_set
typedef struct VkPhysicalDevicePerStageDescriptorSetFeaturesNV {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           perStageDescriptorSet;
    VkBool32           dynamicPipelineLayout;
} VkPhysicalDevicePerStageDescriptorSetFeaturesNV;
```

This structure describes the following feature:

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- perStageDescriptorSet indicates that the implementation allows the VK_DESCRIPTOR_SET_LAYOUT_CREATE_PER_STAGE_BIT_NV descriptor set layout creation flag to be used so the bindings are specified per-stage rather than across all stages.
- dynamicPipelineLayout indicates the implementation allows the layout member of VkBindDescriptorSetsInfoKHR, VkPushConstantsInfoKHR, VkPushDescriptorSetInfoKHR, VkPushDescriptorSetWithTemplateInfoKHR, VkSetDescriptorBufferOffsetsInfoEXT and VkBindDescriptorBufferEmbeddedSamplersInfoEXT to be VK_NULL_HANDLE and VkPipelineLayoutCreateInfo **can** be chained off those structures' pNext instead.

If the VkPhysicalDevicePerStageDescriptorSetFeaturesNV structure is included in the pNext chain of

the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDevicePerStageDescriptorSetFeaturesNV **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDevicePerStageDescriptorSetFeaturesNV-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PER_STAGE_DESCRIPTOR_SET_FEATURES_NV

The VkPhysicalDeviceCudaKernelLaunchFeaturesNV structure is defined as:

```
// Provided by VK_NV_cuda_kernel_launch
typedef struct VkPhysicalDeviceCudaKernelLaunchFeaturesNV {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           cudaKernelLaunchFeatures;
} VkPhysicalDeviceCudaKernelLaunchFeaturesNV;
```

This structure describes the following features:

- cudaKernelLaunchFeatures is non-zero if cuda kernel launch is supported.

If the VkPhysicalDeviceCudaKernelLaunchFeaturesNV structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceCudaKernelLaunchFeaturesNV **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceCudaKernelLaunchFeaturesNV-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_CUDA_KERNEL_LAUNCH_FEATURES_NV

The VkPhysicalDeviceShaderMaximalReconvergenceFeaturesKHR structure is defined as:

```
// Provided by VK_KHR_shader_maximal_reconvergence
typedef struct VkPhysicalDeviceShaderMaximalReconvergenceFeaturesKHR {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           shaderMaximalReconvergence;
} VkPhysicalDeviceShaderMaximalReconvergenceFeaturesKHR;
```

This structure describes the following feature:

- sType is a VkStructureType value identifying this structure.

- pNext is NULL or a pointer to a structure extending this structure.
- shaderMaximalReconvergence specifies whether the implementation supports the shader execution mode MaximallyReconvergesKHR

If the VkPhysicalDevicePrivateDataFeaturesEXT structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDevicePrivateDataFeaturesEXT **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceShaderMaximalReconvergenceFeaturesKHR-sType-sType sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_MAXIMAL_RECONVERGENCE_FEATURES_KHR

The VkPhysicalDeviceExternalFormatResolveFeaturesANDROID structure is defined as:

```
// Provided by VK_ANDROID_external_format_resolve
typedef struct VkPhysicalDeviceExternalFormatResolveFeaturesANDROID {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           externalFormatResolve;
} VkPhysicalDeviceExternalFormatResolveFeaturesANDROID;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- externalFormatResolve specifies whether external format resolves are supported.

If the VkPhysicalDeviceExternalFormatResolveFeaturesANDROID structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceExternalFormatResolveFeaturesANDROID **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceExternalFormatResolveFeaturesANDROID-sType-sType sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTERNAL_FORMAT_RESOLVE_FEATURES_ANDROID

The VkPhysicalDeviceSchedulingControlsFeaturesARM structure is defined as:

```
// Provided by VK_ARM_scheduling_controls
typedef struct VkPhysicalDeviceSchedulingControlsFeaturesARM {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           schedulingControls;
} VkPhysicalDeviceSchedulingControlsFeaturesARM;
```

This structure describes the following features:

- `schedulingControls` indicates that the implementation supports scheduling controls.

If the `VkPhysicalDeviceSchedulingControlsFeaturesARM` structure is included in the `pNext` chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. `VkPhysicalDeviceSchedulingControlsFeaturesARM` **can** also be used in the `pNext` chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceSchedulingControlsFeaturesARM-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SCHEDULING_CONTROLS_FEATURES_ARM`

The `VkPhysicalDeviceRelaxedLineRasterizationFeaturesIMG` structure is defined as:

```
// Provided by VK_IMG_relaxed_line_rasterization
typedef struct VkPhysicalDeviceRelaxedLineRasterizationFeaturesIMG {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           relaxedLineRasterization;
} VkPhysicalDeviceRelaxedLineRasterizationFeaturesIMG;
```

This structure describes the following feature:

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `relaxedLineRasterization` indicates that the implementation supports relaxed line rasterization control.

If the `VkPhysicalDeviceRelaxedLineRasterizationFeaturesIMG` structure is included in the `pNext` chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. `VkPhysicalDeviceRelaxedLineRasterizationFeaturesIMG` **can** also be used in the `pNext` chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceRelaxedLineRasterizationFeaturesIMG-sType-sType sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_RELAXED_LINE_RASTERIZATION_FEATURES_IMG

The VkPhysicalDeviceRenderPassStripedFeaturesARM structure is defined as:

```
// Provided by VK_ARM_render_pass_striped
typedef struct VkPhysicalDeviceRenderPassStripedFeaturesARM {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           renderPassStriped;
} VkPhysicalDeviceRenderPassStripedFeaturesARM;
```

The members of the VkPhysicalDeviceRenderPassStripedFeaturesARM structure describe the following features:

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- renderPassStriped indicates that striped rendering is supported by the implementation.

If the VkPhysicalDeviceRenderPassStripedFeaturesARM structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceRenderPassStripedFeaturesARM **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceRenderPassStripedFeaturesARM-sType-sType sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_RENDER_PASS_STRIPED_FEATURES_ARM

The VkPhysicalDeviceShaderSubgroupRotateFeaturesKHR structure is defined as:

```
// Provided by VK_KHR_shader_subgroup_rotate
typedef struct VkPhysicalDeviceShaderSubgroupRotateFeaturesKHR {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           shaderSubgroupRotate;
    VkBool32           shaderSubgroupRotateClustered;
} VkPhysicalDeviceShaderSubgroupRotateFeaturesKHR;
```

- sType is a VkStructureType value identifying this structure.

- pNext is NULL or a pointer to a structure extending this structure.
- shaderSubgroupRotate specifies whether shader modules **can** declare the GroupNonUniformRotateKHR capability.
- shaderSubgroupRotateClustered specifies whether shader modules **can** use the ClusterSize operand to OpGroupNonUniformRotateKHR.

If the VkPhysicalDeviceShaderSubgroupRotateFeaturesKHR structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceShaderSubgroupRotateFeaturesKHR **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceShaderSubgroupRotateFeaturesKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_SUBGROUP_ROTATE_FEATURES_KHR

The VkPhysicalDeviceShaderExpectAssumeFeaturesKHR structure is defined as:

```
// Provided by VK_KHR_shader_expect_assume
typedef struct VkPhysicalDeviceShaderExpectAssumeFeaturesKHR {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           shaderExpectAssume;
} VkPhysicalDeviceShaderExpectAssumeFeaturesKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- shaderExpectAssume specifies whether shader modules **can** declare the ExpectAssumeKHR capability.

If the VkPhysicalDeviceShaderExpectAssumeFeaturesKHR structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceShaderExpectAssumeFeaturesKHR **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceShaderExpectAssumeFeaturesKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_EXPECT_ASSUME_FEATURES_KHR

The VkPhysicalDeviceDynamicRenderingLocalReadFeaturesKHR structure is defined as:

```
// Provided by VK_KHR_dynamic_rendering_local_read
typedef struct VkPhysicalDeviceDynamicRenderingLocalReadFeaturesKHR {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           dynamicRenderingLocalRead;
} VkPhysicalDeviceDynamicRenderingLocalReadFeaturesKHR;
```

This structure describes the following feature:

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `dynamicRenderingLocalRead` specifies that the implementation supports local reads inside dynamic render pass instances using the vkCmdBeginRenderingKHR command.

If the `VkPhysicalDeviceDynamicRenderingLocalReadFeaturesKHR` structure is included in the `pNext` chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. `VkPhysicalDeviceDynamicRenderingLocalReadFeaturesKHR` **can** also be used in the `pNext` chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceDynamicRenderingLocalReadFeaturesKHR-sType-sType `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DYNAMIC_RENDERING_LOCAL_READ_FEATURES_KHR`

The `VkPhysicalDeviceShaderQuadControlFeaturesKHR` structure is defined as:

```
// Provided by VK_KHR_shader_quad_control
typedef struct VkPhysicalDeviceShaderQuadControlFeaturesKHR {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           shaderQuadControl;
} VkPhysicalDeviceShaderQuadControlFeaturesKHR;
```

This structure describes the following features:

- `shaderQuadControl` indicates whether the implementation supports shaders with the `QuadControlKHR` capability.

If the `VkPhysicalDeviceShaderQuadControlFeaturesKHR` structure is included in the `pNext` chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. `VkPhysicalDeviceShaderQuadControlFeaturesKHR` **can** also be used in the `pNext` chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceShaderQuadControlFeaturesKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_QUAD_CONTROL_FEATURES_KHR`

The `VkPhysicalDeviceMapMemoryPlacedFeaturesEXT` structure is defined as:

```
// Provided by VK_EXT_map_memory_placed
typedef struct VkPhysicalDeviceMapMemoryPlacedFeaturesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           memoryMapPlaced;
    VkBool32           memoryMapRangePlaced;
    VkBool32           memoryUnmapReserve;
} VkPhysicalDeviceMapMemoryPlacedFeaturesEXT;
```

This structure describes the following features:

- `memoryMapPlaced` indicates that the implementation supports placing memory maps at application-specified virtual addresses.
- `memoryMapRangePlaced` indicates that the implementation supports placing memory maps of a subrange of a memory object at application-specified virtual addresses.
- `memoryUnmapReserve` indicates that the implementation supports leaving the memory range reserved when unmapping a memory object.

If the `VkPhysicalDeviceMapMemoryPlacedFeaturesEXT` structure is included in the `pNext` chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. `VkPhysicalDeviceMapMemoryPlacedFeaturesEXT` **can** also be used in the `pNext` chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceMapMemoryPlacedFeaturesEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MAP_MEMORY_PLACED_FEATURES_EXT`

The VkPhysicalDeviceShaderAtomicFloat16VectorFeaturesNV structure is defined as:

```
// Provided by VK_NV_shader_atomic_float16_vector
typedef struct VkPhysicalDeviceShaderAtomicFloat16VectorFeaturesNV {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           shaderFloat16VectorAtomics;
} VkPhysicalDeviceShaderAtomicFloat16VectorFeaturesNV;
```

This structure describes the following features:

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- shaderFloat16VectorAtomics indicates whether shaders **can** perform 16-bit floating-point, 2- and 4-component vector atomic operations.

If the VkPhysicalDeviceShaderAtomicFloat16VectorFeaturesNV structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceShaderAtomicFloat16VectorFeaturesNV **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceShaderAtomicFloat16VectorFeaturesNV-sType-sType sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_ATOMIC_FLOAT16_VECTOR_FEATURES_NV

The VkPhysicalDeviceRawAccessChainsFeaturesNV structure is defined as:

```
// Provided by VK_NV_raw_access_chains
typedef struct VkPhysicalDeviceRawAccessChainsFeaturesNV {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           shaderRawAccessChains;
} VkPhysicalDeviceRawAccessChainsFeaturesNV;
```

This structure describes the following feature:

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- shaderRawAccessChains specifies whether shader modules **can** declare the RawAccessChainsNV capability.

If the VkPhysicalDeviceRawAccessChainsFeaturesNV structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceRawAccessChainsFeaturesNV **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceRawAccessChainsFeaturesNV-sType-sType sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_RAW_ACCESS_CHAINS_FEATURES_NV

The VkPhysicalDeviceCommandBufferInheritanceFeaturesNV structure is defined as:

```
// Provided by VK_NV_command_buffer_inheritance
typedef struct VkPhysicalDeviceCommandBufferInheritanceFeaturesNV {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           commandBufferInheritance;
} VkPhysicalDeviceCommandBufferInheritanceFeaturesNV;
```

This structure describes the following feature:

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- commandBufferInheritance indicates that command buffers executed in a queue inherit graphics and compute state from the previously executed command buffer in that queue.

If the VkPhysicalDeviceCommandBufferInheritanceFeaturesNV structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceCommandBufferInheritanceFeaturesNV **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceCommandBufferInheritanceFeaturesNV-sType-sType sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_COMMAND_BUFFER_INHERITANCE_FEATURES_NV

The VkPhysicalDeviceImageAlignmentControlFeaturesMESA structure is defined as:

```
// Provided by VK_MESA_image_alignment_control
typedef struct VkPhysicalDeviceImageAlignmentControlFeaturesMESA {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           imageAlignmentControl;
} VkPhysicalDeviceImageAlignmentControlFeaturesMESA;
```

This structure describes the following feature:

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- imageAlignmentControl specifies that VkImageAlignmentControlCreateInfoMESA **can** be chained in VkImageCreateInfo

If the VkPhysicalDeviceImageAlignmentControlFeaturesMESA structure is included in the pNext chain of

the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceImageAlignmentControlFeaturesMESA **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceImageAlignmentControlFeaturesMESA-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_IMAGE_ALIGNMENT_CONTROL_FEATURES_MESA

The VkPhysicalDeviceShaderReplicatedCompositesFeaturesEXT structure is defined as:

```
// Provided by VK_EXT_shader_replicated_composites
typedef struct VkPhysicalDeviceShaderReplicatedCompositesFeaturesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           shaderReplicatedComposites;
} VkPhysicalDeviceShaderReplicatedCompositesFeaturesEXT;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- shaderReplicatedComposites specifies whether shader modules **can** declare the ReplicatedCompositesEXT capability.

If the VkPhysicalDeviceShaderReplicatedCompositesFeaturesEXT structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceShaderReplicatedCompositesFeaturesEXT **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceShaderReplicatedCompositesFeaturesEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_REPLICATED_COMPOSITES_FEATURES_EXT

The VkPhysicalDeviceShaderRelaxedExtendedInstructionFeaturesKHR structure is defined as:

```
// Provided by VK_KHR_shader_relaxed_extended_instruction
typedef struct VkPhysicalDeviceShaderRelaxedExtendedInstructionFeaturesKHR {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           shaderRelaxedExtendedInstruction;
} VkPhysicalDeviceShaderRelaxedExtendedInstructionFeaturesKHR;
```

This structure describes the following feature:

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- shaderRelaxedExtendedInstruction specifies whether the implementation supports SPIR-V modules that use the SPV_KHR_relaxed_extended_instruction extension.

If the VkPhysicalDeviceShaderRelaxedExtendedInstructionFeaturesKHR structure is included in the pNext chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. VkPhysicalDeviceShaderRelaxedExtendedInstructionFeaturesKHR **can** also be used in the pNext chain of VkDeviceCreateInfo to selectively enable these features.

**Valid Usage (Implicit)**

- VUID-VkPhysicalDeviceShaderRelaxedExtendedInstructionFeaturesKHR-sType-sType sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_RELAXED_EXTENDED_INSTRUCTION_FEATURES_KHR

## 46.1. Feature Requirements

All Vulkan graphics implementations **must** support the following features:

- If Vulkan 1.0 is supported, robustBufferAccess **must** be supported if VK_KHR_portability_subset is not advertised
- If Vulkan 1.1 is supported, the following features **must** be supported:
  - multiview
  - shaderDrawParameters if VK_KHR_shader_draw_parameters is supported
- If VK_KHR_dynamic_rendering is supported, dynamicRendering **must** be supported
- If VK_KHR_16bit_storage is supported, storageBuffer16BitAccess **must** be supported
- If VK_KHR_imageless_framebuffer is supported, imagelessFramebuffer **must** be supported
- If VK_KHR_performance_query is supported, performanceCounterQueryPools **must** be supported
- If VK_KHR_variable_pointers is supported, variablePointersStorageBuffer **must** be supported
- If VK_KHR_sampler_ycbcr_conversion is supported, samplerYcbcrConversion **must** be supported
- If VK_KHR_shader_subgroup_extended_types is supported, shaderSubgroupExtendedTypes **must** be supported
- If VK_KHR_8bit_storage is supported, storageBuffer8BitAccess **must** be supported
- If VK_KHR_shader_atomic_int64 is supported, shaderBufferInt64Atomics **must** be supported
- If VK_KHR_shader_clock is supported, shaderSubgroupClock **must** be supported
- If VK_KHR_global_priority is supported, globalPriorityQuery **must** be supported

- If VK_KHR_timeline_semaphore is supported, timelineSemaphore **must** be supported
- If VK_KHR_vulkan_memory_model is supported, vulkanMemoryModel **must** be supported
- If VK_KHR_shader_terminate_invocation is supported, shaderTerminateInvocation **must** be supported
- If VK_KHR_fragment_shading_rate is supported, pipelineFragmentShadingRate **must** be supported
- If VK_KHR_dynamic_rendering_local_read is supported, dynamicRenderingLocalRead **must** be supported
- If VK_KHR_shader_quad_control is supported, shaderQuadControl **must** be supported
- If VK_KHR_separate_depth_stencil_layouts is supported, separateDepthStencilLayouts **must** be supported
- If VK_KHR_present_wait is supported, presentWait **must** be supported
- If VK_KHR_uniform_buffer_standard_layout is supported, uniformBufferStandardLayout **must** be supported
- If VK_KHR_buffer_device_address is supported, bufferDeviceAddress **must** be supported
- If VK_KHR_pipeline_executable_properties is supported, pipelineExecutableInfo **must** be supported
- If VK_KHR_shader_integer_dot_product is supported, shaderIntegerDotProduct **must** be supported
- If VK_KHR_present_id is supported, presentId **must** be supported
- If VK_KHR_synchronization2 is supported, synchronization2 **must** be supported
- If VK_KHR_shader_subgroup_uniform_control_flow is supported, shaderSubgroupUniformControlFlow **must** be supported
- If VK_KHR_zero_initialize_workgroup_memory is supported, shaderZeroInitializeWorkgroupMemory **must** be supported
- If VK_KHR_workgroup_memory_explicit_layout is supported, workgroupMemoryExplicitLayout **must** be supported
- If VK_KHR_ray_tracing_maintenance1 is supported, rayTracingMaintenance1 **must** be supported
- If VK_KHR_maintenance4 is supported, maintenance4 **must** be supported
- If VK_KHR_shader_subgroup_rotate is supported, shaderSubgroupRotate **must** be supported
- If VK_KHR_shader_maximal_reconvergence is supported, shaderMaximalReconvergence **must** be supported
- If VK_KHR_maintenance5 is supported, maintenance5 **must** be supported
- If VK_KHR_ray_tracing_position_fetch is supported, rayTracingPositionFetch **must** be supported
- If VK_KHR_pipeline_binary is supported, pipelineBinaries **must** be supported
- If VK_KHR_cooperative_matrix is supported, cooperativeMatrix **must** be supported
- If VK_KHR_video_maintenance1 is supported, videoMaintenance1 **must** be supported
- If VK_KHR_vertex_attribute_divisor is supported, vertexAttributeInstanceRateDivisor **must** be supported

- If VK_KHR_shader_float_controls2 is supported, shaderFloatControls2 **must** be supported
- If VK_KHR_index_type_uint8 is supported, indexTypeUint8 **must** be supported
- If VK_KHR_shader_expect_assume is supported, shaderExpectAssume **must** be supported
- If VK_KHR_maintenance6 is supported, maintenance6 **must** be supported
- If VK_KHR_shader_relaxed_extended_instruction is supported, shaderRelaxedExtendedInstruction **must** be supported
- If VK_KHR_maintenance7 is supported, maintenance7 **must** be supported
- If VK_EXT_transform_feedback is supported, transformFeedback **must** be supported
- If VK_EXT_texture_compression_astc_hdr is supported, textureCompressionASTC_HDR **must** be supported
- If VK_EXT_pipeline_robustness is supported, pipelineRobustness **must** be supported
- If VK_EXT_conditional_rendering is supported, conditionalRendering **must** be supported
- If VK_EXT_depth_clip_enable is supported, depthClipEnable **must** be supported
- If VK_IMG_relaxed_line_rasterization is supported, relaxedLineRasterization **must** be supported
- If VK_AMDX_shader_enqueue is supported, shaderEnqueue **must** be supported
- If VK_EXT_inline_uniform_block is supported, the following features **must** be supported:
  - inlineUniformBlock
  - descriptorBindingInlineUniformBlockUpdateAfterBind if VK_EXT_descriptor_indexing is supported
- If VK_EXT_descriptor_indexing is supported, the following features **must** be supported:
  - shaderSampledImageArrayDynamicIndexing
  - shaderStorageBufferArrayDynamicIndexing
  - shaderUniformTexelBufferArrayDynamicIndexing
  - shaderStorageTexelBufferArrayDynamicIndexing
  - shaderSampledImageArrayNonUniformIndexing
  - shaderStorageBufferArrayNonUniformIndexing
  - shaderUniformTexelBufferArrayNonUniformIndexing
  - descriptorBindingSampledImageUpdateAfterBind
  - descriptorBindingStorageImageUpdateAfterBind
  - descriptorBindingStorageBufferUpdateAfterBind
  - descriptorBindingUniformTexelBufferUpdateAfterBind
  - descriptorBindingStorageTexelBufferUpdateAfterBind
  - descriptorBindingUpdateUnusedWhilePending
  - descriptorBindingPartiallyBound
  - runtimeDescriptorArray

- If VK_EXT_vertex_attribute_divisor is supported, vertexAttributeInstanceRateDivisor **must** be supported
- If VK_EXT_fragment_density_map is supported, fragmentDensityMap **must** be supported
- If VK_EXT_scalar_block_layout is supported, scalarBlockLayout **must** be supported
- If VK_EXT_subgroup_size_control is supported, the following features **must** be supported:
  - subgroupSizeControl
  - computeFullSubgroups
- If VK_EXT_shader_image_atomic_int64 is supported, the following features **must** be supported:
  - shaderInt64
  - shaderImageInt64Atomics
- If VK_EXT_memory_priority is supported, memoryPriority **must** be supported
- If VK_EXT_ycbcr_image_arrays is supported, ycbcrImageArrays **must** be supported
- If VK_EXT_provoking_vertex is supported, provokingVertexLast **must** be supported
- If VK_EXT_host_query_reset is supported, hostQueryReset **must** be supported
- If VK_EXT_index_type_uint8 is supported, indexTypeUint8 **must** be supported
- If VK_EXT_extended_dynamic_state is supported, extendedDynamicState **must** be supported
- If VK_EXT_host_image_copy is supported, hostImageCopy **must** be supported
- If VK_EXT_map_memory_placed is supported, memoryMapPlaced **must** be supported
- If VK_EXT_swapchain_maintenance1 is supported, swapchainMaintenance1 **must** be supported
- If VK_EXT_shader_demote_to_helper_invocation is supported, shaderDemoteToHelperInvocation **must** be supported
- If VK_EXT_texel_buffer_alignment is supported, texelBufferAlignment **must** be supported
- If VK_EXT_depth_bias_control is supported, depthBiasControl **must** be supported
- If VK_EXT_device_memory_report is supported, deviceMemoryReport **must** be supported
- If VK_EXT_custom_border_color is supported, customBorderColors **must** be supported
- If VK_NV_present_barrier is supported, presentBarrier **must** be supported
- If VK_EXT_private_data is supported, privateData **must** be supported
- If VK_EXT_pipeline_creation_cache_control is supported, pipelineCreationCacheControl **must** be supported
- If VK_EXT_descriptor_buffer is supported, descriptorBuffer **must** be supported
- If VK_EXT_graphics_pipeline_library is supported, graphicsPipelineLibrary **must** be supported
- If VK_AMD_shader_early_and_late_fragment_tests is supported, shaderEarlyAndLateFragmentTests **must** be supported
- If VK_EXT_image_robustness is supported, robustImageAccess **must** be supported
- If VK_EXT_image_compression_control is supported, imageCompressionControl **must** be supported
- If VK_EXT_attachment_feedback_loop_layout is supported, attachmentFeedbackLoopLayout **must** be

supported

- If VK_EXT_4444_formats is supported, formatA4R4G4B4 **must** be supported
- If VK_EXT_device_fault is supported, deviceFault **must** be supported
- If VK_VALVE_mutable_descriptor_type is supported, mutableDescriptorType **must** be supported
- If VK_EXT_vertex_input_dynamic_state is supported, vertexInputDynamicState **must** be supported
- If VK_EXT_device_address_binding_report is supported, reportAddressBinding **must** be supported
- If VK_EXT_depth_clip_control is supported, depthClipControl **must** be supported
- If VK_EXT_primitive_topology_list_restart is supported, primitiveTopologyListRestart **must** be supported
- If VK_EXT_frame_boundary is supported, frameBoundary **must** be supported
- If VK_EXT_multisampled_render_to_single_sampled is supported, multisampledRenderToSingleSampled **must** be supported
- If VK_EXT_extended_dynamic_state2 is supported, extendedDynamicState2 **must** be supported
- If VK_EXT_color_write_enable is supported, colorWriteEnable **must** be supported
- If VK_EXT_primitives_generated_query is supported, primitivesGeneratedQuery **must** be supported
- If VK_EXT_global_priority_query is supported, globalPriorityQuery **must** be supported
- If VK_EXT_image_view_min_lod is supported, minLod **must** be supported
- If VK_EXT_multi_draw is supported, multiDraw **must** be supported
- If VK_EXT_image_2d_view_of_3d is supported, image2DViewOf3D **must** be supported
- If VK_EXT_shader_tile_image is supported, shaderTileImageColorReadAccess **must** be supported
- If VK_EXT_opacity_micromap is supported, micromap **must** be supported
- If VK_NV_displacement_micromap is supported, displacementMicromap **must** be supported
- If VK_EXT_border_color_swizzle is supported, borderColorSwizzle **must** be supported
- If VK_ARM_scheduling_controls is supported, schedulingControls **must** be supported
- If VK_EXT_image_sliced_view_of_3d is supported, imageSlicedViewOf3D **must** be supported
- If VK_EXT_depth_clamp_zero_one is supported, depthClampZeroOne **must** be supported
- If VK_EXT_non_seamless_cube_map is supported, nonSeamlessCubeMap **must** be supported
- If VK_ARM_render_pass_striped is supported, renderPassStriped **must** be supported
- If VK_NV_device_generated_commands_compute is supported, deviceGeneratedCompute **must** be supported
- If VK_NV_linear_color_attachment is supported, linearColorAttachment **must** be supported
- If VK_EXT_image_compression_control_swapchain is supported, imageCompressionControlSwapchain **must** be supported
- If VK_QCOM_image_processing is supported, the following features **must** be supported:
  - textureSampleWeighted
  - textureBlockMatch

    - textureBoxFilter
- If VK_EXT_nested_command_buffer is supported, nestedCommandBuffer **must** be supported
- If VK_EXT_subpass_merge_feedback is supported, subpassMergeFeedback **must** be supported
- If VK_EXT_shader_module_identifier is supported, shaderModuleIdentifier **must** be supported
- If VK_EXT_legacy_dithering is supported, legacyDithering **must** be supported
- If VK_EXT_pipeline_protected_access is supported, pipelineProtectedAccess **must** be supported
- If VK_ANDROID_external_format_resolve is supported, externalFormatResolve **must** be supported
- If VK_EXT_shader_object is supported, shaderObject **must** be supported
- If VK_QCOM_tile_properties is supported, tileProperties **must** be supported
- If VK_QCOM_multiview_per_view_viewports is supported, multiviewPerViewViewports **must** be supported
- If VK_NV_extended_sparse_address_space is supported, extendedSparseAddressSpace **must** be supported
- If VK_EXT_mutable_descriptor_type is supported, mutableDescriptorType **must** be supported
- If VK_EXT_legacy_vertex_attributes is supported, legacyVertexAttributes **must** be supported
- If VK_EXT_pipeline_library_group_handles is supported, pipelineLibraryGroupHandles **must** be supported
- If VK_NV_per_stage_descriptor_set is supported, perStageDescriptorSet **must** be supported
- If VK_QCOM_image_processing2 is supported, textureBlockMatch2 **must** be supported
- If VK_QCOM_ycbcr_degamma is supported, ycbcrDegamma **must** be supported
- If VK_EXT_attachment_feedback_loop_dynamic_state is supported, attachmentFeedbackLoopDynamicState **must** be supported
- If VK_NV_descriptor_pool_overallocation is supported, descriptorPoolOverallocation **must** be supported
- If VK_NV_raw_access_chains is supported, shaderRawAccessChains **must** be supported
- If VK_NV_command_buffer_inheritance is supported, commandBufferInheritance **must** be supported
- If VK_NV_shader_atomic_float16_vector is supported, shaderFloat16VectorAtomics **must** be supported
- If VK_EXT_shader_replicated_composites is supported, shaderReplicatedComposites **must** be supported
- If VK_MESA_image_alignment_control is supported, imageAlignmentControl **must** be supported
- If VK_KHR_acceleration_structure is supported, the following features **must** be supported:
    - accelerationStructure
    - bufferDeviceAddress
    - descriptorBindingAccelerationStructureUpdateAfterBind
    - descriptorIndexing if Vulkan 1.2 is supported
- If VK_KHR_ray_tracing_pipeline is supported, the following features **must** be supported:

  - rayTracingPipeline
  - rayTracingPipelineTraceRaysIndirect
  - rayTraversalPrimitiveCulling if VK_KHR_ray_query is supported
- If VK_KHR_ray_query is supported, rayQuery **must** be supported
- If VK_EXT_mesh_shader is supported, the following features **must** be supported:
  - taskShader
  - meshShader
- storageBuffer8BitAccess, if uniformAndStorageBuffer8BitAccess is enabled.
- shaderInt64, if the shaderSharedInt64Atomics or shaderBufferInt64Atomics features are supported.
- storageBuffer16BitAccess, if uniformAndStorageBuffer16BitAccess is enabled.
- shaderImageFloat32Atomics, if the sparseImageFloat32Atomics feature is supported.
- shaderImageFloat32AtomicAdd, if the sparseImageFloat32AtomicAdd feature is supported.
- shaderImageFloat32AtomicMinMax, if the sparseImageFloat32AtomicMinMax feature is supported.
- primitiveFragmentShadingRate if primitiveFragmentShadingRateMeshShader feature is supported.
- fragmentShaderSampleInterlock or fragmentShaderPixelInterlock or fragmentShaderShadingRateInterlock, if the VK_EXT_fragment_shader_interlock extension is supported.
- rectangularLines or bresenhamLines or smoothLines or stippledRectangularLines or stippledBresenhamLines or stippledSmoothLines, if the VK_KHR_line_rasterization extension is supported.
- rectangularLines or bresenhamLines or smoothLines or stippledRectangularLines or stippledBresenhamLines or stippledSmoothLines, if the VK_EXT_line_rasterization extension is supported.
- tessellationShader, if the extendedDynamicState3TessellationDomainOrigin feature is supported.
- depthClamp, if the extendedDynamicState3DepthClampEnable feature is supported.
- fillModeNonSolid, if the extendedDynamicState3PolygonMode feature is supported.
- alphaToOne, if the extendedDynamicState3AlphaToOneEnable feature is supported.
- logicOp, if the extendedDynamicState3LogicOpEnable feature is supported.
- geometryStreams, if the extendedDynamicState3RasterizationStream feature is supported.
- VK_EXT_conservative_rasterization extension, if the extendedDynamicState3ConservativeRasterizationMode feature is supported.
- VK_EXT_conservative_rasterization extension, if the extendedDynamicState3ExtraPrimitiveOverestimationSize feature is supported.
- VK_EXT_sample_locations extension, if the extendedDynamicState3SampleLocationsEnable feature is supported.
- VK_EXT_blend_operation_advanced extension, if the extendedDynamicState3ColorBlendAdvanced feature is supported.

- provokingVertexLast, if the extendedDynamicState3ProvokingVertexMode feature is supported.
- VK_KHR_line_rasterization or VK_EXT_line_rasterization extension, if the extendedDynamicState3LineRasterizationMode feature is supported.
- VK_KHR_line_rasterization or VK_EXT_line_rasterization extension, if the extendedDynamicState3LineStippleEnable feature is supported.
- depthClipControl, if the extendedDynamicState3DepthClipNegativeOneToOne feature is supported.
- VK_NV_clip_space_w_scaling extension, if the extendedDynamicState3ViewportWScalingEnable feature is supported.
- VK_NV_viewport_swizzle extension, if the extendedDynamicState3ViewportSwizzle feature is supported.
- VK_NV_fragment_coverage_to_color extension, if the extendedDynamicState3CoverageToColorEnable feature is supported.
- VK_NV_fragment_coverage_to_color extension, if the extendedDynamicState3CoverageToColorLocation feature is supported.
- VK_NV_framebuffer_mixed_samples extension, if the extendedDynamicState3CoverageModulationMode feature is supported.
- VK_NV_framebuffer_mixed_samples extension, if the extendedDynamicState3CoverageModulationTableEnable feature is supported.
- VK_NV_framebuffer_mixed_samples extension, if the extendedDynamicState3CoverageModulationTable feature is supported.
- coverageReductionMode, if the extendedDynamicState3CoverageReductionMode feature is supported.
- representativeFragmentTest, if the extendedDynamicState3RepresentativeFragmentTestEnable feature is supported.
- shadingRateImage, if the extendedDynamicState3ShadingRateImageEnable feature is supported.
- Either shaderFloat16 or shaderInt8 if VK_KHR_shader_float16_int8 is supported.
- computeDerivativeGroupLinear, if the VK_KHR_compute_shader_derivatives extension is supported.

All other features defined in the Specification are **optional**.

# Chapter 47. Limits

*Limits* are implementation-dependent minimums, maximums, and other device characteristics that an application **may** need to be aware of.

**NOTE** Limits are reported via the basic VkPhysicalDeviceLimits structure as well as the extensible structure VkPhysicalDeviceProperties2, which was added in VK_KHR_get_physical_device_properties2 and included in Vulkan 1.1. When limits are added in future Vulkan versions or extensions, each extension **should** introduce one new limit structure, if needed. This structure **can** be added to the pNext chain of the VkPhysicalDeviceProperties2 structure.

The VkPhysicalDeviceLimits structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkPhysicalDeviceLimits {
    uint32_t              maxImageDimension1D;
    uint32_t              maxImageDimension2D;
    uint32_t              maxImageDimension3D;
    uint32_t              maxImageDimensionCube;
    uint32_t              maxImageArrayLayers;
    uint32_t              maxTexelBufferElements;
    uint32_t              maxUniformBufferRange;
    uint32_t              maxStorageBufferRange;
    uint32_t              maxPushConstantsSize;
    uint32_t              maxMemoryAllocationCount;
    uint32_t              maxSamplerAllocationCount;
    VkDeviceSize          bufferImageGranularity;
    VkDeviceSize          sparseAddressSpaceSize;
    uint32_t              maxBoundDescriptorSets;
    uint32_t              maxPerStageDescriptorSamplers;
    uint32_t              maxPerStageDescriptorUniformBuffers;
    uint32_t              maxPerStageDescriptorStorageBuffers;
    uint32_t              maxPerStageDescriptorSampledImages;
    uint32_t              maxPerStageDescriptorStorageImages;
    uint32_t              maxPerStageDescriptorInputAttachments;
    uint32_t              maxPerStageResources;
    uint32_t              maxDescriptorSetSamplers;
    uint32_t              maxDescriptorSetUniformBuffers;
    uint32_t              maxDescriptorSetUniformBuffersDynamic;
    uint32_t              maxDescriptorSetStorageBuffers;
    uint32_t              maxDescriptorSetStorageBuffersDynamic;
    uint32_t              maxDescriptorSetSampledImages;
    uint32_t              maxDescriptorSetStorageImages;
    uint32_t              maxDescriptorSetInputAttachments;
    uint32_t              maxVertexInputAttributes;
    uint32_t              maxVertexInputBindings;
    uint32_t              maxVertexInputAttributeOffset;
    uint32_t              maxVertexInputBindingStride;
```

```
    uint32_t              maxVertexOutputComponents;
    uint32_t              maxTessellationGenerationLevel;
    uint32_t              maxTessellationPatchSize;
    uint32_t              maxTessellationControlPerVertexInputComponents;
    uint32_t              maxTessellationControlPerVertexOutputComponents;
    uint32_t              maxTessellationControlPerPatchOutputComponents;
    uint32_t              maxTessellationControlTotalOutputComponents;
    uint32_t              maxTessellationEvaluationInputComponents;
    uint32_t              maxTessellationEvaluationOutputComponents;
    uint32_t              maxGeometryShaderInvocations;
    uint32_t              maxGeometryInputComponents;
    uint32_t              maxGeometryOutputComponents;
    uint32_t              maxGeometryOutputVertices;
    uint32_t              maxGeometryTotalOutputComponents;
    uint32_t              maxFragmentInputComponents;
    uint32_t              maxFragmentOutputAttachments;
    uint32_t              maxFragmentDualSrcAttachments;
    uint32_t              maxFragmentCombinedOutputResources;
    uint32_t              maxComputeSharedMemorySize;
    uint32_t              maxComputeWorkGroupCount[3];
    uint32_t              maxComputeWorkGroupInvocations;
    uint32_t              maxComputeWorkGroupSize[3];
    uint32_t              subPixelPrecisionBits;
    uint32_t              subTexelPrecisionBits;
    uint32_t              mipmapPrecisionBits;
    uint32_t              maxDrawIndexedIndexValue;
    uint32_t              maxDrawIndirectCount;
    float                 maxSamplerLodBias;
    float                 maxSamplerAnisotropy;
    uint32_t              maxViewports;
    uint32_t              maxViewportDimensions[2];
    float                 viewportBoundsRange[2];
    uint32_t              viewportSubPixelBits;
    size_t                minMemoryMapAlignment;
    VkDeviceSize          minTexelBufferOffsetAlignment;
    VkDeviceSize          minUniformBufferOffsetAlignment;
    VkDeviceSize          minStorageBufferOffsetAlignment;
    int32_t               minTexelOffset;
    uint32_t              maxTexelOffset;
    int32_t               minTexelGatherOffset;
    uint32_t              maxTexelGatherOffset;
    float                 minInterpolationOffset;
    float                 maxInterpolationOffset;
    uint32_t              subPixelInterpolationOffsetBits;
    uint32_t              maxFramebufferWidth;
    uint32_t              maxFramebufferHeight;
    uint32_t              maxFramebufferLayers;
    VkSampleCountFlags    framebufferColorSampleCounts;
    VkSampleCountFlags    framebufferDepthSampleCounts;
    VkSampleCountFlags    framebufferStencilSampleCounts;
    VkSampleCountFlags    framebufferNoAttachmentsSampleCounts;
```

```
    uint32_t                maxColorAttachments;
    VkSampleCountFlags      sampledImageColorSampleCounts;
    VkSampleCountFlags      sampledImageIntegerSampleCounts;
    VkSampleCountFlags      sampledImageDepthSampleCounts;
    VkSampleCountFlags      sampledImageStencilSampleCounts;
    VkSampleCountFlags      storageImageSampleCounts;
    uint32_t                maxSampleMaskWords;
    VkBool32                timestampComputeAndGraphics;
    float                   timestampPeriod;
    uint32_t                maxClipDistances;
    uint32_t                maxCullDistances;
    uint32_t                maxCombinedClipAndCullDistances;
    uint32_t                discreteQueuePriorities;
    float                   pointSizeRange[2];
    float                   lineWidthRange[2];
    float                   pointSizeGranularity;
    float                   lineWidthGranularity;
    VkBool32                strictLines;
    VkBool32                standardSampleLocations;
    VkDeviceSize            optimalBufferCopyOffsetAlignment;
    VkDeviceSize            optimalBufferCopyRowPitchAlignment;
    VkDeviceSize            nonCoherentAtomSize;
} VkPhysicalDeviceLimits;
```

The `VkPhysicalDeviceLimits` are properties of the physical device. These are available in the `limits` member of the VkPhysicalDeviceProperties structure which is returned from vkGetPhysicalDeviceProperties.

- `maxImageDimension1D` is the largest dimension (`width`) that is guaranteed to be supported for all images created with an `imageType` of `VK_IMAGE_TYPE_1D`. Some combinations of image parameters (format, usage, etc.) **may** allow support for larger dimensions, which **can** be queried using vkGetPhysicalDeviceImageFormatProperties.
- `maxImageDimension2D` is the largest dimension (`width` or `height`) that is guaranteed to be supported for all images created with an `imageType` of `VK_IMAGE_TYPE_2D` and without `VK_IMAGE_CREATE_CUBE_COMPATIBLE_BIT` set in `flags`. Some combinations of image parameters (format, usage, etc.) **may** allow support for larger dimensions, which **can** be queried using vkGetPhysicalDeviceImageFormatProperties.
- `maxImageDimension3D` is the largest dimension (`width`, `height`, or `depth`) that is guaranteed to be supported for all images created with an `imageType` of `VK_IMAGE_TYPE_3D`. Some combinations of image parameters (format, usage, etc.) **may** allow support for larger dimensions, which **can** be queried using vkGetPhysicalDeviceImageFormatProperties.
- `maxImageDimensionCube` is the largest dimension (`width` or `height`) that is guaranteed to be supported for all images created with an `imageType` of `VK_IMAGE_TYPE_2D` and with `VK_IMAGE_CREATE_CUBE_COMPATIBLE_BIT` set in `flags`. Some combinations of image parameters (format, usage, etc.) **may** allow support for larger dimensions, which **can** be queried using vkGetPhysicalDeviceImageFormatProperties.
- `maxImageArrayLayers` is the maximum number of layers (`arrayLayers`) for an image.

- maxTexelBufferElements is the maximum number of addressable texels for a buffer view created on a buffer which was created with the VK_BUFFER_USAGE_UNIFORM_TEXEL_BUFFER_BIT or VK_BUFFER_USAGE_STORAGE_TEXEL_BUFFER_BIT set in the usage member of the VkBufferCreateInfo structure.
- maxUniformBufferRange is the maximum value that **can** be specified in the range member of a VkDescriptorBufferInfo structure passed to vkUpdateDescriptorSets for descriptors of type VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER or VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC.
- maxStorageBufferRange is the maximum value that **can** be specified in the range member of a VkDescriptorBufferInfo structure passed to vkUpdateDescriptorSets for descriptors of type VK_DESCRIPTOR_TYPE_STORAGE_BUFFER or VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC.
- maxPushConstantsSize is the maximum size, in bytes, of the pool of push constant memory. For each of the push constant ranges indicated by the pPushConstantRanges member of the VkPipelineLayoutCreateInfo structure, (offset + size) **must** be less than or equal to this limit.
- maxMemoryAllocationCount is the maximum number of device memory allocations, as created by vkAllocateMemory, which **can** simultaneously exist.
- maxSamplerAllocationCount is the maximum number of sampler objects, as created by vkCreateSampler, which **can** simultaneously exist on a device.
- bufferImageGranularity is the granularity, in bytes, at which buffer or linear image resources, and optimal image resources **can** be bound to adjacent offsets in the same VkDeviceMemory object without aliasing. See Buffer-Image Granularity for more details.
- sparseAddressSpaceSize is the total amount of address space available, in bytes, for sparse memory resources. This is an upper bound on the sum of the sizes of all sparse resources, regardless of whether any memory is bound to them. If the extendedSparseAddressSpace feature is enabled, then the difference between extendedSparseAddressSpaceSize and sparseAddressSpaceSize can also be used, by VkImage created with the usage member of VkImageCreateInfo only containing bits in extendedSparseImageUsageFlags and VkBuffer created with the usage member of VkBufferCreateInfo only containing bits in extendedSparseBufferUsageFlags.
- maxBoundDescriptorSets is the maximum number of descriptor sets that **can** be simultaneously used by a pipeline. All DescriptorSet decorations in shader modules **must** have a value less than maxBoundDescriptorSets. See Descriptor Sets.
- maxPerStageDescriptorSamplers is the maximum number of samplers that **can** be accessible to a single shader stage in a pipeline layout. Descriptors with a type of VK_DESCRIPTOR_TYPE_SAMPLER or VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER count against this limit. Only descriptors in descriptor set layouts created without the VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT bit set count against this limit. A descriptor is accessible to a shader stage when the stageFlags member of the VkDescriptorSetLayoutBinding structure has the bit for that shader stage set. See Sampler and Combined Image Sampler.
- maxPerStageDescriptorUniformBuffers is the maximum number of uniform buffers that **can** be accessible to a single shader stage in a pipeline layout. Descriptors with a type of VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER or VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC count against this limit. Only descriptors in descriptor set layouts created without the VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT bit set count against this limit. A

descriptor is accessible to a shader stage when the stageFlags member of the VkDescriptorSetLayoutBinding structure has the bit for that shader stage set. See Uniform Buffer and Dynamic Uniform Buffer.

- maxPerStageDescriptorStorageBuffers is the maximum number of storage buffers that **can** be accessible to a single shader stage in a pipeline layout. Descriptors with a type of VK_DESCRIPTOR_TYPE_STORAGE_BUFFER or VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC count against this limit. Only descriptors in descriptor set layouts created without the VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT bit set count against this limit. A descriptor is accessible to a pipeline shader stage when the stageFlags member of the VkDescriptorSetLayoutBinding structure has the bit for that shader stage set. See Storage Buffer and Dynamic Storage Buffer.
- maxPerStageDescriptorSampledImages is the maximum number of sampled images that **can** be accessible to a single shader stage in a pipeline layout. Descriptors with a type of VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER, VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE, or VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER count against this limit. Only descriptors in descriptor set layouts created without the VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT bit set count against this limit. A descriptor is accessible to a pipeline shader stage when the stageFlags member of the VkDescriptorSetLayoutBinding structure has the bit for that shader stage set. See Combined Image Sampler, Sampled Image, and Uniform Texel Buffer.
- maxPerStageDescriptorStorageImages is the maximum number of storage images that **can** be accessible to a single shader stage in a pipeline layout. Descriptors with a type of VK_DESCRIPTOR_TYPE_STORAGE_IMAGE, or VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER count against this limit. Only descriptors in descriptor set layouts created without the VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT bit set count against this limit. A descriptor is accessible to a pipeline shader stage when the stageFlags member of the VkDescriptorSetLayoutBinding structure has the bit for that shader stage set. See Storage Image, and Storage Texel Buffer.
- maxPerStageDescriptorInputAttachments is the maximum number of input attachments that **can** be accessible to a single shader stage in a pipeline layout. Descriptors with a type of VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT count against this limit. Only descriptors in descriptor set layouts created without the VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT bit set count against this limit. A descriptor is accessible to a pipeline shader stage when the stageFlags member of the VkDescriptorSetLayoutBinding structure has the bit for that shader stage set. These are only supported for the fragment stage. See Input Attachment.
- maxPerStageResources is the maximum number of resources that **can** be accessible to a single shader stage in a pipeline layout. Descriptors with a type of VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER, VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE, VK_DESCRIPTOR_TYPE_STORAGE_IMAGE, VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER, VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER, VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER, VK_DESCRIPTOR_TYPE_STORAGE_BUFFER, VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC, VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC, or VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT count against this limit. Only descriptors in descriptor set layouts created without the VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT bit set count against this limit. For the fragment shader stage the framebuffer color attachments also count against this limit.
- maxDescriptorSetSamplers is the maximum number of samplers that **can** be included in a

pipeline layout. Descriptors with a type of VK_DESCRIPTOR_TYPE_SAMPLER or VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER count against this limit. Only descriptors in descriptor set layouts created without the VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT bit set count against this limit. See Sampler and Combined Image Sampler.

- maxDescriptorSetUniformBuffers is the maximum number of uniform buffers that **can** be included in a pipeline layout. Descriptors with a type of VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER or VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC count against this limit. Only descriptors in descriptor set layouts created without the VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT bit set count against this limit. See Uniform Buffer and Dynamic Uniform Buffer.
- maxDescriptorSetUniformBuffersDynamic is the maximum number of dynamic uniform buffers that **can** be included in a pipeline layout. Descriptors with a type of VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC count against this limit. Only descriptors in descriptor set layouts created without the VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT bit set count against this limit. See Dynamic Uniform Buffer.
- maxDescriptorSetStorageBuffers is the maximum number of storage buffers that **can** be included in a pipeline layout. Descriptors with a type of VK_DESCRIPTOR_TYPE_STORAGE_BUFFER or VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC count against this limit. Only descriptors in descriptor set layouts created without the VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT bit set count against this limit. See Storage Buffer and Dynamic Storage Buffer.
- maxDescriptorSetStorageBuffersDynamic is the maximum number of dynamic storage buffers that **can** be included in a pipeline layout. Descriptors with a type of VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC count against this limit. Only descriptors in descriptor set layouts created without the VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT bit set count against this limit. See Dynamic Storage Buffer.
- maxDescriptorSetSampledImages is the maximum number of sampled images that **can** be included in a pipeline layout. Descriptors with a type of VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER, VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE, or VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER count against this limit. Only descriptors in descriptor set layouts created without the VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT bit set count against this limit. See Combined Image Sampler, Sampled Image, and Uniform Texel Buffer.
- maxDescriptorSetStorageImages is the maximum number of storage images that **can** be included in a pipeline layout. Descriptors with a type of VK_DESCRIPTOR_TYPE_STORAGE_IMAGE, or VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER count against this limit. Only descriptors in descriptor set layouts created without the VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT bit set count against this limit. See Storage Image, and Storage Texel Buffer.
- maxDescriptorSetInputAttachments is the maximum number of input attachments that **can** be included in a pipeline layout. Descriptors with a type of VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT count against this limit. Only descriptors in descriptor set layouts created without the VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT bit set count against this limit. See Input Attachment.

- `maxVertexInputAttributes` is the maximum number of vertex input attributes that **can** be specified for a graphics pipeline. These are described in the array of `VkVertexInputAttributeDescription` structures that are provided at graphics pipeline creation time via the `pVertexAttributeDescriptions` member of the VkPipelineVertexInputStateCreateInfo structure. See Vertex Attributes and Vertex Input Description.
- `maxVertexInputBindings` is the maximum number of vertex buffers that **can** be specified for providing vertex attributes to a graphics pipeline. These are described in the array of `VkVertexInputBindingDescription` structures that are provided at graphics pipeline creation time via the `pVertexBindingDescriptions` member of the VkPipelineVertexInputStateCreateInfo structure. The `binding` member of `VkVertexInputBindingDescription` **must** be less than this limit. See Vertex Input Description.
- `maxVertexInputAttributeOffset` is the maximum vertex input attribute offset that **can** be added to the vertex input binding stride. The `offset` member of the `VkVertexInputAttributeDescription` structure **must** be less than or equal to this limit. See Vertex Input Description.
- `maxVertexInputBindingStride` is the maximum vertex input binding stride that **can** be specified in a vertex input binding. The `stride` member of the `VkVertexInputBindingDescription` structure **must** be less than or equal to this limit. See Vertex Input Description.
- `maxVertexOutputComponents` is the maximum number of components of output variables which **can** be output by a vertex shader. See Vertex Shaders.
- `maxTessellationGenerationLevel` is the maximum tessellation generation level supported by the fixed-function tessellation primitive generator. See Tessellation.
- `maxTessellationPatchSize` is the maximum patch size, in vertices, of patches that **can** be processed by the tessellation control shader and tessellation primitive generator. The `patchControlPoints` member of the VkPipelineTessellationStateCreateInfo structure specified at pipeline creation time and the value provided in the `OutputVertices` execution mode of shader modules **must** be less than or equal to this limit. See Tessellation.
- `maxTessellationControlPerVertexInputComponents` is the maximum number of components of input variables which **can** be provided as per-vertex inputs to the tessellation control shader stage.
- `maxTessellationControlPerVertexOutputComponents` is the maximum number of components of per-vertex output variables which **can** be output from the tessellation control shader stage.
- `maxTessellationControlPerPatchOutputComponents` is the maximum number of components of per-patch output variables which **can** be output from the tessellation control shader stage.
- `maxTessellationControlTotalOutputComponents` is the maximum total number of components of per-vertex and per-patch output variables which **can** be output from the tessellation control shader stage.
- `maxTessellationEvaluationInputComponents` is the maximum number of components of input variables which **can** be provided as per-vertex inputs to the tessellation evaluation shader stage.
- `maxTessellationEvaluationOutputComponents` is the maximum number of components of per-vertex output variables which **can** be output from the tessellation evaluation shader stage.
- `maxGeometryShaderInvocations` is the maximum invocation count supported for instanced geometry shaders. The value provided in the `Invocations` execution mode of shader modules

**must** be less than or equal to this limit. See Geometry Shading.

- `maxGeometryInputComponents` is the maximum number of components of input variables which **can** be provided as inputs to the geometry shader stage.
- `maxGeometryOutputComponents` is the maximum number of components of output variables which **can** be output from the geometry shader stage.
- `maxGeometryOutputVertices` is the maximum number of vertices which **can** be emitted by any geometry shader.
- `maxGeometryTotalOutputComponents` is the maximum total number of components of output variables, across all emitted vertices, which **can** be output from the geometry shader stage.
- `maxFragmentInputComponents` is the maximum number of components of input variables which **can** be provided as inputs to the fragment shader stage.
- `maxFragmentOutputAttachments` is the maximum number of output attachments which **can** be written to by the fragment shader stage.
- `maxFragmentDualSrcAttachments` is the maximum number of output attachments which **can** be written to by the fragment shader stage when blending is enabled and one of the dual source blend modes is in use. See Dual-Source Blending and `dualSrcBlend`.
- `maxFragmentCombinedOutputResources` is the total number of storage buffers, storage images, and output `Location` decorated color attachments (described in Fragment Output Interface) which **can** be used in the fragment shader stage.
- `maxComputeSharedMemorySize` is the maximum total storage size, in bytes, available for variables declared with the `Workgroup` storage class in shader modules (or with the `shared` storage qualifier in GLSL) in the compute shader stage.
- `maxComputeWorkGroupCount`[3] is the maximum number of local workgroups that **can** be dispatched by a single dispatching command. These three values represent the maximum number of local workgroups for the X, Y, and Z dimensions, respectively. The workgroup count parameters to the dispatching commands **must** be less than or equal to the corresponding limit. See Dispatching Commands.
- `maxComputeWorkGroupInvocations` is the maximum total number of compute shader invocations in a single local workgroup. The product of the X, Y, and Z sizes, as specified by the `LocalSize` or `LocalSizeId` execution mode in shader modules or by the object decorated by the `WorkgroupSize` decoration, **must** be less than or equal to this limit.
- `maxComputeWorkGroupSize`[3] is the maximum size of a local compute workgroup, per dimension. These three values represent the maximum local workgroup size in the X, Y, and Z dimensions, respectively. The `x`, `y`, and `z` sizes, as specified by the `LocalSize` or `LocalSizeId` execution mode or by the object decorated by the `WorkgroupSize` decoration in shader modules, **must** be less than or equal to the corresponding limit.
- `subPixelPrecisionBits` is the number of bits of subpixel precision in framebuffer coordinates $x_f$ and $y_f$. See Rasterization.
- `subTexelPrecisionBits` is the number of bits of precision in the division along an axis of an image used for minification and magnification filters. $2^{\texttt{subTexelPrecisionBits}}$ is the actual number of divisions along each axis of the image represented. Sub-texel values calculated during image sampling will snap to these locations when generating the filtered results.

- mipmapPrecisionBits is the number of bits of division that the LOD calculation for mipmap fetching get snapped to when determining the contribution from each mip level to the mip filtered results. $2^{mipmapPrecisionBits}$ is the actual number of divisions.
- maxDrawIndexedIndexValue is the maximum index value that **can** be used for indexed draw calls when using 32-bit indices. This excludes the primitive restart index value of 0xFFFFFFFF. See fullDrawIndexUint32.
- maxDrawIndirectCount is the maximum draw count that is supported for indirect drawing calls. See multiDrawIndirect.
- maxSamplerLodBias is the maximum absolute sampler LOD bias. The sum of the mipLodBias member of the VkSamplerCreateInfo structure and the Bias operand of image sampling operations in shader modules (or 0 if no Bias operand is provided to an image sampling operation) are clamped to the range [-maxSamplerLodBias,+maxSamplerLodBias]. See [samplers-mipLodBias].
- maxSamplerAnisotropy is the maximum degree of sampler anisotropy. The maximum degree of anisotropic filtering used for an image sampling operation is the minimum of the maxAnisotropy member of the VkSamplerCreateInfo structure and this limit. See [samplers-maxAnisotropy].
- maxViewports is the maximum number of active viewports. The viewportCount member of the VkPipelineViewportStateCreateInfo structure that is provided at pipeline creation **must** be less than or equal to this limit.
- maxViewportDimensions[2] are the maximum viewport dimensions in the X (width) and Y (height) dimensions, respectively. The maximum viewport dimensions **must** be greater than or equal to the largest image which **can** be created and used as a framebuffer attachment. See Controlling the Viewport.
- viewportBoundsRange[2] is the [minimum, maximum] range that the corners of a viewport **must** be contained in. This range **must** be at least [-2 × size, 2 × size - 1], where size = max(maxViewportDimensions[0], maxViewportDimensions[1]). See Controlling the Viewport.

> **NOTE** The intent of the viewportBoundsRange limit is to allow a maximum sized viewport to be arbitrarily shifted relative to the output target as long as at least some portion intersects. This would give a bounds limit of [-size + 1, 2 × size - 1] which would allow all possible non-empty-set intersections of the output target and the viewport. Since these numbers are typically powers of two, picking the signed number range using the smallest possible number of bits ends up with the specified range.

- viewportSubPixelBits is the number of bits of subpixel precision for viewport bounds. The subpixel precision that floating-point viewport bounds are interpreted at is given by this limit.
- minMemoryMapAlignment is the minimum **required** alignment, in bytes, of host visible memory allocations within the host address space. When mapping a memory allocation with vkMapMemory, subtracting offset bytes from the returned pointer will always produce an integer multiple of this limit. See Host Access to Device Memory Objects. The value **must** be a power of two.
- minTexelBufferOffsetAlignment is the minimum **required** alignment, in bytes, for the offset member of the VkBufferViewCreateInfo structure for texel buffers. The value **must** be a power

of two. If `texelBufferAlignment` is enabled, this limit is equivalent to the maximum of the `uniformTexelBufferOffsetAlignmentBytes` and `storageTexelBufferOffsetAlignmentBytes` members of VkPhysicalDeviceTexelBufferAlignmentProperties, but smaller alignment is **optionally** allowed by `storageTexelBufferOffsetSingleTexelAlignment` and `uniformTexelBufferOffsetSingleTexelAlignment`. If `texelBufferAlignment` is not enabled, VkBufferViewCreateInfo::`offset` **must** be a multiple of this value.

- `minUniformBufferOffsetAlignment` is the minimum **required** alignment, in bytes, for the `offset` member of the `VkDescriptorBufferInfo` structure for uniform buffers. When a descriptor of type `VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER` or `VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC` is updated, the `offset` **must** be an integer multiple of this limit. Similarly, dynamic offsets for uniform buffers **must** be multiples of this limit. The value **must** be a power of two.
- `minStorageBufferOffsetAlignment` is the minimum **required** alignment, in bytes, for the `offset` member of the `VkDescriptorBufferInfo` structure for storage buffers. When a descriptor of type `VK_DESCRIPTOR_TYPE_STORAGE_BUFFER` or `VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC` is updated, the `offset` **must** be an integer multiple of this limit. Similarly, dynamic offsets for storage buffers **must** be multiples of this limit. The value **must** be a power of two.
- `minTexelOffset` is the minimum offset value for the `ConstOffset` image operand of any of the `OpImageSample*` or `OpImageFetch*` image instructions.
- `maxTexelOffset` is the maximum offset value for the `ConstOffset` image operand of any of the `OpImageSample*` or `OpImageFetch*` image instructions.
- `minTexelGatherOffset` is the minimum offset value for the `Offset`, `ConstOffset`, or `ConstOffsets` image operands of any of the `OpImage*Gather` image instructions.
- `maxTexelGatherOffset` is the maximum offset value for the `Offset`, `ConstOffset`, or `ConstOffsets` image operands of any of the `OpImage*Gather` image instructions.
- `minInterpolationOffset` is the base minimum (inclusive) negative offset value for the `Offset` operand of the `InterpolateAtOffset` extended instruction.
- `maxInterpolationOffset` is the base maximum (inclusive) positive offset value for the `Offset` operand of the `InterpolateAtOffset` extended instruction.
- `subPixelInterpolationOffsetBits` is the number of fractional bits that the `x` and `y` offsets to the `InterpolateAtOffset` extended instruction **may** be rounded to as fixed-point values.
- `maxFramebufferWidth` is the maximum width for a framebuffer. The `width` member of the VkFramebufferCreateInfo structure **must** be less than or equal to this limit.
- `maxFramebufferHeight` is the maximum height for a framebuffer. The `height` member of the VkFramebufferCreateInfo structure **must** be less than or equal to this limit.
- `maxFramebufferLayers` is the maximum layer count for a layered framebuffer. The `layers` member of the VkFramebufferCreateInfo structure **must** be less than or equal to this limit.
- `framebufferColorSampleCounts` is a bitmask[1] of VkSampleCountFlagBits indicating the color sample counts that are supported for all framebuffer color attachments with floating- or fixed-point formats. There is no limit specifying the color sample counts that are supported for all color attachments with integer formats.
- `framebufferDepthSampleCounts` is a bitmask[1] of VkSampleCountFlagBits indicating the supported depth sample counts for all framebuffer depth/stencil attachments, when the format includes a

depth component.

- `framebufferStencilSampleCounts` is a bitmask[1] of VkSampleCountFlagBits indicating the supported stencil sample counts for all framebuffer depth/stencil attachments, when the format includes a stencil component.
- `framebufferNoAttachmentsSampleCounts` is a bitmask[1] of VkSampleCountFlagBits indicating the supported sample counts for a subpass which uses no attachments.
- `maxColorAttachments` is the maximum number of color attachments that **can** be used by a subpass in a render pass. The `colorAttachmentCount` member of the `VkSubpassDescription` or `VkSubpassDescription2` structure **must** be less than or equal to this limit.
- `sampledImageColorSampleCounts` is a bitmask[1] of VkSampleCountFlagBits indicating the sample counts supported for all 2D images created with `VK_IMAGE_TILING_OPTIMAL`, `usage` containing `VK_IMAGE_USAGE_SAMPLED_BIT`, and a non-integer color format.
- `sampledImageIntegerSampleCounts` is a bitmask[1] of VkSampleCountFlagBits indicating the sample counts supported for all 2D images created with `VK_IMAGE_TILING_OPTIMAL`, `usage` containing `VK_IMAGE_USAGE_SAMPLED_BIT`, and an integer color format.
- `sampledImageDepthSampleCounts` is a bitmask[1] of VkSampleCountFlagBits indicating the sample counts supported for all 2D images created with `VK_IMAGE_TILING_OPTIMAL`, `usage` containing `VK_IMAGE_USAGE_SAMPLED_BIT`, and a depth format.
- `sampledImageStencilSampleCounts` is a bitmask[1] of VkSampleCountFlagBits indicating the sample counts supported for all 2D images created with `VK_IMAGE_TILING_OPTIMAL`, `usage` containing `VK_IMAGE_USAGE_SAMPLED_BIT`, and a stencil format.
- `storageImageSampleCounts` is a bitmask[1] of VkSampleCountFlagBits indicating the sample counts supported for all 2D images created with `VK_IMAGE_TILING_OPTIMAL`, and `usage` containing `VK_IMAGE_USAGE_STORAGE_BIT`.
- `maxSampleMaskWords` is the maximum number of array elements of a variable decorated with the `SampleMask` built-in decoration.
- `timestampComputeAndGraphics` specifies support for timestamps on all graphics and compute queues. If this limit is set to `VK_TRUE`, all queues that advertise the `VK_QUEUE_GRAPHICS_BIT` or `VK_QUEUE_COMPUTE_BIT` in the `VkQueueFamilyProperties::queueFlags` support `VkQueueFamilyProperties::timestampValidBits` of at least 36. See Timestamp Queries.
- `timestampPeriod` is the number of nanoseconds **required** for a timestamp query to be incremented by 1. See Timestamp Queries.
- `maxClipDistances` is the maximum number of clip distances that **can** be used in a single shader stage. The size of any array declared with the `ClipDistance` built-in decoration in a shader module **must** be less than or equal to this limit.
- `maxCullDistances` is the maximum number of cull distances that **can** be used in a single shader stage. The size of any array declared with the `CullDistance` built-in decoration in a shader module **must** be less than or equal to this limit.
- `maxCombinedClipAndCullDistances` is the maximum combined number of clip and cull distances that **can** be used in a single shader stage. The sum of the sizes of all arrays declared with the `ClipDistance` and `CullDistance` built-in decoration used by a single shader stage in a shader module **must** be less than or equal to this limit.

- `discreteQueuePriorities` is the number of discrete priorities that **can** be assigned to a queue based on the value of each member of VkDeviceQueueCreateInfo::`pQueuePriorities`. This **must** be at least 2, and levels **must** be spread evenly over the range, with at least one level at 1.0, and another at 0.0. See Queue Priority.
- `pointSizeRange`[2] is the range [`minimum`,`maximum`] of supported sizes for points. Values written to variables decorated with the `PointSize` built-in decoration are clamped to this range.
- `lineWidthRange`[2] is the range [`minimum`,`maximum`] of supported widths for lines. Values specified by the `lineWidth` member of the VkPipelineRasterizationStateCreateInfo or the `lineWidth` parameter to `vkCmdSetLineWidth` are clamped to this range.
- `pointSizeGranularity` is the granularity of supported point sizes. Not all point sizes in the range defined by `pointSizeRange` are supported. This limit specifies the granularity (or increment) between successive supported point sizes.
- `lineWidthGranularity` is the granularity of supported line widths. Not all line widths in the range defined by `lineWidthRange` are supported. This limit specifies the granularity (or increment) between successive supported line widths.
- `strictLines` specifies whether lines are rasterized according to the preferred method of rasterization. If set to `VK_FALSE`, lines **may** be rasterized under a relaxed set of rules. If set to `VK_TRUE`, lines are rasterized as per the strict definition. See Basic Line Segment Rasterization.
- `standardSampleLocations` specifies whether rasterization uses the standard sample locations as documented in Multisampling. If set to `VK_TRUE`, the implementation uses the documented sample locations. If set to `VK_FALSE`, the implementation **may** use different sample locations.
- `optimalBufferCopyOffsetAlignment` is the optimal buffer offset alignment in bytes for vkCmdCopyBufferToImage2KHR, vkCmdCopyBufferToImage, vkCmdCopyImageToBuffer2KHR, and vkCmdCopyImageToBuffer. This value is also the optimal host memory offset alignment in bytes for vkCopyMemoryToImageEXT and vkCopyImageToMemoryEXT. The per texel alignment requirements are enforced, but applications **should** use the optimal alignment for optimal performance and power use. The value **must** be a power of two.
- `optimalBufferCopyRowPitchAlignment` is the optimal buffer row pitch alignment in bytes for vkCmdCopyBufferToImage2KHR, vkCmdCopyBufferToImage, vkCmdCopyImageToBuffer2KHR, and vkCmdCopyImageToBuffer. This value is also the optimal host memory row pitch alignment in bytes for vkCopyMemoryToImageEXT and vkCopyImageToMemoryEXT. Row pitch is the number of bytes between texels with the same X coordinate in adjacent rows (Y coordinates differ by one). The per texel alignment requirements are enforced, but applications **should** use the optimal alignment for optimal performance and power use. The value **must** be a power of two.
- `nonCoherentAtomSize` is the size and alignment in bytes that bounds concurrent access to host-mapped device memory. The value **must** be a power of two.

**1**

For all bitmasks of VkSampleCountFlagBits, the sample count limits defined above represent the minimum supported sample counts for each image type. Individual images **may** support additional sample counts, which are queried using vkGetPhysicalDeviceImageFormatProperties as described in Supported Sample Counts.

Bits which **may** be set in the sample count limits returned by VkPhysicalDeviceLimits, as well as in other queries and structures representing image sample counts, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkSampleCountFlagBits {
    VK_SAMPLE_COUNT_1_BIT = 0x00000001,
    VK_SAMPLE_COUNT_2_BIT = 0x00000002,
    VK_SAMPLE_COUNT_4_BIT = 0x00000004,
    VK_SAMPLE_COUNT_8_BIT = 0x00000008,
    VK_SAMPLE_COUNT_16_BIT = 0x00000010,
    VK_SAMPLE_COUNT_32_BIT = 0x00000020,
    VK_SAMPLE_COUNT_64_BIT = 0x00000040,
} VkSampleCountFlagBits;
```

- VK_SAMPLE_COUNT_1_BIT specifies an image with one sample per pixel.
- VK_SAMPLE_COUNT_2_BIT specifies an image with 2 samples per pixel.
- VK_SAMPLE_COUNT_4_BIT specifies an image with 4 samples per pixel.
- VK_SAMPLE_COUNT_8_BIT specifies an image with 8 samples per pixel.
- VK_SAMPLE_COUNT_16_BIT specifies an image with 16 samples per pixel.
- VK_SAMPLE_COUNT_32_BIT specifies an image with 32 samples per pixel.
- VK_SAMPLE_COUNT_64_BIT specifies an image with 64 samples per pixel.

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkSampleCountFlags;
```

VkSampleCountFlags is a bitmask type for setting a mask of zero or more VkSampleCountFlagBits.

The VkPhysicalDevicePushDescriptorPropertiesKHR structure is defined as:

```
// Provided by VK_KHR_push_descriptor
typedef struct VkPhysicalDevicePushDescriptorPropertiesKHR {
    VkStructureType    sType;
    void*              pNext;
    uint32_t           maxPushDescriptors;
} VkPhysicalDevicePushDescriptorPropertiesKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- maxPushDescriptors is the maximum number of descriptors that **can** be used in a descriptor set layout created with VK_DESCRIPTOR_SET_LAYOUT_CREATE_PUSH_DESCRIPTOR_BIT_KHR set.

If the VkPhysicalDevicePushDescriptorPropertiesKHR structure is included in the pNext chain of the VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with

each corresponding implementation-dependent property.

### Valid Usage (Implicit)

- VUID-VkPhysicalDevicePushDescriptorPropertiesKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PUSH_DESCRIPTOR_PROPERTIES_KHR`

The `VkPhysicalDeviceMultiviewProperties` structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkPhysicalDeviceMultiviewProperties {
    VkStructureType    sType;
    void*              pNext;
    uint32_t           maxMultiviewViewCount;
    uint32_t           maxMultiviewInstanceIndex;
} VkPhysicalDeviceMultiviewProperties;
```

or the equivalent

```
// Provided by VK_KHR_multiview
typedef VkPhysicalDeviceMultiviewProperties VkPhysicalDeviceMultiviewPropertiesKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `maxMultiviewViewCount` is one greater than the maximum view index that **can** be used in a subpass.
- `maxMultiviewInstanceIndex` is the maximum valid value of instance index allowed to be generated by a drawing command recorded within a subpass of a multiview render pass instance.

If the `VkPhysicalDeviceMultiviewProperties` structure is included in the `pNext` chain of the VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with each corresponding implementation-dependent property.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceMultiviewProperties-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MULTIVIEW_PROPERTIES`

The `VkPhysicalDeviceFloatControlsProperties` structure is defined as:

```
typedef struct VkPhysicalDeviceFloatControlsProperties {
    VkStructureType                     sType;
    void*                               pNext;
    VkShaderFloatControlsIndependence   denormBehaviorIndependence;
    VkShaderFloatControlsIndependence   roundingModeIndependence;
    VkBool32                            shaderSignedZeroInfNanPreserveFloat16;
    VkBool32                            shaderSignedZeroInfNanPreserveFloat32;
    VkBool32                            shaderSignedZeroInfNanPreserveFloat64;
    VkBool32                            shaderDenormPreserveFloat16;
    VkBool32                            shaderDenormPreserveFloat32;
    VkBool32                            shaderDenormPreserveFloat64;
    VkBool32                            shaderDenormFlushToZeroFloat16;
    VkBool32                            shaderDenormFlushToZeroFloat32;
    VkBool32                            shaderDenormFlushToZeroFloat64;
    VkBool32                            shaderRoundingModeRTEFloat16;
    VkBool32                            shaderRoundingModeRTEFloat32;
    VkBool32                            shaderRoundingModeRTEFloat64;
    VkBool32                            shaderRoundingModeRTZFloat16;
    VkBool32                            shaderRoundingModeRTZFloat32;
    VkBool32                            shaderRoundingModeRTZFloat64;
} VkPhysicalDeviceFloatControlsProperties;
```

or the equivalent

```
// Provided by VK_KHR_shader_float_controls
typedef VkPhysicalDeviceFloatControlsProperties
VkPhysicalDeviceFloatControlsPropertiesKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `denormBehaviorIndependence` is a VkShaderFloatControlsIndependence value indicating whether, and how, denorm behavior can be set independently for different bit widths.
- `roundingModeIndependence` is a VkShaderFloatControlsIndependence value indicating whether, and how, rounding modes can be set independently for different bit widths.
- `shaderSignedZeroInfNanPreserveFloat16` is a boolean value indicating whether sign of a zero, Nans and ±∞ **can** be preserved in 16-bit floating-point computations. It also indicates whether the `SignedZeroInfNanPreserve` execution mode **can** be used for 16-bit floating-point types.
- `shaderSignedZeroInfNanPreserveFloat32` is a boolean value indicating whether sign of a zero, Nans and ±∞ **can** be preserved in 32-bit floating-point computations. It also indicates whether the `SignedZeroInfNanPreserve` execution mode **can** be used for 32-bit floating-point types.
- `shaderSignedZeroInfNanPreserveFloat64` is a boolean value indicating whether sign of a zero, Nans and ±∞ **can** be preserved in 64-bit floating-point computations. It also indicates whether the `SignedZeroInfNanPreserve` execution mode **can** be used for 64-bit floating-point types.
- `shaderDenormPreserveFloat16` is a boolean value indicating whether denormals **can** be preserved

in 16-bit floating-point computations. It also indicates whether the `DenormPreserve` execution mode **can** be used for 16-bit floating-point types.

- `shaderDenormPreserveFloat32` is a boolean value indicating whether denormals **can** be preserved in 32-bit floating-point computations. It also indicates whether the `DenormPreserve` execution mode **can** be used for 32-bit floating-point types.
- `shaderDenormPreserveFloat64` is a boolean value indicating whether denormals **can** be preserved in 64-bit floating-point computations. It also indicates whether the `DenormPreserve` execution mode **can** be used for 64-bit floating-point types.
- `shaderDenormFlushToZeroFloat16` is a boolean value indicating whether denormals **can** be flushed to zero in 16-bit floating-point computations. It also indicates whether the `DenormFlushToZero` execution mode **can** be used for 16-bit floating-point types.
- `shaderDenormFlushToZeroFloat32` is a boolean value indicating whether denormals **can** be flushed to zero in 32-bit floating-point computations. It also indicates whether the `DenormFlushToZero` execution mode **can** be used for 32-bit floating-point types.
- `shaderDenormFlushToZeroFloat64` is a boolean value indicating whether denormals **can** be flushed to zero in 64-bit floating-point computations. It also indicates whether the `DenormFlushToZero` execution mode **can** be used for 64-bit floating-point types.
- `shaderRoundingModeRTEFloat16` is a boolean value indicating whether an implementation supports the round-to-nearest-even rounding mode for 16-bit floating-point arithmetic and conversion instructions. It also indicates whether the `RoundingModeRTE` execution mode **can** be used for 16-bit floating-point types.
- `shaderRoundingModeRTEFloat32` is a boolean value indicating whether an implementation supports the round-to-nearest-even rounding mode for 32-bit floating-point arithmetic and conversion instructions. It also indicates whether the `RoundingModeRTE` execution mode **can** be used for 32-bit floating-point types.
- `shaderRoundingModeRTEFloat64` is a boolean value indicating whether an implementation supports the round-to-nearest-even rounding mode for 64-bit floating-point arithmetic and conversion instructions. It also indicates whether the `RoundingModeRTE` execution mode **can** be used for 64-bit floating-point types.
- `shaderRoundingModeRTZFloat16` is a boolean value indicating whether an implementation supports the round-towards-zero rounding mode for 16-bit floating-point arithmetic and conversion instructions. It also indicates whether the `RoundingModeRTZ` execution mode **can** be used for 16-bit floating-point types.
- `shaderRoundingModeRTZFloat32` is a boolean value indicating whether an implementation supports the round-towards-zero rounding mode for 32-bit floating-point arithmetic and conversion instructions. It also indicates whether the `RoundingModeRTZ` execution mode **can** be used for 32-bit floating-point types.
- `shaderRoundingModeRTZFloat64` is a boolean value indicating whether an implementation supports the round-towards-zero rounding mode for 64-bit floating-point arithmetic and conversion instructions. It also indicates whether the `RoundingModeRTZ` execution mode **can** be used for 64-bit floating-point types.

If the `VkPhysicalDeviceFloatControlsProperties` structure is included in the `pNext` chain of the VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with

each corresponding implementation-dependent property.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceFloatControlsProperties-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_FLOAT_CONTROLS_PROPERTIES`

Values which **may** be returned in the `denormBehaviorIndependence` and `roundingModeIndependence` fields of `VkPhysicalDeviceFloatControlsProperties` are:

```
typedef enum VkShaderFloatControlsIndependence {
    VK_SHADER_FLOAT_CONTROLS_INDEPENDENCE_32_BIT_ONLY = 0,
    VK_SHADER_FLOAT_CONTROLS_INDEPENDENCE_ALL = 1,
    VK_SHADER_FLOAT_CONTROLS_INDEPENDENCE_NONE = 2,
  // Provided by VK_KHR_shader_float_controls
    VK_SHADER_FLOAT_CONTROLS_INDEPENDENCE_32_BIT_ONLY_KHR =
VK_SHADER_FLOAT_CONTROLS_INDEPENDENCE_32_BIT_ONLY,
  // Provided by VK_KHR_shader_float_controls
    VK_SHADER_FLOAT_CONTROLS_INDEPENDENCE_ALL_KHR =
VK_SHADER_FLOAT_CONTROLS_INDEPENDENCE_ALL,
  // Provided by VK_KHR_shader_float_controls
    VK_SHADER_FLOAT_CONTROLS_INDEPENDENCE_NONE_KHR =
VK_SHADER_FLOAT_CONTROLS_INDEPENDENCE_NONE,
} VkShaderFloatControlsIndependence;
```

or the equivalent

```
// Provided by VK_KHR_shader_float_controls
typedef VkShaderFloatControlsIndependence VkShaderFloatControlsIndependenceKHR;
```

- `VK_SHADER_FLOAT_CONTROLS_INDEPENDENCE_32_BIT_ONLY` specifies that shader float controls for 32-bit floating-point **can** be set independently; other bit widths **must** be set identically to each other.
- `VK_SHADER_FLOAT_CONTROLS_INDEPENDENCE_ALL` specifies that shader float controls for all bit widths **can** be set independently.
- `VK_SHADER_FLOAT_CONTROLS_INDEPENDENCE_NONE` specifies that shader float controls for all bit widths **must** be set identically.

The `VkPhysicalDeviceDiscardRectanglePropertiesEXT` structure is defined as:

```
// Provided by VK_EXT_discard_rectangles
typedef struct VkPhysicalDeviceDiscardRectanglePropertiesEXT {
    VkStructureType    sType;
    void*              pNext;
    uint32_t           maxDiscardRectangles;
} VkPhysicalDeviceDiscardRectanglePropertiesEXT;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- maxDiscardRectangles is the maximum number of active discard rectangles that **can** be specified.

If the VkPhysicalDeviceDiscardRectanglePropertiesEXT structure is included in the pNext chain of the VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with each corresponding implementation-dependent property.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceDiscardRectanglePropertiesEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DISCARD_RECTANGLE_PROPERTIES_EXT

The VkPhysicalDeviceSampleLocationsPropertiesEXT structure is defined as:

```
// Provided by VK_EXT_sample_locations
typedef struct VkPhysicalDeviceSampleLocationsPropertiesEXT {
    VkStructureType       sType;
    void*                 pNext;
    VkSampleCountFlags    sampleLocationSampleCounts;
    VkExtent2D            maxSampleLocationGridSize;
    float                 sampleLocationCoordinateRange[2];
    uint32_t              sampleLocationSubPixelBits;
    VkBool32              variableSampleLocations;
} VkPhysicalDeviceSampleLocationsPropertiesEXT;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- sampleLocationSampleCounts is a bitmask of VkSampleCountFlagBits indicating the sample counts supporting custom sample locations.
- maxSampleLocationGridSize is the maximum size of the pixel grid in which sample locations **can** vary that is supported for all sample counts in sampleLocationSampleCounts.
- sampleLocationCoordinateRange[2] is the range of supported sample location coordinates.
- sampleLocationSubPixelBits is the number of bits of subpixel precision for sample locations.
- variableSampleLocations specifies whether the sample locations used by all pipelines that will be bound to a command buffer during a subpass **must** match. If set to VK_TRUE, the implementation supports variable sample locations in a subpass. If set to VK_FALSE, then the sample locations **must** stay constant in each subpass.

If the VkPhysicalDeviceSampleLocationsPropertiesEXT structure is included in the pNext chain of the VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with each corresponding implementation-dependent property.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceSampleLocationsPropertiesEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SAMPLE_LOCATIONS_PROPERTIES_EXT`

The `VkPhysicalDeviceLegacyVertexAttributesPropertiesEXT` structure is defined as:

```
// Provided by VK_EXT_legacy_vertex_attributes
typedef struct VkPhysicalDeviceLegacyVertexAttributesPropertiesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           nativeUnalignedPerformance;
} VkPhysicalDeviceLegacyVertexAttributesPropertiesEXT;
```

This structure describes the following features:

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `nativeUnalignedPerformance` specifies whether unaligned vertex fetches do not incur significant performance penalties as compared to aligned fetches.

If the `VkPhysicalDeviceLegacyVertexAttributesPropertiesEXT` structure is included in the `pNext` chain of the VkPhysicalDeviceFeatures2 structure passed to vkGetPhysicalDeviceFeatures2, it is filled in to indicate whether each corresponding feature is supported. `VkPhysicalDeviceLegacyVertexAttributesPropertiesEXT` **can** also be used in the `pNext` chain of VkDeviceCreateInfo to selectively enable these features.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceLegacyVertexAttributesPropertiesEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_LEGACY_VERTEX_ATTRIBUTES_PROPERTIES_EXT`

The `VkPhysicalDeviceExternalMemoryHostPropertiesEXT` structure is defined as:

```
// Provided by VK_EXT_external_memory_host
typedef struct VkPhysicalDeviceExternalMemoryHostPropertiesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkDeviceSize       minImportedHostPointerAlignment;
} VkPhysicalDeviceExternalMemoryHostPropertiesEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.

- minImportedHostPointerAlignment is the minimum **required** alignment, in bytes, for the base address and size of host pointers that **can** be imported to a Vulkan memory object. The value **must** be a power of two.

If the VkPhysicalDeviceExternalMemoryHostPropertiesEXT structure is included in the pNext chain of the VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with each corresponding implementation-dependent property.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceExternalMemoryHostPropertiesEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTERNAL_MEMORY_HOST_PROPERTIES_EXT

The VkPhysicalDeviceMultiviewPerViewAttributesPropertiesNVX structure is defined as:

```
// Provided by VK_NVX_multiview_per_view_attributes
typedef struct VkPhysicalDeviceMultiviewPerViewAttributesPropertiesNVX {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           perViewPositionAllComponents;
} VkPhysicalDeviceMultiviewPerViewAttributesPropertiesNVX;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- perViewPositionAllComponents is VK_TRUE if the implementation supports per-view position values that differ in components other than the X component.

If the VkPhysicalDeviceMultiviewPerViewAttributesPropertiesNVX structure is included in the pNext chain of the VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with each corresponding implementation-dependent property.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceMultiviewPerViewAttributesPropertiesNVX-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MULTIVIEW_PER_VIEW_ATTRIBUTES_PROPERTIES_NVX

The VkPhysicalDevicePointClippingProperties structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkPhysicalDevicePointClippingProperties {
    VkStructureType            sType;
    void*                      pNext;
    VkPointClippingBehavior    pointClippingBehavior;
} VkPhysicalDevicePointClippingProperties;
```

or the equivalent

```
// Provided by VK_KHR_maintenance2
typedef VkPhysicalDevicePointClippingProperties
VkPhysicalDevicePointClippingPropertiesKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `pointClippingBehavior` is a VkPointClippingBehavior value specifying the point clipping behavior supported by the implementation.

If the `VkPhysicalDevicePointClippingProperties` structure is included in the `pNext` chain of the VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with each corresponding implementation-dependent property.

### Valid Usage (Implicit)

- VUID-VkPhysicalDevicePointClippingProperties-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_POINT_CLIPPING_PROPERTIES`

The `VkPhysicalDeviceSubgroupProperties` structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkPhysicalDeviceSubgroupProperties {
    VkStructureType           sType;
    void*                     pNext;
    uint32_t                  subgroupSize;
    VkShaderStageFlags        supportedStages;
    VkSubgroupFeatureFlags    supportedOperations;
    VkBool32                  quadOperationsInAllStages;
} VkPhysicalDeviceSubgroupProperties;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `subgroupSize` is the default number of invocations in each subgroup. `subgroupSize` is at least 1 if any of the physical device's queues support `VK_QUEUE_GRAPHICS_BIT` or `VK_QUEUE_COMPUTE_BIT`.

subgroupSize is a power-of-two.

- supportedStages is a bitfield of VkShaderStageFlagBits describing the shader stages that group operations with subgroup scope are supported in. supportedStages will have the VK_SHADER_STAGE_COMPUTE_BIT bit set if any of the physical device's queues support VK_QUEUE_COMPUTE_BIT.
- supportedOperations is a bitmask of VkSubgroupFeatureFlagBits specifying the sets of group operations with subgroup scope supported on this device. supportedOperations will have the VK_SUBGROUP_FEATURE_BASIC_BIT bit set if any of the physical device's queues support VK_QUEUE_GRAPHICS_BIT or VK_QUEUE_COMPUTE_BIT.
- quadOperationsInAllStages is a boolean specifying whether quad group operations are available in all stages, or are restricted to fragment and compute stages.

If the VkPhysicalDeviceSubgroupProperties structure is included in the pNext chain of the VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with each corresponding implementation-dependent property.

If supportedOperations includes VK_SUBGROUP_FEATURE_QUAD_BIT, or shaderSubgroupUniformControlFlow is enabled, subgroupSize **must** be greater than or equal to 4.

If the shaderQuadControl feature is supported, supportedOperations **must** include VK_SUBGROUP_FEATURE_QUAD_BIT.

If VK_KHR_shader_subgroup_rotate is supported, and the implementation advertises support with a VkExtensionProperties::specVersion greater than or equal to 2, and shaderSubgroupRotate is supported, VK_SUBGROUP_FEATURE_ROTATE_BIT_KHR **must** be returned in supportedOperations. If VK_KHR_shader_subgroup_rotate is supported, and the implementation advertises support with a VkExtensionProperties::specVersion greater than or equal to 2, and shaderSubgroupRotateClustered is supported, VK_SUBGROUP_FEATURE_ROTATE_CLUSTERED_BIT_KHR **must** be returned in supportedOperations.

**NOTE** VK_SUBGROUP_FEATURE_ROTATE_BIT_KHR and VK_SUBGROUP_FEATURE_ROTATE_CLUSTERED_BIT_KHR were added in version 2 of the VK_KHR_shader_subgroup_rotate extension, after the initial release, so there are implementations that do not advertise these bits. Applications should use the shaderSubgroupRotate and shaderSubgroupRotateClustered features to determine and enable support. These bits are advertised here for consistency and for future dependencies.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceSubgroupProperties-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SUBGROUP_PROPERTIES

Bits which **can** be set in VkPhysicalDeviceSubgroupProperties::supportedOperations to specify supported group operations with subgroup scope are:

```
// Provided by VK_VERSION_1_1
typedef enum VkSubgroupFeatureFlagBits {
    VK_SUBGROUP_FEATURE_BASIC_BIT = 0x00000001,
    VK_SUBGROUP_FEATURE_VOTE_BIT = 0x00000002,
    VK_SUBGROUP_FEATURE_ARITHMETIC_BIT = 0x00000004,
    VK_SUBGROUP_FEATURE_BALLOT_BIT = 0x00000008,
    VK_SUBGROUP_FEATURE_SHUFFLE_BIT = 0x00000010,
    VK_SUBGROUP_FEATURE_SHUFFLE_RELATIVE_BIT = 0x00000020,
    VK_SUBGROUP_FEATURE_CLUSTERED_BIT = 0x00000040,
    VK_SUBGROUP_FEATURE_QUAD_BIT = 0x00000080,
  // Provided by VK_NV_shader_subgroup_partitioned
    VK_SUBGROUP_FEATURE_PARTITIONED_BIT_NV = 0x00000100,
  // Provided by VK_KHR_shader_subgroup_rotate
    VK_SUBGROUP_FEATURE_ROTATE_BIT_KHR = 0x00000200,
  // Provided by VK_KHR_shader_subgroup_rotate
    VK_SUBGROUP_FEATURE_ROTATE_CLUSTERED_BIT_KHR = 0x00000400,
} VkSubgroupFeatureFlagBits;
```

- `VK_SUBGROUP_FEATURE_BASIC_BIT` specifies the device will accept SPIR-V shader modules containing the `GroupNonUniform` capability.
- `VK_SUBGROUP_FEATURE_VOTE_BIT` specifies the device will accept SPIR-V shader modules containing the `GroupNonUniformVote` capability.
- `VK_SUBGROUP_FEATURE_ARITHMETIC_BIT` specifies the device will accept SPIR-V shader modules containing the `GroupNonUniformArithmetic` capability.
- `VK_SUBGROUP_FEATURE_BALLOT_BIT` specifies the device will accept SPIR-V shader modules containing the `GroupNonUniformBallot` capability.
- `VK_SUBGROUP_FEATURE_SHUFFLE_BIT` specifies the device will accept SPIR-V shader modules containing the `GroupNonUniformShuffle` capability.
- `VK_SUBGROUP_FEATURE_SHUFFLE_RELATIVE_BIT` specifies the device will accept SPIR-V shader modules containing the `GroupNonUniformShuffleRelative` capability.
- `VK_SUBGROUP_FEATURE_CLUSTERED_BIT` specifies the device will accept SPIR-V shader modules containing the `GroupNonUniformClustered` capability.
- `VK_SUBGROUP_FEATURE_QUAD_BIT` specifies the device will accept SPIR-V shader modules containing the `GroupNonUniformQuad` capability.
- `VK_SUBGROUP_FEATURE_PARTITIONED_BIT_NV` specifies the device will accept SPIR-V shader modules containing the `GroupNonUniformPartitionedNV` capability.
- `VK_SUBGROUP_FEATURE_ROTATE_BIT_KHR` specifies the device will accept SPIR-V shader modules containing the `GroupNonUniformRotateKHR` capability.
- `VK_SUBGROUP_FEATURE_ROTATE_CLUSTERED_BIT_KHR` specifies the device will accept SPIR-V shader modules that use the `ClusterSize` operand to `OpGroupNonUniformRotateKHR`.

```
// Provided by VK_VERSION_1_1
typedef VkFlags VkSubgroupFeatureFlags;
```

VkSubgroupFeatureFlags is a bitmask type for setting a mask of zero or more VkSubgroupFeatureFlagBits.

The VkPhysicalDeviceSubgroupSizeControlProperties structure is defined as:

```
typedef struct VkPhysicalDeviceSubgroupSizeControlProperties {
    VkStructureType       sType;
    void*                 pNext;
    uint32_t              minSubgroupSize;
    uint32_t              maxSubgroupSize;
    uint32_t              maxComputeWorkgroupSubgroups;
    VkShaderStageFlags    requiredSubgroupSizeStages;
} VkPhysicalDeviceSubgroupSizeControlProperties;
```

or the equivalent

```
// Provided by VK_EXT_subgroup_size_control
typedef VkPhysicalDeviceSubgroupSizeControlProperties
VkPhysicalDeviceSubgroupSizeControlPropertiesEXT;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- minSubgroupSize is the minimum subgroup size supported by this device. minSubgroupSize is at least one if any of the physical device's queues support VK_QUEUE_GRAPHICS_BIT or VK_QUEUE_COMPUTE_BIT. minSubgroupSize is a power-of-two. minSubgroupSize is less than or equal to maxSubgroupSize. minSubgroupSize is less than or equal to subgroupSize.
- maxSubgroupSize is the maximum subgroup size supported by this device. maxSubgroupSize is at least one if any of the physical device's queues support VK_QUEUE_GRAPHICS_BIT or VK_QUEUE_COMPUTE_BIT. maxSubgroupSize is a power-of-two. maxSubgroupSize is greater than or equal to minSubgroupSize. maxSubgroupSize is greater than or equal to subgroupSize.
- maxComputeWorkgroupSubgroups is the maximum number of subgroups supported by the implementation within a workgroup.
- requiredSubgroupSizeStages is a bitfield of what shader stages support having a required subgroup size specified.

If the VkPhysicalDeviceSubgroupSizeControlProperties structure is included in the pNext chain of the VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with each corresponding implementation-dependent property.

If VkPhysicalDeviceSubgroupProperties::supportedOperations includes VK_SUBGROUP_FEATURE_QUAD_BIT, minSubgroupSize **must** be greater than or equal to 4.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceSubgroupSizeControlProperties-sType-sType

sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SUBGROUP_SIZE_CONTROL_PROPERTIES

The VkPhysicalDeviceBlendOperationAdvancedPropertiesEXT structure is defined as:

```
// Provided by VK_EXT_blend_operation_advanced
typedef struct VkPhysicalDeviceBlendOperationAdvancedPropertiesEXT {
    VkStructureType    sType;
    void*              pNext;
    uint32_t           advancedBlendMaxColorAttachments;
    VkBool32           advancedBlendIndependentBlend;
    VkBool32           advancedBlendNonPremultipliedSrcColor;
    VkBool32           advancedBlendNonPremultipliedDstColor;
    VkBool32           advancedBlendCorrelatedOverlap;
    VkBool32           advancedBlendAllOperations;
} VkPhysicalDeviceBlendOperationAdvancedPropertiesEXT;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- advancedBlendMaxColorAttachments is one greater than the highest color attachment index that **can** be used in a subpass, for a pipeline that uses an advanced blend operation.
- advancedBlendIndependentBlend specifies whether advanced blend operations **can** vary per-attachment.
- advancedBlendNonPremultipliedSrcColor specifies whether the source color **can** be treated as non-premultiplied. If this is VK_FALSE, then VkPipelineColorBlendAdvancedStateCreateInfoEXT::srcPremultiplied **must** be VK_TRUE.
- advancedBlendNonPremultipliedDstColor specifies whether the destination color **can** be treated as non-premultiplied. If this is VK_FALSE, then VkPipelineColorBlendAdvancedStateCreateInfoEXT::dstPremultiplied **must** be VK_TRUE.
- advancedBlendCorrelatedOverlap specifies whether the overlap mode **can** be treated as correlated. If this is VK_FALSE, then VkPipelineColorBlendAdvancedStateCreateInfoEXT::blendOverlap **must** be VK_BLEND_OVERLAP_UNCORRELATED_EXT.
- advancedBlendAllOperations specifies whether all advanced blend operation enums are supported. See the valid usage of VkPipelineColorBlendAttachmentState.

If the VkPhysicalDeviceBlendOperationAdvancedPropertiesEXT structure is included in the pNext chain of the VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with each corresponding implementation-dependent property.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceBlendOperationAdvancedPropertiesEXT-sType-sType sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_BLEND_OPERATION_ADVANCED_PROPERTIES_EXT

The VkPhysicalDeviceVertexAttributeDivisorPropertiesEXT structure is defined as:

```
// Provided by VK_EXT_vertex_attribute_divisor
typedef struct VkPhysicalDeviceVertexAttributeDivisorPropertiesEXT {
    VkStructureType    sType;
    void*              pNext;
    uint32_t           maxVertexAttribDivisor;
} VkPhysicalDeviceVertexAttributeDivisorPropertiesEXT;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- maxVertexAttribDivisor is the maximum value of the number of instances that will repeat the value of vertex attribute data when instanced rendering is enabled.

If the VkPhysicalDeviceVertexAttributeDivisorPropertiesEXT structure is included in the pNext chain of the VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with each corresponding implementation-dependent property.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceVertexAttributeDivisorPropertiesEXT-sType-sType sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_VERTEX_ATTRIBUTE_DIVISOR_PROPERTIES_EXT

The VkPhysicalDeviceVertexAttributeDivisorPropertiesKHR structure is defined as:

```
// Provided by VK_KHR_vertex_attribute_divisor
typedef struct VkPhysicalDeviceVertexAttributeDivisorPropertiesKHR {
    VkStructureType    sType;
    void*              pNext;
    uint32_t           maxVertexAttribDivisor;
    VkBool32           supportsNonZeroFirstInstance;
} VkPhysicalDeviceVertexAttributeDivisorPropertiesKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- maxVertexAttribDivisor is the maximum value of the number of instances that will repeat the value of vertex attribute data when instanced rendering is enabled.
- supportsNonZeroFirstInstance specifies whether a non-zero value for the firstInstance parameter of drawing commands is supported when VkVertexInputBindingDivisorDescriptionKHR::divisor is not 1.

If the VkPhysicalDeviceVertexAttributeDivisorPropertiesKHR structure is included in the pNext chain of the VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled

in with each corresponding implementation-dependent property.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceVertexAttributeDivisorPropertiesKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_VERTEX_ATTRIBUTE_DIVISOR_PROPERTIES_KHR

The VkPhysicalDeviceSamplerFilterMinmaxProperties structure is defined as:

```
typedef struct VkPhysicalDeviceSamplerFilterMinmaxProperties {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           filterMinmaxSingleComponentFormats;
    VkBool32           filterMinmaxImageComponentMapping;
} VkPhysicalDeviceSamplerFilterMinmaxProperties;
```

or the equivalent

```
// Provided by VK_EXT_sampler_filter_minmax
typedef VkPhysicalDeviceSamplerFilterMinmaxProperties
VkPhysicalDeviceSamplerFilterMinmaxPropertiesEXT;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- filterMinmaxSingleComponentFormats is a boolean value indicating whether a minimum set of required formats support min/max filtering.
- filterMinmaxImageComponentMapping is a boolean value indicating whether the implementation supports non-identity component mapping of the image when doing min/max filtering.

If the VkPhysicalDeviceSamplerFilterMinmaxProperties structure is included in the pNext chain of the VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with each corresponding implementation-dependent property.

If filterMinmaxSingleComponentFormats is VK_TRUE, the following formats **must** support the VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT feature with VK_IMAGE_TILING_OPTIMAL, if they support VK_FORMAT_FEATURE_SAMPLED_IMAGE_BIT:

- VK_FORMAT_R8_UNORM
- VK_FORMAT_R8_SNORM
- VK_FORMAT_R16_UNORM
- VK_FORMAT_R16_SNORM
- VK_FORMAT_R16_SFLOAT

- VK_FORMAT_R32_SFLOAT
- VK_FORMAT_D16_UNORM
- VK_FORMAT_X8_D24_UNORM_PACK32
- VK_FORMAT_D32_SFLOAT
- VK_FORMAT_D16_UNORM_S8_UINT
- VK_FORMAT_D24_UNORM_S8_UINT
- VK_FORMAT_D32_SFLOAT_S8_UINT

If the format is a depth/stencil format, this bit only specifies that the depth aspect (not the stencil aspect) of an image of this format supports min/max filtering, and that min/max filtering of the depth aspect is supported when depth compare is disabled in the sampler.

If filterMinmaxImageComponentMapping is VK_FALSE the component mapping of the image view used with min/max filtering **must** have been created with the r component set to the identity swizzle. Only the r component of the sampled image value is defined and the other component values are undefined. If filterMinmaxImageComponentMapping is VK_TRUE this restriction does not apply and image component mapping works as normal.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceSamplerFilterMinmaxProperties-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SAMPLER_FILTER_MINMAX_PROPERTIES

The VkPhysicalDeviceProtectedMemoryProperties structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkPhysicalDeviceProtectedMemoryProperties {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           protectedNoFault;
} VkPhysicalDeviceProtectedMemoryProperties;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- protectedNoFault specifies how an implementation behaves when an application attempts to write to unprotected memory in a protected queue operation, read from protected memory in an unprotected queue operation, or perform a query in a protected queue operation. If this limit is VK_TRUE, such writes will be discarded or have undefined values written, reads and queries will return undefined values. If this limit is VK_FALSE, applications **must** not perform these operations. See Protected Memory Access Rules for more information.

If the VkPhysicalDeviceProtectedMemoryProperties structure is included in the pNext chain of the VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with

each corresponding implementation-dependent property.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceProtectedMemoryProperties-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PROTECTED_MEMORY_PROPERTIES`

The `VkPhysicalDeviceMaintenance3Properties` structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkPhysicalDeviceMaintenance3Properties {
    VkStructureType    sType;
    void*              pNext;
    uint32_t           maxPerSetDescriptors;
    VkDeviceSize       maxMemoryAllocationSize;
} VkPhysicalDeviceMaintenance3Properties;
```

or the equivalent

```
// Provided by VK_KHR_maintenance3
typedef VkPhysicalDeviceMaintenance3Properties
VkPhysicalDeviceMaintenance3PropertiesKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `maxPerSetDescriptors` is a maximum number of descriptors (summed over all descriptor types) in a single descriptor set that is guaranteed to satisfy any implementation-dependent constraints on the size of a descriptor set itself. Applications **can** query whether a descriptor set that goes beyond this limit is supported using vkGetDescriptorSetLayoutSupport.
- `maxMemoryAllocationSize` is the maximum size of a memory allocation that **can** be created, even if there is more space available in the heap.

If the `VkPhysicalDeviceMaintenance3Properties` structure is included in the `pNext` chain of the VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with each corresponding implementation-dependent property.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceMaintenance3Properties-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MAINTENANCE_3_PROPERTIES`

The `VkPhysicalDeviceMaintenance4Properties` structure is defined as:

```
typedef struct VkPhysicalDeviceMaintenance4Properties {
    VkStructureType    sType;
    void*              pNext;
    VkDeviceSize       maxBufferSize;
} VkPhysicalDeviceMaintenance4Properties;
```

or the equivalent

```
// Provided by VK_KHR_maintenance4
typedef VkPhysicalDeviceMaintenance4Properties
VkPhysicalDeviceMaintenance4PropertiesKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `maxBufferSize` is the maximum size `VkBuffer` that **can** be created.

If the `VkPhysicalDeviceMaintenance4Properties` structure is included in the `pNext` chain of the VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with each corresponding implementation-dependent property.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceMaintenance4Properties-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MAINTENANCE_4_PROPERTIES`

The `VkPhysicalDeviceMaintenance5PropertiesKHR` structure is defined as:

```
// Provided by VK_KHR_maintenance5
typedef struct VkPhysicalDeviceMaintenance5PropertiesKHR {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           earlyFragmentMultisampleCoverageAfterSampleCounting;
    VkBool32           earlyFragmentSampleMaskTestBeforeSampleCounting;
    VkBool32           depthStencilSwizzleOneSupport;
    VkBool32           polygonModePointSize;
    VkBool32           nonStrictSinglePixelWideLinesUseParallelogram;
    VkBool32           nonStrictWideLinesUseParallelogram;
} VkPhysicalDeviceMaintenance5PropertiesKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `earlyFragmentMultisampleCoverageAfterSampleCounting` is a boolean value indicating whether the fragment shading and multisample coverage operations are performed after sample counting

for fragment shaders with EarlyFragmentTests execution mode.

- earlyFragmentSampleMaskTestBeforeSampleCounting is a boolean value indicating whether the sample mask test operation is performed before sample counting for fragment shaders using the EarlyFragmentTests execution mode.
- depthStencilSwizzleOneSupport is a boolean indicating that depth/stencil texturing operations with VK_COMPONENT_SWIZZLE_ONE have defined behavior.
- polygonModePointSize is a boolean value indicating whether the point size of the final rasterization of polygons with VK_POLYGON_MODE_POINT is controlled by PointSize.
- nonStrictSinglePixelWideLinesUseParallelogram is a boolean value indicating whether non-strict lines with a width of 1.0 are rasterized as parallelograms or using Bresenham's algorithm.
- nonStrictWideLinesUseParallelogram is a boolean value indicating whether non-strict lines with a width greater than 1.0 are rasterized as parallelograms or using Bresenham's algorithm.

If the VkPhysicalDeviceMaintenance5PropertiesKHR structure is included in the pNext chain of the VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with each corresponding implementation-dependent property.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceMaintenance5PropertiesKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MAINTENANCE_5_PROPERTIES_KHR

The VkPhysicalDeviceMaintenance6PropertiesKHR structure is defined as:

```
// Provided by VK_KHR_maintenance6
typedef struct VkPhysicalDeviceMaintenance6PropertiesKHR {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           blockTexelViewCompatibleMultipleLayers;
    uint32_t           maxCombinedImageSamplerDescriptorCount;
    VkBool32           fragmentShadingRateClampCombinerInputs;
} VkPhysicalDeviceMaintenance6PropertiesKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- blockTexelViewCompatibleMultipleLayers is a boolean value indicating that an implementation supports creating image views with VK_IMAGE_CREATE_BLOCK_TEXEL_VIEW_COMPATIBLE_BIT where the layerCount member of subresourceRange is greater than 1.
- maxCombinedImageSamplerDescriptorCount is the maximum number of combined image sampler descriptors that the implementation uses to access any of the formats that require a sampler $Y'C_BC_R$ conversion supported by the implementation.
- fragmentShadingRateClampCombinerInputs is a boolean value indicating that an implementation clamps the inputs to combiner operations.

If the VkPhysicalDeviceMaintenance6PropertiesKHR structure is included in the pNext chain of the VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with each corresponding implementation-dependent property.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceMaintenance6PropertiesKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MAINTENANCE_6_PROPERTIES_KHR

The VkPhysicalDeviceMaintenance7PropertiesKHR structure is defined as:

```
// Provided by VK_KHR_maintenance7
typedef struct VkPhysicalDeviceMaintenance7PropertiesKHR {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           robustFragmentShadingRateAttachmentAccess;
    VkBool32           separateDepthStencilAttachmentAccess;
    uint32_t           maxDescriptorSetTotalUniformBuffersDynamic;
    uint32_t           maxDescriptorSetTotalStorageBuffersDynamic;
    uint32_t           maxDescriptorSetTotalBuffersDynamic;
    uint32_t           maxDescriptorSetUpdateAfterBindTotalUniformBuffersDynamic;
    uint32_t           maxDescriptorSetUpdateAfterBindTotalStorageBuffersDynamic;
    uint32_t           maxDescriptorSetUpdateAfterBindTotalBuffersDynamic;
} VkPhysicalDeviceMaintenance7PropertiesKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- robustFragmentShadingRateAttachmentAccess indicates whether the scaled size of a fragment shading rate attachment **can** be less than the size of the render area. If robustFragmentShadingRateAttachmentAccess is VK_FALSE, the size of the attachment multiplied by the texel size **must** be greater than or equal to the size of the render area. If it is VK_TRUE and the fragment shading rate attachment was created with VkImageSubresourceRange::baseMipLevel equal to 0, the scaled size **can** be smaller than the render area, and shading rates for missing texels are defined by texel replacement for invalid texels.
- separateDepthStencilAttachmentAccess indicates support for writing to one aspect of a depth/stencil attachment without performing read-modify-write operations on the other aspect. If this property is VK_TRUE, writes to one aspect **must** not result in read-modify-write operations on the other aspect. If VK_FALSE, writes to one aspect **may** result in writes to the other aspect as defined by render pass load operations, render pass store operations and render pass resolve operations.
- maxDescriptorSetTotalUniformBuffersDynamic is the maximum total count of dynamic uniform buffers that **can** be included in a pipeline layout. Descriptors with a type of VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC count against this limit. Only descriptors in descriptor set layouts created without the VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT bit set count against this limit. See

Dynamic Uniform Buffer.

- maxDescriptorSetTotalStorageBuffersDynamic is the maximum total count of dynamic storage buffers that **can** be included in a pipeline layout. Descriptors with a type of VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC count against this limit. Only descriptors in descriptor set layouts created without the VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT bit set count against this limit. See Dynamic Storage Buffer.
- maxDescriptorSetTotalBuffersDynamic is the maximum total count of dynamic uniform buffers and storage buffers that **can** be included in a pipeline layout. Descriptors with a type of VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC or VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC count against this limit. Only descriptors in descriptor set layouts created without the VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT bit set count against this limit.
- maxDescriptorSetUpdateAfterBindTotalUniformBuffersDynamic is similar to maxDescriptorSetTotalUniformBuffersDynamic but counts descriptors from descriptor sets created with or without the VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT bit set.
- maxDescriptorSetUpdateAfterBindTotalStorageBuffersDynamic is similar to maxDescriptorSetTotalStorageBuffersDynamic but counts descriptors from descriptor sets created with or without the VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT bit set.
- maxDescriptorSetUpdateAfterBindTotalBuffersDynamic is similar to maxDescriptorSetTotalBuffersDynamic but counts descriptors from descriptor sets created with or without the VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT bit set. While an application **can** allocate dynamic storage buffer descriptors from a pool created with the VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT, bindings for these descriptors **must** not be present in any descriptor set layout that includes bindings created with VK_DESCRIPTOR_BINDING_UPDATE_AFTER_BIND_BIT.

If the VkPhysicalDeviceMaintenance7PropertiesKHR structure is included in the pNext chain of the VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with each corresponding implementation-dependent property.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceMaintenance7PropertiesKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MAINTENANCE_7_PROPERTIES_KHR

The VkPhysicalDeviceLayeredApiPropertiesListKHR structure is defined as:

```
// Provided by VK_KHR_maintenance7
typedef struct VkPhysicalDeviceLayeredApiPropertiesListKHR {
    VkStructureType                             sType;
    void*                                       pNext;
    uint32_t                                    layeredApiCount;
    VkPhysicalDeviceLayeredApiPropertiesKHR*    pLayeredApis;
} VkPhysicalDeviceLayeredApiPropertiesListKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- layeredApiCount is an integer related to the number of layered implementations underneath the Vulkan physical device, as described below.
- pLayeredApis is a pointer to an array of VkPhysicalDeviceLayeredApiPropertiesKHR in which information regarding the layered implementations underneath the Vulkan physical device are returned.

If pLayeredApis is NULL, then the number of layered implementations that are underneath the top-most Vulkan physical device (i.e. the one returned by vkGetPhysicalDeviceProperties2) is returned in layeredApiCount. Otherwise, layeredApiCount **must** be set by the application to the number of elements in the pLayeredApis array, and on return the variable is overwritten with the number of values actually written to pLayeredApis. If the value of layeredApiCount is less than the number of layered implementations underneath the Vulkan physical device, at most layeredApiCount values will be written to pLayeredApis. An implementation that is not a layer will return 0 in layeredApiCount.

In the presence of multiple layered implementations, each element of pLayeredApis corresponds to an API implementation that is implemented on top of the API at the previous index. If there are layered implementations underneath a non-Vulkan implementation, they may not be visible in this query as the corresponding APIs may lack such a query.

If the VkPhysicalDeviceLayeredApiPropertiesListKHR structure is included in the pNext chain of the VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with each corresponding implementation-dependent property.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceLayeredApiPropertiesListKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_LAYERED_API_PROPERTIES_LIST_KHR
- VUID-VkPhysicalDeviceLayeredApiPropertiesListKHR-pLayeredApis-parameter
  If layeredApiCount is not 0, and pLayeredApis is not NULL, pLayeredApis **must** be a valid pointer to an array of layeredApiCount VkPhysicalDeviceLayeredApiPropertiesKHR structures

The VkPhysicalDeviceLayeredApiPropertiesKHR structure is defined as:

```
// Provided by VK_KHR_maintenance7
typedef struct VkPhysicalDeviceLayeredApiPropertiesKHR {
    VkStructureType                    sType;
    void*                              pNext;
    uint32_t                           vendorID;
    uint32_t                           deviceID;
    VkPhysicalDeviceLayeredApiKHR      layeredAPI;
    char                               deviceName[VK_MAX_PHYSICAL_DEVICE_NAME_SIZE];
} VkPhysicalDeviceLayeredApiPropertiesKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `vendorID` is a unique identifier for the vendor of the layered implementation.
- `deviceID` is a unique identifier for the layered implementation among devices available from the vendor.
- `layeredAPI` is a VkPhysicalDeviceLayeredApiKHR specifying the API implemented by the layered implementation.
- `deviceName` is an array of `VK_MAX_PHYSICAL_DEVICE_NAME_SIZE` `char` containing a null-terminated UTF-8 string which is the name of the device.

If `layeredAPI` is `VK_PHYSICAL_DEVICE_LAYERED_API_VULKAN_KHR`, additional Vulkan-specific information can be queried by including the VkPhysicalDeviceLayeredApiVulkanPropertiesKHR structure in the `pNext` chain. Otherwise if such a structure is included in the `pNext` chain, it is ignored.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceLayeredApiPropertiesKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_LAYERED_API_PROPERTIES_KHR`
- VUID-VkPhysicalDeviceLayeredApiPropertiesKHR-pNext-pNext
  `pNext` **must** be `NULL` or a pointer to a valid instance of VkPhysicalDeviceLayeredApiVulkanPropertiesKHR
- VUID-VkPhysicalDeviceLayeredApiPropertiesKHR-sType-unique
  The `sType` value of each struct in the `pNext` chain **must** be unique

The list of possible API implementations of a layered implementation underneath the Vulkan physical device, as returned in VkPhysicalDeviceLayeredApiPropertiesKHR::`layeredAPI`, are:

```
// Provided by VK_KHR_maintenance7
typedef enum VkPhysicalDeviceLayeredApiKHR {
    VK_PHYSICAL_DEVICE_LAYERED_API_VULKAN_KHR = 0,
    VK_PHYSICAL_DEVICE_LAYERED_API_D3D12_KHR = 1,
    VK_PHYSICAL_DEVICE_LAYERED_API_METAL_KHR = 2,
    VK_PHYSICAL_DEVICE_LAYERED_API_OPENGL_KHR = 3,
    VK_PHYSICAL_DEVICE_LAYERED_API_OPENGLES_KHR = 4,
} VkPhysicalDeviceLayeredApiKHR;
```

- `VK_PHYSICAL_DEVICE_LAYERED_API_VULKAN_KHR` - the device implements the Vulkan API.
- `VK_PHYSICAL_DEVICE_LAYERED_API_D3D12_KHR` - the device implements the D3D12 API.
- `VK_PHYSICAL_DEVICE_LAYERED_API_METAL_KHR` - the device implements the Metal API.
- `VK_PHYSICAL_DEVICE_LAYERED_API_OPENGL_KHR` - the device implements the OpenGL API.
- `VK_PHYSICAL_DEVICE_LAYERED_API_OPENGLES_KHR` - the device implements the OpenGL ES API.

The `VkPhysicalDeviceLayeredApiVulkanPropertiesKHR` structure is defined as:

```
// Provided by VK_KHR_maintenance7
typedef struct VkPhysicalDeviceLayeredApiVulkanPropertiesKHR {
    VkStructureType                sType;
    void*                          pNext;
    VkPhysicalDeviceProperties2    properties;
} VkPhysicalDeviceLayeredApiVulkanPropertiesKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `properties` is a VkPhysicalDeviceProperties2 in which properties of the underlying layered Vulkan implementation are returned.

The implementation **must** zero-fill the contents of `properties.properties.limits` and `properties.properties.sparseProperties`.

### Valid Usage

- VUID-VkPhysicalDeviceLayeredApiVulkanPropertiesKHR-pNext-10011
  Only VkPhysicalDeviceDriverProperties and VkPhysicalDeviceIDProperties are allowed in the `pNext` chain of `properties`

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceLayeredApiVulkanPropertiesKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_LAYERED_API_VULKAN_PROPERTIES_KHR`

The VkPhysicalDeviceMeshShaderPropertiesNV structure is defined as:

```
// Provided by VK_NV_mesh_shader
typedef struct VkPhysicalDeviceMeshShaderPropertiesNV {
    VkStructureType    sType;
    void*              pNext;
    uint32_t           maxDrawMeshTasksCount;
    uint32_t           maxTaskWorkGroupInvocations;
    uint32_t           maxTaskWorkGroupSize[3];
    uint32_t           maxTaskTotalMemorySize;
    uint32_t           maxTaskOutputCount;
    uint32_t           maxMeshWorkGroupInvocations;
    uint32_t           maxMeshWorkGroupSize[3];
    uint32_t           maxMeshTotalMemorySize;
    uint32_t           maxMeshOutputVertices;
    uint32_t           maxMeshOutputPrimitives;
    uint32_t           maxMeshMultiviewViewCount;
    uint32_t           meshOutputPerVertexGranularity;
    uint32_t           meshOutputPerPrimitiveGranularity;
} VkPhysicalDeviceMeshShaderPropertiesNV;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- maxDrawMeshTasksCount is the maximum number of local workgroups that **can** be launched by a single draw mesh tasks command. See Programmable Mesh Shading.
- maxTaskWorkGroupInvocations is the maximum total number of task shader invocations in a single local workgroup. The product of the X, Y, and Z sizes, as specified by the LocalSize or LocalSizeId execution mode in shader modules or by the object decorated by the WorkgroupSize decoration, **must** be less than or equal to this limit.
- maxTaskWorkGroupSize[3] is the maximum size of a local task workgroup. These three values represent the maximum local workgroup size in the X, Y, and Z dimensions, respectively. The x, y, and z sizes, as specified by the LocalSize or LocalSizeId execution mode or by the object decorated by the WorkgroupSize decoration in shader modules, **must** be less than or equal to the corresponding limit.
- maxTaskTotalMemorySize is the maximum number of bytes that the task shader can use in total for shared and output memory combined.
- maxTaskOutputCount is the maximum number of output tasks a single task shader workgroup can emit.
- maxMeshWorkGroupInvocations is the maximum total number of mesh shader invocations in a single local workgroup. The product of the X, Y, and Z sizes, as specified by the LocalSize or LocalSizeId execution mode in shader modules or by the object decorated by the WorkgroupSize decoration, **must** be less than or equal to this limit.
- maxMeshWorkGroupSize[3] is the maximum size of a local mesh workgroup. These three values represent the maximum local workgroup size in the X, Y, and Z dimensions, respectively. The x, y, and z sizes, as specified by the LocalSize or LocalSizeId execution mode or by the object

decorated by the `WorkgroupSize` decoration in shader modules, **must** be less than or equal to the corresponding limit.

- `maxMeshTotalMemorySize` is the maximum number of bytes that the mesh shader can use in total for shared and output memory combined.
- `maxMeshOutputVertices` is the maximum number of vertices a mesh shader output can store.
- `maxMeshOutputPrimitives` is the maximum number of primitives a mesh shader output can store.
- `maxMeshMultiviewViewCount` is the maximum number of multiview views a mesh shader can use.
- `meshOutputPerVertexGranularity` is the granularity with which mesh vertex outputs are allocated. The value can be used to compute the memory size used by the mesh shader, which **must** be less than or equal to `maxMeshTotalMemorySize`.
- `meshOutputPerPrimitiveGranularity` is the granularity with which mesh outputs qualified as per-primitive are allocated. The value can be used to compute the memory size used by the mesh shader, which **must** be less than or equal to `maxMeshTotalMemorySize`.

If the `VkPhysicalDeviceMeshShaderPropertiesNV` structure is included in the `pNext` chain of the VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with each corresponding implementation-dependent property.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceMeshShaderPropertiesNV-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MESH_SHADER_PROPERTIES_NV`

The `VkPhysicalDeviceMeshShaderPropertiesEXT` structure is defined as:

```
// Provided by VK_EXT_mesh_shader
typedef struct VkPhysicalDeviceMeshShaderPropertiesEXT {
    VkStructureType    sType;
    void*              pNext;
    uint32_t           maxTaskWorkGroupTotalCount;
    uint32_t           maxTaskWorkGroupCount[3];
    uint32_t           maxTaskWorkGroupInvocations;
    uint32_t           maxTaskWorkGroupSize[3];
    uint32_t           maxTaskPayloadSize;
    uint32_t           maxTaskSharedMemorySize;
    uint32_t           maxTaskPayloadAndSharedMemorySize;
    uint32_t           maxMeshWorkGroupTotalCount;
    uint32_t           maxMeshWorkGroupCount[3];
    uint32_t           maxMeshWorkGroupInvocations;
    uint32_t           maxMeshWorkGroupSize[3];
    uint32_t           maxMeshSharedMemorySize;
    uint32_t           maxMeshPayloadAndSharedMemorySize;
    uint32_t           maxMeshOutputMemorySize;
    uint32_t           maxMeshPayloadAndOutputMemorySize;
    uint32_t           maxMeshOutputComponents;
    uint32_t           maxMeshOutputVertices;
    uint32_t           maxMeshOutputPrimitives;
    uint32_t           maxMeshOutputLayers;
    uint32_t           maxMeshMultiviewViewCount;
    uint32_t           meshOutputPerVertexGranularity;
    uint32_t           meshOutputPerPrimitiveGranularity;
    uint32_t           maxPreferredTaskWorkGroupInvocations;
    uint32_t           maxPreferredMeshWorkGroupInvocations;
    VkBool32           prefersLocalInvocationVertexOutput;
    VkBool32           prefersLocalInvocationPrimitiveOutput;
    VkBool32           prefersCompactVertexOutput;
    VkBool32           prefersCompactPrimitiveOutput;
} VkPhysicalDeviceMeshShaderPropertiesEXT;
```

The members of the `VkPhysicalDeviceMeshShaderPropertiesEXT` structure describe the following implementation-dependent limits:

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `maxTaskWorkGroupTotalCount` is the maximum number of total local workgroups that **can** be launched by a single mesh tasks drawing command. See Programmable Mesh Shading.
- `maxTaskWorkGroupCount`[3] is the maximum number of local workgroups that **can** be launched by a single mesh tasks drawing command. These three values represent the maximum number of local workgroups for the X, Y, and Z dimensions, respectively. The workgroup count parameters to the drawing commands **must** be less than or equal to the corresponding limit. The product of these dimensions **must** be less than or equal to `maxTaskWorkGroupTotalCount`.
- `maxTaskWorkGroupInvocations` is the maximum total number of task shader invocations in a

single local workgroup. The product of the X, Y, and Z sizes, as specified by the `LocalSize` or `LocalSizeId` execution mode in shader modules or by the object decorated by the `WorkgroupSize` decoration, **must** be less than or equal to this limit.

- `maxTaskWorkGroupSize`[3] is the maximum size of a local task workgroup, per dimension. These three values represent the maximum local workgroup size in the X, Y, and Z dimensions, respectively. The `x`, `y`, and `z` sizes, as specified by the `LocalSize` or `LocalSizeId` execution mode or by the object decorated by the `WorkgroupSize` decoration in shader modules, **must** be less than or equal to the corresponding limit.
- `maxTaskPayloadSize` is the maximum total storage size, in bytes, available for variables declared with the `TaskPayloadWorkgroupEXT` storage class in shader modules in the task shader stage.
- `maxTaskSharedMemorySize` is the maximum total storage size, in bytes, available for variables declared with the `Workgroup` storage class in shader modules in the task shader stage.
- `maxTaskPayloadAndSharedMemorySize` is the maximum total storage size, in bytes, available for variables that are declared with the `TaskPayloadWorkgroupEXT` or `Workgroup` storage class, in shader modules in the task shader stage.
- `maxMeshWorkGroupTotalCount` is the maximum number of local output tasks a single task shader workgroup can emit.
- `maxMeshWorkGroupCount`[3] is the maximum number of local output tasks a single task shader workgroup can emit, per dimension. These three values represent the maximum number of local output tasks for the X, Y, and Z dimensions, respectively. The workgroup count parameters to the `OpEmitMeshTasksEXT` **must** be less than or equal to the corresponding limit. The product of these dimensions **must** be less than or equal to `maxMeshWorkGroupTotalCount`.
- `maxMeshWorkGroupInvocations` is the maximum total number of mesh shader invocations in a single local workgroup. The product of the X, Y, and Z sizes, as specified by the `LocalSize` or `LocalSizeId` execution mode in shader modules or by the object decorated by the `WorkgroupSize` decoration, **must** be less than or equal to this limit.
- `maxMeshWorkGroupSize`[3] is the maximum size of a local mesh workgroup, per dimension. These three values represent the maximum local workgroup size in the X, Y, and Z dimensions, respectively. The `x`, `y`, and `z` sizes, as specified by the `LocalSize` or `LocalSizeId` execution mode or by the object decorated by the `WorkgroupSize` decoration in shader modules, **must** be less than or equal to the corresponding limit.
- `maxMeshSharedMemorySize` is the maximum total storage size, in bytes, available for variables declared with the `Workgroup` storage class in shader modules in the mesh shader stage.
- `maxMeshPayloadAndSharedMemorySize` is the maximum total storage size, in bytes, available for variables that are declared with the `TaskPayloadWorkgroupEXT` or `Workgroup` storage class in shader modules in the mesh shader stage.
- `maxMeshOutputMemorySize` is the maximum total storage size, in bytes, available for output variables in shader modules in the mesh shader stage, according to the formula in Mesh Shader Output.
- `maxMeshPayloadAndOutputMemorySize` is the maximum total storage size, in bytes, available for variables that are declared with the `TaskPayloadWorkgroupEXT` storage class, or output variables in shader modules in the mesh shader stage, according to the formula in Mesh Shader Output.
- `maxMeshOutputComponents` is the maximum number of components of output variables which **can**

be output from the mesh shader stage.

- `maxMeshOutputVertices` is the maximum number of vertices which **can** be emitted by a single mesh shader workgroup.
- `maxMeshOutputPrimitives` is the maximum number of primitives which **can** be emitted by a single mesh shader workgroup.
- `maxMeshOutputLayers` is one greater than the maximum layer index that **can** be output from the mesh shader stage.
- `maxMeshMultiviewViewCount` is one greater than the maximum view index that **can** be used by any mesh shader.
- `meshOutputPerVertexGranularity` is the granularity of vertex allocation. The number of output vertices allocated for the mesh shader stage is padded to a multiple of this number. The value can be used to calculate the required storage size for output variables in shader modules in the mesh shader stage, which **must** be less than or equal to `maxMeshOutputMemorySize`.
- `meshOutputPerPrimitiveGranularity` is the granularity of primitive allocation. The number of output primitives allocated for the mesh shader stage is padded to a multiple of this number. The value can be used to calculate the required storage size for output variables in shader modules in the mesh shader stage, which **must** be less than or equal to `maxMeshOutputMemorySize`.
- `maxPreferredTaskWorkGroupInvocations` is the maximum number of task shader invocations in a single workgroup that is preferred by the implementation for optimal performance. The value is guaranteed to be a multiple of a supported subgroup size for the task shader stage.
- `maxPreferredMeshWorkGroupInvocations` is the maximum number of mesh shader invocations in a single workgroup that is preferred by the implementation for optimal performance. The value is guaranteed to be a multiple of a supported subgroup size for the mesh shader stage.
- `prefersLocalInvocationVertexOutput` specifies whether writes to the vertex output array in a mesh shader yield best performance when the array index matches `LocalInvocationIndex`.
- `prefersLocalInvocationPrimitiveOutput` specifies whether writes to the primitive output array in a mesh shader yield best performance when the array index matches `LocalInvocationIndex`.
- `prefersCompactVertexOutput` specifies whether output vertices should be compacted after custom culling in the mesh shader for best performance, otherwise keeping the vertices at their original location may be better.
- `prefersCompactPrimitiveOutput` specifies whether output primitives should be compacted after custom culling in the mesh shader for best performance, otherwise the use of `CullPrimitiveEXT` may be better.

If the `VkPhysicalDeviceMeshShaderPropertiesEXT` structure is included in the `pNext` chain of VkPhysicalDeviceProperties2, it is filled with the implementation-dependent limits.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceMeshShaderPropertiesEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MESH_SHADER_PROPERTIES_EXT`

The VkPhysicalDeviceDescriptorIndexingProperties structure is defined as:

```
typedef struct VkPhysicalDeviceDescriptorIndexingProperties {
    VkStructureType    sType;
    void*              pNext;
    uint32_t           maxUpdateAfterBindDescriptorsInAllPools;
    VkBool32           shaderUniformBufferArrayNonUniformIndexingNative;
    VkBool32           shaderSampledImageArrayNonUniformIndexingNative;
    VkBool32           shaderStorageBufferArrayNonUniformIndexingNative;
    VkBool32           shaderStorageImageArrayNonUniformIndexingNative;
    VkBool32           shaderInputAttachmentArrayNonUniformIndexingNative;
    VkBool32           robustBufferAccessUpdateAfterBind;
    VkBool32           quadDivergentImplicitLod;
    uint32_t           maxPerStageDescriptorUpdateAfterBindSamplers;
    uint32_t           maxPerStageDescriptorUpdateAfterBindUniformBuffers;
    uint32_t           maxPerStageDescriptorUpdateAfterBindStorageBuffers;
    uint32_t           maxPerStageDescriptorUpdateAfterBindSampledImages;
    uint32_t           maxPerStageDescriptorUpdateAfterBindStorageImages;
    uint32_t           maxPerStageDescriptorUpdateAfterBindInputAttachments;
    uint32_t           maxPerStageUpdateAfterBindResources;
    uint32_t           maxDescriptorSetUpdateAfterBindSamplers;
    uint32_t           maxDescriptorSetUpdateAfterBindUniformBuffers;
    uint32_t           maxDescriptorSetUpdateAfterBindUniformBuffersDynamic;
    uint32_t           maxDescriptorSetUpdateAfterBindStorageBuffers;
    uint32_t           maxDescriptorSetUpdateAfterBindStorageBuffersDynamic;
    uint32_t           maxDescriptorSetUpdateAfterBindSampledImages;
    uint32_t           maxDescriptorSetUpdateAfterBindStorageImages;
    uint32_t           maxDescriptorSetUpdateAfterBindInputAttachments;
} VkPhysicalDeviceDescriptorIndexingProperties;
```

or the equivalent

```
// Provided by VK_EXT_descriptor_indexing
typedef VkPhysicalDeviceDescriptorIndexingProperties
VkPhysicalDeviceDescriptorIndexingPropertiesEXT;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- maxUpdateAfterBindDescriptorsInAllPools is the maximum number of descriptors (summed over all descriptor types) that **can** be created across all pools that are created with the VK_DESCRIPTOR_POOL_CREATE_UPDATE_AFTER_BIND_BIT bit set. Pool creation **may** fail when this limit is exceeded, or when the space this limit represents is unable to satisfy a pool creation due to fragmentation.
- shaderUniformBufferArrayNonUniformIndexingNative is a boolean value indicating whether uniform buffer descriptors natively support non-uniform indexing. If this is VK_FALSE, then a single dynamic instance of an instruction that non-uniformly indexes an array of uniform

buffers **may** execute multiple times in order to access all the descriptors.

- `shaderSampledImageArrayNonUniformIndexingNative` is a boolean value indicating whether sampler and image descriptors natively support non-uniform indexing. If this is `VK_FALSE`, then a single dynamic instance of an instruction that non-uniformly indexes an array of samplers or images **may** execute multiple times in order to access all the descriptors.
- `shaderStorageBufferArrayNonUniformIndexingNative` is a boolean value indicating whether storage buffer descriptors natively support non-uniform indexing. If this is `VK_FALSE`, then a single dynamic instance of an instruction that non-uniformly indexes an array of storage buffers **may** execute multiple times in order to access all the descriptors.
- `shaderStorageImageArrayNonUniformIndexingNative` is a boolean value indicating whether storage image descriptors natively support non-uniform indexing. If this is `VK_FALSE`, then a single dynamic instance of an instruction that non-uniformly indexes an array of storage images **may** execute multiple times in order to access all the descriptors.
- `shaderInputAttachmentArrayNonUniformIndexingNative` is a boolean value indicating whether input attachment descriptors natively support non-uniform indexing. If this is `VK_FALSE`, then a single dynamic instance of an instruction that non-uniformly indexes an array of input attachments **may** execute multiple times in order to access all the descriptors.
- `robustBufferAccessUpdateAfterBind` is a boolean value indicating whether `robustBufferAccess` **can** be enabled on a device simultaneously with `descriptorBindingUniformBufferUpdateAfterBind`, `descriptorBindingStorageBufferUpdateAfterBind`, `descriptorBindingUniformTexelBufferUpdateAfterBind`, and/or `descriptorBindingStorageTexelBufferUpdateAfterBind`. If this is `VK_FALSE`, then either `robustBufferAccess` **must** be disabled or all of these update-after-bind features **must** be disabled.
- `quadDivergentImplicitLod` is a boolean value indicating whether implicit LOD calculations for image operations have well-defined results when the image and/or sampler objects used for the instruction are not uniform within a quad. See Derivative Image Operations.
- `maxPerStageDescriptorUpdateAfterBindSamplers` is similar to `maxPerStageDescriptorSamplers` but counts descriptors from descriptor sets created with or without the `VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT` bit set.
- `maxPerStageDescriptorUpdateAfterBindUniformBuffers` is similar to `maxPerStageDescriptorUniformBuffers` but counts descriptors from descriptor sets created with or without the `VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT` bit set.
- `maxPerStageDescriptorUpdateAfterBindStorageBuffers` is similar to `maxPerStageDescriptorStorageBuffers` but counts descriptors from descriptor sets created with or without the `VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT` bit set.
- `maxPerStageDescriptorUpdateAfterBindSampledImages` is similar to `maxPerStageDescriptorSampledImages` but counts descriptors from descriptor sets created with or without the `VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT` bit set.
- `maxPerStageDescriptorUpdateAfterBindStorageImages` is similar to `maxPerStageDescriptorStorageImages` but counts descriptors from descriptor sets created with or without the `VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT` bit set.
- `maxPerStageDescriptorUpdateAfterBindInputAttachments` is similar to `maxPerStageDescriptorInputAttachments` but counts descriptors from descriptor sets created with

or without the VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT bit set.

- maxPerStageUpdateAfterBindResources is similar to maxPerStageResources but counts descriptors from descriptor sets created with or without the VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT bit set.
- maxDescriptorSetUpdateAfterBindSamplers is similar to maxDescriptorSetSamplers but counts descriptors from descriptor sets created with or without the VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT bit set.
- maxDescriptorSetUpdateAfterBindUniformBuffers is similar to maxDescriptorSetUniformBuffers but counts descriptors from descriptor sets created with or without the VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT bit set.
- maxDescriptorSetUpdateAfterBindUniformBuffersDynamic is similar to maxDescriptorSetUniformBuffersDynamic but counts descriptors from descriptor sets created with or without the VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT bit set. While an application **can** allocate dynamic uniform buffer descriptors from a pool created with the VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT, bindings for these descriptors **must** not be present in any descriptor set layout that includes bindings created with VK_DESCRIPTOR_BINDING_UPDATE_AFTER_BIND_BIT.
- maxDescriptorSetUpdateAfterBindStorageBuffers is similar to maxDescriptorSetStorageBuffers but counts descriptors from descriptor sets created with or without the VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT bit set.
- maxDescriptorSetUpdateAfterBindStorageBuffersDynamic is similar to maxDescriptorSetStorageBuffersDynamic but counts descriptors from descriptor sets created with or without the VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT bit set. While an application **can** allocate dynamic storage buffer descriptors from a pool created with the VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT, bindings for these descriptors **must** not be present in any descriptor set layout that includes bindings created with VK_DESCRIPTOR_BINDING_UPDATE_AFTER_BIND_BIT.
- maxDescriptorSetUpdateAfterBindSampledImages is similar to maxDescriptorSetSampledImages but counts descriptors from descriptor sets created with or without the VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT bit set.
- maxDescriptorSetUpdateAfterBindStorageImages is similar to maxDescriptorSetStorageImages but counts descriptors from descriptor sets created with or without the VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT bit set.
- maxDescriptorSetUpdateAfterBindInputAttachments is similar to maxDescriptorSetInputAttachments but counts descriptors from descriptor sets created with or without the VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT bit set.

If the VkPhysicalDeviceDescriptorIndexingProperties structure is included in the pNext chain of the VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with each corresponding implementation-dependent property.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceDescriptorIndexingProperties-sType-sType

sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DESCRIPTOR_INDEXING_PROPERTIES

The VkPhysicalDeviceInlineUniformBlockProperties structure is defined as:

```
typedef struct VkPhysicalDeviceInlineUniformBlockProperties {
    VkStructureType    sType;
    void*              pNext;
    uint32_t           maxInlineUniformBlockSize;
    uint32_t           maxPerStageDescriptorInlineUniformBlocks;
    uint32_t           maxPerStageDescriptorUpdateAfterBindInlineUniformBlocks;
    uint32_t           maxDescriptorSetInlineUniformBlocks;
    uint32_t           maxDescriptorSetUpdateAfterBindInlineUniformBlocks;
} VkPhysicalDeviceInlineUniformBlockProperties;
```

or the equivalent

```
// Provided by VK_EXT_inline_uniform_block
typedef VkPhysicalDeviceInlineUniformBlockProperties
VkPhysicalDeviceInlineUniformBlockPropertiesEXT;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- maxInlineUniformBlockSize is the maximum size in bytes of an inline uniform block binding.
- maxPerStageDescriptorInlineUniformBlocks is the maximum number of inline uniform block bindings that **can** be accessible to a single shader stage in a pipeline layout. Descriptor bindings with a descriptor type of VK_DESCRIPTOR_TYPE_INLINE_UNIFORM_BLOCK count against this limit. Only descriptor bindings in descriptor set layouts created without the VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT bit set count against this limit.
- maxPerStageDescriptorUpdateAfterBindInlineUniformBlocks is similar to maxPerStageDescriptorInlineUniformBlocks but counts descriptor bindings from descriptor sets created with or without the VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT bit set.
- maxDescriptorSetInlineUniformBlocks is the maximum number of inline uniform block bindings that **can** be included in descriptor bindings in a pipeline layout across all pipeline shader stages and descriptor set numbers. Descriptor bindings with a descriptor type of VK_DESCRIPTOR_TYPE_INLINE_UNIFORM_BLOCK count against this limit. Only descriptor bindings in descriptor set layouts created without the VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT bit set count against this limit.
- maxDescriptorSetUpdateAfterBindInlineUniformBlocks is similar to maxDescriptorSetInlineUniformBlocks but counts descriptor bindings from descriptor sets created with or without the VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT bit set.

If the VkPhysicalDeviceInlineUniformBlockProperties structure is included in the pNext chain of the VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with each corresponding implementation-dependent property.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceInlineUniformBlockProperties-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_INLINE_UNIFORM_BLOCK_PROPERTIES`

The `VkPhysicalDeviceConservativeRasterizationPropertiesEXT` structure is defined as:

```
// Provided by VK_EXT_conservative_rasterization
typedef struct VkPhysicalDeviceConservativeRasterizationPropertiesEXT {
    VkStructureType    sType;
    void*              pNext;
    float              primitiveOverestimationSize;
    float              maxExtraPrimitiveOverestimationSize;
    float              extraPrimitiveOverestimationSizeGranularity;
    VkBool32           primitiveUnderestimation;
    VkBool32           conservativePointAndLineRasterization;
    VkBool32           degenerateTrianglesRasterized;
    VkBool32           degenerateLinesRasterized;
    VkBool32           fullyCoveredFragmentShaderInputVariable;
    VkBool32           conservativeRasterizationPostDepthCoverage;
} VkPhysicalDeviceConservativeRasterizationPropertiesEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `primitiveOverestimationSize` is the size in pixels the generating primitive is increased at each of its edges during conservative rasterization overestimation mode. Even with a size of 0.0, conservative rasterization overestimation rules still apply and if any part of the pixel rectangle is covered by the generating primitive, fragments are generated for the entire pixel. However implementations **may** make the pixel coverage area even more conservative by increasing the size of the generating primitive.
- `maxExtraPrimitiveOverestimationSize` is the maximum size in pixels of extra overestimation the implementation supports in the pipeline state. A value of 0.0 means the implementation does not support any additional overestimation of the generating primitive during conservative rasterization. A value above 0.0 allows the application to further increase the size of the generating primitive during conservative rasterization overestimation.
- `extraPrimitiveOverestimationSizeGranularity` is the granularity of extra overestimation that can be specified in the pipeline state between 0.0 and `maxExtraPrimitiveOverestimationSize` inclusive. A value of 0.0 means the implementation can use the smallest representable non-zero value in the screen space pixel fixed-point grid.
- `primitiveUnderestimation` is `VK_TRUE` if the implementation supports the `VK_CONSERVATIVE_RASTERIZATION_MODE_UNDERESTIMATE_EXT` conservative rasterization mode in addition to `VK_CONSERVATIVE_RASTERIZATION_MODE_OVERESTIMATE_EXT`. Otherwise the implementation only supports `VK_CONSERVATIVE_RASTERIZATION_MODE_OVERESTIMATE_EXT`.
- `conservativePointAndLineRasterization` is `VK_TRUE` if the implementation supports conservative

rasterization of point and line primitives as well as triangle primitives. Otherwise the implementation only supports triangle primitives.

- degenerateTrianglesRasterized is VK_FALSE if the implementation culls primitives generated from triangles that become zero area after they are quantized to the fixed-point rasterization pixel grid. degenerateTrianglesRasterized is VK_TRUE if these primitives are not culled and the provoking vertex attributes and depth value are used for the fragments. The primitive area calculation is done on the primitive generated from the clipped triangle if applicable. Zero area primitives are backfacing and the application **can** enable backface culling if desired.
- degenerateLinesRasterized is VK_FALSE if the implementation culls lines that become zero length after they are quantized to the fixed-point rasterization pixel grid. degenerateLinesRasterized is VK_TRUE if zero length lines are not culled and the provoking vertex attributes and depth value are used for the fragments.
- fullyCoveredFragmentShaderInputVariable is VK_TRUE if the implementation supports the SPIR-V builtin fragment shader input variable FullyCoveredEXT specifying that conservative rasterization is enabled and the fragment area is fully covered by the generating primitive.
- conservativeRasterizationPostDepthCoverage is VK_TRUE if the implementation supports conservative rasterization with the PostDepthCoverage execution mode enabled. Otherwise the PostDepthCoverage execution mode **must** not be used when conservative rasterization is enabled.

If the VkPhysicalDeviceConservativeRasterizationPropertiesEXT structure is included in the pNext chain of the VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with each corresponding implementation-dependent property.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceConservativeRasterizationPropertiesEXT-sType-sType sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_CONSERVATIVE_RASTERIZATION_PROPERTIES_EXT

The VkPhysicalDeviceFragmentDensityMapPropertiesEXT structure is defined as:

```
// Provided by VK_EXT_fragment_density_map
typedef struct VkPhysicalDeviceFragmentDensityMapPropertiesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkExtent2D         minFragmentDensityTexelSize;
    VkExtent2D         maxFragmentDensityTexelSize;
    VkBool32           fragmentDensityInvocations;
} VkPhysicalDeviceFragmentDensityMapPropertiesEXT;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- minFragmentDensityTexelSize is the minimum fragment density texel size.

- maxFragmentDensityTexelSize is the maximum fragment density texel size.
- fragmentDensityInvocations specifies whether the implementation **may** invoke additional fragment shader invocations for each covered sample.

If the VkPhysicalDeviceFragmentDensityMapPropertiesEXT structure is included in the pNext chain of the VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with each corresponding implementation-dependent property.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceFragmentDensityMapPropertiesEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_FRAGMENT_DENSITY_MAP_PROPERTIES_EXT

The VkPhysicalDeviceFragmentDensityMap2PropertiesEXT structure is defined as:

```
// Provided by VK_EXT_fragment_density_map2
typedef struct VkPhysicalDeviceFragmentDensityMap2PropertiesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           subsampledLoads;
    VkBool32           subsampledCoarseReconstructionEarlyAccess;
    uint32_t           maxSubsampledArrayLayers;
    uint32_t           maxDescriptorSetSubsampledSamplers;
} VkPhysicalDeviceFragmentDensityMap2PropertiesEXT;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- subsampledLoads specifies if performing image data read with load operations on subsampled attachments will be resampled to the fragment density of the render pass
- subsampledCoarseReconstructionEarlyAccess specifies if performing image data read with samplers created with flags containing VK_SAMPLER_CREATE_SUBSAMPLED_COARSE_RECONSTRUCTION_BIT_EXT in fragment shader will trigger additional reads during VK_PIPELINE_STAGE_VERTEX_SHADER_BIT
- maxSubsampledArrayLayers is the maximum number of VkImageView array layers for usages supporting subsampled samplers
- maxDescriptorSetSubsampledSamplers is the maximum number of subsampled samplers that **can** be included in a VkPipelineLayout

If the VkPhysicalDeviceFragmentDensityMap2PropertiesEXT structure is included in the pNext chain of the VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with each corresponding implementation-dependent property.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceFragmentDensityMap2PropertiesEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_FRAGMENT_DENSITY_MAP_2_PROPERTIES_EXT`

The `VkPhysicalDeviceFragmentDensityMapOffsetPropertiesQCOM` structure is defined as:

```
// Provided by VK_QCOM_fragment_density_map_offset
typedef struct VkPhysicalDeviceFragmentDensityMapOffsetPropertiesQCOM {
    VkStructureType    sType;
    void*              pNext;
    VkExtent2D         fragmentDensityOffsetGranularity;
} VkPhysicalDeviceFragmentDensityMapOffsetPropertiesQCOM;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `fragmentDensityOffsetGranularity` is the granularity for fragment density offsets.

If the `VkPhysicalDeviceFragmentDensityMapOffsetPropertiesQCOM` structure is included in the `pNext` chain of the VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with each corresponding implementation-dependent property.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceFragmentDensityMapOffsetPropertiesQCOM-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_FRAGMENT_DENSITY_MAP_OFFSET_PROPERTIES_QCOM`

The `VkPhysicalDeviceShaderCorePropertiesAMD` structure is defined as:

```
// Provided by VK_AMD_shader_core_properties
typedef struct VkPhysicalDeviceShaderCorePropertiesAMD {
    VkStructureType    sType;
    void*              pNext;
    uint32_t           shaderEngineCount;
    uint32_t           shaderArraysPerEngineCount;
    uint32_t           computeUnitsPerShaderArray;
    uint32_t           simdPerComputeUnit;
    uint32_t           wavefrontsPerSimd;
    uint32_t           wavefrontSize;
    uint32_t           sgprsPerSimd;
    uint32_t           minSgprAllocation;
    uint32_t           maxSgprAllocation;
    uint32_t           sgprAllocationGranularity;
    uint32_t           vgprsPerSimd;
    uint32_t           minVgprAllocation;
    uint32_t           maxVgprAllocation;
    uint32_t           vgprAllocationGranularity;
} VkPhysicalDeviceShaderCorePropertiesAMD;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `shaderEngineCount` is an unsigned integer value indicating the number of shader engines found inside the shader core of the physical device.
- `shaderArraysPerEngineCount` is an unsigned integer value indicating the number of shader arrays inside a shader engine. Each shader array has its own scan converter, set of compute units, and a render back end (color and depth attachments). Shader arrays within a shader engine share shader processor input (wave launcher) and shader export (export buffer) units. Currently, a shader engine can have one or two shader arrays.
- `computeUnitsPerShaderArray` is an unsigned integer value indicating the physical number of compute units within a shader array. The active number of compute units in a shader array **may** be lower. A compute unit houses a set of SIMDs along with a sequencer module and a local data store.
- `simdPerComputeUnit` is an unsigned integer value indicating the number of SIMDs inside a compute unit. Each SIMD processes a single instruction at a time.
- `wavefrontSize` is an unsigned integer value indicating the maximum size of a subgroup.
- `sgprsPerSimd` is an unsigned integer value indicating the number of physical Scalar General-Purpose Registers (SGPRs) per SIMD.
- `minSgprAllocation` is an unsigned integer value indicating the minimum number of SGPRs allocated for a wave.
- `maxSgprAllocation` is an unsigned integer value indicating the maximum number of SGPRs allocated for a wave.
- `sgprAllocationGranularity` is an unsigned integer value indicating the granularity of SGPR allocation for a wave.

- vgprsPerSimd is an unsigned integer value indicating the number of physical Vector General-Purpose Registers (VGPRs) per SIMD.
- minVgprAllocation is an unsigned integer value indicating the minimum number of VGPRs allocated for a wave.
- maxVgprAllocation is an unsigned integer value indicating the maximum number of VGPRs allocated for a wave.
- vgprAllocationGranularity is an unsigned integer value indicating the granularity of VGPR allocation for a wave.

If the VkPhysicalDeviceShaderCorePropertiesAMD structure is included in the pNext chain of the VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with each corresponding implementation-dependent property.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceShaderCorePropertiesAMD-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_CORE_PROPERTIES_AMD

The VkPhysicalDeviceShaderCoreProperties2AMD structure is defined as:

```
// Provided by VK_AMD_shader_core_properties2
typedef struct VkPhysicalDeviceShaderCoreProperties2AMD {
    VkStructureType                    sType;
    void*                              pNext;
    VkShaderCorePropertiesFlagsAMD     shaderCoreFeatures;
    uint32_t                           activeComputeUnitCount;
} VkPhysicalDeviceShaderCoreProperties2AMD;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- shaderCoreFeatures is a bitmask of VkShaderCorePropertiesFlagBitsAMD indicating the set of features supported by the shader core.
- activeComputeUnitCount is an unsigned integer value indicating the number of compute units that have been enabled.

If the VkPhysicalDeviceShaderCoreProperties2AMD structure is included in the pNext chain of the VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with each corresponding implementation-dependent property.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceShaderCoreProperties2AMD-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_CORE_PROPERTIES_2_AMD

Bits for this type **may** be defined by future extensions, or new versions of the VK_AMD_shader_core_properties2 extension. Possible values of the flags member of VkShaderCorePropertiesFlagsAMD are:

```
// Provided by VK_AMD_shader_core_properties2
typedef enum VkShaderCorePropertiesFlagBitsAMD {
} VkShaderCorePropertiesFlagBitsAMD;
```

```
// Provided by VK_AMD_shader_core_properties2
typedef VkFlags VkShaderCorePropertiesFlagsAMD;
```

VkShaderCorePropertiesFlagsAMD is a bitmask type for providing zero or more VkShaderCorePropertiesFlagBitsAMD.

The VkPhysicalDeviceDepthStencilResolveProperties structure is defined as:

```
typedef struct VkPhysicalDeviceDepthStencilResolveProperties {
    VkStructureType       sType;
    void*                 pNext;
    VkResolveModeFlags    supportedDepthResolveModes;
    VkResolveModeFlags    supportedStencilResolveModes;
    VkBool32              independentResolveNone;
    VkBool32              independentResolve;
} VkPhysicalDeviceDepthStencilResolveProperties;
```

or the equivalent

```
// Provided by VK_KHR_depth_stencil_resolve
typedef VkPhysicalDeviceDepthStencilResolveProperties
VkPhysicalDeviceDepthStencilResolvePropertiesKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- supportedDepthResolveModes is a bitmask of VkResolveModeFlagBits indicating the set of supported depth resolve modes. VK_RESOLVE_MODE_SAMPLE_ZERO_BIT **must** be included in the set but implementations **may** support additional modes.
- supportedStencilResolveModes is a bitmask of VkResolveModeFlagBits indicating the set of supported stencil resolve modes. VK_RESOLVE_MODE_SAMPLE_ZERO_BIT **must** be included in the set but implementations **may** support additional modes. VK_RESOLVE_MODE_AVERAGE_BIT **must** not be included in the set.
- independentResolveNone is VK_TRUE if the implementation supports setting the depth and stencil resolve modes to different values when one of those modes is VK_RESOLVE_MODE_NONE. Otherwise the implementation only supports setting both modes to the same value.

- independentResolve is VK_TRUE if the implementation supports all combinations of the supported depth and stencil resolve modes, including setting either depth or stencil resolve mode to VK_RESOLVE_MODE_NONE. An implementation that supports independentResolve **must** also support independentResolveNone.

If the VkPhysicalDeviceDepthStencilResolveProperties structure is included in the pNext chain of the VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with each corresponding implementation-dependent property.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceDepthStencilResolveProperties-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DEPTH_STENCIL_RESOLVE_PROPERTIES

The VkPhysicalDevicePerformanceQueryPropertiesKHR structure is defined as:

```
// Provided by VK_KHR_performance_query
typedef struct VkPhysicalDevicePerformanceQueryPropertiesKHR {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           allowCommandBufferQueryCopies;
} VkPhysicalDevicePerformanceQueryPropertiesKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- allowCommandBufferQueryCopies is VK_TRUE if the performance query pools are allowed to be used with vkCmdCopyQueryPoolResults.

If the VkPhysicalDevicePerformanceQueryPropertiesKHR structure is included in the pNext chain of the VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with each corresponding implementation-dependent property.

### Valid Usage (Implicit)

- VUID-VkPhysicalDevicePerformanceQueryPropertiesKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PERFORMANCE_QUERY_PROPERTIES_KHR

The VkPhysicalDeviceShadingRateImagePropertiesNV structure is defined as:

```
// Provided by VK_NV_shading_rate_image
typedef struct VkPhysicalDeviceShadingRateImagePropertiesNV {
    VkStructureType    sType;
    void*              pNext;
    VkExtent2D         shadingRateTexelSize;
    uint32_t           shadingRatePaletteSize;
    uint32_t           shadingRateMaxCoarseSamples;
} VkPhysicalDeviceShadingRateImagePropertiesNV;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- shadingRateTexelSize indicates the width and height of the portion of the framebuffer corresponding to each texel in the shading rate image.
- shadingRatePaletteSize indicates the maximum number of palette entries supported for the shading rate image.
- shadingRateMaxCoarseSamples specifies the maximum number of coverage samples supported in a single fragment. If the product of the fragment size derived from the base shading rate and the number of coverage samples per pixel exceeds this limit, the final shading rate will be adjusted so that its product does not exceed the limit.

If the VkPhysicalDeviceShadingRateImagePropertiesNV structure is included in the pNext chain of the VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with each corresponding implementation-dependent property.

These properties are related to the shading rate image feature.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceShadingRateImagePropertiesNV-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADING_RATE_IMAGE_PROPERTIES_NV

The VkPhysicalDeviceMemoryDecompressionPropertiesNV structure is defined as:

```
// Provided by VK_NV_memory_decompression
typedef struct VkPhysicalDeviceMemoryDecompressionPropertiesNV {
    VkStructureType                       sType;
    void*                                 pNext;
    VkMemoryDecompressionMethodFlagsNV    decompressionMethods;
    uint64_t                              maxDecompressionIndirectCount;
} VkPhysicalDeviceMemoryDecompressionPropertiesNV;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- decompressionMethods is a bitmask of VkMemoryDecompressionMethodFlagBitsNV specifying

memory decompression methods supported by the implementation.

- maxDecompressionIndirectCount specifies the maximum supported count value in the countBuffer of vkCmdDecompressMemoryIndirectCountNV

If the VkPhysicalDeviceMemoryDecompressionPropertiesNV structure is included in the pNext chain of the VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with each corresponding implementation-dependent property.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceMemoryDecompressionPropertiesNV-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MEMORY_DECOMPRESSION_PROPERTIES_NV

The VkPhysicalDeviceTransformFeedbackPropertiesEXT structure is defined as:

```
// Provided by VK_EXT_transform_feedback
typedef struct VkPhysicalDeviceTransformFeedbackPropertiesEXT {
    VkStructureType    sType;
    void*              pNext;
    uint32_t           maxTransformFeedbackStreams;
    uint32_t           maxTransformFeedbackBuffers;
    VkDeviceSize       maxTransformFeedbackBufferSize;
    uint32_t           maxTransformFeedbackStreamDataSize;
    uint32_t           maxTransformFeedbackBufferDataSize;
    uint32_t           maxTransformFeedbackBufferDataStride;
    VkBool32           transformFeedbackQueries;
    VkBool32           transformFeedbackStreamsLinesTriangles;
    VkBool32           transformFeedbackRasterizationStreamSelect;
    VkBool32           transformFeedbackDraw;
} VkPhysicalDeviceTransformFeedbackPropertiesEXT;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- maxTransformFeedbackStreams is the maximum number of vertex streams that can be output from geometry shaders declared with the GeometryStreams capability. If the implementation does not support VkPhysicalDeviceTransformFeedbackFeaturesEXT::geometryStreams then maxTransformFeedbackStreams **must** be set to 1.
- maxTransformFeedbackBuffers is the maximum number of transform feedback buffers that can be bound for capturing shader outputs from the last pre-rasterization shader stage.
- maxTransformFeedbackBufferSize is the maximum size that can be specified when binding a buffer for transform feedback in vkCmdBindTransformFeedbackBuffersEXT.
- maxTransformFeedbackStreamDataSize is the maximum amount of data in bytes for each vertex that captured to one or more transform feedback buffers associated with a specific vertex stream.

- maxTransformFeedbackBufferDataSize is the maximum amount of data in bytes for each vertex that can be captured to a specific transform feedback buffer.
- maxTransformFeedbackBufferDataStride is the maximum stride between each capture of vertex data to the buffer.
- transformFeedbackQueries is VK_TRUE if the implementation supports the VK_QUERY_TYPE_TRANSFORM_FEEDBACK_STREAM_EXT query type. transformFeedbackQueries is VK_FALSE if queries of this type **cannot** be created.
- transformFeedbackStreamsLinesTriangles is VK_TRUE if the implementation supports the geometry shader OpExecutionMode of OutputLineStrip and OutputTriangleStrip in addition to OutputPoints when more than one vertex stream is output. If transformFeedbackStreamsLinesTriangles is VK_FALSE the implementation only supports an OpExecutionMode of OutputPoints when more than one vertex stream is output from the geometry shader.
- transformFeedbackRasterizationStreamSelect is VK_TRUE if the implementation supports the GeometryStreams SPIR-V capability and the application can use VkPipelineRasterizationStateStreamCreateInfoEXT to modify which vertex stream output is used for rasterization. Otherwise vertex stream 0 **must** always be used for rasterization.
- transformFeedbackDraw is VK_TRUE if the implementation supports the vkCmdDrawIndirectByteCountEXT function otherwise the function **must** not be called.

If the VkPhysicalDeviceTransformFeedbackPropertiesEXT structure is included in the pNext chain of the VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with each corresponding implementation-dependent property.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceTransformFeedbackPropertiesEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_TRANSFORM_FEEDBACK_PROPERTIES_EXT

The VkPhysicalDeviceCopyMemoryIndirectPropertiesNV structure is defined as:

```
// Provided by VK_NV_copy_memory_indirect
typedef struct VkPhysicalDeviceCopyMemoryIndirectPropertiesNV {
    VkStructureType    sType;
    void*              pNext;
    VkQueueFlags       supportedQueues;
} VkPhysicalDeviceCopyMemoryIndirectPropertiesNV;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- supportedQueues is a bitmask of VkQueueFlagBits indicating the queues on which indirect copy commands are supported.

If the indirectCopy feature is supported, supportedQueues **must** return at least one supported queue.

If the VkPhysicalDeviceCopyMemoryIndirectPropertiesNV structure is included in the pNext chain of the VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with each corresponding implementation-dependent property.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceCopyMemoryIndirectPropertiesNV-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_COPY_MEMORY_INDIRECT_PROPERTIES_NV

The VkPhysicalDeviceRayTracingPropertiesNV structure is defined as:

```
// Provided by VK_NV_ray_tracing
typedef struct VkPhysicalDeviceRayTracingPropertiesNV {
    VkStructureType    sType;
    void*              pNext;
    uint32_t           shaderGroupHandleSize;
    uint32_t           maxRecursionDepth;
    uint32_t           maxShaderGroupStride;
    uint32_t           shaderGroupBaseAlignment;
    uint64_t           maxGeometryCount;
    uint64_t           maxInstanceCount;
    uint64_t           maxTriangleCount;
    uint32_t           maxDescriptorSetAccelerationStructures;
} VkPhysicalDeviceRayTracingPropertiesNV;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- shaderGroupHandleSize is the size in bytes of the shader header.
- maxRecursionDepth is the maximum number of levels of recursion allowed in a trace command.
- maxShaderGroupStride is the maximum stride in bytes allowed between shader groups in the shader binding table.
- shaderGroupBaseAlignment is the **required** alignment in bytes for the base of the shader binding table.
- maxGeometryCount is the maximum number of geometries in the bottom level acceleration structure.
- maxInstanceCount is the maximum number of instances in the top level acceleration structure.
- maxTriangleCount is the maximum number of triangles in all geometries in the bottom level acceleration structure.
- maxDescriptorSetAccelerationStructures is the maximum number of acceleration structure descriptors that are allowed in a descriptor set.

Due to the fact that the geometry, instance, and triangle counts are specified at acceleration structure creation as 32-bit values, maxGeometryCount, maxInstanceCount, and maxTriangleCount **must**

not exceed $2^{32}$-1.

If the VkPhysicalDeviceRayTracingPropertiesNV structure is included in the pNext chain of the VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with each corresponding implementation-dependent property.

Limits specified by this structure **must** match those specified with the same name in VkPhysicalDeviceAccelerationStructurePropertiesKHR and VkPhysicalDeviceRayTracingPipelinePropertiesKHR.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceRayTracingPropertiesNV-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_RAY_TRACING_PROPERTIES_NV

The VkPhysicalDeviceAccelerationStructurePropertiesKHR structure is defined as:

```
// Provided by VK_KHR_acceleration_structure
typedef struct VkPhysicalDeviceAccelerationStructurePropertiesKHR {
    VkStructureType    sType;
    void*              pNext;
    uint64_t           maxGeometryCount;
    uint64_t           maxInstanceCount;
    uint64_t           maxPrimitiveCount;
    uint32_t           maxPerStageDescriptorAccelerationStructures;
    uint32_t           maxPerStageDescriptorUpdateAfterBindAccelerationStructures;
    uint32_t           maxDescriptorSetAccelerationStructures;
    uint32_t           maxDescriptorSetUpdateAfterBindAccelerationStructures;
    uint32_t           minAccelerationStructureScratchOffsetAlignment;
} VkPhysicalDeviceAccelerationStructurePropertiesKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- maxGeometryCount is the maximum number of geometries in the bottom level acceleration structure.
- maxInstanceCount is the maximum number of instances in the top level acceleration structure.
- maxPrimitiveCount is the maximum number of triangles or AABBs in all geometries in the bottom level acceleration structure.
- maxPerStageDescriptorAccelerationStructures is the maximum number of acceleration structure bindings that **can** be accessible to a single shader stage in a pipeline layout. Descriptor bindings with a descriptor type of VK_DESCRIPTOR_TYPE_ACCELERATION_STRUCTURE_KHR count against this limit. Only descriptor bindings in descriptor set layouts created without the VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT bit set count against this limit.
- maxPerStageDescriptorUpdateAfterBindAccelerationStructures is similar to maxPerStageDescriptorAccelerationStructures but counts descriptor bindings from descriptor

sets created with or without the VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT bit set.

- maxDescriptorSetAccelerationStructures is the maximum number of acceleration structure descriptors that **can** be included in descriptor bindings in a pipeline layout across all pipeline shader stages and descriptor set numbers. Descriptor bindings with a descriptor type of VK_DESCRIPTOR_TYPE_ACCELERATION_STRUCTURE_KHR count against this limit. Only descriptor bindings in descriptor set layouts created without the VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT bit set count against this limit.
- maxDescriptorSetUpdateAfterBindAccelerationStructures is similar to maxDescriptorSetAccelerationStructures but counts descriptor bindings from descriptor sets created with or without the VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT bit set.
- minAccelerationStructureScratchOffsetAlignment is the minimum **required** alignment, in bytes, for scratch data passed in to an acceleration structure build command. The value **must** be a power of two.

Due to the fact that the geometry, instance, and primitive counts are specified at acceleration structure creation as 32-bit values, maxGeometryCount, maxInstanceCount, and maxPrimitiveCount **must** not exceed $2^{32}$-1.

If the VkPhysicalDeviceAccelerationStructurePropertiesKHR structure is included in the pNext chain of the VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with each corresponding implementation-dependent property.

Limits specified by this structure **must** match those specified with the same name in VkPhysicalDeviceRayTracingPropertiesNV.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceAccelerationStructurePropertiesKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_ACCELERATION_STRUCTURE_PROPERTIES_KHR

The VkPhysicalDeviceRayTracingPipelinePropertiesKHR structure is defined as:

```
// Provided by VK_KHR_ray_tracing_pipeline
typedef struct VkPhysicalDeviceRayTracingPipelinePropertiesKHR {
    VkStructureType    sType;
    void*              pNext;
    uint32_t           shaderGroupHandleSize;
    uint32_t           maxRayRecursionDepth;
    uint32_t           maxShaderGroupStride;
    uint32_t           shaderGroupBaseAlignment;
    uint32_t           shaderGroupHandleCaptureReplaySize;
    uint32_t           maxRayDispatchInvocationCount;
    uint32_t           shaderGroupHandleAlignment;
    uint32_t           maxRayHitAttributeSize;
} VkPhysicalDeviceRayTracingPipelinePropertiesKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- shaderGroupHandleSize is the size in bytes of the shader header.
- maxRayRecursionDepth is the maximum number of levels of ray recursion allowed in a trace command.
- maxShaderGroupStride is the maximum stride in bytes allowed between shader groups in the shader binding table.
- shaderGroupBaseAlignment is the **required** alignment in bytes for the base of the shader binding table.
- shaderGroupHandleCaptureReplaySize is the number of bytes for the information required to do capture and replay for shader group handles.
- maxRayDispatchInvocationCount is the maximum number of ray generation shader invocations which **may** be produced by a single vkCmdTraceRaysIndirectKHR or vkCmdTraceRaysKHR command.
- shaderGroupHandleAlignment is the **required** alignment in bytes for each entry in a shader binding table. The value **must** be a power of two.
- maxRayHitAttributeSize is the maximum size in bytes for a ray attribute structure

If the VkPhysicalDeviceRayTracingPipelinePropertiesKHR structure is included in the pNext chain of the VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with each corresponding implementation-dependent property.

Limits specified by this structure **must** match those specified with the same name in VkPhysicalDeviceRayTracingPropertiesNV.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceRayTracingPipelinePropertiesKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_RAY_TRACING_PIPELINE_PROPERTIES_KHR

The VkPhysicalDeviceCooperativeMatrixPropertiesNV structure is defined as:

```
// Provided by VK_NV_cooperative_matrix
typedef struct VkPhysicalDeviceCooperativeMatrixPropertiesNV {
    VkStructureType       sType;
    void*                 pNext;
    VkShaderStageFlags    cooperativeMatrixSupportedStages;
} VkPhysicalDeviceCooperativeMatrixPropertiesNV;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- cooperativeMatrixSupportedStages is a bitfield of VkShaderStageFlagBits describing the shader

stages that cooperative matrix instructions are supported in. cooperativeMatrixSupportedStages will have the VK_SHADER_STAGE_COMPUTE_BIT bit set if any of the physical device's queues support VK_QUEUE_COMPUTE_BIT.

If the VkPhysicalDeviceCooperativeMatrixPropertiesNV structure is included in the pNext chain of the VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with each corresponding implementation-dependent property.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceCooperativeMatrixPropertiesNV-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_COOPERATIVE_MATRIX_PROPERTIES_NV

The VkPhysicalDeviceCooperativeMatrixPropertiesKHR structure is defined as:

```
// Provided by VK_KHR_cooperative_matrix
typedef struct VkPhysicalDeviceCooperativeMatrixPropertiesKHR {
    VkStructureType       sType;
    void*                 pNext;
    VkShaderStageFlags    cooperativeMatrixSupportedStages;
} VkPhysicalDeviceCooperativeMatrixPropertiesKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- cooperativeMatrixSupportedStages is a bitfield of VkShaderStageFlagBits describing the shader stages that cooperative matrix instructions are supported in. cooperativeMatrixSupportedStages will have the VK_SHADER_STAGE_COMPUTE_BIT bit set if any of the physical device's queues support VK_QUEUE_COMPUTE_BIT.

cooperativeMatrixSupportedStages **must** not have any bits other than VK_SHADER_STAGE_COMPUTE_BIT set.

If the VkPhysicalDeviceCooperativeMatrixPropertiesKHR structure is included in the pNext chain of the VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with each corresponding implementation-dependent property.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceCooperativeMatrixPropertiesKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_COOPERATIVE_MATRIX_PROPERTIES_KHR

The VkPhysicalDeviceShaderSMBuiltinsPropertiesNV structure is defined as:

```
// Provided by VK_NV_shader_sm_builtins
typedef struct VkPhysicalDeviceShaderSMBuiltinsPropertiesNV {
    VkStructureType    sType;
    void*              pNext;
    uint32_t           shaderSMCount;
    uint32_t           shaderWarpsPerSM;
} VkPhysicalDeviceShaderSMBuiltinsPropertiesNV;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `shaderSMCount` is the number of SMs on the device.
- `shaderWarpsPerSM` is the maximum number of simultaneously executing warps on an SM.

If the `VkPhysicalDeviceShaderSMBuiltinsPropertiesNV` structure is included in the `pNext` chain of the VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with each corresponding implementation-dependent property.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceShaderSMBuiltinsPropertiesNV-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_SM_BUILTINS_PROPERTIES_NV`

The `VkPhysicalDeviceTexelBufferAlignmentProperties` structure is defined as:

```
typedef struct VkPhysicalDeviceTexelBufferAlignmentProperties {
    VkStructureType    sType;
    void*              pNext;
    VkDeviceSize       storageTexelBufferOffsetAlignmentBytes;
    VkBool32           storageTexelBufferOffsetSingleTexelAlignment;
    VkDeviceSize       uniformTexelBufferOffsetAlignmentBytes;
    VkBool32           uniformTexelBufferOffsetSingleTexelAlignment;
} VkPhysicalDeviceTexelBufferAlignmentProperties;
```

or the equivalent

```
// Provided by VK_EXT_texel_buffer_alignment
typedef VkPhysicalDeviceTexelBufferAlignmentProperties
VkPhysicalDeviceTexelBufferAlignmentPropertiesEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `storageTexelBufferOffsetAlignmentBytes` is a byte alignment that is sufficient for a storage texel buffer of any format. The value **must** be a power of two.

- storageTexelBufferOffsetSingleTexelAlignment indicates whether single texel alignment is sufficient for a storage texel buffer of any format.
- uniformTexelBufferOffsetAlignmentBytes is a byte alignment that is sufficient for a uniform texel buffer of any format. The value **must** be a power of two.
- uniformTexelBufferOffsetSingleTexelAlignment indicates whether single texel alignment is sufficient for a uniform texel buffer of any format.

If the VkPhysicalDeviceTexelBufferAlignmentProperties structure is included in the pNext chain of the VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with each corresponding implementation-dependent property.

If the single texel alignment property is VK_FALSE, then the buffer view's offset **must** be aligned to the corresponding byte alignment value. If the single texel alignment property is VK_TRUE, then the buffer view's offset **must** be aligned to the lesser of the corresponding byte alignment value or the size of a single texel, based on VkBufferViewCreateInfo::format. If the size of a single texel is a multiple of three bytes, then the size of a single component of the format is used instead.

These limits **must** not advertise a larger alignment than the required maximum minimum value of VkPhysicalDeviceLimits::minTexelBufferOffsetAlignment, for any format that supports use as a texel buffer.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceTexelBufferAlignmentProperties-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_TEXEL_BUFFER_ALIGNMENT_PROPERTIES

The VkPhysicalDeviceTimelineSemaphoreProperties structure is defined as:

```
typedef struct VkPhysicalDeviceTimelineSemaphoreProperties {
    VkStructureType    sType;
    void*              pNext;
    uint64_t           maxTimelineSemaphoreValueDifference;
} VkPhysicalDeviceTimelineSemaphoreProperties;
```

or the equivalent

```
// Provided by VK_KHR_timeline_semaphore
typedef VkPhysicalDeviceTimelineSemaphoreProperties
VkPhysicalDeviceTimelineSemaphorePropertiesKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- maxTimelineSemaphoreValueDifference indicates the maximum difference allowed by the implementation between the current value of a timeline semaphore and any pending signal or

wait operations.

If the VkPhysicalDeviceTimelineSemaphoreProperties structure is included in the pNext chain of the VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with each corresponding implementation-dependent property.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceTimelineSemaphoreProperties-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_TIMELINE_SEMAPHORE_PROPERTIES

The VkPhysicalDeviceLineRasterizationPropertiesKHR structure is defined as:

```
// Provided by VK_KHR_line_rasterization
typedef struct VkPhysicalDeviceLineRasterizationPropertiesKHR {
    VkStructureType    sType;
    void*              pNext;
    uint32_t           lineSubPixelPrecisionBits;
} VkPhysicalDeviceLineRasterizationPropertiesKHR;
```

or the equivalent

```
// Provided by VK_EXT_line_rasterization
typedef VkPhysicalDeviceLineRasterizationPropertiesKHR
VkPhysicalDeviceLineRasterizationPropertiesEXT;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- lineSubPixelPrecisionBits is the number of bits of subpixel precision in framebuffer coordinates $x_f$ and $y_f$ when rasterizing line segments.

If the VkPhysicalDeviceLineRasterizationPropertiesKHR structure is included in the pNext chain of the VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with each corresponding implementation-dependent property.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceLineRasterizationPropertiesKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_LINE_RASTERIZATION_PROPERTIES_KHR

The VkPhysicalDeviceRobustness2PropertiesEXT structure is defined as:

```
// Provided by VK_EXT_robustness2
typedef struct VkPhysicalDeviceRobustness2PropertiesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkDeviceSize       robustStorageBufferAccessSizeAlignment;
    VkDeviceSize       robustUniformBufferAccessSizeAlignment;
} VkPhysicalDeviceRobustness2PropertiesEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `robustStorageBufferAccessSizeAlignment` is the number of bytes that the range of a storage buffer descriptor is rounded up to when used for bounds-checking when the `robustBufferAccess2` feature is enabled. This value **must** be either 1 or 4.
- `robustUniformBufferAccessSizeAlignment` is the number of bytes that the range of a uniform buffer descriptor is rounded up to when used for bounds-checking when the `robustBufferAccess2` feature is enabled. This value **must** be a power of two in the range [1, 256].

If the `VkPhysicalDeviceRobustness2PropertiesEXT` structure is included in the `pNext` chain of the VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with each corresponding implementation-dependent property.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceRobustness2PropertiesEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_ROBUSTNESS_2_PROPERTIES_EXT`

The `VkPhysicalDeviceDeviceGeneratedCommandsPropertiesNV` structure is defined as:

```
// Provided by VK_NV_device_generated_commands
typedef struct VkPhysicalDeviceDeviceGeneratedCommandsPropertiesNV {
    VkStructureType    sType;
    void*              pNext;
    uint32_t           maxGraphicsShaderGroupCount;
    uint32_t           maxIndirectSequenceCount;
    uint32_t           maxIndirectCommandsTokenCount;
    uint32_t           maxIndirectCommandsStreamCount;
    uint32_t           maxIndirectCommandsTokenOffset;
    uint32_t           maxIndirectCommandsStreamStride;
    uint32_t           minSequencesCountBufferOffsetAlignment;
    uint32_t           minSequencesIndexBufferOffsetAlignment;
    uint32_t           minIndirectCommandsBufferOffsetAlignment;
} VkPhysicalDeviceDeviceGeneratedCommandsPropertiesNV;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.

- maxGraphicsShaderGroupCount is the maximum number of shader groups in VkGraphicsPipelineShaderGroupsCreateInfoNV.
- maxIndirectSequenceCount is the maximum number of sequences in VkGeneratedCommandsInfoNV and in VkGeneratedCommandsMemoryRequirementsInfoNV.
- maxIndirectCommandsTokenCount is the maximum number of tokens in VkIndirectCommandsLayoutCreateInfoNV.
- maxIndirectCommandsStreamCount is the maximum number of streams in VkIndirectCommandsLayoutCreateInfoNV.
- maxIndirectCommandsTokenOffset is the maximum offset in VkIndirectCommandsLayoutTokenNV.
- maxIndirectCommandsStreamStride is the maximum stream stride in VkIndirectCommandsLayoutCreateInfoNV.
- minSequencesCountBufferOffsetAlignment is the minimum alignment for memory addresses which **can** be used in VkGeneratedCommandsInfoNV.
- minSequencesIndexBufferOffsetAlignment is the minimum alignment for memory addresses which **can** be used in VkGeneratedCommandsInfoNV.
- minIndirectCommandsBufferOffsetAlignment is the minimum alignment for memory addresses used in VkIndirectCommandsStreamNV, and as preprocess buffer in VkGeneratedCommandsInfoNV.

If the VkPhysicalDeviceDeviceGeneratedCommandsPropertiesNV structure is included in the pNext chain of the VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with each corresponding implementation-dependent property.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceDeviceGeneratedCommandsPropertiesNV-sType-sType sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DEVICE_GENERATED_COMMANDS_PROPERTIES_NV

The VkPhysicalDevicePortabilitySubsetPropertiesKHR structure is defined as:

```
// Provided by VK_KHR_portability_subset
typedef struct VkPhysicalDevicePortabilitySubsetPropertiesKHR {
    VkStructureType    sType;
    void*              pNext;
    uint32_t           minVertexInputBindingStrideAlignment;
} VkPhysicalDevicePortabilitySubsetPropertiesKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- minVertexInputBindingStrideAlignment indicates the minimum alignment for vertex input strides. VkVertexInputBindingDescription::stride **must** be a multiple of, and at least as large as, this value. The value **must** be a power of two.

If the VkPhysicalDevicePortabilitySubsetPropertiesKHR structure is included in the pNext chain of the VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with each corresponding implementation-dependent property.

### Valid Usage (Implicit)

- VUID-VkPhysicalDevicePortabilitySubsetPropertiesKHR-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PORTABILITY_SUBSET_PROPERTIES_KHR

The VkPhysicalDeviceFragmentShadingRatePropertiesKHR structure is defined as:

```
// Provided by VK_KHR_fragment_shading_rate
typedef struct VkPhysicalDeviceFragmentShadingRatePropertiesKHR {
    VkStructureType          sType;
    void*                    pNext;
    VkExtent2D               minFragmentShadingRateAttachmentTexelSize;
    VkExtent2D               maxFragmentShadingRateAttachmentTexelSize;
    uint32_t                 maxFragmentShadingRateAttachmentTexelSizeAspectRatio;
    VkBool32                 primitiveFragmentShadingRateWithMultipleViewports;
    VkBool32                 layeredShadingRateAttachments;
    VkBool32                 fragmentShadingRateNonTrivialCombinerOps;
    VkExtent2D               maxFragmentSize;
    uint32_t                 maxFragmentSizeAspectRatio;
    uint32_t                 maxFragmentShadingRateCoverageSamples;
    VkSampleCountFlagBits    maxFragmentShadingRateRasterizationSamples;
    VkBool32                 fragmentShadingRateWithShaderDepthStencilWrites;
    VkBool32                 fragmentShadingRateWithSampleMask;
    VkBool32                 fragmentShadingRateWithShaderSampleMask;
    VkBool32                 fragmentShadingRateWithConservativeRasterization;
    VkBool32                 fragmentShadingRateWithFragmentShaderInterlock;
    VkBool32                 fragmentShadingRateWithCustomSampleLocations;
    VkBool32                 fragmentShadingRateStrictMultiplyCombiner;
} VkPhysicalDeviceFragmentShadingRatePropertiesKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- minFragmentShadingRateAttachmentTexelSize indicates minimum supported width and height of the portion of the framebuffer corresponding to each texel in a fragment shading rate attachment. Each value **must** be less than or equal to the values in maxFragmentShadingRateAttachmentTexelSize. Each value **must** be a power-of-two. It **must** be (0,0) if the attachmentFragmentShadingRate feature is not supported.
- maxFragmentShadingRateAttachmentTexelSize indicates maximum supported width and height of the portion of the framebuffer corresponding to each texel in a fragment shading rate attachment. Each value **must** be greater than or equal to the values in minFragmentShadingRateAttachmentTexelSize. Each value **must** be a power-of-two. It **must** be (0,0) if the attachmentFragmentShadingRate feature is not supported.

- maxFragmentShadingRateAttachmentTexelSizeAspectRatio indicates the maximum ratio between the width and height of the portion of the framebuffer corresponding to each texel in a fragment shading rate attachment. maxFragmentShadingRateAttachmentTexelSizeAspectRatio **must** be a power-of-two value, and **must** be less than or equal to max(maxFragmentShadingRateAttachmentTexelSize.width / minFragmentShadingRateAttachmentTexelSize.height, maxFragmentShadingRateAttachmentTexelSize.height / minFragmentShadingRateAttachmentTexelSize.width). It **must** be 0 if the attachmentFragmentShadingRate feature is not supported.
- primitiveFragmentShadingRateWithMultipleViewports specifies whether the primitive fragment shading rate **can** be used when multiple viewports are used. If this value is VK_FALSE, only a single viewport **must** be used, and applications **must** not write to the ViewportMaskNV or ViewportIndex built-in when setting PrimitiveShadingRateKHR. It **must** be VK_FALSE if the VK_EXT_shader_viewport_index_layer extension, or the geometryShader feature is not supported, or if the primitiveFragmentShadingRate feature is not supported.
- layeredShadingRateAttachments specifies whether a shading rate attachment image view **can** be created with multiple layers. If this value is VK_FALSE, when creating an image view with a usage that includes VK_IMAGE_USAGE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR, layerCount **must** be 1. It **must** be VK_FALSE if the multiview feature, the VK_EXT_shader_viewport_index_layer extension, or the geometryShader feature is not supported, or if the attachmentFragmentShadingRate feature is not supported.
- fragmentShadingRateNonTrivialCombinerOps specifies whether VkFragmentShadingRateCombinerOpKHR enums other than VK_FRAGMENT_SHADING_RATE_COMBINER_OP_KEEP_KHR or VK_FRAGMENT_SHADING_RATE_COMBINER_OP_REPLACE_KHR **can** be used. It **must** be VK_FALSE unless either the primitiveFragmentShadingRate or attachmentFragmentShadingRate feature is supported.
- maxFragmentSize indicates the maximum supported width and height of a fragment. Its width and height members **must** both be power-of-two values. This limit is purely informational, and is not validated.
- maxFragmentSizeAspectRatio indicates the maximum ratio between the width and height of a fragment. maxFragmentSizeAspectRatio **must** be a power-of-two value, and **must** be less than or equal to the maximum of the width and height members of maxFragmentSize. This limit is purely informational, and is not validated.
- maxFragmentShadingRateCoverageSamples specifies the maximum number of coverage samples supported in a single fragment. maxFragmentShadingRateCoverageSamples **must** be less than or equal to the product of the width and height members of maxFragmentSize, and the sample count reported by maxFragmentShadingRateRasterizationSamples. maxFragmentShadingRateCoverageSamples **must** be less than or equal to maxSampleMaskWords × 32 if fragmentShadingRateWithShaderSampleMask is supported. This limit is purely informational, and is not validated.
- maxFragmentShadingRateRasterizationSamples is a VkSampleCountFlagBits value specifying the maximum sample rate supported when a fragment covers multiple pixels. This limit is purely informational, and is not validated.
- fragmentShadingRateWithShaderDepthStencilWrites specifies whether the implementation supports writing FragDepth or FragStencilRefEXT from a fragment shader for multi-pixel

fragments. If this value is VK_FALSE, writing to those built-ins will clamp the fragment shading rate to (1,1).

- fragmentShadingRateWithSampleMask specifies whether the implementation supports setting valid bits of VkPipelineMultisampleStateCreateInfo::pSampleMask to 0 for multi-pixel fragments. If this value is VK_FALSE, zeroing valid bits in the sample mask will clamp the fragment shading rate to (1,1).
- fragmentShadingRateWithShaderSampleMask specifies whether the implementation supports reading or writing SampleMask for multi-pixel fragments. If this value is VK_FALSE, using that built-in will clamp the fragment shading rate to (1,1).
- fragmentShadingRateWithConservativeRasterization specifies whether conservative rasterization is supported for multi-pixel fragments. It **must** be VK_FALSE if VK_EXT_conservative_rasterization is not supported. If this value is VK_FALSE, using conservative rasterization will clamp the fragment shading rate to (1,1).
- fragmentShadingRateWithFragmentShaderInterlock specifies whether fragment shader interlock is supported for multi-pixel fragments. It **must** be VK_FALSE if VK_EXT_fragment_shader_interlock is not supported. If this value is VK_FALSE, using fragment shader interlock will clamp the fragment shading rate to (1,1).
- fragmentShadingRateWithCustomSampleLocations specifies whether custom sample locations are supported for multi-pixel fragments. It **must** be VK_FALSE if VK_EXT_sample_locations is not supported. If this value is VK_FALSE, using custom sample locations will clamp the fragment shading rate to (1,1).
- fragmentShadingRateStrictMultiplyCombiner specifies whether VK_FRAGMENT_SHADING_RATE_COMBINER_OP_MUL_KHR accurately performs a multiplication or not. Implementations where this value is VK_FALSE will instead combine rates with an addition. If fragmentShadingRateNonTrivialCombinerOps is VK_FALSE, implementations **must** report this as VK_FALSE. If fragmentShadingRateNonTrivialCombinerOps is VK_TRUE, implementations **should** report this as VK_TRUE.

**NOTE**

Multiplication of the combiner rates using the fragment width/height in linear space is equivalent to an addition of those values in log2 space. Some implementations inadvertently implemented an addition in linear space due to unclear requirements originating outside of this specification. This resulted in fragmentShadingRateStrictMultiplyCombiner being added. Fortunately, this only affects situations where a rate of 1 in either dimension is combined with another rate of 1. All other combinations result in the exact same result as if multiplication was performed in linear space due to the clamping logic, and the fact that both the sum and product of 2 and 2 are equal. In many cases, this limit will not affect the correct operation of applications.

If the VkPhysicalDeviceFragmentShadingRatePropertiesKHR structure is included in the pNext chain of the VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with each corresponding implementation-dependent property.

These properties are related to fragment shading rates.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceFragmentShadingRatePropertiesKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_FRAGMENT_SHADING_RATE_PROPERTIES_KHR`

The `VkPhysicalDeviceFragmentShadingRateEnumsPropertiesNV` structure is defined as:

```
// Provided by VK_NV_fragment_shading_rate_enums
typedef struct VkPhysicalDeviceFragmentShadingRateEnumsPropertiesNV {
    VkStructureType          sType;
    void*                    pNext;
    VkSampleCountFlagBits    maxFragmentShadingRateInvocationCount;
} VkPhysicalDeviceFragmentShadingRateEnumsPropertiesNV;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `maxFragmentShadingRateInvocationCount` is a VkSampleCountFlagBits value indicating the maximum number of fragment shader invocations per fragment supported in pipeline, primitive, and attachment fragment shading rates.

If the `VkPhysicalDeviceFragmentShadingRateEnumsPropertiesNV` structure is included in the `pNext` chain of the VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with each corresponding implementation-dependent property.

These properties are related to fragment shading rates.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceFragmentShadingRateEnumsPropertiesNV-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_FRAGMENT_SHADING_RATE_ENUMS_PROPERTIES_NV`
- VUID-VkPhysicalDeviceFragmentShadingRateEnumsPropertiesNV-maxFragmentShadingRateInvocationCount-parameter
  `maxFragmentShadingRateInvocationCount` **must** be a valid VkSampleCountFlagBits value

The `VkPhysicalDeviceCustomBorderColorPropertiesEXT` structure is defined as:

```
// Provided by VK_EXT_custom_border_color
typedef struct VkPhysicalDeviceCustomBorderColorPropertiesEXT {
    VkStructureType    sType;
    void*              pNext;
    uint32_t           maxCustomBorderColorSamplers;
} VkPhysicalDeviceCustomBorderColorPropertiesEXT;
```

- maxCustomBorderColorSamplers indicates the maximum number of samplers with custom border colors which **can** simultaneously exist on a device.

If the VkPhysicalDeviceCustomBorderColorPropertiesEXT structure is included in the pNext chain of the VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with each corresponding implementation-dependent property.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceCustomBorderColorPropertiesEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_CUSTOM_BORDER_COLOR_PROPERTIES_EXT

The VkPhysicalDeviceProvokingVertexPropertiesEXT structure is defined as:

```
// Provided by VK_EXT_provoking_vertex
typedef struct VkPhysicalDeviceProvokingVertexPropertiesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           provokingVertexModePerPipeline;
    VkBool32           transformFeedbackPreservesTriangleFanProvokingVertex;
} VkPhysicalDeviceProvokingVertexPropertiesEXT;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- provokingVertexModePerPipeline indicates whether the implementation supports graphics pipelines with different provoking vertex modes within the same render pass instance.
- transformFeedbackPreservesTriangleFanProvokingVertex indicates whether the implementation can preserve the provoking vertex order when writing triangle fan vertices to transform feedback.

If the VkPhysicalDeviceProvokingVertexPropertiesEXT structure is included in the pNext chain of the VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with each corresponding implementation-dependent property.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceProvokingVertexPropertiesEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PROVOKING_VERTEX_PROPERTIES_EXT

The VkPhysicalDeviceDescriptorBufferPropertiesEXT structure is defined as:

```
// Provided by VK_EXT_descriptor_buffer
typedef struct VkPhysicalDeviceDescriptorBufferPropertiesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           combinedImageSamplerDescriptorSingleArray;
    VkBool32           bufferlessPushDescriptors;
    VkBool32           allowSamplerImageViewPostSubmitCreation;
    VkDeviceSize       descriptorBufferOffsetAlignment;
    uint32_t           maxDescriptorBufferBindings;
    uint32_t           maxResourceDescriptorBufferBindings;
    uint32_t           maxSamplerDescriptorBufferBindings;
    uint32_t           maxEmbeddedImmutableSamplerBindings;
    uint32_t           maxEmbeddedImmutableSamplers;
    size_t             bufferCaptureReplayDescriptorDataSize;
    size_t             imageCaptureReplayDescriptorDataSize;
    size_t             imageViewCaptureReplayDescriptorDataSize;
    size_t             samplerCaptureReplayDescriptorDataSize;
    size_t             accelerationStructureCaptureReplayDescriptorDataSize;
    size_t             samplerDescriptorSize;
    size_t             combinedImageSamplerDescriptorSize;
    size_t             sampledImageDescriptorSize;
    size_t             storageImageDescriptorSize;
    size_t             uniformTexelBufferDescriptorSize;
    size_t             robustUniformTexelBufferDescriptorSize;
    size_t             storageTexelBufferDescriptorSize;
    size_t             robustStorageTexelBufferDescriptorSize;
    size_t             uniformBufferDescriptorSize;
    size_t             robustUniformBufferDescriptorSize;
    size_t             storageBufferDescriptorSize;
    size_t             robustStorageBufferDescriptorSize;
    size_t             inputAttachmentDescriptorSize;
    size_t             accelerationStructureDescriptorSize;
    VkDeviceSize       maxSamplerDescriptorBufferRange;
    VkDeviceSize       maxResourceDescriptorBufferRange;
    VkDeviceSize       samplerDescriptorBufferAddressSpaceSize;
    VkDeviceSize       resourceDescriptorBufferAddressSpaceSize;
    VkDeviceSize       descriptorBufferAddressSpaceSize;
} VkPhysicalDeviceDescriptorBufferPropertiesEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `combinedImageSamplerDescriptorSingleArray` indicates that the implementation does not require an array of `VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER` descriptors to be written into a descriptor buffer as an array of image descriptors, immediately followed by an array of sampler descriptors.
- `bufferlessPushDescriptors` indicates that the implementation does not require a buffer created with `VK_BUFFER_USAGE_PUSH_DESCRIPTORS_DESCRIPTOR_BUFFER_BIT_EXT` to be bound when using push descriptors.

- allowSamplerImageViewPostSubmitCreation indicates that the implementation does not restrict when the VkSampler or VkImageView objects used to retrieve descriptor data **can** be created in relation to command buffer submission. If this value is VK_FALSE, then the application **must** create any VkSampler or VkImageView objects whose descriptor data is accessed during the execution of a command buffer, before the vkQueueSubmit , or vkQueueSubmit2KHR, call that submits that command buffer.
- descriptorBufferOffsetAlignment indicates the **required** alignment in bytes when setting offsets into the descriptor buffer.
- maxDescriptorBufferBindings indicates the maximum sum total number of descriptor buffers and embedded immutable sampler sets that **can** be bound.
- maxResourceDescriptorBufferBindings indicates the maximum number of resource descriptor buffers that **can** be bound.
- maxSamplerDescriptorBufferBindings indicates the maximum number of sampler descriptor buffers that **can** be bound.
- maxEmbeddedImmutableSamplerBindings indicates the maximum number of embedded immutable sampler sets that **can** be bound.
- maxEmbeddedImmutableSamplers indicates the maximum number of unique immutable samplers in descriptor set layouts created with VK_DESCRIPTOR_SET_LAYOUT_CREATE_EMBEDDED_IMMUTABLE_SAMPLERS_BIT_EXT, and pipeline layouts created from them, which **can** simultaneously exist on a device.
- bufferCaptureReplayDescriptorDataSize indicates the maximum size in bytes of the opaque data used for capture and replay with buffers.
- imageCaptureReplayDescriptorDataSize indicates the maximum size in bytes of the opaque data used for capture and replay with images.
- imageViewCaptureReplayDescriptorDataSize indicates the maximum size in bytes of the opaque data used for capture and replay with image views.
- samplerCaptureReplayDescriptorDataSize indicates the maximum size in bytes of the opaque data used for capture and replay with samplers.
- accelerationStructureCaptureReplayDescriptorDataSize indicates the maximum size in bytes of the opaque data used for capture and replay with acceleration structures.
- samplerDescriptorSize indicates the size in bytes of a VK_DESCRIPTOR_TYPE_SAMPLER descriptor.
- combinedImageSamplerDescriptorSize indicates the size in bytes of a VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER descriptor.
- sampledImageDescriptorSize indicates the size in bytes of a VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE descriptor.
- storageImageDescriptorSize indicates the size in bytes of a VK_DESCRIPTOR_TYPE_STORAGE_IMAGE descriptor.
- uniformTexelBufferDescriptorSize indicates the size in bytes of a VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER descriptor if the robustBufferAccess feature is not enabled.
- robustUniformTexelBufferDescriptorSize indicates the size in bytes of a

VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER descriptor if the robustBufferAccess feature is enabled.

- storageTexelBufferDescriptorSize indicates the size in bytes of a VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER descriptor if the robustBufferAccess feature is not enabled.
- robustStorageTexelBufferDescriptorSize indicates the size in bytes of a VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER descriptor if the robustBufferAccess feature is enabled.
- uniformBufferDescriptorSize indicates the size in bytes of a VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER descriptor.
- robustUniformBufferDescriptorSize indicates the size in bytes of a VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER descriptor if the robustBufferAccess feature is enabled.
- storageBufferDescriptorSize indicates the size in bytes of a VK_DESCRIPTOR_TYPE_STORAGE_BUFFER descriptor.
- robustStorageBufferDescriptorSize indicates the size in bytes of a VK_DESCRIPTOR_TYPE_STORAGE_BUFFER descriptor if the robustBufferAccess feature is enabled.
- inputAttachmentDescriptorSize indicates the size in bytes of a VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT descriptor.
- accelerationStructureDescriptorSize indicates the size in bytes of a VK_DESCRIPTOR_TYPE_ACCELERATION_STRUCTURE_KHR or VK_DESCRIPTOR_TYPE_ACCELERATION_STRUCTURE_NV descriptor.
- maxSamplerDescriptorBufferRange indicates the maximum range in bytes from the address of a sampler descriptor buffer binding that is accessible to a shader.
- maxResourceDescriptorBufferRange indicates the maximum range in bytes from the address of a resource descriptor buffer binding that is accessible to a shader.
- samplerDescriptorBufferAddressSpaceSize indicates the total size in bytes of the address space available for descriptor buffers created with VK_BUFFER_USAGE_SAMPLER_DESCRIPTOR_BUFFER_BIT_EXT.
- resourceDescriptorBufferAddressSpaceSize indicates the total size in bytes of the address space available for descriptor buffers created with VK_BUFFER_USAGE_RESOURCE_DESCRIPTOR_BUFFER_BIT_EXT.
- descriptorBufferAddressSpaceSize indicates the total size in bytes of the address space available for descriptor buffers created with both VK_BUFFER_USAGE_SAMPLER_DESCRIPTOR_BUFFER_BIT_EXT and VK_BUFFER_USAGE_RESOURCE_DESCRIPTOR_BUFFER_BIT_EXT.

A descriptor binding with type VK_DESCRIPTOR_TYPE_MUTABLE_VALVE has a descriptor size which is implied by the descriptor types included in the VkMutableDescriptorTypeCreateInfoVALVE::pDescriptorTypes list. The descriptor size is equal to the maximum size of any descriptor type included in the pDescriptorTypes list.

As there is no way to request robust and non-robust descriptors separately, or specify robust/non-robust descriptors in the set layout, if robustBufferAccess is enabled then robust descriptors are always used.

If the VkPhysicalDeviceDescriptorBufferPropertiesEXT structure is included in the pNext chain of the

VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with each corresponding implementation-dependent property.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceDescriptorBufferPropertiesEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DESCRIPTOR_BUFFER_PROPERTIES_EXT

The VkPhysicalDeviceDescriptorBufferDensityMapPropertiesEXT structure is defined as:

```
// Provided by VK_EXT_descriptor_buffer
typedef struct VkPhysicalDeviceDescriptorBufferDensityMapPropertiesEXT {
    VkStructureType    sType;
    void*              pNext;
    size_t             combinedImageSamplerDensityMapDescriptorSize;
} VkPhysicalDeviceDescriptorBufferDensityMapPropertiesEXT;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- combinedImageSamplerDensityMapDescriptorSize indicates the size in bytes of a VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER descriptor when creating the descriptor with VK_SAMPLER_CREATE_SUBSAMPLED_BIT_EXT set.

If the VkPhysicalDeviceDescriptorBufferDensityMapPropertiesEXT structure is included in the pNext chain of the VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with each corresponding implementation-dependent property.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceDescriptorBufferDensityMapPropertiesEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DESCRIPTOR_BUFFER_DENSITY_MAP_PROPERTIES_EXT

The VkPhysicalDeviceHostImageCopyPropertiesEXT structure is defined as:

```
// Provided by VK_EXT_host_image_copy
typedef struct VkPhysicalDeviceHostImageCopyPropertiesEXT {
    VkStructureType    sType;
    void*              pNext;
    uint32_t           copySrcLayoutCount;
    VkImageLayout*     pCopySrcLayouts;
    uint32_t           copyDstLayoutCount;
    VkImageLayout*     pCopyDstLayouts;
    uint8_t            optimalTilingLayoutUUID[VK_UUID_SIZE];
    VkBool32           identicalMemoryTypeRequirements;
} VkPhysicalDeviceHostImageCopyPropertiesEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `copySrcLayoutCount` is an integer related to the number of image layouts for host copies from images available or queried, as described below.
- `pCopySrcLayouts` is a pointer to an array of VkImageLayout in which supported image layouts for use with host copy operations from images are returned.
- `copyDstLayoutCount` is an integer related to the number of image layouts for host copies to images available or queried, as described below.
- `pCopyDstLayouts` is a pointer to an array of VkImageLayout in which supported image layouts for use with host copy operations to images are returned.
- `optimalTilingLayoutUUID` is an array of `VK_UUID_SIZE` `uint8_t` values representing a universally unique identifier for the implementation's swizzling layout of images created with `VK_IMAGE_TILING_OPTIMAL`.
- `identicalMemoryTypeRequirements` indicates that specifying the `VK_IMAGE_USAGE_HOST_TRANSFER_BIT_EXT` flag in VkImageCreateInfo::`usage` does not affect the memory type requirements of the image.

If the `VkPhysicalDeviceHostImageCopyPropertiesEXT` structure is included in the `pNext` chain of the VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with each corresponding implementation-dependent property.

If `pCopyDstLayouts` is `NULL`, then the number of image layouts that are supported in VkCopyMemoryToImageInfoEXT::`dstImageLayout` and VkCopyImageToImageInfoEXT::`dstImageLayout` is returned in `copyDstLayoutCount`. Otherwise, `copyDstLayoutCount` **must** be set by the application to the number of elements in the `pCopyDstLayouts` array, and on return the variable is overwritten with the number of values actually written to `pCopyDstLayouts`. If the value of `copyDstLayoutCount` is less than the number of image layouts that are supported, at most `copyDstLayoutCount` values will be written to `pCopyDstLayouts`. The implementation **must** include the `VK_IMAGE_LAYOUT_GENERAL` image layout in `pCopyDstLayouts`.

If `pCopySrcLayouts` is `NULL`, then the number of image layouts that are supported in VkCopyImageToMemoryInfoEXT::`srcImageLayout` and VkCopyImageToImageInfoEXT::`srcImageLayout` is returned in `copySrcLayoutCount`. Otherwise, `copySrcLayoutCount` **must** be set by

the application to the number of elements in the pCopySrcLayouts array, and on return the variable is overwritten with the number of values actually written to pCopySrcLayouts. If the value of copySrcLayoutCount is less than the number of image layouts that are supported, at most copySrcLayoutCount values will be written to pCopySrcLayouts. The implementation **must** include the VK_IMAGE_LAYOUT_GENERAL image layout in pCopySrcLayouts.

The optimalTilingLayoutUUID value can be used to ensure compatible data layouts when using the VK_HOST_IMAGE_COPY_MEMCPY_EXT flag in vkCopyMemoryToImageEXT and vkCopyImageToMemoryEXT.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceHostImageCopyPropertiesEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_HOST_IMAGE_COPY_PROPERTIES_EXT
- VUID-VkPhysicalDeviceHostImageCopyPropertiesEXT-pCopySrcLayouts-parameter
  If copySrcLayoutCount is not 0, and pCopySrcLayouts is not NULL, pCopySrcLayouts **must** be a valid pointer to an array of copySrcLayoutCount VkImageLayout values
- VUID-VkPhysicalDeviceHostImageCopyPropertiesEXT-pCopyDstLayouts-parameter
  If copyDstLayoutCount is not 0, and pCopyDstLayouts is not NULL, pCopyDstLayouts **must** be a valid pointer to an array of copyDstLayoutCount VkImageLayout values

The VkPhysicalDeviceSubpassShadingPropertiesHUAWEI structure is defined as:

```
// Provided by VK_HUAWEI_subpass_shading
typedef struct VkPhysicalDeviceSubpassShadingPropertiesHUAWEI {
    VkStructureType    sType;
    void*              pNext;
    uint32_t           maxSubpassShadingWorkgroupSizeAspectRatio;
} VkPhysicalDeviceSubpassShadingPropertiesHUAWEI;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- maxSubpassShadingWorkgroupSizeAspectRatio indicates the maximum ratio between the width and height of the portion of the subpass shading shader workgroup size. maxSubpassShadingWorkgroupSizeAspectRatio **must** be a power-of-two value, and **must** be less than or equal to max(WorkgroupSize.x / WorkgroupSize.y, WorkgroupSize.y / WorkgroupSize.x).

If the VkPhysicalDeviceSubpassShadingPropertiesHUAWEI structure is included in the pNext chain of the VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with each corresponding implementation-dependent property.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceSubpassShadingPropertiesHUAWEI-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SUBPASS_SHADING_PROPERTIES_HUAWEI

The VkPhysicalDeviceMultiDrawPropertiesEXT structure is defined as:

```
// Provided by VK_EXT_multi_draw
typedef struct VkPhysicalDeviceMultiDrawPropertiesEXT {
    VkStructureType    sType;
    void*              pNext;
    uint32_t           maxMultiDrawCount;
} VkPhysicalDeviceMultiDrawPropertiesEXT;
```

The members of the VkPhysicalDeviceMultiDrawPropertiesEXT structure describe the following features:

- maxMultiDrawCount indicates the maximum number of draw calls which **can** be batched into a single multidraw.

If the VkPhysicalDeviceMultiDrawPropertiesEXT structure is included in the pNext chain of the VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with each corresponding implementation-dependent property.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceMultiDrawPropertiesEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MULTI_DRAW_PROPERTIES_EXT

The VkPhysicalDeviceNestedCommandBufferPropertiesEXT structure is defined as:

```
// Provided by VK_EXT_nested_command_buffer
typedef struct VkPhysicalDeviceNestedCommandBufferPropertiesEXT {
    VkStructureType    sType;
    void*              pNext;
    uint32_t           maxCommandBufferNestingLevel;
} VkPhysicalDeviceNestedCommandBufferPropertiesEXT;
```

The members of the VkPhysicalDeviceNestedCommandBufferPropertiesEXT structure describe the following features:

- maxCommandBufferNestingLevel indicates the maximum nesting level of calls to vkCmdExecuteCommands from Secondary Command Buffers. A maxCommandBufferNestingLevel of UINT32_MAX means there is no limit to the nesting level.

If the VkPhysicalDeviceNestedCommandBufferPropertiesEXT structure is included in the pNext chain of the VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with each corresponding implementation-dependent property.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceNestedCommandBufferPropertiesEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_NESTED_COMMAND_BUFFER_PROPERTIES_EXT

The VkPhysicalDeviceGraphicsPipelineLibraryPropertiesEXT structure is defined as:

```
// Provided by VK_EXT_graphics_pipeline_library
typedef struct VkPhysicalDeviceGraphicsPipelineLibraryPropertiesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           graphicsPipelineLibraryFastLinking;
    VkBool32           graphicsPipelineLibraryIndependentInterpolationDecoration;
} VkPhysicalDeviceGraphicsPipelineLibraryPropertiesEXT;
```

- graphicsPipelineLibraryFastLinking indicates whether fast linking of graphics pipelines is supported. If it is VK_TRUE, creating a graphics pipeline entirely from pipeline libraries without VK_PIPELINE_CREATE_LINK_TIME_OPTIMIZATION_BIT_EXT is comparable in cost to recording a command in a command buffer.
- graphicsPipelineLibraryIndependentInterpolationDecoration indicates whether NoPerspective and Flat interpolation decorations in the last vertex processing stage and the fragment shader are required to match when using graphics pipeline libraries. If it is VK_TRUE, the interpolation decorations do not need to match. If it is VK_FALSE, these decorations **must** either be present in both stages or neither stage in order for a given interface variable to match.

If the VkPhysicalDeviceGraphicsPipelineLibraryPropertiesEXT structure is included in the pNext chain of the VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with each corresponding implementation-dependent property.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceGraphicsPipelineLibraryPropertiesEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_GRAPHICS_PIPELINE_LIBRARY_PROPERTIES_EXT

The VkPhysicalDeviceFragmentShaderBarycentricPropertiesKHR structure is defined as:

```
// Provided by VK_KHR_fragment_shader_barycentric
typedef struct VkPhysicalDeviceFragmentShaderBarycentricPropertiesKHR {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           triStripVertexOrderIndependentOfProvokingVertex;
} VkPhysicalDeviceFragmentShaderBarycentricPropertiesKHR;
```

- When the provoking vertex mode is VK_PROVOKING_VERTEX_MODE_LAST_VERTEX_EXT, and the

primitive order is odd in a triangle strip, the ordering of vertices is defined in last vertex table. triStripVertexOrderIndependentOfProvokingVertex equal to VK_TRUE indicates that the implementation ignores this and uses the vertex order defined by VK_PROVOKING_VERTEX_MODE_FIRST_VERTEX_EXT instead.

If the VkPhysicalDeviceFragmentShaderBarycentricPropertiesKHR structure is included in the pNext chain of the VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with each corresponding implementation-dependent property.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceFragmentShaderBarycentricPropertiesKHR-sType-sType sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_FRAGMENT_SHADER_BARYCENTRIC_PROPERTIES_KHR

The VkPhysicalDeviceShaderModuleIdentifierPropertiesEXT structure is defined as:

```
// Provided by VK_EXT_shader_module_identifier
typedef struct VkPhysicalDeviceShaderModuleIdentifierPropertiesEXT {
    VkStructureType    sType;
    void*              pNext;
    uint8_t            shaderModuleIdentifierAlgorithmUUID[VK_UUID_SIZE];
} VkPhysicalDeviceShaderModuleIdentifierPropertiesEXT;
```

The members of the VkPhysicalDeviceShaderModuleIdentifierPropertiesEXT structure describe the following:

- shaderModuleIdentifierAlgorithmUUID is an array of VK_UUID_SIZE uint8_t values which uniquely represents the algorithm used to compute an identifier in vkGetShaderModuleIdentifierEXT and vkGetShaderModuleCreateInfoIdentifierEXT. Implementations **should** not change this value in different driver versions if the algorithm used to compute an identifier is the same.

**NOTE**

The algorithm UUID may be the same in different ICDs if the algorithms are guaranteed to produce the same results. This may happen in driver stacks which support different kinds of hardware with shared code.

Khronos' conformance testing can not guarantee that shaderModuleIdentifierAlgorithmUUID values are actually unique, so implementors should make their own best efforts to ensure that their UUID is unlikely to conflict with other implementations which may use a different algorithm. In particular, hard-coded values which easily conflict, such as all-0 bits, **should** never be used. Hard-coded values are acceptable if best effort is ensured that the value will not accidentally conflict.

If the VkPhysicalDeviceShaderModuleIdentifierPropertiesEXT structure is included in the pNext chain of the VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with each corresponding implementation-dependent property.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceShaderModuleIdentifierPropertiesEXT-sType-sType `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_MODULE_IDENTIFIER_PROPERTIES_EXT`

The `VkPhysicalDevicePipelineRobustnessPropertiesEXT` structure is defined as:

```
// Provided by VK_EXT_pipeline_robustness
typedef struct VkPhysicalDevicePipelineRobustnessPropertiesEXT {
    VkStructureType                         sType;
    void*                                   pNext;
    VkPipelineRobustnessBufferBehaviorEXT   defaultRobustnessStorageBuffers;
    VkPipelineRobustnessBufferBehaviorEXT   defaultRobustnessUniformBuffers;
    VkPipelineRobustnessBufferBehaviorEXT   defaultRobustnessVertexInputs;
    VkPipelineRobustnessImageBehaviorEXT    defaultRobustnessImages;
} VkPhysicalDevicePipelineRobustnessPropertiesEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `defaultRobustnessStorageBuffers` describes the behavior of out of bounds accesses made to storage buffers when no robustness features are enabled
- `defaultRobustnessUniformBuffers` describes the behavior of out of bounds accesses made to uniform buffers when no robustness features are enabled
- `defaultRobustnessVertexInputs` describes the behavior of out of bounds accesses made to vertex input attributes when no robustness features are enabled
- `defaultRobustnessImages` describes the behavior of out of bounds accesses made to images when no robustness features are enabled

Some implementations of Vulkan may be able to guarantee that certain types of accesses are always performed with robustness even when the Vulkan API's robustness features are not explicitly enabled.

Even when an implementation reports that accesses to a given resource type are robust by default, it remains invalid to make an out of bounds access without requesting the appropriate robustness feature.

If the `VkPhysicalDevicePipelineRobustnessPropertiesEXT` structure is included in the `pNext` chain of the VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with each corresponding implementation-dependent property.

### Valid Usage (Implicit)

- VUID-VkPhysicalDevicePipelineRobustnessPropertiesEXT-sType-sType `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PIPELINE_ROBUSTNESS_PROPERTIES_EXT`

The VkPhysicalDeviceExtendedDynamicState3PropertiesEXT structure is defined as:

```
// Provided by VK_EXT_extended_dynamic_state3
typedef struct VkPhysicalDeviceExtendedDynamicState3PropertiesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           dynamicPrimitiveTopologyUnrestricted;
} VkPhysicalDeviceExtendedDynamicState3PropertiesEXT;
```

- dynamicPrimitiveTopologyUnrestricted indicates that the implementation allows vkCmdSetPrimitiveTopologyEXT to use a different primitive topology class to the one specified in the active graphics pipeline.

If the VkPhysicalDeviceExtendedDynamicState3PropertiesEXT structure is included in the pNext chain of the VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with each corresponding implementation-dependent property.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceExtendedDynamicState3PropertiesEXT-sType-sType sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTENDED_DYNAMIC_STATE_3_PROPERTIES_EXT

The VkPhysicalDeviceOpticalFlowPropertiesNV structure is defined as:

```
// Provided by VK_NV_optical_flow
typedef struct VkPhysicalDeviceOpticalFlowPropertiesNV {
    VkStructureType                 sType;
    void*                           pNext;
    VkOpticalFlowGridSizeFlagsNV    supportedOutputGridSizes;
    VkOpticalFlowGridSizeFlagsNV    supportedHintGridSizes;
    VkBool32                        hintSupported;
    VkBool32                        costSupported;
    VkBool32                        bidirectionalFlowSupported;
    VkBool32                        globalFlowSupported;
    uint32_t                        minWidth;
    uint32_t                        minHeight;
    uint32_t                        maxWidth;
    uint32_t                        maxHeight;
    uint32_t                        maxNumRegionsOfInterest;
} VkPhysicalDeviceOpticalFlowPropertiesNV;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- supportedOutputGridSizes are the supported VkOpticalFlowGridSizeFlagsNV which can be specified in VkOpticalFlowSessionCreateInfoNV::outputGridSize.

- supportedHintGridSizes are the supported VkOpticalFlowGridSizeFlagsNV which can be specified in VkOpticalFlowSessionCreateInfoNV::hintGridSize.
- hintSupported is a boolean describing whether using hint flow vector map is supported in an optical flow session.
- costSupported is a boolean describing whether cost map generation is supported in an optical flow session.
- bidirectionalFlowSupported is a boolean describing whether bi-directional flow generation is supported in an optical flow session.
- globalFlowSupported is a boolean describing whether global flow vector map generation is supported in an optical flow session.
- minWidth is the minimum width in pixels for images used in an optical flow session.
- minHeight is the minimum height in pixels for images used in an optical flow session.
- maxWidth is the maximum width in pixels for images used in an optical flow session.
- maxHeight is the maximum height in pixels for images used in an optical flow session.
- maxNumRegionsOfInterest is the maximum number of regions of interest which can be used in an optical flow session. If this maxNumRegionsOfInterest is 0, regions of interest are not supported in an optical flow session.

If the VkPhysicalDeviceOpticalFlowPropertiesNV structure is included in the pNext chain of the VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with each corresponding implementation-dependent property.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceOpticalFlowPropertiesNV-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_OPTICAL_FLOW_PROPERTIES_NV

The VkPhysicalDeviceOpacityMicromapPropertiesEXT structure is defined as:

```
// Provided by VK_EXT_opacity_micromap
typedef struct VkPhysicalDeviceOpacityMicromapPropertiesEXT {
    VkStructureType    sType;
    void*              pNext;
    uint32_t           maxOpacity2StateSubdivisionLevel;
    uint32_t           maxOpacity4StateSubdivisionLevel;
} VkPhysicalDeviceOpacityMicromapPropertiesEXT;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- maxOpacity2StateSubdivisionLevel is the maximum allowed subdivisionLevel when format is VK_OPACITY_MICROMAP_FORMAT_2_STATE_EXT
- maxOpacity4StateSubdivisionLevel is the maximum allowed subdivisionLevel when format is

VK_OPACITY_MICROMAP_FORMAT_4_STATE_EXT

If the VkPhysicalDeviceOpacityMicromapPropertiesEXT structure is included in the pNext chain of the VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with each corresponding implementation-dependent property.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceOpacityMicromapPropertiesEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_OPACITY_MICROMAP_PROPERTIES_EXT

The VkPhysicalDeviceDisplacementMicromapPropertiesNV structure is defined as:

```
// Provided by VK_NV_displacement_micromap
typedef struct VkPhysicalDeviceDisplacementMicromapPropertiesNV {
    VkStructureType    sType;
    void*              pNext;
    uint32_t           maxDisplacementMicromapSubdivisionLevel;
} VkPhysicalDeviceDisplacementMicromapPropertiesNV;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- maxDisplacementMicromapSubdivisionLevel is the maximum allowed subdivisionLevel for displacement micromaps.

If the VkPhysicalDeviceDisplacementMicromapPropertiesNV structure is included in the pNext chain of the VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with each corresponding implementation-dependent property.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceDisplacementMicromapPropertiesNV-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DISPLACEMENT_MICROMAP_PROPERTIES_NV

The VkPhysicalDeviceShaderCoreBuiltinsPropertiesARM structure is defined as:

```
// Provided by VK_ARM_shader_core_builtins
typedef struct VkPhysicalDeviceShaderCoreBuiltinsPropertiesARM {
    VkStructureType    sType;
    void*              pNext;
    uint64_t           shaderCoreMask;
    uint32_t           shaderCoreCount;
    uint32_t           shaderWarpsPerCore;
} VkPhysicalDeviceShaderCoreBuiltinsPropertiesARM;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- shaderCoreMask is a bitfield where each bit set represents the presence of a shader core whose ID is the bit position. The highest ID for any shader core on the device is the position of the most significant bit set.
- shaderCoreCount is the number of shader cores on the device.
- shaderWarpsPerCore is the maximum number of simultaneously executing warps on a shader core.

If the VkPhysicalDeviceShaderCoreBuiltinsPropertiesARM structure is included in the pNext chain of the VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with each corresponding implementation-dependent property.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceShaderCoreBuiltinsPropertiesARM-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_CORE_BUILTINS_PROPERTIES_ARM

Values which **may** be returned in the rayTracingInvocationReorderReorderingHint field of VkPhysicalDeviceRayTracingInvocationReorderPropertiesNV are:

```
// Provided by VK_NV_ray_tracing_invocation_reorder
typedef enum VkRayTracingInvocationReorderModeNV {
    VK_RAY_TRACING_INVOCATION_REORDER_MODE_NONE_NV = 0,
    VK_RAY_TRACING_INVOCATION_REORDER_MODE_REORDER_NV = 1,
} VkRayTracingInvocationReorderModeNV;
```

- VK_RAY_TRACING_INVOCATION_REORDER_MODE_NONE_NV indicates that the implementation is likely to not reorder at reorder calls.
- VK_RAY_TRACING_INVOCATION_REORDER_MODE_REORDER_NV indicates that the implementation is likely to reorder at reorder calls.

The VkPhysicalDeviceRayTracingInvocationReorderPropertiesNV structure is defined as:

```
// Provided by VK_NV_ray_tracing_invocation_reorder
typedef struct VkPhysicalDeviceRayTracingInvocationReorderPropertiesNV {
    VkStructureType                         sType;
    void*                                   pNext;
    VkRayTracingInvocationReorderModeNV     rayTracingInvocationReorderReorderingHint;
} VkPhysicalDeviceRayTracingInvocationReorderPropertiesNV;
```

- rayTracingInvocationReorderReorderingHint is a hint indicating if the implementation will actually reorder at the reorder calls.

| | |
|---|---|
| **NOTE** | Because the extension changes how hits are managed there is a compatibility reason to expose the extension even when an implementation does not have sorting active. |

If the VkPhysicalDeviceRayTracingInvocationReorderPropertiesNV structure is included in the pNext chain of the VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with each corresponding implementation-dependent property.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceRayTracingInvocationReorderPropertiesNV-sType-sType sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_RAY_TRACING_INVOCATION_REORDER_PROPERTIES_NV

The VkPhysicalDeviceClusterCullingShaderPropertiesHUAWEI structure is defined as:

```
// Provided by VK_HUAWEI_cluster_culling_shader
typedef struct VkPhysicalDeviceClusterCullingShaderPropertiesHUAWEI {
    VkStructureType    sType;
    void*              pNext;
    uint32_t           maxWorkGroupCount[3];
    uint32_t           maxWorkGroupSize[3];
    uint32_t           maxOutputClusterCount;
    VkDeviceSize       indirectBufferOffsetAlignment;
} VkPhysicalDeviceClusterCullingShaderPropertiesHUAWEI;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- maxWorkGroupCount[3] is the maximum number of local workgroups that can be launched by a single command. These three value represent the maximum local workgroup count in the X, Y and Z dimensions, respectively. In the current implementation, the values of Y and Z are both implicitly set as one. groupCountX of DrawCluster command **must** be less than or equal to maxWorkGroupCount[0].
- maxWorkGroupSize[3] is the maximum size of a local workgroup. These three value represent the maximum local workgroup size in the X, Y and Z dimensions, respectively. The x, y and z sizes, as specified by the LocalSize or LocalSizeId execution mode or by the object decorated by the WorkgroupSize decoration in shader modules, **must** be less than or equal to the corresponding limit. In the current implementation, the maximum workgroup size of the X dimension is 32, the others are 1.
- maxOutputClusterCount is the maximum number of output cluster a single cluster culling shader workgroup can emit.
- indirectBufferOffsetAlignment indicates the alignment for cluster drawing command buffer stride. vkCmdDrawClusterIndirectHUAWEI::offset **must** be a multiple of this value.

If the VkPhysicalDeviceClusterCullingShaderPropertiesHUAWEI structure is included in the pNext chain of the VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with each corresponding implementation-dependent property.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceClusterCullingShaderPropertiesHUAWEI-sType-sType sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_CLUSTER_CULLING_SHADER_PROPERTIES_HUAWEI

The VkPhysicalDeviceShaderCorePropertiesARM structure is defined as:

```
// Provided by VK_ARM_shader_core_properties
typedef struct VkPhysicalDeviceShaderCorePropertiesARM {
    VkStructureType    sType;
    void*              pNext;
    uint32_t           pixelRate;
    uint32_t           texelRate;
    uint32_t           fmaRate;
} VkPhysicalDeviceShaderCorePropertiesARM;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- pixelRate is an unsigned integer value indicating the maximum number of pixels output per clock per shader core.
- texelRate is an unsigned integer value indicating the maximum number of texels per clock per shader core.
- fmaRate is an unsigned integer value indicating the maximum number of single-precision fused multiply-add operations per clock per shader core.

If a throughput rate cannot be determined on the physical device, the value 0 will be returned for that rate.

If the VkPhysicalDeviceShaderCorePropertiesARM structure is included in the pNext chain of the VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with each corresponding implementation-dependent property.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceShaderCorePropertiesARM-sType-sType sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_CORE_PROPERTIES_ARM

The VkPhysicalDeviceShaderObjectPropertiesEXT structure is defined as:

```
// Provided by VK_EXT_shader_object
typedef struct VkPhysicalDeviceShaderObjectPropertiesEXT {
    VkStructureType    sType;
    void*              pNext;
    uint8_t            shaderBinaryUUID[VK_UUID_SIZE];
    uint32_t           shaderBinaryVersion;
} VkPhysicalDeviceShaderObjectPropertiesEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `shaderBinaryUUID` is an array of `VK_UUID_SIZE` `uint8_t` values representing a universally unique identifier for one or more implementations whose shader binaries are guaranteed to be compatible with each other.
- `shaderBinaryVersion` is an unsigned integer incremented to represent backwards compatible differences between implementations with the same `shaderBinaryUUID`.

The purpose and usage of the values of this structure are described in greater detail in Binary Shader Compatibility.

If the `VkPhysicalDeviceShaderObjectPropertiesEXT` structure is included in the `pNext` chain of the VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with each corresponding implementation-dependent property.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceShaderObjectPropertiesEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_OBJECT_PROPERTIES_EXT`

The `VkPhysicalDeviceShaderEnqueuePropertiesAMDX` structure is defined as:

```
// Provided by VK_AMDX_shader_enqueue
typedef struct VkPhysicalDeviceShaderEnqueuePropertiesAMDX {
    VkStructureType    sType;
    void*              pNext;
    uint32_t           maxExecutionGraphDepth;
    uint32_t           maxExecutionGraphShaderOutputNodes;
    uint32_t           maxExecutionGraphShaderPayloadSize;
    uint32_t           maxExecutionGraphShaderPayloadCount;
    uint32_t           executionGraphDispatchAddressAlignment;
} VkPhysicalDeviceShaderEnqueuePropertiesAMDX;
```

The members of the `VkPhysicalDeviceShaderEnqueuePropertiesAMDX` structure describe the following limits:

- `sType` is a VkStructureType value identifying this structure.

- pNext is NULL or a pointer to a structure extending this structure.
- maxExecutionGraphDepth defines the maximum node chain depth in the graph. The dispatched node is at depth 1 and the node enqueued by it is at depth 2, and so on. If a node enqueues itself, each recursive enqueue increases the depth by 1 as well.
- maxExecutionGraphShaderOutputNodes specifies the maximum number of unique nodes that can be dispatched from a single shader, and **must** be at least 256.
- maxExecutionGraphShaderPayloadSize specifies the maximum total size of payload declarations in a shader. For any payload declarations that share resources, indicated by NodeSharesPayloadLimitsWithAMDX decorations, the maximum size of each set of shared payload declarations is taken. The sum of each shared set's maximum size and the size of each unshared payload is counted against this limit.
- maxExecutionGraphShaderPayloadCount specifies the maximum number of output payloads that can be initialized in a single workgroup.
- executionGraphDispatchAddressAlignment specifies the alignment of non-scratch VkDeviceAddress arguments consumed by graph dispatch commands.

If the VkPhysicalDeviceShaderEnqueuePropertiesAMDX structure is included in the pNext chain of the VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with each corresponding implementation-dependent property.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceShaderEnqueuePropertiesAMDX-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_ENQUEUE_PROPERTIES_AMDX

The VkPhysicalDeviceExtendedSparseAddressSpacePropertiesNV structure is defined as:

```
// Provided by VK_NV_extended_sparse_address_space
typedef struct VkPhysicalDeviceExtendedSparseAddressSpacePropertiesNV {
    VkStructureType       sType;
    void*                 pNext;
    VkDeviceSize          extendedSparseAddressSpaceSize;
    VkImageUsageFlags     extendedSparseImageUsageFlags;
    VkBufferUsageFlags    extendedSparseBufferUsageFlags;
} VkPhysicalDeviceExtendedSparseAddressSpacePropertiesNV;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- extendedSparseAddressSpaceSize is the total amount of address space available, in bytes, for sparse memory resources of all usages if the extendedSparseAddressSpace feature is enabled. This **must** be greater than or equal to VkPhysicalDeviceLimits::sparseAddressSpaceSize, and the difference in space **must** only be used with usages allowed below. This is an upper bound on the sum of the sizes of all sparse resources, regardless of whether any memory is bound to them.
- extendedSparseImageUsageFlags is a bitmask of VkImageUsageFlagBits of usages which **may**

allow an implementation to use the full extendedSparseAddressSpaceSize space.

- extendedSparseBufferUsageFlags is a bitmask of VkBufferUsageFlagBits of usages which **may** allow an implementation to use the full extendedSparseAddressSpaceSize space.

If the VkPhysicalDeviceExtendedSparseAddressSpacePropertiesNV structure is included in the pNext chain of the VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with each corresponding implementation-dependent property.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceExtendedSparseAddressSpacePropertiesNV-sType-sType sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTENDED_SPARSE_ADDRESS_SPACE_PROPERTIES_NV

The VkPhysicalDeviceCudaKernelLaunchPropertiesNV structure is defined as:

```
// Provided by VK_NV_cuda_kernel_launch
typedef struct VkPhysicalDeviceCudaKernelLaunchPropertiesNV {
    VkStructureType    sType;
    void*              pNext;
    uint32_t           computeCapabilityMinor;
    uint32_t           computeCapabilityMajor;
} VkPhysicalDeviceCudaKernelLaunchPropertiesNV;
```

The members of the VkPhysicalDeviceCudaKernelLaunchPropertiesNV structure describe the following features:

- computeCapabilityMinor indicates the minor version number of the compute code.
- computeCapabilityMajor indicates the major version number of the compute code.

If the VkPhysicalDeviceCudaKernelLaunchPropertiesNV structure is included in the pNext chain of the VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with each corresponding implementation-dependent property.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceCudaKernelLaunchPropertiesNV-sType-sType sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_CUDA_KERNEL_LAUNCH_PROPERTIES_NV

The VkPhysicalDeviceExternalFormatResolvePropertiesANDROID structure is defined as:

```
// Provided by VK_ANDROID_external_format_resolve
typedef struct VkPhysicalDeviceExternalFormatResolvePropertiesANDROID {
    VkStructureType     sType;
    void*               pNext;
    VkBool32            nullColorAttachmentWithExternalFormatResolve;
    VkChromaLocation    externalFormatResolveChromaOffsetX;
    VkChromaLocation    externalFormatResolveChromaOffsetY;
} VkPhysicalDeviceExternalFormatResolvePropertiesANDROID;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `nullColorAttachmentWithExternalFormatResolve` indicates that there **must** be no color attachment image when performing external format resolves if it is `VK_TRUE`.
- `externalFormatResolveChromaOffsetX` indicates the VkChromaLocation that an implementation uses in the X axis for accesses to an external format image as a resolve attachment. This **must** be consistent between external format resolves and load operations from external format resolve attachments to color attachments when `nullColorAttachmentWithExternalFormatResolve` is `VK_TRUE`.
- `externalFormatResolveChromaOffsetY` indicates the VkChromaLocation that an implementation uses in the Y axis for accesses to an external format image as a resolve attachment. This **must** be consistent between external format resolves and load operations from external format resolve attachments to color attachments when `nullColorAttachmentWithExternalFormatResolve` is `VK_TRUE`.

If the `VkPhysicalDeviceExternalFormatResolvePropertiesANDROID` structure is included in the `pNext` chain of the VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with each corresponding implementation-dependent property.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceExternalFormatResolvePropertiesANDROID-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTERNAL_FORMAT_RESOLVE_PROPERTIES_ANDROID`

The `VkPhysicalDevicePipelineBinaryPropertiesKHR` structure is defined as:

```
// Provided by VK_KHR_pipeline_binary
typedef struct VkPhysicalDevicePipelineBinaryPropertiesKHR {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           pipelineBinaryInternalCache;
    VkBool32           pipelineBinaryInternalCacheControl;
    VkBool32           pipelineBinaryPrefersInternalCache;
    VkBool32           pipelineBinaryPrecompiledInternalCache;
    VkBool32           pipelineBinaryCompressedData;
} VkPhysicalDevicePipelineBinaryPropertiesKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `pipelineBinaryInternalCache` specifies that the implementation maintains a pipeline cache internal to the implementation. If this is `VK_TRUE`, applications **can** create pipeline binaries with only a pipeline create info, and in this case, an implementation **may** be able to create a pipeline binary directly without application needing to capture the binary itself.
- `pipelineBinaryInternalCacheControl` specifies whether the driver's internal cache **can** be disabled. If this property is `VK_TRUE` VkDevicePipelineBinaryInternalCacheControlKHR::`disableInternalCache` **can** be used to disable the driver's internal cache, allowing an application to take full control of both memory and disk usage.
- `pipelineBinaryPrefersInternalCache` specifies that the implementation prefers to maintain an internal cache, and applications **should** not store pipeline binaries in their own on-disk caches to avoid increased on-disk storage requirements. Applications are encouraged to only store pipeline keys instead, and aim to create pipeline binaries from key alone on subsequent runs of the application.
- `pipelineBinaryPrecompiledInternalCache` specifies that the implementation **may** have pipeline binaries in its internal cache, which is populated without the application ever having generated that pipeline itself. Applications **can** attempt to create binaries without extracting pipeline binary data from the pipeline prior for a set of pipeline keys, including from previous runs of the application.
- `pipelineBinaryCompressedData` specifies that the binary data is already compressed and so applications **should** not attempt to compress it.

**NOTE**

These properties tend to be platform specific and may change depending on external configuration which is outside the scope of this specification. These properties are intended to guide applications when implementations have dedicated caching solutions available. In particular, if the `pipelineBinaryPrefersInternalCache` limit is exposed, relying on the internal cache may provide some advantage compared to an application-specific solution. An application with its own dedicated solution may still use its own caching system even with this limit exposed.

If the `VkPhysicalDevicePipelineBinaryPropertiesKHR` structure is included in the `pNext` chain of the VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with

each corresponding implementation-dependent property.

### Valid Usage (Implicit)

- VUID-VkPhysicalDevicePipelineBinaryPropertiesKHR-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PIPELINE_BINARY_PROPERTIES_KHR`

The `VkPhysicalDeviceRenderPassStripedPropertiesARM` structure is defined as:

```
// Provided by VK_ARM_render_pass_striped
typedef struct VkPhysicalDeviceRenderPassStripedPropertiesARM {
    VkStructureType    sType;
    void*              pNext;
    VkExtent2D         renderPassStripeGranularity;
    uint32_t           maxRenderPassStripes;
} VkPhysicalDeviceRenderPassStripedPropertiesARM;
```

The members of the `VkPhysicalDeviceRenderPassStripedPropertiesARM` structure describe the following limits:

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `renderPassStripeGranularity` indicates the minimum supported granularity of striped render pass regions.
- `maxRenderPassStripes` indicates the maximum number of stripes supported in striped rendering.

If the `VkPhysicalDeviceRenderPassStripedPropertiesARM` structure is included in the `pNext` chain of the VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with each corresponding implementation-dependent property.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceRenderPassStripedPropertiesARM-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_RENDER_PASS_STRIPED_PROPERTIES_ARM`

The `VkPhysicalDeviceComputeShaderDerivativesPropertiesKHR` structure is defined as:

```
// Provided by VK_KHR_compute_shader_derivatives
typedef struct VkPhysicalDeviceComputeShaderDerivativesPropertiesKHR {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           meshAndTaskShaderDerivatives;
} VkPhysicalDeviceComputeShaderDerivativesPropertiesKHR;
```

The members of the VkPhysicalDeviceComputeShaderDerivativesPropertiesKHR structure describe the following:

- meshAndTaskShaderDerivatives indicates whether the mesh and task shader stages support the ComputeDerivativeGroupQuadsKHR and ComputeDerivativeGroupLinearKHR SPIR-V capabilities.

If the VkPhysicalDeviceComputeShaderDerivativesPropertiesKHR structure is included in the pNext chain of the VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with each corresponding implementation-dependent property.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceComputeShaderDerivativesPropertiesKHR-sType-sType sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_COMPUTE_SHADER_DERIVATIVES_PROPERTIES_KHR

The VkPhysicalDeviceMapMemoryPlacedPropertiesEXT structure is defined as:

```
// Provided by VK_EXT_map_memory_placed
typedef struct VkPhysicalDeviceMapMemoryPlacedPropertiesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkDeviceSize       minPlacedMemoryMapAlignment;
} VkPhysicalDeviceMapMemoryPlacedPropertiesEXT;
```

The members of the VkPhysicalDeviceMapMemoryPlacedPropertiesEXT structure describe the following:

- minPlacedMemoryMapAlignment is the minimum alignment required for memory object offsets and virtual address ranges when using placed memory mapping.

If the VkPhysicalDeviceMapMemoryPlacedPropertiesEXT structure is included in the pNext chain of the VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with each corresponding implementation-dependent property.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceMapMemoryPlacedPropertiesEXT-sType-sType sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MAP_MEMORY_PLACED_PROPERTIES_EXT

The VkPhysicalDeviceImageAlignmentControlPropertiesMESA structure is defined as:

```
// Provided by VK_MESA_image_alignment_control
typedef struct VkPhysicalDeviceImageAlignmentControlPropertiesMESA {
    VkStructureType    sType;
    void*              pNext;
    uint32_t           supportedImageAlignmentMask;
} VkPhysicalDeviceImageAlignmentControlPropertiesMESA;
```

The members of the VkPhysicalDeviceImageAlignmentControlPropertiesMESA structure describe the following:

- supportedImageAlignmentMask is a bitwise-or of all potentially supported image alignments for a given physical device when using VK_IMAGE_TILING_OPTIMAL. If a given alignment is supported, the application **can** request an image to have that alignment. A given set of image creation parameters **may** support a subset of these alignments. To determine if a particular alignment is supported for a given set of image creation parameters, check VkMemoryRequirements::alignment after chaining in VkImageAlignmentControlCreateInfoMESA.

If the VkPhysicalDeviceImageAlignmentControlPropertiesMESA structure is included in the pNext chain of the VkPhysicalDeviceProperties2 structure passed to vkGetPhysicalDeviceProperties2, it is filled in with each corresponding implementation-dependent property.

**Valid Usage (Implicit)**

- VUID-VkPhysicalDeviceImageAlignmentControlPropertiesMESA-sType-sType sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_IMAGE_ALIGNMENT_CONTROL_PROPERTIES_MESA

# 47.1. Limit Requirements

The following table specifies the **required** minimum/maximum for all Vulkan graphics implementations. Where a limit corresponds to a fine-grained device feature which is **optional**, the feature name is listed with two **required** limits, one when the feature is supported and one when it is not supported. If an implementation supports a feature, the limits reported are the same whether or not the feature is enabled.

*Table 64. Required Limit Types*

| Type | Limit | Feature |
|---|---|---|
| uint32_t | maxImageDimension1D | - |
| uint32_t | maxImageDimension2D | - |
| uint32_t | maxImageDimension3D | - |
| uint32_t | maxImageDimensionCube | - |
| uint32_t | maxImageArrayLayers | - |
| uint32_t | maxTexelBufferElements | - |

| Type | Limit | Feature |
|---|---|---|
| uint32_t | maxUniformBufferRange | - |
| uint32_t | maxStorageBufferRange | - |
| uint32_t | maxPushConstantsSize | - |
| uint32_t | maxMemoryAllocationCount | - |
| uint32_t | maxSamplerAllocationCount | - |
| VkDeviceSize | bufferImageGranularity | - |
| VkDeviceSize | sparseAddressSpaceSize | sparseBinding |
| uint32_t | maxBoundDescriptorSets | - |
| uint32_t | maxPerStageDescriptorSamplers | - |
| uint32_t | maxPerStageDescriptorUniformBuffers | - |
| uint32_t | maxPerStageDescriptorStorageBuffers | - |
| uint32_t | maxPerStageDescriptorSampledImages | - |
| uint32_t | maxPerStageDescriptorStorageImages | - |
| uint32_t | maxPerStageDescriptorInputAttachments | - |
| uint32_t | maxPerStageResources | - |
| uint32_t | maxDescriptorSetSamplers | - |
| uint32_t | maxDescriptorSetUniformBuffers | - |
| uint32_t | maxDescriptorSetUniformBuffersDynamic | - |
| uint32_t | maxDescriptorSetStorageBuffers | - |
| uint32_t | maxDescriptorSetStorageBuffersDynamic | - |
| uint32_t | maxDescriptorSetSampledImages | - |
| uint32_t | maxDescriptorSetStorageImages | - |
| uint32_t | maxDescriptorSetInputAttachments | - |
| uint32_t | maxVertexInputAttributes | - |
| uint32_t | maxVertexInputBindings | - |
| uint32_t | maxVertexInputAttributeOffset | - |
| uint32_t | maxVertexInputBindingStride | - |
| uint32_t | maxVertexOutputComponents | - |
| uint32_t | maxTessellationGenerationLevel | tessellationShader |
| uint32_t | maxTessellationPatchSize | tessellationShader |
| uint32_t | maxTessellationControlPerVertexInputComponents | tessellationShader |
| uint32_t | maxTessellationControlPerVertexOutputComponents | tessellationShader |
| uint32_t | maxTessellationControlPerPatchOutputComponents | tessellationShader |

| Type | Limit | Feature |
|---|---|---|
| uint32_t | maxTessellationControlTotalOutputComponents | tessellationShader |
| uint32_t | maxTessellationEvaluationInputComponents | tessellationShader |
| uint32_t | maxTessellationEvaluationOutputComponents | tessellationShader |
| uint32_t | maxGeometryShaderInvocations | geometryShader |
| uint32_t | maxGeometryInputComponents | geometryShader |
| uint32_t | maxGeometryOutputComponents | geometryShader |
| uint32_t | maxGeometryOutputVertices | geometryShader |
| uint32_t | maxGeometryTotalOutputComponents | geometryShader |
| uint32_t | maxFragmentInputComponents | - |
| uint32_t | maxFragmentOutputAttachments | - |
| uint32_t | maxFragmentDualSrcAttachments | dualSrcBlend |
| uint32_t | maxFragmentCombinedOutputResources | - |
| uint32_t | maxComputeSharedMemorySize | - |
| 3 × uint32_t | maxComputeWorkGroupCount | - |
| uint32_t | maxComputeWorkGroupInvocations | - |
| 3 × uint32_t | maxComputeWorkGroupSize | - |
| uint32_t | subPixelPrecisionBits | - |
| uint32_t | subTexelPrecisionBits | - |
| uint32_t | mipmapPrecisionBits | - |
| uint32_t | maxDrawIndexedIndexValue | fullDrawIndexUint32 |
| uint32_t | maxDrawIndirectCount | multiDrawIndirect |
| float | maxSamplerLodBias | - |
| float | maxSamplerAnisotropy | samplerAnisotropy |
| uint32_t | maxViewports | multiViewport |
| 2 × uint32_t | maxViewportDimensions | - |
| 2 × float | viewportBoundsRange | - |
| uint32_t | viewportSubPixelBits | - |
| size_t | minMemoryMapAlignment | - |
| VkDeviceSize | minTexelBufferOffsetAlignment | - |
| VkDeviceSize | minUniformBufferOffsetAlignment | - |
| VkDeviceSize | minStorageBufferOffsetAlignment | - |
| int32_t | minTexelOffset | - |
| uint32_t | maxTexelOffset | - |
| int32_t | minTexelGatherOffset | shaderImageGatherExtended |
| uint32_t | maxTexelGatherOffset | shaderImageGatherExtended |

| Type | Limit | Feature |
|---|---|---|
| float | minInterpolationOffset | sampleRateShading |
| float | maxInterpolationOffset | sampleRateShading |
| uint32_t | subPixelInterpolationOffsetBits | sampleRateShading |
| uint32_t | maxFramebufferWidth | - |
| uint32_t | maxFramebufferHeight | - |
| uint32_t | maxFramebufferLayers | - |
| VkSampleCountFlags | framebufferColorSampleCounts | - |
| VkSampleCountFlags | framebufferDepthSampleCounts | - |
| VkSampleCountFlags | framebufferStencilSampleCounts | - |
| VkSampleCountFlags | framebufferNoAttachmentsSampleCounts | - |
| uint32_t | maxColorAttachments | - |
| VkSampleCountFlags | sampledImageColorSampleCounts | - |
| VkSampleCountFlags | sampledImageIntegerSampleCounts | - |
| VkSampleCountFlags | sampledImageDepthSampleCounts | - |
| VkSampleCountFlags | sampledImageStencilSampleCounts | - |
| VkSampleCountFlags | storageImageSampleCounts | shaderStorageImageMultisample |
| uint32_t | maxSampleMaskWords | - |
| VkBool32 | timestampComputeAndGraphics | - |
| float | timestampPeriod | - |
| uint32_t | maxClipDistances | shaderClipDistance |
| uint32_t | maxCullDistances | shaderCullDistance |
| uint32_t | maxCombinedClipAndCullDistances | shaderCullDistance |
| uint32_t | discreteQueuePriorities | - |
| 2 × float | pointSizeRange | largePoints |
| 2 × float | lineWidthRange | wideLines |
| float | pointSizeGranularity | largePoints |
| float | lineWidthGranularity | wideLines |
| VkBool32 | strictLines | - |

| Type | Limit | Feature |
|---|---|---|
| VkBool32 | standardSampleLocations | - |
| VkDeviceSize | optimalBufferCopyOffsetAlignment | - |
| VkDeviceSize | optimalBufferCopyRowPitchAlignment | - |
| VkDeviceSize | nonCoherentAtomSize | - |
| uint32_t | maxDiscardRectangles | VK_EXT_discard_rectangles |
| VkBool32 | filterMinmaxSingleComponentFormats | VK_EXT_sampler_filter_minmax |
| VkBool32 | filterMinmaxImageComponentMapping | VK_EXT_sampler_filter_minmax |
| VkDeviceSize | maxBufferSize | maintenance4 |
| float | primitiveOverestimationSize | VK_EXT_conservative_rasterization |
| VkBool32 | maxExtraPrimitiveOverestimationSize | VK_EXT_conservative_rasterization |
| float | extraPrimitiveOverestimationSizeGranularity | VK_EXT_conservative_rasterization |
| VkBool32 | degenerateTriangleRasterized | VK_EXT_conservative_rasterization |
| float | degenerateLinesRasterized | VK_EXT_conservative_rasterization |
| VkBool32 | fullyCoveredFragmentShaderInputVariable | VK_EXT_conservative_rasterization |
| VkBool32 | conservativeRasterizationPostDepthCoverage | VK_EXT_conservative_rasterization |
| uint32_t | maxUpdateAfterBindDescriptorsInAllPools | descriptorIndexing |
| VkBool32 | shaderUniformBufferArrayNonUniformIndexingNative | - |
| VkBool32 | shaderSampledImageArrayNonUniformIndexingNative | - |
| VkBool32 | shaderStorageBufferArrayNonUniformIndexingNative | - |
| VkBool32 | shaderStorageImageArrayNonUniformIndexingNative | - |
| VkBool32 | shaderInputAttachmentArrayNonUniformIndexingNative | - |
| uint32_t | maxPerStageDescriptorUpdateAfterBindSamplers | descriptorIndexing |
| uint32_t | maxPerStageDescriptorUpdateAfterBindUniformBuffers | descriptorIndexing |
| uint32_t | maxPerStageDescriptorUpdateAfterBindStorageBuffers | descriptorIndexing |
| uint32_t | maxPerStageDescriptorUpdateAfterBindSampledImages | descriptorIndexing |

| Type | Limit | Feature |
|---|---|---|
| uint32_t | maxPerStageDescriptorUpdateAfterBindStorageImages | descriptorIndexing |
| uint32_t | maxPerStageDescriptorUpdateAfterBindInputAttachments | descriptorIndexing |
| uint32_t | maxPerStageUpdateAfterBindResources | descriptorIndexing |
| uint32_t | maxDescriptorSetUpdateAfterBindSamplers | descriptorIndexing |
| uint32_t | maxDescriptorSetUpdateAfterBindUniformBuffers | descriptorIndexing |
| uint32_t | maxDescriptorSetUpdateAfterBindUniformBuffersDynamic | descriptorIndexing |
| uint32_t | maxDescriptorSetUpdateAfterBindStorageBuffers | descriptorIndexing |
| uint32_t | maxDescriptorSetUpdateAfterBindStorageBuffersDynamic | descriptorIndexing |
| uint32_t | maxDescriptorSetUpdateAfterBindSampledImages | descriptorIndexing |
| uint32_t | maxDescriptorSetUpdateAfterBindStorageImages | descriptorIndexing |
| uint32_t | maxDescriptorSetUpdateAfterBindInputAttachments | descriptorIndexing |
| uint32_t | maxInlineUniformBlockSize | inlineUniformBlock |
| uint32_t | maxPerStageDescriptorInlineUniformBlocks | inlineUniformBlock |
| uint32_t | maxPerStageDescriptorUpdateAfterBindInlineUniformBlocks | inlineUniformBlock |
| uint32_t | maxDescriptorSetInlineUniformBlocks | inlineUniformBlock |
| uint32_t | maxDescriptorSetUpdateAfterBindInlineUniformBlocks | inlineUniformBlock |
| uint32_t | maxVertexAttribDivisor | VK_KHR_vertex_attribute_divisor |
| uint32_t | VkPhysicalDeviceMeshShaderPropertiesNV::maxDrawMeshTasksCount | VK_NV_mesh_shader |
| uint32_t | VkPhysicalDeviceMeshShaderPropertiesNV::maxTaskWorkGroupInvocations | VK_NV_mesh_shader |
| 3 × uint32_t | VkPhysicalDeviceMeshShaderPropertiesNV::maxTaskWorkGroupSize | VK_NV_mesh_shader |
| uint32_t | VkPhysicalDeviceMeshShaderPropertiesNV::maxTaskTotalMemorySize | VK_NV_mesh_shader |
| uint32_t | VkPhysicalDeviceMeshShaderPropertiesNV::maxTaskOutputCount | VK_NV_mesh_shader |
| uint32_t | VkPhysicalDeviceMeshShaderPropertiesNV::maxMeshWorkGroupInvocations | VK_NV_mesh_shader |
| 3 × uint32_t | VkPhysicalDeviceMeshShaderPropertiesNV::maxMeshWorkGroupSize | VK_NV_mesh_shader |
| uint32_t | VkPhysicalDeviceMeshShaderPropertiesNV::maxMeshTotalMemorySize | VK_NV_mesh_shader |
| uint32_t | VkPhysicalDeviceMeshShaderPropertiesNV::maxMeshOutputVertices | VK_NV_mesh_shader |

| Type | Limit | Feature |
|---|---|---|
| uint32_t | VkPhysicalDeviceMeshShaderPropertiesNV::maxMeshOutputPrimitives | VK_NV_mesh_shader |
| uint32_t | VkPhysicalDeviceMeshShaderPropertiesNV::maxMeshMultiviewViewCount | VK_NV_mesh_shader |
| uint32_t | VkPhysicalDeviceMeshShaderPropertiesNV::meshOutputPerVertexGranularity | VK_NV_mesh_shader |
| uint32_t | VkPhysicalDeviceMeshShaderPropertiesNV::meshOutputPerPrimitiveGranularity | VK_NV_mesh_shader |
| uint32_t | VkPhysicalDeviceMeshShaderPropertiesEXT::maxTaskWorkGroupTotalCount | VK_EXT_mesh_shader |
| 3 × uint32_t | VkPhysicalDeviceMeshShaderPropertiesEXT::maxTaskWorkGroupCount | VK_EXT_mesh_shader |
| uint32_t | VkPhysicalDeviceMeshShaderPropertiesEXT::maxTaskWorkGroupInvocations | VK_EXT_mesh_shader |
| 3 × uint32_t | VkPhysicalDeviceMeshShaderPropertiesEXT::maxTaskWorkGroupSize | VK_EXT_mesh_shader |
| uint32_t | VkPhysicalDeviceMeshShaderPropertiesEXT::maxTaskPayloadSize | VK_EXT_mesh_shader |
| uint32_t | VkPhysicalDeviceMeshShaderPropertiesEXT::maxTaskSharedMemorySize | VK_EXT_mesh_shader |
| uint32_t | VkPhysicalDeviceMeshShaderPropertiesEXT::maxTaskPayloadAndSharedMemorySize | VK_EXT_mesh_shader |
| uint32_t | VkPhysicalDeviceMeshShaderPropertiesEXT::maxMeshWorkGroupTotalCount | VK_EXT_mesh_shader |
| 3 × uint32_t | VkPhysicalDeviceMeshShaderPropertiesEXT::maxMeshWorkGroupCount | VK_EXT_mesh_shader |
| uint32_t | VkPhysicalDeviceMeshShaderPropertiesEXT::maxMeshWorkGroupInvocations | VK_EXT_mesh_shader |
| 3 × uint32_t | VkPhysicalDeviceMeshShaderPropertiesEXT::maxMeshWorkGroupSize | VK_EXT_mesh_shader |
| uint32_t | VkPhysicalDeviceMeshShaderPropertiesEXT::maxMeshSharedMemorySize | VK_EXT_mesh_shader |
| uint32_t | VkPhysicalDeviceMeshShaderPropertiesEXT::maxMeshPayloadAndSharedMemorySize | VK_EXT_mesh_shader |
| uint32_t | VkPhysicalDeviceMeshShaderPropertiesEXT::maxMeshOutputMemorySize | VK_EXT_mesh_shader |
| uint32_t | VkPhysicalDeviceMeshShaderPropertiesEXT::maxMeshPayloadAndOutputMemorySize | VK_EXT_mesh_shader |
| uint32_t | VkPhysicalDeviceMeshShaderPropertiesEXT::maxMeshOutputComponents | VK_EXT_mesh_shader |
| uint32_t | VkPhysicalDeviceMeshShaderPropertiesEXT::maxMeshOutputVertices | VK_EXT_mesh_shader |

| Type | Limit | Feature |
|---|---|---|
| uint32_t | VkPhysicalDeviceMeshShaderPropertiesEXT::maxMeshOutputPrimitives | VK_EXT_mesh_shader |
| uint32_t | VkPhysicalDeviceMeshShaderPropertiesEXT::maxMeshOutputLayers | VK_EXT_mesh_shader |
| uint32_t | VkPhysicalDeviceMeshShaderPropertiesEXT::maxMeshMultiviewViewCount | VK_EXT_mesh_shader |
| uint32_t | VkPhysicalDeviceMeshShaderPropertiesEXT::meshOutputPerVertexGranularity | VK_EXT_mesh_shader |
| uint32_t | VkPhysicalDeviceMeshShaderPropertiesEXT::meshOutputPerPrimitiveGranularity | VK_EXT_mesh_shader |
| uint32_t | VkPhysicalDeviceMeshShaderPropertiesEXT::maxPreferredTaskWorkGroupInvocations | VK_EXT_mesh_shader |
| uint32_t | VkPhysicalDeviceMeshShaderPropertiesEXT::maxPreferredMeshWorkGroupInvocations | VK_EXT_mesh_shader |
| VkBool32 | VkPhysicalDeviceMeshShaderPropertiesEXT::prefersLocalInvocationVertexOutput | VK_EXT_mesh_shader |
| VkBool32 | VkPhysicalDeviceMeshShaderPropertiesEXT::prefersLocalInvocationPrimitiveOutput | VK_EXT_mesh_shader |
| VkBool32 | VkPhysicalDeviceMeshShaderPropertiesEXT::prefersCompactVertexOutput | VK_EXT_mesh_shader |
| VkBool32 | VkPhysicalDeviceMeshShaderPropertiesEXT::prefersCompactPrimitiveOutput | VK_EXT_mesh_shader |
| uint32_t | maxTransformFeedbackStreams | VK_EXT_transform_feedback |
| uint32_t | maxTransformFeedbackBuffers | VK_EXT_transform_feedback |
| VkDeviceSize | maxTransformFeedbackBufferSize | VK_EXT_transform_feedback |
| uint32_t | maxTransformFeedbackStreamDataSize | VK_EXT_transform_feedback |
| uint32_t | maxTransformFeedbackBufferDataSize | VK_EXT_transform_feedback |
| uint32_t | maxTransformFeedbackBufferDataStride | VK_EXT_transform_feedback |
| VkBool32 | transformFeedbackQueries | VK_EXT_transform_feedback |
| VkBool32 | transformFeedbackStreamsLinesTriangles | VK_EXT_transform_feedback |
| VkBool32 | transformFeedbackRasterizationStreamSelect | VK_EXT_transform_feedback |
| VkBool32 | transformFeedbackDraw | VK_EXT_transform_feedback |
| VkExtent2D | minFragmentDensityTexelSize | fragmentDensityMap |
| VkExtent2D | maxFragmentDensityTexelSize | fragmentDensityMap |
| VkBool32 | fragmentDensityInvocations | fragmentDensityMap |
| VkBool32 | subsampledLoads | VK_EXT_fragment_density_map2 |
| VkBool32 | subsampledCoarseReconstructionEarlyAccess | VK_EXT_fragment_density_map2 |

| Type | Limit | Feature |
|---|---|---|
| uint32_t | maxSubsampledArrayLayers | VK_EXT_fragment_density_map2 |
| uint32_t | maxDescriptorSetSubsampledSamplers | VK_EXT_fragment_density_map2 |
| VkExtent2D | fragmentDensityOffsetGranularity | fragmentDensityMapOffset |
| uint32_t | maxGeometryCount | VK_NV_ray_tracing, VK_KHR_acceleration_structure |
| uint32_t | maxInstanceCount | VK_NV_ray_tracing, VK_KHR_acceleration_structure |
| uint32_t | shaderGroupHandleSize | VK_NV_ray_tracing, VK_KHR_ray_tracing_pipeline |
| uint32_t | maxShaderGroupStride | VK_NV_ray_tracing, VK_KHR_ray_tracing_pipeline |
| uint32_t | shaderGroupBaseAlignment | VK_NV_ray_tracing, VK_KHR_ray_tracing_pipeline |
| uint32_t | maxRecursionDepth | VK_NV_ray_tracing |
| uint32_t | maxTriangleCount | VK_NV_ray_tracing |
| uint32_t | maxPerStageDescriptorAccelerationStructures | VK_KHR_acceleration_structure |
| uint32_t | maxPerStageDescriptorUpdateAfterBindAccelerationStructures | VK_KHR_acceleration_structure |
| uint32_t | maxDescriptorSetAccelerationStructures | VK_NV_ray_tracing, VK_KHR_acceleration_structure |
| uint32_t | maxDescriptorSetUpdateAfterBindAccelerationStructures | VK_KHR_acceleration_structure |
| uint32_t | minAccelerationStructureScratchOffsetAlignment | VK_KHR_acceleration_structure |
| uint32_t | maxRayRecursionDepth | VK_KHR_ray_tracing_pipeline |
| uint32_t | shaderGroupHandleCaptureReplaySize | VK_KHR_ray_tracing_pipeline |
| uint32_t | maxRayDispatchInvocationCount | VK_KHR_ray_tracing_pipeline |
| uint32_t | shaderGroupHandleAlignment | VK_KHR_ray_tracing_pipeline |
| uint32_t | maxRayHitAttributeSize | VK_KHR_ray_tracing_pipeline |
| uint64_t | maxTimelineSemaphoreValueDifference | timelineSemaphore |
| uint32_t | lineSubPixelPrecisionBits | VK_KHR_line_rasterization, VK_EXT_line_rasterization |
| uint32_t | maxCustomBorderColorSamplers | VK_EXT_custom_border_color |
| VkDeviceSize | robustStorageBufferAccessSizeAlignment | VK_EXT_robustness2 |
| VkDeviceSize | robustUniformBufferAccessSizeAlignment | VK_EXT_robustness2 |

| Type | Limit | Feature |
|---|---|---|
| 2 × uint32_t | minFragmentShadingRateAttachmentTexelSize | attachmentFragmentShadingRate |
| 2 × uint32_t | maxFragmentShadingRateAttachmentTexelSize | attachmentFragmentShadingRate |
| uint32_t | maxFragmentShadingRateAttachmentTexelSizeAspectRatio | attachmentFragmentShadingRate |
| VkBool32 | primitiveFragmentShadingRateWithMultipleViewports | primitiveFragmentShadingRate |
| VkBool32 | layeredShadingRateAttachments | attachmentFragmentShadingRate |
| VkBool32 | fragmentShadingRateNonTrivialCombinerOps | pipelineFragmentShadingRate |
| 2 × uint32_t | maxFragmentSize | pipelineFragmentShadingRate |
| uint32_t | maxFragmentSizeAspectRatio | pipelineFragmentShadingRate |
| uint32_t | maxFragmentShadingRateCoverageSamples | pipelineFragmentShadingRate |
| VkSampleCountFlagBits | maxFragmentShadingRateRasterizationSamples | pipelineFragmentShadingRate |
| VkBool32 | fragmentShadingRateWithShaderDepthStencilWrites | pipelineFragmentShadingRate |
| VkBool32 | fragmentShadingRateWithSampleMask | pipelineFragmentShadingRate |
| VkBool32 | fragmentShadingRateWithShaderSampleMask | pipelineFragmentShadingRate |
| VkBool32 | fragmentShadingRateWithConservativeRasterization | pipelineFragmentShadingRate |
| VkBool32 | fragmentShadingRateWithFragmentShaderInterlock | pipelineFragmentShadingRate |
| VkBool32 | fragmentShadingRateWithCustomSampleLocations | pipelineFragmentShadingRate |
| VkBool32 | fragmentShadingRateStrictMultiplyCombiner | pipelineFragmentShadingRate |
| VkSampleCountFlagBits | maxFragmentShadingRateInvocationCount | supersampleFragmentShadingRates |
| VkBool32 | combinedImageSamplerDescriptorSingleArray | VK_EXT_descriptor_buffer |
| VkBool32 | bufferlessPushDescriptors | VK_EXT_descriptor_buffer |
| VkBool32 | allowSamplerImageViewPostSubmitCreation | VK_EXT_descriptor_buffer |
| VkDeviceSize | descriptorBufferOffsetAlignment | VK_EXT_descriptor_buffer |
| uint32_t | maxDescriptorBufferBindings | VK_EXT_descriptor_buffer |
| uint32_t | maxResourceDescriptorBufferBindings | VK_EXT_descriptor_buffer |
| uint32_t | maxSamplerDescriptorBufferBindings | VK_EXT_descriptor_buffer |
| uint32_t | maxEmbeddedImmutableSamplerBindings | VK_EXT_descriptor_buffer |
| uint32_t | maxEmbeddedImmutableSamplers | VK_EXT_descriptor_buffer |
| size_t | bufferCaptureReplayDescriptorDataSize | VK_EXT_descriptor_buffer |
| size_t | imageCaptureReplayDescriptorDataSize | VK_EXT_descriptor_buffer |
| size_t | imageViewCaptureReplayDescriptorDataSize | VK_EXT_descriptor_buffer |

| Type | Limit | Feature |
|---|---|---|
| size_t | samplerCaptureReplayDescriptorDataSize | VK_EXT_descriptor_buffer |
| size_t | accelerationStructureCaptureReplayDescriptorDataSize | VK_EXT_descriptor_buffer |
| size_t | samplerDescriptorSize | VK_EXT_descriptor_buffer |
| size_t | combinedImageSamplerDescriptorSize | VK_EXT_descriptor_buffer |
| size_t | sampledImageDescriptorSize | VK_EXT_descriptor_buffer |
| size_t | storageImageDescriptorSize | VK_EXT_descriptor_buffer |
| size_t | uniformTexelBufferDescriptorSize | VK_EXT_descriptor_buffer |
| size_t | robustUniformTexelBufferDescriptorSize | VK_EXT_descriptor_buffer |
| size_t | storageTexelBufferDescriptorSize | VK_EXT_descriptor_buffer |
| size_t | robustStorageTexelBufferDescriptorSize | VK_EXT_descriptor_buffer |
| size_t | uniformBufferDescriptorSize | VK_EXT_descriptor_buffer |
| size_t | robustUniformBufferDescriptorSize | VK_EXT_descriptor_buffer |
| size_t | storageBufferDescriptorSize | VK_EXT_descriptor_buffer |
| size_t | robustStorageBufferDescriptorSize | VK_EXT_descriptor_buffer |
| size_t | inputAttachmentDescriptorSize | VK_EXT_descriptor_buffer |
| size_t | accelerationStructureDescriptorSize | VK_EXT_descriptor_buffer |
| VkDeviceSize | maxSamplerDescriptorBufferRange | VK_EXT_descriptor_buffer |
| VkDeviceSize | maxResourceDescriptorBufferRange | VK_EXT_descriptor_buffer |
| VkDeviceSize | samplerDescriptorBufferAddressSpaceSize | VK_EXT_descriptor_buffer |
| VkDeviceSize | resourceDescriptorBufferAddressSpaceSize | VK_EXT_descriptor_buffer |
| VkDeviceSize | descriptorBufferAddressSpaceSize | VK_EXT_descriptor_buffer |
| size_t | combinedImageSamplerDensityMapDescriptorSize | VK_EXT_descriptor_buffer |
| uint32_t | maxSubpassShadingWorkgroupSizeAspectRatio | subpassShading |
| VkBool32 | graphicsPipelineLibraryFastLinking | graphicsPipelineLibrary |
| VkBool32 | graphicsPipelineLibraryIndependentInterpolationDecoration | graphicsPipelineLibrary |
| VkBool32 | triStripVertexOrderIndependentOfProvokingVertex | - |
| uint32_t | maxWeightFilterPhases | textureSampleWeighted |
| 2 × uint32_t | maxWeightFilterDimension | textureSampleWeighted |
| 2 × uint32_t | maxBlockMatchRegion | textureBlockMatch |
| 2 × uint32_t | maxBoxFilterBlockSize | textureBoxFilter |
| VkBool32 | dynamicPrimitiveTopologyUnrestricted | VK_EXT_extended_dynamic_state3 |
| uint32_t | maxOpacity2StateSubdivisionLevel | VK_EXT_opacity_micromap |
| uint32_t | maxOpacity4StateSubdivisionLevel | VK_EXT_opacity_micromap |
| uint64_t | maxDecompressionIndirectCount | VK_NV_memory_decompression |

| Type | Limit | Feature |
|---|---|---|
| 3 × uint32_t | maxWorkGroupCount | VK_HUAWEI_cluster_culling_shader |
| 3 × uint32_t | maxWorkGroupSize | VK_HUAWEI_cluster_culling_shader |
| uint32_t | maxOutputClusterCount | VK_HUAWEI_cluster_culling_shader |
| VkDeviceSize | indirectBufferOffsetAlignment | VK_HUAWEI_cluster_culling_shader |
| uint32_t | maxExecutionGraphDepth | shaderEnqueue |
| uint32_t | maxExecutionGraphShaderOutputNodes | shaderEnqueue |
| uint32_t | maxExecutionGraphShaderPayloadSize | shaderEnqueue |
| uint32_t | maxExecutionGraphShaderPayloadCount | shaderEnqueue |
| uint32_t | executionGraphDispatchAddressAlignment | shaderEnqueue |
| VkDeviceSize | extendedSparseAddressSpaceSize | sparseBinding, extendedSparseAddressSpace |
| uint32_t | supportedImageAlignmentMask | imageAlignmentControl |
| VkBool32 | separateDepthStencilAttachmentAccess | maintenance7 |
| uint32_t | maxDescriptorSetTotalUniformBuffersDynamic | maintenance7 |
| uint32_t | maxDescriptorSetTotalStorageBuffersDynamic | maintenance7 |
| uint32_t | maxDescriptorSetTotalBuffersDynamic | maintenance7 |
| uint32_t | maxDescriptorSetUpdateAfterBindTotalUniformBuffersDynamic | maintenance7 |
| uint32_t | maxDescriptorSetUpdateAfterBindTotalStorageBuffersDynamic | maintenance7 |
| uint32_t | maxDescriptorSetUpdateAfterBindTotalBuffersDynamic | maintenance7 |

*Table 65. Required Limits*

| Limit | Unsupported Limit | Supported Limit | Limit Type[1] |
|---|---|---|---|
| maxImageDimension1D | - | 4096 | min |
| maxImageDimension2D | - | 4096 | min |
| maxImageDimension3D | - | 256 | min |
| maxImageDimensionCube | - | 4096 | min |
| maxImageArrayLayers | - | 256 | min |
| maxTexelBufferElements | - | 65536 | min |
| maxUniformBufferRange | - | 16384 | min |
| maxStorageBufferRange | - | $2^{27}$ | min |
| maxPushConstantsSize | - | 128 | min |

| Limit | Unsupported Limit | Supported Limit | Limit Type[1] |
|---|---|---|---|
| maxMemoryAllocationCount | - | 4096 | min |
| maxSamplerAllocationCount | - | 4000 | min |
| bufferImageGranularity | - | 131072 | max |
| sparseAddressSpaceSize | 0 | $2^{31}$ | min |
| maxBoundDescriptorSets | - | 4 | min |
| maxPerStageDescriptorSamplers | - | 16 | min |
| maxPerStageDescriptorUniformBuffers | - | 12 | min |
| maxPerStageDescriptorStorageBuffers | - | 4 | min |
| maxPerStageDescriptorSampledImages | - | 16 | min |
| maxPerStageDescriptorStorageImages | - | 4 | min |
| maxPerStageDescriptorInputAttachments | - | 4 | min |
| maxPerStageResources | - | 128 [2] | min |
| maxDescriptorSetSamplers | - | 96 [8] | min, $n \times$ PerStage |
| maxDescriptorSetUniformBuffers | - | 72 [8] | min, $n \times$ PerStage |
| maxDescriptorSetUniformBuffersDynamic | - | 8 | min |
| maxDescriptorSetStorageBuffers | - | 24 [8] | min, $n \times$ PerStage |
| maxDescriptorSetStorageBuffersDynamic | - | 4 | min |
| maxDescriptorSetTotalUniformBuffersDynamic | - | maxDescriptorSetUniformBuffersDynamic | min |
| maxDescriptorSetTotalStorageBuffersDynamic | - | maxDescriptorSetStorageBuffersDynamic | min |
| maxDescriptorSetTotalBuffersDynamic | - | maxDescriptorSetUniformBuffersDynamic + maxDescriptorSetStorageBuffersDynamic | min |
| maxDescriptorSetSampledImages | - | 96 [8] | min, $n \times$ PerStage |
| maxDescriptorSetStorageImages | - | 24 [8] | min, $n \times$ PerStage |
| maxDescriptorSetInputAttachments | - | 4 | min |
| maxVertexInputAttributes | - | 16 | min |

| Limit | Unsupported Limit | Supported Limit | Limit Type[1] |
|---|---|---|---|
| maxVertexInputBindings | - | 16 [10] | min |
| maxVertexInputAttributeOffset | - | 2047 | min |
| maxVertexInputBindingStride | - | 2048 | min |
| maxVertexOutputComponents | - | 64 | min |
| maxTessellationGenerationLevel | 0 | 64 | min |
| maxTessellationPatchSize | 0 | 32 | min |
| maxTessellationControlPerVertexInputComponents | 0 | 64 | min |
| maxTessellationControlPerVertexOutputComponents | 0 | 64 | min |
| maxTessellationControlPerPatchOutputComponents | 0 | 120 | min |
| maxTessellationControlTotalOutputComponents | 0 | 2048 | min |
| maxTessellationEvaluationInputComponents | 0 | 64 | min |
| maxTessellationEvaluationOutputComponents | 0 | 64 | min |
| maxGeometryShaderInvocations | 0 | 32 | min |
| maxGeometryInputComponents | 0 | 64 | min |
| maxGeometryOutputComponents | 0 | 64 | min |
| maxGeometryOutputVertices | 0 | 256 | min |
| maxGeometryTotalOutputComponents | 0 | 1024 | min |
| maxFragmentInputComponents | - | 64 | min |
| maxFragmentOutputAttachments | - | 4 | min |
| maxFragmentDualSrcAttachments | 0 | 1 | min |
| maxFragmentCombinedOutputResources | - | 4 | min |
| maxComputeSharedMemorySize | - | 16384 | min |
| maxComputeWorkGroupCount | - | (65535,65535,65535) | min |
| maxComputeWorkGroupInvocations | - | 128 | min |
| maxComputeWorkGroupSize | - | (128,128,64) | min |
| subPixelPrecisionBits | - | 4 | min |
| subTexelPrecisionBits | - | 4 | min |
| mipmapPrecisionBits | - | 4 | min |
| maxDrawIndexedIndexValue | $2^{24}$-1 | $2^{32}$-1 | min |
| maxDrawIndirectCount | 1 | $2^{16}$-1 | min |
| maxSamplerLodBias | - | 2 | min |
| maxSamplerAnisotropy | 1 | 16 | min |

| Limit | Unsupported Limit | Supported Limit | Limit Type[1] |
|---|---|---|---|
| maxViewports | 1 | 16 | min |
| maxViewportDimensions | - | (4096,4096) [3] | min |
| viewportBoundsRange | - | (-8192,8191) [4] | (max,min) |
| viewportSubPixelBits | - | 0 | min |
| minMemoryMapAlignment | - | 64 | min |
| minTexelBufferOffsetAlignment | - | 256 | max |
| minUniformBufferOffsetAlignment | - | 256 | max |
| minStorageBufferOffsetAlignment | - | 256 | max |
| minTexelOffset | - | -8 | max |
| maxTexelOffset | - | 7 | min |
| minTexelGatherOffset | 0 | -8 | max |
| maxTexelGatherOffset | 0 | 7 | min |
| minInterpolationOffset | 0.0 | -0.5 [5] | max |
| maxInterpolationOffset | 0.0 | 0.5 - (1 ULP) [5] | min |
| subPixelInterpolationOffsetBits | 0 | 4 [5] | min |
| maxFramebufferWidth | - | 4096 | min |
| maxFramebufferHeight | - | 4096 | min |
| maxFramebufferLayers | - | 256 | min |
| framebufferColorSampleCounts | - | (VK_SAMPLE_COUNT_1_BIT \| VK_SAMPLE_COUNT_4_BIT) | min |
| framebufferDepthSampleCounts | - | (VK_SAMPLE_COUNT_1_BIT \| VK_SAMPLE_COUNT_4_BIT) | min |
| framebufferStencilSampleCounts | - | (VK_SAMPLE_COUNT_1_BIT \| VK_SAMPLE_COUNT_4_BIT) | min |
| framebufferNoAttachmentsSampleCounts | - | (VK_SAMPLE_COUNT_1_BIT \| VK_SAMPLE_COUNT_4_BIT) | min |
| maxColorAttachments | - | 4 | min |

| Limit | Unsupported Limit | Supported Limit | Limit Type[1] |
|---|---|---|---|
| sampledImageColorSampleCounts | - | (VK_SAMPLE_COUNT_1_BIT \| VK_SAMPLE_COUNT_4_BIT) | min |
| sampledImageIntegerSampleCounts | - | VK_SAMPLE_COUNT_1_BIT | min |
| sampledImageDepthSampleCounts | - | (VK_SAMPLE_COUNT_1_BIT \| VK_SAMPLE_COUNT_4_BIT) | min |
| sampledImageStencilSampleCounts | - | (VK_SAMPLE_COUNT_1_BIT \| VK_SAMPLE_COUNT_4_BIT) | min |
| storageImageSampleCounts | VK_SAMPLE_COUNT_1_BIT | (VK_SAMPLE_COUNT_1_BIT \| VK_SAMPLE_COUNT_4_BIT) | min |
| maxSampleMaskWords | - | 1 | min |
| timestampComputeAndGraphics | - | - | implementation-dependent |
| timestampPeriod | - | - | duration |
| maxClipDistances | 0 | 8 | min |
| maxCullDistances | 0 | 8 | min |
| maxCombinedClipAndCullDistances | 0 | 8 | min |
| discreteQueuePriorities | - | 2 | min |
| pointSizeRange | (1.0,1.0) | (1.0,64.0 - ULP) [6] | (max,min) |
| lineWidthRange | (1.0,1.0) | (1.0,8.0 - ULP) [7] | (max,min) |
| pointSizeGranularity | 0.0 | 1.0 [6] | max, fixed point increment |
| lineWidthGranularity | 0.0 | 1.0 [7] | max, fixed point increment |
| strictLines | - | - | implementation-dependent |
| standardSampleLocations | - | - | implementation-dependent |

| Limit | Unsupported Limit | Supported Limit | Limit Type[1] |
|---|---|---|---|
| optimalBufferCopyOffsetAlignment | - | - | recommendation |
| optimalBufferCopyRowPitchAlignment | - | - | recommendation |
| nonCoherentAtomSize | - | 256 | max |
| maxPushDescriptors | - | 32 | min |
| maxMultiviewViewCount | - | 6 | min |
| maxMultiviewInstanceIndex | - | $2^{27}$-1 | min |
| maxDiscardRectangles | 0 | 4 | min |
| sampleLocationSampleCounts | - | VK_SAMPLE_COUNT_4_BIT | min |
| maxSampleLocationGridSize | - | (1,1) | min |
| sampleLocationCoordinateRange | - | (0.0, 0.9375) | (max,min) |
| sampleLocationSubPixelBits | - | 4 | min |
| variableSampleLocations | - | false | implementation-dependent |
| nativeUnalignedPerformance | - | false | implementation-dependent |
| minImportedHostPointerAlignment | - | 65536 | max |
| perViewPositionAllComponents | - | - | implementation-dependent |
| filterMinmaxSingleComponentFormats | - | - | implementation-dependent |
| filterMinmaxImageComponentMapping | - | - | implementation-dependent |
| advancedBlendMaxColorAttachments | - | 1 | min |
| advancedBlendIndependentBlend | - | false | implementation-dependent |
| advancedBlendNonPremultipliedSrcColor | - | false | implementation-dependent |
| advancedBlendNonPremultipliedDstColor | - | false | implementation-dependent |
| advancedBlendCorrelatedOverlap | - | false | implementation-dependent |
| advancedBlendAllOperations | - | false | implementation-dependent |

| Limit | Unsupported Limit | Supported Limit | Limit Type[1] |
|---|---|---|---|
| maxPerSetDescriptors | - | 1024 | min |
| maxMemoryAllocationSize | - | $2^{30}$ | min |
| maxBufferSize | - | $2^{30}$ | min |
| primitiveOverestimationSize | - | 0.0 | min |
| maxExtraPrimitiveOverestimationSize | - | 0.0 | min |
| extraPrimitiveOverestimationSizeGranularity | - | 0.0 | min |
| primitiveUnderestimation | - | false | implementation-dependent |
| conservativePointAndLineRasterization | - | false | implementation-dependent |
| degenerateTrianglesRasterized | - | false | implementation-dependent |
| degenerateLinesRasterized | - | false | implementation-dependent |
| fullyCoveredFragmentShaderInputVariable | - | false | implementation-dependent |
| conservativeRasterizationPostDepthCoverage | - | false | implementation-dependent |
| maxUpdateAfterBindDescriptorsInAllPools | 0 | 500000 | min |
| shaderUniformBufferArrayNonUniformIndexingNative | - | false | implementation-dependent |
| shaderSampledImageArrayNonUniformIndexingNative | - | false | implementation-dependent |
| shaderStorageBufferArrayNonUniformIndexingNative | - | false | implementation-dependent |
| shaderStorageImageArrayNonUniformIndexingNative | - | false | implementation-dependent |
| shaderInputAttachmentArrayNonUniformIndexingNative | - | false | implementation-dependent |
| maxPerStageDescriptorUpdateAfterBindSamplers | 0 [9] | 500000 [9] | min |
| maxPerStageDescriptorUpdateAfterBindUniformBuffers | 0 [9] | 12 [9] | min |
| maxPerStageDescriptorUpdateAfterBindStorageBuffers | 0 [9] | 500000 [9] | min |
| maxPerStageDescriptorUpdateAfterBindSampledImages | 0 [9] | 500000 [9] | min |
| maxPerStageDescriptorUpdateAfterBindStorageImages | 0 [9] | 500000 [9] | min |

| Limit | Unsupported Limit | Supported Limit | Limit Type[1] |
|---|---|---|---|
| maxPerStageDescriptorUpdateAfterBindInputAttachments | 0 [9] | 4 [9] | min |
| maxPerStageUpdateAfterBindResources | 0 [9] | 500000 [9] | min |
| maxDescriptorSetUpdateAfterBindSamplers | 0 [9] | 500000 [9] | min |
| maxDescriptorSetUpdateAfterBindUniformBuffers | 0 [9] | 72 [8] [9] | min, $n \times$ PerStage |
| maxDescriptorSetUpdateAfterBindUniformBuffersDynamic | 0 [9] | 8 [9] | min |
| maxDescriptorSetUpdateAfterBindStorageBuffers | 0 [9] | 500000 [9] | min |
| maxDescriptorSetUpdateAfterBindStorageBuffersDynamic | 0 [9] | 4 [9] | min |
| maxDescriptorSetUpdateAfterBindTotalUniformBuffersDynamic | 0 [9] | maxDescriptorSetUpdateAfterBindUniformBuffersDynamic | min |
| maxDescriptorSetUpdateAfterBindTotalStorageBuffersDynamic | 0 [9] | maxDescriptorSetUpdateAfterBindStorageBuffersDynamic | min |
| maxDescriptorSetUpdateAfterBindTotalBuffersDynamic | 0 [9] | maxDescriptorSetUpdateAfterBindUniformBuffersDynamic + maxDescriptorSetUpdateAfterBindStorageBuffersDynamic | min |
| maxDescriptorSetUpdateAfterBindSampledImages | 0 [9] | 500000 [9] | min |
| maxDescriptorSetUpdateAfterBindStorageImages | 0 [9] | 500000 [9] | min |
| maxDescriptorSetUpdateAfterBindInputAttachments | 0 [9] | 4 [9] | min |
| maxInlineUniformBlockSize | - | 256 | min |
| maxPerStageDescriptorInlineUniformBlocks | - | 4 | min |
| maxPerStageDescriptorUpdateAfterBindInlineUniformBlocks | - | 4 | min |
| maxDescriptorSetInlineUniformBlocks | - | 4 | min |
| maxDescriptorSetUpdateAfterBindInlineUniformBlocks | - | 4 | min |
| maxVertexAttribDivisor | - | $2^{16}$-1 | min |
| VkPhysicalDeviceMeshShaderPropertiesNV::maxDrawMeshTasksCount | - | $2^{16}$-1 | min |
| VkPhysicalDeviceMeshShaderPropertiesNV::maxTaskWorkGroupInvocations | - | 32 | min |
| VkPhysicalDeviceMeshShaderPropertiesNV::maxTaskWorkGroupSize | - | (32,1,1) | min |

| Limit | Unsupported Limit | Supported Limit | Limit Type[1] |
|---|---|---|---|
| VkPhysicalDeviceMeshShaderPropertiesNV::maxTaskTotalMemorySize | - | 16384 | min |
| VkPhysicalDeviceMeshShaderPropertiesNV::maxTaskOutputCount | - | $2^{16}$-1 | min |
| VkPhysicalDeviceMeshShaderPropertiesNV::maxMeshWorkGroupInvocations | - | 32 | min |
| VkPhysicalDeviceMeshShaderPropertiesNV::maxMeshWorkGroupSize | - | (32,1,1) | min |
| VkPhysicalDeviceMeshShaderPropertiesNV::maxMeshTotalMemorySize | - | 16384 | min |
| VkPhysicalDeviceMeshShaderPropertiesNV::maxMeshOutputVertices | - | 256 | min |
| VkPhysicalDeviceMeshShaderPropertiesNV::maxMeshOutputPrimitives | - | 256 | min |
| VkPhysicalDeviceMeshShaderPropertiesNV::maxMeshMultiviewViewCount | - | 1 | min |
| VkPhysicalDeviceMeshShaderPropertiesNV::meshOutputPerVertexGranularity | - | - | implementation-dependent |
| VkPhysicalDeviceMeshShaderPropertiesNV::meshOutputPerPrimitiveGranularity | - | - | implementation-dependent |
| VkPhysicalDeviceMeshShaderPropertiesEXT::maxTaskWorkGroupTotalCount | - | 2^22 | min |
| VkPhysicalDeviceMeshShaderPropertiesEXT::maxTaskWorkGroupCount | - | (65535,65535,65535) | min |
| VkPhysicalDeviceMeshShaderPropertiesEXT::maxTaskWorkGroupInvocations | - | 128 | min |
| VkPhysicalDeviceMeshShaderPropertiesEXT::maxTaskWorkGroupSize | - | (128,128,128) | min |
| VkPhysicalDeviceMeshShaderPropertiesEXT::maxTaskPayloadSize | - | 16384 | min |
| VkPhysicalDeviceMeshShaderPropertiesEXT::maxTaskSharedMemorySize | - | 32768 | min |
| VkPhysicalDeviceMeshShaderPropertiesEXT::maxTaskPayloadAndSharedMemorySize | - | 32768 | min |
| VkPhysicalDeviceMeshShaderPropertiesEXT::maxMeshWorkGroupTotalCount | - | 2^22 | min |
| VkPhysicalDeviceMeshShaderPropertiesEXT::maxMeshWorkGroupCount | - | (65535,65535,65535) | min |
| VkPhysicalDeviceMeshShaderPropertiesEXT::maxMeshWorkGroupInvocations | - | 128 | min |

| Limit | Unsupported Limit | Supported Limit | Limit Type[1] |
|---|---|---|---|
| VkPhysicalDeviceMeshShaderPropertiesEXT::maxMeshWorkGroupSize | - | (128,128,128) | min |
| VkPhysicalDeviceMeshShaderPropertiesEXT::maxMeshSharedMemorySize | - | 28672 | min |
| VkPhysicalDeviceMeshShaderPropertiesEXT::maxMeshPayloadAndSharedMemorySize | - | 28672 | min |
| VkPhysicalDeviceMeshShaderPropertiesEXT::maxMeshOutputMemorySize | - | 32768 | min |
| VkPhysicalDeviceMeshShaderPropertiesEXT::maxMeshPayloadAndOutputMemorySize | - | 48128 | min |
| VkPhysicalDeviceMeshShaderPropertiesEXT::maxMeshOutputComponents | - | 128 | min |
| VkPhysicalDeviceMeshShaderPropertiesEXT::maxMeshOutputVertices | - | 256 | min |
| VkPhysicalDeviceMeshShaderPropertiesEXT::maxMeshOutputPrimitives | - | 256 | min |
| VkPhysicalDeviceMeshShaderPropertiesEXT::maxMeshOutputLayers | - | 8 | min |
| VkPhysicalDeviceMeshShaderPropertiesEXT::maxMeshMultiviewViewCount | - | 1 | min |
| VkPhysicalDeviceMeshShaderPropertiesEXT::meshOutputPerVertexGranularity | 0 | 32 | max |
| VkPhysicalDeviceMeshShaderPropertiesEXT::meshOutputPerPrimitiveGranularity | 0 | 32 | max |
| VkPhysicalDeviceMeshShaderPropertiesEXT::maxPreferredTaskWorkGroupInvocations | - | - | implementation-dependent |
| VkPhysicalDeviceMeshShaderPropertiesEXT::maxPreferredMeshWorkGroupInvocations | - | - | implementation-dependent |
| VkPhysicalDeviceMeshShaderPropertiesEXT::prefersLocalInvocationVertexOutput | - | - | implementation-dependent |
| VkPhysicalDeviceMeshShaderPropertiesEXT::prefersLocalInvocationPrimitiveOutput | - | - | implementation-dependent |
| VkPhysicalDeviceMeshShaderPropertiesEXT::prefersCompactVertexOutput | - | - | implementation-dependent |
| VkPhysicalDeviceMeshShaderPropertiesEXT::prefersCompactPrimitiveOutput | - | - | implementation-dependent |
| maxTransformFeedbackStreams | - | 1 | min |
| maxTransformFeedbackBuffers | - | 1 | min |
| maxTransformFeedbackBufferSize | - | $2^{27}$ | min |

| Limit | Unsupported Limit | Supported Limit | Limit Type[1] |
|---|---|---|---|
| maxTransformFeedbackStreamDataSize | - | 512 | min |
| maxTransformFeedbackBufferDataSize | - | 512 | min |
| maxTransformFeedbackBufferDataStride | - | 512 | min |
| transformFeedbackQueries | - | false | implementation-dependent |
| transformFeedbackStreamsLinesTriangles | - | false | implementation-dependent |
| transformFeedbackRasterizationStreamSelect | - | false | implementation-dependent |
| transformFeedbackDraw | - | false | implementation-dependent |
| minFragmentDensityTexelSize | - | (1,1) | min |
| maxFragmentDensityTexelSize | - | (1,1) | min |
| fragmentDensityInvocations | - | - | implementation-dependent |
| subsampledLoads | true | false | implementation-dependent |
| subsampledCoarseReconstructionEarlyAccess | false | false | implementation-dependent |
| maxSubsampledArrayLayers | 2 | 2 | min |
| maxDescriptorSetSubsampledSamplers | 1 | 1 | min |
| fragmentDensityOffsetGranularity | - | (1024,1024) | max |
| VkPhysicalDeviceRayTracingPropertiesNV::shaderGroupHandleSize | - | 16 | min |
| VkPhysicalDeviceRayTracingPropertiesNV::maxRecursionDepth | - | 31 | min |
| VkPhysicalDeviceRayTracingPipelinePropertiesKHR::shaderGroupHandleSize | - | 32 | exact |
| VkPhysicalDeviceRayTracingPipelinePropertiesKHR::maxRayRecursionDepth | - | 1 | min |
| maxShaderGroupStride | - | 4096 | min |
| shaderGroupBaseAlignment | - | 64 | max |
| maxGeometryCount | - | $2^{24}$-1 | min |
| maxInstanceCount | - | $2^{24}$-1 | min |
| maxTriangleCount | - | $2^{29}$-1 | min |
| maxPrimitiveCount | - | $2^{29}$-1 | min |

| Limit | Unsupported Limit | Supported Limit | Limit Type[1] |
|---|---|---|---|
| maxPerStageDescriptorAccelerationStructures | - | 16 | min |
| maxPerStageDescriptorUpdateAfterBindAccelerationStructures | - | 500000 [9] | min |
| maxDescriptorSetAccelerationStructures | - | 16 | min |
| maxDescriptorSetUpdateAfterBindAccelerationStructures | - | 500000 [9] | min |
| minAccelerationStructureScratchOffsetAlignment | - | 256 | max |
| shaderGroupHandleCaptureReplaySize | - | 64 | max |
| maxRayDispatchInvocationCount | - | $2^{30}$ | min |
| shaderGroupHandleAlignment | - | 32 | max |
| maxRayHitAttributeSize | - | 32 | min |
| maxTimelineSemaphoreValueDifference | - | $2^{31}$-1 | min |
| lineSubPixelPrecisionBits | - | 4 | min |
| maxGraphicsShaderGroupCount | - | $2^{12}$ | min |
| maxIndirectSequenceCount | - | $2^{20}$ | min |
| maxIndirectCommandsTokenCount | - | 16 | min |
| maxIndirectCommandsStreamCount | - | 16 | min |
| maxIndirectCommandsTokenOffset | - | 2047 | min |
| maxIndirectCommandsStreamStride | - | 2048 | min |
| minSequencesCountBufferOffsetAlignment | - | 256 | max |
| minSequencesIndexBufferOffsetAlignment | - | 256 | max |
| minIndirectCommandsBufferOffsetAlignment | - | 256 | max |
| maxCustomBorderColorSamplers | - | 32 | min |
| robustStorageBufferAccessSizeAlignment | - | 4 | max |
| robustUniformBufferAccessSizeAlignment | - | 256 | max |
| minFragmentShadingRateAttachmentTexelSize | (0,0) | (32,32) | max |
| maxFragmentShadingRateAttachmentTexelSize | (0,0) | (8,8) | min |
| maxFragmentShadingRateAttachmentTexelSizeAspectRatio | 0 | 1 | min |
| primitiveFragmentShadingRateWithMultipleViewports | false | false | implementation-dependent |
| layeredShadingRateAttachments | false | false | implementation-dependent |
| fragmentShadingRateNonTrivialCombinerOps | - | false | implementation-dependent |

| Limit | Unsupported Limit | Supported Limit | Limit Type[1] |
|---|---|---|---|
| maxFragmentSize | - | (2,2) | min |
| maxFragmentSizeAspectRatio | - | 2 | min |
| maxFragmentShadingRateCoverageSamples | - | 16 | min |
| maxFragmentShadingRateRasterizationSamples | - | VK_SAMPLE_COUNT_4_BIT | min |
| fragmentShadingRateWithShaderDepthStencilWrites | - | false | implementation-dependent |
| fragmentShadingRateWithSampleMask | - | false | implementation-dependent |
| fragmentShadingRateWithShaderSampleMask | - | false | implementation-dependent |
| fragmentShadingRateWithConservativeRasterization | - | false | implementation-dependent |
| fragmentShadingRateWithFragmentShaderInterlock | - | false | implementation-dependent |
| fragmentShadingRateWithCustomSampleLocations | - | false | implementation-dependent |
| fragmentShadingRateStrictMultiplyCombiner | - | false | implementation-dependent |
| maxFragmentShadingRateInvocationCount | - | VK_SAMPLE_COUNT_4_BIT | min |
| combinedImageSamplerDescriptorSingleArray | - | false | implementation-dependent |
| bufferlessPushDescriptors | - | false | implementation-dependent |
| allowSamplerImageViewPostSubmitCreation | - | false | implementation-dependent |
| descriptorBufferOffsetAlignment | - | 256 | max |
| maxDescriptorBufferBindings | - | 3 | min |
| maxResourceDescriptorBufferBindings | - | 1 | min |
| maxSamplerDescriptorBufferBindings | - | 1 | min |
| maxEmbeddedImmutableSamplerBindings | - | 1 | min |
| maxEmbeddedImmutableSamplers | - | 2032 | min |
| bufferCaptureReplayDescriptorDataSize | - | 64 | max |
| imageCaptureReplayDescriptorDataSize | - | 64 | max |
| imageViewCaptureReplayDescriptorDataSize | - | 64 | max |

| Limit | Unsupported Limit | Supported Limit | Limit Type[1] |
|---|---|---|---|
| samplerCaptureReplayDescriptorDataSize | - | 64 | max |
| accelerationStructureCaptureReplayDescriptorDataSize | - | 64 | max |
| samplerDescriptorSize | - | 256 | max |
| combinedImageSamplerDescriptorSize | - | 256 | max |
| sampledImageDescriptorSize | - | 256 | max |
| storageImageDescriptorSize | - | 256 | max |
| uniformTexelBufferDescriptorSize | - | 256 | max |
| robustUniformTexelBufferDescriptorSize | - | 256 | max |
| storageTexelBufferDescriptorSize | - | 256 | max |
| robustStorageTexelBufferDescriptorSize | - | 256 | max |
| uniformBufferDescriptorSize | - | 256 | max |
| robustUniformBufferDescriptorSize | - | 256 | max |
| storageBufferDescriptorSize | - | 256 | max |
| robustStorageBufferDescriptorSize | - | 256 | max |
| inputAttachmentDescriptorSize | - | 256 | max |
| accelerationStructureDescriptorSize | - | 256 | max |
| maxSamplerDescriptorBufferRange | - | $2^{11}$ × samplerDescriptorSize | min |
| maxResourceDescriptorBufferRange | - | ($2^{20}$ - $2^{15}$) × maxResourceDescriptorSize [12] | min |
| samplerDescriptorBufferAddressSpaceSize | - | $2^{27}$ | min |
| resourceDescriptorBufferAddressSpaceSize | - | $2^{27}$ | min |
| descriptorBufferAddressSpaceSize | - | $2^{27}$ | min |
| combinedImageSamplerDensityMapDescriptorSize | - | 256 | max |
| maxSubpassShadingWorkgroupSizeAspectRatio | 0 | 1 | min |
| maxMultiDrawCount | - | 1024 | min |
| maxCommandBufferNestingLevel | - | 1 | min |
| graphicsPipelineLibraryFastLinking | - | false | implementation-dependent |
| graphicsPipelineLibraryIndependentInterpolationDecoration | - | false | implementation-dependent |

| Limit | Unsupported Limit | Supported Limit | Limit Type[1] |
|---|---|---|---|
| triStripVertexOrderIndependentOfProvokingVertex | - | false | implementation-dependent |
| maxWeightFilterPhases | - | 1024 | min |
| maxWeightFilterDimension | - | (64,64) | min |
| maxBlockMatchRegion | - | (64,64) | min |
| maxBoxFilterBlockSize | - | (64,64) | min |
| dynamicPrimitiveTopologyUnrestricted | - | - | implementation-dependent |
| maxOpacity2StateSubdivisionLevel | - | 3 | min |
| maxOpacity4StateSubdivisionLevel | - | 3 | min |
| maxDecompressionIndirectCount | 1 | $2^{16}$-1 | min |
| maxWorkGroupCount | - | (65536,1,1) | min |
| maxWorkGroupSize | - | (32,1,1) | min |
| maxOutputClusterCount | - | 1024 | min |
| indirectBufferOffsetAlignment | - | - | implementation-dependent |
| maxExecutionGraphDepth | - | 32 | min |
| maxExecutionGraphShaderOutputNodes | - | 256 | min |
| maxExecutionGraphShaderPayloadSize | - | 32768 | min |
| maxExecutionGraphShaderPayloadCount | - | 256 | min |
| executionGraphDispatchAddressAlignment | - | 4 | max |
| extendedSparseAddressSpaceSize | 0 | sparseAddressSpaceSize | min |
| renderPassStripeGranularity | - | (64,64) | max |
| maxRenderPassStripes | - | 32 | min |
| minPlacedMemoryMapAlignment | - | 65536 | max |
| supportedImageAlignmentMask | - | 1 | min |
| separateDepthStencilAttachmentAccess | false | - | implementation-dependent |

**1**

The **Limit Type** column specifies the limit is either the minimum limit all implementations **must** support, the maximum limit all implementations **must** support, or the exact value all implementations **must** support. For bitmasks a minimum limit is the least bits all implementations **must** set, but they **may** have additional bits set beyond this minimum.

**2**

The `maxPerStageResources` **must** be at least the smallest of the following:

- the sum of the `maxPerStageDescriptorUniformBuffers`, `maxPerStageDescriptorStorageBuffers`, `maxPerStageDescriptorSampledImages`, `maxPerStageDescriptorStorageImages`, `maxPerStageDescriptorInputAttachments`, `maxColorAttachments` limits, or
- 128.

  It **may** not be possible to reach this limit in every stage.

**3**

See `maxViewportDimensions` for the **required** relationship to other limits.

**4**

See `viewportBoundsRange` for the **required** relationship to other limits.

**5**

The values `minInterpolationOffset` and `maxInterpolationOffset` describe the closed interval of supported interpolation offsets: [`minInterpolationOffset`, `maxInterpolationOffset`]. The ULP is determined by `subPixelInterpolationOffsetBits`. If `subPixelInterpolationOffsetBits` is 4, this provides increments of $(1/2^4) = 0.0625$, and thus the range of supported interpolation offsets would be [-0.5, 0.4375].

**6**

The point size ULP is determined by `pointSizeGranularity`. If the `pointSizeGranularity` is 0.125, the range of supported point sizes **must** be at least [1.0, 63.875].

**7**

The line width ULP is determined by `lineWidthGranularity`. If the `lineWidthGranularity` is 0.0625, the range of supported line widths **must** be at least [1.0, 7.9375].

**8**

The minimum `maxDescriptorSet*` limit is $n$ times the corresponding *specification* minimum `maxPerStageDescriptor*` limit, where $n$ is the number of shader stages supported by the VkPhysicalDevice. If all shader stages are supported, $n = 6$ (vertex, tessellation control, tessellation evaluation, geometry, fragment, compute).

**9**

The `UpdateAfterBind` descriptor limits **must** each be greater than or equal to the corresponding `non`-UpdateAfterBind limit.

**10**

If the `VK_KHR_portability_subset` extension is enabled, the required minimum value of `maxVertexInputBindings` is `8`.

**12**

`maxResourceDescriptorSize` is defined as the maximum value of `storageImageDescriptorSize`, `sampledImageDescriptorSize`, `robustUniformTexelBufferDescriptorSize`,

robustStorageTexelBufferDescriptorSize, robustUniformBufferDescriptorSize, robustStorageBufferDescriptorSize, inputAttachmentDescriptorSize, and accelerationStructureDescriptorSize.

## 47.2. Additional Multisampling Capabilities

To query additional multisampling capabilities which **may** be supported for a specific sample count, beyond the minimum capabilities described for Limits above, call:

```
// Provided by VK_EXT_sample_locations
void vkGetPhysicalDeviceMultisamplePropertiesEXT(
    VkPhysicalDevice                            physicalDevice,
    VkSampleCountFlagBits                       samples,
    VkMultisamplePropertiesEXT*                 pMultisampleProperties);
```

- physicalDevice is the physical device from which to query the additional multisampling capabilities.
- samples is a VkSampleCountFlagBits value specifying the sample count to query capabilities for.
- pMultisampleProperties is a pointer to a VkMultisamplePropertiesEXT structure in which information about additional multisampling capabilities specific to the sample count is returned.

### Valid Usage (Implicit)

- VUID-vkGetPhysicalDeviceMultisamplePropertiesEXT-physicalDevice-parameter
  physicalDevice **must** be a valid VkPhysicalDevice handle
- VUID-vkGetPhysicalDeviceMultisamplePropertiesEXT-samples-parameter
  samples **must** be a valid VkSampleCountFlagBits value
- VUID-vkGetPhysicalDeviceMultisamplePropertiesEXT-pMultisampleProperties-parameter
  pMultisampleProperties **must** be a valid pointer to a VkMultisamplePropertiesEXT structure

The VkMultisamplePropertiesEXT structure is defined as

```
// Provided by VK_EXT_sample_locations
typedef struct VkMultisamplePropertiesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkExtent2D         maxSampleLocationGridSize;
} VkMultisamplePropertiesEXT;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.

- `maxSampleLocationGridSize` is the maximum size of the pixel grid in which sample locations **can** vary.

### Valid Usage (Implicit)

- VUID-VkMultisamplePropertiesEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_MULTISAMPLE_PROPERTIES_EXT`
- VUID-VkMultisamplePropertiesEXT-pNext-pNext
  `pNext` **must** be `NULL`

If the sample count for which additional multisampling capabilities are requested using `vkGetPhysicalDeviceMultisamplePropertiesEXT` is set in `sampleLocationSampleCounts` the `width` and `height` members of `VkMultisamplePropertiesEXT`::`maxSampleLocationGridSize` **must** be greater than or equal to the corresponding members of `maxSampleLocationGridSize`, respectively, otherwise both members **must** be `0`.

# Chapter 48. Formats

Supported buffer and image formats **may** vary across implementations. A minimum set of format features are guaranteed, but others **must** be explicitly queried before use to ensure they are supported by the implementation.

The features for the set of formats (VkFormat) supported by the implementation are queried individually using the vkGetPhysicalDeviceFormatProperties command.

## 48.1. Format Definition

The following image formats **can** be passed to, and **may** be returned from Vulkan commands. The memory required to store each format is discussed with that format, and also summarized in the Representation and Texel Block Size section and the Compatible formats table.

```
// Provided by VK_VERSION_1_0
typedef enum VkFormat {
    VK_FORMAT_UNDEFINED = 0,
    VK_FORMAT_R4G4_UNORM_PACK8 = 1,
    VK_FORMAT_R4G4B4A4_UNORM_PACK16 = 2,
    VK_FORMAT_B4G4R4A4_UNORM_PACK16 = 3,
    VK_FORMAT_R5G6B5_UNORM_PACK16 = 4,
    VK_FORMAT_B5G6R5_UNORM_PACK16 = 5,
    VK_FORMAT_R5G5B5A1_UNORM_PACK16 = 6,
    VK_FORMAT_B5G5R5A1_UNORM_PACK16 = 7,
    VK_FORMAT_A1R5G5B5_UNORM_PACK16 = 8,
    VK_FORMAT_R8_UNORM = 9,
    VK_FORMAT_R8_SNORM = 10,
    VK_FORMAT_R8_USCALED = 11,
    VK_FORMAT_R8_SSCALED = 12,
    VK_FORMAT_R8_UINT = 13,
    VK_FORMAT_R8_SINT = 14,
    VK_FORMAT_R8_SRGB = 15,
    VK_FORMAT_R8G8_UNORM = 16,
    VK_FORMAT_R8G8_SNORM = 17,
    VK_FORMAT_R8G8_USCALED = 18,
    VK_FORMAT_R8G8_SSCALED = 19,
    VK_FORMAT_R8G8_UINT = 20,
    VK_FORMAT_R8G8_SINT = 21,
    VK_FORMAT_R8G8_SRGB = 22,
    VK_FORMAT_R8G8B8_UNORM = 23,
    VK_FORMAT_R8G8B8_SNORM = 24,
    VK_FORMAT_R8G8B8_USCALED = 25,
    VK_FORMAT_R8G8B8_SSCALED = 26,
    VK_FORMAT_R8G8B8_UINT = 27,
    VK_FORMAT_R8G8B8_SINT = 28,
    VK_FORMAT_R8G8B8_SRGB = 29,
    VK_FORMAT_B8G8R8_UNORM = 30,
    VK_FORMAT_B8G8R8_SNORM = 31,
```

```
    VK_FORMAT_B8G8R8_USCALED = 32,
    VK_FORMAT_B8G8R8_SSCALED = 33,
    VK_FORMAT_B8G8R8_UINT = 34,
    VK_FORMAT_B8G8R8_SINT = 35,
    VK_FORMAT_B8G8R8_SRGB = 36,
    VK_FORMAT_R8G8B8A8_UNORM = 37,
    VK_FORMAT_R8G8B8A8_SNORM = 38,
    VK_FORMAT_R8G8B8A8_USCALED = 39,
    VK_FORMAT_R8G8B8A8_SSCALED = 40,
    VK_FORMAT_R8G8B8A8_UINT = 41,
    VK_FORMAT_R8G8B8A8_SINT = 42,
    VK_FORMAT_R8G8B8A8_SRGB = 43,
    VK_FORMAT_B8G8R8A8_UNORM = 44,
    VK_FORMAT_B8G8R8A8_SNORM = 45,
    VK_FORMAT_B8G8R8A8_USCALED = 46,
    VK_FORMAT_B8G8R8A8_SSCALED = 47,
    VK_FORMAT_B8G8R8A8_UINT = 48,
    VK_FORMAT_B8G8R8A8_SINT = 49,
    VK_FORMAT_B8G8R8A8_SRGB = 50,
    VK_FORMAT_A8B8G8R8_UNORM_PACK32 = 51,
    VK_FORMAT_A8B8G8R8_SNORM_PACK32 = 52,
    VK_FORMAT_A8B8G8R8_USCALED_PACK32 = 53,
    VK_FORMAT_A8B8G8R8_SSCALED_PACK32 = 54,
    VK_FORMAT_A8B8G8R8_UINT_PACK32 = 55,
    VK_FORMAT_A8B8G8R8_SINT_PACK32 = 56,
    VK_FORMAT_A8B8G8R8_SRGB_PACK32 = 57,
    VK_FORMAT_A2R10G10B10_UNORM_PACK32 = 58,
    VK_FORMAT_A2R10G10B10_SNORM_PACK32 = 59,
    VK_FORMAT_A2R10G10B10_USCALED_PACK32 = 60,
    VK_FORMAT_A2R10G10B10_SSCALED_PACK32 = 61,
    VK_FORMAT_A2R10G10B10_UINT_PACK32 = 62,
    VK_FORMAT_A2R10G10B10_SINT_PACK32 = 63,
    VK_FORMAT_A2B10G10R10_UNORM_PACK32 = 64,
    VK_FORMAT_A2B10G10R10_SNORM_PACK32 = 65,
    VK_FORMAT_A2B10G10R10_USCALED_PACK32 = 66,
    VK_FORMAT_A2B10G10R10_SSCALED_PACK32 = 67,
    VK_FORMAT_A2B10G10R10_UINT_PACK32 = 68,
    VK_FORMAT_A2B10G10R10_SINT_PACK32 = 69,
    VK_FORMAT_R16_UNORM = 70,
    VK_FORMAT_R16_SNORM = 71,
    VK_FORMAT_R16_USCALED = 72,
    VK_FORMAT_R16_SSCALED = 73,
    VK_FORMAT_R16_UINT = 74,
    VK_FORMAT_R16_SINT = 75,
    VK_FORMAT_R16_SFLOAT = 76,
    VK_FORMAT_R16G16_UNORM = 77,
    VK_FORMAT_R16G16_SNORM = 78,
    VK_FORMAT_R16G16_USCALED = 79,
    VK_FORMAT_R16G16_SSCALED = 80,
    VK_FORMAT_R16G16_UINT = 81,
    VK_FORMAT_R16G16_SINT = 82,
```

```
    VK_FORMAT_R16G16_SFLOAT = 83,
    VK_FORMAT_R16G16B16_UNORM = 84,
    VK_FORMAT_R16G16B16_SNORM = 85,
    VK_FORMAT_R16G16B16_USCALED = 86,
    VK_FORMAT_R16G16B16_SSCALED = 87,
    VK_FORMAT_R16G16B16_UINT = 88,
    VK_FORMAT_R16G16B16_SINT = 89,
    VK_FORMAT_R16G16B16_SFLOAT = 90,
    VK_FORMAT_R16G16B16A16_UNORM = 91,
    VK_FORMAT_R16G16B16A16_SNORM = 92,
    VK_FORMAT_R16G16B16A16_USCALED = 93,
    VK_FORMAT_R16G16B16A16_SSCALED = 94,
    VK_FORMAT_R16G16B16A16_UINT = 95,
    VK_FORMAT_R16G16B16A16_SINT = 96,
    VK_FORMAT_R16G16B16A16_SFLOAT = 97,
    VK_FORMAT_R32_UINT = 98,
    VK_FORMAT_R32_SINT = 99,
    VK_FORMAT_R32_SFLOAT = 100,
    VK_FORMAT_R32G32_UINT = 101,
    VK_FORMAT_R32G32_SINT = 102,
    VK_FORMAT_R32G32_SFLOAT = 103,
    VK_FORMAT_R32G32B32_UINT = 104,
    VK_FORMAT_R32G32B32_SINT = 105,
    VK_FORMAT_R32G32B32_SFLOAT = 106,
    VK_FORMAT_R32G32B32A32_UINT = 107,
    VK_FORMAT_R32G32B32A32_SINT = 108,
    VK_FORMAT_R32G32B32A32_SFLOAT = 109,
    VK_FORMAT_R64_UINT = 110,
    VK_FORMAT_R64_SINT = 111,
    VK_FORMAT_R64_SFLOAT = 112,
    VK_FORMAT_R64G64_UINT = 113,
    VK_FORMAT_R64G64_SINT = 114,
    VK_FORMAT_R64G64_SFLOAT = 115,
    VK_FORMAT_R64G64B64_UINT = 116,
    VK_FORMAT_R64G64B64_SINT = 117,
    VK_FORMAT_R64G64B64_SFLOAT = 118,
    VK_FORMAT_R64G64B64A64_UINT = 119,
    VK_FORMAT_R64G64B64A64_SINT = 120,
    VK_FORMAT_R64G64B64A64_SFLOAT = 121,
    VK_FORMAT_B10G11R11_UFLOAT_PACK32 = 122,
    VK_FORMAT_E5B9G9R9_UFLOAT_PACK32 = 123,
    VK_FORMAT_D16_UNORM = 124,
    VK_FORMAT_X8_D24_UNORM_PACK32 = 125,
    VK_FORMAT_D32_SFLOAT = 126,
    VK_FORMAT_S8_UINT = 127,
    VK_FORMAT_D16_UNORM_S8_UINT = 128,
    VK_FORMAT_D24_UNORM_S8_UINT = 129,
    VK_FORMAT_D32_SFLOAT_S8_UINT = 130,
    VK_FORMAT_BC1_RGB_UNORM_BLOCK = 131,
    VK_FORMAT_BC1_RGB_SRGB_BLOCK = 132,
    VK_FORMAT_BC1_RGBA_UNORM_BLOCK = 133,
```

```
    VK_FORMAT_BC1_RGBA_SRGB_BLOCK = 134,
    VK_FORMAT_BC2_UNORM_BLOCK = 135,
    VK_FORMAT_BC2_SRGB_BLOCK = 136,
    VK_FORMAT_BC3_UNORM_BLOCK = 137,
    VK_FORMAT_BC3_SRGB_BLOCK = 138,
    VK_FORMAT_BC4_UNORM_BLOCK = 139,
    VK_FORMAT_BC4_SNORM_BLOCK = 140,
    VK_FORMAT_BC5_UNORM_BLOCK = 141,
    VK_FORMAT_BC5_SNORM_BLOCK = 142,
    VK_FORMAT_BC6H_UFLOAT_BLOCK = 143,
    VK_FORMAT_BC6H_SFLOAT_BLOCK = 144,
    VK_FORMAT_BC7_UNORM_BLOCK = 145,
    VK_FORMAT_BC7_SRGB_BLOCK = 146,
    VK_FORMAT_ETC2_R8G8B8_UNORM_BLOCK = 147,
    VK_FORMAT_ETC2_R8G8B8_SRGB_BLOCK = 148,
    VK_FORMAT_ETC2_R8G8B8A1_UNORM_BLOCK = 149,
    VK_FORMAT_ETC2_R8G8B8A1_SRGB_BLOCK = 150,
    VK_FORMAT_ETC2_R8G8B8A8_UNORM_BLOCK = 151,
    VK_FORMAT_ETC2_R8G8B8A8_SRGB_BLOCK = 152,
    VK_FORMAT_EAC_R11_UNORM_BLOCK = 153,
    VK_FORMAT_EAC_R11_SNORM_BLOCK = 154,
    VK_FORMAT_EAC_R11G11_UNORM_BLOCK = 155,
    VK_FORMAT_EAC_R11G11_SNORM_BLOCK = 156,
    VK_FORMAT_ASTC_4x4_UNORM_BLOCK = 157,
    VK_FORMAT_ASTC_4x4_SRGB_BLOCK = 158,
    VK_FORMAT_ASTC_5x4_UNORM_BLOCK = 159,
    VK_FORMAT_ASTC_5x4_SRGB_BLOCK = 160,
    VK_FORMAT_ASTC_5x5_UNORM_BLOCK = 161,
    VK_FORMAT_ASTC_5x5_SRGB_BLOCK = 162,
    VK_FORMAT_ASTC_6x5_UNORM_BLOCK = 163,
    VK_FORMAT_ASTC_6x5_SRGB_BLOCK = 164,
    VK_FORMAT_ASTC_6x6_UNORM_BLOCK = 165,
    VK_FORMAT_ASTC_6x6_SRGB_BLOCK = 166,
    VK_FORMAT_ASTC_8x5_UNORM_BLOCK = 167,
    VK_FORMAT_ASTC_8x5_SRGB_BLOCK = 168,
    VK_FORMAT_ASTC_8x6_UNORM_BLOCK = 169,
    VK_FORMAT_ASTC_8x6_SRGB_BLOCK = 170,
    VK_FORMAT_ASTC_8x8_UNORM_BLOCK = 171,
    VK_FORMAT_ASTC_8x8_SRGB_BLOCK = 172,
    VK_FORMAT_ASTC_10x5_UNORM_BLOCK = 173,
    VK_FORMAT_ASTC_10x5_SRGB_BLOCK = 174,
    VK_FORMAT_ASTC_10x6_UNORM_BLOCK = 175,
    VK_FORMAT_ASTC_10x6_SRGB_BLOCK = 176,
    VK_FORMAT_ASTC_10x8_UNORM_BLOCK = 177,
    VK_FORMAT_ASTC_10x8_SRGB_BLOCK = 178,
    VK_FORMAT_ASTC_10x10_UNORM_BLOCK = 179,
    VK_FORMAT_ASTC_10x10_SRGB_BLOCK = 180,
    VK_FORMAT_ASTC_12x10_UNORM_BLOCK = 181,
    VK_FORMAT_ASTC_12x10_SRGB_BLOCK = 182,
    VK_FORMAT_ASTC_12x12_UNORM_BLOCK = 183,
    VK_FORMAT_ASTC_12x12_SRGB_BLOCK = 184,
```

```
// Provided by VK_VERSION_1_1
  VK_FORMAT_G8B8G8R8_422_UNORM = 1000156000,
// Provided by VK_VERSION_1_1
  VK_FORMAT_B8G8R8G8_422_UNORM = 1000156001,
// Provided by VK_VERSION_1_1
  VK_FORMAT_G8_B8_R8_3PLANE_420_UNORM = 1000156002,
// Provided by VK_VERSION_1_1
  VK_FORMAT_G8_B8R8_2PLANE_420_UNORM = 1000156003,
// Provided by VK_VERSION_1_1
  VK_FORMAT_G8_B8_R8_3PLANE_422_UNORM = 1000156004,
// Provided by VK_VERSION_1_1
  VK_FORMAT_G8_B8R8_2PLANE_422_UNORM = 1000156005,
// Provided by VK_VERSION_1_1
  VK_FORMAT_G8_B8_R8_3PLANE_444_UNORM = 1000156006,
// Provided by VK_VERSION_1_1
  VK_FORMAT_R10X6_UNORM_PACK16 = 1000156007,
// Provided by VK_VERSION_1_1
  VK_FORMAT_R10X6G10X6_UNORM_2PACK16 = 1000156008,
// Provided by VK_VERSION_1_1
  VK_FORMAT_R10X6G10X6B10X6A10X6_UNORM_4PACK16 = 1000156009,
// Provided by VK_VERSION_1_1
  VK_FORMAT_G10X6B10X6G10X6R10X6_422_UNORM_4PACK16 = 1000156010,
// Provided by VK_VERSION_1_1
  VK_FORMAT_B10X6G10X6R10X6G10X6_422_UNORM_4PACK16 = 1000156011,
// Provided by VK_VERSION_1_1
  VK_FORMAT_G10X6_B10X6_R10X6_3PLANE_420_UNORM_3PACK16 = 1000156012,
// Provided by VK_VERSION_1_1
  VK_FORMAT_G10X6_B10X6R10X6_2PLANE_420_UNORM_3PACK16 = 1000156013,
// Provided by VK_VERSION_1_1
  VK_FORMAT_G10X6_B10X6_R10X6_3PLANE_422_UNORM_3PACK16 = 1000156014,
// Provided by VK_VERSION_1_1
  VK_FORMAT_G10X6_B10X6R10X6_2PLANE_422_UNORM_3PACK16 = 1000156015,
// Provided by VK_VERSION_1_1
  VK_FORMAT_G10X6_B10X6_R10X6_3PLANE_444_UNORM_3PACK16 = 1000156016,
// Provided by VK_VERSION_1_1
  VK_FORMAT_R12X4_UNORM_PACK16 = 1000156017,
// Provided by VK_VERSION_1_1
  VK_FORMAT_R12X4G12X4_UNORM_2PACK16 = 1000156018,
// Provided by VK_VERSION_1_1
  VK_FORMAT_R12X4G12X4B12X4A12X4_UNORM_4PACK16 = 1000156019,
// Provided by VK_VERSION_1_1
  VK_FORMAT_G12X4B12X4G12X4R12X4_422_UNORM_4PACK16 = 1000156020,
// Provided by VK_VERSION_1_1
  VK_FORMAT_B12X4G12X4R12X4G12X4_422_UNORM_4PACK16 = 1000156021,
// Provided by VK_VERSION_1_1
  VK_FORMAT_G12X4_B12X4_R12X4_3PLANE_420_UNORM_3PACK16 = 1000156022,
// Provided by VK_VERSION_1_1
  VK_FORMAT_G12X4_B12X4R12X4_2PLANE_420_UNORM_3PACK16 = 1000156023,
// Provided by VK_VERSION_1_1
  VK_FORMAT_G12X4_B12X4_R12X4_3PLANE_422_UNORM_3PACK16 = 1000156024,
// Provided by VK_VERSION_1_1
```

```
    VK_FORMAT_G12X4_B12X4R12X4_2PLANE_422_UNORM_3PACK16 = 1000156025,
  // Provided by VK_VERSION_1_1
    VK_FORMAT_G12X4_B12X4_R12X4_3PLANE_444_UNORM_3PACK16 = 1000156026,
  // Provided by VK_VERSION_1_1
    VK_FORMAT_G16B16G16R16_422_UNORM = 1000156027,
  // Provided by VK_VERSION_1_1
    VK_FORMAT_B16G16R16G16_422_UNORM = 1000156028,
  // Provided by VK_VERSION_1_1
    VK_FORMAT_G16_B16_R16_3PLANE_420_UNORM = 1000156029,
  // Provided by VK_VERSION_1_1
    VK_FORMAT_G16_B16R16_2PLANE_420_UNORM = 1000156030,
  // Provided by VK_VERSION_1_1
    VK_FORMAT_G16_B16_R16_3PLANE_422_UNORM = 1000156031,
  // Provided by VK_VERSION_1_1
    VK_FORMAT_G16_B16R16_2PLANE_422_UNORM = 1000156032,
  // Provided by VK_VERSION_1_1
    VK_FORMAT_G16_B16_R16_3PLANE_444_UNORM = 1000156033,
    VK_FORMAT_G8_B8R8_2PLANE_444_UNORM = 1000330000,
    VK_FORMAT_G10X6_B10X6R10X6_2PLANE_444_UNORM_3PACK16 = 1000330001,
    VK_FORMAT_G12X4_B12X4R12X4_2PLANE_444_UNORM_3PACK16 = 1000330002,
    VK_FORMAT_G16_B16R16_2PLANE_444_UNORM = 1000330003,
    VK_FORMAT_A4R4G4B4_UNORM_PACK16 = 1000340000,
    VK_FORMAT_A4B4G4R4_UNORM_PACK16 = 1000340001,
    VK_FORMAT_ASTC_4x4_SFLOAT_BLOCK = 1000066000,
    VK_FORMAT_ASTC_5x4_SFLOAT_BLOCK = 1000066001,
    VK_FORMAT_ASTC_5x5_SFLOAT_BLOCK = 1000066002,
    VK_FORMAT_ASTC_6x5_SFLOAT_BLOCK = 1000066003,
    VK_FORMAT_ASTC_6x6_SFLOAT_BLOCK = 1000066004,
    VK_FORMAT_ASTC_8x5_SFLOAT_BLOCK = 1000066005,
    VK_FORMAT_ASTC_8x6_SFLOAT_BLOCK = 1000066006,
    VK_FORMAT_ASTC_8x8_SFLOAT_BLOCK = 1000066007,
    VK_FORMAT_ASTC_10x5_SFLOAT_BLOCK = 1000066008,
    VK_FORMAT_ASTC_10x6_SFLOAT_BLOCK = 1000066009,
    VK_FORMAT_ASTC_10x8_SFLOAT_BLOCK = 1000066010,
    VK_FORMAT_ASTC_10x10_SFLOAT_BLOCK = 1000066011,
    VK_FORMAT_ASTC_12x10_SFLOAT_BLOCK = 1000066012,
    VK_FORMAT_ASTC_12x12_SFLOAT_BLOCK = 1000066013,
  // Provided by VK_IMG_format_pvrtc
    VK_FORMAT_PVRTC1_2BPP_UNORM_BLOCK_IMG = 1000054000,
  // Provided by VK_IMG_format_pvrtc
    VK_FORMAT_PVRTC1_4BPP_UNORM_BLOCK_IMG = 1000054001,
  // Provided by VK_IMG_format_pvrtc
    VK_FORMAT_PVRTC2_2BPP_UNORM_BLOCK_IMG = 1000054002,
  // Provided by VK_IMG_format_pvrtc
    VK_FORMAT_PVRTC2_4BPP_UNORM_BLOCK_IMG = 1000054003,
  // Provided by VK_IMG_format_pvrtc
    VK_FORMAT_PVRTC1_2BPP_SRGB_BLOCK_IMG = 1000054004,
  // Provided by VK_IMG_format_pvrtc
    VK_FORMAT_PVRTC1_4BPP_SRGB_BLOCK_IMG = 1000054005,
  // Provided by VK_IMG_format_pvrtc
    VK_FORMAT_PVRTC2_2BPP_SRGB_BLOCK_IMG = 1000054006,
```

```
// Provided by VK_IMG_format_pvrtc
  VK_FORMAT_PVRTC2_4BPP_SRGB_BLOCK_IMG = 1000054007,
// Provided by VK_NV_optical_flow
  VK_FORMAT_R16G16_SFIXED5_NV = 1000464000,
// Provided by VK_KHR_maintenance5
  VK_FORMAT_A1B5G5R5_UNORM_PACK16_KHR = 1000470000,
// Provided by VK_KHR_maintenance5
  VK_FORMAT_A8_UNORM_KHR = 1000470001,
// Provided by VK_EXT_texture_compression_astc_hdr
  VK_FORMAT_ASTC_4x4_SFLOAT_BLOCK_EXT = VK_FORMAT_ASTC_4x4_SFLOAT_BLOCK,
// Provided by VK_EXT_texture_compression_astc_hdr
  VK_FORMAT_ASTC_5x4_SFLOAT_BLOCK_EXT = VK_FORMAT_ASTC_5x4_SFLOAT_BLOCK,
// Provided by VK_EXT_texture_compression_astc_hdr
  VK_FORMAT_ASTC_5x5_SFLOAT_BLOCK_EXT = VK_FORMAT_ASTC_5x5_SFLOAT_BLOCK,
// Provided by VK_EXT_texture_compression_astc_hdr
  VK_FORMAT_ASTC_6x5_SFLOAT_BLOCK_EXT = VK_FORMAT_ASTC_6x5_SFLOAT_BLOCK,
// Provided by VK_EXT_texture_compression_astc_hdr
  VK_FORMAT_ASTC_6x6_SFLOAT_BLOCK_EXT = VK_FORMAT_ASTC_6x6_SFLOAT_BLOCK,
// Provided by VK_EXT_texture_compression_astc_hdr
  VK_FORMAT_ASTC_8x5_SFLOAT_BLOCK_EXT = VK_FORMAT_ASTC_8x5_SFLOAT_BLOCK,
// Provided by VK_EXT_texture_compression_astc_hdr
  VK_FORMAT_ASTC_8x6_SFLOAT_BLOCK_EXT = VK_FORMAT_ASTC_8x6_SFLOAT_BLOCK,
// Provided by VK_EXT_texture_compression_astc_hdr
  VK_FORMAT_ASTC_8x8_SFLOAT_BLOCK_EXT = VK_FORMAT_ASTC_8x8_SFLOAT_BLOCK,
// Provided by VK_EXT_texture_compression_astc_hdr
  VK_FORMAT_ASTC_10x5_SFLOAT_BLOCK_EXT = VK_FORMAT_ASTC_10x5_SFLOAT_BLOCK,
// Provided by VK_EXT_texture_compression_astc_hdr
  VK_FORMAT_ASTC_10x6_SFLOAT_BLOCK_EXT = VK_FORMAT_ASTC_10x6_SFLOAT_BLOCK,
// Provided by VK_EXT_texture_compression_astc_hdr
  VK_FORMAT_ASTC_10x8_SFLOAT_BLOCK_EXT = VK_FORMAT_ASTC_10x8_SFLOAT_BLOCK,
// Provided by VK_EXT_texture_compression_astc_hdr
  VK_FORMAT_ASTC_10x10_SFLOAT_BLOCK_EXT = VK_FORMAT_ASTC_10x10_SFLOAT_BLOCK,
// Provided by VK_EXT_texture_compression_astc_hdr
  VK_FORMAT_ASTC_12x10_SFLOAT_BLOCK_EXT = VK_FORMAT_ASTC_12x10_SFLOAT_BLOCK,
// Provided by VK_EXT_texture_compression_astc_hdr
  VK_FORMAT_ASTC_12x12_SFLOAT_BLOCK_EXT = VK_FORMAT_ASTC_12x12_SFLOAT_BLOCK,
// Provided by VK_KHR_sampler_ycbcr_conversion
  VK_FORMAT_G8B8G8R8_422_UNORM_KHR = VK_FORMAT_G8B8G8R8_422_UNORM,
// Provided by VK_KHR_sampler_ycbcr_conversion
  VK_FORMAT_B8G8R8G8_422_UNORM_KHR = VK_FORMAT_B8G8R8G8_422_UNORM,
// Provided by VK_KHR_sampler_ycbcr_conversion
  VK_FORMAT_G8_B8_R8_3PLANE_420_UNORM_KHR = VK_FORMAT_G8_B8_R8_3PLANE_420_UNORM,
// Provided by VK_KHR_sampler_ycbcr_conversion
  VK_FORMAT_G8_B8R8_2PLANE_420_UNORM_KHR = VK_FORMAT_G8_B8R8_2PLANE_420_UNORM,
// Provided by VK_KHR_sampler_ycbcr_conversion
  VK_FORMAT_G8_B8_R8_3PLANE_422_UNORM_KHR = VK_FORMAT_G8_B8_R8_3PLANE_422_UNORM,
// Provided by VK_KHR_sampler_ycbcr_conversion
  VK_FORMAT_G8_B8R8_2PLANE_422_UNORM_KHR = VK_FORMAT_G8_B8R8_2PLANE_422_UNORM,
// Provided by VK_KHR_sampler_ycbcr_conversion
  VK_FORMAT_G8_B8_R8_3PLANE_444_UNORM_KHR = VK_FORMAT_G8_B8_R8_3PLANE_444_UNORM,
// Provided by VK_KHR_sampler_ycbcr_conversion
```

```
    VK_FORMAT_R10X6_UNORM_PACK16_KHR = VK_FORMAT_R10X6_UNORM_PACK16,
  // Provided by VK_KHR_sampler_ycbcr_conversion
    VK_FORMAT_R10X6G10X6_UNORM_2PACK16_KHR = VK_FORMAT_R10X6G10X6_UNORM_2PACK16,
  // Provided by VK_KHR_sampler_ycbcr_conversion
    VK_FORMAT_R10X6G10X6B10X6A10X6_UNORM_4PACK16_KHR =
VK_FORMAT_R10X6G10X6B10X6A10X6_UNORM_4PACK16,
  // Provided by VK_KHR_sampler_ycbcr_conversion
    VK_FORMAT_G10X6B10X6G10X6R10X6_422_UNORM_4PACK16_KHR =
VK_FORMAT_G10X6B10X6G10X6R10X6_422_UNORM_4PACK16,
  // Provided by VK_KHR_sampler_ycbcr_conversion
    VK_FORMAT_B10X6G10X6R10X6G10X6_422_UNORM_4PACK16_KHR =
VK_FORMAT_B10X6G10X6R10X6G10X6_422_UNORM_4PACK16,
  // Provided by VK_KHR_sampler_ycbcr_conversion
    VK_FORMAT_G10X6_B10X6_R10X6_3PLANE_420_UNORM_3PACK16_KHR =
VK_FORMAT_G10X6_B10X6_R10X6_3PLANE_420_UNORM_3PACK16,
  // Provided by VK_KHR_sampler_ycbcr_conversion
    VK_FORMAT_G10X6_B10X6R10X6_2PLANE_420_UNORM_3PACK16_KHR =
VK_FORMAT_G10X6_B10X6R10X6_2PLANE_420_UNORM_3PACK16,
  // Provided by VK_KHR_sampler_ycbcr_conversion
    VK_FORMAT_G10X6_B10X6_R10X6_3PLANE_422_UNORM_3PACK16_KHR =
VK_FORMAT_G10X6_B10X6_R10X6_3PLANE_422_UNORM_3PACK16,
  // Provided by VK_KHR_sampler_ycbcr_conversion
    VK_FORMAT_G10X6_B10X6R10X6_2PLANE_422_UNORM_3PACK16_KHR =
VK_FORMAT_G10X6_B10X6R10X6_2PLANE_422_UNORM_3PACK16,
  // Provided by VK_KHR_sampler_ycbcr_conversion
    VK_FORMAT_G10X6_B10X6_R10X6_3PLANE_444_UNORM_3PACK16_KHR =
VK_FORMAT_G10X6_B10X6_R10X6_3PLANE_444_UNORM_3PACK16,
  // Provided by VK_KHR_sampler_ycbcr_conversion
    VK_FORMAT_R12X4_UNORM_PACK16_KHR = VK_FORMAT_R12X4_UNORM_PACK16,
  // Provided by VK_KHR_sampler_ycbcr_conversion
    VK_FORMAT_R12X4G12X4_UNORM_2PACK16_KHR = VK_FORMAT_R12X4G12X4_UNORM_2PACK16,
  // Provided by VK_KHR_sampler_ycbcr_conversion
    VK_FORMAT_R12X4G12X4B12X4A12X4_UNORM_4PACK16_KHR =
VK_FORMAT_R12X4G12X4B12X4A12X4_UNORM_4PACK16,
  // Provided by VK_KHR_sampler_ycbcr_conversion
    VK_FORMAT_G12X4B12X4G12X4R12X4_422_UNORM_4PACK16_KHR =
VK_FORMAT_G12X4B12X4G12X4R12X4_422_UNORM_4PACK16,
  // Provided by VK_KHR_sampler_ycbcr_conversion
    VK_FORMAT_B12X4G12X4R12X4G12X4_422_UNORM_4PACK16_KHR =
VK_FORMAT_B12X4G12X4R12X4G12X4_422_UNORM_4PACK16,
  // Provided by VK_KHR_sampler_ycbcr_conversion
    VK_FORMAT_G12X4_B12X4_R12X4_3PLANE_420_UNORM_3PACK16_KHR =
VK_FORMAT_G12X4_B12X4_R12X4_3PLANE_420_UNORM_3PACK16,
  // Provided by VK_KHR_sampler_ycbcr_conversion
    VK_FORMAT_G12X4_B12X4R12X4_2PLANE_420_UNORM_3PACK16_KHR =
VK_FORMAT_G12X4_B12X4R12X4_2PLANE_420_UNORM_3PACK16,
  // Provided by VK_KHR_sampler_ycbcr_conversion
    VK_FORMAT_G12X4_B12X4_R12X4_3PLANE_422_UNORM_3PACK16_KHR =
VK_FORMAT_G12X4_B12X4_R12X4_3PLANE_422_UNORM_3PACK16,
  // Provided by VK_KHR_sampler_ycbcr_conversion
    VK_FORMAT_G12X4_B12X4R12X4_2PLANE_422_UNORM_3PACK16_KHR =
```

```
VK_FORMAT_G12X4_B12X4R12X4_2PLANE_422_UNORM_3PACK16,
  // Provided by VK_KHR_sampler_ycbcr_conversion
    VK_FORMAT_G12X4_B12X4_R12X4_3PLANE_444_UNORM_3PACK16_KHR =
VK_FORMAT_G12X4_B12X4_R12X4_3PLANE_444_UNORM_3PACK16,
  // Provided by VK_KHR_sampler_ycbcr_conversion
    VK_FORMAT_G16B16G16R16_422_UNORM_KHR = VK_FORMAT_G16B16G16R16_422_UNORM,
  // Provided by VK_KHR_sampler_ycbcr_conversion
    VK_FORMAT_B16G16R16G16_422_UNORM_KHR = VK_FORMAT_B16G16R16G16_422_UNORM,
  // Provided by VK_KHR_sampler_ycbcr_conversion
    VK_FORMAT_G16_B16_R16_3PLANE_420_UNORM_KHR =
VK_FORMAT_G16_B16_R16_3PLANE_420_UNORM,
  // Provided by VK_KHR_sampler_ycbcr_conversion
    VK_FORMAT_G16_B16R16_2PLANE_420_UNORM_KHR = VK_FORMAT_G16_B16R16_2PLANE_420_UNORM,
  // Provided by VK_KHR_sampler_ycbcr_conversion
    VK_FORMAT_G16_B16_R16_3PLANE_422_UNORM_KHR =
VK_FORMAT_G16_B16_R16_3PLANE_422_UNORM,
  // Provided by VK_KHR_sampler_ycbcr_conversion
    VK_FORMAT_G16_B16R16_2PLANE_422_UNORM_KHR = VK_FORMAT_G16_B16R16_2PLANE_422_UNORM,
  // Provided by VK_KHR_sampler_ycbcr_conversion
    VK_FORMAT_G16_B16_R16_3PLANE_444_UNORM_KHR =
VK_FORMAT_G16_B16_R16_3PLANE_444_UNORM,
  // Provided by VK_EXT_ycbcr_2plane_444_formats
    VK_FORMAT_G8_B8R8_2PLANE_444_UNORM_EXT = VK_FORMAT_G8_B8R8_2PLANE_444_UNORM,
  // Provided by VK_EXT_ycbcr_2plane_444_formats
    VK_FORMAT_G10X6_B10X6R10X6_2PLANE_444_UNORM_3PACK16_EXT =
VK_FORMAT_G10X6_B10X6R10X6_2PLANE_444_UNORM_3PACK16,
  // Provided by VK_EXT_ycbcr_2plane_444_formats
    VK_FORMAT_G12X4_B12X4R12X4_2PLANE_444_UNORM_3PACK16_EXT =
VK_FORMAT_G12X4_B12X4R12X4_2PLANE_444_UNORM_3PACK16,
  // Provided by VK_EXT_ycbcr_2plane_444_formats
    VK_FORMAT_G16_B16R16_2PLANE_444_UNORM_EXT = VK_FORMAT_G16_B16R16_2PLANE_444_UNORM,
  // Provided by VK_EXT_4444_formats
    VK_FORMAT_A4R4G4B4_UNORM_PACK16_EXT = VK_FORMAT_A4R4G4B4_UNORM_PACK16,
  // Provided by VK_EXT_4444_formats
    VK_FORMAT_A4B4G4R4_UNORM_PACK16_EXT = VK_FORMAT_A4B4G4R4_UNORM_PACK16,
  // Provided by VK_NV_optical_flow
  // VK_FORMAT_R16G16_S10_5_NV is a deprecated alias
    VK_FORMAT_R16G16_S10_5_NV = VK_FORMAT_R16G16_SFIXED5_NV,
} VkFormat;
```

- `VK_FORMAT_UNDEFINED` specifies that the format is not specified.
- `VK_FORMAT_R4G4_UNORM_PACK8` specifies a two-component, 8-bit packed unsigned normalized format that has a 4-bit R component in bits 4..7, and a 4-bit G component in bits 0..3.
- `VK_FORMAT_R4G4B4A4_UNORM_PACK16` specifies a four-component, 16-bit packed unsigned normalized format that has a 4-bit R component in bits 12..15, a 4-bit G component in bits 8..11, a 4-bit B component in bits 4..7, and a 4-bit A component in bits 0..3.
- `VK_FORMAT_B4G4R4A4_UNORM_PACK16` specifies a four-component, 16-bit packed unsigned normalized format that has a 4-bit B component in bits 12..15, a 4-bit G component in bits 8..11, a 4-bit R component in bits 4..7, and a 4-bit A component in bits 0..3.

- VK_FORMAT_A4R4G4B4_UNORM_PACK16 specifies a four-component, 16-bit packed unsigned normalized format that has a 4-bit A component in bits 12..15, a 4-bit R component in bits 8..11, a 4-bit G component in bits 4..7, and a 4-bit B component in bits 0..3.
- VK_FORMAT_A4B4G4R4_UNORM_PACK16 specifies a four-component, 16-bit packed unsigned normalized format that has a 4-bit A component in bits 12..15, a 4-bit B component in bits 8..11, a 4-bit G component in bits 4..7, and a 4-bit R component in bits 0..3.
- VK_FORMAT_R5G6B5_UNORM_PACK16 specifies a three-component, 16-bit packed unsigned normalized format that has a 5-bit R component in bits 11..15, a 6-bit G component in bits 5..10, and a 5-bit B component in bits 0..4.
- VK_FORMAT_B5G6R5_UNORM_PACK16 specifies a three-component, 16-bit packed unsigned normalized format that has a 5-bit B component in bits 11..15, a 6-bit G component in bits 5..10, and a 5-bit R component in bits 0..4.
- VK_FORMAT_R5G5B5A1_UNORM_PACK16 specifies a four-component, 16-bit packed unsigned normalized format that has a 5-bit R component in bits 11..15, a 5-bit G component in bits 6..10, a 5-bit B component in bits 1..5, and a 1-bit A component in bit 0.
- VK_FORMAT_B5G5R5A1_UNORM_PACK16 specifies a four-component, 16-bit packed unsigned normalized format that has a 5-bit B component in bits 11..15, a 5-bit G component in bits 6..10, a 5-bit R component in bits 1..5, and a 1-bit A component in bit 0.
- VK_FORMAT_A1R5G5B5_UNORM_PACK16 specifies a four-component, 16-bit packed unsigned normalized format that has a 1-bit A component in bit 15, a 5-bit R component in bits 10..14, a 5-bit G component in bits 5..9, and a 5-bit B component in bits 0..4.
- VK_FORMAT_A1B5G5R5_UNORM_PACK16_KHR specifies a four-component, 16-bit packed unsigned normalized format that has a 1-bit A component in bit 15, a 5-bit B component in bits 10..14, a 5-bit G component in bits 5..9, and a 5-bit R component in bits 0..4.
- VK_FORMAT_A8_UNORM_KHR specifies a one-component, 8-bit unsigned normalized format that has a single 8-bit A component.
- VK_FORMAT_R8_UNORM specifies a one-component, 8-bit unsigned normalized format that has a single 8-bit R component.
- VK_FORMAT_R8_SNORM specifies a one-component, 8-bit signed normalized format that has a single 8-bit R component.
- VK_FORMAT_R8_USCALED specifies a one-component, 8-bit unsigned scaled integer format that has a single 8-bit R component.
- VK_FORMAT_R8_SSCALED specifies a one-component, 8-bit signed scaled integer format that has a single 8-bit R component.
- VK_FORMAT_R8_UINT specifies a one-component, 8-bit unsigned integer format that has a single 8-bit R component.
- VK_FORMAT_R8_SINT specifies a one-component, 8-bit signed integer format that has a single 8-bit R component.
- VK_FORMAT_R8_SRGB specifies a one-component, 8-bit unsigned normalized format that has a single 8-bit R component stored with sRGB nonlinear encoding.
- VK_FORMAT_R8G8_UNORM specifies a two-component, 16-bit unsigned normalized format that has an

8-bit R component in byte 0, and an 8-bit G component in byte 1.

- VK_FORMAT_R8G8_SNORM specifies a two-component, 16-bit signed normalized format that has an 8-bit R component in byte 0, and an 8-bit G component in byte 1.
- VK_FORMAT_R8G8_USCALED specifies a two-component, 16-bit unsigned scaled integer format that has an 8-bit R component in byte 0, and an 8-bit G component in byte 1.
- VK_FORMAT_R8G8_SSCALED specifies a two-component, 16-bit signed scaled integer format that has an 8-bit R component in byte 0, and an 8-bit G component in byte 1.
- VK_FORMAT_R8G8_UINT specifies a two-component, 16-bit unsigned integer format that has an 8-bit R component in byte 0, and an 8-bit G component in byte 1.
- VK_FORMAT_R8G8_SINT specifies a two-component, 16-bit signed integer format that has an 8-bit R component in byte 0, and an 8-bit G component in byte 1.
- VK_FORMAT_R8G8_SRGB specifies a two-component, 16-bit unsigned normalized format that has an 8-bit R component stored with sRGB nonlinear encoding in byte 0, and an 8-bit G component stored with sRGB nonlinear encoding in byte 1.
- VK_FORMAT_R8G8B8_UNORM specifies a three-component, 24-bit unsigned normalized format that has an 8-bit R component in byte 0, an 8-bit G component in byte 1, and an 8-bit B component in byte 2.
- VK_FORMAT_R8G8B8_SNORM specifies a three-component, 24-bit signed normalized format that has an 8-bit R component in byte 0, an 8-bit G component in byte 1, and an 8-bit B component in byte 2.
- VK_FORMAT_R8G8B8_USCALED specifies a three-component, 24-bit unsigned scaled format that has an 8-bit R component in byte 0, an 8-bit G component in byte 1, and an 8-bit B component in byte 2.
- VK_FORMAT_R8G8B8_SSCALED specifies a three-component, 24-bit signed scaled format that has an 8-bit R component in byte 0, an 8-bit G component in byte 1, and an 8-bit B component in byte 2.
- VK_FORMAT_R8G8B8_UINT specifies a three-component, 24-bit unsigned integer format that has an 8-bit R component in byte 0, an 8-bit G component in byte 1, and an 8-bit B component in byte 2.
- VK_FORMAT_R8G8B8_SINT specifies a three-component, 24-bit signed integer format that has an 8-bit R component in byte 0, an 8-bit G component in byte 1, and an 8-bit B component in byte 2.
- VK_FORMAT_R8G8B8_SRGB specifies a three-component, 24-bit unsigned normalized format that has an 8-bit R component stored with sRGB nonlinear encoding in byte 0, an 8-bit G component stored with sRGB nonlinear encoding in byte 1, and an 8-bit B component stored with sRGB nonlinear encoding in byte 2.
- VK_FORMAT_B8G8R8_UNORM specifies a three-component, 24-bit unsigned normalized format that has an 8-bit B component in byte 0, an 8-bit G component in byte 1, and an 8-bit R component in byte 2.
- VK_FORMAT_B8G8R8_SNORM specifies a three-component, 24-bit signed normalized format that has an 8-bit B component in byte 0, an 8-bit G component in byte 1, and an 8-bit R component in byte 2.
- VK_FORMAT_B8G8R8_USCALED specifies a three-component, 24-bit unsigned scaled format that has an 8-bit B component in byte 0, an 8-bit G component in byte 1, and an 8-bit R component in byte 2.
- VK_FORMAT_B8G8R8_SSCALED specifies a three-component, 24-bit signed scaled format that has an 8-

bit B component in byte 0, an 8-bit G component in byte 1, and an 8-bit R component in byte 2.

- VK_FORMAT_B8G8R8_UINT specifies a three-component, 24-bit unsigned integer format that has an 8-bit B component in byte 0, an 8-bit G component in byte 1, and an 8-bit R component in byte 2.
- VK_FORMAT_B8G8R8_SINT specifies a three-component, 24-bit signed integer format that has an 8-bit B component in byte 0, an 8-bit G component in byte 1, and an 8-bit R component in byte 2.
- VK_FORMAT_B8G8R8_SRGB specifies a three-component, 24-bit unsigned normalized format that has an 8-bit B component stored with sRGB nonlinear encoding in byte 0, an 8-bit G component stored with sRGB nonlinear encoding in byte 1, and an 8-bit R component stored with sRGB nonlinear encoding in byte 2.
- VK_FORMAT_R8G8B8A8_UNORM specifies a four-component, 32-bit unsigned normalized format that has an 8-bit R component in byte 0, an 8-bit G component in byte 1, an 8-bit B component in byte 2, and an 8-bit A component in byte 3.
- VK_FORMAT_R8G8B8A8_SNORM specifies a four-component, 32-bit signed normalized format that has an 8-bit R component in byte 0, an 8-bit G component in byte 1, an 8-bit B component in byte 2, and an 8-bit A component in byte 3.
- VK_FORMAT_R8G8B8A8_USCALED specifies a four-component, 32-bit unsigned scaled format that has an 8-bit R component in byte 0, an 8-bit G component in byte 1, an 8-bit B component in byte 2, and an 8-bit A component in byte 3.
- VK_FORMAT_R8G8B8A8_SSCALED specifies a four-component, 32-bit signed scaled format that has an 8-bit R component in byte 0, an 8-bit G component in byte 1, an 8-bit B component in byte 2, and an 8-bit A component in byte 3.
- VK_FORMAT_R8G8B8A8_UINT specifies a four-component, 32-bit unsigned integer format that has an 8-bit R component in byte 0, an 8-bit G component in byte 1, an 8-bit B component in byte 2, and an 8-bit A component in byte 3.
- VK_FORMAT_R8G8B8A8_SINT specifies a four-component, 32-bit signed integer format that has an 8-bit R component in byte 0, an 8-bit G component in byte 1, an 8-bit B component in byte 2, and an 8-bit A component in byte 3.
- VK_FORMAT_R8G8B8A8_SRGB specifies a four-component, 32-bit unsigned normalized format that has an 8-bit R component stored with sRGB nonlinear encoding in byte 0, an 8-bit G component stored with sRGB nonlinear encoding in byte 1, an 8-bit B component stored with sRGB nonlinear encoding in byte 2, and an 8-bit A component in byte 3.
- VK_FORMAT_B8G8R8A8_UNORM specifies a four-component, 32-bit unsigned normalized format that has an 8-bit B component in byte 0, an 8-bit G component in byte 1, an 8-bit R component in byte 2, and an 8-bit A component in byte 3.
- VK_FORMAT_B8G8R8A8_SNORM specifies a four-component, 32-bit signed normalized format that has an 8-bit B component in byte 0, an 8-bit G component in byte 1, an 8-bit R component in byte 2, and an 8-bit A component in byte 3.
- VK_FORMAT_B8G8R8A8_USCALED specifies a four-component, 32-bit unsigned scaled format that has an 8-bit B component in byte 0, an 8-bit G component in byte 1, an 8-bit R component in byte 2, and an 8-bit A component in byte 3.
- VK_FORMAT_B8G8R8A8_SSCALED specifies a four-component, 32-bit signed scaled format that has an 8-bit B component in byte 0, an 8-bit G component in byte 1, an 8-bit R component in byte 2, and

an 8-bit A component in byte 3.

- VK_FORMAT_B8G8R8A8_UINT specifies a four-component, 32-bit unsigned integer format that has an 8-bit B component in byte 0, an 8-bit G component in byte 1, an 8-bit R component in byte 2, and an 8-bit A component in byte 3.
- VK_FORMAT_B8G8R8A8_SINT specifies a four-component, 32-bit signed integer format that has an 8-bit B component in byte 0, an 8-bit G component in byte 1, an 8-bit R component in byte 2, and an 8-bit A component in byte 3.
- VK_FORMAT_B8G8R8A8_SRGB specifies a four-component, 32-bit unsigned normalized format that has an 8-bit B component stored with sRGB nonlinear encoding in byte 0, an 8-bit G component stored with sRGB nonlinear encoding in byte 1, an 8-bit R component stored with sRGB nonlinear encoding in byte 2, and an 8-bit A component in byte 3.
- VK_FORMAT_A8B8G8R8_UNORM_PACK32 specifies a four-component, 32-bit packed unsigned normalized format that has an 8-bit A component in bits 24..31, an 8-bit B component in bits 16..23, an 8-bit G component in bits 8..15, and an 8-bit R component in bits 0..7.
- VK_FORMAT_A8B8G8R8_SNORM_PACK32 specifies a four-component, 32-bit packed signed normalized format that has an 8-bit A component in bits 24..31, an 8-bit B component in bits 16..23, an 8-bit G component in bits 8..15, and an 8-bit R component in bits 0..7.
- VK_FORMAT_A8B8G8R8_USCALED_PACK32 specifies a four-component, 32-bit packed unsigned scaled integer format that has an 8-bit A component in bits 24..31, an 8-bit B component in bits 16..23, an 8-bit G component in bits 8..15, and an 8-bit R component in bits 0..7.
- VK_FORMAT_A8B8G8R8_SSCALED_PACK32 specifies a four-component, 32-bit packed signed scaled integer format that has an 8-bit A component in bits 24..31, an 8-bit B component in bits 16..23, an 8-bit G component in bits 8..15, and an 8-bit R component in bits 0..7.
- VK_FORMAT_A8B8G8R8_UINT_PACK32 specifies a four-component, 32-bit packed unsigned integer format that has an 8-bit A component in bits 24..31, an 8-bit B component in bits 16..23, an 8-bit G component in bits 8..15, and an 8-bit R component in bits 0..7.
- VK_FORMAT_A8B8G8R8_SINT_PACK32 specifies a four-component, 32-bit packed signed integer format that has an 8-bit A component in bits 24..31, an 8-bit B component in bits 16..23, an 8-bit G component in bits 8..15, and an 8-bit R component in bits 0..7.
- VK_FORMAT_A8B8G8R8_SRGB_PACK32 specifies a four-component, 32-bit packed unsigned normalized format that has an 8-bit A component in bits 24..31, an 8-bit B component stored with sRGB nonlinear encoding in bits 16..23, an 8-bit G component stored with sRGB nonlinear encoding in bits 8..15, and an 8-bit R component stored with sRGB nonlinear encoding in bits 0..7.
- VK_FORMAT_A2R10G10B10_UNORM_PACK32 specifies a four-component, 32-bit packed unsigned normalized format that has a 2-bit A component in bits 30..31, a 10-bit R component in bits 20..29, a 10-bit G component in bits 10..19, and a 10-bit B component in bits 0..9.
- VK_FORMAT_A2R10G10B10_SNORM_PACK32 specifies a four-component, 32-bit packed signed normalized format that has a 2-bit A component in bits 30..31, a 10-bit R component in bits 20..29, a 10-bit G component in bits 10..19, and a 10-bit B component in bits 0..9.
- VK_FORMAT_A2R10G10B10_USCALED_PACK32 specifies a four-component, 32-bit packed unsigned scaled integer format that has a 2-bit A component in bits 30..31, a 10-bit R component in bits 20..29, a 10-bit G component in bits 10..19, and a 10-bit B component in bits 0..9.

- VK_FORMAT_A2R10G10B10_SSCALED_PACK32 specifies a four-component, 32-bit packed signed scaled integer format that has a 2-bit A component in bits 30..31, a 10-bit R component in bits 20..29, a 10-bit G component in bits 10..19, and a 10-bit B component in bits 0..9.
- VK_FORMAT_A2R10G10B10_UINT_PACK32 specifies a four-component, 32-bit packed unsigned integer format that has a 2-bit A component in bits 30..31, a 10-bit R component in bits 20..29, a 10-bit G component in bits 10..19, and a 10-bit B component in bits 0..9.
- VK_FORMAT_A2R10G10B10_SINT_PACK32 specifies a four-component, 32-bit packed signed integer format that has a 2-bit A component in bits 30..31, a 10-bit R component in bits 20..29, a 10-bit G component in bits 10..19, and a 10-bit B component in bits 0..9.
- VK_FORMAT_A2B10G10R10_UNORM_PACK32 specifies a four-component, 32-bit packed unsigned normalized format that has a 2-bit A component in bits 30..31, a 10-bit B component in bits 20..29, a 10-bit G component in bits 10..19, and a 10-bit R component in bits 0..9.
- VK_FORMAT_A2B10G10R10_SNORM_PACK32 specifies a four-component, 32-bit packed signed normalized format that has a 2-bit A component in bits 30..31, a 10-bit B component in bits 20..29, a 10-bit G component in bits 10..19, and a 10-bit R component in bits 0..9.
- VK_FORMAT_A2B10G10R10_USCALED_PACK32 specifies a four-component, 32-bit packed unsigned scaled integer format that has a 2-bit A component in bits 30..31, a 10-bit B component in bits 20..29, a 10-bit G component in bits 10..19, and a 10-bit R component in bits 0..9.
- VK_FORMAT_A2B10G10R10_SSCALED_PACK32 specifies a four-component, 32-bit packed signed scaled integer format that has a 2-bit A component in bits 30..31, a 10-bit B component in bits 20..29, a 10-bit G component in bits 10..19, and a 10-bit R component in bits 0..9.
- VK_FORMAT_A2B10G10R10_UINT_PACK32 specifies a four-component, 32-bit packed unsigned integer format that has a 2-bit A component in bits 30..31, a 10-bit B component in bits 20..29, a 10-bit G component in bits 10..19, and a 10-bit R component in bits 0..9.
- VK_FORMAT_A2B10G10R10_SINT_PACK32 specifies a four-component, 32-bit packed signed integer format that has a 2-bit A component in bits 30..31, a 10-bit B component in bits 20..29, a 10-bit G component in bits 10..19, and a 10-bit R component in bits 0..9.
- VK_FORMAT_R16_UNORM specifies a one-component, 16-bit unsigned normalized format that has a single 16-bit R component.
- VK_FORMAT_R16_SNORM specifies a one-component, 16-bit signed normalized format that has a single 16-bit R component.
- VK_FORMAT_R16_USCALED specifies a one-component, 16-bit unsigned scaled integer format that has a single 16-bit R component.
- VK_FORMAT_R16_SSCALED specifies a one-component, 16-bit signed scaled integer format that has a single 16-bit R component.
- VK_FORMAT_R16_UINT specifies a one-component, 16-bit unsigned integer format that has a single 16-bit R component.
- VK_FORMAT_R16_SINT specifies a one-component, 16-bit signed integer format that has a single 16-bit R component.
- VK_FORMAT_R16_SFLOAT specifies a one-component, 16-bit signed floating-point format that has a single 16-bit R component.

- VK_FORMAT_R16G16_UNORM specifies a two-component, 32-bit unsigned normalized format that has a 16-bit R component in bytes 0..1, and a 16-bit G component in bytes 2..3.
- VK_FORMAT_R16G16_SNORM specifies a two-component, 32-bit signed normalized format that has a 16-bit R component in bytes 0..1, and a 16-bit G component in bytes 2..3.
- VK_FORMAT_R16G16_USCALED specifies a two-component, 32-bit unsigned scaled integer format that has a 16-bit R component in bytes 0..1, and a 16-bit G component in bytes 2..3.
- VK_FORMAT_R16G16_SSCALED specifies a two-component, 32-bit signed scaled integer format that has a 16-bit R component in bytes 0..1, and a 16-bit G component in bytes 2..3.
- VK_FORMAT_R16G16_UINT specifies a two-component, 32-bit unsigned integer format that has a 16-bit R component in bytes 0..1, and a 16-bit G component in bytes 2..3.
- VK_FORMAT_R16G16_SINT specifies a two-component, 32-bit signed integer format that has a 16-bit R component in bytes 0..1, and a 16-bit G component in bytes 2..3.
- VK_FORMAT_R16G16_SFLOAT specifies a two-component, 32-bit signed floating-point format that has a 16-bit R component in bytes 0..1, and a 16-bit G component in bytes 2..3.
- VK_FORMAT_R16G16B16_UNORM specifies a three-component, 48-bit unsigned normalized format that has a 16-bit R component in bytes 0..1, a 16-bit G component in bytes 2..3, and a 16-bit B component in bytes 4..5.
- VK_FORMAT_R16G16B16_SNORM specifies a three-component, 48-bit signed normalized format that has a 16-bit R component in bytes 0..1, a 16-bit G component in bytes 2..3, and a 16-bit B component in bytes 4..5.
- VK_FORMAT_R16G16B16_USCALED specifies a three-component, 48-bit unsigned scaled integer format that has a 16-bit R component in bytes 0..1, a 16-bit G component in bytes 2..3, and a 16-bit B component in bytes 4..5.
- VK_FORMAT_R16G16B16_SSCALED specifies a three-component, 48-bit signed scaled integer format that has a 16-bit R component in bytes 0..1, a 16-bit G component in bytes 2..3, and a 16-bit B component in bytes 4..5.
- VK_FORMAT_R16G16B16_UINT specifies a three-component, 48-bit unsigned integer format that has a 16-bit R component in bytes 0..1, a 16-bit G component in bytes 2..3, and a 16-bit B component in bytes 4..5.
- VK_FORMAT_R16G16B16_SINT specifies a three-component, 48-bit signed integer format that has a 16-bit R component in bytes 0..1, a 16-bit G component in bytes 2..3, and a 16-bit B component in bytes 4..5.
- VK_FORMAT_R16G16B16_SFLOAT specifies a three-component, 48-bit signed floating-point format that has a 16-bit R component in bytes 0..1, a 16-bit G component in bytes 2..3, and a 16-bit B component in bytes 4..5.
- VK_FORMAT_R16G16B16A16_UNORM specifies a four-component, 64-bit unsigned normalized format that has a 16-bit R component in bytes 0..1, a 16-bit G component in bytes 2..3, a 16-bit B component in bytes 4..5, and a 16-bit A component in bytes 6..7.
- VK_FORMAT_R16G16B16A16_SNORM specifies a four-component, 64-bit signed normalized format that has a 16-bit R component in bytes 0..1, a 16-bit G component in bytes 2..3, a 16-bit B component in bytes 4..5, and a 16-bit A component in bytes 6..7.

- VK_FORMAT_R16G16B16A16_USCALED specifies a four-component, 64-bit unsigned scaled integer format that has a 16-bit R component in bytes 0..1, a 16-bit G component in bytes 2..3, a 16-bit B component in bytes 4..5, and a 16-bit A component in bytes 6..7.
- VK_FORMAT_R16G16B16A16_SSCALED specifies a four-component, 64-bit signed scaled integer format that has a 16-bit R component in bytes 0..1, a 16-bit G component in bytes 2..3, a 16-bit B component in bytes 4..5, and a 16-bit A component in bytes 6..7.
- VK_FORMAT_R16G16B16A16_UINT specifies a four-component, 64-bit unsigned integer format that has a 16-bit R component in bytes 0..1, a 16-bit G component in bytes 2..3, a 16-bit B component in bytes 4..5, and a 16-bit A component in bytes 6..7.
- VK_FORMAT_R16G16B16A16_SINT specifies a four-component, 64-bit signed integer format that has a 16-bit R component in bytes 0..1, a 16-bit G component in bytes 2..3, a 16-bit B component in bytes 4..5, and a 16-bit A component in bytes 6..7.
- VK_FORMAT_R16G16B16A16_SFLOAT specifies a four-component, 64-bit signed floating-point format that has a 16-bit R component in bytes 0..1, a 16-bit G component in bytes 2..3, a 16-bit B component in bytes 4..5, and a 16-bit A component in bytes 6..7.
- VK_FORMAT_R32_UINT specifies a one-component, 32-bit unsigned integer format that has a single 32-bit R component.
- VK_FORMAT_R32_SINT specifies a one-component, 32-bit signed integer format that has a single 32-bit R component.
- VK_FORMAT_R32_SFLOAT specifies a one-component, 32-bit signed floating-point format that has a single 32-bit R component.
- VK_FORMAT_R32G32_UINT specifies a two-component, 64-bit unsigned integer format that has a 32-bit R component in bytes 0..3, and a 32-bit G component in bytes 4..7.
- VK_FORMAT_R32G32_SINT specifies a two-component, 64-bit signed integer format that has a 32-bit R component in bytes 0..3, and a 32-bit G component in bytes 4..7.
- VK_FORMAT_R32G32_SFLOAT specifies a two-component, 64-bit signed floating-point format that has a 32-bit R component in bytes 0..3, and a 32-bit G component in bytes 4..7.
- VK_FORMAT_R32G32B32_UINT specifies a three-component, 96-bit unsigned integer format that has a 32-bit R component in bytes 0..3, a 32-bit G component in bytes 4..7, and a 32-bit B component in bytes 8..11.
- VK_FORMAT_R32G32B32_SINT specifies a three-component, 96-bit signed integer format that has a 32-bit R component in bytes 0..3, a 32-bit G component in bytes 4..7, and a 32-bit B component in bytes 8..11.
- VK_FORMAT_R32G32B32_SFLOAT specifies a three-component, 96-bit signed floating-point format that has a 32-bit R component in bytes 0..3, a 32-bit G component in bytes 4..7, and a 32-bit B component in bytes 8..11.
- VK_FORMAT_R32G32B32A32_UINT specifies a four-component, 128-bit unsigned integer format that has a 32-bit R component in bytes 0..3, a 32-bit G component in bytes 4..7, a 32-bit B component in bytes 8..11, and a 32-bit A component in bytes 12..15.
- VK_FORMAT_R32G32B32A32_SINT specifies a four-component, 128-bit signed integer format that has a 32-bit R component in bytes 0..3, a 32-bit G component in bytes 4..7, a 32-bit B component in bytes 8..11, and a 32-bit A component in bytes 12..15.

- VK_FORMAT_R32G32B32A32_SFLOAT specifies a four-component, 128-bit signed floating-point format that has a 32-bit R component in bytes 0..3, a 32-bit G component in bytes 4..7, a 32-bit B component in bytes 8..11, and a 32-bit A component in bytes 12..15.
- VK_FORMAT_R64_UINT specifies a one-component, 64-bit unsigned integer format that has a single 64-bit R component.
- VK_FORMAT_R64_SINT specifies a one-component, 64-bit signed integer format that has a single 64-bit R component.
- VK_FORMAT_R64_SFLOAT specifies a one-component, 64-bit signed floating-point format that has a single 64-bit R component.
- VK_FORMAT_R64G64_UINT specifies a two-component, 128-bit unsigned integer format that has a 64-bit R component in bytes 0..7, and a 64-bit G component in bytes 8..15.
- VK_FORMAT_R64G64_SINT specifies a two-component, 128-bit signed integer format that has a 64-bit R component in bytes 0..7, and a 64-bit G component in bytes 8..15.
- VK_FORMAT_R64G64_SFLOAT specifies a two-component, 128-bit signed floating-point format that has a 64-bit R component in bytes 0..7, and a 64-bit G component in bytes 8..15.
- VK_FORMAT_R64G64B64_UINT specifies a three-component, 192-bit unsigned integer format that has a 64-bit R component in bytes 0..7, a 64-bit G component in bytes 8..15, and a 64-bit B component in bytes 16..23.
- VK_FORMAT_R64G64B64_SINT specifies a three-component, 192-bit signed integer format that has a 64-bit R component in bytes 0..7, a 64-bit G component in bytes 8..15, and a 64-bit B component in bytes 16..23.
- VK_FORMAT_R64G64B64_SFLOAT specifies a three-component, 192-bit signed floating-point format that has a 64-bit R component in bytes 0..7, a 64-bit G component in bytes 8..15, and a 64-bit B component in bytes 16..23.
- VK_FORMAT_R64G64B64A64_UINT specifies a four-component, 256-bit unsigned integer format that has a 64-bit R component in bytes 0..7, a 64-bit G component in bytes 8..15, a 64-bit B component in bytes 16..23, and a 64-bit A component in bytes 24..31.
- VK_FORMAT_R64G64B64A64_SINT specifies a four-component, 256-bit signed integer format that has a 64-bit R component in bytes 0..7, a 64-bit G component in bytes 8..15, a 64-bit B component in bytes 16..23, and a 64-bit A component in bytes 24..31.
- VK_FORMAT_R64G64B64A64_SFLOAT specifies a four-component, 256-bit signed floating-point format that has a 64-bit R component in bytes 0..7, a 64-bit G component in bytes 8..15, a 64-bit B component in bytes 16..23, and a 64-bit A component in bytes 24..31.
- VK_FORMAT_B10G11R11_UFLOAT_PACK32 specifies a three-component, 32-bit packed unsigned floating-point format that has a 10-bit B component in bits 22..31, an 11-bit G component in bits 11..21, an 11-bit R component in bits 0..10. See Unsigned 10-Bit Floating-Point Numbers and Unsigned 11-Bit Floating-Point Numbers.
- VK_FORMAT_E5B9G9R9_UFLOAT_PACK32 specifies a three-component, 32-bit packed unsigned floating-point format that has a 5-bit shared exponent in bits 27..31, a 9-bit B component mantissa in bits 18..26, a 9-bit G component mantissa in bits 9..17, and a 9-bit R component mantissa in bits 0..8.
- VK_FORMAT_D16_UNORM specifies a one-component, 16-bit unsigned normalized format that has a single 16-bit depth component.

- VK_FORMAT_X8_D24_UNORM_PACK32 specifies a two-component, 32-bit format that has 24 unsigned normalized bits in the depth component and, **optionally**, 8 bits that are unused.
- VK_FORMAT_D32_SFLOAT specifies a one-component, 32-bit signed floating-point format that has 32 bits in the depth component.
- VK_FORMAT_S8_UINT specifies a one-component, 8-bit unsigned integer format that has 8 bits in the stencil component.
- VK_FORMAT_D16_UNORM_S8_UINT specifies a two-component, 24-bit format that has 16 unsigned normalized bits in the depth component and 8 unsigned integer bits in the stencil component.
- VK_FORMAT_D24_UNORM_S8_UINT specifies a two-component, 32-bit packed format that has 8 unsigned integer bits in the stencil component, and 24 unsigned normalized bits in the depth component.
- VK_FORMAT_D32_SFLOAT_S8_UINT specifies a two-component format that has 32 signed float bits in the depth component and 8 unsigned integer bits in the stencil component. There are **optionally** 24 bits that are unused.
- VK_FORMAT_BC1_RGB_UNORM_BLOCK specifies a three-component, block-compressed format where each 64-bit compressed texel block encodes a 4×4 rectangle of unsigned normalized RGB texel data. This format has no alpha and is considered opaque.
- VK_FORMAT_BC1_RGB_SRGB_BLOCK specifies a three-component, block-compressed format where each 64-bit compressed texel block encodes a 4×4 rectangle of unsigned normalized RGB texel data with sRGB nonlinear encoding. This format has no alpha and is considered opaque.
- VK_FORMAT_BC1_RGBA_UNORM_BLOCK specifies a four-component, block-compressed format where each 64-bit compressed texel block encodes a 4×4 rectangle of unsigned normalized RGB texel data, and provides 1 bit of alpha.
- VK_FORMAT_BC1_RGBA_SRGB_BLOCK specifies a four-component, block-compressed format where each 64-bit compressed texel block encodes a 4×4 rectangle of unsigned normalized RGB texel data with sRGB nonlinear encoding, and provides 1 bit of alpha.
- VK_FORMAT_BC2_UNORM_BLOCK specifies a four-component, block-compressed format where each 128-bit compressed texel block encodes a 4×4 rectangle of unsigned normalized RGBA texel data with the first 64 bits encoding alpha values followed by 64 bits encoding RGB values.
- VK_FORMAT_BC2_SRGB_BLOCK specifies a four-component, block-compressed format where each 128-bit compressed texel block encodes a 4×4 rectangle of unsigned normalized RGBA texel data with the first 64 bits encoding alpha values followed by 64 bits encoding RGB values with sRGB nonlinear encoding.
- VK_FORMAT_BC3_UNORM_BLOCK specifies a four-component, block-compressed format where each 128-bit compressed texel block encodes a 4×4 rectangle of unsigned normalized RGBA texel data with the first 64 bits encoding alpha values followed by 64 bits encoding RGB values.
- VK_FORMAT_BC3_SRGB_BLOCK specifies a four-component, block-compressed format where each 128-bit compressed texel block encodes a 4×4 rectangle of unsigned normalized RGBA texel data with the first 64 bits encoding alpha values followed by 64 bits encoding RGB values with sRGB nonlinear encoding.
- VK_FORMAT_BC4_UNORM_BLOCK specifies a one-component, block-compressed format where each 64-bit compressed texel block encodes a 4×4 rectangle of unsigned normalized red texel data.

- VK_FORMAT_BC4_SNORM_BLOCK specifies a one-component, block-compressed format where each 64-bit compressed texel block encodes a 4×4 rectangle of signed normalized red texel data.
- VK_FORMAT_BC5_UNORM_BLOCK specifies a two-component, block-compressed format where each 128-bit compressed texel block encodes a 4×4 rectangle of unsigned normalized RG texel data with the first 64 bits encoding red values followed by 64 bits encoding green values.
- VK_FORMAT_BC5_SNORM_BLOCK specifies a two-component, block-compressed format where each 128-bit compressed texel block encodes a 4×4 rectangle of signed normalized RG texel data with the first 64 bits encoding red values followed by 64 bits encoding green values.
- VK_FORMAT_BC6H_UFLOAT_BLOCK specifies a three-component, block-compressed format where each 128-bit compressed texel block encodes a 4×4 rectangle of unsigned floating-point RGB texel data.
- VK_FORMAT_BC6H_SFLOAT_BLOCK specifies a three-component, block-compressed format where each 128-bit compressed texel block encodes a 4×4 rectangle of signed floating-point RGB texel data.
- VK_FORMAT_BC7_UNORM_BLOCK specifies a four-component, block-compressed format where each 128-bit compressed texel block encodes a 4×4 rectangle of unsigned normalized RGBA texel data.
- VK_FORMAT_BC7_SRGB_BLOCK specifies a four-component, block-compressed format where each 128-bit compressed texel block encodes a 4×4 rectangle of unsigned normalized RGBA texel data with sRGB nonlinear encoding applied to the RGB components.
- VK_FORMAT_ETC2_R8G8B8_UNORM_BLOCK specifies a three-component, ETC2 compressed format where each 64-bit compressed texel block encodes a 4×4 rectangle of unsigned normalized RGB texel data. This format has no alpha and is considered opaque.
- VK_FORMAT_ETC2_R8G8B8_SRGB_BLOCK specifies a three-component, ETC2 compressed format where each 64-bit compressed texel block encodes a 4×4 rectangle of unsigned normalized RGB texel data with sRGB nonlinear encoding. This format has no alpha and is considered opaque.
- VK_FORMAT_ETC2_R8G8B8A1_UNORM_BLOCK specifies a four-component, ETC2 compressed format where each 64-bit compressed texel block encodes a 4×4 rectangle of unsigned normalized RGB texel data, and provides 1 bit of alpha.
- VK_FORMAT_ETC2_R8G8B8A1_SRGB_BLOCK specifies a four-component, ETC2 compressed format where each 64-bit compressed texel block encodes a 4×4 rectangle of unsigned normalized RGB texel data with sRGB nonlinear encoding, and provides 1 bit of alpha.
- VK_FORMAT_ETC2_R8G8B8A8_UNORM_BLOCK specifies a four-component, ETC2 compressed format where each 128-bit compressed texel block encodes a 4×4 rectangle of unsigned normalized RGBA texel data with the first 64 bits encoding alpha values followed by 64 bits encoding RGB values.
- VK_FORMAT_ETC2_R8G8B8A8_SRGB_BLOCK specifies a four-component, ETC2 compressed format where each 128-bit compressed texel block encodes a 4×4 rectangle of unsigned normalized RGBA texel data with the first 64 bits encoding alpha values followed by 64 bits encoding RGB values with sRGB nonlinear encoding applied.
- VK_FORMAT_EAC_R11_UNORM_BLOCK specifies a one-component, ETC2 compressed format where each 64-bit compressed texel block encodes a 4×4 rectangle of unsigned normalized red texel data.
- VK_FORMAT_EAC_R11_SNORM_BLOCK specifies a one-component, ETC2 compressed format where each

64-bit compressed texel block encodes a 4×4 rectangle of signed normalized red texel data.

- VK_FORMAT_EAC_R11G11_UNORM_BLOCK specifies a two-component, ETC2 compressed format where each 128-bit compressed texel block encodes a 4×4 rectangle of unsigned normalized RG texel data with the first 64 bits encoding red values followed by 64 bits encoding green values.
- VK_FORMAT_EAC_R11G11_SNORM_BLOCK specifies a two-component, ETC2 compressed format where each 128-bit compressed texel block encodes a 4×4 rectangle of signed normalized RG texel data with the first 64 bits encoding red values followed by 64 bits encoding green values.
- VK_FORMAT_ASTC_4x4_UNORM_BLOCK specifies a four-component, ASTC compressed format where each 128-bit compressed texel block encodes a 4×4 rectangle of unsigned normalized RGBA texel data.
- VK_FORMAT_ASTC_4x4_SRGB_BLOCK specifies a four-component, ASTC compressed format where each 128-bit compressed texel block encodes a 4×4 rectangle of unsigned normalized RGBA texel data with sRGB nonlinear encoding applied to the RGB components.
- VK_FORMAT_ASTC_4x4_SFLOAT_BLOCK specifies a four-component, ASTC compressed format where each 128-bit compressed texel block encodes a 4×4 rectangle of signed floating-point RGBA texel data.
- VK_FORMAT_ASTC_5x4_UNORM_BLOCK specifies a four-component, ASTC compressed format where each 128-bit compressed texel block encodes a 5×4 rectangle of unsigned normalized RGBA texel data.
- VK_FORMAT_ASTC_5x4_SRGB_BLOCK specifies a four-component, ASTC compressed format where each 128-bit compressed texel block encodes a 5×4 rectangle of unsigned normalized RGBA texel data with sRGB nonlinear encoding applied to the RGB components.
- VK_FORMAT_ASTC_5x4_SFLOAT_BLOCK specifies a four-component, ASTC compressed format where each 128-bit compressed texel block encodes a 5×4 rectangle of signed floating-point RGBA texel data.
- VK_FORMAT_ASTC_5x5_UNORM_BLOCK specifies a four-component, ASTC compressed format where each 128-bit compressed texel block encodes a 5×5 rectangle of unsigned normalized RGBA texel data.
- VK_FORMAT_ASTC_5x5_SRGB_BLOCK specifies a four-component, ASTC compressed format where each 128-bit compressed texel block encodes a 5×5 rectangle of unsigned normalized RGBA texel data with sRGB nonlinear encoding applied to the RGB components.
- VK_FORMAT_ASTC_5x5_SFLOAT_BLOCK specifies a four-component, ASTC compressed format where each 128-bit compressed texel block encodes a 5×5 rectangle of signed floating-point RGBA texel data.
- VK_FORMAT_ASTC_6x5_UNORM_BLOCK specifies a four-component, ASTC compressed format where each 128-bit compressed texel block encodes a 6×5 rectangle of unsigned normalized RGBA texel data.
- VK_FORMAT_ASTC_6x5_SRGB_BLOCK specifies a four-component, ASTC compressed format where each 128-bit compressed texel block encodes a 6×5 rectangle of unsigned normalized RGBA texel data with sRGB nonlinear encoding applied to the RGB components.
- VK_FORMAT_ASTC_6x5_SFLOAT_BLOCK specifies a four-component, ASTC compressed format where each 128-bit compressed texel block encodes a 6×5 rectangle of signed floating-point RGBA texel

data.

- VK_FORMAT_ASTC_6x6_UNORM_BLOCK specifies a four-component, ASTC compressed format where each 128-bit compressed texel block encodes a 6×6 rectangle of unsigned normalized RGBA texel data.
- VK_FORMAT_ASTC_6x6_SRGB_BLOCK specifies a four-component, ASTC compressed format where each 128-bit compressed texel block encodes a 6×6 rectangle of unsigned normalized RGBA texel data with sRGB nonlinear encoding applied to the RGB components.
- VK_FORMAT_ASTC_6x6_SFLOAT_BLOCK specifies a four-component, ASTC compressed format where each 128-bit compressed texel block encodes a 6×6 rectangle of signed floating-point RGBA texel data.
- VK_FORMAT_ASTC_8x5_UNORM_BLOCK specifies a four-component, ASTC compressed format where each 128-bit compressed texel block encodes an 8×5 rectangle of unsigned normalized RGBA texel data.
- VK_FORMAT_ASTC_8x5_SRGB_BLOCK specifies a four-component, ASTC compressed format where each 128-bit compressed texel block encodes an 8×5 rectangle of unsigned normalized RGBA texel data with sRGB nonlinear encoding applied to the RGB components.
- VK_FORMAT_ASTC_8x5_SFLOAT_BLOCK specifies a four-component, ASTC compressed format where each 128-bit compressed texel block encodes a 8×5 rectangle of signed floating-point RGBA texel data.
- VK_FORMAT_ASTC_8x6_UNORM_BLOCK specifies a four-component, ASTC compressed format where each 128-bit compressed texel block encodes an 8×6 rectangle of unsigned normalized RGBA texel data.
- VK_FORMAT_ASTC_8x6_SRGB_BLOCK specifies a four-component, ASTC compressed format where each 128-bit compressed texel block encodes an 8×6 rectangle of unsigned normalized RGBA texel data with sRGB nonlinear encoding applied to the RGB components.
- VK_FORMAT_ASTC_8x6_SFLOAT_BLOCK specifies a four-component, ASTC compressed format where each 128-bit compressed texel block encodes a 8×6 rectangle of signed floating-point RGBA texel data.
- VK_FORMAT_ASTC_8x8_UNORM_BLOCK specifies a four-component, ASTC compressed format where each 128-bit compressed texel block encodes an 8×8 rectangle of unsigned normalized RGBA texel data.
- VK_FORMAT_ASTC_8x8_SRGB_BLOCK specifies a four-component, ASTC compressed format where each 128-bit compressed texel block encodes an 8×8 rectangle of unsigned normalized RGBA texel data with sRGB nonlinear encoding applied to the RGB components.
- VK_FORMAT_ASTC_8x8_SFLOAT_BLOCK specifies a four-component, ASTC compressed format where each 128-bit compressed texel block encodes a 8×8 rectangle of signed floating-point RGBA texel data.
- VK_FORMAT_ASTC_10x5_UNORM_BLOCK specifies a four-component, ASTC compressed format where each 128-bit compressed texel block encodes a 10×5 rectangle of unsigned normalized RGBA texel data.
- VK_FORMAT_ASTC_10x5_SRGB_BLOCK specifies a four-component, ASTC compressed format where each 128-bit compressed texel block encodes a 10×5 rectangle of unsigned normalized RGBA

texel data with sRGB nonlinear encoding applied to the RGB components.

- VK_FORMAT_ASTC_10x5_SFLOAT_BLOCK specifies a four-component, ASTC compressed format where each 128-bit compressed texel block encodes a 10×5 rectangle of signed floating-point RGBA texel data.
- VK_FORMAT_ASTC_10x6_UNORM_BLOCK specifies a four-component, ASTC compressed format where each 128-bit compressed texel block encodes a 10×6 rectangle of unsigned normalized RGBA texel data.
- VK_FORMAT_ASTC_10x6_SRGB_BLOCK specifies a four-component, ASTC compressed format where each 128-bit compressed texel block encodes a 10×6 rectangle of unsigned normalized RGBA texel data with sRGB nonlinear encoding applied to the RGB components.
- VK_FORMAT_ASTC_10x6_SFLOAT_BLOCK specifies a four-component, ASTC compressed format where each 128-bit compressed texel block encodes a 10×6 rectangle of signed floating-point RGBA texel data.
- VK_FORMAT_ASTC_10x8_UNORM_BLOCK specifies a four-component, ASTC compressed format where each 128-bit compressed texel block encodes a 10×8 rectangle of unsigned normalized RGBA texel data.
- VK_FORMAT_ASTC_10x8_SRGB_BLOCK specifies a four-component, ASTC compressed format where each 128-bit compressed texel block encodes a 10×8 rectangle of unsigned normalized RGBA texel data with sRGB nonlinear encoding applied to the RGB components.
- VK_FORMAT_ASTC_10x8_SFLOAT_BLOCK specifies a four-component, ASTC compressed format where each 128-bit compressed texel block encodes a 10×8 rectangle of signed floating-point RGBA texel data.
- VK_FORMAT_ASTC_10x10_UNORM_BLOCK specifies a four-component, ASTC compressed format where each 128-bit compressed texel block encodes a 10×10 rectangle of unsigned normalized RGBA texel data.
- VK_FORMAT_ASTC_10x10_SRGB_BLOCK specifies a four-component, ASTC compressed format where each 128-bit compressed texel block encodes a 10×10 rectangle of unsigned normalized RGBA texel data with sRGB nonlinear encoding applied to the RGB components.
- VK_FORMAT_ASTC_10x10_SFLOAT_BLOCK specifies a four-component, ASTC compressed format where each 128-bit compressed texel block encodes a 10×10 rectangle of signed floating-point RGBA texel data.
- VK_FORMAT_ASTC_12x10_UNORM_BLOCK specifies a four-component, ASTC compressed format where each 128-bit compressed texel block encodes a 12×10 rectangle of unsigned normalized RGBA texel data.
- VK_FORMAT_ASTC_12x10_SRGB_BLOCK specifies a four-component, ASTC compressed format where each 128-bit compressed texel block encodes a 12×10 rectangle of unsigned normalized RGBA texel data with sRGB nonlinear encoding applied to the RGB components.
- VK_FORMAT_ASTC_12x10_SFLOAT_BLOCK specifies a four-component, ASTC compressed format where each 128-bit compressed texel block encodes a 12×10 rectangle of signed floating-point RGBA texel data.
- VK_FORMAT_ASTC_12x12_UNORM_BLOCK specifies a four-component, ASTC compressed format where each 128-bit compressed texel block encodes a 12×12 rectangle of unsigned normalized RGBA

texel data.

- VK_FORMAT_ASTC_12x12_SRGB_BLOCK specifies a four-component, ASTC compressed format where each 128-bit compressed texel block encodes a 12×12 rectangle of unsigned normalized RGBA texel data with sRGB nonlinear encoding applied to the RGB components.
- VK_FORMAT_ASTC_12x12_SFLOAT_BLOCK specifies a four-component, ASTC compressed format where each 128-bit compressed texel block encodes a 12×12 rectangle of signed floating-point RGBA texel data.
- VK_FORMAT_G8B8G8R8_422_UNORM specifies a four-component, 32-bit format containing a pair of G components, an R component, and a B component, collectively encoding a 2×1 rectangle of unsigned normalized RGB texel data. One G value is present at each *i* coordinate, with the B and R values shared across both G values and thus recorded at half the horizontal resolution of the image. This format has an 8-bit G component for the even *i* coordinate in byte 0, an 8-bit B component in byte 1, an 8-bit G component for the odd *i* coordinate in byte 2, and an 8-bit R component in byte 3. This format only supports images with a width that is a multiple of two. For the purposes of the constraints on copy extents, this format is treated as a compressed format with a 2×1 compressed texel block.
- VK_FORMAT_B8G8R8G8_422_UNORM specifies a four-component, 32-bit format containing a pair of G components, an R component, and a B component, collectively encoding a 2×1 rectangle of unsigned normalized RGB texel data. One G value is present at each *i* coordinate, with the B and R values shared across both G values and thus recorded at half the horizontal resolution of the image. This format has an 8-bit B component in byte 0, an 8-bit G component for the even *i* coordinate in byte 1, an 8-bit R component in byte 2, and an 8-bit G component for the odd *i* coordinate in byte 3. This format only supports images with a width that is a multiple of two. For the purposes of the constraints on copy extents, this format is treated as a compressed format with a 2×1 compressed texel block.
- VK_FORMAT_G8_B8_R8_3PLANE_420_UNORM specifies an unsigned normalized *multi-planar format* that has an 8-bit G component in plane 0, an 8-bit B component in plane 1, and an 8-bit R component in plane 2. The horizontal and vertical dimensions of the R and B planes are halved relative to the image dimensions, and each R and B component is shared with the G components for which $\lfloor i_G \times 0.5 \rfloor = i_B = i_R$ and $\lfloor j_G \times 0.5 \rfloor = j_B = j_R$. The location of each plane when this image is in linear layout can be determined via vkGetImageSubresourceLayout, using VK_IMAGE_ASPECT_PLANE_0_BIT for the G plane, VK_IMAGE_ASPECT_PLANE_1_BIT for the B plane, and VK_IMAGE_ASPECT_PLANE_2_BIT for the R plane. This format only supports images with a width and height that is a multiple of two.
- VK_FORMAT_G8_B8R8_2PLANE_420_UNORM specifies an unsigned normalized *multi-planar format* that has an 8-bit G component in plane 0, and a two-component, 16-bit BR plane 1 consisting of an 8-bit B component in byte 0 and an 8-bit R component in byte 1. The horizontal and vertical dimensions of the BR plane are halved relative to the image dimensions, and each R and B value is shared with the G components for which $\lfloor i_G \times 0.5 \rfloor = i_B = i_R$ and $\lfloor j_G \times 0.5 \rfloor = j_B = j_R$. The location of each plane when this image is in linear layout can be determined via vkGetImageSubresourceLayout, using VK_IMAGE_ASPECT_PLANE_0_BIT for the G plane, and VK_IMAGE_ASPECT_PLANE_1_BIT for the BR plane. This format only supports images with a width and height that is a multiple of two.
- VK_FORMAT_G8_B8_R8_3PLANE_422_UNORM specifies an unsigned normalized *multi-planar format* that has an 8-bit G component in plane 0, an 8-bit B component in plane 1, and an 8-bit R component

in plane 2. The horizontal dimension of the R and B plane is halved relative to the image dimensions, and each R and B value is shared with the G components for which $\lfloor i_G \times 0.5 \rfloor = i_B = i_R$. The location of each plane when this image is in linear layout can be determined via vkGetImageSubresourceLayout, using VK_IMAGE_ASPECT_PLANE_0_BIT for the G plane, VK_IMAGE_ASPECT_PLANE_1_BIT for the B plane, and VK_IMAGE_ASPECT_PLANE_2_BIT for the R plane. This format only supports images with a width that is a multiple of two.

- VK_FORMAT_G8_B8R8_2PLANE_422_UNORM specifies an unsigned normalized *multi-planar format* that has an 8-bit G component in plane 0, and a two-component, 16-bit BR plane 1 consisting of an 8-bit B component in byte 0 and an 8-bit R component in byte 1. The horizontal dimension of the BR plane is halved relative to the image dimensions, and each R and B value is shared with the G components for which $\lfloor i_G \times 0.5 \rfloor = i_B = i_R$. The location of each plane when this image is in linear layout can be determined via vkGetImageSubresourceLayout, using VK_IMAGE_ASPECT_PLANE_0_BIT for the G plane, and VK_IMAGE_ASPECT_PLANE_1_BIT for the BR plane. This format only supports images with a width that is a multiple of two.
- VK_FORMAT_G8_B8_R8_3PLANE_444_UNORM specifies an unsigned normalized *multi-planar format* that has an 8-bit G component in plane 0, an 8-bit B component in plane 1, and an 8-bit R component in plane 2. Each plane has the same dimensions and each R, G and B component contributes to a single texel. The location of each plane when this image is in linear layout can be determined via vkGetImageSubresourceLayout, using VK_IMAGE_ASPECT_PLANE_0_BIT for the G plane, VK_IMAGE_ASPECT_PLANE_1_BIT for the B plane, and VK_IMAGE_ASPECT_PLANE_2_BIT for the R plane.
- VK_FORMAT_R10X6_UNORM_PACK16 specifies a one-component, 16-bit unsigned normalized format that has a single 10-bit R component in the top 10 bits of a 16-bit word, with the bottom 6 bits unused.
- VK_FORMAT_R10X6G10X6_UNORM_2PACK16 specifies a two-component, 32-bit unsigned normalized format that has a 10-bit R component in the top 10 bits of the word in bytes 0..1, and a 10-bit G component in the top 10 bits of the word in bytes 2..3, with the bottom 6 bits of each word unused.
- VK_FORMAT_R10X6G10X6B10X6A10X6_UNORM_4PACK16 specifies a four-component, 64-bit unsigned normalized format that has a 10-bit R component in the top 10 bits of the word in bytes 0..1, a 10-bit G component in the top 10 bits of the word in bytes 2..3, a 10-bit B component in the top 10 bits of the word in bytes 4..5, and a 10-bit A component in the top 10 bits of the word in bytes 6..7, with the bottom 6 bits of each word unused.
- VK_FORMAT_G10X6B10X6G10X6R10X6_422_UNORM_4PACK16 specifies a four-component, 64-bit format containing a pair of G components, an R component, and a B component, collectively encoding a 2×1 rectangle of unsigned normalized RGB texel data. One G value is present at each $i$ coordinate, with the B and R values shared across both G values and thus recorded at half the horizontal resolution of the image. This format has a 10-bit G component for the even $i$ coordinate in the top 10 bits of the word in bytes 0..1, a 10-bit B component in the top 10 bits of the word in bytes 2..3, a 10-bit G component for the odd $i$ coordinate in the top 10 bits of the word in bytes 4..5, and a 10-bit R component in the top 10 bits of the word in bytes 6..7, with the bottom 6 bits of each word unused. This format only supports images with a width that is a multiple of two. For the purposes of the constraints on copy extents, this format is treated as a compressed format with a 2×1 compressed texel block.
- VK_FORMAT_B10X6G10X6R10X6G10X6_422_UNORM_4PACK16 specifies a four-component, 64-bit format containing a pair of G components, an R component, and a B component, collectively encoding a

2×1 rectangle of unsigned normalized RGB texel data. One G value is present at each *i* coordinate, with the B and R values shared across both G values and thus recorded at half the horizontal resolution of the image. This format has a 10-bit B component in the top 10 bits of the word in bytes 0..1, a 10-bit G component for the even *i* coordinate in the top 10 bits of the word in bytes 2..3, a 10-bit R component in the top 10 bits of the word in bytes 4..5, and a 10-bit G component for the odd *i* coordinate in the top 10 bits of the word in bytes 6..7, with the bottom 6 bits of each word unused. This format only supports images with a width that is a multiple of two. For the purposes of the constraints on copy extents, this format is treated as a compressed format with a 2×1 compressed texel block.

- `VK_FORMAT_G10X6_B10X6_R10X6_3PLANE_420_UNORM_3PACK16` specifies an unsigned normalized *multi-planar format* that has a 10-bit G component in the top 10 bits of each 16-bit word of plane 0, a 10-bit B component in the top 10 bits of each 16-bit word of plane 1, and a 10-bit R component in the top 10 bits of each 16-bit word of plane 2, with the bottom 6 bits of each word unused. The horizontal and vertical dimensions of the R and B planes are halved relative to the image dimensions, and each R and B component is shared with the G components for which $\lfloor i_G \times 0.5 \rfloor = i_B = i_R$ and $\lfloor j_G \times 0.5 \rfloor = j_B = j_R$. The location of each plane when this image is in linear layout can be determined via vkGetImageSubresourceLayout, using `VK_IMAGE_ASPECT_PLANE_0_BIT` for the G plane, `VK_IMAGE_ASPECT_PLANE_1_BIT` for the B plane, and `VK_IMAGE_ASPECT_PLANE_2_BIT` for the R plane. This format only supports images with a width and height that is a multiple of two.
- `VK_FORMAT_G10X6_B10X6R10X6_2PLANE_420_UNORM_3PACK16` specifies an unsigned normalized *multi-planar format* that has a 10-bit G component in the top 10 bits of each 16-bit word of plane 0, and a two-component, 32-bit BR plane 1 consisting of a 10-bit B component in the top 10 bits of the word in bytes 0..1, and a 10-bit R component in the top 10 bits of the word in bytes 2..3, with the bottom 6 bits of each word unused. The horizontal and vertical dimensions of the BR plane are halved relative to the image dimensions, and each R and B value is shared with the G components for which $\lfloor i_G \times 0.5 \rfloor = i_B = i_R$ and $\lfloor j_G \times 0.5 \rfloor = j_B = j_R$. The location of each plane when this image is in linear layout can be determined via vkGetImageSubresourceLayout, using `VK_IMAGE_ASPECT_PLANE_0_BIT` for the G plane, and `VK_IMAGE_ASPECT_PLANE_1_BIT` for the BR plane. This format only supports images with a width and height that is a multiple of two.
- `VK_FORMAT_G10X6_B10X6_R10X6_3PLANE_422_UNORM_3PACK16` specifies an unsigned normalized *multi-planar format* that has a 10-bit G component in the top 10 bits of each 16-bit word of plane 0, a 10-bit B component in the top 10 bits of each 16-bit word of plane 1, and a 10-bit R component in the top 10 bits of each 16-bit word of plane 2, with the bottom 6 bits of each word unused. The horizontal dimension of the R and B plane is halved relative to the image dimensions, and each R and B value is shared with the G components for which $\lfloor i_G \times 0.5 \rfloor = i_B = i_R$. The location of each plane when this image is in linear layout can be determined via vkGetImageSubresourceLayout, using `VK_IMAGE_ASPECT_PLANE_0_BIT` for the G plane, `VK_IMAGE_ASPECT_PLANE_1_BIT` for the B plane, and `VK_IMAGE_ASPECT_PLANE_2_BIT` for the R plane. This format only supports images with a width that is a multiple of two.
- `VK_FORMAT_G10X6_B10X6R10X6_2PLANE_422_UNORM_3PACK16` specifies an unsigned normalized *multi-planar format* that has a 10-bit G component in the top 10 bits of each 16-bit word of plane 0, and a two-component, 32-bit BR plane 1 consisting of a 10-bit B component in the top 10 bits of the word in bytes 0..1, and a 10-bit R component in the top 10 bits of the word in bytes 2..3, with the bottom 6 bits of each word unused. The horizontal dimension of the BR plane is halved relative to the image dimensions, and each R and B value is shared with the G components for

which $\lfloor i_G \times 0.5 \rfloor = i_B = i_R$. The location of each plane when this image is in linear layout can be determined via vkGetImageSubresourceLayout, using VK_IMAGE_ASPECT_PLANE_0_BIT for the G plane, and VK_IMAGE_ASPECT_PLANE_1_BIT for the BR plane. This format only supports images with a width that is a multiple of two.

- VK_FORMAT_G10X6_B10X6_R10X6_3PLANE_444_UNORM_3PACK16 specifies an unsigned normalized *multi-planar format* that has a 10-bit G component in the top 10 bits of each 16-bit word of plane 0, a 10-bit B component in the top 10 bits of each 16-bit word of plane 1, and a 10-bit R component in the top 10 bits of each 16-bit word of plane 2, with the bottom 6 bits of each word unused. Each plane has the same dimensions and each R, G and B component contributes to a single texel. The location of each plane when this image is in linear layout can be determined via vkGetImageSubresourceLayout, using VK_IMAGE_ASPECT_PLANE_0_BIT for the G plane, VK_IMAGE_ASPECT_PLANE_1_BIT for the B plane, and VK_IMAGE_ASPECT_PLANE_2_BIT for the R plane.
- VK_FORMAT_R12X4_UNORM_PACK16 specifies a one-component, 16-bit unsigned normalized format that has a single 12-bit R component in the top 12 bits of a 16-bit word, with the bottom 4 bits unused.
- VK_FORMAT_R12X4G12X4_UNORM_2PACK16 specifies a two-component, 32-bit unsigned normalized format that has a 12-bit R component in the top 12 bits of the word in bytes 0..1, and a 12-bit G component in the top 12 bits of the word in bytes 2..3, with the bottom 4 bits of each word unused.
- VK_FORMAT_R12X4G12X4B12X4A12X4_UNORM_4PACK16 specifies a four-component, 64-bit unsigned normalized format that has a 12-bit R component in the top 12 bits of the word in bytes 0..1, a 12-bit G component in the top 12 bits of the word in bytes 2..3, a 12-bit B component in the top 12 bits of the word in bytes 4..5, and a 12-bit A component in the top 12 bits of the word in bytes 6..7, with the bottom 4 bits of each word unused.
- VK_FORMAT_G12X4B12X4G12X4R12X4_422_UNORM_4PACK16 specifies a four-component, 64-bit format containing a pair of G components, an R component, and a B component, collectively encoding a 2×1 rectangle of unsigned normalized RGB texel data. One G value is present at each $i$ coordinate, with the B and R values shared across both G values and thus recorded at half the horizontal resolution of the image. This format has a 12-bit G component for the even $i$ coordinate in the top 12 bits of the word in bytes 0..1, a 12-bit B component in the top 12 bits of the word in bytes 2..3, a 12-bit G component for the odd $i$ coordinate in the top 12 bits of the word in bytes 4..5, and a 12-bit R component in the top 12 bits of the word in bytes 6..7, with the bottom 4 bits of each word unused. This format only supports images with a width that is a multiple of two. For the purposes of the constraints on copy extents, this format is treated as a compressed format with a 2×1 compressed texel block.
- VK_FORMAT_B12X4G12X4R12X4G12X4_422_UNORM_4PACK16 specifies a four-component, 64-bit format containing a pair of G components, an R component, and a B component, collectively encoding a 2×1 rectangle of unsigned normalized RGB texel data. One G value is present at each $i$ coordinate, with the B and R values shared across both G values and thus recorded at half the horizontal resolution of the image. This format has a 12-bit B component in the top 12 bits of the word in bytes 0..1, a 12-bit G component for the even $i$ coordinate in the top 12 bits of the word in bytes 2..3, a 12-bit R component in the top 12 bits of the word in bytes 4..5, and a 12-bit G component for the odd $i$ coordinate in the top 12 bits of the word in bytes 6..7, with the bottom 4 bits of each word unused. This format only supports images with a width that is a multiple of two. For the purposes of the constraints on copy extents, this format is treated as a compressed

format with a 2×1 compressed texel block.

- VK_FORMAT_G12X4_B12X4_R12X4_3PLANE_420_UNORM_3PACK16 specifies an unsigned normalized *multi-planar format* that has a 12-bit G component in the top 12 bits of each 16-bit word of plane 0, a 12-bit B component in the top 12 bits of each 16-bit word of plane 1, and a 12-bit R component in the top 12 bits of each 16-bit word of plane 2, with the bottom 4 bits of each word unused. The horizontal and vertical dimensions of the R and B planes are halved relative to the image dimensions, and each R and B component is shared with the G components for which $\lfloor i_G \times 0.5 \rfloor = i_B = i_R$ and $\lfloor j_G \times 0.5 \rfloor = j_B = j_R$. The location of each plane when this image is in linear layout can be determined via vkGetImageSubresourceLayout, using VK_IMAGE_ASPECT_PLANE_0_BIT for the G plane, VK_IMAGE_ASPECT_PLANE_1_BIT for the B plane, and VK_IMAGE_ASPECT_PLANE_2_BIT for the R plane. This format only supports images with a width and height that is a multiple of two.
- VK_FORMAT_G12X4_B12X4R12X4_2PLANE_420_UNORM_3PACK16 specifies an unsigned normalized *multi-planar format* that has a 12-bit G component in the top 12 bits of each 16-bit word of plane 0, and a two-component, 32-bit BR plane 1 consisting of a 12-bit B component in the top 12 bits of the word in bytes 0..1, and a 12-bit R component in the top 12 bits of the word in bytes 2..3, with the bottom 4 bits of each word unused. The horizontal and vertical dimensions of the BR plane are halved relative to the image dimensions, and each R and B value is shared with the G components for which $\lfloor i_G \times 0.5 \rfloor = i_B = i_R$ and $\lfloor j_G \times 0.5 \rfloor = j_B = j_R$. The location of each plane when this image is in linear layout can be determined via vkGetImageSubresourceLayout, using VK_IMAGE_ASPECT_PLANE_0_BIT for the G plane, and VK_IMAGE_ASPECT_PLANE_1_BIT for the BR plane. This format only supports images with a width and height that is a multiple of two.
- VK_FORMAT_G12X4_B12X4_R12X4_3PLANE_422_UNORM_3PACK16 specifies an unsigned normalized *multi-planar format* that has a 12-bit G component in the top 12 bits of each 16-bit word of plane 0, a 12-bit B component in the top 12 bits of each 16-bit word of plane 1, and a 12-bit R component in the top 12 bits of each 16-bit word of plane 2, with the bottom 4 bits of each word unused. The horizontal dimension of the R and B plane is halved relative to the image dimensions, and each R and B value is shared with the G components for which $\lfloor i_G \times 0.5 \rfloor = i_B = i_R$. The location of each plane when this image is in linear layout can be determined via vkGetImageSubresourceLayout, using VK_IMAGE_ASPECT_PLANE_0_BIT for the G plane, VK_IMAGE_ASPECT_PLANE_1_BIT for the B plane, and VK_IMAGE_ASPECT_PLANE_2_BIT for the R plane. This format only supports images with a width that is a multiple of two.
- VK_FORMAT_G12X4_B12X4R12X4_2PLANE_422_UNORM_3PACK16 specifies an unsigned normalized *multi-planar format* that has a 12-bit G component in the top 12 bits of each 16-bit word of plane 0, and a two-component, 32-bit BR plane 1 consisting of a 12-bit B component in the top 12 bits of the word in bytes 0..1, and a 12-bit R component in the top 12 bits of the word in bytes 2..3, with the bottom 4 bits of each word unused. The horizontal dimension of the BR plane is halved relative to the image dimensions, and each R and B value is shared with the G components for which $\lfloor i_G \times 0.5 \rfloor = i_B = i_R$. The location of each plane when this image is in linear layout can be determined via vkGetImageSubresourceLayout, using VK_IMAGE_ASPECT_PLANE_0_BIT for the G plane, and VK_IMAGE_ASPECT_PLANE_1_BIT for the BR plane. This format only supports images with a width that is a multiple of two.
- VK_FORMAT_G12X4_B12X4_R12X4_3PLANE_444_UNORM_3PACK16 specifies an unsigned normalized *multi-planar format* that has a 12-bit G component in the top 12 bits of each 16-bit word of plane 0, a 12-bit B component in the top 12 bits of each 16-bit word of plane 1, and a 12-bit R component in the top 12 bits of each 16-bit word of plane 2, with the bottom 4 bits of each word unused. Each

plane has the same dimensions and each R, G and B component contributes to a single texel. The location of each plane when this image is in linear layout can be determined via vkGetImageSubresourceLayout, using VK_IMAGE_ASPECT_PLANE_0_BIT for the G plane, VK_IMAGE_ASPECT_PLANE_1_BIT for the B plane, and VK_IMAGE_ASPECT_PLANE_2_BIT for the R plane.

- VK_FORMAT_G16B16G16R16_422_UNORM specifies a four-component, 64-bit format containing a pair of G components, an R component, and a B component, collectively encoding a 2×1 rectangle of unsigned normalized RGB texel data. One G value is present at each $i$ coordinate, with the B and R values shared across both G values and thus recorded at half the horizontal resolution of the image. This format has a 16-bit G component for the even $i$ coordinate in the word in bytes 0..1, a 16-bit B component in the word in bytes 2..3, a 16-bit G component for the odd $i$ coordinate in the word in bytes 4..5, and a 16-bit R component in the word in bytes 6..7. This format only supports images with a width that is a multiple of two. For the purposes of the constraints on copy extents, this format is treated as a compressed format with a 2×1 compressed texel block.
- VK_FORMAT_B16G16R16G16_422_UNORM specifies a four-component, 64-bit format containing a pair of G components, an R component, and a B component, collectively encoding a 2×1 rectangle of unsigned normalized RGB texel data. One G value is present at each $i$ coordinate, with the B and R values shared across both G values and thus recorded at half the horizontal resolution of the image. This format has a 16-bit B component in the word in bytes 0..1, a 16-bit G component for the even $i$ coordinate in the word in bytes 2..3, a 16-bit R component in the word in bytes 4..5, and a 16-bit G component for the odd $i$ coordinate in the word in bytes 6..7. This format only supports images with a width that is a multiple of two. For the purposes of the constraints on copy extents, this format is treated as a compressed format with a 2×1 compressed texel block.
- VK_FORMAT_G16_B16_R16_3PLANE_420_UNORM specifies an unsigned normalized *multi-planar format* that has a 16-bit G component in each 16-bit word of plane 0, a 16-bit B component in each 16-bit word of plane 1, and a 16-bit R component in each 16-bit word of plane 2. The horizontal and vertical dimensions of the R and B planes are halved relative to the image dimensions, and each R and B component is shared with the G components for which $\lfloor i_G \times 0.5 \rfloor = i_B = i_R$ and $\lfloor j_G \times 0.5 \rfloor = j_B = j_R$. The location of each plane when this image is in linear layout can be determined via vkGetImageSubresourceLayout, using VK_IMAGE_ASPECT_PLANE_0_BIT for the G plane, VK_IMAGE_ASPECT_PLANE_1_BIT for the B plane, and VK_IMAGE_ASPECT_PLANE_2_BIT for the R plane. This format only supports images with a width and height that is a multiple of two.
- VK_FORMAT_G16_B16R16_2PLANE_420_UNORM specifies an unsigned normalized *multi-planar format* that has a 16-bit G component in each 16-bit word of plane 0, and a two-component, 32-bit BR plane 1 consisting of a 16-bit B component in the word in bytes 0..1, and a 16-bit R component in the word in bytes 2..3. The horizontal and vertical dimensions of the BR plane are halved relative to the image dimensions, and each R and B value is shared with the G components for which $\lfloor i_G \times 0.5 \rfloor = i_B = i_R$ and $\lfloor j_G \times 0.5 \rfloor = j_B = j_R$. The location of each plane when this image is in linear layout can be determined via vkGetImageSubresourceLayout, using VK_IMAGE_ASPECT_PLANE_0_BIT for the G plane, and VK_IMAGE_ASPECT_PLANE_1_BIT for the BR plane. This format only supports images with a width and height that is a multiple of two.
- VK_FORMAT_G16_B16_R16_3PLANE_422_UNORM specifies an unsigned normalized *multi-planar format* that has a 16-bit G component in each 16-bit word of plane 0, a 16-bit B component in each 16-bit word of plane 1, and a 16-bit R component in each 16-bit word of plane 2. The horizontal dimension of the R and B plane is halved relative to the image dimensions, and each R and B value is shared with the G components for which $\lfloor i_G \times 0.5 \rfloor = i_B = i_R$. The location of each plane when this image is in linear layout can be determined via vkGetImageSubresourceLayout, using

VK_IMAGE_ASPECT_PLANE_0_BIT for the G plane, VK_IMAGE_ASPECT_PLANE_1_BIT for the B plane, and VK_IMAGE_ASPECT_PLANE_2_BIT for the R plane. This format only supports images with a width that is a multiple of two.

- VK_FORMAT_G16_B16R16_2PLANE_422_UNORM specifies an unsigned normalized *multi-planar format* that has a 16-bit G component in each 16-bit word of plane 0, and a two-component, 32-bit BR plane 1 consisting of a 16-bit B component in the word in bytes 0..1, and a 16-bit R component in the word in bytes 2..3. The horizontal dimension of the BR plane is halved relative to the image dimensions, and each R and B value is shared with the G components for which $\lfloor i_G \times 0.5 \rfloor = i_B = i_R$. The location of each plane when this image is in linear layout can be determined via vkGetImageSubresourceLayout, using VK_IMAGE_ASPECT_PLANE_0_BIT for the G plane, and VK_IMAGE_ASPECT_PLANE_1_BIT for the BR plane. This format only supports images with a width that is a multiple of two.
- VK_FORMAT_G16_B16_R16_3PLANE_444_UNORM specifies an unsigned normalized *multi-planar format* that has a 16-bit G component in each 16-bit word of plane 0, a 16-bit B component in each 16-bit word of plane 1, and a 16-bit R component in each 16-bit word of plane 2. Each plane has the same dimensions and each R, G and B component contributes to a single texel. The location of each plane when this image is in linear layout can be determined via vkGetImageSubresourceLayout, using VK_IMAGE_ASPECT_PLANE_0_BIT for the G plane, VK_IMAGE_ASPECT_PLANE_1_BIT for the B plane, and VK_IMAGE_ASPECT_PLANE_2_BIT for the R plane.
- VK_FORMAT_G8_B8R8_2PLANE_444_UNORM specifies an unsigned normalized *multi-planar format* that has an 8-bit G component in plane 0, and a two-component, 16-bit BR plane 1 consisting of an 8-bit B component in byte 0 and an 8-bit R component in byte 1. Both planes have the same dimensions and each R, G and B component contributes to a single texel. The location of each plane when this image is in linear layout can be determined via vkGetImageSubresourceLayout, using VK_IMAGE_ASPECT_PLANE_0_BIT for the G plane, and VK_IMAGE_ASPECT_PLANE_1_BIT for the BR plane.
- VK_FORMAT_G10X6_B10X6R10X6_2PLANE_444_UNORM_3PACK16 specifies an unsigned normalized *multi-planar format* that has a 10-bit G component in the top 10 bits of each 16-bit word of plane 0, and a two-component, 32-bit BR plane 1 consisting of a 10-bit B component in the top 10 bits of the word in bytes 0..1, and a 10-bit R component in the top 10 bits of the word in bytes 2..3, the bottom 6 bits of each word unused. Both planes have the same dimensions and each R, G and B component contributes to a single texel. The location of each plane when this image is in linear layout can be determined via vkGetImageSubresourceLayout, using VK_IMAGE_ASPECT_PLANE_0_BIT for the G plane, and VK_IMAGE_ASPECT_PLANE_1_BIT for the BR plane.
- VK_FORMAT_G12X4_B12X4R12X4_2PLANE_444_UNORM_3PACK16 specifies an unsigned normalized *multi-planar format* that has a 12-bit G component in the top 12 bits of each 16-bit word of plane 0, and a two-component, 32-bit BR plane 1 consisting of a 12-bit B component in the top 12 bits of the word in bytes 0..1, and a 12-bit R component in the top 12 bits of the word in bytes 2..3, the bottom 4 bits of each word unused. Both planes have the same dimensions and each R, G and B component contributes to a single texel. The location of each plane when this image is in linear layout can be determined via vkGetImageSubresourceLayout, using VK_IMAGE_ASPECT_PLANE_0_BIT for the G plane, and VK_IMAGE_ASPECT_PLANE_1_BIT for the BR plane.
- VK_FORMAT_G16_B16R16_2PLANE_444_UNORM specifies an unsigned normalized *multi-planar format* that has a 16-bit G component in each 16-bit word of plane 0, and a two-component, 32-bit BR plane 1 consisting of a 16-bit B component in the word in bytes 0..1, and a 16-bit R component in

the word in bytes 2..3. Both planes have the same dimensions and each R, G and B component contributes to a single texel. The location of each plane when this image is in linear layout can be determined via vkGetImageSubresourceLayout, using VK_IMAGE_ASPECT_PLANE_0_BIT for the G plane, and VK_IMAGE_ASPECT_PLANE_1_BIT for the BR plane.

- VK_FORMAT_PVRTC1_2BPP_UNORM_BLOCK_IMG specifies a four-component, PVRTC compressed format where each 64-bit compressed texel block encodes an 8×4 rectangle of unsigned normalized RGBA texel data.
- VK_FORMAT_PVRTC1_4BPP_UNORM_BLOCK_IMG specifies a four-component, PVRTC compressed format where each 64-bit compressed texel block encodes a 4×4 rectangle of unsigned normalized RGBA texel data.
- VK_FORMAT_PVRTC2_2BPP_UNORM_BLOCK_IMG specifies a four-component, PVRTC compressed format where each 64-bit compressed texel block encodes an 8×4 rectangle of unsigned normalized RGBA texel data.
- VK_FORMAT_PVRTC2_4BPP_UNORM_BLOCK_IMG specifies a four-component, PVRTC compressed format where each 64-bit compressed texel block encodes a 4×4 rectangle of unsigned normalized RGBA texel data.
- VK_FORMAT_PVRTC1_2BPP_SRGB_BLOCK_IMG specifies a four-component, PVRTC compressed format where each 64-bit compressed texel block encodes an 8×4 rectangle of unsigned normalized RGBA texel data with sRGB nonlinear encoding applied to the RGB components.
- VK_FORMAT_PVRTC1_4BPP_SRGB_BLOCK_IMG specifies a four-component, PVRTC compressed format where each 64-bit compressed texel block encodes a 4×4 rectangle of unsigned normalized RGBA texel data with sRGB nonlinear encoding applied to the RGB components.
- VK_FORMAT_PVRTC2_2BPP_SRGB_BLOCK_IMG specifies a four-component, PVRTC compressed format where each 64-bit compressed texel block encodes an 8×4 rectangle of unsigned normalized RGBA texel data with sRGB nonlinear encoding applied to the RGB components.
- VK_FORMAT_PVRTC2_4BPP_SRGB_BLOCK_IMG specifies a four-component, PVRTC compressed format where each 64-bit compressed texel block encodes a 4×4 rectangle of unsigned normalized RGBA texel data with sRGB nonlinear encoding applied to the RGB components.
- VK_FORMAT_R16G16_SFIXED5_NV specifies a two-component, 16-bit signed fixed-point format with linear encoding. The components are signed two's-complement integers where the most significant bit specifies the sign bit, the next 10 bits specify the integer value, and the last 5 bits represent the fractional value. The signed 16-bit values **can** be converted to floats in the range [-1024,1023.96875] by dividing the value by 32 ($2^5$).

### 48.1.1. Compatible Formats of Planes of Multi-Planar Formats

Individual planes of multi-planar formats are size-compatible with single-plane color formats if they occupy the same number of bits per texel block, and are compatible with those formats if they have the same block extent.

In the following table, individual planes of a *multi-planar* format are compatible with the format listed against the relevant plane index for that multi-planar format, and any format compatible with the listed single-plane format according to Format Compatibility Classes. These planes are also size-compatible with any format that is size-compatible with the listed single-plane format.

*Table 66. Plane Format Compatibility Table*

| Plane | Compatible format for plane | Width relative to the width *w* of the plane with the largest dimensions | Height relative to the height *h* of the plane with the largest dimensions |
|---|---|---|---|
| VK_FORMAT_G8_B8_R8_3PLANE_420_UNORM | | | |
| 0 | VK_FORMAT_R8_UNORM | w | h |
| 1 | VK_FORMAT_R8_UNORM | w/2 | h/2 |
| 2 | VK_FORMAT_R8_UNORM | w/2 | h/2 |
| VK_FORMAT_G8_B8R8_2PLANE_420_UNORM | | | |
| 0 | VK_FORMAT_R8_UNORM | w | h |
| 1 | VK_FORMAT_R8G8_UNORM | w/2 | h/2 |
| VK_FORMAT_G8_B8_R8_3PLANE_422_UNORM | | | |
| 0 | VK_FORMAT_R8_UNORM | w | h |
| 1 | VK_FORMAT_R8_UNORM | w/2 | h |
| 2 | VK_FORMAT_R8_UNORM | w/2 | h |
| VK_FORMAT_G8_B8R8_2PLANE_422_UNORM | | | |
| 0 | VK_FORMAT_R8_UNORM | w | h |
| 1 | VK_FORMAT_R8G8_UNORM | w/2 | h |
| VK_FORMAT_G8_B8_R8_3PLANE_444_UNORM | | | |
| 0 | VK_FORMAT_R8_UNORM | w | h |
| 1 | VK_FORMAT_R8_UNORM | w | h |
| 2 | VK_FORMAT_R8_UNORM | w | h |
| VK_FORMAT_G10X6_B10X6_R10X6_3PLANE_420_UNORM_3PACK16 | | | |
| 0 | VK_FORMAT_R10X6_UNORM_PACK16 | w | h |
| 1 | VK_FORMAT_R10X6_UNORM_PACK16 | w/2 | h/2 |
| 2 | VK_FORMAT_R10X6_UNORM_PACK16 | w/2 | h/2 |
| VK_FORMAT_G10X6_B10X6R10X6_2PLANE_420_UNORM_3PACK16 | | | |
| 0 | VK_FORMAT_R10X6_UNORM_PACK16 | w | h |
| 1 | VK_FORMAT_R10X6G10X6_UNORM_2PACK16 | w/2 | h/2 |
| VK_FORMAT_G10X6_B10X6_R10X6_3PLANE_422_UNORM_3PACK16 | | | |
| 0 | VK_FORMAT_R10X6_UNORM_PACK16 | w | h |
| 1 | VK_FORMAT_R10X6_UNORM_PACK16 | w/2 | h |
| 2 | VK_FORMAT_R10X6_UNORM_PACK16 | w/2 | h |
| VK_FORMAT_G10X6_B10X6R10X6_2PLANE_422_UNORM_3PACK16 | | | |
| 0 | VK_FORMAT_R10X6_UNORM_PACK16 | w | h |

| Plane | Compatible format for plane | Width relative to the width $w$ of the plane with the largest dimensions | Height relative to the height $h$ of the plane with the largest dimensions |
|---|---|---|---|
| 1 | VK_FORMAT_R10X6G10X6_UNORM_2PACK16 | w/2 | h |
| VK_FORMAT_G10X6_B10X6_R10X6_3PLANE_444_UNORM_3PACK16 | | | |
| 0 | VK_FORMAT_R10X6_UNORM_PACK16 | w | h |
| 1 | VK_FORMAT_R10X6_UNORM_PACK16 | w | h |
| 2 | VK_FORMAT_R10X6_UNORM_PACK16 | w | h |
| VK_FORMAT_G12X4_B12X4_R12X4_3PLANE_420_UNORM_3PACK16 | | | |
| 0 | VK_FORMAT_R12X4_UNORM_PACK16 | w | h |
| 1 | VK_FORMAT_R12X4_UNORM_PACK16 | w/2 | h/2 |
| 2 | VK_FORMAT_R12X4_UNORM_PACK16 | w/2 | h/2 |
| VK_FORMAT_G12X4_B12X4R12X4_2PLANE_420_UNORM_3PACK16 | | | |
| 0 | VK_FORMAT_R12X4_UNORM_PACK16 | w | h |
| 1 | VK_FORMAT_R12X4G12X4_UNORM_2PACK16 | w/2 | h/2 |
| VK_FORMAT_G12X4_B12X4_R12X4_3PLANE_422_UNORM_3PACK16 | | | |
| 0 | VK_FORMAT_R12X4_UNORM_PACK16 | w | h |
| 1 | VK_FORMAT_R12X4_UNORM_PACK16 | w/2 | h |
| 2 | VK_FORMAT_R12X4_UNORM_PACK16 | w/2 | h |
| VK_FORMAT_G12X4_B12X4R12X4_2PLANE_422_UNORM_3PACK16 | | | |
| 0 | VK_FORMAT_R12X4_UNORM_PACK16 | w | h |
| 1 | VK_FORMAT_R12X4G12X4_UNORM_2PACK16 | w/2 | h |
| VK_FORMAT_G12X4_B12X4_R12X4_3PLANE_444_UNORM_3PACK16 | | | |
| 0 | VK_FORMAT_R12X4_UNORM_PACK16 | w | h |
| 1 | VK_FORMAT_R12X4_UNORM_PACK16 | w | h |
| 2 | VK_FORMAT_R12X4_UNORM_PACK16 | w | h |
| VK_FORMAT_G16_B16_R16_3PLANE_420_UNORM | | | |
| 0 | VK_FORMAT_R16_UNORM | w | h |
| 1 | VK_FORMAT_R16_UNORM | w/2 | h/2 |
| 2 | VK_FORMAT_R16_UNORM | w/2 | h/2 |
| VK_FORMAT_G16_B16R16_2PLANE_420_UNORM | | | |
| 0 | VK_FORMAT_R16_UNORM | w | h |
| 1 | VK_FORMAT_R16G16_UNORM | w/2 | h/2 |
| VK_FORMAT_G16_B16_R16_3PLANE_422_UNORM | | | |
| 0 | VK_FORMAT_R16_UNORM | w | h |

| Plane | Compatible format for plane | Width relative to the width $w$ of the plane with the largest dimensions | Height relative to the height $h$ of the plane with the largest dimensions |
|---|---|---|---|
| 1 | VK_FORMAT_R16_UNORM | w/2 | h |
| 2 | VK_FORMAT_R16_UNORM | w/2 | h |
| VK_FORMAT_G16_B16R16_2PLANE_422_UNORM | | | |
| 0 | VK_FORMAT_R16_UNORM | w | h |
| 1 | VK_FORMAT_R16G16_UNORM | w/2 | h |
| VK_FORMAT_G16_B16_R16_3PLANE_444_UNORM | | | |
| 0 | VK_FORMAT_R16_UNORM | w | h |
| 1 | VK_FORMAT_R16_UNORM | w | h |
| 2 | VK_FORMAT_R16_UNORM | w | h |
| VK_FORMAT_G8_B8R8_2PLANE_444_UNORM | | | |
| 0 | VK_FORMAT_R8_UNORM | w | h |
| 1 | VK_FORMAT_R8G8_UNORM | w | h |
| VK_FORMAT_G10X6_B10X6R10X6_2PLANE_444_UNORM_3PACK16 | | | |
| 0 | VK_FORMAT_R10X6_UNORM_PACK16 | w | h |
| 1 | VK_FORMAT_R10X6G10X6_UNORM_2PACK16 | w | h |
| VK_FORMAT_G12X4_B12X4R12X4_2PLANE_444_UNORM_3PACK16 | | | |
| 0 | VK_FORMAT_R12X4_UNORM_PACK16 | w | h |
| 1 | VK_FORMAT_R12X4G12X4_UNORM_2PACK16 | w | h |
| VK_FORMAT_G16_B16R16_2PLANE_444_UNORM | | | |
| 0 | VK_FORMAT_R16_UNORM | w | h |
| 1 | VK_FORMAT_R16G16_UNORM | w | h |

### 48.1.2. Multi-planar Format Image Aspect

When using VkImageAspectFlagBits to select a plane of a multi-planar format, the following are the valid options:

- Two planes
  - VK_IMAGE_ASPECT_PLANE_0_BIT
  - VK_IMAGE_ASPECT_PLANE_1_BIT
- Three planes
  - VK_IMAGE_ASPECT_PLANE_0_BIT
  - VK_IMAGE_ASPECT_PLANE_1_BIT
  - VK_IMAGE_ASPECT_PLANE_2_BIT

### 48.1.3. Packed Formats

For the purposes of address alignment when accessing buffer memory containing vertex attribute or texel data, the following formats are considered *packed* - components of the texels or attributes are stored in bitfields packed into one or more 8-, 16-, or 32-bit fundamental data type.

- Packed into 8-bit data types:
  - VK_FORMAT_R4G4_UNORM_PACK8
- Packed into 16-bit data types:
  - VK_FORMAT_R4G4B4A4_UNORM_PACK16
  - VK_FORMAT_B4G4R4A4_UNORM_PACK16
  - VK_FORMAT_R5G6B5_UNORM_PACK16
  - VK_FORMAT_B5G6R5_UNORM_PACK16
  - VK_FORMAT_R5G5B5A1_UNORM_PACK16
  - VK_FORMAT_B5G5R5A1_UNORM_PACK16
  - VK_FORMAT_A1R5G5B5_UNORM_PACK16
  - VK_FORMAT_A1B5G5R5_UNORM_PACK16_KHR
  - VK_FORMAT_R10X6_UNORM_PACK16
  - VK_FORMAT_R10X6G10X6_UNORM_2PACK16
  - VK_FORMAT_R10X6G10X6B10X6A10X6_UNORM_4PACK16
  - VK_FORMAT_G10X6B10X6G10X6R10X6_422_UNORM_4PACK16
  - VK_FORMAT_B10X6G10X6R10X6G10X6_422_UNORM_4PACK16
  - VK_FORMAT_G10X6_B10X6_R10X6_3PLANE_420_UNORM_3PACK16
  - VK_FORMAT_G10X6_B10X6R10X6_2PLANE_420_UNORM_3PACK16
  - VK_FORMAT_G10X6_B10X6_R10X6_3PLANE_422_UNORM_3PACK16
  - VK_FORMAT_G10X6_B10X6R10X6_2PLANE_422_UNORM_3PACK16
  - VK_FORMAT_G10X6_B10X6_R10X6_3PLANE_444_UNORM_3PACK16
  - VK_FORMAT_R12X4_UNORM_PACK16
  - VK_FORMAT_R12X4G12X4_UNORM_2PACK16
  - VK_FORMAT_R12X4G12X4B12X4A12X4_UNORM_4PACK16
  - VK_FORMAT_G12X4B12X4G12X4R12X4_422_UNORM_4PACK16
  - VK_FORMAT_B12X4G12X4R12X4G12X4_422_UNORM_4PACK16
  - VK_FORMAT_G12X4_B12X4_R12X4_3PLANE_420_UNORM_3PACK16
  - VK_FORMAT_G12X4_B12X4R12X4_2PLANE_420_UNORM_3PACK16
  - VK_FORMAT_G12X4_B12X4_R12X4_3PLANE_422_UNORM_3PACK16
  - VK_FORMAT_G12X4_B12X4R12X4_2PLANE_422_UNORM_3PACK16
  - VK_FORMAT_G12X4_B12X4_R12X4_3PLANE_444_UNORM_3PACK16

    - VK_FORMAT_G10X6_B10X6R10X6_2PLANE_444_UNORM_3PACK16
    - VK_FORMAT_G12X4_B12X4R12X4_2PLANE_444_UNORM_3PACK16
    - VK_FORMAT_A4R4G4B4_UNORM_PACK16
    - VK_FORMAT_A4B4G4R4_UNORM_PACK16
- Packed into 32-bit data types:
    - VK_FORMAT_A8B8G8R8_UNORM_PACK32
    - VK_FORMAT_A8B8G8R8_SNORM_PACK32
    - VK_FORMAT_A8B8G8R8_USCALED_PACK32
    - VK_FORMAT_A8B8G8R8_SSCALED_PACK32
    - VK_FORMAT_A8B8G8R8_UINT_PACK32
    - VK_FORMAT_A8B8G8R8_SINT_PACK32
    - VK_FORMAT_A8B8G8R8_SRGB_PACK32
    - VK_FORMAT_A2R10G10B10_UNORM_PACK32
    - VK_FORMAT_A2R10G10B10_SNORM_PACK32
    - VK_FORMAT_A2R10G10B10_USCALED_PACK32
    - VK_FORMAT_A2R10G10B10_SSCALED_PACK32
    - VK_FORMAT_A2R10G10B10_UINT_PACK32
    - VK_FORMAT_A2R10G10B10_SINT_PACK32
    - VK_FORMAT_A2B10G10R10_UNORM_PACK32
    - VK_FORMAT_A2B10G10R10_SNORM_PACK32
    - VK_FORMAT_A2B10G10R10_USCALED_PACK32
    - VK_FORMAT_A2B10G10R10_SSCALED_PACK32
    - VK_FORMAT_A2B10G10R10_UINT_PACK32
    - VK_FORMAT_A2B10G10R10_SINT_PACK32
    - VK_FORMAT_B10G11R11_UFLOAT_PACK32
    - VK_FORMAT_E5B9G9R9_UFLOAT_PACK32
    - VK_FORMAT_X8_D24_UNORM_PACK32

### 48.1.4. Identification of Formats

A “format” is represented by a single enum value. The name of a format is usually built up by using the following pattern:

```
VK_FORMAT_{component-format|compression-scheme}_{numeric-format}
```

The component-format indicates either the size of the R, G, B, and A components (if they are present) in the case of a color format, or the size of the depth (D) and stencil (S) components (if they

are present) in the case of a depth/stencil format (see below). An X indicates a component that is unused, but **may** be present for padding.

*Table 67. Interpretation of Numeric Format*

| Numeric format | Type-Declaration instructions | Numeric type | Description |
|---|---|---|---|
| `UNORM` | OpTypeFloat | floating-point | The components are unsigned normalized values in the range [0,1] |
| `SNORM` | OpTypeFloat | floating-point | The components are signed normalized values in the range [-1,1] |
| `USCALED` | OpTypeFloat | floating-point | The components are unsigned integer values that get converted to floating-point in the range $[0,2^n-1]$ |
| `SSCALED` | OpTypeFloat | floating-point | The components are signed integer values that get converted to floating-point in the range $[-2^{n-1},2^{n-1}-1]$ |
| `UINT` | OpTypeInt | unsigned integer | The components are unsigned integer values in the range $[0,2^n-1]$ |
| `SINT` | OpTypeInt | signed integer | The components are signed integer values in the range $[-2^{n-1},2^{n-1}-1]$ |
| `UFLOAT` | OpTypeFloat | floating-point | The components are unsigned floating-point numbers (used by packed, shared exponent, and some compressed formats) |
| `SFLOAT` | OpTypeFloat | floating-point | The components are signed floating-point numbers |
| `SRGB` | OpTypeFloat | floating-point | The R, G, and B components are unsigned normalized values that represent values using sRGB nonlinear encoding, while the A component (if one exists) is a regular unsigned normalized value |
| `SFIXED5` | OpTypeInt | scaled signed integer | The components are signed fractional integer values that get converted to floating-point in the range [-1024,1023.96875] |
| n is the number of bits in the component. | | | |

The suffix `_PACKnn` indicates that the format is packed into an underlying type with `nn` bits. The suffix `_mPACKnn` is a short-hand that indicates that the format has `m` groups of components (which may or may not be stored in separate *planes*) that are each packed into an underlying type with `nn` bits.

The suffix `_BLOCK` indicates that the format is a block-compressed format, with the representation of multiple pixels encoded interdependently within a region.

*Table 68. Interpretation of Compression Scheme*

| Compression scheme | Description |
|---|---|
| `BC` | Block Compression. See Block-Compressed Image Formats. |

| Compression scheme | Description |
|---|---|
| ETC2 | Ericsson Texture Compression. See ETC Compressed Image Formats. |
| EAC | ETC2 Alpha Compression. See ETC Compressed Image Formats. |
| ASTC | Adaptive Scalable Texture Compression (LDR Profile). See ASTC Compressed Image Formats. |

For *multi-planar* images, the components in separate *planes* are separated by underscores, and the number of planes is indicated by the addition of a `_2PLANE` or `_3PLANE` suffix. Similarly, the separate aspects of depth-stencil formats are separated by underscores, although these are not considered separate planes. Formats are suffixed by `_422` to indicate that planes other than the first are reduced in size by a factor of two horizontally or that the R and B values appear at half the horizontal frequency of the G values, `_420` to indicate that planes other than the first are reduced in size by a factor of two both horizontally and vertically, and `_444` for consistency to indicate that all three planes of a three-planar image are the same size.

**NOTE** No common format has a single plane containing both R and B components but does not store these components at reduced horizontal resolution.

### 48.1.5. Representation and Texel Block Size

Color formats **must** be represented in memory in exactly the form indicated by the format's name. This means that promoting one format to another with more bits per component and/or additional components **must** not occur for color formats. Depth/stencil formats have more relaxed requirements as discussed below.

Each format has a *texel block size*, the number of bytes used to store one *texel block* (a single addressable element of an uncompressed image, or a single compressed block of a compressed image). The texel block size for each format is shown in the Compatible formats table.

The representation of non-packed formats is that the first component specified in the name of the format is in the lowest memory addresses and the last component specified is in the highest memory addresses. See Byte mappings for non-packed/compressed color formats. The in-memory ordering of bytes within a component is determined by the host endianness.

*Table 69. Byte mappings for non-packed/compressed color formats*

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | ← Byte |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | | | | | | | | | | | | | | | | VK_FORMAT_R8_* |
| R | G | | | | | | | | | | | | | | | VK_FORMAT_R8G8_* |
| R | G | B | | | | | | | | | | | | | | VK_FORMAT_R8G8B8_* |
| B | G | R | | | | | | | | | | | | | | VK_FORMAT_B8G8R8_* |
| R | G | B | A | | | | | | | | | | | | | VK_FORMAT_R8G8B8A8_* |
| B | G | R | A | | | | | | | | | | | | | VK_FORMAT_B8G8R8A8_* |
| A | | | | | | | | | | | | | | | | VK_FORMAT_A8_UNORM_KHR |

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | ← Byte |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $G_0$ | B | $G_1$ | R | | | | | | | | | | | | | VK_FORMAT_G8B8G8R8_422_UNORM |
| B | $G_0$ | R | $G_1$ | | | | | | | | | | | | | VK_FORMAT_B8G8R8G8_422_UNORM |
| R | | | | | | | | | | | | | | | | VK_FORMAT_R16_* |
| R | | G | | | | | | | | | | | | | | VK_FORMAT_R16G16_* |
| R | | G | | B | | | | | | | | | | | | VK_FORMAT_R16G16B16_* |
| R | | G | | B | | A | | | | | | | | | | VK_FORMAT_R16G16B16A16_* |
| $G_0$ | | B | | $G_1$ | | R | | | | | | | | | | VK_FORMAT_G10X6B10X6G10X6R10X6_4PACK16_422_UNORM<br>VK_FORMAT_G12X4B12X4G12X4R12X4_4PACK16_422_UNORM<br>VK_FORMAT_G16B16G16R16_UNORM |
| B | | $G_0$ | | R | | $G_1$ | | | | | | | | | | VK_FORMAT_B10X6G10X6R10X6G10X6_4PACK16_422_UNORM<br>VK_FORMAT_B12X4G12X4R12X4G12X4_4PACK16_422_UNORM<br>VK_FORMAT_B16G16R16G16_422_UNORM |
| R | | | | | | | | | | | | | | | | VK_FORMAT_R32_* |
| R | | | | G | | | | | | | | | | | | VK_FORMAT_R32G32_* |
| R | | | | G | | | | B | | | | | | | | VK_FORMAT_R32G32B32_* |
| R | | | | G | | | | B | | | | A | | | | VK_FORMAT_R32G32B32A32_* |
| R | | | | | | | | | | | | | | | | VK_FORMAT_R64_* |
| R | | | | | | | | G | | | | | | | | VK_FORMAT_R64G64_* |
| VK_FORMAT_R64G64B64_* as VK_FORMAT_R64G64_* but with B in bytes 16-23 | | | | | | | | | | | | | | | | |
| VK_FORMAT_R64G64B64A64_* as VK_FORMAT_R64G64B64_* but with A in bytes 24-31 | | | | | | | | | | | | | | | | |

Packed formats store multiple components within one underlying type. The bit representation is that the first component specified in the name of the format is in the most-significant bits and the last component specified is in the least-significant bits of the underlying type. The in-memory ordering of bytes comprising the underlying type is determined by the host endianness.

*Table 70. Bit mappings for packed 8-bit formats*

| Bit | | | | | | | |
|---|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| VK_FORMAT_R4G4_UNORM_PACK8 | | | | | | | |
| R | | | | G | | | |
| 3 | 2 | 1 | 0 | 3 | 2 | 1 | 0 |

*Table 71. Bit mappings for packed 16-bit formats*

| Bit | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| VK_FORMAT_R4G4B4A4_UNORM_PACK16 | | | | | | | | | | | | | | | |
| R | | | | G | | | | B | | | | A | | | |
| 3 | 2 | 1 | 0 | 3 | 2 | 1 | 0 | 3 | 2 | 1 | 0 | 3 | 2 | 1 | 0 |

| Bit | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| VK_FORMAT_B4G4R4A4_UNORM_PACK16 | | | | | | | | | | | | | | | |
| B | | | | G | | | | R | | | | A | | | |
| 3 | 2 | 1 | 0 | 3 | 2 | 1 | 0 | 3 | 2 | 1 | 0 | 3 | 2 | 1 | 0 |
| VK_FORMAT_A4R4G4B4_UNORM_PACK16 | | | | | | | | | | | | | | | |
| A | | | | R | | | | G | | | | B | | | |
| 3 | 2 | 1 | 0 | 3 | 2 | 1 | 0 | 3 | 2 | 1 | 0 | 3 | 2 | 1 | 0 |
| VK_FORMAT_A4B4G4R4_UNORM_PACK16 | | | | | | | | | | | | | | | |
| A | | | | B | | | | G | | | | R | | | |
| 3 | 2 | 1 | 0 | 3 | 2 | 1 | 0 | 3 | 2 | 1 | 0 | 3 | 2 | 1 | 0 |
| VK_FORMAT_R5G6B5_UNORM_PACK16 | | | | | | | | | | | | | | | |
| R | | | | | G | | | | | | B | | | | |
| 4 | 3 | 2 | 1 | 0 | 5 | 4 | 3 | 2 | 1 | 0 | 4 | 3 | 2 | 1 | 0 |
| VK_FORMAT_B5G6R5_UNORM_PACK16 | | | | | | | | | | | | | | | |
| B | | | | | G | | | | | | R | | | | |
| 4 | 3 | 2 | 1 | 0 | 5 | 4 | 3 | 2 | 1 | 0 | 4 | 3 | 2 | 1 | 0 |
| VK_FORMAT_R5G5B5A1_UNORM_PACK16 | | | | | | | | | | | | | | | |
| R | | | | | G | | | | | B | | | | | A |
| 4 | 3 | 2 | 1 | 0 | 4 | 3 | 2 | 1 | 0 | 4 | 3 | 2 | 1 | 0 | 0 |
| VK_FORMAT_B5G5R5A1_UNORM_PACK16 | | | | | | | | | | | | | | | |
| B | | | | | G | | | | | R | | | | | A |
| 4 | 3 | 2 | 1 | 0 | 4 | 3 | 2 | 1 | 0 | 4 | 3 | 2 | 1 | 0 | 0 |
| VK_FORMAT_A1R5G5B5_UNORM_PACK16 | | | | | | | | | | | | | | | |
| A | R | | | | | G | | | | | B | | | | |
| 0 | 4 | 3 | 2 | 1 | 0 | 4 | 3 | 2 | 1 | 0 | 4 | 3 | 2 | 1 | 0 |
| VK_FORMAT_A1B5G5R5_UNORM_PACK16_KHR | | | | | | | | | | | | | | | |
| A | B | | | | | G | | | | | R | | | | |
| 0 | 4 | 3 | 2 | 1 | 0 | 4 | 3 | 2 | 1 | 0 | 4 | 3 | 2 | 1 | 0 |
| VK_FORMAT_R10X6_UNORM_PACK16 | | | | | | | | | | | | | | | |
| R | | | | | | | | | | X | | | | | |
| 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 5 | 4 | 3 | 2 | 1 | 0 |
| VK_FORMAT_R12X4_UNORM_PACK16 | | | | | | | | | | | | | | | |
| R | | | | | | | | | | | | X | | | |
| 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 3 | 2 | 1 | 0 |

*Table 72. Bit mappings for packed 32-bit formats*

| Bit | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| VK_FORMAT_A8B8G8R8_*_PACK32 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| A | | | | | | | | B | | | | | | | | G | | | | | | | | R | | | | | | | |

| Bit | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| VK_FORMAT_A2R10G10B10_*_PACK32 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| A | | R | | | | | | | | | | G | | | | | | | | | | | | B | | | | | | | |
| 1 | 0 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| VK_FORMAT_A2B10G10R10_*_PACK32 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| A | | B | | | | | | | | | | G | | | | | | | | | | | | R | | | | | | | |
| 1 | 0 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| VK_FORMAT_B10G11R11_UFLOAT_PACK32 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| B | | | | | | | | | | G | | | | | | | | | | R | | | | | | | | | | | |
| 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| VK_FORMAT_E5B9G9R9_UFLOAT_PACK32 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| E | | | | | B | | | | | | | | | G | | | | | | | | | | R | | | | | | | |
| 4 | 3 | 2 | 1 | 0 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| VK_FORMAT_X8_D24_UNORM_PACK32 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| X | | | | | | | | D | | | | | | | | | | | | | | | | | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

### 48.1.6. Depth/Stencil Formats

Depth/stencil formats are considered opaque and need not be stored in the exact number of bits per texel or component ordering indicated by the format enum. However, implementations **must** not substitute a different depth or stencil precision than is described in the format (e.g. D16 **must** not be implemented as D24 or D32).

### 48.1.7. Format Compatibility Classes

Uncompressed color formats are *compatible* with each other if they occupy the same number of bits per texel block as long as neither or both are alpha formats (e.g., VK_FORMAT_A8_UNORM_KHR) . Compressed color formats are compatible with each other if the only difference between them is the numeric format of the uncompressed pixels. Each depth/stencil format is only compatible with itself. In the following table, all the formats in the same row are compatible. Each format has a defined *texel block extent* specifying how many texels each texel block represents in each dimension.

*Table 73. Compatible Formats*

| Class, Texel Block Size, Texel Block Extent, # Texels/Block | Formats |
|---|---|
| 8-bit<br>Block size 1 byte<br>1x1x1 block extent<br>1 texel/block | VK_FORMAT_R4G4_UNORM_PACK8,<br>VK_FORMAT_R8_UNORM,<br>VK_FORMAT_R8_SNORM,<br>VK_FORMAT_R8_USCALED,<br>VK_FORMAT_R8_SSCALED,<br>VK_FORMAT_R8_UINT,<br>VK_FORMAT_R8_SINT,<br>VK_FORMAT_R8_SRGB |
| 16-bit<br>Block size 2 byte<br>1x1x1 block extent<br>1 texel/block | VK_FORMAT_A1B5G5R5_UNORM_PACK16_KHR,<br>VK_FORMAT_R10X6_UNORM_PACK16,<br>VK_FORMAT_R12X4_UNORM_PACK16,<br>VK_FORMAT_A4R4G4B4_UNORM_PACK16,<br>VK_FORMAT_A4B4G4R4_UNORM_PACK16,<br>VK_FORMAT_R4G4B4A4_UNORM_PACK16,<br>VK_FORMAT_B4G4R4A4_UNORM_PACK16,<br>VK_FORMAT_R5G6B5_UNORM_PACK16,<br>VK_FORMAT_B5G6R5_UNORM_PACK16,<br>VK_FORMAT_R5G5B5A1_UNORM_PACK16,<br>VK_FORMAT_B5G5R5A1_UNORM_PACK16,<br>VK_FORMAT_A1R5G5B5_UNORM_PACK16,<br>VK_FORMAT_R8G8_UNORM,<br>VK_FORMAT_R8G8_SNORM,<br>VK_FORMAT_R8G8_USCALED,<br>VK_FORMAT_R8G8_SSCALED,<br>VK_FORMAT_R8G8_UINT,<br>VK_FORMAT_R8G8_SINT,<br>VK_FORMAT_R8G8_SRGB,<br>VK_FORMAT_R16_UNORM,<br>VK_FORMAT_R16_SNORM,<br>VK_FORMAT_R16_USCALED,<br>VK_FORMAT_R16_SSCALED,<br>VK_FORMAT_R16_UINT,<br>VK_FORMAT_R16_SINT,<br>VK_FORMAT_R16_SFLOAT |
| 8-bit alpha<br>Block size 1 byte<br>1x1x1 block extent<br>1 texel/block | VK_FORMAT_A8_UNORM_KHR |

| Class, Texel Block Size, Texel Block Extent, # Texels/Block | Formats |
| --- | --- |
| 24-bit<br>Block size 3 byte<br>1x1x1 block extent<br>1 texel/block | VK_FORMAT_R8G8B8_UNORM,<br>VK_FORMAT_R8G8B8_SNORM,<br>VK_FORMAT_R8G8B8_USCALED,<br>VK_FORMAT_R8G8B8_SSCALED,<br>VK_FORMAT_R8G8B8_UINT,<br>VK_FORMAT_R8G8B8_SINT,<br>VK_FORMAT_R8G8B8_SRGB,<br>VK_FORMAT_B8G8R8_UNORM,<br>VK_FORMAT_B8G8R8_SNORM,<br>VK_FORMAT_B8G8R8_USCALED,<br>VK_FORMAT_B8G8R8_SSCALED,<br>VK_FORMAT_B8G8R8_UINT,<br>VK_FORMAT_B8G8R8_SINT,<br>VK_FORMAT_B8G8R8_SRGB |

| Class, Texel Block Size, Texel Block Extent, # Texels/Block | Formats |
| --- | --- |
| 32-bit<br>Block size 4 byte<br>1x1x1 block extent<br>1 texel/block | VK_FORMAT_R10X6G10X6_UNORM_2PACK16,<br>VK_FORMAT_R12X4G12X4_UNORM_2PACK16,<br>VK_FORMAT_R16G16_SFIXED5_NV,<br>VK_FORMAT_R8G8B8A8_UNORM,<br>VK_FORMAT_R8G8B8A8_SNORM,<br>VK_FORMAT_R8G8B8A8_USCALED,<br>VK_FORMAT_R8G8B8A8_SSCALED,<br>VK_FORMAT_R8G8B8A8_UINT,<br>VK_FORMAT_R8G8B8A8_SINT,<br>VK_FORMAT_R8G8B8A8_SRGB,<br>VK_FORMAT_B8G8R8A8_UNORM,<br>VK_FORMAT_B8G8R8A8_SNORM,<br>VK_FORMAT_B8G8R8A8_USCALED,<br>VK_FORMAT_B8G8R8A8_SSCALED,<br>VK_FORMAT_B8G8R8A8_UINT,<br>VK_FORMAT_B8G8R8A8_SINT,<br>VK_FORMAT_B8G8R8A8_SRGB,<br>VK_FORMAT_A8B8G8R8_UNORM_PACK32,<br>VK_FORMAT_A8B8G8R8_SNORM_PACK32,<br>VK_FORMAT_A8B8G8R8_USCALED_PACK32,<br>VK_FORMAT_A8B8G8R8_SSCALED_PACK32,<br>VK_FORMAT_A8B8G8R8_UINT_PACK32,<br>VK_FORMAT_A8B8G8R8_SINT_PACK32,<br>VK_FORMAT_A8B8G8R8_SRGB_PACK32,<br>VK_FORMAT_A2R10G10B10_UNORM_PACK32,<br>VK_FORMAT_A2R10G10B10_SNORM_PACK32,<br>VK_FORMAT_A2R10G10B10_USCALED_PACK32,<br>VK_FORMAT_A2R10G10B10_SSCALED_PACK32,<br>VK_FORMAT_A2R10G10B10_UINT_PACK32,<br>VK_FORMAT_A2R10G10B10_SINT_PACK32,<br>VK_FORMAT_A2B10G10R10_UNORM_PACK32,<br>VK_FORMAT_A2B10G10R10_SNORM_PACK32,<br>VK_FORMAT_A2B10G10R10_USCALED_PACK32,<br>VK_FORMAT_A2B10G10R10_SSCALED_PACK32,<br>VK_FORMAT_A2B10G10R10_UINT_PACK32,<br>VK_FORMAT_A2B10G10R10_SINT_PACK32,<br>VK_FORMAT_R16G16_UNORM,<br>VK_FORMAT_R16G16_SNORM,<br>VK_FORMAT_R16G16_USCALED,<br>VK_FORMAT_R16G16_SSCALED,<br>VK_FORMAT_R16G16_UINT,<br>VK_FORMAT_R16G16_SINT,<br>VK_FORMAT_R16G16_SFLOAT,<br>VK_FORMAT_R32_UINT,<br>VK_FORMAT_R32_SINT,<br>VK_FORMAT_R32_SFLOAT,<br>VK_FORMAT_B10G11R11_UFLOAT_PACK32,<br>VK_FORMAT_E5B9G9R9_UFLOAT_PACK32, |

| Class, Texel Block Size, Texel Block Extent, # Texels/Block | Formats |
|---|---|
| 48-bit<br>Block size 6 byte<br>1x1x1 block extent<br>1 texel/block | VK_FORMAT_R16G16B16_UNORM,<br>VK_FORMAT_R16G16B16_SNORM,<br>VK_FORMAT_R16G16B16_USCALED,<br>VK_FORMAT_R16G16B16_SSCALED,<br>VK_FORMAT_R16G16B16_UINT,<br>VK_FORMAT_R16G16B16_SINT,<br>VK_FORMAT_R16G16B16_SFLOAT |
| 64-bit<br>Block size 8 byte<br>1x1x1 block extent<br>1 texel/block | VK_FORMAT_R16G16B16A16_UNORM,<br>VK_FORMAT_R16G16B16A16_SNORM,<br>VK_FORMAT_R16G16B16A16_USCALED,<br>VK_FORMAT_R16G16B16A16_SSCALED,<br>VK_FORMAT_R16G16B16A16_UINT,<br>VK_FORMAT_R16G16B16A16_SINT,<br>VK_FORMAT_R16G16B16A16_SFLOAT,<br>VK_FORMAT_R32G32_UINT,<br>VK_FORMAT_R32G32_SINT,<br>VK_FORMAT_R32G32_SFLOAT,<br>VK_FORMAT_R64_UINT,<br>VK_FORMAT_R64_SINT,<br>VK_FORMAT_R64_SFLOAT |
| 96-bit<br>Block size 12 byte<br>1x1x1 block extent<br>1 texel/block | VK_FORMAT_R32G32B32_UINT,<br>VK_FORMAT_R32G32B32_SINT,<br>VK_FORMAT_R32G32B32_SFLOAT |
| 128-bit<br>Block size 16 byte<br>1x1x1 block extent<br>1 texel/block | VK_FORMAT_R32G32B32A32_UINT,<br>VK_FORMAT_R32G32B32A32_SINT,<br>VK_FORMAT_R32G32B32A32_SFLOAT,<br>VK_FORMAT_R64G64_UINT,<br>VK_FORMAT_R64G64_SINT,<br>VK_FORMAT_R64G64_SFLOAT |
| 192-bit<br>Block size 24 byte<br>1x1x1 block extent<br>1 texel/block | VK_FORMAT_R64G64B64_UINT,<br>VK_FORMAT_R64G64B64_SINT,<br>VK_FORMAT_R64G64B64_SFLOAT |
| 256-bit<br>Block size 32 byte<br>1x1x1 block extent<br>1 texel/block | VK_FORMAT_R64G64B64A64_UINT,<br>VK_FORMAT_R64G64B64A64_SINT,<br>VK_FORMAT_R64G64B64A64_SFLOAT |

| Class, Texel Block Size, Texel Block Extent, # Texels/Block | Formats |
|---|---|
| D16<br>Block size 2 byte<br>1x1x1 block extent<br>1 texel/block | VK_FORMAT_D16_UNORM |
| D24<br>Block size 4 byte<br>1x1x1 block extent<br>1 texel/block | VK_FORMAT_X8_D24_UNORM_PACK32 |
| D32<br>Block size 4 byte<br>1x1x1 block extent<br>1 texel/block | VK_FORMAT_D32_SFLOAT |
| S8<br>Block size 1 byte<br>1x1x1 block extent<br>1 texel/block | VK_FORMAT_S8_UINT |
| D16S8<br>Block size 3 byte<br>1x1x1 block extent<br>1 texel/block | VK_FORMAT_D16_UNORM_S8_UINT |
| D24S8<br>Block size 4 byte<br>1x1x1 block extent<br>1 texel/block | VK_FORMAT_D24_UNORM_S8_UINT |
| D32S8<br>Block size 5 byte<br>1x1x1 block extent<br>1 texel/block | VK_FORMAT_D32_SFLOAT_S8_UINT |
| BC1_RGB<br>Block size 8 byte<br>4x4x1 block extent<br>16 texel/block | VK_FORMAT_BC1_RGB_UNORM_BLOCK,<br>VK_FORMAT_BC1_RGB_SRGB_BLOCK |
| BC1_RGBA<br>Block size 8 byte<br>4x4x1 block extent<br>16 texel/block | VK_FORMAT_BC1_RGBA_UNORM_BLOCK,<br>VK_FORMAT_BC1_RGBA_SRGB_BLOCK |
| BC2<br>Block size 16 byte<br>4x4x1 block extent<br>16 texel/block | VK_FORMAT_BC2_UNORM_BLOCK,<br>VK_FORMAT_BC2_SRGB_BLOCK |

| Class, Texel Block Size, Texel Block Extent, # Texels/Block | Formats |
|---|---|
| BC3<br>Block size 16 byte<br>4x4x1 block extent<br>16 texel/block | VK_FORMAT_BC3_UNORM_BLOCK,<br>VK_FORMAT_BC3_SRGB_BLOCK |
| BC4<br>Block size 8 byte<br>4x4x1 block extent<br>16 texel/block | VK_FORMAT_BC4_UNORM_BLOCK,<br>VK_FORMAT_BC4_SNORM_BLOCK |
| BC5<br>Block size 16 byte<br>4x4x1 block extent<br>16 texel/block | VK_FORMAT_BC5_UNORM_BLOCK,<br>VK_FORMAT_BC5_SNORM_BLOCK |
| BC6H<br>Block size 16 byte<br>4x4x1 block extent<br>16 texel/block | VK_FORMAT_BC6H_UFLOAT_BLOCK,<br>VK_FORMAT_BC6H_SFLOAT_BLOCK |
| BC7<br>Block size 16 byte<br>4x4x1 block extent<br>16 texel/block | VK_FORMAT_BC7_UNORM_BLOCK,<br>VK_FORMAT_BC7_SRGB_BLOCK |
| ETC2_RGB<br>Block size 8 byte<br>4x4x1 block extent<br>16 texel/block | VK_FORMAT_ETC2_R8G8B8_UNORM_BLOCK,<br>VK_FORMAT_ETC2_R8G8B8_SRGB_BLOCK |
| ETC2_RGBA<br>Block size 8 byte<br>4x4x1 block extent<br>16 texel/block | VK_FORMAT_ETC2_R8G8B8A1_UNORM_BLOCK,<br>VK_FORMAT_ETC2_R8G8B8A1_SRGB_BLOCK |
| ETC2_EAC_RGBA<br>Block size 16 byte<br>4x4x1 block extent<br>16 texel/block | VK_FORMAT_ETC2_R8G8B8A8_UNORM_BLOCK,<br>VK_FORMAT_ETC2_R8G8B8A8_SRGB_BLOCK |
| EAC_R<br>Block size 8 byte<br>4x4x1 block extent<br>16 texel/block | VK_FORMAT_EAC_R11_UNORM_BLOCK,<br>VK_FORMAT_EAC_R11_SNORM_BLOCK |
| EAC_RG<br>Block size 16 byte<br>4x4x1 block extent<br>16 texel/block | VK_FORMAT_EAC_R11G11_UNORM_BLOCK,<br>VK_FORMAT_EAC_R11G11_SNORM_BLOCK |

| Class, Texel Block Size, Texel Block Extent, # Texels/Block | Formats |
|---|---|
| ASTC_4x4<br>Block size 16 byte<br>4x4x1 block extent<br>16 texel/block | VK_FORMAT_ASTC_4x4_SFLOAT_BLOCK,<br>VK_FORMAT_ASTC_4x4_UNORM_BLOCK,<br>VK_FORMAT_ASTC_4x4_SRGB_BLOCK |
| ASTC_5x4<br>Block size 16 byte<br>5x4x1 block extent<br>20 texel/block | VK_FORMAT_ASTC_5x4_SFLOAT_BLOCK,<br>VK_FORMAT_ASTC_5x4_UNORM_BLOCK,<br>VK_FORMAT_ASTC_5x4_SRGB_BLOCK |
| ASTC_5x5<br>Block size 16 byte<br>5x5x1 block extent<br>25 texel/block | VK_FORMAT_ASTC_5x5_SFLOAT_BLOCK,<br>VK_FORMAT_ASTC_5x5_UNORM_BLOCK,<br>VK_FORMAT_ASTC_5x5_SRGB_BLOCK |
| ASTC_6x5<br>Block size 16 byte<br>6x5x1 block extent<br>30 texel/block | VK_FORMAT_ASTC_6x5_SFLOAT_BLOCK,<br>VK_FORMAT_ASTC_6x5_UNORM_BLOCK,<br>VK_FORMAT_ASTC_6x5_SRGB_BLOCK |
| ASTC_6x6<br>Block size 16 byte<br>6x6x1 block extent<br>36 texel/block | VK_FORMAT_ASTC_6x6_SFLOAT_BLOCK,<br>VK_FORMAT_ASTC_6x6_UNORM_BLOCK,<br>VK_FORMAT_ASTC_6x6_SRGB_BLOCK |
| ASTC_8x5<br>Block size 16 byte<br>8x5x1 block extent<br>40 texel/block | VK_FORMAT_ASTC_8x5_SFLOAT_BLOCK,<br>VK_FORMAT_ASTC_8x5_UNORM_BLOCK,<br>VK_FORMAT_ASTC_8x5_SRGB_BLOCK |
| ASTC_8x6<br>Block size 16 byte<br>8x6x1 block extent<br>48 texel/block | VK_FORMAT_ASTC_8x6_SFLOAT_BLOCK,<br>VK_FORMAT_ASTC_8x6_UNORM_BLOCK,<br>VK_FORMAT_ASTC_8x6_SRGB_BLOCK |
| ASTC_8x8<br>Block size 16 byte<br>8x8x1 block extent<br>64 texel/block | VK_FORMAT_ASTC_8x8_SFLOAT_BLOCK,<br>VK_FORMAT_ASTC_8x8_UNORM_BLOCK,<br>VK_FORMAT_ASTC_8x8_SRGB_BLOCK |
| ASTC_10x5<br>Block size 16 byte<br>10x5x1 block extent<br>50 texel/block | VK_FORMAT_ASTC_10x5_SFLOAT_BLOCK,<br>VK_FORMAT_ASTC_10x5_UNORM_BLOCK,<br>VK_FORMAT_ASTC_10x5_SRGB_BLOCK |
| ASTC_10x6<br>Block size 16 byte<br>10x6x1 block extent<br>60 texel/block | VK_FORMAT_ASTC_10x6_SFLOAT_BLOCK,<br>VK_FORMAT_ASTC_10x6_UNORM_BLOCK,<br>VK_FORMAT_ASTC_10x6_SRGB_BLOCK |

| Class, Texel Block Size, Texel Block Extent, # Texels/Block | Formats |
|---|---|
| ASTC_10x8<br>Block size 16 byte<br>10x8x1 block extent<br>80 texel/block | VK_FORMAT_ASTC_10x8_SFLOAT_BLOCK,<br>VK_FORMAT_ASTC_10x8_UNORM_BLOCK,<br>VK_FORMAT_ASTC_10x8_SRGB_BLOCK |
| ASTC_10x10<br>Block size 16 byte<br>10x10x1 block extent<br>100 texel/block | VK_FORMAT_ASTC_10x10_SFLOAT_BLOCK,<br>VK_FORMAT_ASTC_10x10_UNORM_BLOCK,<br>VK_FORMAT_ASTC_10x10_SRGB_BLOCK |
| ASTC_12x10<br>Block size 16 byte<br>12x10x1 block extent<br>120 texel/block | VK_FORMAT_ASTC_12x10_SFLOAT_BLOCK,<br>VK_FORMAT_ASTC_12x10_UNORM_BLOCK,<br>VK_FORMAT_ASTC_12x10_SRGB_BLOCK |
| ASTC_12x12<br>Block size 16 byte<br>12x12x1 block extent<br>144 texel/block | VK_FORMAT_ASTC_12x12_SFLOAT_BLOCK,<br>VK_FORMAT_ASTC_12x12_UNORM_BLOCK,<br>VK_FORMAT_ASTC_12x12_SRGB_BLOCK |
| 32-bit G8B8G8R8<br>Block size 4 byte<br>2x1x1 block extent<br>1 texel/block | VK_FORMAT_G8B8G8R8_422_UNORM |
| 32-bit B8G8R8G8<br>Block size 4 byte<br>2x1x1 block extent<br>1 texel/block | VK_FORMAT_B8G8R8G8_422_UNORM |
| 8-bit 3-plane 420<br>Block size 3 byte<br>1x1x1 block extent<br>1 texel/block | VK_FORMAT_G8_B8_R8_3PLANE_420_UNORM |
| 8-bit 2-plane 420<br>Block size 3 byte<br>1x1x1 block extent<br>1 texel/block | VK_FORMAT_G8_B8R8_2PLANE_420_UNORM |
| 8-bit 3-plane 422<br>Block size 3 byte<br>1x1x1 block extent<br>1 texel/block | VK_FORMAT_G8_B8_R8_3PLANE_422_UNORM |
| 8-bit 2-plane 422<br>Block size 3 byte<br>1x1x1 block extent<br>1 texel/block | VK_FORMAT_G8_B8R8_2PLANE_422_UNORM |

| Class, Texel Block Size, Texel Block Extent, # Texels/Block | Formats |
|---|---|
| 8-bit 3-plane 444<br>Block size 3 byte<br>1x1x1 block extent<br>1 texel/block | VK_FORMAT_G8_B8_R8_3PLANE_444_UNORM |
| 64-bit R10G10B10A10<br>Block size 8 byte<br>1x1x1 block extent<br>1 texel/block | VK_FORMAT_R10X6G10X6B10X6A10X6_UNORM_4PACK16 |
| 64-bit G10B10G10R10<br>Block size 8 byte<br>2x1x1 block extent<br>1 texel/block | VK_FORMAT_G10X6B10X6G10X6R10X6_422_UNORM_4PACK16 |
| 64-bit B10G10R10G10<br>Block size 8 byte<br>2x1x1 block extent<br>1 texel/block | VK_FORMAT_B10X6G10X6R10X6G10X6_422_UNORM_4PACK16 |
| 10-bit 3-plane 420<br>Block size 6 byte<br>1x1x1 block extent<br>1 texel/block | VK_FORMAT_G10X6_B10X6_R10X6_3PLANE_420_UNORM_3PACK16 |
| 10-bit 2-plane 420<br>Block size 6 byte<br>1x1x1 block extent<br>1 texel/block | VK_FORMAT_G10X6_B10X6R10X6_2PLANE_420_UNORM_3PACK16 |
| 10-bit 3-plane 422<br>Block size 6 byte<br>1x1x1 block extent<br>1 texel/block | VK_FORMAT_G10X6_B10X6_R10X6_3PLANE_422_UNORM_3PACK16 |
| 10-bit 2-plane 422<br>Block size 6 byte<br>1x1x1 block extent<br>1 texel/block | VK_FORMAT_G10X6_B10X6R10X6_2PLANE_422_UNORM_3PACK16 |
| 10-bit 3-plane 444<br>Block size 6 byte<br>1x1x1 block extent<br>1 texel/block | VK_FORMAT_G10X6_B10X6_R10X6_3PLANE_444_UNORM_3PACK16 |
| 64-bit R12G12B12A12<br>Block size 8 byte<br>1x1x1 block extent<br>1 texel/block | VK_FORMAT_R12X4G12X4B12X4A12X4_UNORM_4PACK16 |

| Class, Texel Block Size, Texel Block Extent, # Texels/Block | Formats |
|---|---|
| 64-bit G12B12G12R12<br>Block size 8 byte<br>2x1x1 block extent<br>1 texel/block | VK_FORMAT_G12X4B12X4G12X4R12X4_422_UNORM_4PACK16 |
| 64-bit B12G12R12G12<br>Block size 8 byte<br>2x1x1 block extent<br>1 texel/block | VK_FORMAT_B12X4G12X4R12X4G12X4_422_UNORM_4PACK16 |
| 12-bit 3-plane 420<br>Block size 6 byte<br>1x1x1 block extent<br>1 texel/block | VK_FORMAT_G12X4_B12X4_R12X4_3PLANE_420_UNORM_3PACK16 |
| 12-bit 2-plane 420<br>Block size 6 byte<br>1x1x1 block extent<br>1 texel/block | VK_FORMAT_G12X4_B12X4R12X4_2PLANE_420_UNORM_3PACK16 |
| 12-bit 3-plane 422<br>Block size 6 byte<br>1x1x1 block extent<br>1 texel/block | VK_FORMAT_G12X4_B12X4_R12X4_3PLANE_422_UNORM_3PACK16 |
| 12-bit 2-plane 422<br>Block size 6 byte<br>1x1x1 block extent<br>1 texel/block | VK_FORMAT_G12X4_B12X4R12X4_2PLANE_422_UNORM_3PACK16 |
| 12-bit 3-plane 444<br>Block size 6 byte<br>1x1x1 block extent<br>1 texel/block | VK_FORMAT_G12X4_B12X4_R12X4_3PLANE_444_UNORM_3PACK16 |
| 64-bit G16B16G16R16<br>Block size 8 byte<br>2x1x1 block extent<br>1 texel/block | VK_FORMAT_G16B16G16R16_422_UNORM |
| 64-bit B16G16R16G16<br>Block size 8 byte<br>2x1x1 block extent<br>1 texel/block | VK_FORMAT_B16G16R16G16_422_UNORM |
| 16-bit 3-plane 420<br>Block size 6 byte<br>1x1x1 block extent<br>1 texel/block | VK_FORMAT_G16_B16_R16_3PLANE_420_UNORM |

| Class, Texel Block Size, Texel Block Extent, # Texels/Block | Formats |
|---|---|
| 16-bit 2-plane 420<br>Block size 6 byte<br>1x1x1 block extent<br>1 texel/block | VK_FORMAT_G16_B16R16_2PLANE_420_UNORM |
| 16-bit 3-plane 422<br>Block size 6 byte<br>1x1x1 block extent<br>1 texel/block | VK_FORMAT_G16_B16_R16_3PLANE_422_UNORM |
| 16-bit 2-plane 422<br>Block size 6 byte<br>1x1x1 block extent<br>1 texel/block | VK_FORMAT_G16_B16R16_2PLANE_422_UNORM |
| 16-bit 3-plane 444<br>Block size 6 byte<br>1x1x1 block extent<br>1 texel/block | VK_FORMAT_G16_B16_R16_3PLANE_444_UNORM |
| PVRTC1_2BPP<br>Block size 8 byte<br>8x4x1 block extent<br>1 texel/block | VK_FORMAT_PVRTC1_2BPP_UNORM_BLOCK_IMG,<br>VK_FORMAT_PVRTC1_2BPP_SRGB_BLOCK_IMG |
| PVRTC1_4BPP<br>Block size 8 byte<br>4x4x1 block extent<br>1 texel/block | VK_FORMAT_PVRTC1_4BPP_UNORM_BLOCK_IMG,<br>VK_FORMAT_PVRTC1_4BPP_SRGB_BLOCK_IMG |
| PVRTC2_2BPP<br>Block size 8 byte<br>8x4x1 block extent<br>1 texel/block | VK_FORMAT_PVRTC2_2BPP_UNORM_BLOCK_IMG,<br>VK_FORMAT_PVRTC2_2BPP_SRGB_BLOCK_IMG |
| PVRTC2_4BPP<br>Block size 8 byte<br>4x4x1 block extent<br>1 texel/block | VK_FORMAT_PVRTC2_4BPP_UNORM_BLOCK_IMG,<br>VK_FORMAT_PVRTC2_4BPP_SRGB_BLOCK_IMG |
| 8-bit 2-plane 444<br>Block size 3 byte<br>1x1x1 block extent<br>1 texel/block | VK_FORMAT_G8_B8R8_2PLANE_444_UNORM |
| 10-bit 2-plane 444<br>Block size 6 byte<br>1x1x1 block extent<br>1 texel/block | VK_FORMAT_G10X6_B10X6R10X6_2PLANE_444_UNORM_3PACK16 |

| Class, Texel Block Size, Texel Block Extent, # Texels/Block | Formats |
|---|---|
| 12-bit 2-plane 444<br>Block size 6 byte<br>1x1x1 block extent<br>1 texel/block | VK_FORMAT_G12X4_B12X4R12X4_2PLANE_444_UNORM_3PACK16 |
| 16-bit 2-plane 444<br>Block size 6 byte<br>1x1x1 block extent<br>1 texel/block | VK_FORMAT_G16_B16R16_2PLANE_444_UNORM |

**Size Compatibility**

Color formats with the same texel block size are considered *size-compatible* as long as neither or both are alpha formats (e.g., VK_FORMAT_A8_UNORM_KHR). If two size-compatible formats have different block extents (i.e. for compressed formats), then an image with size A × B × C in one format with a block extent of a × b × c can be represented as an image with size X × Y × Z in the other format with block extent x × y × z at the ratio between the block extents for each format, where

$$\lceil A/a \rceil = \lceil X/x \rceil$$

$$\lceil B/b \rceil = \lceil Y/y \rceil$$

$$\lceil C/c \rceil = \lceil Z/z \rceil$$

**NOTE** For example, a 7x3 image in the VK_FORMAT_ASTC_8x5_UNORM_BLOCK format can be represented as a 1x1 VK_FORMAT_R64G64_UINT image.

Images created with the VK_IMAGE_CREATE_BLOCK_TEXEL_VIEW_COMPATIBLE_BIT flag can have size-compatible views created from them to enable access via different size-compatible formats. Image views created in this way will be sized to match the expectations of the block extents noted above.

Copy operations are able to copy between size-compatible formats in different resources to enable manipulation of data in different formats. The extent used in these copy operations always matches the source image, and is resized to the expectations of the block extents noted above for the destination image.

# 48.2. Format Properties

To query supported format features which are properties of the physical device, call:

```
// Provided by VK_VERSION_1_0
void vkGetPhysicalDeviceFormatProperties(
    VkPhysicalDevice                            physicalDevice,
    VkFormat                                    format,
    VkFormatProperties*                         pFormatProperties);
```

- `physicalDevice` is the physical device from which to query the format properties.
- `format` is the format whose properties are queried.
- `pFormatProperties` is a pointer to a VkFormatProperties structure in which physical device properties for `format` are returned.

### Valid Usage (Implicit)

- VUID-vkGetPhysicalDeviceFormatProperties-physicalDevice-parameter
  `physicalDevice` **must** be a valid VkPhysicalDevice handle
- VUID-vkGetPhysicalDeviceFormatProperties-format-parameter
  `format` **must** be a valid VkFormat value
- VUID-vkGetPhysicalDeviceFormatProperties-pFormatProperties-parameter
  `pFormatProperties` **must** be a valid pointer to a VkFormatProperties structure

The `VkFormatProperties` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkFormatProperties {
    VkFormatFeatureFlags    linearTilingFeatures;
    VkFormatFeatureFlags    optimalTilingFeatures;
    VkFormatFeatureFlags    bufferFeatures;
} VkFormatProperties;
```

- `linearTilingFeatures` is a bitmask of VkFormatFeatureFlagBits specifying features supported by images created with a `tiling` parameter of `VK_IMAGE_TILING_LINEAR`.
- `optimalTilingFeatures` is a bitmask of VkFormatFeatureFlagBits specifying features supported by images created with a `tiling` parameter of `VK_IMAGE_TILING_OPTIMAL`.
- `bufferFeatures` is a bitmask of VkFormatFeatureFlagBits specifying features supported by buffers.

**NOTE** If no format feature flags are supported, the format itself is not supported, and images of that format cannot be created.

If `format` is a block-compressed format, then `bufferFeatures` **must** not support any features for the format.

If `format` is not a multi-plane format then `linearTilingFeatures` and `optimalTilingFeatures` **must** not

contain VK_FORMAT_FEATURE_DISJOINT_BIT.

Bits which **can** be set in the VkFormatProperties features linearTilingFeatures, optimalTilingFeatures, VkDrmFormatModifierPropertiesEXT::drmFormatModifierTilingFeatures, and bufferFeatures are:

```
// Provided by VK_VERSION_1_0
typedef enum VkFormatFeatureFlagBits {
    VK_FORMAT_FEATURE_SAMPLED_IMAGE_BIT = 0x00000001,
    VK_FORMAT_FEATURE_STORAGE_IMAGE_BIT = 0x00000002,
    VK_FORMAT_FEATURE_STORAGE_IMAGE_ATOMIC_BIT = 0x00000004,
    VK_FORMAT_FEATURE_UNIFORM_TEXEL_BUFFER_BIT = 0x00000008,
    VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_BIT = 0x00000010,
    VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_ATOMIC_BIT = 0x00000020,
    VK_FORMAT_FEATURE_VERTEX_BUFFER_BIT = 0x00000040,
    VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BIT = 0x00000080,
    VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT = 0x00000100,
    VK_FORMAT_FEATURE_DEPTH_STENCIL_ATTACHMENT_BIT = 0x00000200,
    VK_FORMAT_FEATURE_BLIT_SRC_BIT = 0x00000400,
    VK_FORMAT_FEATURE_BLIT_DST_BIT = 0x00000800,
    VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT = 0x00001000,
  // Provided by VK_VERSION_1_1
    VK_FORMAT_FEATURE_TRANSFER_SRC_BIT = 0x00004000,
  // Provided by VK_VERSION_1_1
    VK_FORMAT_FEATURE_TRANSFER_DST_BIT = 0x00008000,
  // Provided by VK_VERSION_1_1
    VK_FORMAT_FEATURE_MIDPOINT_CHROMA_SAMPLES_BIT = 0x00020000,
  // Provided by VK_VERSION_1_1
    VK_FORMAT_FEATURE_SAMPLED_IMAGE_YCBCR_CONVERSION_LINEAR_FILTER_BIT = 0x00040000,
  // Provided by VK_VERSION_1_1

VK_FORMAT_FEATURE_SAMPLED_IMAGE_YCBCR_CONVERSION_SEPARATE_RECONSTRUCTION_FILTER_BIT =
0x00080000,
  // Provided by VK_VERSION_1_1

VK_FORMAT_FEATURE_SAMPLED_IMAGE_YCBCR_CONVERSION_CHROMA_RECONSTRUCTION_EXPLICIT_BIT =
0x00100000,
  // Provided by VK_VERSION_1_1

VK_FORMAT_FEATURE_SAMPLED_IMAGE_YCBCR_CONVERSION_CHROMA_RECONSTRUCTION_EXPLICIT_FORCEA
BLE_BIT = 0x00200000,
  // Provided by VK_VERSION_1_1
    VK_FORMAT_FEATURE_DISJOINT_BIT = 0x00400000,
  // Provided by VK_VERSION_1_1
    VK_FORMAT_FEATURE_COSITED_CHROMA_SAMPLES_BIT = 0x00800000,
    VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT = 0x00010000,
  // Provided by VK_KHR_video_decode_queue
    VK_FORMAT_FEATURE_VIDEO_DECODE_OUTPUT_BIT_KHR = 0x02000000,
  // Provided by VK_KHR_video_decode_queue
    VK_FORMAT_FEATURE_VIDEO_DECODE_DPB_BIT_KHR = 0x04000000,
```

```
  // Provided by VK_KHR_acceleration_structure
    VK_FORMAT_FEATURE_ACCELERATION_STRUCTURE_VERTEX_BUFFER_BIT_KHR = 0x20000000,
  // Provided by VK_EXT_filter_cubic
    VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_CUBIC_BIT_EXT = 0x00002000,
  // Provided by VK_EXT_fragment_density_map
    VK_FORMAT_FEATURE_FRAGMENT_DENSITY_MAP_BIT_EXT = 0x01000000,
  // Provided by VK_KHR_fragment_shading_rate
    VK_FORMAT_FEATURE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR = 0x40000000,
  // Provided by VK_KHR_video_encode_queue
    VK_FORMAT_FEATURE_VIDEO_ENCODE_INPUT_BIT_KHR = 0x08000000,
  // Provided by VK_KHR_video_encode_queue
    VK_FORMAT_FEATURE_VIDEO_ENCODE_DPB_BIT_KHR = 0x10000000,
  // Provided by VK_IMG_filter_cubic
    VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_CUBIC_BIT_IMG =
VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_CUBIC_BIT_EXT,
  // Provided by VK_KHR_maintenance1
    VK_FORMAT_FEATURE_TRANSFER_SRC_BIT_KHR = VK_FORMAT_FEATURE_TRANSFER_SRC_BIT,
  // Provided by VK_KHR_maintenance1
    VK_FORMAT_FEATURE_TRANSFER_DST_BIT_KHR = VK_FORMAT_FEATURE_TRANSFER_DST_BIT,
  // Provided by VK_EXT_sampler_filter_minmax
    VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT_EXT =
VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT,
  // Provided by VK_KHR_sampler_ycbcr_conversion
    VK_FORMAT_FEATURE_MIDPOINT_CHROMA_SAMPLES_BIT_KHR =
VK_FORMAT_FEATURE_MIDPOINT_CHROMA_SAMPLES_BIT,
  // Provided by VK_KHR_sampler_ycbcr_conversion
    VK_FORMAT_FEATURE_SAMPLED_IMAGE_YCBCR_CONVERSION_LINEAR_FILTER_BIT_KHR =
VK_FORMAT_FEATURE_SAMPLED_IMAGE_YCBCR_CONVERSION_LINEAR_FILTER_BIT,
  // Provided by VK_KHR_sampler_ycbcr_conversion

VK_FORMAT_FEATURE_SAMPLED_IMAGE_YCBCR_CONVERSION_SEPARATE_RECONSTRUCTION_FILTER_BIT_KH
R =
VK_FORMAT_FEATURE_SAMPLED_IMAGE_YCBCR_CONVERSION_SEPARATE_RECONSTRUCTION_FILTER_BIT,
  // Provided by VK_KHR_sampler_ycbcr_conversion

VK_FORMAT_FEATURE_SAMPLED_IMAGE_YCBCR_CONVERSION_CHROMA_RECONSTRUCTION_EXPLICIT_BIT_KH
R =
VK_FORMAT_FEATURE_SAMPLED_IMAGE_YCBCR_CONVERSION_CHROMA_RECONSTRUCTION_EXPLICIT_BIT,
  // Provided by VK_KHR_sampler_ycbcr_conversion

VK_FORMAT_FEATURE_SAMPLED_IMAGE_YCBCR_CONVERSION_CHROMA_RECONSTRUCTION_EXPLICIT_FORCEA
BLE_BIT_KHR =
VK_FORMAT_FEATURE_SAMPLED_IMAGE_YCBCR_CONVERSION_CHROMA_RECONSTRUCTION_EXPLICIT_FORCEA
BLE_BIT,
  // Provided by VK_KHR_sampler_ycbcr_conversion
    VK_FORMAT_FEATURE_DISJOINT_BIT_KHR = VK_FORMAT_FEATURE_DISJOINT_BIT,
  // Provided by VK_KHR_sampler_ycbcr_conversion
    VK_FORMAT_FEATURE_COSITED_CHROMA_SAMPLES_BIT_KHR =
VK_FORMAT_FEATURE_COSITED_CHROMA_SAMPLES_BIT,
} VkFormatFeatureFlagBits;
```

These values all have the same meaning as the equivalently named values for VkFormatFeatureFlags2 and **may** be set in linearTilingFeatures, optimalTilingFeatures, and VkDrmFormatModifierPropertiesEXT::drmFormatModifierTilingFeatures, specifying that the features are supported by images or image views or sampler $Y'C_BC_R$ conversion objects created with the queried vkGetPhysicalDeviceFormatProperties::format:

- VK_FORMAT_FEATURE_SAMPLED_IMAGE_BIT specifies that an image view **can** be sampled from.
- VK_FORMAT_FEATURE_STORAGE_IMAGE_BIT specifies that an image view **can** be used as a storage image.
- VK_FORMAT_FEATURE_STORAGE_IMAGE_ATOMIC_BIT specifies that an image view **can** be used as storage image that supports atomic operations.
- VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BIT specifies that an image view **can** be used as a framebuffer color attachment and as an input attachment.
- VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT specifies that an image view **can** be used as a framebuffer color attachment that supports blending.
- VK_FORMAT_FEATURE_DEPTH_STENCIL_ATTACHMENT_BIT specifies that an image view **can** be used as a framebuffer depth/stencil attachment and as an input attachment.
- VK_FORMAT_FEATURE_BLIT_SRC_BIT specifies that an image **can** be used as srcImage for the vkCmdBlitImage2KHR and vkCmdBlitImage commands.
- VK_FORMAT_FEATURE_BLIT_DST_BIT specifies that an image **can** be used as dstImage for the vkCmdBlitImage2KHR and vkCmdBlitImage commands.
- VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT specifies that if VK_FORMAT_FEATURE_SAMPLED_IMAGE_BIT is also set, an image view **can** be used with a sampler that has either of magFilter or minFilter set to VK_FILTER_LINEAR, or mipmapMode set to VK_SAMPLER_MIPMAP_MODE_LINEAR. If VK_FORMAT_FEATURE_BLIT_SRC_BIT is also set, an image can be used as the srcImage to vkCmdBlitImage2KHR and vkCmdBlitImage with a filter of VK_FILTER_LINEAR. This bit **must** only be exposed for formats that also support the VK_FORMAT_FEATURE_SAMPLED_IMAGE_BIT or VK_FORMAT_FEATURE_BLIT_SRC_BIT.

  If the format being queried is a depth/stencil format, this bit only specifies that the depth aspect (not the stencil aspect) of an image of this format supports linear filtering, and that linear filtering of the depth aspect is supported whether depth compare is enabled in the sampler or not. Where depth comparison is supported it **may** be linear filtered whether this bit is present or not, but where this bit is not present the filtered value **may** be computed in an implementation-dependent manner which differs from the normal rules of linear filtering. The resulting value **must** be in the range [0,1] and **should** be proportional to, or a weighted average of, the number of comparison passes or failures.

- VK_FORMAT_FEATURE_TRANSFER_SRC_BIT specifies that an image **can** be used as a source image for copy commands. If the application apiVersion is Vulkan 1.0 and VK_KHR_maintenance1 is not supported, VK_FORMAT_FEATURE_TRANSFER_SRC_BIT is implied to be set when the format feature flag is not 0.
- VK_FORMAT_FEATURE_TRANSFER_DST_BIT specifies that an image **can** be used as a destination image for copy commands and clear commands. If the application apiVersion is Vulkan 1.0 and VK_KHR_maintenance1 is not supported, VK_FORMAT_FEATURE_TRANSFER_DST_BIT is implied to be set

when the format feature flag is not 0.

- VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT specifies VkImage **can** be used as a sampled image with a min or max VkSamplerReductionMode. This bit **must** only be exposed for formats that also support the VK_FORMAT_FEATURE_SAMPLED_IMAGE_BIT.
- VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_CUBIC_BIT_EXT specifies that VkImage **can** be used with a sampler that has either of magFilter or minFilter set to VK_FILTER_CUBIC_EXT, or be the source image for a blit with filter set to VK_FILTER_CUBIC_EXT. This bit **must** only be exposed for formats that also support the VK_FORMAT_FEATURE_SAMPLED_IMAGE_BIT. If the format being queried is a depth/stencil format, this only specifies that the depth aspect is cubic filterable.
- VK_FORMAT_FEATURE_MIDPOINT_CHROMA_SAMPLES_BIT specifies that an application **can** define a sampler $Y'C_BC_R$ conversion using this format as a source, and that an image of this format **can** be used with a VkSamplerYcbcrConversionCreateInfo xChromaOffset and/or yChromaOffset of VK_CHROMA_LOCATION_MIDPOINT. Otherwise both xChromaOffset and yChromaOffset **must** be VK_CHROMA_LOCATION_COSITED_EVEN. If a format does not incorporate chroma downsampling (it is not a “422” or “420” format) but the implementation supports sampler $Y'C_BC_R$ conversion for this format, the implementation **must** set VK_FORMAT_FEATURE_MIDPOINT_CHROMA_SAMPLES_BIT.
- VK_FORMAT_FEATURE_COSITED_CHROMA_SAMPLES_BIT specifies that an application **can** define a sampler $Y'C_BC_R$ conversion using this format as a source, and that an image of this format **can** be used with a VkSamplerYcbcrConversionCreateInfo xChromaOffset and/or yChromaOffset of VK_CHROMA_LOCATION_COSITED_EVEN. Otherwise both xChromaOffset and yChromaOffset **must** be VK_CHROMA_LOCATION_MIDPOINT. If neither VK_FORMAT_FEATURE_COSITED_CHROMA_SAMPLES_BIT nor VK_FORMAT_FEATURE_MIDPOINT_CHROMA_SAMPLES_BIT is set, the application **must** not define a sampler $Y'C_BC_R$ conversion using this format as a source.
- VK_FORMAT_FEATURE_SAMPLED_IMAGE_YCBCR_CONVERSION_LINEAR_FILTER_BIT specifies that an application **can** define a sampler $Y'C_BC_R$ conversion using this format as a source with chromaFilter set to VK_FILTER_LINEAR.
- VK_FORMAT_FEATURE_SAMPLED_IMAGE_YCBCR_CONVERSION_SEPARATE_RECONSTRUCTION_FILTER_BIT specifies that the format can have different chroma, min, and mag filters.
- VK_FORMAT_FEATURE_SAMPLED_IMAGE_YCBCR_CONVERSION_CHROMA_RECONSTRUCTION_EXPLICIT_BIT specifies that reconstruction is explicit, as described in Chroma Reconstruction. If this bit is not present, reconstruction is implicit by default.
- VK_FORMAT_FEATURE_SAMPLED_IMAGE_YCBCR_CONVERSION_CHROMA_RECONSTRUCTION_EXPLICIT_FORCEABLE_BIT specifies that reconstruction **can** be forcibly made explicit by setting VkSamplerYcbcrConversionCreateInfo::forceExplicitReconstruction to VK_TRUE. If the format being queried supports VK_FORMAT_FEATURE_SAMPLED_IMAGE_YCBCR_CONVERSION_CHROMA_RECONSTRUCTION_EXPLICIT_BIT it **must** also support VK_FORMAT_FEATURE_SAMPLED_IMAGE_YCBCR_CONVERSION_CHROMA_RECONSTRUCTION_EXPLICIT_FORCEABLE_BIT.
- VK_FORMAT_FEATURE_DISJOINT_BIT specifies that a multi-planar image **can** have the VK_IMAGE_CREATE_DISJOINT_BIT set during image creation. An implementation **must** not set VK_FORMAT_FEATURE_DISJOINT_BIT for *single-plane formats*.
- VK_FORMAT_FEATURE_FRAGMENT_DENSITY_MAP_BIT_EXT specifies that an image view **can** be used as a fragment density map attachment.

- VK_FORMAT_FEATURE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR specifies that an image view **can** be used as a fragment shading rate attachment. An implementation **must** not set this feature for formats with a numeric format other than UINT, or set it as a buffer feature.
- VK_FORMAT_FEATURE_VIDEO_DECODE_OUTPUT_BIT_KHR specifies that an image view with this format **can** be used as a decode output picture in video decode operations.
- VK_FORMAT_FEATURE_VIDEO_DECODE_DPB_BIT_KHR specifies that an image view with this format **can** be used as an output reconstructed picture or an input reference picture in video decode operations.
- VK_FORMAT_FEATURE_VIDEO_ENCODE_INPUT_BIT_KHR specifies that an image view with this format **can** be used as an encode input picture in video encode operations.
- VK_FORMAT_FEATURE_VIDEO_ENCODE_DPB_BIT_KHR specifies that an image view with this format **can** be used as an output reconstructed picture or an input reference picture in video encode operations.

**NOTE** Specific video profiles **may** have additional restrictions on the format and other image creation parameters corresponding to image views used by video coding operations that **can** be enumerated using the vkGetPhysicalDeviceVideoFormatPropertiesKHR command.

The following bits **may** be set in bufferFeatures, specifying that the features are supported by buffers or buffer views created with the queried vkGetPhysicalDeviceFormatProperties::format:

- VK_FORMAT_FEATURE_UNIFORM_TEXEL_BUFFER_BIT specifies that the format **can** be used to create a buffer view that **can** be bound to a VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER descriptor.
- VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_BIT specifies that the format **can** be used to create a buffer view that **can** be bound to a VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER descriptor.
- VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_ATOMIC_BIT specifies that atomic operations are supported on VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER with this format.
- VK_FORMAT_FEATURE_VERTEX_BUFFER_BIT specifies that the format **can** be used as a vertex attribute format (VkVertexInputAttributeDescription::format).
- VK_FORMAT_FEATURE_ACCELERATION_STRUCTURE_VERTEX_BUFFER_BIT_KHR specifies that the format **can** be used as the vertex format when creating an acceleration structure (VkAccelerationStructureGeometryTrianglesDataKHR::vertexFormat). This format **can** also be used as the vertex format in host memory when doing host acceleration structure builds.

**NOTE** VK_FORMAT_FEATURE_STORAGE_IMAGE_ATOMIC_BIT and VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_ATOMIC_BIT are only intended to be advertised for single-component formats, since SPIR-V atomic operations require a scalar type.

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkFormatFeatureFlags;
```

VkFormatFeatureFlags is a bitmask type for setting a mask of zero or more VkFormatFeatureFlagBits.

To query supported format features which are properties of the physical device, call:

```
// Provided by VK_VERSION_1_1
void vkGetPhysicalDeviceFormatProperties2(
    VkPhysicalDevice                            physicalDevice,
    VkFormat                                    format,
    VkFormatProperties2*                        pFormatProperties);
```

or the equivalent command

```
// Provided by VK_KHR_get_physical_device_properties2
void vkGetPhysicalDeviceFormatProperties2KHR(
    VkPhysicalDevice                            physicalDevice,
    VkFormat                                    format,
    VkFormatProperties2*                        pFormatProperties);
```

- `physicalDevice` is the physical device from which to query the format properties.
- `format` is the format whose properties are queried.
- `pFormatProperties` is a pointer to a VkFormatProperties2 structure in which physical device properties for `format` are returned.

`vkGetPhysicalDeviceFormatProperties2` behaves similarly to vkGetPhysicalDeviceFormatProperties, with the ability to return extended information in a `pNext` chain of output structures.

### Valid Usage (Implicit)

- VUID-vkGetPhysicalDeviceFormatProperties2-physicalDevice-parameter
  `physicalDevice` **must** be a valid VkPhysicalDevice handle
- VUID-vkGetPhysicalDeviceFormatProperties2-format-parameter
  `format` **must** be a valid VkFormat value
- VUID-vkGetPhysicalDeviceFormatProperties2-pFormatProperties-parameter
  `pFormatProperties` **must** be a valid pointer to a VkFormatProperties2 structure

The `VkFormatProperties2` structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkFormatProperties2 {
    VkStructureType       sType;
    void*                 pNext;
    VkFormatProperties    formatProperties;
} VkFormatProperties2;
```

or the equivalent

```
// Provided by VK_KHR_get_physical_device_properties2
typedef VkFormatProperties2 VkFormatProperties2KHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- formatProperties is a VkFormatProperties structure describing features supported by the requested format.

### Valid Usage (Implicit)

- VUID-VkFormatProperties2-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_FORMAT_PROPERTIES_2
- VUID-VkFormatProperties2-pNext-pNext
  Each pNext member of any structure (including this one) in the pNext chain **must** be either NULL or a pointer to a valid instance of VkDrmFormatModifierPropertiesList2EXT, VkDrmFormatModifierPropertiesListEXT, VkFormatProperties3, or VkSubpassResolvePerformanceQueryEXT
- VUID-VkFormatProperties2-sType-unique
  The sType value of each struct in the pNext chain **must** be unique

To obtain the list of Linux DRM format modifiers compatible with a VkFormat, add a VkDrmFormatModifierPropertiesListEXT structure to the pNext chain of VkFormatProperties2.

The VkDrmFormatModifierPropertiesListEXT structure is defined as:

```
// Provided by VK_EXT_image_drm_format_modifier
typedef struct VkDrmFormatModifierPropertiesListEXT {
    VkStructureType                     sType;
    void*                               pNext;
    uint32_t                            drmFormatModifierCount;
    VkDrmFormatModifierPropertiesEXT*   pDrmFormatModifierProperties;
} VkDrmFormatModifierPropertiesListEXT;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- drmFormatModifierCount is an inout parameter related to the number of modifiers compatible with the format, as described below.
- pDrmFormatModifierProperties is either NULL or a pointer to an array of VkDrmFormatModifierPropertiesEXT structures.

If pDrmFormatModifierProperties is NULL, then the function returns in drmFormatModifierCount the number of modifiers compatible with the queried format. Otherwise, the application **must** set drmFormatModifierCount to the length of the array pDrmFormatModifierProperties; the function will

write at most drmFormatModifierCount elements to the array, and will return in drmFormatModifierCount the number of elements written.

Among the elements in array pDrmFormatModifierProperties, each returned drmFormatModifier **must** be unique.

### Valid Usage (Implicit)

- VUID-VkDrmFormatModifierPropertiesListEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_DRM_FORMAT_MODIFIER_PROPERTIES_LIST_EXT

The VkDrmFormatModifierPropertiesEXT structure describes properties of a VkFormat when that format is combined with a Linux DRM format modifier. These properties, like those of VkFormatProperties2, are independent of any particular image.

The VkDrmFormatModifierPropertiesEXT structure is defined as:

```
// Provided by VK_EXT_image_drm_format_modifier
typedef struct VkDrmFormatModifierPropertiesEXT {
    uint64_t                drmFormatModifier;
    uint32_t                drmFormatModifierPlaneCount;
    VkFormatFeatureFlags    drmFormatModifierTilingFeatures;
} VkDrmFormatModifierPropertiesEXT;
```

- drmFormatModifier is a *Linux DRM format modifier*.
- drmFormatModifierPlaneCount is the number of *memory planes* in any image created with format and drmFormatModifier. An image's *memory planecount* is distinct from its *format planecount*, as explained below.
- drmFormatModifierTilingFeatures is a bitmask of VkFormatFeatureFlagBits that are supported by any image created with format and drmFormatModifier.

The returned drmFormatModifierTilingFeatures **must** contain at least one bit.

The implementation **must** not return DRM_FORMAT_MOD_INVALID in drmFormatModifier.

An image's *memory planecount* (as returned by drmFormatModifierPlaneCount) is distinct from its *format planecount* (in the sense of multi-planar $Y'C_BC_R$ formats). In VkImageAspectFlags, each VK_IMAGE_ASPECT_MEMORY_PLANE_i_BIT_EXT represents a *memory plane* and each VK_IMAGE_ASPECT_PLANE_i_BIT a *format plane*.

An image's set of *format planes* is an ordered partition of the image's **content** into separable groups of format components. The ordered partition is encoded in the name of each VkFormat. For example, VK_FORMAT_G8_B8R8_2PLANE_420_UNORM contains two *format planes*; the first plane contains the green component and the second plane contains the blue component and red component. If the format name does not contain PLANE, then the format contains a single plane; for example, VK_FORMAT_R8G8B8A8_UNORM. Some commands, such as vkCmdCopyBufferToImage, do not operate on all format components in the image, but instead operate only on the *format planes* explicitly chosen

by the application and operate on each *format plane* independently.

An image's set of *memory planes* is an ordered partition of the image's **memory** rather than the image's **content**. Each *memory plane* is a contiguous range of memory. The union of an image's *memory planes* is not necessarily contiguous.

If an image is linear, then the partition is the same for *memory planes* and for *format planes*. Therefore, if the returned `drmFormatModifier` is `DRM_FORMAT_MOD_LINEAR`, then `drmFormatModifierPlaneCount` **must** equal the *format planecount*, and `drmFormatModifierTilingFeatures` **must** be identical to the VkFormatProperties2::`linearTilingFeatures` returned in the same `pNext` chain.

If an image is non-linear, then the partition of the image's **memory** into *memory planes* is implementation-specific and **may** be unrelated to the partition of the image's **content** into *format planes*. For example, consider an image whose `format` is `VK_FORMAT_G8_B8_R8_3PLANE_420_UNORM`, `tiling` is `VK_IMAGE_TILING_DRM_FORMAT_MODIFIER_EXT`, whose `drmFormatModifier` is not `DRM_FORMAT_MOD_LINEAR`, and `flags` lacks `VK_IMAGE_CREATE_DISJOINT_BIT`. The image has 3 *format planes*, and commands such vkCmdCopyBufferToImage act on each *format plane* independently as if the data of each *format plane* were separable from the data of the other planes. In a straightforward implementation, the implementation **may** store the image's content in 3 adjacent *memory planes* where each *memory plane* corresponds exactly to a *format plane*. However, the implementation **may** also store the image's content in a single *memory plane* where all format components are combined using an implementation-private block-compressed format; or the implementation **may** store the image's content in a collection of 7 adjacent *memory planes* using an implementation-private sharding technique. Because the image is non-linear and non-disjoint, the implementation has much freedom when choosing the image's placement in memory.

The *memory planecount* applies to function parameters and structures only when the API specifies an explicit requirement on `drmFormatModifierPlaneCount`. In all other cases, the *memory planecount* is ignored.

The list of Linux DRM format modifiers compatible with a VkFormat **can** be obtained by adding a VkDrmFormatModifierPropertiesList2EXT structure to the `pNext` chain of VkFormatProperties2.

The VkDrmFormatModifierPropertiesList2EXT structure is defined as:

```
// Provided by VK_EXT_image_drm_format_modifier with VK_KHR_format_feature_flags2 or
VK_VERSION_1_3
typedef struct VkDrmFormatModifierPropertiesList2EXT {
    VkStructureType                      sType;
    void*                                pNext;
    uint32_t                             drmFormatModifierCount;
    VkDrmFormatModifierProperties2EXT*   pDrmFormatModifierProperties;
} VkDrmFormatModifierPropertiesList2EXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `drmFormatModifierCount` is an inout parameter related to the number of modifiers compatible

with the format, as described below.

- pDrmFormatModifierProperties is either NULL or a pointer to an array of VkDrmFormatModifierProperties2EXT structures.

If pDrmFormatModifierProperties is NULL, the number of modifiers compatible with the queried format is returned in drmFormatModifierCount. Otherwise, the application **must** set drmFormatModifierCount to the length of the array pDrmFormatModifierProperties; the function will write at most drmFormatModifierCount elements to the array, and will return in drmFormatModifierCount the number of elements written.

Among the elements in array pDrmFormatModifierProperties, each returned drmFormatModifier **must** be unique.

Among the elements in array pDrmFormatModifierProperties, the bits reported in drmFormatModifierTilingFeatures **must** include the bits reported in the corresponding element of VkDrmFormatModifierPropertiesListEXT::pDrmFormatModifierProperties.

### Valid Usage (Implicit)

- VUID-VkDrmFormatModifierPropertiesList2EXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_DRM_FORMAT_MODIFIER_PROPERTIES_LIST_2_EXT

The VkDrmFormatModifierProperties2EXT structure describes properties of a VkFormat when that format is combined with a Linux DRM format modifier. These properties, like those of VkFormatProperties2, are independent of any particular image.

The VkDrmFormatModifierPropertiesEXT structure is defined as:

```
// Provided by VK_EXT_image_drm_format_modifier with VK_KHR_format_feature_flags2 or
VK_VERSION_1_3
typedef struct VkDrmFormatModifierProperties2EXT {
    uint64_t                drmFormatModifier;
    uint32_t                drmFormatModifierPlaneCount;
    VkFormatFeatureFlags2   drmFormatModifierTilingFeatures;
} VkDrmFormatModifierProperties2EXT;
```

- drmFormatModifier is a *Linux DRM format modifier*.
- drmFormatModifierPlaneCount is the number of *memory planes* in any image created with format and drmFormatModifier. An image's *memory planecount* is distinct from its *format planecount*, as explained below.
- drmFormatModifierTilingFeatures is a bitmask of VkFormatFeatureFlagBits2 that are supported by any image created with format and drmFormatModifier.

To query supported format extended features which are properties of the physical device, add VkFormatProperties3 structure to the pNext chain of VkFormatProperties2.

The VkFormatProperties3 structure is defined as:

```
typedef struct VkFormatProperties3 {
    VkStructureType          sType;
    void*                    pNext;
    VkFormatFeatureFlags2    linearTilingFeatures;
    VkFormatFeatureFlags2    optimalTilingFeatures;
    VkFormatFeatureFlags2    bufferFeatures;
} VkFormatProperties3;
```

or the equivalent

```
// Provided by VK_KHR_format_feature_flags2
typedef VkFormatProperties3 VkFormatProperties3KHR;
```

- `linearTilingFeatures` is a bitmask of VkFormatFeatureFlagBits2 specifying features supported by images created with a `tiling` parameter of `VK_IMAGE_TILING_LINEAR`.
- `optimalTilingFeatures` is a bitmask of VkFormatFeatureFlagBits2 specifying features supported by images created with a `tiling` parameter of `VK_IMAGE_TILING_OPTIMAL`.
- `bufferFeatures` is a bitmask of VkFormatFeatureFlagBits2 specifying features supported by buffers.

The bits reported in `linearTilingFeatures`, `optimalTilingFeatures` and `bufferFeatures` **must** include the bits reported in the corresponding fields of `VkFormatProperties2::formatProperties`.

### Valid Usage (Implicit)

- VUID-VkFormatProperties3-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_FORMAT_PROPERTIES_3`

Bits which **can** be set in the VkFormatProperties3 features `linearTilingFeatures`, `optimalTilingFeatures`, and `bufferFeatures` are:

```
// Flag bits for VkFormatFeatureFlagBits2
typedef VkFlags64 VkFormatFeatureFlagBits2;
static const VkFormatFeatureFlagBits2 VK_FORMAT_FEATURE_2_SAMPLED_IMAGE_BIT =
0x00000001ULL;
static const VkFormatFeatureFlagBits2 VK_FORMAT_FEATURE_2_SAMPLED_IMAGE_BIT_KHR =
0x00000001ULL;
static const VkFormatFeatureFlagBits2 VK_FORMAT_FEATURE_2_STORAGE_IMAGE_BIT =
0x00000002ULL;
static const VkFormatFeatureFlagBits2 VK_FORMAT_FEATURE_2_STORAGE_IMAGE_BIT_KHR =
0x00000002ULL;
static const VkFormatFeatureFlagBits2 VK_FORMAT_FEATURE_2_STORAGE_IMAGE_ATOMIC_BIT =
0x00000004ULL;
static const VkFormatFeatureFlagBits2 VK_FORMAT_FEATURE_2_STORAGE_IMAGE_ATOMIC_BIT_KHR
= 0x00000004ULL;
```

```
static const VkFormatFeatureFlagBits2 VK_FORMAT_FEATURE_2_UNIFORM_TEXEL_BUFFER_BIT =
0x00000008ULL;
static const VkFormatFeatureFlagBits2 VK_FORMAT_FEATURE_2_UNIFORM_TEXEL_BUFFER_BIT_KHR
= 0x00000008ULL;
static const VkFormatFeatureFlagBits2 VK_FORMAT_FEATURE_2_STORAGE_TEXEL_BUFFER_BIT =
0x00000010ULL;
static const VkFormatFeatureFlagBits2 VK_FORMAT_FEATURE_2_STORAGE_TEXEL_BUFFER_BIT_KHR
= 0x00000010ULL;
static const VkFormatFeatureFlagBits2
VK_FORMAT_FEATURE_2_STORAGE_TEXEL_BUFFER_ATOMIC_BIT = 0x00000020ULL;
static const VkFormatFeatureFlagBits2
VK_FORMAT_FEATURE_2_STORAGE_TEXEL_BUFFER_ATOMIC_BIT_KHR = 0x00000020ULL;
static const VkFormatFeatureFlagBits2 VK_FORMAT_FEATURE_2_VERTEX_BUFFER_BIT =
0x00000040ULL;
static const VkFormatFeatureFlagBits2 VK_FORMAT_FEATURE_2_VERTEX_BUFFER_BIT_KHR =
0x00000040ULL;
static const VkFormatFeatureFlagBits2 VK_FORMAT_FEATURE_2_COLOR_ATTACHMENT_BIT =
0x00000080ULL;
static const VkFormatFeatureFlagBits2 VK_FORMAT_FEATURE_2_COLOR_ATTACHMENT_BIT_KHR =
0x00000080ULL;
static const VkFormatFeatureFlagBits2 VK_FORMAT_FEATURE_2_COLOR_ATTACHMENT_BLEND_BIT =
0x00000100ULL;
static const VkFormatFeatureFlagBits2
VK_FORMAT_FEATURE_2_COLOR_ATTACHMENT_BLEND_BIT_KHR = 0x00000100ULL;
static const VkFormatFeatureFlagBits2 VK_FORMAT_FEATURE_2_DEPTH_STENCIL_ATTACHMENT_BIT
= 0x00000200ULL;
static const VkFormatFeatureFlagBits2
VK_FORMAT_FEATURE_2_DEPTH_STENCIL_ATTACHMENT_BIT_KHR = 0x00000200ULL;
static const VkFormatFeatureFlagBits2 VK_FORMAT_FEATURE_2_BLIT_SRC_BIT =
0x00000400ULL;
static const VkFormatFeatureFlagBits2 VK_FORMAT_FEATURE_2_BLIT_SRC_BIT_KHR =
0x00000400ULL;
static const VkFormatFeatureFlagBits2 VK_FORMAT_FEATURE_2_BLIT_DST_BIT =
0x00000800ULL;
static const VkFormatFeatureFlagBits2 VK_FORMAT_FEATURE_2_BLIT_DST_BIT_KHR =
0x00000800ULL;
static const VkFormatFeatureFlagBits2
VK_FORMAT_FEATURE_2_SAMPLED_IMAGE_FILTER_LINEAR_BIT = 0x00001000ULL;
static const VkFormatFeatureFlagBits2
VK_FORMAT_FEATURE_2_SAMPLED_IMAGE_FILTER_LINEAR_BIT_KHR = 0x00001000ULL;
static const VkFormatFeatureFlagBits2
VK_FORMAT_FEATURE_2_SAMPLED_IMAGE_FILTER_CUBIC_BIT = 0x00002000ULL;
static const VkFormatFeatureFlagBits2
VK_FORMAT_FEATURE_2_SAMPLED_IMAGE_FILTER_CUBIC_BIT_EXT = 0x00002000ULL;
static const VkFormatFeatureFlagBits2 VK_FORMAT_FEATURE_2_TRANSFER_SRC_BIT =
0x00004000ULL;
static const VkFormatFeatureFlagBits2 VK_FORMAT_FEATURE_2_TRANSFER_SRC_BIT_KHR =
0x00004000ULL;
static const VkFormatFeatureFlagBits2 VK_FORMAT_FEATURE_2_TRANSFER_DST_BIT =
0x00008000ULL;
static const VkFormatFeatureFlagBits2 VK_FORMAT_FEATURE_2_TRANSFER_DST_BIT_KHR =
```

```
0x00008000ULL;
static const VkFormatFeatureFlagBits2
VK_FORMAT_FEATURE_2_SAMPLED_IMAGE_FILTER_MINMAX_BIT = 0x00010000ULL;
static const VkFormatFeatureFlagBits2
VK_FORMAT_FEATURE_2_SAMPLED_IMAGE_FILTER_MINMAX_BIT_KHR = 0x00010000ULL;
static const VkFormatFeatureFlagBits2 VK_FORMAT_FEATURE_2_MIDPOINT_CHROMA_SAMPLES_BIT
= 0x00020000ULL;
static const VkFormatFeatureFlagBits2
VK_FORMAT_FEATURE_2_MIDPOINT_CHROMA_SAMPLES_BIT_KHR = 0x00020000ULL;
static const VkFormatFeatureFlagBits2
VK_FORMAT_FEATURE_2_SAMPLED_IMAGE_YCBCR_CONVERSION_LINEAR_FILTER_BIT = 0x00040000ULL;
static const VkFormatFeatureFlagBits2
VK_FORMAT_FEATURE_2_SAMPLED_IMAGE_YCBCR_CONVERSION_LINEAR_FILTER_BIT_KHR =
0x00040000ULL;
static const VkFormatFeatureFlagBits2
VK_FORMAT_FEATURE_2_SAMPLED_IMAGE_YCBCR_CONVERSION_SEPARATE_RECONSTRUCTION_FILTER_BIT
= 0x00080000ULL;
static const VkFormatFeatureFlagBits2
VK_FORMAT_FEATURE_2_SAMPLED_IMAGE_YCBCR_CONVERSION_SEPARATE_RECONSTRUCTION_FILTER_BIT_
KHR = 0x00080000ULL;
static const VkFormatFeatureFlagBits2
VK_FORMAT_FEATURE_2_SAMPLED_IMAGE_YCBCR_CONVERSION_CHROMA_RECONSTRUCTION_EXPLICIT_BIT
= 0x00100000ULL;
static const VkFormatFeatureFlagBits2
VK_FORMAT_FEATURE_2_SAMPLED_IMAGE_YCBCR_CONVERSION_CHROMA_RECONSTRUCTION_EXPLICIT_BIT_
KHR = 0x00100000ULL;
static const VkFormatFeatureFlagBits2
VK_FORMAT_FEATURE_2_SAMPLED_IMAGE_YCBCR_CONVERSION_CHROMA_RECONSTRUCTION_EXPLICIT_FORC
EABLE_BIT = 0x00200000ULL;
static const VkFormatFeatureFlagBits2
VK_FORMAT_FEATURE_2_SAMPLED_IMAGE_YCBCR_CONVERSION_CHROMA_RECONSTRUCTION_EXPLICIT_FORC
EABLE_BIT_KHR = 0x00200000ULL;
static const VkFormatFeatureFlagBits2 VK_FORMAT_FEATURE_2_DISJOINT_BIT =
0x00400000ULL;
static const VkFormatFeatureFlagBits2 VK_FORMAT_FEATURE_2_DISJOINT_BIT_KHR =
0x00400000ULL;
static const VkFormatFeatureFlagBits2 VK_FORMAT_FEATURE_2_COSITED_CHROMA_SAMPLES_BIT =
0x00800000ULL;
static const VkFormatFeatureFlagBits2
VK_FORMAT_FEATURE_2_COSITED_CHROMA_SAMPLES_BIT_KHR = 0x00800000ULL;
static const VkFormatFeatureFlagBits2
VK_FORMAT_FEATURE_2_STORAGE_READ_WITHOUT_FORMAT_BIT = 0x80000000ULL;
static const VkFormatFeatureFlagBits2
VK_FORMAT_FEATURE_2_STORAGE_READ_WITHOUT_FORMAT_BIT_KHR = 0x80000000ULL;
static const VkFormatFeatureFlagBits2
VK_FORMAT_FEATURE_2_STORAGE_WRITE_WITHOUT_FORMAT_BIT = 0x100000000ULL;
static const VkFormatFeatureFlagBits2
VK_FORMAT_FEATURE_2_STORAGE_WRITE_WITHOUT_FORMAT_BIT_KHR = 0x100000000ULL;
static const VkFormatFeatureFlagBits2
VK_FORMAT_FEATURE_2_SAMPLED_IMAGE_DEPTH_COMPARISON_BIT = 0x200000000ULL;
static const VkFormatFeatureFlagBits2
```

```
VK_FORMAT_FEATURE_2_SAMPLED_IMAGE_DEPTH_COMPARISON_BIT_KHR = 0x200000000ULL;
// Provided by VK_KHR_video_decode_queue with VK_KHR_format_feature_flags2 or
VK_VERSION_1_3
static const VkFormatFeatureFlagBits2 VK_FORMAT_FEATURE_2_VIDEO_DECODE_OUTPUT_BIT_KHR
= 0x02000000ULL;
// Provided by VK_KHR_video_decode_queue with VK_KHR_format_feature_flags2 or
VK_VERSION_1_3
static const VkFormatFeatureFlagBits2 VK_FORMAT_FEATURE_2_VIDEO_DECODE_DPB_BIT_KHR =
0x04000000ULL;
// Provided by VK_KHR_acceleration_structure with VK_KHR_format_feature_flags2 or
VK_VERSION_1_3
static const VkFormatFeatureFlagBits2
VK_FORMAT_FEATURE_2_ACCELERATION_STRUCTURE_VERTEX_BUFFER_BIT_KHR = 0x20000000ULL;
// Provided by VK_EXT_fragment_density_map with VK_KHR_format_feature_flags2 or
VK_VERSION_1_3
static const VkFormatFeatureFlagBits2 VK_FORMAT_FEATURE_2_FRAGMENT_DENSITY_MAP_BIT_EXT
= 0x01000000ULL;
// Provided by VK_KHR_fragment_shading_rate with VK_KHR_format_feature_flags2 or
VK_VERSION_1_3
static const VkFormatFeatureFlagBits2
VK_FORMAT_FEATURE_2_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR = 0x40000000ULL;
// Provided by VK_EXT_host_image_copy
static const VkFormatFeatureFlagBits2 VK_FORMAT_FEATURE_2_HOST_IMAGE_TRANSFER_BIT_EXT
= 0x400000000000ULL;
// Provided by VK_KHR_video_encode_queue with VK_KHR_format_feature_flags2 or
VK_VERSION_1_3
static const VkFormatFeatureFlagBits2 VK_FORMAT_FEATURE_2_VIDEO_ENCODE_INPUT_BIT_KHR =
0x08000000ULL;
// Provided by VK_KHR_video_encode_queue with VK_KHR_format_feature_flags2 or
VK_VERSION_1_3
static const VkFormatFeatureFlagBits2 VK_FORMAT_FEATURE_2_VIDEO_ENCODE_DPB_BIT_KHR =
0x10000000ULL;
// Provided by VK_NV_linear_color_attachment with VK_KHR_format_feature_flags2 or
VK_VERSION_1_3
static const VkFormatFeatureFlagBits2
VK_FORMAT_FEATURE_2_LINEAR_COLOR_ATTACHMENT_BIT_NV = 0x4000000000ULL;
// Provided by VK_QCOM_image_processing with VK_KHR_format_feature_flags2 or
VK_VERSION_1_3
static const VkFormatFeatureFlagBits2 VK_FORMAT_FEATURE_2_WEIGHT_IMAGE_BIT_QCOM =
0x400000000ULL;
// Provided by VK_QCOM_image_processing with VK_KHR_format_feature_flags2 or
VK_VERSION_1_3
static const VkFormatFeatureFlagBits2
VK_FORMAT_FEATURE_2_WEIGHT_SAMPLED_IMAGE_BIT_QCOM = 0x800000000ULL;
// Provided by VK_QCOM_image_processing with VK_KHR_format_feature_flags2 or
VK_VERSION_1_3
static const VkFormatFeatureFlagBits2 VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM =
0x1000000000ULL;
// Provided by VK_QCOM_image_processing with VK_KHR_format_feature_flags2 or
VK_VERSION_1_3
static const VkFormatFeatureFlagBits2 VK_FORMAT_FEATURE_2_BOX_FILTER_SAMPLED_BIT_QCOM
```

```
= 0x2000000000ULL;
// Provided by VK_NV_optical_flow
static const VkFormatFeatureFlagBits2 VK_FORMAT_FEATURE_2_OPTICAL_FLOW_IMAGE_BIT_NV =
0x10000000000ULL;
// Provided by VK_NV_optical_flow
static const VkFormatFeatureFlagBits2 VK_FORMAT_FEATURE_2_OPTICAL_FLOW_VECTOR_BIT_NV =
0x20000000000ULL;
// Provided by VK_NV_optical_flow
static const VkFormatFeatureFlagBits2 VK_FORMAT_FEATURE_2_OPTICAL_FLOW_COST_BIT_NV =
0x40000000000ULL;
```

or the equivalent

```
// Provided by VK_KHR_format_feature_flags2
typedef VkFormatFeatureFlagBits2 VkFormatFeatureFlagBits2KHR;
```

The following bits **may** be set in `linearTilingFeatures` and `optimalTilingFeatures`, specifying that the features are supported by images or image views or sampler $Y'C_BC_R$ conversion objects created with the queried vkGetPhysicalDeviceFormatProperties2::`format`:

- `VK_FORMAT_FEATURE_2_SAMPLED_IMAGE_BIT` specifies that an image view **can** be sampled from.
- `VK_FORMAT_FEATURE_2_STORAGE_IMAGE_BIT` specifies that an image view **can** be used as a storage image.
- `VK_FORMAT_FEATURE_2_STORAGE_IMAGE_ATOMIC_BIT` specifies that an image view **can** be used as storage image that supports atomic operations.
- `VK_FORMAT_FEATURE_2_COLOR_ATTACHMENT_BIT` specifies that an image view **can** be used as a framebuffer color attachment and as an input attachment.
- `VK_FORMAT_FEATURE_2_COLOR_ATTACHMENT_BLEND_BIT` specifies that an image view **can** be used as a framebuffer color attachment that supports blending.
- `VK_FORMAT_FEATURE_2_DEPTH_STENCIL_ATTACHMENT_BIT` specifies that an image view **can** be used as a framebuffer depth/stencil attachment and as an input attachment.
- `VK_FORMAT_FEATURE_2_BLIT_SRC_BIT` specifies that an image **can** be used as the `srcImage` for vkCmdBlitImage2KHR and vkCmdBlitImage.
- `VK_FORMAT_FEATURE_2_BLIT_DST_BIT` specifies that an image **can** be used as the `dstImage` for vkCmdBlitImage2KHR and vkCmdBlitImage.
- `VK_FORMAT_FEATURE_2_SAMPLED_IMAGE_FILTER_LINEAR_BIT` specifies that if `VK_FORMAT_FEATURE_2_SAMPLED_IMAGE_BIT` is also set, an image view **can** be used with a sampler that has either of `magFilter` or `minFilter` set to `VK_FILTER_LINEAR`, or `mipmapMode` set to `VK_SAMPLER_MIPMAP_MODE_LINEAR`. If `VK_FORMAT_FEATURE_2_BLIT_SRC_BIT` is also set, an image can be used as the `srcImage` for vkCmdBlitImage2KHR and `vkCmdBlitImage` with a `filter` of `VK_FILTER_LINEAR`. This bit **must** only be exposed for formats that also support the `VK_FORMAT_FEATURE_2_SAMPLED_IMAGE_BIT` or `VK_FORMAT_FEATURE_2_BLIT_SRC_BIT`.

  If the format being queried is a depth/stencil format, this bit only specifies that the depth aspect

(not the stencil aspect) of an image of this format supports linear filtering. Where depth comparison is supported it **may** be linear filtered whether this bit is present or not, but where this bit is not present the filtered value **may** be computed in an implementation-dependent manner which differs from the normal rules of linear filtering. The resulting value **must** be in the range [0,1] and **should** be proportional to, or a weighted average of, the number of comparison passes or failures.

- `VK_FORMAT_FEATURE_2_TRANSFER_SRC_BIT` specifies that an image **can** be used as a source image for copy commands.
- `VK_FORMAT_FEATURE_2_TRANSFER_DST_BIT` specifies that an image **can** be used as a destination image for copy commands and clear commands.
- `VK_FORMAT_FEATURE_2_SAMPLED_IMAGE_FILTER_MINMAX_BIT` specifies `VkImage` **can** be used as a sampled image with a min or max VkSamplerReductionMode. This bit **must** only be exposed for formats that also support the `VK_FORMAT_FEATURE_2_SAMPLED_IMAGE_BIT`.
- `VK_FORMAT_FEATURE_2_SAMPLED_IMAGE_FILTER_CUBIC_BIT` specifies that `VkImage` **can** be used with a sampler that has either of `magFilter` or `minFilter` set to `VK_FILTER_CUBIC_EXT`, or be the source image for a blit with `filter` set to `VK_FILTER_CUBIC_EXT`. This bit **must** only be exposed for formats that also support the `VK_FORMAT_FEATURE_2_SAMPLED_IMAGE_BIT`. If the format being queried is a depth/stencil format, this only specifies that the depth aspect is cubic filterable.
- `VK_FORMAT_FEATURE_2_MIDPOINT_CHROMA_SAMPLES_BIT` specifies that an application **can** define a sampler $Y'C_BC_R$ conversion using this format as a source, and that an image of this format **can** be used with a VkSamplerYcbcrConversionCreateInfo `xChromaOffset` and/or `yChromaOffset` of `VK_CHROMA_LOCATION_MIDPOINT`. Otherwise both `xChromaOffset` and `yChromaOffset` **must** be `VK_CHROMA_LOCATION_COSITED_EVEN`. If a format does not incorporate chroma downsampling (it is not a “422” or “420” format) but the implementation supports sampler $Y'C_BC_R$ conversion for this format, the implementation **must** set `VK_FORMAT_FEATURE_2_MIDPOINT_CHROMA_SAMPLES_BIT`.
- `VK_FORMAT_FEATURE_2_COSITED_CHROMA_SAMPLES_BIT` specifies that an application **can** define a sampler $Y'C_BC_R$ conversion using this format as a source, and that an image of this format **can** be used with a VkSamplerYcbcrConversionCreateInfo `xChromaOffset` and/or `yChromaOffset` of `VK_CHROMA_LOCATION_COSITED_EVEN`. Otherwise both `xChromaOffset` and `yChromaOffset` **must** be `VK_CHROMA_LOCATION_MIDPOINT`. If neither `VK_FORMAT_FEATURE_2_COSITED_CHROMA_SAMPLES_BIT` nor `VK_FORMAT_FEATURE_2_MIDPOINT_CHROMA_SAMPLES_BIT` is set, the application **must** not define a sampler $Y'C_BC_R$ conversion using this format as a source.
- `VK_FORMAT_FEATURE_2_SAMPLED_IMAGE_YCBCR_CONVERSION_LINEAR_FILTER_BIT` specifies that an application **can** define a sampler $Y'C_BC_R$ conversion using this format as a source with `chromaFilter` set to `VK_FILTER_LINEAR`.
- `VK_FORMAT_FEATURE_2_SAMPLED_IMAGE_YCBCR_CONVERSION_SEPARATE_RECONSTRUCTION_FILTER_BIT` specifies that the format can have different chroma, min, and mag filters.
- `VK_FORMAT_FEATURE_2_SAMPLED_IMAGE_YCBCR_CONVERSION_CHROMA_RECONSTRUCTION_EXPLICIT_BIT` specifies that reconstruction is explicit, as described in Chroma Reconstruction. If this bit is not present, reconstruction is implicit by default.
- `VK_FORMAT_FEATURE_2_SAMPLED_IMAGE_YCBCR_CONVERSION_CHROMA_RECONSTRUCTION_EXPLICIT_FORCEABLE_BIT` specifies that reconstruction **can** be forcibly made explicit by setting VkSamplerYcbcrConversionCreateInfo::`forceExplicitReconstruction` to `VK_TRUE`. If the format

being queried supports VK_FORMAT_FEATURE_2_SAMPLED_IMAGE_YCBCR_CONVERSION_CHROMA_RECONSTRUCTION_EXPLICIT_BIT it **must** also support VK_FORMAT_FEATURE_2_SAMPLED_IMAGE_YCBCR_CONVERSION_CHROMA_RECONSTRUCTION_EXPLICIT_FORCEABLE_BIT.

- VK_FORMAT_FEATURE_2_DISJOINT_BIT specifies that a multi-planar image **can** have the VK_IMAGE_CREATE_DISJOINT_BIT set during image creation. An implementation **must** not set VK_FORMAT_FEATURE_2_DISJOINT_BIT for *single-plane formats*.
- VK_FORMAT_FEATURE_2_FRAGMENT_DENSITY_MAP_BIT_EXT specifies that an image view **can** be used as a fragment density map attachment.
- VK_FORMAT_FEATURE_2_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR specifies that an image view **can** be used as a fragment shading rate attachment. An implementation **must** not set this feature for formats with a numeric format other than UINT, or set it as a buffer feature.
- VK_FORMAT_FEATURE_2_VIDEO_DECODE_OUTPUT_BIT_KHR specifies that an image view with this format **can** be used as a decode output picture in video decode operations.
- VK_FORMAT_FEATURE_2_VIDEO_DECODE_DPB_BIT_KHR specifies that an image view with this format **can** be used as an output reconstructed picture or an input reference picture in video decode operations.
- VK_FORMAT_FEATURE_2_VIDEO_ENCODE_INPUT_BIT_KHR specifies that an image view with this format **can** be used as an encode input picture in video encode operations.
- VK_FORMAT_FEATURE_2_VIDEO_ENCODE_DPB_BIT_KHR specifies that an image view with this format **can** be used as an output reconstructed picture or an input reference picture in video encode operations.

| | |
|---|---|
| **NOTE** | Specific video profiles **may** have additional restrictions on the format and other image creation parameters corresponding to image views used by video coding operations that **can** be enumerated using the vkGetPhysicalDeviceVideoFormatPropertiesKHR command. |

- VK_FORMAT_FEATURE_2_STORAGE_READ_WITHOUT_FORMAT_BIT specifies that image views or buffer views created with this format **can** be used as storage images or storage texel buffers respectively for read operations without specifying a format.
- VK_FORMAT_FEATURE_2_STORAGE_WRITE_WITHOUT_FORMAT_BIT specifies that image views or buffer views created with this format **can** be used as storage images or storage texel buffers respectively for write operations without specifying a format.
- VK_FORMAT_FEATURE_2_SAMPLED_IMAGE_DEPTH_COMPARISON_BIT specifies that image views created with this format **can** be used for depth comparison performed by OpImage*Dref* instructions.
- VK_FORMAT_FEATURE_2_LINEAR_COLOR_ATTACHMENT_BIT_NV specifies that the format is supported as a renderable Linear Color Attachment. This bit will be set for renderable color formats in the linearTilingFeatures. This **must** not be set in the optimalTilingFeatures or bufferFeatures members.
- VK_FORMAT_FEATURE_2_WEIGHT_IMAGE_BIT_QCOM specifies that image views created with this format **can** be used as the weight image input to weight image sampling operations.

- VK_FORMAT_FEATURE_2_WEIGHT_SAMPLED_IMAGE_BIT_QCOM specifies that image views created with this format **can** be sampled in weight image sampling operations.
- VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM specifies that image views created with this format **can** be used in block matching operations.
- VK_FORMAT_FEATURE_2_BOX_FILTER_SAMPLED_BIT_QCOM specifies that image views created with this format **can** be sampled in box filter sampling operations.
- VK_FORMAT_FEATURE_2_HOST_IMAGE_TRANSFER_BIT_EXT specifies that an image **can** be created with VK_IMAGE_USAGE_HOST_TRANSFER_BIT_EXT.

The following bits **may** be set in bufferFeatures, specifying that the features are supported by buffers or buffer views created with the queried vkGetPhysicalDeviceFormatProperties2::format:

- VK_FORMAT_FEATURE_2_UNIFORM_TEXEL_BUFFER_BIT specifies that the format **can** be used to create a buffer view that **can** be bound to a VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER descriptor.
- VK_FORMAT_FEATURE_2_STORAGE_TEXEL_BUFFER_BIT specifies that the format **can** be used to create a buffer view that **can** be bound to a VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER descriptor.
- VK_FORMAT_FEATURE_2_STORAGE_TEXEL_BUFFER_ATOMIC_BIT specifies that atomic operations are supported on VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER with this format.
- VK_FORMAT_FEATURE_2_VERTEX_BUFFER_BIT specifies that the format **can** be used as a vertex attribute format (VkVertexInputAttributeDescription::format).
- VK_FORMAT_FEATURE_2_ACCELERATION_STRUCTURE_VERTEX_BUFFER_BIT_KHR specifies that the format **can** be used as the vertex format when creating an acceleration structure (VkAccelerationStructureGeometryTrianglesDataKHR::vertexFormat). This format **can** also be used as the vertex format in host memory when doing host acceleration structure builds.
- VK_FORMAT_FEATURE_2_STORAGE_READ_WITHOUT_FORMAT_BIT specifies that buffer views created with this format **can** be used as storage texel buffers for read operations without specifying a format.
- VK_FORMAT_FEATURE_2_STORAGE_WRITE_WITHOUT_FORMAT_BIT specifies that buffer views created with this format **can** be used as storage texel buffers for write operations without specifying a format.
- VK_FORMAT_FEATURE_2_OPTICAL_FLOW_IMAGE_BIT_NV specifies that an image view with this format **can** be used as an input or reference to optical flow operations
- VK_FORMAT_FEATURE_2_OPTICAL_FLOW_VECTOR_BIT_NV specifies that an image view with this format **can** be used as a flow vector map (either as hint, output or global flow) for optical flow operations
- VK_FORMAT_FEATURE_2_OPTICAL_FLOW_COST_BIT_NV specifies that an image view with this format **can** be used as an output cost map for optical flow operations

```
typedef VkFlags64 VkFormatFeatureFlags2;
```

or the equivalent

```
// Provided by VK_KHR_format_feature_flags2
typedef VkFormatFeatureFlags2 VkFormatFeatureFlags2KHR;
```

VkFormatFeatureFlags2 is a bitmask type for setting a mask of zero or more VkFormatFeatureFlagBits2.

To query the performance characteristics of a subpass resolve operation for an attachment with a VkFormat, add a VkSubpassResolvePerformanceQueryEXT structure to the pNext chain of VkFormatProperties2.

The VkSubpassResolvePerformanceQueryEXT structure is defined as:

```
// Provided by VK_EXT_multisampled_render_to_single_sampled
typedef struct VkSubpassResolvePerformanceQueryEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           optimal;
} VkSubpassResolvePerformanceQueryEXT;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- optimal specifies that a subpass resolve operation is optimally performed.

If optimal is VK_FALSE for a VkFormat, using a subpass resolve operation on a multisampled attachment with this format can incur additional costs, including additional memory bandwidth usage and a higher memory footprint. If an attachment with such a format is used in a multisampled-render-to-single-sampled subpass, the additional memory and memory bandwidth usage can nullify the benefits of using the VK_EXT_multisampled_render_to_single_sampled extension.

**Valid Usage (Implicit)**

- VUID-VkSubpassResolvePerformanceQueryEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_SUBPASS_RESOLVE_PERFORMANCE_QUERY_EXT

### 48.2.1. Potential Format Features

Some valid usage conditions depend on the format features supported by a VkImage whose VkImageTiling is unknown. In such cases the exact VkFormatFeatureFlagBits supported by the VkImage cannot be determined, so the valid usage conditions are expressed in terms of the *potential format features* of the VkImage format.

The *potential format features* of a VkFormat are defined as follows:

- The union of VkFormatFeatureFlagBits and VkFormatFeatureFlagBits2, supported when the VkImageTiling is VK_IMAGE_TILING_OPTIMAL , VK_IMAGE_TILING_DRM_FORMAT_MODIFIER_EXT, or VK_IMAGE_TILING_LINEAR if VkFormat is not VK_FORMAT_UNDEFINED

- VkAndroidHardwareBufferFormatPropertiesANDROID::formatFeatures and VkAndroidHardwareBufferFormatProperties2ANDROID::formatFeatures of a valid external format if VkFormat is VK_FORMAT_UNDEFINED
- VkScreenBufferFormatPropertiesQNX::formatFeatures of a valid external format if VkFormat is VK_FORMAT_UNDEFINED

## 48.3. Required Format Support

Implementations **must** support at least the following set of features on the listed formats. For images, these features **must** be supported for every VkImageType (including arrayed and cube variants) unless otherwise noted. These features are supported on existing formats without needing to advertise an extension or needing to explicitly enable them. Support for additional functionality beyond the requirements listed here is queried using the vkGetPhysicalDeviceFormatProperties command.

| | |
|---|---|
| **NOTE** | Unless otherwise excluded below, the required formats are supported for all VkImageCreateFlags values as long as those flag values are otherwise allowed. |

The following tables show which feature bits **must** be supported for each format. Formats that are required to support VK_FORMAT_FEATURE_SAMPLED_IMAGE_BIT **must** also support VK_FORMAT_FEATURE_TRANSFER_SRC_BIT and VK_FORMAT_FEATURE_TRANSFER_DST_BIT.

*Table 74. Key for format feature tables*

| | |
|---|---|
| □ | This feature **must** be supported on the named format |
| † | This feature **must** be supported on at least some of the named formats, with more information in the table where the symbol appears |
| ‡ | This feature **must** be supported with some caveats or preconditions, with more information in the table where the symbol appears |
| § | This feature **must** be supported with some caveats or preconditions, with more information in the table where the symbol appears |

*Table 75. Feature bits in optimalTilingFeatures*

| |
|---|
| VK_FORMAT_FEATURE_TRANSFER_SRC_BIT |
| VK_FORMAT_FEATURE_TRANSFER_DST_BIT |
| VK_FORMAT_FEATURE_SAMPLED_IMAGE_BIT |
| VK_FORMAT_FEATURE_BLIT_SRC_BIT |
| VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT |
| VK_FORMAT_FEATURE_STORAGE_IMAGE_BIT |
| VK_FORMAT_FEATURE_STORAGE_IMAGE_ATOMIC_BIT |
| VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BIT |
| VK_FORMAT_FEATURE_BLIT_DST_BIT |

| |
|---|
| VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT |
| VK_FORMAT_FEATURE_DEPTH_STENCIL_ATTACHMENT_BIT |
| VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT |

*Table 76. Feature bits in* `bufferFeatures`

| |
|---|
| VK_FORMAT_FEATURE_VERTEX_BUFFER_BIT |
| VK_FORMAT_FEATURE_UNIFORM_TEXEL_BUFFER_BIT |
| VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_BIT |
| VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_ATOMIC_BIT |

*Table 77. Mandatory format support: sub-byte components*

| Format | VK_FORMAT_FEATURE_SAMPLED_IMAGE_BIT | VK_FORMAT_FEATURE_BLIT_SRC_BIT | VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT | VK_FORMAT_FEATURE_STORAGE_IMAGE_BIT | VK_FORMAT_FEATURE_STORAGE_IMAGE_ATOMIC_BIT | VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BIT | VK_FORMAT_FEATURE_BLIT_DST_BIT | VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT | VK_FORMAT_FEATURE_DEPTH_STENCIL_ATTACHMENT_BIT | VK_FORMAT_FEATURE_VERTEX_BUFFER_BIT | VK_FORMAT_FEATURE_UNIFORM_TEXEL_BUFFER_BIT | VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_BIT | VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_ATOMIC_BIT |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| VK_FORMAT_UNDEFINED | | | | | | | | | | | | | |
| VK_FORMAT_R4G4_UNORM_PACK8 | | | | | | | | | | | | | |
| VK_FORMAT_R4G4B4A4_UNORM_PACK16 | | | | | | | | | | | | | |
| VK_FORMAT_B4G4R4A4_UNORM_PACK16 | ▯ | ▯ | ▯ | | | | | | | | | | |
| VK_FORMAT_R5G6B5_UNORM_PACK16 | ▯ | ▯ | ▯ | | | ▯ | ▯ | ▯ | | | | | |
| VK_FORMAT_B5G6R5_UNORM_PACK16 | | | | | | | | | | | | | |
| VK_FORMAT_R5G5B5A1_UNORM_PACK16 | | | | | | | | | | | | | |
| VK_FORMAT_B5G5R5A1_UNORM_PACK16 | | | | | | | | | | | | | |
| VK_FORMAT_A1R5G5B5_UNORM_PACK16 | ▯ | ▯ | ▯ | | | ▯ | ▯ | ▯ | | | | | |
| VK_FORMAT_A1B5G5R5_UNORM_PACK16_KHR | | | | | | | | | | | | | |
| VK_FORMAT_A4R4G4B4_UNORM_PACK16 | † | † | † | | | | | | | | | | |
| VK_FORMAT_A4B4G4R4_UNORM_PACK16 | ‡ | ‡ | ‡ | | | | | | | | | | |

Format features marked † **must** be supported for `optimalTilingFeatures` if the `VkPhysicalDevice` supports the VkPhysicalDevice4444FormatsFeaturesEXT::`formatA4R4G4B4` feature.

Format features marked ‡ **must** be supported for `optimalTilingFeatures` if the `VkPhysicalDevice` supports the VkPhysicalDevice4444FormatsFeaturesEXT::`formatA4B4G4R4` feature.

*Table 78. Mandatory format support: 1-3 byte-sized components*

| Format | VK_FORMAT_FEATURE_SAMPLED_IMAGE_BIT | VK_FORMAT_FEATURE_BLIT_SRC_BIT | VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT | VK_FORMAT_FEATURE_STORAGE_IMAGE_BIT | VK_FORMAT_FEATURE_STORAGE_IMAGE_ATOMIC_BIT | VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BIT | VK_FORMAT_FEATURE_BLIT_DST_BIT | VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT | VK_FORMAT_FEATURE_DEPTH_STENCIL_ATTACHMENT_BIT | VK_FORMAT_FEATURE_VERTEX_BUFFER_BIT | VK_FORMAT_FEATURE_UNIFORM_TEXEL_BUFFER_BIT | VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_BIT | VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_ATOMIC_BIT |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| VK_FORMAT_R8_UNORM | ◻ | ◻ | ◻ | ‡ | | ◻ | ◻ | ◻ | | ◻ | ◻ | | |
| VK_FORMAT_R8_SNORM | ◻ | ◻ | ◻ | ‡ | | | | | | ◻ | ◻ | | |
| VK_FORMAT_R8_USCALED | | | | | | | | | | | | | |
| VK_FORMAT_R8_SSCALED | | | | | | | | | | | | | |
| VK_FORMAT_R8_UINT | ◻ | ◻ | | ‡ | | ◻ | ◻ | | | ◻ | ◻ | | |
| VK_FORMAT_R8_SINT | ◻ | ◻ | | ‡ | | ◻ | ◻ | | | ◻ | ◻ | | |
| VK_FORMAT_R8_SRGB | | | | | | | | | | | | | |
| VK_FORMAT_R8G8_UNORM | ◻ | ◻ | ◻ | ‡ | | ◻ | ◻ | ◻ | | ◻ | ◻ | | |
| VK_FORMAT_R8G8_SNORM | ◻ | ◻ | ◻ | ‡ | | | | | | ◻ | ◻ | | |
| VK_FORMAT_R8G8_USCALED | | | | | | | | | | | | | |
| VK_FORMAT_R8G8_SSCALED | | | | | | | | | | | | | |
| VK_FORMAT_R8G8_UINT | ◻ | ◻ | | ‡ | | ◻ | ◻ | | | ◻ | ◻ | | |
| VK_FORMAT_R8G8_SINT | ◻ | ◻ | | ‡ | | ◻ | ◻ | | | ◻ | ◻ | | |
| VK_FORMAT_R8G8_SRGB | | | | | | | | | | | | | |
| VK_FORMAT_R8G8B8_UNORM | | | | | | | | | | | | | |
| VK_FORMAT_R8G8B8_SNORM | | | | | | | | | | | | | |
| VK_FORMAT_R8G8B8_USCALED | | | | | | | | | | | | | |
| VK_FORMAT_R8G8B8_SSCALED | | | | | | | | | | | | | |
| VK_FORMAT_R8G8B8_UINT | | | | | | | | | | | | | |
| VK_FORMAT_R8G8B8_SINT | | | | | | | | | | | | | |
| VK_FORMAT_R8G8B8_SRGB | | | | | | | | | | | | | |
| VK_FORMAT_B8G8R8_UNORM | | | | | | | | | | | | | |
| VK_FORMAT_B8G8R8_SNORM | | | | | | | | | | | | | |

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| VK_FORMAT_B8G8R8_USCALED | | | | | | | | | | | | | |
| VK_FORMAT_B8G8R8_SSCALED | | | | | | | | | | | | | |
| VK_FORMAT_B8G8R8_UINT | | | | | | | | | | | | | |
| VK_FORMAT_B8G8R8_SINT | | | | | | | | | | | | | |
| VK_FORMAT_B8G8R8_SRGB | | | | | | | | | | | | | |
| VK_FORMAT_A8_UNORM_KHR | | | | | | | | | | | | | |
| Format features marked with ‡ **must** be supported for optimalTilingFeatures if the VkPhysicalDevice supports the shaderStorageImageExtendedFormats feature. | | | | | | | | | | | | | |

*Table 79. Mandatory format support: 4 byte-sized components*

| Format | VK_FORMAT_FEATURE_SAMPLED_IMAGE_BIT | VK_FORMAT_FEATURE_BLIT_SRC_BIT | VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT | VK_FORMAT_FEATURE_STORAGE_IMAGE_BIT | VK_FORMAT_FEATURE_STORAGE_IMAGE_ATOMIC_BIT | VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BIT | VK_FORMAT_FEATURE_BLIT_DST_BIT | VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT | VK_FORMAT_FEATURE_DEPTH_STENCIL_ATTACHMENT_BIT | VK_FORMAT_FEATURE_VERTEX_BUFFER_BIT | VK_FORMAT_FEATURE_UNIFORM_TEXEL_BUFFER_BIT | VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_BIT | VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_ATOMIC_BIT |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| VK_FORMAT_R8G8B8A8_UNORM | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ | ✓ | | ✓ | ✓ | ✓ | |
| VK_FORMAT_R8G8B8A8_SNORM | ✓ | ✓ | ✓ | ✓ | | | | | | ✓ | ✓ | ✓ | |
| VK_FORMAT_R8G8B8A8_USCALED | | | | | | | | | | | | | |
| VK_FORMAT_R8G8B8A8_SSCALED | | | | | | | | | | | | | |
| VK_FORMAT_R8G8B8A8_UINT | ✓ | ✓ | | ✓ | | ✓ | ✓ | | | ✓ | ✓ | ✓ | |
| VK_FORMAT_R8G8B8A8_SINT | ✓ | ✓ | | ✓ | | ✓ | ✓ | | | ✓ | ✓ | ✓ | |
| VK_FORMAT_R8G8B8A8_SRGB | ✓ | ✓ | ✓ | | | ✓ | ✓ | ✓ | | | | | |
| VK_FORMAT_B8G8R8A8_UNORM | ✓ | ✓ | ✓ | | | ✓ | ✓ | ✓ | | ✓ | ✓ | | |
| VK_FORMAT_B8G8R8A8_SNORM | | | | | | | | | | | | | |
| VK_FORMAT_B8G8R8A8_USCALED | | | | | | | | | | | | | |
| VK_FORMAT_B8G8R8A8_SSCALED | | | | | | | | | | | | | |
| VK_FORMAT_B8G8R8A8_UINT | | | | | | | | | | | | | |
| VK_FORMAT_B8G8R8A8_SINT | | | | | | | | | | | | | |
| VK_FORMAT_B8G8R8A8_SRGB | ✓ | ✓ | ✓ | | | ✓ | ✓ | ✓ | | | | | |
| VK_FORMAT_A8B8G8R8_UNORM_PACK32 | ✓ | ✓ | ✓ | | | ✓ | ✓ | ✓ | | ✓ | ✓ | ✓ | |
| VK_FORMAT_A8B8G8R8_SNORM_PACK32 | ✓ | ✓ | ✓ | | | | | | | ✓ | ✓ | ✓ | |
| VK_FORMAT_A8B8G8R8_USCALED_PACK32 | | | | | | | | | | | | | |
| VK_FORMAT_A8B8G8R8_SSCALED_PACK32 | | | | | | | | | | | | | |
| VK_FORMAT_A8B8G8R8_UINT_PACK32 | ✓ | ✓ | | | | ✓ | ✓ | | | ✓ | ✓ | ✓ | |
| VK_FORMAT_A8B8G8R8_SINT_PACK32 | ✓ | ✓ | | | | ✓ | ✓ | | | ✓ | ✓ | ✓ | |
| VK_FORMAT_A8B8G8R8_SRGB_PACK32 | ✓ | ✓ | ✓ | | | ✓ | ✓ | ✓ | | | | | |

*Table 80. Mandatory format support: 10- and 12-bit components*

| Format | VK_FORMAT_FEATURE_SAMPLED_IMAGE_BIT | VK_FORMAT_FEATURE_BLIT_SRC_BIT | VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT | VK_FORMAT_FEATURE_STORAGE_IMAGE_BIT | VK_FORMAT_FEATURE_STORAGE_IMAGE_ATOMIC_BIT | VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BIT | VK_FORMAT_FEATURE_BLIT_DST_BIT | VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT | VK_FORMAT_FEATURE_DEPTH_STENCIL_ATTACHMENT_BIT | VK_FORMAT_FEATURE_VERTEX_BUFFER_BIT | VK_FORMAT_FEATURE_UNIFORM_TEXEL_BUFFER_BIT | VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_BIT | VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_ATOMIC_BIT |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| VK_FORMAT_A2R10G10B10_UNORM_PACK32 | | | | | | | | | | | | | |
| VK_FORMAT_A2R10G10B10_SNORM_PACK32 | | | | | | | | | | | | | |
| VK_FORMAT_A2R10G10B10_USCALED_PACK32 | | | | | | | | | | | | | |
| VK_FORMAT_A2R10G10B10_SSCALED_PACK32 | | | | | | | | | | | | | |
| VK_FORMAT_A2R10G10B10_UINT_PACK32 | | | | | | | | | | | | | |
| VK_FORMAT_A2R10G10B10_SINT_PACK32 | | | | | | | | | | | | | |
| VK_FORMAT_A2B10G10R10_UNORM_PACK32 | ✓ | ✓ | ✓ | ‡ | | ✓ | ✓ | ✓ | | ✓ | ✓ | | |
| VK_FORMAT_A2B10G10R10_SNORM_PACK32 | | | | | | | | | | | | | |
| VK_FORMAT_A2B10G10R10_USCALED_PACK32 | | | | | | | | | | | | | |
| VK_FORMAT_A2B10G10R10_SSCALED_PACK32 | | | | | | | | | | | | | |
| VK_FORMAT_A2B10G10R10_UINT_PACK32 | ✓ | ✓ | | ‡ | | ✓ | ✓ | | | | ✓ | | |
| VK_FORMAT_A2B10G10R10_SINT_PACK32 | | | | | | | | | | | | | |
| VK_FORMAT_R10X6_UNORM_PACK16 | | | | | | | | | | | | | |
| VK_FORMAT_R10X6G10X6_UNORM_2PACK16 | | | | | | | | | | | | | |
| VK_FORMAT_R12X4_UNORM_PACK16 | | | | | | | | | | | | | |
| VK_FORMAT_R12X4G12X4_UNORM_2PACK16 | | | | | | | | | | | | | |

Format features marked with ‡ **must** be supported for `optimalTilingFeatures` if the `VkPhysicalDevice` supports the `shaderStorageImageExtendedFormats` feature.

*Table 81. Mandatory format support: 16-bit components*

| Format | VK_FORMAT_FEATURE_SAMPLED_IMAGE_BIT | VK_FORMAT_FEATURE_BLIT_SRC_BIT | VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT | VK_FORMAT_FEATURE_STORAGE_IMAGE_BIT | VK_FORMAT_FEATURE_STORAGE_IMAGE_ATOMIC_BIT | VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BIT | VK_FORMAT_FEATURE_BLIT_DST_BIT | VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT | VK_FORMAT_FEATURE_DEPTH_STENCIL_ATTACHMENT_BIT | VK_FORMAT_FEATURE_VERTEX_BUFFER_BIT | VK_FORMAT_FEATURE_UNIFORM_TEXEL_BUFFER_BIT | VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_BIT | VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_ATOMIC_BIT |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| VK_FORMAT_R16_UNORM | | | | ‡ | | | | | | ▯ | | | |
| VK_FORMAT_R16_SNORM | | | | ‡ | | | | | | ▯ | | | |
| VK_FORMAT_R16_USCALED | | | | | | | | | | | | | |
| VK_FORMAT_R16_SSCALED | | | | | | | | | | | | | |
| VK_FORMAT_R16_UINT | ▯ | ▯ | | ‡ | | ▯ | ▯ | | | ▯ | ▯ | | |
| VK_FORMAT_R16_SINT | ▯ | ▯ | | ‡ | | ▯ | ▯ | | | ▯ | ▯ | | |
| VK_FORMAT_R16_SFLOAT | ▯ | ▯ | ▯ | ‡ | | ▯ | ▯ | ▯ | | ▯ | ▯ | | |
| VK_FORMAT_R16G16_UNORM | | | | ‡ | | | | | | ▯ | | | |
| VK_FORMAT_R16G16_SNORM | | | | ‡ | | | | | | ▯ | | | |
| VK_FORMAT_R16G16_USCALED | | | | | | | | | | | | | |
| VK_FORMAT_R16G16_SSCALED | | | | | | | | | | | | | |
| VK_FORMAT_R16G16_UINT | ▯ | ▯ | | ‡ | | ▯ | ▯ | | | ▯ | ▯ | | |
| VK_FORMAT_R16G16_SINT | ▯ | ▯ | | ‡ | | ▯ | ▯ | | | ▯ | ▯ | | |
| VK_FORMAT_R16G16_SFLOAT | ▯ | ▯ | ▯ | ‡ | § | ▯ | ▯ | ▯ | | ▯ | ▯ | § | § |
| VK_FORMAT_R16G16B16_UNORM | | | | | | | | | | | | | |
| VK_FORMAT_R16G16B16_SNORM | | | | | | | | | | | | | |
| VK_FORMAT_R16G16B16_USCALED | | | | | | | | | | | | | |
| VK_FORMAT_R16G16B16_SSCALED | | | | | | | | | | | | | |
| VK_FORMAT_R16G16B16_UINT | | | | | | | | | | | | | |
| VK_FORMAT_R16G16B16_SINT | | | | | | | | | | | | | |
| VK_FORMAT_R16G16B16_SFLOAT | | | | | | | | | | | | | |
| VK_FORMAT_R16G16B16A16_UNORM | | | | ‡ | | | | | | ▯ | | | |
| VK_FORMAT_R16G16B16A16_SNORM | | | | ‡ | | | | | | ▯ | | | |

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| VK_FORMAT_R16G16B16A16_USCALED | | | | | | | | | | | | | |
| VK_FORMAT_R16G16B16A16_SSCALED | | | | | | | | | | | | | |
| VK_FORMAT_R16G16B16A16_UINT | □ | □ | | □ | | □ | □ | | | □ | □ | □ | |
| VK_FORMAT_R16G16B16A16_SINT | □ | □ | | □ | | □ | □ | | | □ | □ | □ | |
| VK_FORMAT_R16G16B16A16_SFLOAT | □ | □ | □ | □ | § | □ | □ | □ | | □ | □ | □ | § |
| Format features marked with ‡ **must** be supported for optimalTilingFeatures if the VkPhysicalDevice supports the shaderStorageImageExtendedFormats feature. | | | | | | | | | | | | | |
| Format features marked with § **must** be supported for optimalTilingFeatures if the VkPhysicalDevice supports the shaderFloat16VectorAtomics feature. | | | | | | | | | | | | | |

*Table 82. Mandatory format support: 32-bit components*

| Format | VK_FORMAT_FEATURE_SAMPLED_IMAGE_BIT | VK_FORMAT_FEATURE_BLIT_SRC_BIT | VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT | VK_FORMAT_FEATURE_STORAGE_IMAGE_BIT | VK_FORMAT_FEATURE_STORAGE_IMAGE_ATOMIC_BIT | VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BIT | VK_FORMAT_FEATURE_BLIT_DST_BIT | VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT | VK_FORMAT_FEATURE_DEPTH_STENCIL_ATTACHMENT_BIT | VK_FORMAT_FEATURE_VERTEX_BUFFER_BIT | VK_FORMAT_FEATURE_UNIFORM_TEXEL_BUFFER_BIT | VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_BIT | VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_ATOMIC_BIT |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| VK_FORMAT_R32_UINT | ✓ | ✓ | | ✓ | ✓ | ✓ | ✓ | | | ✓ | ✓ | ✓ | ✓ |
| VK_FORMAT_R32_SINT | ✓ | ✓ | | ✓ | ✓ | ✓ | ✓ | | | ✓ | ✓ | ✓ | ✓ |
| VK_FORMAT_R32_SFLOAT | ✓ | ✓ | | ✓ | † | ✓ | ✓ | | | ✓ | ✓ | ✓ | |
| VK_FORMAT_R32G32_UINT | ✓ | ✓ | | ✓ | | ✓ | ✓ | | | ✓ | ✓ | ✓ | |
| VK_FORMAT_R32G32_SINT | ✓ | ✓ | | ✓ | | ✓ | ✓ | | | ✓ | ✓ | ✓ | |
| VK_FORMAT_R32G32_SFLOAT | ✓ | ✓ | | ✓ | | ✓ | ✓ | | | ✓ | ✓ | ✓ | |
| VK_FORMAT_R32G32B32_UINT | | | | | | | | | | ✓ | | | |
| VK_FORMAT_R32G32B32_SINT | | | | | | | | | | ✓ | | | |
| VK_FORMAT_R32G32B32_SFLOAT | | | | | | | | | | ✓ | | | |
| VK_FORMAT_R32G32B32A32_UINT | ✓ | ✓ | | ✓ | | ✓ | ✓ | | | ✓ | ✓ | ✓ | |
| VK_FORMAT_R32G32B32A32_SINT | ✓ | ✓ | | ✓ | | ✓ | ✓ | | | ✓ | ✓ | ✓ | |
| VK_FORMAT_R32G32B32A32_SFLOAT | ✓ | ✓ | | ✓ | | ✓ | ✓ | | | ✓ | ✓ | ✓ | |

Format features marked with † **must** be supported for `optimalTilingFeatures` if the `VkPhysicalDevice` supports the `shaderImageFloat32Atomics` or the `shaderImageFloat32AtomicAdd` or the `shaderImageFloat32AtomicMinMax` feature.

*Table 83. Mandatory format support: 64-bit/uneven components*

| Format | VK_FORMAT_FEATURE_SAMPLED_IMAGE_BIT | VK_FORMAT_FEATURE_BLIT_SRC_BIT | VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT | VK_FORMAT_FEATURE_STORAGE_IMAGE_BIT | VK_FORMAT_FEATURE_STORAGE_IMAGE_ATOMIC_BIT | VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BIT | VK_FORMAT_FEATURE_BLIT_DST_BIT | VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT | VK_FORMAT_FEATURE_DEPTH_STENCIL_ATTACHMENT_BIT | VK_FORMAT_FEATURE_VERTEX_BUFFER_BIT | VK_FORMAT_FEATURE_UNIFORM_TEXEL_BUFFER_BIT | VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_BIT | VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_ATOMIC_BIT |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| VK_FORMAT_R64_UINT | | | | † | † | | | | | | | | |
| VK_FORMAT_R64_SINT | | | | † | † | | | | | | | | |
| VK_FORMAT_R64_SFLOAT | | | | | | | | | | | | | |
| VK_FORMAT_R64G64_UINT | | | | | | | | | | | | | |
| VK_FORMAT_R64G64_SINT | | | | | | | | | | | | | |
| VK_FORMAT_R64G64_SFLOAT | | | | | | | | | | | | | |
| VK_FORMAT_R64G64B64_UINT | | | | | | | | | | | | | |
| VK_FORMAT_R64G64B64_SINT | | | | | | | | | | | | | |
| VK_FORMAT_R64G64B64_SFLOAT | | | | | | | | | | | | | |
| VK_FORMAT_R64G64B64A64_UINT | | | | | | | | | | | | | |
| VK_FORMAT_R64G64B64A64_SINT | | | | | | | | | | | | | |
| VK_FORMAT_R64G64B64A64_SFLOAT | | | | | | | | | | | | | |
| VK_FORMAT_B10G11R11_UFLOAT_PACK32 | | | | ‡ | | | | | | | | | |
| VK_FORMAT_E5B9G9R9_UFLOAT_PACK32 | | | | | | | | | | | | | |

Format features marked with ‡ **must** be supported for `optimalTilingFeatures` if the `VkPhysicalDevice` supports the `shaderStorageImageExtendedFormats` feature.

If the `shaderImageInt64Atomics` feature is supported, `VK_FORMAT_FEATURE_STORAGE_IMAGE_BIT` and `VK_FORMAT_FEATURE_STORAGE_IMAGE_ATOMIC_BIT` **must** be advertised in `optimalTilingFeatures` for both `VK_FORMAT_R64_UINT` and `VK_FORMAT_R64_SINT`.

*Table 84. Mandatory format support: depth/stencil with* `VkImageType` `VK_IMAGE_TYPE_2D`

| Format | VK_FORMAT_FEATURE_SAMPLED_IMAGE_BIT | VK_FORMAT_FEATURE_BLIT_SRC_BIT | VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT | VK_FORMAT_FEATURE_STORAGE_IMAGE_BIT | VK_FORMAT_FEATURE_STORAGE_IMAGE_ATOMIC_BIT | VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BIT | VK_FORMAT_FEATURE_BLIT_DST_BIT | VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT | VK_FORMAT_FEATURE_DEPTH_STENCIL_ATTACHMENT_BIT | VK_FORMAT_FEATURE_VERTEX_BUFFER_BIT | VK_FORMAT_FEATURE_UNIFORM_TEXEL_BUFFER_BIT | VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_BIT | VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_ATOMIC_BIT |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| VK_FORMAT_D16_UNORM | ◻ | ◻ | | | | | | | ◻ | | | | |
| VK_FORMAT_X8_D24_UNORM_PACK32 | | | | | | | | | † | | | | |
| VK_FORMAT_D32_SFLOAT | ◻ | ◻ | | | | | | | † | | | | |
| VK_FORMAT_S8_UINT | | | | | | | | | | | | | |
| VK_FORMAT_D16_UNORM_S8_UINT | | | | | | | | | | | | | |
| VK_FORMAT_D24_UNORM_S8_UINT | | | | | | | | | † | | | | |
| VK_FORMAT_D32_SFLOAT_S8_UINT | | | | | | | | | † | | | | |
| VK_FORMAT_FEATURE_DEPTH_STENCIL_ATTACHMENT_BIT feature **must** be supported for at least one of VK_FORMAT_X8_D24_UNORM_PACK32 and VK_FORMAT_D32_SFLOAT, and **must** be supported for at least one of VK_FORMAT_D24_UNORM_S8_UINT and VK_FORMAT_D32_SFLOAT_S8_UINT. | | | | | | | | | | | | | |
| bufferFeatures **must** not support any features for these formats | | | | | | | | | | | | | |

*Table 85. Mandatory format support: BC compressed formats with `VkImageType` `VK_IMAGE_TYPE_2D` and `VK_IMAGE_TYPE_3D`*

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_ATOMIC_BIT |
| | | | | | | | | | | | VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_BIT | |
| | | | | | | | | | | VK_FORMAT_FEATURE_UNIFORM_TEXEL_BUFFER_BIT | | |
| | | | | | | | | | VK_FORMAT_FEATURE_VERTEX_BUFFER_BIT | | | |
| | | | | | | | | VK_FORMAT_FEATURE_DEPTH_STENCIL_ATTACHMENT_BIT | | | | |
| | | | | | | | VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT | | | | | |
| | | | | | | VK_FORMAT_FEATURE_BLIT_DST_BIT | | | | | | |
| | | | | | VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BIT | | | | | | | |
| | | | | VK_FORMAT_FEATURE_STORAGE_IMAGE_ATOMIC_BIT | | | | | | | | |
| | | | VK_FORMAT_FEATURE_STORAGE_IMAGE_BIT | | | | | | | | | |
| | | VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT | | | | | | | | | | |
| | VK_FORMAT_FEATURE_BLIT_SRC_BIT | | | | | | | | | | | |
| VK_FORMAT_FEATURE_SAMPLED_IMAGE_BIT | | | | | | | | | | | | |
| **Format** | | | | | | | | | | | | |
| VK_FORMAT_BC1_RGB_UNORM_BLOCK | † | † | † | | | | | | | | | |
| VK_FORMAT_BC1_RGB_SRGB_BLOCK | † | † | † | | | | | | | | | |
| VK_FORMAT_BC1_RGBA_UNORM_BLOCK | † | † | † | | | | | | | | | |
| VK_FORMAT_BC1_RGBA_SRGB_BLOCK | † | † | † | | | | | | | | | |
| VK_FORMAT_BC2_UNORM_BLOCK | † | † | † | | | | | | | | | |
| VK_FORMAT_BC2_SRGB_BLOCK | † | † | † | | | | | | | | | |
| VK_FORMAT_BC3_UNORM_BLOCK | † | † | † | | | | | | | | | |
| VK_FORMAT_BC3_SRGB_BLOCK | † | † | † | | | | | | | | | |
| VK_FORMAT_BC4_UNORM_BLOCK | † | † | † | | | | | | | | | |
| VK_FORMAT_BC4_SNORM_BLOCK | † | † | † | | | | | | | | | |
| VK_FORMAT_BC5_UNORM_BLOCK | † | † | † | | | | | | | | | |
| VK_FORMAT_BC5_SNORM_BLOCK | † | † | † | | | | | | | | | |
| VK_FORMAT_BC6H_UFLOAT_BLOCK | † | † | † | | | | | | | | | |
| VK_FORMAT_BC6H_SFLOAT_BLOCK | † | † | † | | | | | | | | | |
| VK_FORMAT_BC7_UNORM_BLOCK | † | † | † | | | | | | | | | |
| VK_FORMAT_BC7_SRGB_BLOCK | † | † | † | | | | | | | | | |

The `VK_FORMAT_FEATURE_SAMPLED_IMAGE_BIT`, `VK_FORMAT_FEATURE_BLIT_SRC_BIT` and `VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT` features **must** be supported in `optimalTilingFeatures` for all the formats in at least one of: this table, Mandatory format support: ETC2 and EAC compressed formats with `VkImageType` `VK_IMAGE_TYPE_2D`, or Mandatory format support: ASTC LDR compressed formats with `VkImageType` `VK_IMAGE_TYPE_2D`.

*Table 86. Mandatory format support: ETC2 and EAC compressed formats with* `VkImageType` `VK_IMAGE_TYPE_2D`

| | VK_FORMAT_FEATURE_SAMPLED_IMAGE_BIT | VK_FORMAT_FEATURE_BLIT_SRC_BIT | VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT | VK_FORMAT_FEATURE_STORAGE_IMAGE_BIT | VK_FORMAT_FEATURE_STORAGE_IMAGE_ATOMIC_BIT | VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BIT | VK_FORMAT_FEATURE_BLIT_DST_BIT | VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT | VK_FORMAT_FEATURE_DEPTH_STENCIL_ATTACHMENT_BIT | VK_FORMAT_FEATURE_VERTEX_BUFFER_BIT | VK_FORMAT_FEATURE_UNIFORM_TEXEL_BUFFER_BIT | VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_BIT | VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_ATOMIC_BIT |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Format** | | | | | | | | | | | | | |
| VK_FORMAT_ETC2_R8G8B8_UNORM_BLOCK | † | † | † | | | | | | | | | | |
| VK_FORMAT_ETC2_R8G8B8_SRGB_BLOCK | † | † | † | | | | | | | | | | |
| VK_FORMAT_ETC2_R8G8B8A1_UNORM_BLOCK | † | † | † | | | | | | | | | | |
| VK_FORMAT_ETC2_R8G8B8A1_SRGB_BLOCK | † | † | † | | | | | | | | | | |
| VK_FORMAT_ETC2_R8G8B8A8_UNORM_BLOCK | † | † | † | | | | | | | | | | |
| VK_FORMAT_ETC2_R8G8B8A8_SRGB_BLOCK | † | † | † | | | | | | | | | | |
| VK_FORMAT_EAC_R11_UNORM_BLOCK | † | † | † | | | | | | | | | | |
| VK_FORMAT_EAC_R11_SNORM_BLOCK | † | † | † | | | | | | | | | | |
| VK_FORMAT_EAC_R11G11_UNORM_BLOCK | † | † | † | | | | | | | | | | |
| VK_FORMAT_EAC_R11G11_SNORM_BLOCK | † | † | † | | | | | | | | | | |

The `VK_FORMAT_FEATURE_SAMPLED_IMAGE_BIT`, `VK_FORMAT_FEATURE_BLIT_SRC_BIT` and `VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT` features **must** be supported in `optimalTilingFeatures` for all the formats in at least one of: this table, Mandatory format support: BC compressed formats with `VkImageType` `VK_IMAGE_TYPE_2D` and `VK_IMAGE_TYPE_3D`, or Mandatory format support: ASTC LDR compressed formats with `VkImageType` `VK_IMAGE_TYPE_2D`.

*Table 87. Mandatory format support: ASTC LDR compressed formats with* `VkImageType` `VK_IMAGE_TYPE_2D`

| Format | VK_FORMAT_FEATURE_SAMPLED_IMAGE_BIT | VK_FORMAT_FEATURE_BLIT_SRC_BIT | VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT | VK_FORMAT_FEATURE_STORAGE_IMAGE_BIT | VK_FORMAT_FEATURE_STORAGE_IMAGE_ATOMIC_BIT | VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BIT | VK_FORMAT_FEATURE_BLIT_DST_BIT | VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT | VK_FORMAT_FEATURE_DEPTH_STENCIL_ATTACHMENT_BIT | VK_FORMAT_FEATURE_VERTEX_BUFFER_BIT | VK_FORMAT_FEATURE_UNIFORM_TEXEL_BUFFER_BIT | VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_BIT | VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_ATOMIC_BIT |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| VK_FORMAT_ASTC_4x4_UNORM_BLOCK | † | † | † | | | | | | | | | | |
| VK_FORMAT_ASTC_4x4_SRGB_BLOCK | † | † | † | | | | | | | | | | |
| VK_FORMAT_ASTC_5x4_UNORM_BLOCK | † | † | † | | | | | | | | | | |
| VK_FORMAT_ASTC_5x4_SRGB_BLOCK | † | † | † | | | | | | | | | | |
| VK_FORMAT_ASTC_5x5_UNORM_BLOCK | † | † | † | | | | | | | | | | |
| VK_FORMAT_ASTC_5x5_SRGB_BLOCK | † | † | † | | | | | | | | | | |
| VK_FORMAT_ASTC_6x5_UNORM_BLOCK | † | † | † | | | | | | | | | | |
| VK_FORMAT_ASTC_6x5_SRGB_BLOCK | † | † | † | | | | | | | | | | |
| VK_FORMAT_ASTC_6x6_UNORM_BLOCK | † | † | † | | | | | | | | | | |
| VK_FORMAT_ASTC_6x6_SRGB_BLOCK | † | † | † | | | | | | | | | | |
| VK_FORMAT_ASTC_8x5_UNORM_BLOCK | † | † | † | | | | | | | | | | |
| VK_FORMAT_ASTC_8x5_SRGB_BLOCK | † | † | † | | | | | | | | | | |
| VK_FORMAT_ASTC_8x6_UNORM_BLOCK | † | † | † | | | | | | | | | | |
| VK_FORMAT_ASTC_8x6_SRGB_BLOCK | † | † | † | | | | | | | | | | |
| VK_FORMAT_ASTC_8x8_UNORM_BLOCK | † | † | † | | | | | | | | | | |
| VK_FORMAT_ASTC_8x8_SRGB_BLOCK | † | † | † | | | | | | | | | | |
| VK_FORMAT_ASTC_10x5_UNORM_BLOCK | † | † | † | | | | | | | | | | |
| VK_FORMAT_ASTC_10x5_SRGB_BLOCK | † | † | † | | | | | | | | | | |
| VK_FORMAT_ASTC_10x6_UNORM_BLOCK | † | † | † | | | | | | | | | | |
| VK_FORMAT_ASTC_10x6_SRGB_BLOCK | † | † | † | | | | | | | | | | |
| VK_FORMAT_ASTC_10x8_UNORM_BLOCK | † | † | † | | | | | | | | | | |

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| VK_FORMAT_ASTC_10x8_SRGB_BLOCK | † | † | † | | | | | | | | | | |
| VK_FORMAT_ASTC_10x10_UNORM_BLOCK | † | † | † | | | | | | | | | | |
| VK_FORMAT_ASTC_10x10_SRGB_BLOCK | † | † | † | | | | | | | | | | |
| VK_FORMAT_ASTC_12x10_UNORM_BLOCK | † | † | † | | | | | | | | | | |
| VK_FORMAT_ASTC_12x10_SRGB_BLOCK | † | † | † | | | | | | | | | | |
| VK_FORMAT_ASTC_12x12_UNORM_BLOCK | † | † | † | | | | | | | | | | |
| VK_FORMAT_ASTC_12x12_SRGB_BLOCK | † | † | † | | | | | | | | | | |
| The VK_FORMAT_FEATURE_SAMPLED_IMAGE_BIT, VK_FORMAT_FEATURE_BLIT_SRC_BIT and VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT features **must** be supported in optimalTilingFeatures for all the formats in at least one of: this table, Mandatory format support: BC compressed formats with VkImageType VK_IMAGE_TYPE_2D and VK_IMAGE_TYPE_3D, or Mandatory format support: ETC2 and EAC compressed formats with VkImageType VK_IMAGE_TYPE_2D. | | | | | | | | | | | | | |

If cubic filtering is supported, VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_CUBIC_BIT_EXT **must** be supported for the following image view types:

- VK_IMAGE_VIEW_TYPE_2D
- VK_IMAGE_VIEW_TYPE_2D_ARRAY

for the following formats:

- VK_FORMAT_R4G4_UNORM_PACK8
- VK_FORMAT_R4G4B4A4_UNORM_PACK16
- VK_FORMAT_B4G4R4A4_UNORM_PACK16
- VK_FORMAT_R5G6B5_UNORM_PACK16
- VK_FORMAT_B5G6R5_UNORM_PACK16
- VK_FORMAT_R5G5B5A1_UNORM_PACK16
- VK_FORMAT_B5G5R5A1_UNORM_PACK16
- VK_FORMAT_A1R5G5B5_UNORM_PACK16
- VK_FORMAT_R8_UNORM
- VK_FORMAT_R8_SNORM
- VK_FORMAT_R8_SRGB
- VK_FORMAT_R8G8_UNORM
- VK_FORMAT_R8G8_SNORM
- VK_FORMAT_R8G8_SRGB
- VK_FORMAT_R8G8B8_UNORM
- VK_FORMAT_R8G8B8_SNORM
- VK_FORMAT_R8G8B8_SRGB
- VK_FORMAT_B8G8R8_UNORM

- VK_FORMAT_B8G8R8_SNORM
- VK_FORMAT_B8G8R8_SRGB
- VK_FORMAT_R8G8B8A8_UNORM
- VK_FORMAT_R8G8B8A8_SNORM
- VK_FORMAT_R8G8B8A8_SRGB
- VK_FORMAT_B8G8R8A8_UNORM
- VK_FORMAT_B8G8R8A8_SNORM
- VK_FORMAT_B8G8R8A8_SRGB
- VK_FORMAT_A8B8G8R8_UNORM_PACK32
- VK_FORMAT_A8B8G8R8_SNORM_PACK32
- VK_FORMAT_A8B8G8R8_SRGB_PACK32

If ETC compressed formats are supported, VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_CUBIC_BIT_EXT **must** be supported for the following image view types:

- VK_IMAGE_VIEW_TYPE_2D
- VK_IMAGE_VIEW_TYPE_2D_ARRAY

for the following additional formats:

- VK_FORMAT_ETC2_R8G8B8_UNORM_BLOCK
- VK_FORMAT_ETC2_R8G8B8_SRGB_BLOCK
- VK_FORMAT_ETC2_R8G8B8A1_UNORM_BLOCK
- VK_FORMAT_ETC2_R8G8B8A1_SRGB_BLOCK
- VK_FORMAT_ETC2_R8G8B8A8_UNORM_BLOCK
- VK_FORMAT_ETC2_R8G8B8A8_SRGB_BLOCK

If cubic filtering is supported for any other formats, the following image view types **must** be supported for those formats:

- VK_IMAGE_VIEW_TYPE_2D
- VK_IMAGE_VIEW_TYPE_2D_ARRAY

To be used with VkImageView with subresourceRange.aspectMask equal to VK_IMAGE_ASPECT_COLOR_BIT, sampler $Y'C_BC_R$ conversion **must** be enabled for the following formats:

*Table 88. Formats requiring sampler $Y'C_BC_R$ conversion for VK_IMAGE_ASPECT_COLOR_BIT image views*

| | | VK_FORMAT_FEATURE_SAMPLED_IMAGE_YCBCR_CONVERSION_CHROMA_RECONSTRUCTION_EXPLICIT_FORCEABLE_BIT | VK_FORMAT_FEATURE_SAMPLED_IMAGE_YCBCR_CONVERSION_CHROMA_RECONSTRUCTION_EXPLICIT_BIT | VK_FORMAT_FEATURE_SAMPLED_IMAGE_YCBCR_CONVERSION_SEPARATE_RECONSTRUCTION_FILTER_BIT | VK_FORMAT_FEATURE_SAMPLED_IMAGE_YCBCR_CONVERSION_LINEAR_FILTER_BIT | VK_FORMAT_FEATURE_MIDPOINT_CHROMA_SAMPLES_BIT | VK_FORMAT_FEATURE_COSITED_CHROMA_SAMPLES_BIT | VK_FORMAT_FEATURE_TRANSFER_DST_BIT | VK_FORMAT_FEATURE_TRANSFER_SRC_BIT | VK_FORMAT_FEATURE_SAMPLED_IMAGE_BIT | VK_FORMAT_FEATURE_DISJOINT_BIT | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Format** | **Planes** | | | | | | | | | | | |
| VK_FORMAT_G8B8G8R8_422_UNORM | 1 | | | | | | | | | | | |
| VK_FORMAT_B8G8R8G8_422_UNORM | 1 | | | | | | | | | | | |
| VK_FORMAT_G8_B8_R8_3PLANE_420_UNORM | 3 | | | † | † | † | † | | | | | |
| VK_FORMAT_G8_B8R8_2PLANE_420_UNORM | 2 | | | † | † | † | † | | | | | |
| VK_FORMAT_G8_B8_R8_3PLANE_422_UNORM | 3 | | | | | | | | | | | |
| VK_FORMAT_G8_B8R8_2PLANE_422_UNORM | 2 | | | | | | | | | | | |
| VK_FORMAT_G8_B8_R8_3PLANE_444_UNORM | 3 | | | | | | | | | | | |
| VK_FORMAT_R10X6G10X6B10X6A10X6_UNORM_4PACK16 ‡ | 1 | | | | | | | | | | | |
| VK_FORMAT_G10X6B10X6G10X6R10X6_422_UNORM_4PACK16 | 1 | | | | | | | | | | | |
| VK_FORMAT_B10X6G10X6R10X6G10X6_422_UNORM_4PACK16 | 1 | | | | | | | | | | | |
| VK_FORMAT_G10X6_B10X6_R10X6_3PLANE_420_UNORM_3PACK16 | 3 | | | | | | | | | | | |
| VK_FORMAT_G10X6_B10X6R10X6_2PLANE_420_UNORM_3PACK16 | 2 | | | | | | | | | | | |
| VK_FORMAT_G10X6_B10X6_R10X6_3PLANE_422_UNORM_3PACK16 | 3 | | | | | | | | | | | |
| VK_FORMAT_G10X6_B10X6R10X6_2PLANE_422_UNORM_3PACK16 | 2 | | | | | | | | | | | |
| VK_FORMAT_G10X6_B10X6_R10X6_3PLANE_444_UNORM_3PACK16 | 3 | | | | | | | | | | | |
| VK_FORMAT_R12X4G12X4B12X4A12X4_UNORM_4PACK16 | 1 | | | | | | | | | | | |
| VK_FORMAT_G12X4B12X4G12X4R12X4_422_UNORM_4PACK16 | 1 | | | | | | | | | | | |
| VK_FORMAT_B12X4G12X4R12X4G12X4_422_UNORM_4PACK16 | 1 | | | | | | | | | | | |
| VK_FORMAT_G12X4_B12X4_R12X4_3PLANE_420_UNORM_3PACK16 | 3 | | | | | | | | | | | |
| VK_FORMAT_G12X4_B12X4R12X4_2PLANE_420_UNORM_3PACK16 | 2 | | | | | | | | | | | |
| VK_FORMAT_G12X4_B12X4_R12X4_3PLANE_422_UNORM_3PACK16 | 3 | | | | | | | | | | | |
| VK_FORMAT_G12X4_B12X4R12X4_2PLANE_422_UNORM_3PACK16 | 2 | | | | | | | | | | | |
| VK_FORMAT_G12X4_B12X4_R12X4_3PLANE_444_UNORM_3PACK16 | 3 | | | | | | | | | | | |

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| VK_FORMAT_G16B16G16R16_422_UNORM | 1 | | | | | | | | | | |
| VK_FORMAT_B16G16R16G16_422_UNORM | 1 | | | | | | | | | | |
| VK_FORMAT_G16_B16_R16_3PLANE_420_UNORM | 3 | | | | | | | | | | |
| VK_FORMAT_G16_B16R16_2PLANE_420_UNORM | 2 | | | | | | | | | | |
| VK_FORMAT_G16_B16_R16_3PLANE_422_UNORM | 3 | | | | | | | | | | |
| VK_FORMAT_G16_B16R16_2PLANE_422_UNORM | 2 | | | | | | | | | | |
| VK_FORMAT_G16_B16_R16_3PLANE_444_UNORM | 3 | | | | | | | | | | |
| VK_FORMAT_G8_B8R8_2PLANE_444_UNORM | 2 | | | | | | | | | | |
| VK_FORMAT_G10X6_B10X6R10X6_2PLANE_444_UNORM_3PACK16 | 2 | | | | | | | | | | |
| VK_FORMAT_G12X4_B12X4R12X4_2PLANE_444_UNORM_3PACK16 | 2 | | | | | | | | | | |
| VK_FORMAT_G16_B16R16_2PLANE_444_UNORM | 2 | | | | | | | | | | |
| Format features marked † **must** be supported for optimalTilingFeatures with VkImageType VK_IMAGE_TYPE_2D if the VkPhysicalDevice supports the VkPhysicalDeviceSamplerYcbcrConversionFeatures feature. | | | | | | | | | | | |
| Formats marked ‡ do not require a sampler $Y'C_BC_R$ conversion for VK_IMAGE_ASPECT_COLOR_BIT image views if the VkPhysicalDeviceRGBA10X6FormatsFeaturesEXT::formatRgba10x6WithoutYCbCrSampler feature is enabled. | | | | | | | | | | | |

Implementations are not required to support the VK_IMAGE_CREATE_SPARSE_BINDING_BIT, VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT, or VK_IMAGE_CREATE_SPARSE_ALIASED_BIT VkImageCreateFlags for the above formats that require sampler $Y'C_BC_R$ conversion. To determine whether the implementation supports sparse image creation flags with these formats use vkGetPhysicalDeviceImageFormatProperties or vkGetPhysicalDeviceImageFormatProperties2.

VK_FORMAT_FEATURE_FRAGMENT_DENSITY_MAP_BIT_EXT **must** be supported for the following formats if the fragmentDensityMap feature is enabled:

- VK_FORMAT_R8G8_UNORM

VK_FORMAT_FEATURE_ACCELERATION_STRUCTURE_VERTEX_BUFFER_BIT_KHR **must** be supported in bufferFeatures for the following formats if the accelerationStructure feature is supported:

- VK_FORMAT_R32G32_SFLOAT
- VK_FORMAT_R32G32B32_SFLOAT
- VK_FORMAT_R16G16_SFLOAT
- VK_FORMAT_R16G16B16A16_SFLOAT
- VK_FORMAT_R16G16_SNORM
- VK_FORMAT_R16G16B16A16_SNORM

VK_FORMAT_FEATURE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR **must** be supported for the following formats if the attachmentFragmentShadingRate feature is supported:

- VK_FORMAT_R8_UINT

If VK_EXT_host_image_copy is supported and VK_FORMAT_FEATURE_SAMPLED_IMAGE_BIT is supported in optimalTilingFeatures or linearTilingFeatures for a color format, VK_FORMAT_FEATURE_2_HOST_IMAGE_TRANSFER_BIT_EXT **must** also be supported in optimalTilingFeatures or linearTilingFeatures respectively.

### 48.3.1. Formats Without Shader Storage Format

The device-level features for using a storage image or a storage texel buffer with an image format of Unknown, shaderStorageImageReadWithoutFormat and shaderStorageImageWriteWithoutFormat, only apply to the following formats:

- VK_FORMAT_R8G8B8A8_UNORM
- VK_FORMAT_R8G8B8A8_SNORM
- VK_FORMAT_R8G8B8A8_UINT
- VK_FORMAT_R8G8B8A8_SINT
- VK_FORMAT_R32_UINT
- VK_FORMAT_R32_SINT
- VK_FORMAT_R32_SFLOAT
- VK_FORMAT_R32G32_UINT
- VK_FORMAT_R32G32_SINT
- VK_FORMAT_R32G32_SFLOAT
- VK_FORMAT_R32G32B32A32_UINT
- VK_FORMAT_R32G32B32A32_SINT
- VK_FORMAT_R32G32B32A32_SFLOAT
- VK_FORMAT_R16G16B16A16_UINT
- VK_FORMAT_R16G16B16A16_SINT
- VK_FORMAT_R16G16B16A16_SFLOAT
- VK_FORMAT_R16G16_SFLOAT
- VK_FORMAT_B10G11R11_UFLOAT_PACK32
- VK_FORMAT_R16_SFLOAT
- VK_FORMAT_R16G16B16A16_UNORM
- VK_FORMAT_A2B10G10R10_UNORM_PACK32
- VK_FORMAT_R16G16_UNORM
- VK_FORMAT_R8G8_UNORM
- VK_FORMAT_R16_UNORM
- VK_FORMAT_R8_UNORM
- VK_FORMAT_R16G16B16A16_SNORM

- VK_FORMAT_R16G16_SNORM
- VK_FORMAT_R8G8_SNORM
- VK_FORMAT_R16_SNORM
- VK_FORMAT_R8_SNORM
- VK_FORMAT_R16G16_SINT
- VK_FORMAT_R8G8_SINT
- VK_FORMAT_R16_SINT
- VK_FORMAT_R8_SINT
- VK_FORMAT_A2B10G10R10_UINT_PACK32
- VK_FORMAT_R16G16_UINT
- VK_FORMAT_R8G8_UINT
- VK_FORMAT_R16_UINT
- VK_FORMAT_R8_UINT
- VK_FORMAT_A8_UNORM_KHR

**NOTE** This list of formats is the union of required storage formats from Required Format Support section and formats listed in shaderStorageImageExtendedFormats.

An implementation that supports VK_FORMAT_FEATURE_STORAGE_IMAGE_BIT for any format from the given list of formats and supports shaderStorageImageReadWithoutFormat **must** support VK_FORMAT_FEATURE_2_STORAGE_READ_WITHOUT_FORMAT_BIT for that same format if Vulkan 1.3 or the VK_KHR_format_feature_flags2 extension is supported.

An implementation that supports VK_FORMAT_FEATURE_STORAGE_IMAGE_BIT for any format from the given list of formats and supports shaderStorageImageWriteWithoutFormat **must** support VK_FORMAT_FEATURE_2_STORAGE_WRITE_WITHOUT_FORMAT_BIT for that same format if Vulkan 1.3 or the VK_KHR_format_feature_flags2 extension is supported.

An implementation that does not support either of VK_FORMAT_FEATURE_2_STORAGE_READ_WITHOUT_FORMAT_BIT or VK_FORMAT_FEATURE_2_STORAGE_WRITE_WITHOUT_FORMAT_BIT for a format **must** not report support for VK_FORMAT_FEATURE_STORAGE_IMAGE_BIT or VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_BIT for that format if it is not listed in the SPIR-V and Vulkan Image Format Compatibility table.

**NOTE** Some older implementations do not follow this restriction. They report support for formats as storage images even though they do not support access without the Format qualifier and there is no matching Format token. Such images cannot be either read from or written to.

Drivers which pass Vulkan conformance test suite version 1.3.9.0, or any subsequent version will conform to the requirement above.

### 48.3.2. Depth Comparison Format Support

If Vulkan 1.3 or the VK_KHR_format_feature_flags2 extension is supported, a depth/stencil format with a depth component supporting VK_FORMAT_FEATURE_SAMPLED_IMAGE_BIT **must** support VK_FORMAT_FEATURE_2_SAMPLED_IMAGE_DEPTH_COMPARISON_BIT.

### 48.3.3. Format Feature Dependent Usage Flags

Certain resource usage flags depend on support for the corresponding format feature flag for the format in question. The following tables list the VkBufferUsageFlagBits and VkImageUsageFlagBits that have such dependencies, and the format feature flags they depend on. Additional restrictions, including, but not limited to, further required format feature flags specific to the particular use of the resource **may** apply, as described in the respective sections of this specification.

*Table 89. Format feature dependent buffer usage flags*

| Buffer usage flag | Required format feature flag |
|---|---|
| VK_BUFFER_USAGE_UNIFORM_TEXEL_BUFFER_BIT | VK_FORMAT_FEATURE_UNIFORM_TEXEL_BUFFER_BIT |
| VK_BUFFER_USAGE_STORAGE_TEXEL_BUFFER_BIT | VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_BIT |
| VK_BUFFER_USAGE_VERTEX_BUFFER_BIT | VK_FORMAT_FEATURE_VERTEX_BUFFER_BIT |

*Table 90. Format feature dependent image usage flags*

| Image usage flag | Required format feature flag |
|---|---|
| VK_IMAGE_USAGE_SAMPLED_BIT | VK_FORMAT_FEATURE_SAMPLED_IMAGE_BIT |
| VK_IMAGE_USAGE_STORAGE_BIT | VK_FORMAT_FEATURE_STORAGE_IMAGE_BIT |
| VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT | VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BIT |
| VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT | VK_FORMAT_FEATURE_DEPTH_STENCIL_ATTACHMENT_BIT |
| VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT | VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BIT or VK_FORMAT_FEATURE_DEPTH_STENCIL_ATTACHMENT_BIT |
| VK_IMAGE_USAGE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR | VK_FORMAT_FEATURE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR |
| VK_IMAGE_USAGE_VIDEO_DECODE_DST_BIT_KHR | VK_FORMAT_FEATURE_VIDEO_DECODE_OUTPUT_BIT_KHR |
| VK_IMAGE_USAGE_VIDEO_DECODE_DPB_BIT_KHR | VK_FORMAT_FEATURE_VIDEO_DECODE_DPB_BIT_KHR |
| VK_IMAGE_USAGE_VIDEO_ENCODE_SRC_BIT_KHR | VK_FORMAT_FEATURE_VIDEO_ENCODE_INPUT_BIT_KHR |
| VK_IMAGE_USAGE_VIDEO_ENCODE_DPB_BIT_KHR | VK_FORMAT_FEATURE_VIDEO_ENCODE_DPB_BIT_KHR |

# Chapter 49. Additional Capabilities

This chapter describes additional capabilities beyond the minimum capabilities described in the Limits and Formats chapters, including:

- Additional Image Capabilities
- Additional Buffer Capabilities
- Optional Semaphore Capabilities
- Optional Fence Capabilities
- Timestamp Calibration Capabilities

## 49.1. Additional Image Capabilities

Additional image capabilities, such as larger dimensions or additional sample counts for certain image types, or additional capabilities for *linear* tiling format images, are described in this section.

To query additional capabilities specific to image types, call:

```
// Provided by VK_VERSION_1_0
VkResult vkGetPhysicalDeviceImageFormatProperties(
    VkPhysicalDevice                            physicalDevice,
    VkFormat                                    format,
    VkImageType                                 type,
    VkImageTiling                               tiling,
    VkImageUsageFlags                           usage,
    VkImageCreateFlags                          flags,
    VkImageFormatProperties*                    pImageFormatProperties);
```

- `physicalDevice` is the physical device from which to query the image capabilities.
- `format` is a VkFormat value specifying the image format, corresponding to VkImageCreateInfo::`format`.
- `type` is a VkImageType value specifying the image type, corresponding to VkImageCreateInfo::`imageType`.
- `tiling` is a VkImageTiling value specifying the image tiling, corresponding to VkImageCreateInfo::`tiling`.
- `usage` is a bitmask of VkImageUsageFlagBits specifying the intended usage of the image, corresponding to VkImageCreateInfo::`usage`.
- `flags` is a bitmask of VkImageCreateFlagBits specifying additional parameters of the image, corresponding to VkImageCreateInfo::`flags`.
- `pImageFormatProperties` is a pointer to a VkImageFormatProperties structure in which capabilities are returned.

The `format`, `type`, `tiling`, `usage`, and `flags` parameters correspond to parameters that would be

consumed by vkCreateImage (as members of VkImageCreateInfo).

If format is not a supported image format, or if the combination of format, type, tiling, usage, and flags is not supported for images, then vkGetPhysicalDeviceImageFormatProperties returns VK_ERROR_FORMAT_NOT_SUPPORTED.

The limitations on an image format that are reported by vkGetPhysicalDeviceImageFormatProperties have the following property: if usage1 and usage2 of type VkImageUsageFlags are such that the bits set in usage1 are a subset of the bits set in usage2, and flags1 and flags2 of type VkImageCreateFlags are such that the bits set in flags1 are a subset of the bits set in flags2, then the limitations for usage1 and flags1 **must** be no more strict than the limitations for usage2 and flags2, for all values of format, type, and tiling.

If VK_EXT_host_image_copy is supported, usage includes VK_IMAGE_USAGE_SAMPLED_BIT, and flags does not include either of VK_IMAGE_CREATE_SPARSE_BINDING_BIT, VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT, or VK_IMAGE_CREATE_SPARSE_ALIASED_BIT, then the result of calls to vkGetPhysicalDeviceImageFormatProperties with identical parameters except for the inclusion of VK_IMAGE_USAGE_HOST_TRANSFER_BIT_EXT in usage **must** be identical.

## Valid Usage

- VUID-vkGetPhysicalDeviceImageFormatProperties-tiling-02248
  tiling **must** not be VK_IMAGE_TILING_DRM_FORMAT_MODIFIER_EXT. (Use vkGetPhysicalDeviceImageFormatProperties2 instead)

## Valid Usage (Implicit)

- VUID-vkGetPhysicalDeviceImageFormatProperties-physicalDevice-parameter
  physicalDevice **must** be a valid VkPhysicalDevice handle
- VUID-vkGetPhysicalDeviceImageFormatProperties-format-parameter
  format **must** be a valid VkFormat value
- VUID-vkGetPhysicalDeviceImageFormatProperties-type-parameter
  type **must** be a valid VkImageType value
- VUID-vkGetPhysicalDeviceImageFormatProperties-tiling-parameter
  tiling **must** be a valid VkImageTiling value
- VUID-vkGetPhysicalDeviceImageFormatProperties-usage-parameter
  usage **must** be a valid combination of VkImageUsageFlagBits values
- VUID-vkGetPhysicalDeviceImageFormatProperties-usage-requiredbitmask
  usage **must** not be 0
- VUID-vkGetPhysicalDeviceImageFormatProperties-flags-parameter
  flags **must** be a valid combination of VkImageCreateFlagBits values
- VUID-vkGetPhysicalDeviceImageFormatProperties-pImageFormatProperties-parameter
  pImageFormatProperties **must** be a valid pointer to a VkImageFormatProperties structure

## Return Codes

**Success**

- VK_SUCCESS

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY
- VK_ERROR_FORMAT_NOT_SUPPORTED

The VkImageFormatProperties structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkImageFormatProperties {
    VkExtent3D            maxExtent;
    uint32_t              maxMipLevels;
    uint32_t              maxArrayLayers;
    VkSampleCountFlags    sampleCounts;
    VkDeviceSize          maxResourceSize;
} VkImageFormatProperties;
```

- maxExtent are the maximum image dimensions. See the Allowed Extent Values section below for how these values are constrained by type.
- maxMipLevels is the maximum number of mipmap levels. maxMipLevels **must** be equal to the number of levels in the complete mipmap chain based on the maxExtent.width, maxExtent.height, and maxExtent.depth, except when one of the following conditions is true, in which case it **may** instead be 1:
  - vkGetPhysicalDeviceImageFormatProperties::tiling was VK_IMAGE_TILING_LINEAR
  - VkPhysicalDeviceImageFormatInfo2::tiling was VK_IMAGE_TILING_DRM_FORMAT_MODIFIER_EXT
  - the VkPhysicalDeviceImageFormatInfo2::pNext chain included a VkPhysicalDeviceExternalImageFormatInfo structure with a handle type included in the handleTypes member for which mipmap image support is not required
  - image format is one of the formats that require a sampler $Y'C_BC_R$ conversion
  - flags contains VK_IMAGE_CREATE_SUBSAMPLED_BIT_EXT
- maxArrayLayers is the maximum number of array layers. maxArrayLayers **must** be no less than VkPhysicalDeviceLimits::maxImageArrayLayers, except when one of the following conditions is true, in which case it **may** instead be 1:
  - tiling is VK_IMAGE_TILING_LINEAR
  - tiling is VK_IMAGE_TILING_OPTIMAL and type is VK_IMAGE_TYPE_3D
  - format is one of the formats that require a sampler $Y'C_BC_R$ conversion
- If tiling is VK_IMAGE_TILING_DRM_FORMAT_MODIFIER_EXT, then maxArrayLayers **must** not be 0.

- sampleCounts is a bitmask of VkSampleCountFlagBits specifying all the supported sample counts for this image as described below.
- maxResourceSize is an upper bound on the total image size in bytes, inclusive of all image subresources. Implementations **may** have an address space limit on total size of a resource, which is advertised by this property. maxResourceSize **must** be at least $2^{31}$.

**NOTE** There is no mechanism to query the size of an image before creating it, to compare that size against maxResourceSize. If an application attempts to create an image that exceeds this limit, the creation will fail and vkCreateImage will return VK_ERROR_OUT_OF_DEVICE_MEMORY. While the advertised limit **must** be at least $2^{31}$, it **may** not be possible to create an image that approaches that size, particularly for VK_IMAGE_TYPE_1D.

If the combination of parameters to vkGetPhysicalDeviceImageFormatProperties is not supported by the implementation for use in vkCreateImage, then all members of VkImageFormatProperties will be filled with zero.

**NOTE** Filling VkImageFormatProperties with zero for unsupported formats is an exception to the usual rule that output structures have undefined contents on error. This exception was unintentional, but is preserved for backwards compatibility.

To determine the image capabilities compatible with an external memory handle type, call:

```
// Provided by VK_NV_external_memory_capabilities
VkResult vkGetPhysicalDeviceExternalImageFormatPropertiesNV(
    VkPhysicalDevice                            physicalDevice,
    VkFormat                                    format,
    VkImageType                                 type,
    VkImageTiling                               tiling,
    VkImageUsageFlags                           usage,
    VkImageCreateFlags                          flags,
    VkExternalMemoryHandleTypeFlagsNV           externalHandleType,
    VkExternalImageFormatPropertiesNV*          pExternalImageFormatProperties);
```

- physicalDevice is the physical device from which to query the image capabilities
- format is the image format, corresponding to VkImageCreateInfo::format.
- type is the image type, corresponding to VkImageCreateInfo::imageType.
- tiling is the image tiling, corresponding to VkImageCreateInfo::tiling.
- usage is the intended usage of the image, corresponding to VkImageCreateInfo::usage.
- flags is a bitmask describing additional parameters of the image, corresponding to VkImageCreateInfo::flags.
- externalHandleType is either one of the bits from VkExternalMemoryHandleTypeFlagBitsNV, or 0.
- pExternalImageFormatProperties is a pointer to a VkExternalImageFormatPropertiesNV structure

in which capabilities are returned.

If externalHandleType is 0, pExternalImageFormatProperties->imageFormatProperties will return the same values as a call to vkGetPhysicalDeviceImageFormatProperties, and the other members of pExternalImageFormatProperties will all be 0. Otherwise, they are filled in as described for VkExternalImageFormatPropertiesNV.

## Valid Usage

- VUID-vkGetPhysicalDeviceExternalImageFormatPropertiesNV-externalHandleType-07721 externalHandleType **must** not have more than one bit set

## Valid Usage (Implicit)

- VUID-vkGetPhysicalDeviceExternalImageFormatPropertiesNV-physicalDevice-parameter physicalDevice **must** be a valid VkPhysicalDevice handle
- VUID-vkGetPhysicalDeviceExternalImageFormatPropertiesNV-format-parameter format **must** be a valid VkFormat value
- VUID-vkGetPhysicalDeviceExternalImageFormatPropertiesNV-type-parameter type **must** be a valid VkImageType value
- VUID-vkGetPhysicalDeviceExternalImageFormatPropertiesNV-tiling-parameter tiling **must** be a valid VkImageTiling value
- VUID-vkGetPhysicalDeviceExternalImageFormatPropertiesNV-usage-parameter usage **must** be a valid combination of VkImageUsageFlagBits values
- VUID-vkGetPhysicalDeviceExternalImageFormatPropertiesNV-usage-requiredbitmask usage **must** not be 0
- VUID-vkGetPhysicalDeviceExternalImageFormatPropertiesNV-flags-parameter flags **must** be a valid combination of VkImageCreateFlagBits values
- VUID-vkGetPhysicalDeviceExternalImageFormatPropertiesNV-externalHandleType-parameter externalHandleType **must** be a valid combination of VkExternalMemoryHandleTypeFlagBitsNV values
- VUID-vkGetPhysicalDeviceExternalImageFormatPropertiesNV-pExternalImageFormatProperties-parameter pExternalImageFormatProperties **must** be a valid pointer to a VkExternalImageFormatPropertiesNV structure

## Return Codes

### Success

- VK_SUCCESS

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY
- VK_ERROR_FORMAT_NOT_SUPPORTED

The VkExternalImageFormatPropertiesNV structure is defined as:

```
// Provided by VK_NV_external_memory_capabilities
typedef struct VkExternalImageFormatPropertiesNV {
    VkImageFormatProperties              imageFormatProperties;
    VkExternalMemoryFeatureFlagsNV       externalMemoryFeatures;
    VkExternalMemoryHandleTypeFlagsNV    exportFromImportedHandleTypes;
    VkExternalMemoryHandleTypeFlagsNV    compatibleHandleTypes;
} VkExternalImageFormatPropertiesNV;
```

- imageFormatProperties will be filled in as when calling vkGetPhysicalDeviceImageFormatProperties, but the values returned **may** vary depending on the external handle type requested.
- externalMemoryFeatures is a bitmask of VkExternalMemoryFeatureFlagBitsNV, indicating properties of the external memory handle type (vkGetPhysicalDeviceExternalImageFormatPropertiesNV::externalHandleType) being queried, or 0 if the external memory handle type is 0.
- exportFromImportedHandleTypes is a bitmask of VkExternalMemoryHandleTypeFlagBitsNV containing a bit set for every external handle type that **may** be used to create memory from which the handles of the type specified in vkGetPhysicalDeviceExternalImageFormatPropertiesNV::externalHandleType **can** be exported, or 0 if the external memory handle type is 0.
- compatibleHandleTypes is a bitmask of VkExternalMemoryHandleTypeFlagBitsNV containing a bit set for every external handle type that **may** be specified simultaneously with the handle type specified by vkGetPhysicalDeviceExternalImageFormatPropertiesNV::externalHandleType when calling vkAllocateMemory, or 0 if the external memory handle type is 0. compatibleHandleTypes will always contain vkGetPhysicalDeviceExternalImageFormatPropertiesNV::externalHandleType

Bits which **can** be set in VkExternalImageFormatPropertiesNV::externalMemoryFeatures, indicating properties of the external memory handle type, are:

```
// Provided by VK_NV_external_memory_capabilities
typedef enum VkExternalMemoryFeatureFlagBitsNV {
    VK_EXTERNAL_MEMORY_FEATURE_DEDICATED_ONLY_BIT_NV = 0x00000001,
    VK_EXTERNAL_MEMORY_FEATURE_EXPORTABLE_BIT_NV = 0x00000002,
    VK_EXTERNAL_MEMORY_FEATURE_IMPORTABLE_BIT_NV = 0x00000004,
} VkExternalMemoryFeatureFlagBitsNV;
```

- VK_EXTERNAL_MEMORY_FEATURE_DEDICATED_ONLY_BIT_NV specifies that external memory of the

specified type **must** be created as a dedicated allocation when used in the manner specified.

- VK_EXTERNAL_MEMORY_FEATURE_EXPORTABLE_BIT_NV specifies that the implementation supports exporting handles of the specified type.
- VK_EXTERNAL_MEMORY_FEATURE_IMPORTABLE_BIT_NV specifies that the implementation supports importing handles of the specified type.

```
// Provided by VK_NV_external_memory_capabilities
typedef VkFlags VkExternalMemoryFeatureFlagsNV;
```

VkExternalMemoryFeatureFlagsNV is a bitmask type for setting a mask of zero or more VkExternalMemoryFeatureFlagBitsNV.

To query additional capabilities specific to image types, call:

```
// Provided by VK_VERSION_1_1
VkResult vkGetPhysicalDeviceImageFormatProperties2(
    VkPhysicalDevice                            physicalDevice,
    const VkPhysicalDeviceImageFormatInfo2*     pImageFormatInfo,
    VkImageFormatProperties2*                   pImageFormatProperties);
```

or the equivalent command

```
// Provided by VK_KHR_get_physical_device_properties2
VkResult vkGetPhysicalDeviceImageFormatProperties2KHR(
    VkPhysicalDevice                            physicalDevice,
    const VkPhysicalDeviceImageFormatInfo2*     pImageFormatInfo,
    VkImageFormatProperties2*                   pImageFormatProperties);
```

- physicalDevice is the physical device from which to query the image capabilities.
- pImageFormatInfo is a pointer to a VkPhysicalDeviceImageFormatInfo2 structure describing the parameters that would be consumed by vkCreateImage.
- pImageFormatProperties is a pointer to a VkImageFormatProperties2 structure in which capabilities are returned.

vkGetPhysicalDeviceImageFormatProperties2 behaves similarly to vkGetPhysicalDeviceImageFormatProperties, with the ability to return extended information in a pNext chain of output structures.

If the pNext chain of pImageFormatInfo includes a VkVideoProfileListInfoKHR structure with a profileCount member greater than 0, then this command returns format capabilities specific to image types used in conjunction with the specified video profiles. In this case, this command will return one of the video-profile-specific error codes if any of the profiles specified via VkVideoProfileListInfoKHR::pProfiles are not supported. Furthermore, if VkPhysicalDeviceImageFormatInfo2::usage includes any image usage flag not supported by the specified video profiles, then this command returns VK_ERROR_IMAGE_USAGE_NOT_SUPPORTED_KHR.

## Valid Usage

- VUID-vkGetPhysicalDeviceImageFormatProperties2-pNext-01868
  If the `pNext` chain of `pImageFormatProperties` includes a VkAndroidHardwareBufferUsageANDROID structure, the `pNext` chain of `pImageFormatInfo` **must** include a VkPhysicalDeviceExternalImageFormatInfo structure with `handleType` set to `VK_EXTERNAL_MEMORY_HANDLE_TYPE_ANDROID_HARDWARE_BUFFER_BIT_ANDROID`
- VUID-vkGetPhysicalDeviceImageFormatProperties2-pNext-09004
  If the `pNext` chain of `pImageFormatProperties` includes a VkHostImageCopyDevicePerformanceQueryEXT structure, `pImageFormatInfo->usage` **must** contain `VK_IMAGE_USAGE_HOST_TRANSFER_BIT_EXT`

## Valid Usage (Implicit)

- VUID-vkGetPhysicalDeviceImageFormatProperties2-physicalDevice-parameter
  `physicalDevice` **must** be a valid VkPhysicalDevice handle
- VUID-vkGetPhysicalDeviceImageFormatProperties2-pImageFormatInfo-parameter
  `pImageFormatInfo` **must** be a valid pointer to a valid VkPhysicalDeviceImageFormatInfo2 structure
- VUID-vkGetPhysicalDeviceImageFormatProperties2-pImageFormatProperties-parameter
  `pImageFormatProperties` **must** be a valid pointer to a VkImageFormatProperties2 structure

## Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_FORMAT_NOT_SUPPORTED`
- `VK_ERROR_IMAGE_USAGE_NOT_SUPPORTED_KHR`
- `VK_ERROR_VIDEO_PROFILE_OPERATION_NOT_SUPPORTED_KHR`
- `VK_ERROR_VIDEO_PROFILE_FORMAT_NOT_SUPPORTED_KHR`
- `VK_ERROR_VIDEO_PICTURE_LAYOUT_NOT_SUPPORTED_KHR`
- `VK_ERROR_VIDEO_PROFILE_CODEC_NOT_SUPPORTED_KHR`

The `VkPhysicalDeviceImageFormatInfo2` structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkPhysicalDeviceImageFormatInfo2 {
    VkStructureType       sType;
    const void*           pNext;
    VkFormat              format;
    VkImageType           type;
    VkImageTiling         tiling;
    VkImageUsageFlags     usage;
    VkImageCreateFlags    flags;
} VkPhysicalDeviceImageFormatInfo2;
```

or the equivalent

```
// Provided by VK_KHR_get_physical_device_properties2
typedef VkPhysicalDeviceImageFormatInfo2 VkPhysicalDeviceImageFormatInfo2KHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure. The `pNext` chain of `VkPhysicalDeviceImageFormatInfo2` is used to provide additional image parameters to `vkGetPhysicalDeviceImageFormatProperties2`.
- `format` is a VkFormat value indicating the image format, corresponding to VkImageCreateInfo::`format`.
- `type` is a VkImageType value indicating the image type, corresponding to VkImageCreateInfo::`imageType`.
- `tiling` is a VkImageTiling value indicating the image tiling, corresponding to VkImageCreateInfo::`tiling`.
- `usage` is a bitmask of VkImageUsageFlagBits indicating the intended usage of the image, corresponding to VkImageCreateInfo::`usage`.
- `flags` is a bitmask of VkImageCreateFlagBits indicating additional parameters of the image, corresponding to VkImageCreateInfo::`flags`.

The members of `VkPhysicalDeviceImageFormatInfo2` correspond to the arguments to vkGetPhysicalDeviceImageFormatProperties, with `sType` and `pNext` added for extensibility.

### Valid Usage

- VUID-VkPhysicalDeviceImageFormatInfo2-tiling-02249
  `tiling` **must** be `VK_IMAGE_TILING_DRM_FORMAT_MODIFIER_EXT` if and only if the `pNext` chain includes VkPhysicalDeviceImageDrmFormatModifierInfoEXT
- VUID-VkPhysicalDeviceImageFormatInfo2-tiling-02313
  If `tiling` is `VK_IMAGE_TILING_DRM_FORMAT_MODIFIER_EXT` and `flags` contains `VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT`, then the `pNext` chain **must** include a VkImageFormatListCreateInfo structure with non-zero `viewFormatCount`

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceImageFormatInfo2-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_IMAGE_FORMAT_INFO_2`
- VUID-VkPhysicalDeviceImageFormatInfo2-pNext-pNext
  Each `pNext` member of any structure (including this one) in the `pNext` chain **must** be either `NULL` or a pointer to a valid instance of VkImageCompressionControlEXT, VkImageFormatListCreateInfo, VkImageStencilUsageCreateInfo, VkOpticalFlowImageFormatInfoNV, VkPhysicalDeviceExternalImageFormatInfo, VkPhysicalDeviceImageDrmFormatModifierInfoEXT, VkPhysicalDeviceImageViewImageFormatInfoEXT, or VkVideoProfileListInfoKHR
- VUID-VkPhysicalDeviceImageFormatInfo2-sType-unique
  The `sType` value of each struct in the `pNext` chain **must** be unique
- VUID-VkPhysicalDeviceImageFormatInfo2-format-parameter
  `format` **must** be a valid VkFormat value
- VUID-VkPhysicalDeviceImageFormatInfo2-type-parameter
  `type` **must** be a valid VkImageType value
- VUID-VkPhysicalDeviceImageFormatInfo2-tiling-parameter
  `tiling` **must** be a valid VkImageTiling value
- VUID-VkPhysicalDeviceImageFormatInfo2-usage-parameter
  `usage` **must** be a valid combination of VkImageUsageFlagBits values
- VUID-VkPhysicalDeviceImageFormatInfo2-usage-requiredbitmask
  `usage` **must** not be `0`
- VUID-VkPhysicalDeviceImageFormatInfo2-flags-parameter
  `flags` **must** be a valid combination of VkImageCreateFlagBits values

The `VkImageFormatProperties2` structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkImageFormatProperties2 {
    VkStructureType            sType;
    void*                      pNext;
    VkImageFormatProperties    imageFormatProperties;
} VkImageFormatProperties2;
```

or the equivalent

```
// Provided by VK_KHR_get_physical_device_properties2
typedef VkImageFormatProperties2 VkImageFormatProperties2KHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure. The `pNext` chain of

VkImageFormatProperties2 is used to allow the specification of additional capabilities to be returned from vkGetPhysicalDeviceImageFormatProperties2.

- imageFormatProperties is a VkImageFormatProperties structure in which capabilities are returned.

If the combination of parameters to vkGetPhysicalDeviceImageFormatProperties2 is not supported by the implementation for use in vkCreateImage, then all members of imageFormatProperties will be filled with zero.

**NOTE** Filling imageFormatProperties with zero for unsupported formats is an exception to the usual rule that output structures have undefined contents on error. This exception was unintentional, but is preserved for backwards compatibility. This exception only applies to imageFormatProperties, not sType, pNext, or any structures chained from pNext.

### Valid Usage (Implicit)

- VUID-VkImageFormatProperties2-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_IMAGE_FORMAT_PROPERTIES_2
- VUID-VkImageFormatProperties2-pNext-pNext
  Each pNext member of any structure (including this one) in the pNext chain **must** be either NULL or a pointer to a valid instance of VkAndroidHardwareBufferUsageANDROID, VkExternalImageFormatProperties, VkFilterCubicImageViewImageFormatPropertiesEXT, VkHostImageCopyDevicePerformanceQueryEXT, VkImageCompressionPropertiesEXT, VkSamplerYcbcrConversionImageFormatProperties, or VkTextureLODGatherFormatPropertiesAMD
- VUID-VkImageFormatProperties2-sType-unique
  The sType value of each struct in the pNext chain **must** be unique

To determine if texture gather functions that take explicit LOD and/or bias argument values **can** be used with a given image format, add a VkTextureLODGatherFormatPropertiesAMD structure to the pNext chain of the VkImageFormatProperties2 structure in a call to vkGetPhysicalDeviceImageFormatProperties2.

The VkTextureLODGatherFormatPropertiesAMD structure is defined as:

```
// Provided by VK_AMD_texture_gather_bias_lod
typedef struct VkTextureLODGatherFormatPropertiesAMD {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           supportsTextureGatherLODBiasAMD;
} VkTextureLODGatherFormatPropertiesAMD;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.

- supportsTextureGatherLODBiasAMD tells if the image format can be used with texture gather bias/LOD functions, as introduced by the VK_AMD_texture_gather_bias_lod extension. This field is set by the implementation. An application-specified value is ignored.

### Valid Usage (Implicit)

- VUID-VkTextureLODGatherFormatPropertiesAMD-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_TEXTURE_LOD_GATHER_FORMAT_PROPERTIES_AMD

To determine the image capabilities compatible with an external memory handle type, add a VkPhysicalDeviceExternalImageFormatInfo structure to the pNext chain of the VkPhysicalDeviceImageFormatInfo2 structure and a VkExternalImageFormatProperties structure to the pNext chain of the VkImageFormatProperties2 structure.

The VkPhysicalDeviceExternalImageFormatInfo structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkPhysicalDeviceExternalImageFormatInfo {
    VkStructureType                       sType;
    const void*                           pNext;
    VkExternalMemoryHandleTypeFlagBits    handleType;
} VkPhysicalDeviceExternalImageFormatInfo;
```

or the equivalent

```
// Provided by VK_KHR_external_memory_capabilities
typedef VkPhysicalDeviceExternalImageFormatInfo
VkPhysicalDeviceExternalImageFormatInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- handleType is a VkExternalMemoryHandleTypeFlagBits value specifying the memory handle type that will be used with the memory associated with the image.

If handleType is 0, vkGetPhysicalDeviceImageFormatProperties2 will behave as if VkPhysicalDeviceExternalImageFormatInfo was not present, and VkExternalImageFormatProperties will be ignored.

If handleType is not compatible with the format, type, tiling, usage, and flags specified in VkPhysicalDeviceImageFormatInfo2, then vkGetPhysicalDeviceImageFormatProperties2 returns VK_ERROR_FORMAT_NOT_SUPPORTED.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceExternalImageFormatInfo-sType-sType

`sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTERNAL_IMAGE_FORMAT_INFO`

- VUID-VkPhysicalDeviceExternalImageFormatInfo-handleType-parameter
  If `handleType` is not `0`, `handleType` **must** be a valid VkExternalMemoryHandleTypeFlagBits value

Possible values of VkPhysicalDeviceExternalImageFormatInfo::`handleType`, specifying an external memory handle type, are:

```
// Provided by VK_VERSION_1_1
typedef enum VkExternalMemoryHandleTypeFlagBits {
    VK_EXTERNAL_MEMORY_HANDLE_TYPE_OPAQUE_FD_BIT = 0x00000001,
    VK_EXTERNAL_MEMORY_HANDLE_TYPE_OPAQUE_WIN32_BIT = 0x00000002,
    VK_EXTERNAL_MEMORY_HANDLE_TYPE_OPAQUE_WIN32_KMT_BIT = 0x00000004,
    VK_EXTERNAL_MEMORY_HANDLE_TYPE_D3D11_TEXTURE_BIT = 0x00000008,
    VK_EXTERNAL_MEMORY_HANDLE_TYPE_D3D11_TEXTURE_KMT_BIT = 0x00000010,
    VK_EXTERNAL_MEMORY_HANDLE_TYPE_D3D12_HEAP_BIT = 0x00000020,
    VK_EXTERNAL_MEMORY_HANDLE_TYPE_D3D12_RESOURCE_BIT = 0x00000040,
  // Provided by VK_EXT_external_memory_dma_buf
    VK_EXTERNAL_MEMORY_HANDLE_TYPE_DMA_BUF_BIT_EXT = 0x00000200,
  // Provided by VK_ANDROID_external_memory_android_hardware_buffer
    VK_EXTERNAL_MEMORY_HANDLE_TYPE_ANDROID_HARDWARE_BUFFER_BIT_ANDROID = 0x00000400,
  // Provided by VK_EXT_external_memory_host
    VK_EXTERNAL_MEMORY_HANDLE_TYPE_HOST_ALLOCATION_BIT_EXT = 0x00000080,
  // Provided by VK_EXT_external_memory_host
    VK_EXTERNAL_MEMORY_HANDLE_TYPE_HOST_MAPPED_FOREIGN_MEMORY_BIT_EXT = 0x00000100,
  // Provided by VK_FUCHSIA_external_memory
    VK_EXTERNAL_MEMORY_HANDLE_TYPE_ZIRCON_VMO_BIT_FUCHSIA = 0x00000800,
  // Provided by VK_NV_external_memory_rdma
    VK_EXTERNAL_MEMORY_HANDLE_TYPE_RDMA_ADDRESS_BIT_NV = 0x00001000,
  // Provided by VK_QNX_external_memory_screen_buffer
    VK_EXTERNAL_MEMORY_HANDLE_TYPE_SCREEN_BUFFER_BIT_QNX = 0x00004000,
  // Provided by VK_KHR_external_memory_capabilities
    VK_EXTERNAL_MEMORY_HANDLE_TYPE_OPAQUE_FD_BIT_KHR =
VK_EXTERNAL_MEMORY_HANDLE_TYPE_OPAQUE_FD_BIT,
  // Provided by VK_KHR_external_memory_capabilities
    VK_EXTERNAL_MEMORY_HANDLE_TYPE_OPAQUE_WIN32_BIT_KHR =
VK_EXTERNAL_MEMORY_HANDLE_TYPE_OPAQUE_WIN32_BIT,
  // Provided by VK_KHR_external_memory_capabilities
    VK_EXTERNAL_MEMORY_HANDLE_TYPE_OPAQUE_WIN32_KMT_BIT_KHR =
VK_EXTERNAL_MEMORY_HANDLE_TYPE_OPAQUE_WIN32_KMT_BIT,
  // Provided by VK_KHR_external_memory_capabilities
    VK_EXTERNAL_MEMORY_HANDLE_TYPE_D3D11_TEXTURE_BIT_KHR =
VK_EXTERNAL_MEMORY_HANDLE_TYPE_D3D11_TEXTURE_BIT,
  // Provided by VK_KHR_external_memory_capabilities
    VK_EXTERNAL_MEMORY_HANDLE_TYPE_D3D11_TEXTURE_KMT_BIT_KHR =
VK_EXTERNAL_MEMORY_HANDLE_TYPE_D3D11_TEXTURE_KMT_BIT,
  // Provided by VK_KHR_external_memory_capabilities
    VK_EXTERNAL_MEMORY_HANDLE_TYPE_D3D12_HEAP_BIT_KHR =
VK_EXTERNAL_MEMORY_HANDLE_TYPE_D3D12_HEAP_BIT,
  // Provided by VK_KHR_external_memory_capabilities
    VK_EXTERNAL_MEMORY_HANDLE_TYPE_D3D12_RESOURCE_BIT_KHR =
VK_EXTERNAL_MEMORY_HANDLE_TYPE_D3D12_RESOURCE_BIT,
} VkExternalMemoryHandleTypeFlagBits;
```

or the equivalent

```
// Provided by VK_KHR_external_memory_capabilities
typedef VkExternalMemoryHandleTypeFlagBits VkExternalMemoryHandleTypeFlagBitsKHR;
```

- VK_EXTERNAL_MEMORY_HANDLE_TYPE_OPAQUE_FD_BIT specifies a POSIX file descriptor handle that has only limited valid usage outside of Vulkan and other compatible APIs. It **must** be compatible with the POSIX system calls dup, dup2, close, and the non-standard system call dup3. Additionally, it **must** be transportable over a socket using an SCM_RIGHTS control message. It owns a reference to the underlying memory resource represented by its Vulkan memory object.
- VK_EXTERNAL_MEMORY_HANDLE_TYPE_OPAQUE_WIN32_BIT specifies an NT handle that has only limited valid usage outside of Vulkan and other compatible APIs. It **must** be compatible with the functions DuplicateHandle, CloseHandle, CompareObjectHandles, GetHandleInformation, and SetHandleInformation. It owns a reference to the underlying memory resource represented by its Vulkan memory object.
- VK_EXTERNAL_MEMORY_HANDLE_TYPE_OPAQUE_WIN32_KMT_BIT specifies a global share handle that has only limited valid usage outside of Vulkan and other compatible APIs. It is not compatible with any native APIs. It does not own a reference to the underlying memory resource represented by its Vulkan memory object, and will therefore become invalid when all Vulkan memory objects associated with it are destroyed.
- VK_EXTERNAL_MEMORY_HANDLE_TYPE_D3D11_TEXTURE_BIT specifies an NT handle returned by IDXGIResource1::CreateSharedHandle referring to a Direct3D 10 or 11 texture resource. It owns a reference to the memory used by the Direct3D resource.
- VK_EXTERNAL_MEMORY_HANDLE_TYPE_D3D11_TEXTURE_KMT_BIT specifies a global share handle returned by IDXGIResource::GetSharedHandle referring to a Direct3D 10 or 11 texture resource. It does not own a reference to the underlying Direct3D resource, and will therefore become invalid when all Vulkan memory objects and Direct3D resources associated with it are destroyed.
- VK_EXTERNAL_MEMORY_HANDLE_TYPE_D3D12_HEAP_BIT specifies an NT handle returned by ID3D12Device::CreateSharedHandle referring to a Direct3D 12 heap resource. It owns a reference to the resources used by the Direct3D heap.
- VK_EXTERNAL_MEMORY_HANDLE_TYPE_D3D12_RESOURCE_BIT specifies an NT handle returned by ID3D12Device::CreateSharedHandle referring to a Direct3D 12 committed resource. It owns a reference to the memory used by the Direct3D resource.
- VK_EXTERNAL_MEMORY_HANDLE_TYPE_HOST_ALLOCATION_BIT_EXT specifies a host pointer returned by a host memory allocation command. It does not own a reference to the underlying memory resource, and will therefore become invalid if the host memory is freed.
- VK_EXTERNAL_MEMORY_HANDLE_TYPE_HOST_MAPPED_FOREIGN_MEMORY_BIT_EXT specifies a host pointer to *host mapped foreign memory*. It does not own a reference to the underlying memory resource, and will therefore become invalid if the foreign memory is unmapped or otherwise becomes no longer available.
- VK_EXTERNAL_MEMORY_HANDLE_TYPE_DMA_BUF_BIT_EXT is a file descriptor for a Linux dma_buf. It owns a reference to the underlying memory resource represented by its Vulkan memory object.
- VK_EXTERNAL_MEMORY_HANDLE_TYPE_ANDROID_HARDWARE_BUFFER_BIT_ANDROID specifies an AHardwareBuffer object defined by the Android NDK. See Android Hardware Buffers for more details of this handle type.

- `VK_EXTERNAL_MEMORY_HANDLE_TYPE_ZIRCON_VMO_BIT_FUCHSIA` is a Zircon handle to a virtual memory object.
- `VK_EXTERNAL_MEMORY_HANDLE_TYPE_RDMA_ADDRESS_BIT_NV` is a handle to an allocation accessible by remote devices. It owns a reference to the underlying memory resource represented by its Vulkan memory object.
- `VK_EXTERNAL_MEMORY_HANDLE_TYPE_SCREEN_BUFFER_BIT_QNX` specifies a `_screen_buffer` object defined by the QNX SDP. See QNX Screen Buffer for more details of this handle type.

Some external memory handle types can only be shared within the same underlying physical device and/or the same driver version, as defined in the following table:

*Table 91. External memory handle types compatibility*

| Handle type | VkPhysicalDeviceIDProperties::driverUUID | VkPhysicalDeviceIDProperties::deviceUUID |
|---|---|---|
| VK_EXTERNAL_MEMORY_HANDLE_TYPE_OPAQUE_FD_BIT | Must match | Must match |
| VK_EXTERNAL_MEMORY_HANDLE_TYPE_OPAQUE_WIN32_BIT | Must match | Must match |
| VK_EXTERNAL_MEMORY_HANDLE_TYPE_OPAQUE_WIN32_KMT_BIT | Must match | Must match |
| VK_EXTERNAL_MEMORY_HANDLE_TYPE_D3D11_TEXTURE_BIT | Must match | Must match |
| VK_EXTERNAL_MEMORY_HANDLE_TYPE_D3D11_TEXTURE_KMT_BIT | Must match | Must match |
| VK_EXTERNAL_MEMORY_HANDLE_TYPE_D3D12_HEAP_BIT | Must match | Must match |
| VK_EXTERNAL_MEMORY_HANDLE_TYPE_D3D12_RESOURCE_BIT | Must match | Must match |
| VK_EXTERNAL_MEMORY_HANDLE_TYPE_HOST_ALLOCATION_BIT_EXT | No restriction | No restriction |
| VK_EXTERNAL_MEMORY_HANDLE_TYPE_HOST_MAPPED_FOREIGN_MEMORY_BIT_EXT | No restriction | No restriction |
| VK_EXTERNAL_MEMORY_HANDLE_TYPE_DMA_BUF_BIT_EXT | No restriction | No restriction |
| VK_EXTERNAL_MEMORY_HANDLE_TYPE_ANDROID_HARDWARE_BUFFER_BIT_ANDROID | No restriction | No restriction |
| VK_EXTERNAL_MEMORY_HANDLE_TYPE_ZIRCON_VMO_BIT_FUCHSIA | No restriction | No restriction |
| VK_EXTERNAL_MEMORY_HANDLE_TYPE_RDMA_ADDRESS_BIT_NV | No restriction | No restriction |
| VK_EXTERNAL_MEMORY_HANDLE_TYPE_SCREEN_BUFFER_BIT_QNX | No restriction | No restriction |

**NOTE** The above table does not restrict the drivers and devices with which VK_EXTERNAL_MEMORY_HANDLE_TYPE_HOST_ALLOCATION_BIT_EXT and VK_EXTERNAL_MEMORY_HANDLE_TYPE_HOST_MAPPED_FOREIGN_MEMORY_BIT_EXT **may** be shared, as these handle types inherently mean memory that does not come from the same device, as they import memory from the host or a foreign device, respectively.

**NOTE** Even though the above table does not restrict the drivers and devices with which VK_EXTERNAL_MEMORY_HANDLE_TYPE_DMA_BUF_BIT_EXT **may** be shared, query mechanisms exist in the Vulkan API that prevent the import of incompatible dma-bufs (such as vkGetMemoryFdPropertiesKHR) and that prevent incompatible usage of dma-bufs

(such as VkPhysicalDeviceExternalBufferInfo and VkPhysicalDeviceExternalImageFormatInfo).

```
// Provided by VK_VERSION_1_1
typedef VkFlags VkExternalMemoryHandleTypeFlags;
```

or the equivalent

```
// Provided by VK_KHR_external_memory_capabilities
typedef VkExternalMemoryHandleTypeFlags VkExternalMemoryHandleTypeFlagsKHR;
```

`VkExternalMemoryHandleTypeFlags` is a bitmask type for setting a mask of zero or more VkExternalMemoryHandleTypeFlagBits.

The `VkExternalImageFormatProperties` structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkExternalImageFormatProperties {
    VkStructureType                 sType;
    void*                           pNext;
    VkExternalMemoryProperties      externalMemoryProperties;
} VkExternalImageFormatProperties;
```

or the equivalent

```
// Provided by VK_KHR_external_memory_capabilities
typedef VkExternalImageFormatProperties VkExternalImageFormatPropertiesKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `externalMemoryProperties` is a VkExternalMemoryProperties structure specifying various capabilities of the external handle type when used with the specified image creation parameters.

### Valid Usage (Implicit)

- VUID-VkExternalImageFormatProperties-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_EXTERNAL_IMAGE_FORMAT_PROPERTIES`

The `VkExternalMemoryProperties` structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkExternalMemoryProperties {
    VkExternalMemoryFeatureFlags       externalMemoryFeatures;
    VkExternalMemoryHandleTypeFlags    exportFromImportedHandleTypes;
    VkExternalMemoryHandleTypeFlags    compatibleHandleTypes;
} VkExternalMemoryProperties;
```

or the equivalent

```
// Provided by VK_KHR_external_memory_capabilities
typedef VkExternalMemoryProperties VkExternalMemoryPropertiesKHR;
```

- `externalMemoryFeatures` is a bitmask of VkExternalMemoryFeatureFlagBits specifying the features of `handleType`.
- `exportFromImportedHandleTypes` is a bitmask of VkExternalMemoryHandleTypeFlagBits specifying which types of imported handle `handleType` **can** be exported from.
- `compatibleHandleTypes` is a bitmask of VkExternalMemoryHandleTypeFlagBits specifying handle types which **can** be specified at the same time as `handleType` when creating an image compatible with external memory.

`compatibleHandleTypes` **must** include at least `handleType`. Inclusion of a handle type in `compatibleHandleTypes` does not imply the values returned in VkImageFormatProperties2 will be the same when VkPhysicalDeviceExternalImageFormatInfo::`handleType` is set to that type. The application is responsible for querying the capabilities of all handle types intended for concurrent use in a single image and intersecting them to obtain the compatible set of capabilities.

Bits which **may** be set in VkExternalMemoryProperties::`externalMemoryFeatures`, specifying features of an external memory handle type, are:

```
// Provided by VK_VERSION_1_1
typedef enum VkExternalMemoryFeatureFlagBits {
    VK_EXTERNAL_MEMORY_FEATURE_DEDICATED_ONLY_BIT = 0x00000001,
    VK_EXTERNAL_MEMORY_FEATURE_EXPORTABLE_BIT = 0x00000002,
    VK_EXTERNAL_MEMORY_FEATURE_IMPORTABLE_BIT = 0x00000004,
  // Provided by VK_KHR_external_memory_capabilities
    VK_EXTERNAL_MEMORY_FEATURE_DEDICATED_ONLY_BIT_KHR =
VK_EXTERNAL_MEMORY_FEATURE_DEDICATED_ONLY_BIT,
  // Provided by VK_KHR_external_memory_capabilities
    VK_EXTERNAL_MEMORY_FEATURE_EXPORTABLE_BIT_KHR =
VK_EXTERNAL_MEMORY_FEATURE_EXPORTABLE_BIT,
  // Provided by VK_KHR_external_memory_capabilities
    VK_EXTERNAL_MEMORY_FEATURE_IMPORTABLE_BIT_KHR =
VK_EXTERNAL_MEMORY_FEATURE_IMPORTABLE_BIT,
} VkExternalMemoryFeatureFlagBits;
```

or the equivalent

```
// Provided by VK_KHR_external_memory_capabilities
typedef VkExternalMemoryFeatureFlagBits VkExternalMemoryFeatureFlagBitsKHR;
```

- VK_EXTERNAL_MEMORY_FEATURE_DEDICATED_ONLY_BIT specifies that images or buffers created with the specified parameters and handle type **must** use the mechanisms defined by VkMemoryDedicatedRequirements and VkMemoryDedicatedAllocateInfo to create (or import) a dedicated allocation for the image or buffer.
- VK_EXTERNAL_MEMORY_FEATURE_EXPORTABLE_BIT specifies that handles of this type **can** be exported from Vulkan memory objects.
- VK_EXTERNAL_MEMORY_FEATURE_IMPORTABLE_BIT specifies that handles of this type **can** be imported as Vulkan memory objects.

Because their semantics in external APIs roughly align with that of an image or buffer with a dedicated allocation in Vulkan, implementations are **required** to report VK_EXTERNAL_MEMORY_FEATURE_DEDICATED_ONLY_BIT for the following external handle types:

- VK_EXTERNAL_MEMORY_HANDLE_TYPE_D3D11_TEXTURE_BIT
- VK_EXTERNAL_MEMORY_HANDLE_TYPE_D3D11_TEXTURE_KMT_BIT
- VK_EXTERNAL_MEMORY_HANDLE_TYPE_D3D12_RESOURCE_BIT
- VK_EXTERNAL_MEMORY_HANDLE_TYPE_ANDROID_HARDWARE_BUFFER_BIT_ANDROID for images only
- VK_EXTERNAL_MEMORY_HANDLE_TYPE_SCREEN_BUFFER_BIT_QNX for images only

Implementations **must** not report VK_EXTERNAL_MEMORY_FEATURE_DEDICATED_ONLY_BIT for buffers with external handle type VK_EXTERNAL_MEMORY_HANDLE_TYPE_ANDROID_HARDWARE_BUFFER_BIT_ANDROID. Implementations **must** not report VK_EXTERNAL_MEMORY_FEATURE_DEDICATED_ONLY_BIT for buffers with external handle type VK_EXTERNAL_MEMORY_HANDLE_TYPE_SCREEN_BUFFER_BIT_QNX. Implementations **must** not report VK_EXTERNAL_MEMORY_FEATURE_DEDICATED_ONLY_BIT for images or buffers with external handle type VK_EXTERNAL_MEMORY_HANDLE_TYPE_HOST_ALLOCATION_BIT_EXT, or VK_EXTERNAL_MEMORY_HANDLE_TYPE_HOST_MAPPED_FOREIGN_MEMORY_BIT_EXT.

```
// Provided by VK_VERSION_1_1
typedef VkFlags VkExternalMemoryFeatureFlags;
```

or the equivalent

```
// Provided by VK_KHR_external_memory_capabilities
typedef VkExternalMemoryFeatureFlags VkExternalMemoryFeatureFlagsKHR;
```

VkExternalMemoryFeatureFlags is a bitmask type for setting a mask of zero or more VkExternalMemoryFeatureFlagBits.

To query the image capabilities that are compatible with a Linux DRM format modifier, set VkPhysicalDeviceImageFormatInfo2::tiling to VK_IMAGE_TILING_DRM_FORMAT_MODIFIER_EXT and add a VkPhysicalDeviceImageDrmFormatModifierInfoEXT structure to the pNext chain of

VkPhysicalDeviceImageFormatInfo2.

The VkPhysicalDeviceImageDrmFormatModifierInfoEXT structure is defined as:

```
// Provided by VK_EXT_image_drm_format_modifier
typedef struct VkPhysicalDeviceImageDrmFormatModifierInfoEXT {
    VkStructureType    sType;
    const void*        pNext;
    uint64_t           drmFormatModifier;
    VkSharingMode      sharingMode;
    uint32_t           queueFamilyIndexCount;
    const uint32_t*    pQueueFamilyIndices;
} VkPhysicalDeviceImageDrmFormatModifierInfoEXT;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- drmFormatModifier is the image's *Linux DRM format modifier*, corresponding to VkImageDrmFormatModifierExplicitCreateInfoEXT::modifier or to VkImageDrmFormatModifierListCreateInfoEXT::pModifiers.
- sharingMode specifies how the image will be accessed by multiple queue families.
- queueFamilyIndexCount is the number of entries in the pQueueFamilyIndices array.
- pQueueFamilyIndices is a pointer to an array of queue families that will access the image. It is ignored if sharingMode is not VK_SHARING_MODE_CONCURRENT.

If the drmFormatModifier is incompatible with the parameters specified in VkPhysicalDeviceImageFormatInfo2 and its pNext chain, then vkGetPhysicalDeviceImageFormatProperties2 returns VK_ERROR_FORMAT_NOT_SUPPORTED. The implementation **must** support the query of any drmFormatModifier, including unknown and invalid modifier values.

### Valid Usage

- VUID-VkPhysicalDeviceImageDrmFormatModifierInfoEXT-sharingMode-02314
  If sharingMode is VK_SHARING_MODE_CONCURRENT, then pQueueFamilyIndices **must** be a valid pointer to an array of queueFamilyIndexCount uint32_t values
- VUID-VkPhysicalDeviceImageDrmFormatModifierInfoEXT-sharingMode-02315
  If sharingMode is VK_SHARING_MODE_CONCURRENT, then queueFamilyIndexCount **must** be greater than 1
- VUID-VkPhysicalDeviceImageDrmFormatModifierInfoEXT-sharingMode-02316
  If sharingMode is VK_SHARING_MODE_CONCURRENT, each element of pQueueFamilyIndices **must** be unique and **must** be less than the pQueueFamilyPropertyCount returned by vkGetPhysicalDeviceQueueFamilyProperties2 for the physicalDevice that was used to create device

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceImageDrmFormatModifierInfoEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_IMAGE_DRM_FORMAT_MODIFIER_INFO_EXT`
- VUID-VkPhysicalDeviceImageDrmFormatModifierInfoEXT-sharingMode-parameter
  `sharingMode` **must** be a valid VkSharingMode value

To determine the number of combined image samplers required to support a multi-planar format, add VkSamplerYcbcrConversionImageFormatProperties to the `pNext` chain of the VkImageFormatProperties2 structure in a call to `vkGetPhysicalDeviceImageFormatProperties2`.

The `VkSamplerYcbcrConversionImageFormatProperties` structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkSamplerYcbcrConversionImageFormatProperties {
    VkStructureType    sType;
    void*              pNext;
    uint32_t           combinedImageSamplerDescriptorCount;
} VkSamplerYcbcrConversionImageFormatProperties;
```

or the equivalent

```
// Provided by VK_KHR_sampler_ycbcr_conversion
typedef VkSamplerYcbcrConversionImageFormatProperties
VkSamplerYcbcrConversionImageFormatPropertiesKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `combinedImageSamplerDescriptorCount` is the number of combined image sampler descriptors that the implementation uses to access the format.

### Valid Usage (Implicit)

- VUID-VkSamplerYcbcrConversionImageFormatProperties-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_SAMPLER_YCBCR_CONVERSION_IMAGE_FORMAT_PROPERTIES`

`combinedImageSamplerDescriptorCount` is a number between 1 and the number of planes in the format. A descriptor set layout binding with immutable $Y'C_BC_R$ conversion samplers will have a maximum `combinedImageSamplerDescriptorCount` which is the maximum across all formats supported by its samplers of the `combinedImageSamplerDescriptorCount` for each format. Descriptor sets with that layout will internally use that maximum `combinedImageSamplerDescriptorCount` descriptors for each descriptor in the binding. This expanded number of descriptors will be consumed from the descriptor pool when a descriptor set is allocated, and counts towards the

maxDescriptorSetSamplers, maxDescriptorSetSampledImages, maxPerStageDescriptorSamplers, and maxPerStageDescriptorSampledImages limits.

**NOTE**

All descriptors in a binding use the same maximum combinedImageSamplerDescriptorCount descriptors to allow implementations to use a uniform stride for dynamic indexing of the descriptors in the binding.

For example, consider a descriptor set layout binding with two descriptors and immutable samplers for multi-planar formats that have VkSamplerYcbcrConversionImageFormatProperties::combinedImageSamplerDescriptorCount values of 2 and 3 respectively. There are two descriptors in the binding and the maximum combinedImageSamplerDescriptorCount is 3, so descriptor sets with this layout consume 6 descriptors from the descriptor pool. To create a descriptor pool that allows allocating four descriptor sets with this layout, descriptorCount must be at least 24.

Instead of querying all the potential formats that the application might use in the descriptor layout, the application **can** use the VkPhysicalDeviceMaintenance6PropertiesKHR::maxCombinedImageSamplerDescriptorCount property to determine the maximum descriptor size that will accommodate any and all formats that require a sampler $Y'C_BC_R$ conversion supported by the implementation.

To obtain optimal Android hardware buffer usage flags for specific image creation parameters, add a VkAndroidHardwareBufferUsageANDROID structure to the pNext chain of a VkImageFormatProperties2 structure passed to vkGetPhysicalDeviceImageFormatProperties2. This structure is defined as:

```
// Provided by VK_ANDROID_external_memory_android_hardware_buffer
typedef struct VkAndroidHardwareBufferUsageANDROID {
    VkStructureType    sType;
    void*              pNext;
    uint64_t           androidHardwareBufferUsage;
} VkAndroidHardwareBufferUsageANDROID;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- androidHardwareBufferUsage returns the Android hardware buffer usage flags.

The androidHardwareBufferUsage field **must** include Android hardware buffer usage flags listed in the AHardwareBuffer Usage Equivalence table when the corresponding Vulkan image usage or image creation flags are included in the usage or flags fields of VkPhysicalDeviceImageFormatInfo2. It **must** include at least one GPU usage flag (AHARDWAREBUFFER_USAGE_GPU_*), even if none of the corresponding Vulkan usages or flags are requested.

**NOTE**

Requiring at least one GPU usage flag ensures that Android hardware buffer memory will be allocated in a memory pool accessible to the Vulkan implementation, and that specializing the memory layout based on usage flags does

not prevent it from being compatible with Vulkan. Implementations **may** avoid unnecessary restrictions caused by this requirement by using vendor usage flags to indicate that only the Vulkan uses indicated in VkImageFormatProperties2 are required.

### Valid Usage (Implicit)

- VUID-VkAndroidHardwareBufferUsageANDROID-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_ANDROID_HARDWARE_BUFFER_USAGE_ANDROID`

To query if using `VK_IMAGE_USAGE_HOST_TRANSFER_BIT_EXT` has a negative impact on device performance when accessing an image, add `VK_IMAGE_USAGE_HOST_TRANSFER_BIT_EXT` to VkPhysicalDeviceImageFormatInfo2::`usage`, and add a `VkHostImageCopyDevicePerformanceQueryEXT` structure to the `pNext` chain of a VkImageFormatProperties2 structure passed to vkGetPhysicalDeviceImageFormatProperties2. This structure is defined as:

```
// Provided by VK_EXT_host_image_copy
typedef struct VkHostImageCopyDevicePerformanceQueryEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           optimalDeviceAccess;
    VkBool32           identicalMemoryLayout;
} VkHostImageCopyDevicePerformanceQueryEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `optimalDeviceAccess` returns `VK_TRUE` if use of host image copy has no adverse effect on device access performance, compared to an image that is created with exact same creation parameters, and bound to the same VkDeviceMemory, except that `VK_IMAGE_USAGE_HOST_TRANSFER_BIT_EXT` is replaced with `VK_IMAGE_USAGE_TRANSFER_SRC_BIT` and `VK_IMAGE_USAGE_TRANSFER_DST_BIT`.
- `identicalMemoryLayout` returns `VK_TRUE` if use of host image copy has no impact on memory layout compared to an image that is created with exact same creation parameters, and bound to the same VkDeviceMemory, except that `VK_IMAGE_USAGE_HOST_TRANSFER_BIT_EXT` is replaced with `VK_IMAGE_USAGE_TRANSFER_SRC_BIT` and `VK_IMAGE_USAGE_TRANSFER_DST_BIT`.

The implementation **may** return `VK_FALSE` in `optimalDeviceAccess` if `identicalMemoryLayout` is `VK_FALSE`. If `identicalMemoryLayout` is `VK_TRUE`, `optimalDeviceAccess` **must** be `VK_TRUE`.

The implementation **may** return `VK_TRUE` in `optimalDeviceAccess` while `identicalMemoryLayout` is `VK_FALSE`. In this situation, any device performance impact **should** not be measurable.

If VkPhysicalDeviceImageFormatInfo2::`format` is a block-compressed format and vkGetPhysicalDeviceImageFormatProperties2 returns `VK_SUCCESS`, the implementation **must** return `VK_TRUE` in `optimalDeviceAccess`.

**NOTE** Applications can make use of `optimalDeviceAccess` to determine their resource

copying strategy. If a resource is expected to be accessed more on device than on the host, and the implementation considers the resource sub-optimally accessed, it is likely better to use device copies instead.

**NOTE** Layout not being identical yet still considered optimal for device access could happen if the implementation has different memory layout patterns, some of which are easier to access on the host.

**NOTE** The most practical reason for `optimalDeviceAccess` to be `VK_FALSE` is that host image access may disable framebuffer compression where it would otherwise have been enabled. This represents far more efficient host image access since no compression algorithm is required to read or write to the image, but it would impact device access performance. Some implementations may only set `optimalDeviceAccess` to `VK_FALSE` if certain conditions are met, such as specific image usage flags or creation flags.

### Valid Usage (Implicit)

- VUID-VkHostImageCopyDevicePerformanceQueryEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_HOST_IMAGE_COPY_DEVICE_PERFORMANCE_QUERY_EXT`

To determine if cubic filtering can be used with a given image format and a given image view type add a VkPhysicalDeviceImageViewImageFormatInfoEXT structure to the `pNext` chain of the VkPhysicalDeviceImageFormatInfo2 structure, and a VkFilterCubicImageViewImageFormatPropertiesEXT structure to the `pNext` chain of the VkImageFormatProperties2 structure.

The `VkPhysicalDeviceImageViewImageFormatInfoEXT` structure is defined as:

```
// Provided by VK_EXT_filter_cubic
typedef struct VkPhysicalDeviceImageViewImageFormatInfoEXT {
    VkStructureType    sType;
    void*              pNext;
    VkImageViewType    imageViewType;
} VkPhysicalDeviceImageViewImageFormatInfoEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `imageViewType` is a VkImageViewType value specifying the type of the image view.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceImageViewImageFormatInfoEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_IMAGE_VIEW_IMAGE_FORMAT_INFO_EXT`

- VUID-VkPhysicalDeviceImageViewImageFormatInfoEXT-imageViewType-parameter
  `imageViewType` **must** be a valid VkImageViewType value

The `VkFilterCubicImageViewImageFormatPropertiesEXT` structure is defined as:

```
// Provided by VK_EXT_filter_cubic
typedef struct VkFilterCubicImageViewImageFormatPropertiesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           filterCubic;
    VkBool32           filterCubicMinmax;
} VkFilterCubicImageViewImageFormatPropertiesEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `filterCubic` tells if image format, image type and image view type **can** be used with cubic filtering. This field is set by the implementation. An application-specified value is ignored.
- `filterCubicMinmax` tells if image format, image type and image view type **can** be used with cubic filtering and minmax filtering. This field is set by the implementation. An application-specified value is ignored.

### Valid Usage (Implicit)

- VUID-VkFilterCubicImageViewImageFormatPropertiesEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_FILTER_CUBIC_IMAGE_VIEW_IMAGE_FORMAT_PROPERTIES_EXT`

### Valid Usage

- VUID-VkFilterCubicImageViewImageFormatPropertiesEXT-pNext-02627
  If the `pNext` chain of the VkImageFormatProperties2 structure includes a VkFilterCubicImageViewImageFormatPropertiesEXT structure, the `pNext` chain of the VkPhysicalDeviceImageFormatInfo2 structure **must** include a VkPhysicalDeviceImageViewImageFormatInfoEXT structure with an `imageViewType` that is compatible with `imageType`

## 49.1.1. Supported Sample Counts

`vkGetPhysicalDeviceImageFormatProperties` returns a bitmask of VkSampleCountFlagBits in `sampleCounts` specifying the supported sample counts for the image parameters.

`sampleCounts` will be set to `VK_SAMPLE_COUNT_1_BIT` if at least one of the following conditions is true:

- `tiling` is `VK_IMAGE_TILING_LINEAR`
- `type` is not `VK_IMAGE_TYPE_2D`

- flags contains VK_IMAGE_CREATE_CUBE_COMPATIBLE_BIT
- Neither the VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BIT flag nor the VK_FORMAT_FEATURE_DEPTH_STENCIL_ATTACHMENT_BIT flag in VkFormatProperties::optimalTilingFeatures returned by vkGetPhysicalDeviceFormatProperties is set
- VkPhysicalDeviceExternalImageFormatInfo::handleType is an external handle type for which multisampled image support is not required.
- format is one of the formats that require a sampler $Y'C_BC_R$ conversion
- usage contains VK_IMAGE_USAGE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR
- usage contains VK_IMAGE_USAGE_FRAGMENT_DENSITY_MAP_BIT_EXT

Otherwise, the bits set in sampleCounts will be the sample counts supported for the specified values of usage and format. For each bit set in usage, the supported sample counts relate to the limits in VkPhysicalDeviceLimits as follows:

- If usage includes VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT and format is a floating- or fixed-point color format, a superset of VkPhysicalDeviceLimits::framebufferColorSampleCounts
- If usage includes VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT, and format includes a depth component, a superset of VkPhysicalDeviceLimits::framebufferDepthSampleCounts
- If usage includes VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT, and format includes a stencil component, a superset of VkPhysicalDeviceLimits::framebufferStencilSampleCounts
- If usage includes VK_IMAGE_USAGE_SAMPLED_BIT, and format includes a color component, a superset of VkPhysicalDeviceLimits::sampledImageColorSampleCounts
- If usage includes VK_IMAGE_USAGE_SAMPLED_BIT, and format includes a depth component, a superset of VkPhysicalDeviceLimits::sampledImageDepthSampleCounts
- If usage includes VK_IMAGE_USAGE_SAMPLED_BIT, and format is an integer format, a superset of VkPhysicalDeviceLimits::sampledImageIntegerSampleCounts
- If usage includes VK_IMAGE_USAGE_STORAGE_BIT, a superset of VkPhysicalDeviceLimits::storageImageSampleCounts

If multiple bits are set in usage, sampleCounts will be the intersection of the per-usage values described above.

If none of the bits described above are set in usage, then there is no corresponding limit in VkPhysicalDeviceLimits. In this case, sampleCounts **must** include at least VK_SAMPLE_COUNT_1_BIT.

### 49.1.2. Allowed Extent Values Based on Image Type

Implementations **may** support extent values larger than the required minimum/maximum values for certain types of images. VkImageFormatProperties::maxExtent for each type is subject to the constraints below.

| | |
|---|---|
| **NOTE** | Implementations **must** support images with dimensions up to the required minimum/maximum values for all types of images. It follows that the query for additional capabilities **must** return extent values that are at least as large as the |

required values.

For VK_IMAGE_TYPE_1D:

- maxExtent.width ≥ VkPhysicalDeviceLimits::maxImageDimension1D
- maxExtent.height = 1
- maxExtent.depth = 1

For VK_IMAGE_TYPE_2D when flags does not contain VK_IMAGE_CREATE_CUBE_COMPATIBLE_BIT:

- maxExtent.width ≥ VkPhysicalDeviceLimits::maxImageDimension2D
- maxExtent.height ≥ VkPhysicalDeviceLimits::maxImageDimension2D
- maxExtent.depth = 1

For VK_IMAGE_TYPE_2D when flags contains VK_IMAGE_CREATE_CUBE_COMPATIBLE_BIT:

- maxExtent.width ≥ VkPhysicalDeviceLimits::maxImageDimensionCube
- maxExtent.height ≥ VkPhysicalDeviceLimits::maxImageDimensionCube
- maxExtent.depth = 1

For VK_IMAGE_TYPE_3D:

- maxExtent.width ≥ VkPhysicalDeviceLimits::maxImageDimension3D
- maxExtent.height ≥ VkPhysicalDeviceLimits::maxImageDimension3D
- maxExtent.depth ≥ VkPhysicalDeviceLimits::maxImageDimension3D

## 49.2. Additional Buffer Capabilities

To query the external handle types supported by buffers, call:

```
// Provided by VK_VERSION_1_1
void vkGetPhysicalDeviceExternalBufferProperties(
    VkPhysicalDevice                            physicalDevice,
    const VkPhysicalDeviceExternalBufferInfo*   pExternalBufferInfo,
    VkExternalBufferProperties*                 pExternalBufferProperties);
```

or the equivalent command

```
// Provided by VK_KHR_external_memory_capabilities
void vkGetPhysicalDeviceExternalBufferPropertiesKHR(
    VkPhysicalDevice                            physicalDevice,
    const VkPhysicalDeviceExternalBufferInfo*   pExternalBufferInfo,
    VkExternalBufferProperties*                 pExternalBufferProperties);
```

- physicalDevice is the physical device from which to query the buffer capabilities.

- pExternalBufferInfo is a pointer to a VkPhysicalDeviceExternalBufferInfo structure describing the parameters that would be consumed by vkCreateBuffer.
- pExternalBufferProperties is a pointer to a VkExternalBufferProperties structure in which capabilities are returned.

### Valid Usage (Implicit)

- VUID-vkGetPhysicalDeviceExternalBufferProperties-physicalDevice-parameter
  physicalDevice **must** be a valid VkPhysicalDevice handle
- VUID-vkGetPhysicalDeviceExternalBufferProperties-pExternalBufferInfo-parameter
  pExternalBufferInfo **must** be a valid pointer to a valid VkPhysicalDeviceExternalBufferInfo structure
- VUID-vkGetPhysicalDeviceExternalBufferProperties-pExternalBufferProperties-parameter
  pExternalBufferProperties **must** be a valid pointer to a VkExternalBufferProperties structure

The VkPhysicalDeviceExternalBufferInfo structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkPhysicalDeviceExternalBufferInfo {
    VkStructureType                       sType;
    const void*                           pNext;
    VkBufferCreateFlags                   flags;
    VkBufferUsageFlags                    usage;
    VkExternalMemoryHandleTypeFlagBits    handleType;
} VkPhysicalDeviceExternalBufferInfo;
```

or the equivalent

```
// Provided by VK_KHR_external_memory_capabilities
typedef VkPhysicalDeviceExternalBufferInfo VkPhysicalDeviceExternalBufferInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- flags is a bitmask of VkBufferCreateFlagBits describing additional parameters of the buffer, corresponding to VkBufferCreateInfo::flags.
- usage is a bitmask of VkBufferUsageFlagBits describing the intended usage of the buffer, corresponding to VkBufferCreateInfo::usage.
- handleType is a VkExternalMemoryHandleTypeFlagBits value specifying the memory handle type that will be used with the memory associated with the buffer.

Only usage flags representable in VkBufferUsageFlagBits are returned in this structure's usage. If

the pNext chain includes a VkBufferUsageFlags2CreateInfoKHR structure, all usage flags of the buffer are returned in VkBufferUsageFlags2CreateInfoKHR::usage.

### Valid Usage

- VUID-VkPhysicalDeviceExternalBufferInfo-None-09499
  If the pNext chain does not include a VkBufferUsageFlags2CreateInfoKHR structure, usage **must** be a valid combination of VkBufferUsageFlagBits values
- VUID-VkPhysicalDeviceExternalBufferInfo-None-09500
  If the pNext chain does not include a VkBufferUsageFlags2CreateInfoKHR structure, usage **must** not be 0

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceExternalBufferInfo-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTERNAL_BUFFER_INFO
- VUID-VkPhysicalDeviceExternalBufferInfo-pNext-pNext
  pNext **must** be NULL or a pointer to a valid instance of VkBufferUsageFlags2CreateInfoKHR
- VUID-VkPhysicalDeviceExternalBufferInfo-sType-unique
  The sType value of each struct in the pNext chain **must** be unique
- VUID-VkPhysicalDeviceExternalBufferInfo-flags-parameter
  flags **must** be a valid combination of VkBufferCreateFlagBits values
- VUID-VkPhysicalDeviceExternalBufferInfo-handleType-parameter
  handleType **must** be a valid VkExternalMemoryHandleTypeFlagBits value

The VkExternalBufferProperties structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkExternalBufferProperties {
    VkStructureType                 sType;
    void*                           pNext;
    VkExternalMemoryProperties      externalMemoryProperties;
} VkExternalBufferProperties;
```

or the equivalent

```
// Provided by VK_KHR_external_memory_capabilities
typedef VkExternalBufferProperties VkExternalBufferPropertiesKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- externalMemoryProperties is a VkExternalMemoryProperties structure specifying various

capabilities of the external handle type when used with the specified buffer creation parameters.

**Valid Usage (Implicit)**

- VUID-VkExternalBufferProperties-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_EXTERNAL_BUFFER_PROPERTIES`
- VUID-VkExternalBufferProperties-pNext-pNext
  `pNext` **must** be `NULL`

## 49.3. Optional Semaphore Capabilities

Semaphores **may** support import and export of their payload to external handles. To query the external handle types supported by semaphores, call:

```
// Provided by VK_VERSION_1_1
void vkGetPhysicalDeviceExternalSemaphoreProperties(
    VkPhysicalDevice                            physicalDevice,
    const VkPhysicalDeviceExternalSemaphoreInfo* pExternalSemaphoreInfo,
    VkExternalSemaphoreProperties*              pExternalSemaphoreProperties);
```

or the equivalent command

```
// Provided by VK_KHR_external_semaphore_capabilities
void vkGetPhysicalDeviceExternalSemaphorePropertiesKHR(
    VkPhysicalDevice                            physicalDevice,
    const VkPhysicalDeviceExternalSemaphoreInfo* pExternalSemaphoreInfo,
    VkExternalSemaphoreProperties*              pExternalSemaphoreProperties);
```

- `physicalDevice` is the physical device from which to query the semaphore capabilities.
- `pExternalSemaphoreInfo` is a pointer to a VkPhysicalDeviceExternalSemaphoreInfo structure describing the parameters that would be consumed by vkCreateSemaphore.
- `pExternalSemaphoreProperties` is a pointer to a VkExternalSemaphoreProperties structure in which capabilities are returned.

**Valid Usage (Implicit)**

- VUID-vkGetPhysicalDeviceExternalSemaphoreProperties-physicalDevice-parameter
  `physicalDevice` **must** be a valid VkPhysicalDevice handle
- VUID-vkGetPhysicalDeviceExternalSemaphoreProperties-pExternalSemaphoreInfo-parameter
  `pExternalSemaphoreInfo` **must** be a valid pointer to a valid VkPhysicalDeviceExternalSemaphoreInfo structure

- VUID-vkGetPhysicalDeviceExternalSemaphoreProperties-pExternalSemaphoreProperties-parameter
  pExternalSemaphoreProperties **must** be a valid pointer to a VkExternalSemaphoreProperties structure

The VkPhysicalDeviceExternalSemaphoreInfo structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkPhysicalDeviceExternalSemaphoreInfo {
    VkStructureType                           sType;
    const void*                               pNext;
    VkExternalSemaphoreHandleTypeFlagBits     handleType;
} VkPhysicalDeviceExternalSemaphoreInfo;
```

or the equivalent

```
// Provided by VK_KHR_external_semaphore_capabilities
typedef VkPhysicalDeviceExternalSemaphoreInfo
VkPhysicalDeviceExternalSemaphoreInfoKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- handleType is a VkExternalSemaphoreHandleTypeFlagBits value specifying the external semaphore handle type for which capabilities will be returned.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceExternalSemaphoreInfo-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTERNAL_SEMAPHORE_INFO
- VUID-VkPhysicalDeviceExternalSemaphoreInfo-pNext-pNext
  pNext **must** be NULL or a pointer to a valid instance of VkSemaphoreTypeCreateInfo
- VUID-VkPhysicalDeviceExternalSemaphoreInfo-sType-unique
  The sType value of each struct in the pNext chain **must** be unique
- VUID-VkPhysicalDeviceExternalSemaphoreInfo-handleType-parameter
  handleType **must** be a valid VkExternalSemaphoreHandleTypeFlagBits value

Bits which **may** be set in VkPhysicalDeviceExternalSemaphoreInfo::handleType, specifying an external semaphore handle type, are:

```
// Provided by VK_VERSION_1_1
typedef enum VkExternalSemaphoreHandleTypeFlagBits {
    VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_OPAQUE_FD_BIT = 0x00000001,
    VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_OPAQUE_WIN32_BIT = 0x00000002,
    VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_OPAQUE_WIN32_KMT_BIT = 0x00000004,
    VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_D3D12_FENCE_BIT = 0x00000008,
    VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_SYNC_FD_BIT = 0x00000010,
  // Provided by VK_FUCHSIA_external_semaphore
    VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_ZIRCON_EVENT_BIT_FUCHSIA = 0x00000080,
    VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_D3D11_FENCE_BIT =
VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_D3D12_FENCE_BIT,
  // Provided by VK_KHR_external_semaphore_capabilities
    VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_OPAQUE_FD_BIT_KHR =
VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_OPAQUE_FD_BIT,
  // Provided by VK_KHR_external_semaphore_capabilities
    VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_OPAQUE_WIN32_BIT_KHR =
VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_OPAQUE_WIN32_BIT,
  // Provided by VK_KHR_external_semaphore_capabilities
    VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_OPAQUE_WIN32_KMT_BIT_KHR =
VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_OPAQUE_WIN32_KMT_BIT,
  // Provided by VK_KHR_external_semaphore_capabilities
    VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_D3D12_FENCE_BIT_KHR =
VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_D3D12_FENCE_BIT,
  // Provided by VK_KHR_external_semaphore_capabilities
    VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_SYNC_FD_BIT_KHR =
VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_SYNC_FD_BIT,
} VkExternalSemaphoreHandleTypeFlagBits;
```

or the equivalent

```
// Provided by VK_KHR_external_semaphore_capabilities
typedef VkExternalSemaphoreHandleTypeFlagBits
VkExternalSemaphoreHandleTypeFlagBitsKHR;
```

- `VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_OPAQUE_FD_BIT` specifies a POSIX file descriptor handle that has only limited valid usage outside of Vulkan and other compatible APIs. It **must** be compatible with the POSIX system calls `dup`, `dup2`, `close`, and the non-standard system call `dup3`. Additionally, it **must** be transportable over a socket using an `SCM_RIGHTS` control message. It owns a reference to the underlying synchronization primitive represented by its Vulkan semaphore object.
- `VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_OPAQUE_WIN32_BIT` specifies an NT handle that has only limited valid usage outside of Vulkan and other compatible APIs. It **must** be compatible with the functions `DuplicateHandle`, `CloseHandle`, `CompareObjectHandles`, `GetHandleInformation`, and `SetHandleInformation`. It owns a reference to the underlying synchronization primitive represented by its Vulkan semaphore object.
- `VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_OPAQUE_WIN32_KMT_BIT` specifies a global share handle that has only limited valid usage outside of Vulkan and other compatible APIs. It is not compatible with any native APIs. It does not own a reference to the underlying synchronization primitive

represented by its Vulkan semaphore object, and will therefore become invalid when all Vulkan semaphore objects associated with it are destroyed.

- `VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_D3D12_FENCE_BIT` specifies an NT handle returned by `ID3D12Device::CreateSharedHandle` referring to a Direct3D 12 fence, or `ID3D11Device5::CreateFence` referring to a Direct3D 11 fence. It owns a reference to the underlying synchronization primitive associated with the Direct3D fence.
- `VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_D3D11_FENCE_BIT` is an alias of `VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_D3D12_FENCE_BIT` with the same meaning. It is provided for convenience and code clarity when interacting with D3D11 fences.
- `VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_SYNC_FD_BIT` specifies a POSIX file descriptor handle to a Linux Sync File or Android Fence object. It can be used with any native API accepting a valid sync file or fence as input. It owns a reference to the underlying synchronization primitive associated with the file descriptor. Implementations which support importing this handle type **must** accept any type of sync or fence FD supported by the native system they are running on.
- `VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_ZIRCON_EVENT_BIT_FUCHSIA` specifies a handle to a Zircon event object. It can be used with any native API that accepts a Zircon event handle. Zircon event handles are created with `ZX_RIGHTS_BASIC` and `ZX_RIGHTS_SIGNAL` rights. Vulkan on Fuchsia uses only the ZX_EVENT_SIGNALED bit when signaling or waiting.

**NOTE**

Handles of type `VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_SYNC_FD_BIT` generated by the implementation may represent either Linux Sync Files or Android Fences at the implementation's discretion. Applications **should** only use operations defined for both types of file descriptors, unless they know via means external to Vulkan the type of the file descriptor, or are prepared to deal with the system-defined operation failures resulting from using the wrong type.

Some external semaphore handle types can only be shared within the same underlying physical device and/or the same driver version, as defined in the following table:

*Table 92. External semaphore handle types compatibility*

| Handle type | VkPhysicalDeviceIDProperties::driverUUID | VkPhysicalDeviceIDProperties::deviceUUID |
|---|---|---|
| VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_OPAQUE_FD_BIT | Must match | Must match |
| VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_OPAQUE_WIN32_BIT | Must match | Must match |
| VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_OPAQUE_WIN32_KMT_BIT | Must match | Must match |
| VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_D3D12_FENCE_BIT | Must match | Must match |
| VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_SYNC_FD_BIT | No restriction | No restriction |
| VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_ZIRCON_EVENT_BIT_FUCHSIA | No restriction | No restriction |

```
// Provided by VK_VERSION_1_1
typedef VkFlags VkExternalSemaphoreHandleTypeFlags;
```

or the equivalent

```
// Provided by VK_KHR_external_semaphore_capabilities
typedef VkExternalSemaphoreHandleTypeFlags VkExternalSemaphoreHandleTypeFlagsKHR;
```

VkExternalSemaphoreHandleTypeFlags is a bitmask type for setting a mask of zero or more VkExternalSemaphoreHandleTypeFlagBits.

The VkExternalSemaphoreProperties structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkExternalSemaphoreProperties {
    VkStructureType                         sType;
    void*                                   pNext;
    VkExternalSemaphoreHandleTypeFlags      exportFromImportedHandleTypes;
    VkExternalSemaphoreHandleTypeFlags      compatibleHandleTypes;
    VkExternalSemaphoreFeatureFlags         externalSemaphoreFeatures;
} VkExternalSemaphoreProperties;
```

or the equivalent

```
// Provided by VK_KHR_external_semaphore_capabilities
typedef VkExternalSemaphoreProperties VkExternalSemaphorePropertiesKHR;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- exportFromImportedHandleTypes is a bitmask of VkExternalSemaphoreHandleTypeFlagBits specifying which types of imported handle handleType **can** be exported from.
- compatibleHandleTypes is a bitmask of VkExternalSemaphoreHandleTypeFlagBits specifying handle types which **can** be specified at the same time as handleType when creating a semaphore.
- externalSemaphoreFeatures is a bitmask of VkExternalSemaphoreFeatureFlagBits describing the features of handleType.

If handleType is not supported by the implementation, then VkExternalSemaphoreProperties::externalSemaphoreFeatures will be set to zero.

### Valid Usage (Implicit)

- VUID-VkExternalSemaphoreProperties-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_EXTERNAL_SEMAPHORE_PROPERTIES
- VUID-VkExternalSemaphoreProperties-pNext-pNext
  pNext **must** be NULL

Bits which **may** be set in VkExternalSemaphoreProperties::externalSemaphoreFeatures, specifying the features of an external semaphore handle type, are:

```
// Provided by VK_VERSION_1_1
typedef enum VkExternalSemaphoreFeatureFlagBits {
    VK_EXTERNAL_SEMAPHORE_FEATURE_EXPORTABLE_BIT = 0x00000001,
    VK_EXTERNAL_SEMAPHORE_FEATURE_IMPORTABLE_BIT = 0x00000002,
  // Provided by VK_KHR_external_semaphore_capabilities
    VK_EXTERNAL_SEMAPHORE_FEATURE_EXPORTABLE_BIT_KHR =
VK_EXTERNAL_SEMAPHORE_FEATURE_EXPORTABLE_BIT,
  // Provided by VK_KHR_external_semaphore_capabilities
    VK_EXTERNAL_SEMAPHORE_FEATURE_IMPORTABLE_BIT_KHR =
VK_EXTERNAL_SEMAPHORE_FEATURE_IMPORTABLE_BIT,
} VkExternalSemaphoreFeatureFlagBits;
```

or the equivalent

```
// Provided by VK_KHR_external_semaphore_capabilities
typedef VkExternalSemaphoreFeatureFlagBits VkExternalSemaphoreFeatureFlagBitsKHR;
```

- VK_EXTERNAL_SEMAPHORE_FEATURE_EXPORTABLE_BIT specifies that handles of this type **can** be exported from Vulkan semaphore objects.
- VK_EXTERNAL_SEMAPHORE_FEATURE_IMPORTABLE_BIT specifies that handles of this type **can** be imported as Vulkan semaphore objects.

```
// Provided by VK_VERSION_1_1
typedef VkFlags VkExternalSemaphoreFeatureFlags;
```

or the equivalent

```
// Provided by VK_KHR_external_semaphore_capabilities
typedef VkExternalSemaphoreFeatureFlags VkExternalSemaphoreFeatureFlagsKHR;
```

VkExternalSemaphoreFeatureFlags is a bitmask type for setting a mask of zero or more VkExternalSemaphoreFeatureFlagBits.

## 49.4. Optional Fence Capabilities

Fences **may** support import and export of their payload to external handles. To query the external handle types supported by fences, call:

```
// Provided by VK_VERSION_1_1
void vkGetPhysicalDeviceExternalFenceProperties(
    VkPhysicalDevice                            physicalDevice,
    const VkPhysicalDeviceExternalFenceInfo*    pExternalFenceInfo,
    VkExternalFenceProperties*                  pExternalFenceProperties);
```

or the equivalent command

```
// Provided by VK_KHR_external_fence_capabilities
void vkGetPhysicalDeviceExternalFencePropertiesKHR(
    VkPhysicalDevice                            physicalDevice,
    const VkPhysicalDeviceExternalFenceInfo*    pExternalFenceInfo,
    VkExternalFenceProperties*                  pExternalFenceProperties);
```

- physicalDevice is the physical device from which to query the fence capabilities.
- pExternalFenceInfo is a pointer to a VkPhysicalDeviceExternalFenceInfo structure describing the parameters that would be consumed by vkCreateFence.
- pExternalFenceProperties is a pointer to a VkExternalFenceProperties structure in which capabilities are returned.

### Valid Usage (Implicit)

- VUID-vkGetPhysicalDeviceExternalFenceProperties-physicalDevice-parameter
  physicalDevice **must** be a valid VkPhysicalDevice handle
- VUID-vkGetPhysicalDeviceExternalFenceProperties-pExternalFenceInfo-parameter
  pExternalFenceInfo **must** be a valid pointer to a valid VkPhysicalDeviceExternalFenceInfo structure

- VUID-vkGetPhysicalDeviceExternalFenceProperties-pExternalFenceProperties-parameter
  `pExternalFenceProperties` **must** be a valid pointer to a VkExternalFenceProperties structure

The `VkPhysicalDeviceExternalFenceInfo` structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkPhysicalDeviceExternalFenceInfo {
    VkStructureType                     sType;
    const void*                         pNext;
    VkExternalFenceHandleTypeFlagBits   handleType;
} VkPhysicalDeviceExternalFenceInfo;
```

or the equivalent

```
// Provided by VK_KHR_external_fence_capabilities
typedef VkPhysicalDeviceExternalFenceInfo VkPhysicalDeviceExternalFenceInfoKHR;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `handleType` is a VkExternalFenceHandleTypeFlagBits value specifying an external fence handle type for which capabilities will be returned.

**NOTE** Handles of type `VK_EXTERNAL_FENCE_HANDLE_TYPE_SYNC_FD_BIT` generated by the implementation may represent either Linux Sync Files or Android Fences at the implementation's discretion. Applications **should** only use operations defined for both types of file descriptors, unless they know via means external to Vulkan the type of the file descriptor, or are prepared to deal with the system-defined operation failures resulting from using the wrong type.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceExternalFenceInfo-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTERNAL_FENCE_INFO`
- VUID-VkPhysicalDeviceExternalFenceInfo-pNext-pNext
  `pNext` **must** be `NULL`
- VUID-VkPhysicalDeviceExternalFenceInfo-handleType-parameter
  `handleType` **must** be a valid VkExternalFenceHandleTypeFlagBits value

Bits which **may** be set in

- VkPhysicalDeviceExternalFenceInfo::`handleType`
- VkExternalFenceProperties::`exportFromImportedHandleTypes`

- VkExternalFenceProperties::compatibleHandleTypes

indicate external fence handle types, and are:

```
// Provided by VK_VERSION_1_1
typedef enum VkExternalFenceHandleTypeFlagBits {
    VK_EXTERNAL_FENCE_HANDLE_TYPE_OPAQUE_FD_BIT = 0x00000001,
    VK_EXTERNAL_FENCE_HANDLE_TYPE_OPAQUE_WIN32_BIT = 0x00000002,
    VK_EXTERNAL_FENCE_HANDLE_TYPE_OPAQUE_WIN32_KMT_BIT = 0x00000004,
    VK_EXTERNAL_FENCE_HANDLE_TYPE_SYNC_FD_BIT = 0x00000008,
  // Provided by VK_KHR_external_fence_capabilities
    VK_EXTERNAL_FENCE_HANDLE_TYPE_OPAQUE_FD_BIT_KHR =
VK_EXTERNAL_FENCE_HANDLE_TYPE_OPAQUE_FD_BIT,
  // Provided by VK_KHR_external_fence_capabilities
    VK_EXTERNAL_FENCE_HANDLE_TYPE_OPAQUE_WIN32_BIT_KHR =
VK_EXTERNAL_FENCE_HANDLE_TYPE_OPAQUE_WIN32_BIT,
  // Provided by VK_KHR_external_fence_capabilities
    VK_EXTERNAL_FENCE_HANDLE_TYPE_OPAQUE_WIN32_KMT_BIT_KHR =
VK_EXTERNAL_FENCE_HANDLE_TYPE_OPAQUE_WIN32_KMT_BIT,
  // Provided by VK_KHR_external_fence_capabilities
    VK_EXTERNAL_FENCE_HANDLE_TYPE_SYNC_FD_BIT_KHR =
VK_EXTERNAL_FENCE_HANDLE_TYPE_SYNC_FD_BIT,
} VkExternalFenceHandleTypeFlagBits;
```

or the equivalent

```
// Provided by VK_KHR_external_fence_capabilities
typedef VkExternalFenceHandleTypeFlagBits VkExternalFenceHandleTypeFlagBitsKHR;
```

- VK_EXTERNAL_FENCE_HANDLE_TYPE_OPAQUE_FD_BIT specifies a POSIX file descriptor handle that has only limited valid usage outside of Vulkan and other compatible APIs. It **must** be compatible with the POSIX system calls dup, dup2, close, and the non-standard system call dup3. Additionally, it **must** be transportable over a socket using an SCM_RIGHTS control message. It owns a reference to the underlying synchronization primitive represented by its Vulkan fence object.
- VK_EXTERNAL_FENCE_HANDLE_TYPE_OPAQUE_WIN32_BIT specifies an NT handle that has only limited valid usage outside of Vulkan and other compatible APIs. It **must** be compatible with the functions DuplicateHandle, CloseHandle, CompareObjectHandles, GetHandleInformation, and SetHandleInformation. It owns a reference to the underlying synchronization primitive represented by its Vulkan fence object.
- VK_EXTERNAL_FENCE_HANDLE_TYPE_OPAQUE_WIN32_KMT_BIT specifies a global share handle that has only limited valid usage outside of Vulkan and other compatible APIs. It is not compatible with any native APIs. It does not own a reference to the underlying synchronization primitive represented by its Vulkan fence object, and will therefore become invalid when all Vulkan fence objects associated with it are destroyed.
- VK_EXTERNAL_FENCE_HANDLE_TYPE_SYNC_FD_BIT specifies a POSIX file descriptor handle to a Linux Sync File or Android Fence. It can be used with any native API accepting a valid sync file or

fence as input. It owns a reference to the underlying synchronization primitive associated with the file descriptor. Implementations which support importing this handle type **must** accept any type of sync or fence FD supported by the native system they are running on.

Some external fence handle types can only be shared within the same underlying physical device and/or the same driver version, as defined in the following table:

*Table 93. External fence handle types compatibility*

| Handle type | VkPhysicalDeviceIDProperties::driverUUID | VkPhysicalDeviceIDProperties::deviceUUID |
|---|---|---|
| VK_EXTERNAL_FENCE_HANDLE_TYPE_OPAQUE_FD_BIT | Must match | Must match |
| VK_EXTERNAL_FENCE_HANDLE_TYPE_OPAQUE_WIN32_BIT | Must match | Must match |
| VK_EXTERNAL_FENCE_HANDLE_TYPE_OPAQUE_WIN32_KMT_BIT | Must match | Must match |
| VK_EXTERNAL_FENCE_HANDLE_TYPE_SYNC_FD_BIT | No restriction | No restriction |

```
// Provided by VK_VERSION_1_1
typedef VkFlags VkExternalFenceHandleTypeFlags;
```

or the equivalent

```
// Provided by VK_KHR_external_fence_capabilities
typedef VkExternalFenceHandleTypeFlags VkExternalFenceHandleTypeFlagsKHR;
```

VkExternalFenceHandleTypeFlags is a bitmask type for setting a mask of zero or more VkExternalFenceHandleTypeFlagBits.

The VkExternalFenceProperties structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkExternalFenceProperties {
    VkStructureType                 sType;
    void*                           pNext;
    VkExternalFenceHandleTypeFlags  exportFromImportedHandleTypes;
    VkExternalFenceHandleTypeFlags  compatibleHandleTypes;
    VkExternalFenceFeatureFlags     externalFenceFeatures;
} VkExternalFenceProperties;
```

or the equivalent

```
// Provided by VK_KHR_external_fence_capabilities
typedef VkExternalFenceProperties VkExternalFencePropertiesKHR;
```

- exportFromImportedHandleTypes is a bitmask of VkExternalFenceHandleTypeFlagBits indicating which types of imported handle handleType **can** be exported from.

- compatibleHandleTypes is a bitmask of VkExternalFenceHandleTypeFlagBits specifying handle types which **can** be specified at the same time as handleType when creating a fence.
- externalFenceFeatures is a bitmask of VkExternalFenceFeatureFlagBits indicating the features of handleType.

If handleType is not supported by the implementation, then VkExternalFenceProperties::externalFenceFeatures will be set to zero.

### Valid Usage (Implicit)

- VUID-VkExternalFenceProperties-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_EXTERNAL_FENCE_PROPERTIES
- VUID-VkExternalFenceProperties-pNext-pNext
  pNext **must** be NULL

Bits which **may** be set in VkExternalFenceProperties::externalFenceFeatures, indicating features of a fence external handle type, are:

```
// Provided by VK_VERSION_1_1
typedef enum VkExternalFenceFeatureFlagBits {
    VK_EXTERNAL_FENCE_FEATURE_EXPORTABLE_BIT = 0x00000001,
    VK_EXTERNAL_FENCE_FEATURE_IMPORTABLE_BIT = 0x00000002,
  // Provided by VK_KHR_external_fence_capabilities
    VK_EXTERNAL_FENCE_FEATURE_EXPORTABLE_BIT_KHR =
VK_EXTERNAL_FENCE_FEATURE_EXPORTABLE_BIT,
  // Provided by VK_KHR_external_fence_capabilities
    VK_EXTERNAL_FENCE_FEATURE_IMPORTABLE_BIT_KHR =
VK_EXTERNAL_FENCE_FEATURE_IMPORTABLE_BIT,
} VkExternalFenceFeatureFlagBits;
```

or the equivalent

```
// Provided by VK_KHR_external_fence_capabilities
typedef VkExternalFenceFeatureFlagBits VkExternalFenceFeatureFlagBitsKHR;
```

- VK_EXTERNAL_FENCE_FEATURE_EXPORTABLE_BIT specifies handles of this type **can** be exported from Vulkan fence objects.
- VK_EXTERNAL_FENCE_FEATURE_IMPORTABLE_BIT specifies handles of this type **can** be imported to Vulkan fence objects.

```
// Provided by VK_VERSION_1_1
typedef VkFlags VkExternalFenceFeatureFlags;
```

or the equivalent

```
// Provided by VK_KHR_external_fence_capabilities
typedef VkExternalFenceFeatureFlags VkExternalFenceFeatureFlagsKHR;
```

VkExternalFenceFeatureFlags is a bitmask type for setting a mask of zero or more VkExternalFenceFeatureFlagBits.

## 49.5. Timestamp Calibration Capabilities

To query the set of time domains for which a physical device supports timestamp calibration, call:

```
// Provided by VK_KHR_calibrated_timestamps
VkResult vkGetPhysicalDeviceCalibrateableTimeDomainsKHR(
    VkPhysicalDevice                            physicalDevice,
    uint32_t*                                   pTimeDomainCount,
    VkTimeDomainKHR*                            pTimeDomains);
```

or the equivalent command

```
// Provided by VK_EXT_calibrated_timestamps
VkResult vkGetPhysicalDeviceCalibrateableTimeDomainsEXT(
    VkPhysicalDevice                            physicalDevice,
    uint32_t*                                   pTimeDomainCount,
    VkTimeDomainKHR*                            pTimeDomains);
```

- physicalDevice is the physical device from which to query the set of calibrateable time domains.
- pTimeDomainCount is a pointer to an integer related to the number of calibrateable time domains available or queried, as described below.
- pTimeDomains is either NULL or a pointer to an array of VkTimeDomainKHR values, indicating the supported calibrateable time domains.

If pTimeDomains is NULL, then the number of calibrateable time domains supported for the given physicalDevice is returned in pTimeDomainCount. Otherwise, pTimeDomainCount **must** point to a variable set by the application to the number of elements in the pTimeDomains array, and on return the variable is overwritten with the number of values actually written to pTimeDomains. If the value of pTimeDomainCount is less than the number of calibrateable time domains supported, at most pTimeDomainCount values will be written to pTimeDomains, and VK_INCOMPLETE will be returned instead of VK_SUCCESS, to indicate that not all the available time domains were returned.

### Valid Usage (Implicit)

- VUID-vkGetPhysicalDeviceCalibrateableTimeDomainsKHR-physicalDevice-parameter
  physicalDevice **must** be a valid VkPhysicalDevice handle
- VUID-vkGetPhysicalDeviceCalibrateableTimeDomainsKHR-pTimeDomainCount-parameter

pTimeDomainCount **must** be a valid pointer to a uint32_t value

- VUID-vkGetPhysicalDeviceCalibrateableTimeDomainsKHR-pTimeDomains-parameter
  If the value referenced by pTimeDomainCount is not 0, and pTimeDomains is not NULL, pTimeDomains **must** be a valid pointer to an array of pTimeDomainCount VkTimeDomainKHR values

## Return Codes

### Success

- VK_SUCCESS
- VK_INCOMPLETE

### Failure

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY

# Chapter 50. Debugging

To aid developers in tracking down errors in the application's use of Vulkan, particularly in combination with an external debugger or profiler, *debugging extensions* may be available.

The VkObjectType enumeration defines values, each of which corresponds to a specific Vulkan handle type. These values **can** be used to associate debug information with a particular type of object through one or more extensions.

```
// Provided by VK_VERSION_1_0
typedef enum VkObjectType {
    VK_OBJECT_TYPE_UNKNOWN = 0,
    VK_OBJECT_TYPE_INSTANCE = 1,
    VK_OBJECT_TYPE_PHYSICAL_DEVICE = 2,
    VK_OBJECT_TYPE_DEVICE = 3,
    VK_OBJECT_TYPE_QUEUE = 4,
    VK_OBJECT_TYPE_SEMAPHORE = 5,
    VK_OBJECT_TYPE_COMMAND_BUFFER = 6,
    VK_OBJECT_TYPE_FENCE = 7,
    VK_OBJECT_TYPE_DEVICE_MEMORY = 8,
    VK_OBJECT_TYPE_BUFFER = 9,
    VK_OBJECT_TYPE_IMAGE = 10,
    VK_OBJECT_TYPE_EVENT = 11,
    VK_OBJECT_TYPE_QUERY_POOL = 12,
    VK_OBJECT_TYPE_BUFFER_VIEW = 13,
    VK_OBJECT_TYPE_IMAGE_VIEW = 14,
    VK_OBJECT_TYPE_SHADER_MODULE = 15,
    VK_OBJECT_TYPE_PIPELINE_CACHE = 16,
    VK_OBJECT_TYPE_PIPELINE_LAYOUT = 17,
    VK_OBJECT_TYPE_RENDER_PASS = 18,
    VK_OBJECT_TYPE_PIPELINE = 19,
    VK_OBJECT_TYPE_DESCRIPTOR_SET_LAYOUT = 20,
    VK_OBJECT_TYPE_SAMPLER = 21,
    VK_OBJECT_TYPE_DESCRIPTOR_POOL = 22,
    VK_OBJECT_TYPE_DESCRIPTOR_SET = 23,
    VK_OBJECT_TYPE_FRAMEBUFFER = 24,
    VK_OBJECT_TYPE_COMMAND_POOL = 25,
  // Provided by VK_VERSION_1_1
    VK_OBJECT_TYPE_SAMPLER_YCBCR_CONVERSION = 1000156000,
  // Provided by VK_VERSION_1_1
    VK_OBJECT_TYPE_DESCRIPTOR_UPDATE_TEMPLATE = 1000085000,
    VK_OBJECT_TYPE_PRIVATE_DATA_SLOT = 1000295000,
  // Provided by VK_KHR_surface
    VK_OBJECT_TYPE_SURFACE_KHR = 1000000000,
  // Provided by VK_KHR_swapchain
    VK_OBJECT_TYPE_SWAPCHAIN_KHR = 1000001000,
  // Provided by VK_KHR_display
    VK_OBJECT_TYPE_DISPLAY_KHR = 1000002000,
  // Provided by VK_KHR_display
    VK_OBJECT_TYPE_DISPLAY_MODE_KHR = 1000002001,
```

```
  // Provided by VK_EXT_debug_report
    VK_OBJECT_TYPE_DEBUG_REPORT_CALLBACK_EXT = 1000011000,
  // Provided by VK_KHR_video_queue
    VK_OBJECT_TYPE_VIDEO_SESSION_KHR = 1000023000,
  // Provided by VK_KHR_video_queue
    VK_OBJECT_TYPE_VIDEO_SESSION_PARAMETERS_KHR = 1000023001,
  // Provided by VK_NVX_binary_import
    VK_OBJECT_TYPE_CU_MODULE_NVX = 1000029000,
  // Provided by VK_NVX_binary_import
    VK_OBJECT_TYPE_CU_FUNCTION_NVX = 1000029001,
  // Provided by VK_EXT_debug_utils
    VK_OBJECT_TYPE_DEBUG_UTILS_MESSENGER_EXT = 1000128000,
  // Provided by VK_KHR_acceleration_structure
    VK_OBJECT_TYPE_ACCELERATION_STRUCTURE_KHR = 1000150000,
  // Provided by VK_EXT_validation_cache
    VK_OBJECT_TYPE_VALIDATION_CACHE_EXT = 1000160000,
  // Provided by VK_NV_ray_tracing
    VK_OBJECT_TYPE_ACCELERATION_STRUCTURE_NV = 1000165000,
  // Provided by VK_INTEL_performance_query
    VK_OBJECT_TYPE_PERFORMANCE_CONFIGURATION_INTEL = 1000210000,
  // Provided by VK_KHR_deferred_host_operations
    VK_OBJECT_TYPE_DEFERRED_OPERATION_KHR = 1000268000,
  // Provided by VK_NV_device_generated_commands
    VK_OBJECT_TYPE_INDIRECT_COMMANDS_LAYOUT_NV = 1000277000,
  // Provided by VK_NV_cuda_kernel_launch
    VK_OBJECT_TYPE_CUDA_MODULE_NV = 1000307000,
  // Provided by VK_NV_cuda_kernel_launch
    VK_OBJECT_TYPE_CUDA_FUNCTION_NV = 1000307001,
  // Provided by VK_FUCHSIA_buffer_collection
    VK_OBJECT_TYPE_BUFFER_COLLECTION_FUCHSIA = 1000366000,
  // Provided by VK_EXT_opacity_micromap
    VK_OBJECT_TYPE_MICROMAP_EXT = 1000396000,
  // Provided by VK_NV_optical_flow
    VK_OBJECT_TYPE_OPTICAL_FLOW_SESSION_NV = 1000464000,
  // Provided by VK_EXT_shader_object
    VK_OBJECT_TYPE_SHADER_EXT = 1000482000,
  // Provided by VK_KHR_pipeline_binary
    VK_OBJECT_TYPE_PIPELINE_BINARY_KHR = 1000483000,
  // Provided by VK_KHR_descriptor_update_template
    VK_OBJECT_TYPE_DESCRIPTOR_UPDATE_TEMPLATE_KHR =
VK_OBJECT_TYPE_DESCRIPTOR_UPDATE_TEMPLATE,
  // Provided by VK_KHR_sampler_ycbcr_conversion
    VK_OBJECT_TYPE_SAMPLER_YCBCR_CONVERSION_KHR =
VK_OBJECT_TYPE_SAMPLER_YCBCR_CONVERSION,
  // Provided by VK_EXT_private_data
    VK_OBJECT_TYPE_PRIVATE_DATA_SLOT_EXT = VK_OBJECT_TYPE_PRIVATE_DATA_SLOT,
} VkObjectType;
```

*Table 94. `VkObjectType` and Vulkan Handle Relationship*

| VkObjectType | Vulkan Handle Type |
| --- | --- |
| VK_OBJECT_TYPE_UNKNOWN | Unknown/Undefined Handle |
| VK_OBJECT_TYPE_INSTANCE | VkInstance |
| VK_OBJECT_TYPE_PHYSICAL_DEVICE | VkPhysicalDevice |
| VK_OBJECT_TYPE_DEVICE | VkDevice |
| VK_OBJECT_TYPE_QUEUE | VkQueue |
| VK_OBJECT_TYPE_SEMAPHORE | VkSemaphore |
| VK_OBJECT_TYPE_COMMAND_BUFFER | VkCommandBuffer |
| VK_OBJECT_TYPE_FENCE | VkFence |
| VK_OBJECT_TYPE_DEVICE_MEMORY | VkDeviceMemory |
| VK_OBJECT_TYPE_BUFFER | VkBuffer |
| VK_OBJECT_TYPE_IMAGE | VkImage |
| VK_OBJECT_TYPE_EVENT | VkEvent |
| VK_OBJECT_TYPE_QUERY_POOL | VkQueryPool |
| VK_OBJECT_TYPE_BUFFER_VIEW | VkBufferView |
| VK_OBJECT_TYPE_IMAGE_VIEW | VkImageView |
| VK_OBJECT_TYPE_SHADER_MODULE | VkShaderModule |
| VK_OBJECT_TYPE_PIPELINE_CACHE | VkPipelineCache |
| VK_OBJECT_TYPE_PIPELINE_LAYOUT | VkPipelineLayout |
| VK_OBJECT_TYPE_RENDER_PASS | VkRenderPass |
| VK_OBJECT_TYPE_PIPELINE | VkPipeline |
| VK_OBJECT_TYPE_DESCRIPTOR_SET_LAYOUT | VkDescriptorSetLayout |
| VK_OBJECT_TYPE_SAMPLER | VkSampler |
| VK_OBJECT_TYPE_DESCRIPTOR_POOL | VkDescriptorPool |
| VK_OBJECT_TYPE_DESCRIPTOR_SET | VkDescriptorSet |
| VK_OBJECT_TYPE_FRAMEBUFFER | VkFramebuffer |
| VK_OBJECT_TYPE_COMMAND_POOL | VkCommandPool |
| VK_OBJECT_TYPE_SAMPLER_YCBCR_CONVERSION | VkSamplerYcbcrConversion |
| VK_OBJECT_TYPE_DESCRIPTOR_UPDATE_TEMPLATE | VkDescriptorUpdateTemplate |
| VK_OBJECT_TYPE_PRIVATE_DATA_SLOT | VkPrivateDataSlot |
| VK_OBJECT_TYPE_SURFACE_KHR | VkSurfaceKHR |
| VK_OBJECT_TYPE_SWAPCHAIN_KHR | VkSwapchainKHR |
| VK_OBJECT_TYPE_DISPLAY_KHR | VkDisplayKHR |
| VK_OBJECT_TYPE_DISPLAY_MODE_KHR | VkDisplayModeKHR |

| VkObjectType | Vulkan Handle Type |
| --- | --- |
| VK_OBJECT_TYPE_DEBUG_REPORT_CALLBACK_EXT | VkDebugReportCallbackEXT |
| VK_OBJECT_TYPE_VIDEO_SESSION_KHR | VkVideoSessionKHR |
| VK_OBJECT_TYPE_VIDEO_SESSION_PARAMETERS_KHR | VkVideoSessionParametersKHR |
| VK_OBJECT_TYPE_DEBUG_UTILS_MESSENGER_EXT | VkDebugUtilsMessengerEXT |
| VK_OBJECT_TYPE_ACCELERATION_STRUCTURE_KHR | VkAccelerationStructureKHR |
| VK_OBJECT_TYPE_VALIDATION_CACHE_EXT | VkValidationCacheEXT |
| VK_OBJECT_TYPE_ACCELERATION_STRUCTURE_NV | VkAccelerationStructureNV |
| VK_OBJECT_TYPE_PERFORMANCE_CONFIGURATION_INTEL | VkPerformanceConfigurationINTEL |
| VK_OBJECT_TYPE_DEFERRED_OPERATION_KHR | VkDeferredOperationKHR |
| VK_OBJECT_TYPE_INDIRECT_COMMANDS_LAYOUT_NV | VkIndirectCommandsLayoutNV |
| VK_OBJECT_TYPE_BUFFER_COLLECTION_FUCHSIA | VkBufferCollectionFUCHSIA |
| VK_OBJECT_TYPE_MICROMAP_EXT | VkMicromapEXT |
| VK_OBJECT_TYPE_OPTICAL_FLOW_SESSION_NV | VkOpticalFlowSessionNV |
| VK_OBJECT_TYPE_SHADER_EXT | VkShaderEXT |

If this Specification was generated with any such extensions included, they will be described in the remainder of this chapter.

# 50.1. Debug Utilities

Vulkan provides flexible debugging utilities for debugging an application.

The Object Debug Annotation section describes how to associate either a name or binary data with a specific Vulkan object.

The Queue Labels section describes how to annotate and group the work submitted to a queue.

The Command Buffer Labels section describes how to associate logical elements of the scene with commands in a VkCommandBuffer.

The Debug Messengers section describes how to create debug messenger objects associated with an application supplied callback to capture debug messages from a variety of Vulkan components.

## 50.1.1. Object Debug Annotation

It can be useful for an application to provide its own content relative to a specific Vulkan object.

The following commands allow application developers to associate application-defined information with Vulkan objects. These commands are device-level commands but they **may** reference instance-

level objects (such as VkInstance) and physical device-level objects (such as VkPhysicalDevice) with a few restrictions: * The data for the corresponding object **may** still be available after the VkDevice used in the corresponding API call to set it is destroyed, but access to this data is not guaranteed and should be avoided. * Subsequent calls to change the data of the same object across multiple VkDevice objects, **may** result in the data being changed to the most recent version for all VkDevice objects and not just the VkDevice used in the most recent API call.

### Object Naming

An object can be given an application-defined name by calling:

```
// Provided by VK_EXT_debug_utils
VkResult vkSetDebugUtilsObjectNameEXT(
    VkDevice                                    device,
    const VkDebugUtilsObjectNameInfoEXT*        pNameInfo);
```

- device is the device that is associated with the named object passed in via objectHandle.
- pNameInfo is a pointer to a VkDebugUtilsObjectNameInfoEXT structure specifying parameters of the name to set on the object.

## Valid Usage

- VUID-vkSetDebugUtilsObjectNameEXT-pNameInfo-02587
  pNameInfo->objectType **must** not be VK_OBJECT_TYPE_UNKNOWN
- VUID-vkSetDebugUtilsObjectNameEXT-pNameInfo-02588
  pNameInfo->objectHandle **must** not be VK_NULL_HANDLE
- VUID-vkSetDebugUtilsObjectNameEXT-pNameInfo-07872
  If pNameInfo->objectHandle is the valid handle of an instance-level object, the VkDevice identified by device **must** be a descendent of the same VkInstance as the object identified by pNameInfo->objectHandle
- VUID-vkSetDebugUtilsObjectNameEXT-pNameInfo-07873
  If pNameInfo->objectHandle is the valid handle of a physical-device-level object, the VkDevice identified by device **must** be a descendant of the same VkPhysicalDevice as the object identified by pNameInfo->objectHandle
- VUID-vkSetDebugUtilsObjectNameEXT-pNameInfo-07874
  If pNameInfo->objectHandle is the valid handle of a device-level object, that object **must** be a descendent of the VkDevice identified by device

## Valid Usage (Implicit)

- VUID-vkSetDebugUtilsObjectNameEXT-device-parameter
  device **must** be a valid VkDevice handle
- VUID-vkSetDebugUtilsObjectNameEXT-pNameInfo-parameter
  pNameInfo **must** be a valid pointer to a valid VkDebugUtilsObjectNameInfoEXT structure

## Host Synchronization

- Host access to `pNameInfo->objectHandle` **must** be externally synchronized

## Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`

The `VkDebugUtilsObjectNameInfoEXT` structure is defined as:

```
// Provided by VK_EXT_debug_utils
typedef struct VkDebugUtilsObjectNameInfoEXT {
    VkStructureType    sType;
    const void*        pNext;
    VkObjectType       objectType;
    uint64_t           objectHandle;
    const char*        pObjectName;
} VkDebugUtilsObjectNameInfoEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `objectType` is a VkObjectType specifying the type of the object to be named.
- `objectHandle` is the object to be named.
- `pObjectName` is either `NULL` or a null-terminated UTF-8 string specifying the name to apply to `objectHandle`.

Applications **may** change the name associated with an object simply by calling `vkSetDebugUtilsObjectNameEXT` again with a new string. If `pObjectName` is either `NULL` or an empty string, then any previously set name is removed.

The `graphicsPipelineLibrary` feature allows the specification of pipelines without the creation of VkShaderModule objects beforehand. In order to continue to allow naming these shaders independently, `VkDebugUtilsObjectNameInfoEXT` **can** be included in the `pNext` chain of VkPipelineShaderStageCreateInfo, which associates a static name with that particular shader.

## Valid Usage

- VUID-VkDebugUtilsObjectNameInfoEXT-objectType-02589

If objectType is VK_OBJECT_TYPE_UNKNOWN, objectHandle **must** not be VK_NULL_HANDLE

- VUID-VkDebugUtilsObjectNameInfoEXT-objectType-02590
  If objectType is not VK_OBJECT_TYPE_UNKNOWN, objectHandle **must** be VK_NULL_HANDLE or a valid Vulkan handle of the type associated with objectType as defined in the VkObjectType and Vulkan Handle Relationship table

## Valid Usage (Implicit)

- VUID-VkDebugUtilsObjectNameInfoEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_DEBUG_UTILS_OBJECT_NAME_INFO_EXT
- VUID-VkDebugUtilsObjectNameInfoEXT-objectType-parameter
  objectType **must** be a valid VkObjectType value
- VUID-VkDebugUtilsObjectNameInfoEXT-pObjectName-parameter
  If pObjectName is not NULL, pObjectName **must** be a null-terminated UTF-8 string

### Object Data Association

In addition to setting a name for an object, debugging and validation layers **may** have uses for additional binary data on a per-object basis that have no other place in the Vulkan API.

For example, a VkShaderModule could have additional debugging data attached to it to aid in offline shader tracing.

Additional data can be attached to an object by calling vkSetDebugUtilsObjectTagEXT as defined below.

```
// Provided by VK_EXT_debug_utils
VkResult vkSetDebugUtilsObjectTagEXT(
    VkDevice                                    device,
    const VkDebugUtilsObjectTagInfoEXT*         pTagInfo);
```

- device is the device that created the object.
- pTagInfo is a pointer to a VkDebugUtilsObjectTagInfoEXT structure specifying parameters of the tag to attach to the object.

## Valid Usage

- VUID-vkSetDebugUtilsObjectTagEXT-pNameInfo-07875
  If pNameInfo->objectHandle is the valid handle of an instance-level object, the VkDevice identified by device **must** be a descendent of the same VkInstance as the object identified by pNameInfo->objectHandle
- VUID-vkSetDebugUtilsObjectTagEXT-pNameInfo-07876
  If pNameInfo->objectHandle is the valid handle of a physical-device-level object, the VkDevice identified by device **must** be a descendant of the same VkPhysicalDevice as the

object identified by `pNameInfo->objectHandle`

- VUID-vkSetDebugUtilsObjectTagEXT-pNameInfo-07877
  If `pNameInfo->objectHandle` is the valid handle of a device-level object, that object **must** be a descendent of the VkDevice identified by `device`

## Valid Usage (Implicit)

- VUID-vkSetDebugUtilsObjectTagEXT-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkSetDebugUtilsObjectTagEXT-pTagInfo-parameter
  `pTagInfo` **must** be a valid pointer to a valid VkDebugUtilsObjectTagInfoEXT structure

## Host Synchronization

- Host access to `pTagInfo->objectHandle` **must** be externally synchronized

## Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`

The `VkDebugUtilsObjectTagInfoEXT` structure is defined as:

```
// Provided by VK_EXT_debug_utils
typedef struct VkDebugUtilsObjectTagInfoEXT {
    VkStructureType    sType;
    const void*        pNext;
    VkObjectType       objectType;
    uint64_t           objectHandle;
    uint64_t           tagName;
    size_t             tagSize;
    const void*        pTag;
} VkDebugUtilsObjectTagInfoEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `objectType` is a VkObjectType specifying the type of the object to be named.

- `objectHandle` is the object to be tagged.
- `tagName` is a numerical identifier of the tag.
- `tagSize` is the number of bytes of data to attach to the object.
- `pTag` is a pointer to an array of `tagSize` bytes containing the data to be associated with the object.

The `tagName` parameter gives a name or identifier to the type of data being tagged. This can be used by debugging layers to easily filter for only data that can be used by that implementation.

### Valid Usage

- VUID-VkDebugUtilsObjectTagInfoEXT-objectType-01908
  `objectType` **must** not be `VK_OBJECT_TYPE_UNKNOWN`
- VUID-VkDebugUtilsObjectTagInfoEXT-objectHandle-01910
  `objectHandle` **must** be a valid Vulkan handle of the type associated with `objectType` as defined in the `VkObjectType` and Vulkan Handle Relationship table

### Valid Usage (Implicit)

- VUID-VkDebugUtilsObjectTagInfoEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_DEBUG_UTILS_OBJECT_TAG_INFO_EXT`
- VUID-VkDebugUtilsObjectTagInfoEXT-pNext-pNext
  `pNext` **must** be `NULL`
- VUID-VkDebugUtilsObjectTagInfoEXT-objectType-parameter
  `objectType` **must** be a valid VkObjectType value
- VUID-VkDebugUtilsObjectTagInfoEXT-pTag-parameter
  `pTag` **must** be a valid pointer to an array of `tagSize` bytes
- VUID-VkDebugUtilsObjectTagInfoEXT-tagSize-arraylength
  `tagSize` **must** be greater than `0`

## 50.1.2. Queue Labels

All Vulkan work is submitted using queues. It is possible for an application to use multiple queues, each containing multiple command buffers, when performing work. It can be useful to identify which queue, or even where in a queue, something has occurred.

To begin identifying a region using a debug label inside a queue, you may use the vkQueueBeginDebugUtilsLabelEXT command.

Then, when the region of interest has passed, you may end the label region using vkQueueEndDebugUtilsLabelEXT.

Additionally, a single debug label may be inserted at any time using vkQueueInsertDebugUtilsLabelEXT.

A queue debug label region is opened by calling:

```
// Provided by VK_EXT_debug_utils
void vkQueueBeginDebugUtilsLabelEXT(
    VkQueue                                     queue,
    const VkDebugUtilsLabelEXT*                 pLabelInfo);
```

- `queue` is the queue in which to start a debug label region.
- `pLabelInfo` is a pointer to a VkDebugUtilsLabelEXT structure specifying parameters of the label region to open.

### Valid Usage (Implicit)

- VUID-vkQueueBeginDebugUtilsLabelEXT-queue-parameter
  `queue` **must** be a valid VkQueue handle
- VUID-vkQueueBeginDebugUtilsLabelEXT-pLabelInfo-parameter
  `pLabelInfo` **must** be a valid pointer to a valid VkDebugUtilsLabelEXT structure

### Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| - | - | - | Any | - |

The `VkDebugUtilsLabelEXT` structure is defined as:

```
// Provided by VK_EXT_debug_utils
typedef struct VkDebugUtilsLabelEXT {
    VkStructureType    sType;
    const void*        pNext;
    const char*        pLabelName;
    float              color[4];
} VkDebugUtilsLabelEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `pLabelName` is a pointer to a null-terminated UTF-8 string containing the name of the label.
- `color` is an optional RGBA color value that can be associated with the label. A particular implementation **may** choose to ignore this color value. The values contain RGBA values in order, in the range 0.0 to 1.0. If all elements in `color` are set to 0.0 then it is ignored.

### Valid Usage (Implicit)

- VUID-VkDebugUtilsLabelEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_DEBUG_UTILS_LABEL_EXT`
- VUID-VkDebugUtilsLabelEXT-pNext-pNext
  `pNext` **must** be `NULL`
- VUID-VkDebugUtilsLabelEXT-pLabelName-parameter
  `pLabelName` **must** be a null-terminated UTF-8 string

A queue debug label region is closed by calling:

```
// Provided by VK_EXT_debug_utils
void vkQueueEndDebugUtilsLabelEXT(
    VkQueue                                     queue);
```

- `queue` is the queue in which a debug label region should be closed.

The calls to vkQueueBeginDebugUtilsLabelEXT and vkQueueEndDebugUtilsLabelEXT **must** be matched and balanced.

### Valid Usage

- VUID-vkQueueEndDebugUtilsLabelEXT-None-01911
  There **must** be an outstanding `vkQueueBeginDebugUtilsLabelEXT` command prior to the `vkQueueEndDebugUtilsLabelEXT` on the queue

### Valid Usage (Implicit)

- VUID-vkQueueEndDebugUtilsLabelEXT-queue-parameter
  `queue` **must** be a valid VkQueue handle

### Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| - | - | - | Any | - |

A single label can be inserted into a queue by calling:

```
// Provided by VK_EXT_debug_utils
void vkQueueInsertDebugUtilsLabelEXT(
    VkQueue                                     queue,
    const VkDebugUtilsLabelEXT*                 pLabelInfo);
```

- `queue` is the queue into which a debug label will be inserted.
- `pLabelInfo` is a pointer to a VkDebugUtilsLabelEXT structure specifying parameters of the label to insert.

### Valid Usage (Implicit)

- VUID-vkQueueInsertDebugUtilsLabelEXT-queue-parameter
  `queue` **must** be a valid VkQueue handle
- VUID-vkQueueInsertDebugUtilsLabelEXT-pLabelInfo-parameter
  `pLabelInfo` **must** be a valid pointer to a valid VkDebugUtilsLabelEXT structure

### Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| - | - | - | Any | - |

## 50.1.3. Command Buffer Labels

Typical Vulkan applications will submit many command buffers in each frame, with each command buffer containing a large number of individual commands. Being able to logically annotate regions of command buffers that belong together as well as hierarchically subdivide the frame is important to a developer's ability to navigate the commands viewed holistically.

To identify the beginning of a debug label region in a command buffer, vkCmdBeginDebugUtilsLabelEXT **can** be used as defined below.

To indicate the end of a debug label region in a command buffer, vkCmdEndDebugUtilsLabelEXT **can** be used.

To insert a single command buffer debug label inside of a command buffer, vkCmdInsertDebugUtilsLabelEXT **can** be used as defined below.

A command buffer debug label region can be opened by calling:

```
// Provided by VK_EXT_debug_utils
void vkCmdBeginDebugUtilsLabelEXT(
    VkCommandBuffer                             commandBuffer,
    const VkDebugUtilsLabelEXT*                 pLabelInfo);
```

- `commandBuffer` is the command buffer into which the command is recorded.
- `pLabelInfo` is a pointer to a VkDebugUtilsLabelEXT structure specifying parameters of the label region to open.

## Valid Usage (Implicit)

- VUID-vkCmdBeginDebugUtilsLabelEXT-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdBeginDebugUtilsLabelEXT-pLabelInfo-parameter
  `pLabelInfo` **must** be a valid pointer to a valid VkDebugUtilsLabelEXT structure
- VUID-vkCmdBeginDebugUtilsLabelEXT-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdBeginDebugUtilsLabelEXT-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics, or compute operations
- VUID-vkCmdBeginDebugUtilsLabelEXT-videocoding
  This command **must** only be called outside of a video coding scope

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics<br>Compute | Action<br>State |

A command buffer label region can be closed by calling:

```
// Provided by VK_EXT_debug_utils
void vkCmdEndDebugUtilsLabelEXT(
    VkCommandBuffer                             commandBuffer);
```

- `commandBuffer` is the command buffer into which the command is recorded.

An application **may** open a debug label region in one command buffer and close it in another, or otherwise split debug label regions across multiple command buffers or multiple queue submissions. When viewed from the linear series of submissions to a single queue, the calls to vkCmdBeginDebugUtilsLabelEXT and vkCmdEndDebugUtilsLabelEXT **must** be matched and balanced.

There **can** be problems reporting command buffer debug labels during the recording process because command buffers **may** be recorded out of sequence with the resulting execution order. Since the recording order **may** be different, a solitary command buffer **may** have an inconsistent view of the debug label regions by itself. Therefore, if an issue occurs during the recording of a command buffer, and the environment requires returning debug labels, the implementation **may** return only those labels it is aware of. This is true even if the implementation is aware of only the debug labels within the command buffer being actively recorded.

### Valid Usage

- VUID-vkCmdEndDebugUtilsLabelEXT-commandBuffer-01912
  There **must** be an outstanding `vkCmdBeginDebugUtilsLabelEXT` command prior to the `vkCmdEndDebugUtilsLabelEXT` on the queue that `commandBuffer` is submitted to
- VUID-vkCmdEndDebugUtilsLabelEXT-commandBuffer-01913
  If `commandBuffer` is a secondary command buffer, there **must** be an outstanding `vkCmdBeginDebugUtilsLabelEXT` command recorded to `commandBuffer` that has not previously been ended by a call to `vkCmdEndDebugUtilsLabelEXT`

### Valid Usage (Implicit)

- VUID-vkCmdEndDebugUtilsLabelEXT-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdEndDebugUtilsLabelEXT-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdEndDebugUtilsLabelEXT-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics, or compute operations
- VUID-vkCmdEndDebugUtilsLabelEXT-videocoding
  This command **must** only be called outside of a video coding scope

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics<br>Compute | Action<br>State |

A single debug label can be inserted into a command buffer by calling:

```
// Provided by VK_EXT_debug_utils
void vkCmdInsertDebugUtilsLabelEXT(
    VkCommandBuffer                             commandBuffer,
    const VkDebugUtilsLabelEXT*                 pLabelInfo);
```

- `commandBuffer` is the command buffer into which the command is recorded.
- `pInfo` is a pointer to a VkDebugUtilsLabelEXT structure specifying parameters of the label to insert.

## Valid Usage (Implicit)

- VUID-vkCmdInsertDebugUtilsLabelEXT-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdInsertDebugUtilsLabelEXT-pLabelInfo-parameter
  `pLabelInfo` **must** be a valid pointer to a valid VkDebugUtilsLabelEXT structure
- VUID-vkCmdInsertDebugUtilsLabelEXT-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdInsertDebugUtilsLabelEXT-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics, or compute operations
- VUID-vkCmdInsertDebugUtilsLabelEXT-videocoding
  This command **must** only be called outside of a video coding scope

### Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

### Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics<br>Compute | Action |

## 50.1.4. Debug Messengers

Vulkan allows an application to register multiple callbacks with any Vulkan component wishing to report debug information. Some callbacks may log the information to a file, others may cause a debug break point or other application-defined behavior. A primary producer of callback messages are the validation layers. An application **can** register callbacks even when no validation layers are enabled, but they will only be called for the Vulkan loader and, if implemented, other layer and driver events.

A `VkDebugUtilsMessengerEXT` is a messenger object which handles passing along debug messages to a provided debug callback.

```
// Provided by VK_EXT_debug_utils
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkDebugUtilsMessengerEXT)
```

The debug messenger will provide detailed feedback on the application's use of Vulkan when events of interest occur. When an event of interest does occur, the debug messenger will submit a debug message to the debug callback that was provided during its creation. Additionally, the debug messenger is responsible with filtering out debug messages that the callback is not interested in and will only provide desired debug messages.

A debug messenger triggers a debug callback with a debug message when an event of interest occurs. To create a debug messenger which will trigger a debug callback, call:

```
// Provided by VK_EXT_debug_utils
VkResult vkCreateDebugUtilsMessengerEXT(
    VkInstance                                  instance,
    const VkDebugUtilsMessengerCreateInfoEXT*   pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkDebugUtilsMessengerEXT*                   pMessenger);
```

- `instance` is the instance the messenger will be used with.
- `pCreateInfo` is a pointer to a VkDebugUtilsMessengerCreateInfoEXT structure containing the callback pointer, as well as defining conditions under which this messenger will trigger the callback.
- `pAllocator` controls host memory allocation as described in the Memory Allocation chapter.
- `pMessenger` is a pointer to a VkDebugUtilsMessengerEXT handle in which the created object is returned.

### Valid Usage (Implicit)

- VUID-vkCreateDebugUtilsMessengerEXT-instance-parameter
  `instance` **must** be a valid VkInstance handle
- VUID-vkCreateDebugUtilsMessengerEXT-pCreateInfo-parameter
  `pCreateInfo` **must** be a valid pointer to a valid VkDebugUtilsMessengerCreateInfoEXT structure
- VUID-vkCreateDebugUtilsMessengerEXT-pAllocator-parameter
  If `pAllocator` is not `NULL`, `pAllocator` **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkCreateDebugUtilsMessengerEXT-pMessenger-parameter
  `pMessenger` **must** be a valid pointer to a VkDebugUtilsMessengerEXT handle

### Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- `VK_ERROR_OUT_OF_HOST_MEMORY`

The application **must** ensure that vkCreateDebugUtilsMessengerEXT is not executed in parallel with any Vulkan command that is also called with `instance` or child of `instance` as the dispatchable argument.

The definition of `VkDebugUtilsMessengerCreateInfoEXT` is:

```
// Provided by VK_EXT_debug_utils
typedef struct VkDebugUtilsMessengerCreateInfoEXT {
    VkStructureType                         sType;
    const void*                             pNext;
    VkDebugUtilsMessengerCreateFlagsEXT     flags;
    VkDebugUtilsMessageSeverityFlagsEXT     messageSeverity;
    VkDebugUtilsMessageTypeFlagsEXT         messageType;
    PFN_vkDebugUtilsMessengerCallbackEXT    pfnUserCallback;
    void*                                   pUserData;
} VkDebugUtilsMessengerCreateInfoEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is `0` and is reserved for future use.
- `messageSeverity` is a bitmask of VkDebugUtilsMessageSeverityFlagBitsEXT specifying which severity of event(s) will cause this callback to be called.
- `messageType` is a bitmask of VkDebugUtilsMessageTypeFlagBitsEXT specifying which type of event(s) will cause this callback to be called.
- `pfnUserCallback` is the application callback function to call.
- `pUserData` is user data to be passed to the callback.

For each `VkDebugUtilsMessengerEXT` that is created the `VkDebugUtilsMessengerCreateInfoEXT::messageSeverity` and `VkDebugUtilsMessengerCreateInfoEXT::messageType` determine when that `VkDebugUtilsMessengerCreateInfoEXT::pfnUserCallback` is called. The process to determine if the user's `pfnUserCallback` is triggered when an event occurs is as follows:

1. The implementation will perform a bitwise AND of the event's VkDebugUtilsMessageSeverityFlagBitsEXT with the `messageSeverity` provided during creation of the VkDebugUtilsMessengerEXT object.
   a. If the value is 0, the message is skipped.
2. The implementation will perform bitwise AND of the event's VkDebugUtilsMessageTypeFlagBitsEXT with the `messageType` provided during the creation of the VkDebugUtilsMessengerEXT object.
   a. If the value is 0, the message is skipped.
3. The callback will trigger a debug message for the current event

The callback will come directly from the component that detected the event, unless some other layer intercepts the calls for its own purposes (filter them in a different way, log to a system error log, etc.).

An application **can** receive multiple callbacks if multiple `VkDebugUtilsMessengerEXT` objects are created. A callback will always be executed in the same thread as the originating Vulkan call.

A callback **can** be called from multiple threads simultaneously (if the application is making Vulkan

calls from multiple threads).

### Valid Usage (Implicit)

- VUID-VkDebugUtilsMessengerCreateInfoEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_DEBUG_UTILS_MESSENGER_CREATE_INFO_EXT`
- VUID-VkDebugUtilsMessengerCreateInfoEXT-flags-zerobitmask
  `flags` **must** be `0`
- VUID-VkDebugUtilsMessengerCreateInfoEXT-messageSeverity-parameter
  `messageSeverity` **must** be a valid combination of VkDebugUtilsMessageSeverityFlagBitsEXT values
- VUID-VkDebugUtilsMessengerCreateInfoEXT-messageSeverity-requiredbitmask
  `messageSeverity` **must** not be `0`
- VUID-VkDebugUtilsMessengerCreateInfoEXT-messageType-parameter
  `messageType` **must** be a valid combination of VkDebugUtilsMessageTypeFlagBitsEXT values
- VUID-VkDebugUtilsMessengerCreateInfoEXT-messageType-requiredbitmask
  `messageType` **must** not be `0`
- VUID-VkDebugUtilsMessengerCreateInfoEXT-pfnUserCallback-parameter
  `pfnUserCallback` **must** be a valid PFN_vkDebugUtilsMessengerCallbackEXT value

```
// Provided by VK_EXT_debug_utils
typedef VkFlags VkDebugUtilsMessengerCreateFlagsEXT;
```

`VkDebugUtilsMessengerCreateFlagsEXT` is a bitmask type for setting a mask, but is currently reserved for future use.

Bits which **can** be set in VkDebugUtilsMessengerCreateInfoEXT::`messageSeverity`, specifying event severities which cause a debug messenger to call the callback, are:

```
// Provided by VK_EXT_debug_utils
typedef enum VkDebugUtilsMessageSeverityFlagBitsEXT {
    VK_DEBUG_UTILS_MESSAGE_SEVERITY_VERBOSE_BIT_EXT = 0x00000001,
    VK_DEBUG_UTILS_MESSAGE_SEVERITY_INFO_BIT_EXT = 0x00000010,
    VK_DEBUG_UTILS_MESSAGE_SEVERITY_WARNING_BIT_EXT = 0x00000100,
    VK_DEBUG_UTILS_MESSAGE_SEVERITY_ERROR_BIT_EXT = 0x00001000,
} VkDebugUtilsMessageSeverityFlagBitsEXT;
```

- `VK_DEBUG_UTILS_MESSAGE_SEVERITY_VERBOSE_BIT_EXT` specifies the most verbose output indicating all diagnostic messages from the Vulkan loader, layers, and drivers should be captured.
- `VK_DEBUG_UTILS_MESSAGE_SEVERITY_INFO_BIT_EXT` specifies an informational message such as resource details that may be handy when debugging an application.
- `VK_DEBUG_UTILS_MESSAGE_SEVERITY_WARNING_BIT_EXT` specifies use of Vulkan that **may** expose an

application bug. Such cases may not be immediately harmful, such as a fragment shader outputting to a location with no attachment. Other cases **may** point to behavior that is almost certainly bad when unintended such as using an image whose memory has not been filled. In general if you see a warning but you know that the behavior is intended/desired, then simply ignore the warning.

- VK_DEBUG_UTILS_MESSAGE_SEVERITY_ERROR_BIT_EXT specifies that the application has violated a valid usage condition of the specification.

**NOTE**

The values of VkDebugUtilsMessageSeverityFlagBitsEXT are sorted based on severity. The higher the flag value, the more severe the message. This allows for simple boolean operation comparisons when looking at VkDebugUtilsMessageSeverityFlagBitsEXT values.

For example:

```
    if (messageSeverity >=
VK_DEBUG_UTILS_MESSAGE_SEVERITY_WARNING_BIT_EXT) {
        // Do something for warnings and errors
    }
```

In addition, space has been left between the enums to allow for later addition of new severities in between the existing values.

```
// Provided by VK_EXT_debug_utils
typedef VkFlags VkDebugUtilsMessageSeverityFlagsEXT;
```

VkDebugUtilsMessageSeverityFlagsEXT is a bitmask type for setting a mask of zero or more VkDebugUtilsMessageSeverityFlagBitsEXT.

Bits which **can** be set in VkDebugUtilsMessengerCreateInfoEXT::messageType, specifying event types which cause a debug messenger to call the callback, are:

```
// Provided by VK_EXT_debug_utils
typedef enum VkDebugUtilsMessageTypeFlagBitsEXT {
    VK_DEBUG_UTILS_MESSAGE_TYPE_GENERAL_BIT_EXT = 0x00000001,
    VK_DEBUG_UTILS_MESSAGE_TYPE_VALIDATION_BIT_EXT = 0x00000002,
    VK_DEBUG_UTILS_MESSAGE_TYPE_PERFORMANCE_BIT_EXT = 0x00000004,
  // Provided by VK_EXT_device_address_binding_report
    VK_DEBUG_UTILS_MESSAGE_TYPE_DEVICE_ADDRESS_BINDING_BIT_EXT = 0x00000008,
} VkDebugUtilsMessageTypeFlagBitsEXT;
```

- VK_DEBUG_UTILS_MESSAGE_TYPE_GENERAL_BIT_EXT specifies that some general event has occurred. This is typically a non-specification, non-performance event.
- VK_DEBUG_UTILS_MESSAGE_TYPE_VALIDATION_BIT_EXT specifies that something has occurred during validation against the Vulkan specification that may indicate invalid behavior.

- VK_DEBUG_UTILS_MESSAGE_TYPE_PERFORMANCE_BIT_EXT specifies a potentially non-optimal use of Vulkan, e.g. using vkCmdClearColorImage when setting VkAttachmentDescription::loadOp to VK_ATTACHMENT_LOAD_OP_CLEAR would have worked.
- VK_DEBUG_UTILS_MESSAGE_TYPE_DEVICE_ADDRESS_BINDING_BIT_EXT specifies that the implementation has modified the set of GPU-visible virtual addresses associated with a Vulkan object.

```
// Provided by VK_EXT_debug_utils
typedef VkFlags VkDebugUtilsMessageTypeFlagsEXT;
```

VkDebugUtilsMessageTypeFlagsEXT is a bitmask type for setting a mask of zero or more VkDebugUtilsMessageTypeFlagBitsEXT.

The prototype for the VkDebugUtilsMessengerCreateInfoEXT::pfnUserCallback function implemented by the application is:

```
// Provided by VK_EXT_debug_utils
typedef VkBool32 (VKAPI_PTR *PFN_vkDebugUtilsMessengerCallbackEXT)(
    VkDebugUtilsMessageSeverityFlagBitsEXT           messageSeverity,
    VkDebugUtilsMessageTypeFlagsEXT                  messageTypes,
    const VkDebugUtilsMessengerCallbackDataEXT*      pCallbackData,
    void*                                            pUserData);
```

- messageSeverity specifies the VkDebugUtilsMessageSeverityFlagBitsEXT that triggered this callback.
- messageTypes is a bitmask of VkDebugUtilsMessageTypeFlagBitsEXT specifying which type of event(s) triggered this callback.
- pCallbackData contains all the callback related data in the VkDebugUtilsMessengerCallbackDataEXT structure.
- pUserData is the user data provided when the VkDebugUtilsMessengerEXT was created.

The callback returns a VkBool32, which is interpreted in a layer-specified manner. The application **should** always return VK_FALSE. The VK_TRUE value is reserved for use in layer development.

### Valid Usage

- VUID-PFN_vkDebugUtilsMessengerCallbackEXT-None-04769
  The callback **must** not make calls to any Vulkan commands

The definition of VkDebugUtilsMessengerCallbackDataEXT is:

```
// Provided by VK_EXT_debug_utils
typedef struct VkDebugUtilsMessengerCallbackDataEXT {
    VkStructureType                                 sType;
    const void*                                     pNext;
    VkDebugUtilsMessengerCallbackDataFlagsEXT       flags;
    const char*                                     pMessageIdName;
    int32_t                                         messageIdNumber;
    const char*                                     pMessage;
    uint32_t                                        queueLabelCount;
    const VkDebugUtilsLabelEXT*                     pQueueLabels;
    uint32_t                                        cmdBufLabelCount;
    const VkDebugUtilsLabelEXT*                     pCmdBufLabels;
    uint32_t                                        objectCount;
    const VkDebugUtilsObjectNameInfoEXT*            pObjects;
} VkDebugUtilsMessengerCallbackDataEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is `0` and is reserved for future use.
- `pMessageIdName` is `NULL` or a null-terminated UTF-8 string that identifies the particular message ID that is associated with the provided message. If the message corresponds to a validation layer message, then this string may contain the portion of the Vulkan specification that is believed to have been violated.
- `messageIdNumber` is the ID number of the triggering message. If the message corresponds to a validation layer message, then this number is related to the internal number associated with the message being triggered.
- `pMessage` is `NULL` if `messageTypes` is equal to `VK_DEBUG_UTILS_MESSAGE_TYPE_DEVICE_ADDRESS_BINDING_BIT_EXT`, or a null-terminated UTF-8 string detailing the trigger conditions.
- `queueLabelCount` is a count of items contained in the `pQueueLabels` array.
- `pQueueLabels` is `NULL` or a pointer to an array of VkDebugUtilsLabelEXT active in the current `VkQueue` at the time the callback was triggered. Refer to Queue Labels for more information.
- `cmdBufLabelCount` is a count of items contained in the `pCmdBufLabels` array.
- `pCmdBufLabels` is `NULL` or a pointer to an array of VkDebugUtilsLabelEXT active in the current `VkCommandBuffer` at the time the callback was triggered. Refer to Command Buffer Labels for more information.
- `objectCount` is a count of items contained in the `pObjects` array.
- `pObjects` is a pointer to an array of VkDebugUtilsObjectNameInfoEXT objects related to the detected issue. The array is roughly in order or importance, but the 0th element is always guaranteed to be the most important object for this message.

| | |
|---|---|
| **NOTE** | This structure should only be considered valid during the lifetime of the triggered callback. |

Since adding queue and command buffer labels behaves like pushing and popping onto a stack, the order of both `pQueueLabels` and `pCmdBufLabels` is based on the order the labels were defined. The result is that the first label in either `pQueueLabels` or `pCmdBufLabels` will be the first defined (and therefore the oldest) while the last label in each list will be the most recent.

**NOTE**

`pQueueLabels` will only be non-`NULL` if one of the objects in `pObjects` can be related directly to a defined `VkQueue` which has had one or more labels associated with it.

Likewise, `pCmdBufLabels` will only be non-`NULL` if one of the objects in `pObjects` can be related directly to a defined `VkCommandBuffer` which has had one or more labels associated with it. Additionally, while command buffer labels allow for beginning and ending across different command buffers, the debug messaging framework **cannot** guarantee that labels in `pCmdBufLables` will contain those defined outside of the associated command buffer. This is partially due to the fact that the association of one command buffer with another may not have been defined at the time the debug message is triggered.

## Valid Usage (Implicit)

- VUID-VkDebugUtilsMessengerCallbackDataEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_DEBUG_UTILS_MESSENGER_CALLBACK_DATA_EXT`
- VUID-VkDebugUtilsMessengerCallbackDataEXT-pNext-pNext
  `pNext` **must** be `NULL` or a pointer to a valid instance of VkDeviceAddressBindingCallbackDataEXT
- VUID-VkDebugUtilsMessengerCallbackDataEXT-sType-unique
  The `sType` value of each struct in the `pNext` chain **must** be unique
- VUID-VkDebugUtilsMessengerCallbackDataEXT-flags-zerobitmask
  `flags` **must** be `0`
- VUID-VkDebugUtilsMessengerCallbackDataEXT-pMessageIdName-parameter
  If `pMessageIdName` is not `NULL`, `pMessageIdName` **must** be a null-terminated UTF-8 string
- VUID-VkDebugUtilsMessengerCallbackDataEXT-pMessage-parameter
  If `pMessage` is not `NULL`, `pMessage` **must** be a null-terminated UTF-8 string
- VUID-VkDebugUtilsMessengerCallbackDataEXT-pQueueLabels-parameter
  If `queueLabelCount` is not `0`, `pQueueLabels` **must** be a valid pointer to an array of `queueLabelCount` valid VkDebugUtilsLabelEXT structures
- VUID-VkDebugUtilsMessengerCallbackDataEXT-pCmdBufLabels-parameter
  If `cmdBufLabelCount` is not `0`, `pCmdBufLabels` **must** be a valid pointer to an array of `cmdBufLabelCount` valid VkDebugUtilsLabelEXT structures
- VUID-VkDebugUtilsMessengerCallbackDataEXT-pObjects-parameter
  If `objectCount` is not `0`, `pObjects` **must** be a valid pointer to an array of `objectCount` valid VkDebugUtilsObjectNameInfoEXT structures

```
// Provided by VK_EXT_debug_utils
typedef VkFlags VkDebugUtilsMessengerCallbackDataFlagsEXT;
```

VkDebugUtilsMessengerCallbackDataFlagsEXT is a bitmask type for setting a mask, but is currently reserved for future use.

The definition of VkDeviceAddressBindingCallbackDataEXT is:

```
// Provided by VK_EXT_device_address_binding_report
typedef struct VkDeviceAddressBindingCallbackDataEXT {
    VkStructureType                    sType;
    void*                              pNext;
    VkDeviceAddressBindingFlagsEXT     flags;
    VkDeviceAddress                    baseAddress;
    VkDeviceSize                       size;
    VkDeviceAddressBindingTypeEXT      bindingType;
} VkDeviceAddressBindingCallbackDataEXT;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- flags is a bitmask of VkDeviceAddressBindingFlagBitsEXT specifying additional information about the binding event that caused the callback to be called.
- baseAddress is a GPU-accessible virtual address identifying the start of a region of the virtual address space associated with a Vulkan object, as identified by the pObjects member of VkDebugUtilsMessengerCallbackDataEXT.
- size is the size in bytes of a region of GPU-accessible virtual address space.
- bindingType is a VkDeviceAddressBindingTypeEXT specifying the type of binding event that caused the callback to be called.

If the reportAddressBinding feature is enabled and the implementation binds or unbinds a region of virtual address space associated with a Vulkan object, the implementation **must** submit a debug message with the following properties:

- messageSeverity equal to VK_DEBUG_UTILS_MESSAGE_SEVERITY_INFO_BIT_EXT
- messageTypes equal to VK_DEBUG_UTILS_MESSAGE_TYPE_DEVICE_ADDRESS_BINDING_BIT_EXT
- VkDebugUtilsMessengerCallbackDataEXT::pObjects **must** identify the associated Vulkan object
- VkDeviceAddressBindingCallbackDataEXT **must** be included in the pNext chain of VkDebugUtilsMessengerCallbackDataEXT

These debug messages **must** be emitted both for GPU virtual address space regions that are explicitly bound to a Vulkan object via the vkBind*Memory/vkBind*Memory2 functions, and for those that are implicitly generated via memory allocation or importing external memory.

An implementation **may** report binding events associated with a Vulkan object via

VkDebugUtilsMessengerEXT prior to the object becoming visible to an application via other Vulkan commands. For example, object creation functions **may** report binding events that occur during an objects creation. In such cases, VkDeviceAddressBindingCallbackDataEXT::flags **must** include VK_DEVICE_ADDRESS_BINDING_INTERNAL_OBJECT_BIT_EXT.

Object handles reported in this manner are not valid object handles, and **must** not be used as an input parameter to any Vulkan command.

Any valid object handle returned by an object creation function **must** match the handle specified via any previously reported binding events associated with the object's creation.

### Valid Usage (Implicit)

- VUID-VkDeviceAddressBindingCallbackDataEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_DEVICE_ADDRESS_BINDING_CALLBACK_DATA_EXT
- VUID-VkDeviceAddressBindingCallbackDataEXT-flags-parameter
  flags **must** be a valid combination of VkDeviceAddressBindingFlagBitsEXT values
- VUID-VkDeviceAddressBindingCallbackDataEXT-bindingType-parameter
  bindingType **must** be a valid VkDeviceAddressBindingTypeEXT value

Bits which **can** be set in VkDeviceAddressBindingCallbackDataEXT::flags specifying additional information about a binding event are:

```
// Provided by VK_EXT_device_address_binding_report
typedef enum VkDeviceAddressBindingFlagBitsEXT {
    VK_DEVICE_ADDRESS_BINDING_INTERNAL_OBJECT_BIT_EXT = 0x00000001,
} VkDeviceAddressBindingFlagBitsEXT;
```

- VK_DEVICE_ADDRESS_BINDING_INTERNAL_OBJECT_BIT_EXT specifies that VkDeviceAddressBindingCallbackDataEXT describes a Vulkan object that has not been made visible to the application via a Vulkan command.

```
// Provided by VK_EXT_device_address_binding_report
typedef VkFlags VkDeviceAddressBindingFlagsEXT;
```

VkDeviceAddressBindingFlagsEXT is a bitmask type for setting a mask of zero or more VkDeviceAddressBindingFlagBitsEXT.

The VkDeviceAddressBindingTypeEXT enum is defined as:

```
// Provided by VK_EXT_device_address_binding_report
typedef enum VkDeviceAddressBindingTypeEXT {
    VK_DEVICE_ADDRESS_BINDING_TYPE_BIND_EXT = 0,
    VK_DEVICE_ADDRESS_BINDING_TYPE_UNBIND_EXT = 1,
} VkDeviceAddressBindingTypeEXT;
```

- `VK_DEVICE_ADDRESS_BINDING_TYPE_BIND_EXT` specifies that a new GPU-accessible virtual address range has been bound.
- `VK_DEVICE_ADDRESS_BINDING_TYPE_UNBIND_EXT` specifies that a GPU-accessible virtual address range has been unbound.

To intentionally submit a debug message, call:

```
// Provided by VK_EXT_debug_utils
void vkSubmitDebugUtilsMessageEXT(
    VkInstance                                  instance,
    VkDebugUtilsMessageSeverityFlagBitsEXT      messageSeverity,
    VkDebugUtilsMessageTypeFlagsEXT             messageTypes,
    const VkDebugUtilsMessengerCallbackDataEXT* pCallbackData);
```

- `instance` is the debug stream's VkInstance.
- `messageSeverity` is a VkDebugUtilsMessageSeverityFlagBitsEXT value specifying the severity of this event/message.
- `messageTypes` is a bitmask of VkDebugUtilsMessageTypeFlagBitsEXT specifying which type of event(s) to identify with this message.
- `pCallbackData` contains all the callback related data in the VkDebugUtilsMessengerCallbackDataEXT structure.

The call will propagate through the layers and generate callback(s) as indicated by the message's flags. The parameters are passed on to the callback in addition to the `pUserData` value that was defined at the time the messenger was registered.

### Valid Usage

- VUID-vkSubmitDebugUtilsMessageEXT-objectType-02591
  The `objectType` member of each element of `pCallbackData->pObjects` **must** not be `VK_OBJECT_TYPE_UNKNOWN`

### Valid Usage (Implicit)

- VUID-vkSubmitDebugUtilsMessageEXT-instance-parameter
  `instance` **must** be a valid VkInstance handle
- VUID-vkSubmitDebugUtilsMessageEXT-messageSeverity-parameter

messageSeverity **must** be a valid VkDebugUtilsMessageSeverityFlagBitsEXT value

- VUID-vkSubmitDebugUtilsMessageEXT-messageTypes-parameter
  messageTypes **must** be a valid combination of VkDebugUtilsMessageTypeFlagBitsEXT values
- VUID-vkSubmitDebugUtilsMessageEXT-messageTypes-requiredbitmask
  messageTypes **must** not be 0
- VUID-vkSubmitDebugUtilsMessageEXT-pCallbackData-parameter
  pCallbackData **must** be a valid pointer to a valid VkDebugUtilsMessengerCallbackDataEXT structure

To destroy a VkDebugUtilsMessengerEXT object, call:

```
// Provided by VK_EXT_debug_utils
void vkDestroyDebugUtilsMessengerEXT(
    VkInstance                                  instance,
    VkDebugUtilsMessengerEXT                    messenger,
    const VkAllocationCallbacks*                pAllocator);
```

- instance is the instance where the callback was created.
- messenger is the VkDebugUtilsMessengerEXT object to destroy. messenger is an externally synchronized object and **must** not be used on more than one thread at a time. This means that vkDestroyDebugUtilsMessengerEXT **must** not be called when a callback is active.
- pAllocator controls host memory allocation as described in the Memory Allocation chapter.

### Valid Usage

- VUID-vkDestroyDebugUtilsMessengerEXT-messenger-01915
  If VkAllocationCallbacks were provided when messenger was created, a compatible set of callbacks **must** be provided here
- VUID-vkDestroyDebugUtilsMessengerEXT-messenger-01916
  If no VkAllocationCallbacks were provided when messenger was created, pAllocator **must** be NULL

### Valid Usage (Implicit)

- VUID-vkDestroyDebugUtilsMessengerEXT-instance-parameter
  instance **must** be a valid VkInstance handle
- VUID-vkDestroyDebugUtilsMessengerEXT-messenger-parameter
  If messenger is not VK_NULL_HANDLE, messenger **must** be a valid VkDebugUtilsMessengerEXT handle
- VUID-vkDestroyDebugUtilsMessengerEXT-pAllocator-parameter
  If pAllocator is not NULL, pAllocator **must** be a valid pointer to a valid

VkAllocationCallbacks structure

- VUID-vkDestroyDebugUtilsMessengerEXT-messenger-parent
  If `messenger` is a valid handle, it **must** have been created, allocated, or retrieved from `instance`

**Host Synchronization**

- Host access to `messenger` **must** be externally synchronized

The application **must** ensure that vkDestroyDebugUtilsMessengerEXT is not executed in parallel with any Vulkan command that is also called with `instance` or child of `instance` as the dispatchable argument.

# 50.2. Debug Markers

Debug markers provide a flexible way for debugging and validation layers to receive annotation and debug information.

The Object Annotation section describes how to associate a name or binary data with a Vulkan object.

The Command Buffer Markers section describes how to associate logical elements of the scene with commands in the command buffer.

## 50.2.1. Object Annotation

The commands in this section allow application developers to associate application-defined information with Vulkan objects at will.

An object can be given an application-defined name by calling:

```
// Provided by VK_EXT_debug_marker
VkResult vkDebugMarkerSetObjectNameEXT(
    VkDevice                                    device,
    const VkDebugMarkerObjectNameInfoEXT*       pNameInfo);
```

- `device` is the device that created the object.
- `pNameInfo` is a pointer to a VkDebugMarkerObjectNameInfoEXT structure specifying the parameters of the name to set on the object.

**Valid Usage (Implicit)**

- VUID-vkDebugMarkerSetObjectNameEXT-device-parameter
  `device` **must** be a valid VkDevice handle

- VUID-vkDebugMarkerSetObjectNameEXT-pNameInfo-parameter
  `pNameInfo` **must** be a valid pointer to a valid VkDebugMarkerObjectNameInfoEXT structure

## Host Synchronization

- Host access to `pNameInfo->object` **must** be externally synchronized

## Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`

The `VkDebugMarkerObjectNameInfoEXT` structure is defined as:

```
// Provided by VK_EXT_debug_marker
typedef struct VkDebugMarkerObjectNameInfoEXT {
    VkStructureType               sType;
    const void*                   pNext;
    VkDebugReportObjectTypeEXT    objectType;
    uint64_t                      object;
    const char*                   pObjectName;
} VkDebugMarkerObjectNameInfoEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `objectType` is a VkDebugReportObjectTypeEXT specifying the type of the object to be named.
- `object` is the object to be named.
- `pObjectName` is a null-terminated UTF-8 string specifying the name to apply to `object`.

Applications **may** change the name associated with an object simply by calling `vkDebugMarkerSetObjectNameEXT` again with a new string. To remove a previously set name, `pObjectName` **should** be set to an empty string.

## Valid Usage

- VUID-VkDebugMarkerObjectNameInfoEXT-objectType-01490
  `objectType` **must** not be `VK_DEBUG_REPORT_OBJECT_TYPE_UNKNOWN_EXT`

- VUID-VkDebugMarkerObjectNameInfoEXT-object-01491
  `object` **must** not be VK_NULL_HANDLE
- VUID-VkDebugMarkerObjectNameInfoEXT-object-01492
  `object` **must** be a Vulkan object of the type associated with `objectType` as defined in `VkDebugReportObjectTypeEXT` and Vulkan Handle Relationship

### Valid Usage (Implicit)

- VUID-VkDebugMarkerObjectNameInfoEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_DEBUG_MARKER_OBJECT_NAME_INFO_EXT`
- VUID-VkDebugMarkerObjectNameInfoEXT-pNext-pNext
  `pNext` **must** be `NULL`
- VUID-VkDebugMarkerObjectNameInfoEXT-objectType-parameter
  `objectType` **must** be a valid VkDebugReportObjectTypeEXT value
- VUID-VkDebugMarkerObjectNameInfoEXT-pObjectName-parameter
  `pObjectName` **must** be a null-terminated UTF-8 string

In addition to setting a name for an object, debugging and validation layers may have uses for additional binary data on a per-object basis that has no other place in the Vulkan API. For example, a `VkShaderModule` could have additional debugging data attached to it to aid in offline shader tracing. To attach data to an object, call:

```
// Provided by VK_EXT_debug_marker
VkResult vkDebugMarkerSetObjectTagEXT(
    VkDevice                                    device,
    const VkDebugMarkerObjectTagInfoEXT*        pTagInfo);
```

- `device` is the device that created the object.
- `pTagInfo` is a pointer to a VkDebugMarkerObjectTagInfoEXT structure specifying the parameters of the tag to attach to the object.

### Valid Usage (Implicit)

- VUID-vkDebugMarkerSetObjectTagEXT-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkDebugMarkerSetObjectTagEXT-pTagInfo-parameter
  `pTagInfo` **must** be a valid pointer to a valid VkDebugMarkerObjectTagInfoEXT structure

### Host Synchronization

- Host access to `pTagInfo->object` **must** be externally synchronized

### Return Codes

**Success**

- VK_SUCCESS

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY

The VkDebugMarkerObjectTagInfoEXT structure is defined as:

```
// Provided by VK_EXT_debug_marker
typedef struct VkDebugMarkerObjectTagInfoEXT {
    VkStructureType                 sType;
    const void*                     pNext;
    VkDebugReportObjectTypeEXT      objectType;
    uint64_t                        object;
    uint64_t                        tagName;
    size_t                          tagSize;
    const void*                     pTag;
} VkDebugMarkerObjectTagInfoEXT;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- objectType is a VkDebugReportObjectTypeEXT specifying the type of the object to be named.
- object is the object to be tagged.
- tagName is a numerical identifier of the tag.
- tagSize is the number of bytes of data to attach to the object.
- pTag is a pointer to an array of tagSize bytes containing the data to be associated with the object.

The tagName parameter gives a name or identifier to the type of data being tagged. This can be used by debugging layers to easily filter for only data that can be used by that implementation.

### Valid Usage

- VUID-VkDebugMarkerObjectTagInfoEXT-objectType-01493
  objectType **must** not be VK_DEBUG_REPORT_OBJECT_TYPE_UNKNOWN_EXT
- VUID-VkDebugMarkerObjectTagInfoEXT-object-01494
  object **must** not be VK_NULL_HANDLE
- VUID-VkDebugMarkerObjectTagInfoEXT-object-01495
  object **must** be a Vulkan object of the type associated with objectType as defined in VkDebugReportObjectTypeEXT and Vulkan Handle Relationship

### Valid Usage (Implicit)

- VUID-VkDebugMarkerObjectTagInfoEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_DEBUG_MARKER_OBJECT_TAG_INFO_EXT`
- VUID-VkDebugMarkerObjectTagInfoEXT-pNext-pNext
  `pNext` **must** be `NULL`
- VUID-VkDebugMarkerObjectTagInfoEXT-objectType-parameter
  `objectType` **must** be a valid VkDebugReportObjectTypeEXT value
- VUID-VkDebugMarkerObjectTagInfoEXT-pTag-parameter
  `pTag` **must** be a valid pointer to an array of `tagSize` bytes
- VUID-VkDebugMarkerObjectTagInfoEXT-tagSize-arraylength
  `tagSize` **must** be greater than `0`

## 50.2.2. Command Buffer Markers

Typical Vulkan applications will submit many command buffers in each frame, with each command buffer containing a large number of individual commands. Being able to logically annotate regions of command buffers that belong together as well as hierarchically subdivide the frame is important to a developer's ability to navigate the commands viewed holistically.

The marker commands `vkCmdDebugMarkerBeginEXT` and `vkCmdDebugMarkerEndEXT` define regions of a series of commands that are grouped together, and they can be nested to create a hierarchy. The `vkCmdDebugMarkerInsertEXT` command allows insertion of a single label within a command buffer.

A marker region can be opened by calling:

```
// Provided by VK_EXT_debug_marker
void vkCmdDebugMarkerBeginEXT(
    VkCommandBuffer                             commandBuffer,
    const VkDebugMarkerMarkerInfoEXT*           pMarkerInfo);
```

- `commandBuffer` is the command buffer into which the command is recorded.
- `pMarkerInfo` is a pointer to a VkDebugMarkerMarkerInfoEXT structure specifying the parameters of the marker region to open.

### Valid Usage (Implicit)

- VUID-vkCmdDebugMarkerBeginEXT-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdDebugMarkerBeginEXT-pMarkerInfo-parameter
  `pMarkerInfo` **must** be a valid pointer to a valid VkDebugMarkerMarkerInfoEXT structure
- VUID-vkCmdDebugMarkerBeginEXT-commandBuffer-recording
  `commandBuffer` **must** be in the recording state

- VUID-vkCmdDebugMarkerBeginEXT-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics, or compute operations
- VUID-vkCmdDebugMarkerBeginEXT-videocoding
  This command **must** only be called outside of a video coding scope

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics<br>Compute | Action |

The `VkDebugMarkerMarkerInfoEXT` structure is defined as:

```
// Provided by VK_EXT_debug_marker
typedef struct VkDebugMarkerMarkerInfoEXT {
    VkStructureType    sType;
    const void*        pNext;
    const char*        pMarkerName;
    float              color[4];
} VkDebugMarkerMarkerInfoEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `pMarkerName` is a pointer to a null-terminated UTF-8 string containing the name of the marker.
- `color` is an **optional** RGBA color value that can be associated with the marker. A particular implementation **may** choose to ignore this color value. The values contain RGBA values in order, in the range 0.0 to 1.0. If all elements in `color` are set to 0.0 then it is ignored.

## Valid Usage (Implicit)

- VUID-VkDebugMarkerMarkerInfoEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_DEBUG_MARKER_MARKER_INFO_EXT`
- VUID-VkDebugMarkerMarkerInfoEXT-pNext-pNext

pNext **must** be NULL

- VUID-VkDebugMarkerMarkerInfoEXT-pMarkerName-parameter
  pMarkerName **must** be a null-terminated UTF-8 string

A marker region can be closed by calling:

```
// Provided by VK_EXT_debug_marker
void vkCmdDebugMarkerEndEXT(
    VkCommandBuffer                             commandBuffer);
```

- commandBuffer is the command buffer into which the command is recorded.

An application **may** open a marker region in one command buffer and close it in another, or otherwise split marker regions across multiple command buffers or multiple queue submissions. When viewed from the linear series of submissions to a single queue, the calls to vkCmdDebugMarkerBeginEXT and vkCmdDebugMarkerEndEXT **must** be matched and balanced.

### Valid Usage

- VUID-vkCmdDebugMarkerEndEXT-commandBuffer-01239
  There **must** be an outstanding vkCmdDebugMarkerBeginEXT command prior to the vkCmdDebugMarkerEndEXT on the queue that commandBuffer is submitted to
- VUID-vkCmdDebugMarkerEndEXT-commandBuffer-01240
  If commandBuffer is a secondary command buffer, there **must** be an outstanding vkCmdDebugMarkerBeginEXT command recorded to commandBuffer that has not previously been ended by a call to vkCmdDebugMarkerEndEXT

### Valid Usage (Implicit)

- VUID-vkCmdDebugMarkerEndEXT-commandBuffer-parameter
  commandBuffer **must** be a valid VkCommandBuffer handle
- VUID-vkCmdDebugMarkerEndEXT-commandBuffer-recording
  commandBuffer **must** be in the recording state
- VUID-vkCmdDebugMarkerEndEXT-commandBuffer-cmdpool
  The VkCommandPool that commandBuffer was allocated from **must** support graphics, or compute operations
- VUID-vkCmdDebugMarkerEndEXT-videocoding
  This command **must** only be called outside of a video coding scope

### Host Synchronization

- Host access to commandBuffer **must** be externally synchronized

- Host access to the VkCommandPool that commandBuffer was allocated from **must** be externally synchronized

### Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics<br>Compute | Action |

A single marker label can be inserted into a command buffer by calling:

```
// Provided by VK_EXT_debug_marker
void vkCmdDebugMarkerInsertEXT(
    VkCommandBuffer                             commandBuffer,
    const VkDebugMarkerMarkerInfoEXT*           pMarkerInfo);
```

- commandBuffer is the command buffer into which the command is recorded.
- pMarkerInfo is a pointer to a VkDebugMarkerMarkerInfoEXT structure specifying the parameters of the marker to insert.

### Valid Usage (Implicit)

- VUID-vkCmdDebugMarkerInsertEXT-commandBuffer-parameter
  commandBuffer **must** be a valid VkCommandBuffer handle
- VUID-vkCmdDebugMarkerInsertEXT-pMarkerInfo-parameter
  pMarkerInfo **must** be a valid pointer to a valid VkDebugMarkerMarkerInfoEXT structure
- VUID-vkCmdDebugMarkerInsertEXT-commandBuffer-recording
  commandBuffer **must** be in the recording state
- VUID-vkCmdDebugMarkerInsertEXT-commandBuffer-cmdpool
  The VkCommandPool that commandBuffer was allocated from **must** support graphics, or compute operations
- VUID-vkCmdDebugMarkerInsertEXT-videocoding
  This command **must** only be called outside of a video coding scope

### Host Synchronization

- Host access to commandBuffer **must** be externally synchronized
- Host access to the VkCommandPool that commandBuffer was allocated from **must** be externally synchronized

**Command Properties**

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics<br>Compute | Action |

## 50.3. Debug Report Callbacks

Debug report callbacks are represented by `VkDebugReportCallbackEXT` handles:

```
// Provided by VK_EXT_debug_report
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkDebugReportCallbackEXT)
```

Debug report callbacks give more detailed feedback on the application's use of Vulkan when events of interest occur.

To register a debug report callback, an application uses vkCreateDebugReportCallbackEXT.

```
// Provided by VK_EXT_debug_report
VkResult vkCreateDebugReportCallbackEXT(
    VkInstance                                  instance,
    const VkDebugReportCallbackCreateInfoEXT*   pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkDebugReportCallbackEXT*                   pCallback);
```

- `instance` is the instance the callback will be logged on.
- `pCreateInfo` is a pointer to a VkDebugReportCallbackCreateInfoEXT structure defining the conditions under which this callback will be called.
- `pAllocator` controls host memory allocation as described in the Memory Allocation chapter.
- `pCallback` is a pointer to a VkDebugReportCallbackEXT handle in which the created object is returned.

**Valid Usage (Implicit)**

- VUID-vkCreateDebugReportCallbackEXT-instance-parameter
  `instance` **must** be a valid VkInstance handle
- VUID-vkCreateDebugReportCallbackEXT-pCreateInfo-parameter
  `pCreateInfo` **must** be a valid pointer to a valid VkDebugReportCallbackCreateInfoEXT structure
- VUID-vkCreateDebugReportCallbackEXT-pAllocator-parameter
  If `pAllocator` is not `NULL`, `pAllocator` **must** be a valid pointer to a valid

VkAllocationCallbacks structure

- VUID-vkCreateDebugReportCallbackEXT-pCallback-parameter
  `pCallback` **must** be a valid pointer to a VkDebugReportCallbackEXT handle

## Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- `VK_ERROR_OUT_OF_HOST_MEMORY`

The definition of VkDebugReportCallbackCreateInfoEXT is:

```
// Provided by VK_EXT_debug_report
typedef struct VkDebugReportCallbackCreateInfoEXT {
    VkStructureType                 sType;
    const void*                     pNext;
    VkDebugReportFlagsEXT           flags;
    PFN_vkDebugReportCallbackEXT    pfnCallback;
    void*                           pUserData;
} VkDebugReportCallbackCreateInfoEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is a bitmask of VkDebugReportFlagBitsEXT specifying which event(s) will cause this callback to be called.
- `pfnCallback` is the application callback function to call.
- `pUserData` is user data to be passed to the callback.

For each `VkDebugReportCallbackEXT` that is created the `VkDebugReportCallbackCreateInfoEXT::flags` determine when that `VkDebugReportCallbackCreateInfoEXT::pfnCallback` is called. When an event happens, the implementation will do a bitwise AND of the event's VkDebugReportFlagBitsEXT flags to each `VkDebugReportCallbackEXT` object's flags. For each non-zero result the corresponding callback will be called. The callback will come directly from the component that detected the event, unless some other layer intercepts the calls for its own purposes (filter them in a different way, log to a system error log, etc.).

An application **may** receive multiple callbacks if multiple `VkDebugReportCallbackEXT` objects were created. A callback will always be executed in the same thread as the originating Vulkan call.

A callback may be called from multiple threads simultaneously (if the application is making Vulkan calls from multiple threads).

### Valid Usage (Implicit)

- VUID-VkDebugReportCallbackCreateInfoEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_DEBUG_REPORT_CALLBACK_CREATE_INFO_EXT`
- VUID-VkDebugReportCallbackCreateInfoEXT-flags-parameter
  `flags` **must** be a valid combination of VkDebugReportFlagBitsEXT values
- VUID-VkDebugReportCallbackCreateInfoEXT-pfnCallback-parameter
  `pfnCallback` **must** be a valid PFN_vkDebugReportCallbackEXT value

Bits which **can** be set in VkDebugReportCallbackCreateInfoEXT::`flags`, specifying events which cause a debug report, are:

```
// Provided by VK_EXT_debug_report
typedef enum VkDebugReportFlagBitsEXT {
    VK_DEBUG_REPORT_INFORMATION_BIT_EXT = 0x00000001,
    VK_DEBUG_REPORT_WARNING_BIT_EXT = 0x00000002,
    VK_DEBUG_REPORT_PERFORMANCE_WARNING_BIT_EXT = 0x00000004,
    VK_DEBUG_REPORT_ERROR_BIT_EXT = 0x00000008,
    VK_DEBUG_REPORT_DEBUG_BIT_EXT = 0x00000010,
} VkDebugReportFlagBitsEXT;
```

- `VK_DEBUG_REPORT_ERROR_BIT_EXT` specifies that the application has violated a valid usage condition of the specification.
- `VK_DEBUG_REPORT_WARNING_BIT_EXT` specifies use of Vulkan that **may** expose an application bug. Such cases may not be immediately harmful, such as a fragment shader outputting to a location with no attachment. Other cases **may** point to behavior that is almost certainly bad when unintended such as using an image whose memory has not been filled. In general if you see a warning but you know that the behavior is intended/desired, then simply ignore the warning.
- `VK_DEBUG_REPORT_PERFORMANCE_WARNING_BIT_EXT` specifies a potentially non-optimal use of Vulkan, e.g. using vkCmdClearColorImage when setting VkAttachmentDescription::`loadOp` to `VK_ATTACHMENT_LOAD_OP_CLEAR` would have worked.
- `VK_DEBUG_REPORT_INFORMATION_BIT_EXT` specifies an informational message such as resource details that may be handy when debugging an application.
- `VK_DEBUG_REPORT_DEBUG_BIT_EXT` specifies diagnostic information from the implementation and layers.

```
// Provided by VK_EXT_debug_report
typedef VkFlags VkDebugReportFlagsEXT;
```

`VkDebugReportFlagsEXT` is a bitmask type for setting a mask of zero or more VkDebugReportFlagBitsEXT.

The prototype for the VkDebugReportCallbackCreateInfoEXT::`pfnCallback` function implemented by

the application is:

```
// Provided by VK_EXT_debug_report
typedef VkBool32 (VKAPI_PTR *PFN_vkDebugReportCallbackEXT)(
    VkDebugReportFlagsEXT                       flags,
    VkDebugReportObjectTypeEXT                  objectType,
    uint64_t                                    object,
    size_t                                      location,
    int32_t                                     messageCode,
    const char*                                 pLayerPrefix,
    const char*                                 pMessage,
    void*                                       pUserData);
```

- `flags` specifies the VkDebugReportFlagBitsEXT that triggered this callback.
- `objectType` is a VkDebugReportObjectTypeEXT value specifying the type of object being used or created at the time the event was triggered.
- `object` is the object where the issue was detected. If `objectType` is `VK_DEBUG_REPORT_OBJECT_TYPE_UNKNOWN_EXT`, `object` is undefined.
- `location` is a component (layer, driver, loader) defined value specifying the *location* of the trigger. This is an **optional** value.
- `messageCode` is a layer-defined value indicating what test triggered this callback.
- `pLayerPrefix` is a null-terminated UTF-8 string that is an abbreviation of the name of the component making the callback. `pLayerPrefix` is only valid for the duration of the callback.
- `pMessage` is a null-terminated UTF-8 string detailing the trigger conditions. `pMessage` is only valid for the duration of the callback.
- `pUserData` is the user data given when the VkDebugReportCallbackEXT was created.

The callback **must** not call `vkDestroyDebugReportCallbackEXT`.

The callback returns a VkBool32, which is interpreted in a layer-specified manner. The application **should** always return `VK_FALSE`. The `VK_TRUE` value is reserved for use in layer development.

`object` **must** be a Vulkan object or VK_NULL_HANDLE. If `objectType` is not `VK_DEBUG_REPORT_OBJECT_TYPE_UNKNOWN_EXT` and `object` is not VK_NULL_HANDLE, `object` **must** be a Vulkan object of the corresponding type associated with `objectType` as defined in `VkDebugReportObjectTypeEXT` and Vulkan Handle Relationship.

Possible values passed to the `objectType` parameter of the callback function specified by VkDebugReportCallbackCreateInfoEXT::`pfnCallback`, specifying the type of object handle being reported, are:

```
// Provided by VK_EXT_debug_marker, VK_EXT_debug_report
typedef enum VkDebugReportObjectTypeEXT {
    VK_DEBUG_REPORT_OBJECT_TYPE_UNKNOWN_EXT = 0,
    VK_DEBUG_REPORT_OBJECT_TYPE_INSTANCE_EXT = 1,
    VK_DEBUG_REPORT_OBJECT_TYPE_PHYSICAL_DEVICE_EXT = 2,
```

```
    VK_DEBUG_REPORT_OBJECT_TYPE_DEVICE_EXT = 3,
    VK_DEBUG_REPORT_OBJECT_TYPE_QUEUE_EXT = 4,
    VK_DEBUG_REPORT_OBJECT_TYPE_SEMAPHORE_EXT = 5,
    VK_DEBUG_REPORT_OBJECT_TYPE_COMMAND_BUFFER_EXT = 6,
    VK_DEBUG_REPORT_OBJECT_TYPE_FENCE_EXT = 7,
    VK_DEBUG_REPORT_OBJECT_TYPE_DEVICE_MEMORY_EXT = 8,
    VK_DEBUG_REPORT_OBJECT_TYPE_BUFFER_EXT = 9,
    VK_DEBUG_REPORT_OBJECT_TYPE_IMAGE_EXT = 10,
    VK_DEBUG_REPORT_OBJECT_TYPE_EVENT_EXT = 11,
    VK_DEBUG_REPORT_OBJECT_TYPE_QUERY_POOL_EXT = 12,
    VK_DEBUG_REPORT_OBJECT_TYPE_BUFFER_VIEW_EXT = 13,
    VK_DEBUG_REPORT_OBJECT_TYPE_IMAGE_VIEW_EXT = 14,
    VK_DEBUG_REPORT_OBJECT_TYPE_SHADER_MODULE_EXT = 15,
    VK_DEBUG_REPORT_OBJECT_TYPE_PIPELINE_CACHE_EXT = 16,
    VK_DEBUG_REPORT_OBJECT_TYPE_PIPELINE_LAYOUT_EXT = 17,
    VK_DEBUG_REPORT_OBJECT_TYPE_RENDER_PASS_EXT = 18,
    VK_DEBUG_REPORT_OBJECT_TYPE_PIPELINE_EXT = 19,
    VK_DEBUG_REPORT_OBJECT_TYPE_DESCRIPTOR_SET_LAYOUT_EXT = 20,
    VK_DEBUG_REPORT_OBJECT_TYPE_SAMPLER_EXT = 21,
    VK_DEBUG_REPORT_OBJECT_TYPE_DESCRIPTOR_POOL_EXT = 22,
    VK_DEBUG_REPORT_OBJECT_TYPE_DESCRIPTOR_SET_EXT = 23,
    VK_DEBUG_REPORT_OBJECT_TYPE_FRAMEBUFFER_EXT = 24,
    VK_DEBUG_REPORT_OBJECT_TYPE_COMMAND_POOL_EXT = 25,
    VK_DEBUG_REPORT_OBJECT_TYPE_SURFACE_KHR_EXT = 26,
    VK_DEBUG_REPORT_OBJECT_TYPE_SWAPCHAIN_KHR_EXT = 27,
    VK_DEBUG_REPORT_OBJECT_TYPE_DEBUG_REPORT_CALLBACK_EXT_EXT = 28,
    VK_DEBUG_REPORT_OBJECT_TYPE_DISPLAY_KHR_EXT = 29,
    VK_DEBUG_REPORT_OBJECT_TYPE_DISPLAY_MODE_KHR_EXT = 30,
    VK_DEBUG_REPORT_OBJECT_TYPE_VALIDATION_CACHE_EXT_EXT = 33,
  // Provided by VK_VERSION_1_1 with VK_EXT_debug_report,
VK_KHR_sampler_ycbcr_conversion with VK_EXT_debug_report
    VK_DEBUG_REPORT_OBJECT_TYPE_SAMPLER_YCBCR_CONVERSION_EXT = 1000156000,
  // Provided by VK_VERSION_1_1 with VK_EXT_debug_report
    VK_DEBUG_REPORT_OBJECT_TYPE_DESCRIPTOR_UPDATE_TEMPLATE_EXT = 1000085000,
  // Provided by VK_EXT_debug_report with VK_NVX_binary_import
    VK_DEBUG_REPORT_OBJECT_TYPE_CU_MODULE_NVX_EXT = 1000029000,
  // Provided by VK_EXT_debug_report with VK_NVX_binary_import
    VK_DEBUG_REPORT_OBJECT_TYPE_CU_FUNCTION_NVX_EXT = 1000029001,
  // Provided by VK_KHR_acceleration_structure with VK_EXT_debug_report
    VK_DEBUG_REPORT_OBJECT_TYPE_ACCELERATION_STRUCTURE_KHR_EXT = 1000150000,
  // Provided by VK_EXT_debug_report with VK_NV_ray_tracing
    VK_DEBUG_REPORT_OBJECT_TYPE_ACCELERATION_STRUCTURE_NV_EXT = 1000165000,
  // Provided by VK_EXT_debug_report with VK_NV_cuda_kernel_launch
    VK_DEBUG_REPORT_OBJECT_TYPE_CUDA_MODULE_NV_EXT = 1000307000,
  // Provided by VK_EXT_debug_report with VK_NV_cuda_kernel_launch
    VK_DEBUG_REPORT_OBJECT_TYPE_CUDA_FUNCTION_NV_EXT = 1000307001,
  // Provided by VK_EXT_debug_report with VK_FUCHSIA_buffer_collection
    VK_DEBUG_REPORT_OBJECT_TYPE_BUFFER_COLLECTION_FUCHSIA_EXT = 1000366000,
  // VK_DEBUG_REPORT_OBJECT_TYPE_DEBUG_REPORT_EXT is a deprecated alias
    VK_DEBUG_REPORT_OBJECT_TYPE_DEBUG_REPORT_EXT =
VK_DEBUG_REPORT_OBJECT_TYPE_DEBUG_REPORT_CALLBACK_EXT_EXT,
```

```
  // VK_DEBUG_REPORT_OBJECT_TYPE_VALIDATION_CACHE_EXT is a deprecated alias
    VK_DEBUG_REPORT_OBJECT_TYPE_VALIDATION_CACHE_EXT =
VK_DEBUG_REPORT_OBJECT_TYPE_VALIDATION_CACHE_EXT_EXT,
  // Provided by VK_KHR_descriptor_update_template with VK_EXT_debug_report
    VK_DEBUG_REPORT_OBJECT_TYPE_DESCRIPTOR_UPDATE_TEMPLATE_KHR_EXT =
VK_DEBUG_REPORT_OBJECT_TYPE_DESCRIPTOR_UPDATE_TEMPLATE_EXT,
  // Provided by VK_KHR_sampler_ycbcr_conversion with VK_EXT_debug_report
    VK_DEBUG_REPORT_OBJECT_TYPE_SAMPLER_YCBCR_CONVERSION_KHR_EXT =
VK_DEBUG_REPORT_OBJECT_TYPE_SAMPLER_YCBCR_CONVERSION_EXT,
} VkDebugReportObjectTypeEXT;
```

*Table 95.* `VkDebugReportObjectTypeEXT` *and Vulkan Handle Relationship*

| VkDebugReportObjectTypeEXT | Vulkan Handle Type |
|---|---|
| VK_DEBUG_REPORT_OBJECT_TYPE_UNKNOWN_EXT | Unknown/Undefined Handle |
| VK_DEBUG_REPORT_OBJECT_TYPE_INSTANCE_EXT | VkInstance |
| VK_DEBUG_REPORT_OBJECT_TYPE_PHYSICAL_DEVICE_EXT | VkPhysicalDevice |
| VK_DEBUG_REPORT_OBJECT_TYPE_DEVICE_EXT | VkDevice |
| VK_DEBUG_REPORT_OBJECT_TYPE_QUEUE_EXT | VkQueue |
| VK_DEBUG_REPORT_OBJECT_TYPE_SEMAPHORE_EXT | VkSemaphore |
| VK_DEBUG_REPORT_OBJECT_TYPE_COMMAND_BUFFER_EXT | VkCommandBuffer |
| VK_DEBUG_REPORT_OBJECT_TYPE_FENCE_EXT | VkFence |
| VK_DEBUG_REPORT_OBJECT_TYPE_DEVICE_MEMORY_EXT | VkDeviceMemory |
| VK_DEBUG_REPORT_OBJECT_TYPE_BUFFER_EXT | VkBuffer |
| VK_DEBUG_REPORT_OBJECT_TYPE_IMAGE_EXT | VkImage |
| VK_DEBUG_REPORT_OBJECT_TYPE_EVENT_EXT | VkEvent |
| VK_DEBUG_REPORT_OBJECT_TYPE_QUERY_POOL_EXT | VkQueryPool |
| VK_DEBUG_REPORT_OBJECT_TYPE_BUFFER_VIEW_EXT | VkBufferView |
| VK_DEBUG_REPORT_OBJECT_TYPE_IMAGE_VIEW_EXT | VkImageView |
| VK_DEBUG_REPORT_OBJECT_TYPE_SHADER_MODULE_EXT | VkShaderModule |
| VK_DEBUG_REPORT_OBJECT_TYPE_PIPELINE_CACHE_EXT | VkPipelineCache |
| VK_DEBUG_REPORT_OBJECT_TYPE_PIPELINE_LAYOUT_EXT | VkPipelineLayout |
| VK_DEBUG_REPORT_OBJECT_TYPE_RENDER_PASS_EXT | VkRenderPass |
| VK_DEBUG_REPORT_OBJECT_TYPE_PIPELINE_EXT | VkPipeline |
| VK_DEBUG_REPORT_OBJECT_TYPE_DESCRIPTOR_SET_LAYOUT_EXT | VkDescriptorSetLayout |

| VkDebugReportObjectTypeEXT | Vulkan Handle Type |
|---|---|
| VK_DEBUG_REPORT_OBJECT_TYPE_SAMPLER_EXT | VkSampler |
| VK_DEBUG_REPORT_OBJECT_TYPE_DESCRIPTOR_POOL_EXT | VkDescriptorPool |
| VK_DEBUG_REPORT_OBJECT_TYPE_DESCRIPTOR_SET_EXT | VkDescriptorSet |
| VK_DEBUG_REPORT_OBJECT_TYPE_FRAMEBUFFER_EXT | VkFramebuffer |
| VK_DEBUG_REPORT_OBJECT_TYPE_COMMAND_POOL_EXT | VkCommandPool |
| VK_DEBUG_REPORT_OBJECT_TYPE_SURFACE_KHR_EXT | VkSurfaceKHR |
| VK_DEBUG_REPORT_OBJECT_TYPE_SWAPCHAIN_KHR_EXT | VkSwapchainKHR |
| VK_DEBUG_REPORT_OBJECT_TYPE_DEBUG_REPORT_CALLBACK_EXT_EXT | VkDebugReportCallbackEXT |
| VK_DEBUG_REPORT_OBJECT_TYPE_DISPLAY_KHR_EXT | VkDisplayKHR |
| VK_DEBUG_REPORT_OBJECT_TYPE_DISPLAY_MODE_KHR_EXT | VkDisplayModeKHR |
| VK_DEBUG_REPORT_OBJECT_TYPE_DESCRIPTOR_UPDATE_TEMPLATE_EXT | VkDescriptorUpdateTemplate |

**NOTE** The primary expected use of VK_ERROR_VALIDATION_FAILED_EXT is for validation layer testing. It is not expected that an application would see this error code during normal use of the validation layers.

To inject its own messages into the debug stream, call:

```
// Provided by VK_EXT_debug_report
void vkDebugReportMessageEXT(
    VkInstance                                  instance,
    VkDebugReportFlagsEXT                       flags,
    VkDebugReportObjectTypeEXT                  objectType,
    uint64_t                                    object,
    size_t                                      location,
    int32_t                                     messageCode,
    const char*                                 pLayerPrefix,
    const char*                                 pMessage);
```

- instance is the debug stream's VkInstance.
- flags specifies the VkDebugReportFlagBitsEXT classification of this event/message.
- objectType is a VkDebugReportObjectTypeEXT specifying the type of object being used or created at the time the event was triggered.
- object is the object where the issue was detected. object **can** be VK_NULL_HANDLE if there is no object associated with the event.
- location is an application-defined value.

- `messageCode` is an application-defined value.
- `pLayerPrefix` is the abbreviation of the component making this event/message.
- `pMessage` is a null-terminated UTF-8 string detailing the trigger conditions.

The call will propagate through the layers and generate callback(s) as indicated by the message's flags. The parameters are passed on to the callback in addition to the `pUserData` value that was defined at the time the callback was registered.

### Valid Usage

- VUID-vkDebugReportMessageEXT-object-01241
  `object` **must** be a Vulkan object or VK_NULL_HANDLE
- VUID-vkDebugReportMessageEXT-objectType-01498
  If `objectType` is not `VK_DEBUG_REPORT_OBJECT_TYPE_UNKNOWN_EXT` and `object` is not VK_NULL_HANDLE, `object` **must** be a Vulkan object of the corresponding type associated with `objectType` as defined in `VkDebugReportObjectTypeEXT` and Vulkan Handle Relationship

### Valid Usage (Implicit)

- VUID-vkDebugReportMessageEXT-instance-parameter
  `instance` **must** be a valid VkInstance handle
- VUID-vkDebugReportMessageEXT-flags-parameter
  `flags` **must** be a valid combination of VkDebugReportFlagBitsEXT values
- VUID-vkDebugReportMessageEXT-flags-requiredbitmask
  `flags` **must** not be `0`
- VUID-vkDebugReportMessageEXT-objectType-parameter
  `objectType` **must** be a valid VkDebugReportObjectTypeEXT value
- VUID-vkDebugReportMessageEXT-pLayerPrefix-parameter
  `pLayerPrefix` **must** be a null-terminated UTF-8 string
- VUID-vkDebugReportMessageEXT-pMessage-parameter
  `pMessage` **must** be a null-terminated UTF-8 string

To destroy a `VkDebugReportCallbackEXT` object, call:

```
// Provided by VK_EXT_debug_report
void vkDestroyDebugReportCallbackEXT(
    VkInstance                                  instance,
    VkDebugReportCallbackEXT                    callback,
    const VkAllocationCallbacks*                pAllocator);
```

- `instance` is the instance where the callback was created.
- `callback` is the VkDebugReportCallbackEXT object to destroy. `callback` is an externally

synchronized object and **must** not be used on more than one thread at a time. This means that `vkDestroyDebugReportCallbackEXT` **must** not be called when a callback is active.

- `pAllocator` controls host memory allocation as described in the Memory Allocation chapter.

### Valid Usage

- VUID-vkDestroyDebugReportCallbackEXT-instance-01242
  If `VkAllocationCallbacks` were provided when `callback` was created, a compatible set of callbacks **must** be provided here
- VUID-vkDestroyDebugReportCallbackEXT-instance-01243
  If no `VkAllocationCallbacks` were provided when `callback` was created, `pAllocator` **must** be `NULL`

### Valid Usage (Implicit)

- VUID-vkDestroyDebugReportCallbackEXT-instance-parameter
  `instance` **must** be a valid VkInstance handle
- VUID-vkDestroyDebugReportCallbackEXT-callback-parameter
  If `callback` is not VK_NULL_HANDLE, `callback` **must** be a valid VkDebugReportCallbackEXT handle
- VUID-vkDestroyDebugReportCallbackEXT-pAllocator-parameter
  If `pAllocator` is not `NULL`, `pAllocator` **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkDestroyDebugReportCallbackEXT-callback-parent
  If `callback` is a valid handle, it **must** have been created, allocated, or retrieved from `instance`

### Host Synchronization

- Host access to `callback` **must** be externally synchronized

# 50.4. Device Loss Debugging

## 50.4.1. Device Diagnostic Checkpoints

Device execution progress **can** be tracked for the purposes of debugging a device loss by annotating the command stream with application-defined diagnostic checkpoints.

Device diagnostic checkpoints are inserted into the command stream by calling vkCmdSetCheckpointNV.

```
// Provided by VK_NV_device_diagnostic_checkpoints
void vkCmdSetCheckpointNV(
    VkCommandBuffer                             commandBuffer,
    const void*                                 pCheckpointMarker);
```

- `commandBuffer` is the command buffer that will receive the marker
- `pCheckpointMarker` is an opaque application-provided value that will be associated with the checkpoint.

## Valid Usage (Implicit)

- VUID-vkCmdSetCheckpointNV-commandBuffer-parameter
  `commandBuffer` **must** be a valid VkCommandBuffer handle
- VUID-vkCmdSetCheckpointNV-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdSetCheckpointNV-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics, compute, or transfer operations
- VUID-vkCmdSetCheckpointNV-videocoding
  This command **must** only be called outside of a video coding scope

## Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics<br>Compute<br>Transfer | Action |

Note that `pCheckpointMarker` is treated as an opaque value. It does not need to be a valid pointer and will not be dereferenced by the implementation.

If the device encounters an error during execution, the implementation will return a `VK_ERROR_DEVICE_LOST` error to the application at some point during host execution. When this happens, the application **can** call vkGetQueueCheckpointData2NV to retrieve information on the

most recent diagnostic checkpoints that were executed by the device.

```
// Provided by VK_KHR_synchronization2 with VK_NV_device_diagnostic_checkpoints
void vkGetQueueCheckpointData2NV(
    VkQueue                                     queue,
    uint32_t*                                   pCheckpointDataCount,
    VkCheckpointData2NV*                        pCheckpointData);
```

- `queue` is the VkQueue object the caller would like to retrieve checkpoint data for
- `pCheckpointDataCount` is a pointer to an integer related to the number of checkpoint markers available or queried, as described below.
- `pCheckpointData` is either `NULL` or a pointer to an array of `VkCheckpointData2NV` structures.

If `pCheckpointData` is `NULL`, then the number of checkpoint markers available is returned in `pCheckpointDataCount`. Otherwise, `pCheckpointDataCount` **must** point to a variable set by the application to the number of elements in the `pCheckpointData` array, and on return the variable is overwritten with the number of structures actually written to `pCheckpointData`.

If `pCheckpointDataCount` is less than the number of checkpoint markers available, at most `pCheckpointDataCount` structures will be written.

### Valid Usage

- VUID-vkGetQueueCheckpointData2NV-queue-03892
  The device that `queue` belongs to **must** be in the lost state

### Valid Usage (Implicit)

- VUID-vkGetQueueCheckpointData2NV-queue-parameter
  `queue` **must** be a valid VkQueue handle
- VUID-vkGetQueueCheckpointData2NV-pCheckpointDataCount-parameter
  `pCheckpointDataCount` **must** be a valid pointer to a `uint32_t` value
- VUID-vkGetQueueCheckpointData2NV-pCheckpointData-parameter
  If the value referenced by `pCheckpointDataCount` is not `0`, and `pCheckpointData` is not `NULL`, `pCheckpointData` **must** be a valid pointer to an array of `pCheckpointDataCount` VkCheckpointData2NV structures

The VkCheckpointData2NV structure is defined as:

```
// Provided by VK_KHR_synchronization2 with VK_NV_device_diagnostic_checkpoints
typedef struct VkCheckpointData2NV {
    VkStructureType          sType;
    void*                    pNext;
    VkPipelineStageFlags2    stage;
    void*                    pCheckpointMarker;
} VkCheckpointData2NV;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `stage` indicates a single pipeline stage which the checkpoint marker data refers to.
- `pCheckpointMarker` contains the value of the last checkpoint marker executed in the stage that `stage` refers to.

### Valid Usage (Implicit)

- VUID-VkCheckpointData2NV-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_CHECKPOINT_DATA_2_NV`
- VUID-VkCheckpointData2NV-pNext-pNext
  `pNext` **must** be `NULL`

The stages at which a checkpoint marker **can** be executed are implementation-defined and **can** be queried by calling vkGetPhysicalDeviceQueueFamilyProperties2.

If the device encounters an error during execution, the implementation will return a `VK_ERROR_DEVICE_LOST` error to the application at a certain point during host execution. When this happens, the application **can** call vkGetQueueCheckpointDataNV to retrieve information on the most recent diagnostic checkpoints that were executed by the device.

```
// Provided by VK_NV_device_diagnostic_checkpoints
void vkGetQueueCheckpointDataNV(
    VkQueue                                     queue,
    uint32_t*                                   pCheckpointDataCount,
    VkCheckpointDataNV*                         pCheckpointData);
```

- `queue` is the VkQueue object the caller would like to retrieve checkpoint data for
- `pCheckpointDataCount` is a pointer to an integer related to the number of checkpoint markers available or queried, as described below.
- `pCheckpointData` is either `NULL` or a pointer to an array of `VkCheckpointDataNV` structures.

If `pCheckpointData` is `NULL`, then the number of checkpoint markers available is returned in `pCheckpointDataCount`.

Otherwise, `pCheckpointDataCount` **must** point to a variable set by the application to the number of

elements in the pCheckpointData array, and on return the variable is overwritten with the number of structures actually written to pCheckpointData.

If pCheckpointDataCount is less than the number of checkpoint markers available, at most pCheckpointDataCount structures will be written.

### Valid Usage

- VUID-vkGetQueueCheckpointDataNV-queue-02025
  The device that queue belongs to **must** be in the lost state

### Valid Usage (Implicit)

- VUID-vkGetQueueCheckpointDataNV-queue-parameter
  queue **must** be a valid VkQueue handle
- VUID-vkGetQueueCheckpointDataNV-pCheckpointDataCount-parameter
  pCheckpointDataCount **must** be a valid pointer to a uint32_t value
- VUID-vkGetQueueCheckpointDataNV-pCheckpointData-parameter
  If the value referenced by pCheckpointDataCount is not 0, and pCheckpointData is not NULL, pCheckpointData **must** be a valid pointer to an array of pCheckpointDataCount VkCheckpointDataNV structures

The VkCheckpointDataNV structure is defined as:

```
// Provided by VK_NV_device_diagnostic_checkpoints
typedef struct VkCheckpointDataNV {
    VkStructureType            sType;
    void*                      pNext;
    VkPipelineStageFlagBits    stage;
    void*                      pCheckpointMarker;
} VkCheckpointDataNV;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- stage is a VkPipelineStageFlagBits value specifying which pipeline stage the checkpoint marker data refers to.
- pCheckpointMarker contains the value of the last checkpoint marker executed in the stage that stage refers to.

The stages at which a checkpoint marker **can** be executed are implementation-defined and **can** be queried by calling vkGetPhysicalDeviceQueueFamilyProperties2.

### Valid Usage (Implicit)

- VUID-VkCheckpointDataNV-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_CHECKPOINT_DATA_NV`
- VUID-VkCheckpointDataNV-pNext-pNext
  `pNext` **must** be `NULL`

## 50.4.2. Device Fault Diagnosis

To retrieve diagnostic information about faults that **may** have caused device loss, call:

```
// Provided by VK_EXT_device_fault
VkResult vkGetDeviceFaultInfoEXT(
    VkDevice                                    device,
    VkDeviceFaultCountsEXT*                     pFaultCounts,
    VkDeviceFaultInfoEXT*                       pFaultInfo);
```

- `device` is the logical device from which to query the diagnostic fault information.
- `pFaultCounts` is a pointer to a VkDeviceFaultCountsEXT structure in which counts for structures describing additional fault information are returned.
- `pFaultInfo` is `NULL` or a pointer to a VkDeviceFaultInfoEXT structure in which fault information is returned.

If `pFaultInfo` is `NULL`, then the counts of corresponding additional fault information structures available are returned in the `addressInfoCount` and `vendorInfoCount` members of `pFaultCounts`. Additionally, the size of any vendor-specific binary crash dump is returned in the `vendorBinarySize` member of `pFaultCounts`.

If `pFaultInfo` is not `NULL`, `pFaultCounts` **must** point to a VkDeviceFaultCountsEXT structure with each structure count or size member (`addressInfoCount`, `vendorInfoCount`, `vendorBinarySize`) set by the application to the number of elements in the corresponding output array member of `pFaultInfo` (`pAddressInfos` and `pVendorInfos`), or to the size of the output buffer in bytes (`pVendorBinaryData`). On return, each structure count member is overwritten with the number of structures actually written to the corresponding output array member of `pFaultInfo`. Similarly, `vendorBinarySize` is overwritten with the number of bytes actually written to the `pVendorBinaryData` member of `pFaultInfo`.

If the vendor-specific crash dumps feature is not enabled, then implementations **must** set `pFaultCounts`->vendorBinarySize to zero and **must** not modify `pFaultInfo`->pVendorBinaryData.

If any `pFaultCounts` structure count member is less than the number of corresponding fault properties available, at most structure count (`addressInfoCount`, `vendorInfoCount`) elements will be written to the associated `pFaultInfo` output array. Similarly, if `vendorBinarySize` is less than the size in bytes of the available crash dump data, at most `vendorBinarySize` elements will be written to `pVendorBinaryData`.

If `pFaultInfo` is `NULL`, then subsequent calls to vkGetDeviceFaultInfoEXT for the same `device` **must**

return identical values in the `addressInfoCount`, `vendorInfoCount` and `vendorBinarySize` members of `pFaultCounts`.

If `pFaultInfo` is not `NULL`, then subsequent calls to vkGetDeviceFaultInfoEXT for the same `device` **must** return identical values in the output members of `pFaultInfo` (`pAddressInfos`, `pVendorInfos`, `pVendorBinaryData`), up to the limits described by the structure count and buffer size members of `pFaultCounts` (`addressInfoCount`, `vendorInfoCount`, `vendorBinarySize`). If the sizes of the output members of `pFaultInfo` increase for a subsequent call to vkGetDeviceFaultInfoEXT, then supplementary information **may** be returned in the additional available space.

If any `pFaultCounts` structure count member is smaller than the number of corresponding fault properties available, or if `pFaultCounts`->vendorBinarySize is smaller than the size in bytes of the generated binary crash dump data, `VK_INCOMPLETE` will be returned instead of `VK_SUCCESS`, to indicate that not all the available properties were returned.

If `pFaultCounts`->vendorBinarySize is less than what is necessary to store the binary crash dump header, nothing will be written to `pFaultInfo`->pVendorBinaryData and zero will be written to `pFaultCounts`->vendorBinarySize.

## Valid Usage

- VUID-vkGetDeviceFaultInfoEXT-device-07336
  `device` **must** be in the *lost* state
- VUID-vkGetDeviceFaultInfoEXT-pFaultCounts-07337
  If the value referenced by `pFaultCounts->addressInfoCount` is not `0`, and `pFaultInfo->pAddressInfos` is not `NULL`, `pFaultInfo->pAddressInfos` **must** be a valid pointer to an array of `pFaultCounts->addressInfoCount` VkDeviceFaultAddressInfoEXT structures
- VUID-vkGetDeviceFaultInfoEXT-pFaultCounts-07338
  If the value referenced by `pFaultCounts->vendorInfoCount` is not `0`, and `pFaultInfo->pVendorInfos` is not `NULL`, `pFaultInfo->pVendorInfos` **must** be a valid pointer to an array of `pFaultCounts->vendorInfoCount` VkDeviceFaultVendorInfoEXT structures
- VUID-vkGetDeviceFaultInfoEXT-pFaultCounts-07339
  If the value referenced by `pFaultCounts->vendorBinarySize` is not `0`, and `pFaultInfo->pVendorBinaryData` is not `NULL`, `pFaultInfo->pVendorBinaryData` **must** be a valid pointer to an array of `pFaultCounts->vendorBinarySize` bytes

## Valid Usage (Implicit)

- VUID-vkGetDeviceFaultInfoEXT-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkGetDeviceFaultInfoEXT-pFaultCounts-parameter
  `pFaultCounts` **must** be a valid pointer to a VkDeviceFaultCountsEXT structure
- VUID-vkGetDeviceFaultInfoEXT-pFaultInfo-parameter
  If `pFaultInfo` is not `NULL`, `pFaultInfo` **must** be a valid pointer to a VkDeviceFaultInfoEXT structure

### Return Codes

**Success**

- VK_SUCCESS
- VK_INCOMPLETE

**Failure**

- VK_ERROR_OUT_OF_HOST_MEMORY

The VkDeviceFaultCountsEXT structure is defined as:

```
// Provided by VK_EXT_device_fault
typedef struct VkDeviceFaultCountsEXT {
    VkStructureType    sType;
    void*              pNext;
    uint32_t           addressInfoCount;
    uint32_t           vendorInfoCount;
    VkDeviceSize       vendorBinarySize;
} VkDeviceFaultCountsEXT;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- addressInfoCount is the number of VkDeviceFaultAddressInfoEXT structures describing either memory accesses which **may** have caused a page fault, or the addresses of active instructions at the time of the fault.
- vendorInfoCount is the number of VkDeviceFaultVendorInfoEXT structures describing vendor-specific fault information.
- vendorBinarySize is the size in bytes of a vendor-specific binary crash dump, which may provide additional information when imported into external tools.

### Valid Usage (Implicit)

- VUID-VkDeviceFaultCountsEXT-sType-sType
  sType **must** be VK_STRUCTURE_TYPE_DEVICE_FAULT_COUNTS_EXT
- VUID-VkDeviceFaultCountsEXT-pNext-pNext
  pNext **must** be NULL

The VkDeviceFaultInfoEXT structure is defined as:

```
// Provided by VK_EXT_device_fault
typedef struct VkDeviceFaultInfoEXT {
    VkStructureType                 sType;
    void*                           pNext;
    char                            description[VK_MAX_DESCRIPTION_SIZE];
    VkDeviceFaultAddressInfoEXT*    pAddressInfos;
    VkDeviceFaultVendorInfoEXT*     pVendorInfos;
    void*                           pVendorBinaryData;
} VkDeviceFaultInfoEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `description` is an array of `VK_MAX_DESCRIPTION_SIZE` `char` containing a null-terminated UTF-8 string which is a human readable description of the fault.
- `pAddressInfos` is `NULL` or a pointer to an array of VkDeviceFaultAddressInfoEXT structures describing either memory accesses which **may** have caused a page fault, or describing active instruction pointers at the time of the fault. If not `NULL`, each element of `pAddressInfos` describes the a bounded region of GPU virtual address space containing either the GPU virtual address accessed, or the value of an active instruction pointer.
- `pVendorInfos` is `NULL` or a pointer to an array of VkDeviceFaultVendorInfoEXT structures describing vendor-specific fault information.
- `pVendorBinaryData` is `NULL` or a pointer to `vendorBinarySize` number of bytes of data, which will be populated with a vendor-specific binary crash dump, as described in Vendor Binary Crash Dumps.

An implementation **should** populate as many members of VkDeviceFaultInfoEXT as possible, given the information available at the time of the fault and the constraints of the implementation itself.

Due to hardware limitations, `pAddressInfos` describes ranges of GPU virtual address space, rather than precise addresses. The precise memory address accessed or the precise value of the instruction pointer **must** lie within the region described.

**NOTE**

Each element of `pAddressInfos` describes either:

- A memory access which may have triggered a page fault and may have contributed to device loss
- The value of an active instruction pointer at the time a fault occurred. This value may be indicative of the active pipeline or shader at the time of device loss

Comparison of the GPU virtual addresses described by `pAddressInfos` to GPU virtual address ranges reported by the `VK_EXT_device_address_binding_report` extension may allow applications to correlate between these addresses and Vulkan objects. Applications should be aware that these addresses may also correspond to resources internal to an implementation, which will not be reported via the `VK_EXT_device_address_binding_report` extension.

### Valid Usage (Implicit)

- VUID-VkDeviceFaultInfoEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_DEVICE_FAULT_INFO_EXT`
- VUID-VkDeviceFaultInfoEXT-pNext-pNext
  `pNext` **must** be `NULL`

The `VkDeviceFaultAddressInfoEXT` structure is defined as:

```
// Provided by VK_EXT_device_fault
typedef struct VkDeviceFaultAddressInfoEXT {
    VkDeviceFaultAddressTypeEXT    addressType;
    VkDeviceAddress                reportedAddress;
    VkDeviceSize                   addressPrecision;
} VkDeviceFaultAddressInfoEXT;
```

- `addressType` is either the type of memory operation that triggered a page fault, or the type of association between an instruction pointer and a fault.
- `reportedAddress` is the GPU virtual address recorded by the device.
- `addressPrecision` is a power of two value that specifies how precisely the device can report the address.

The combination of `reportedAddress` and `addressPrecision` allow the possible range of addresses to be calculated, such that:

```
lower_address = (pInfo->reportedAddress & ~(pInfo->addressPrecision-1))
upper_address = (pInfo->reportedAddress |  (pInfo->addressPrecision-1))
```

**NOTE** It is valid for the `reportedAddress` to contain a more precise address than indicated by `addressPrecision`. In this case, the value of `reportedAddress` should be treated as an additional hint as to the value of the address that triggered the page fault, or to the value of an instruction pointer.

### Valid Usage (Implicit)

- VUID-VkDeviceFaultAddressInfoEXT-addressType-parameter
  `addressType` **must** be a valid VkDeviceFaultAddressTypeEXT value

Possible values of VkDeviceFaultAddressInfoEXT::`addressType` are:

```
// Provided by VK_EXT_device_fault
typedef enum VkDeviceFaultAddressTypeEXT {
    VK_DEVICE_FAULT_ADDRESS_TYPE_NONE_EXT = 0,
    VK_DEVICE_FAULT_ADDRESS_TYPE_READ_INVALID_EXT = 1,
    VK_DEVICE_FAULT_ADDRESS_TYPE_WRITE_INVALID_EXT = 2,
    VK_DEVICE_FAULT_ADDRESS_TYPE_EXECUTE_INVALID_EXT = 3,
    VK_DEVICE_FAULT_ADDRESS_TYPE_INSTRUCTION_POINTER_UNKNOWN_EXT = 4,
    VK_DEVICE_FAULT_ADDRESS_TYPE_INSTRUCTION_POINTER_INVALID_EXT = 5,
    VK_DEVICE_FAULT_ADDRESS_TYPE_INSTRUCTION_POINTER_FAULT_EXT = 6,
} VkDeviceFaultAddressTypeEXT;
```

- `VK_DEVICE_FAULT_ADDRESS_TYPE_NONE_EXT` specifies that VkDeviceFaultAddressInfoEXT does not describe a page fault, or an instruction address.
- `VK_DEVICE_FAULT_ADDRESS_TYPE_READ_INVALID_EXT` specifies that VkDeviceFaultAddressInfoEXT describes a page fault triggered by an invalid read operation.
- `VK_DEVICE_FAULT_ADDRESS_TYPE_WRITE_INVALID_EXT` specifies that VkDeviceFaultAddressInfoEXT describes a page fault triggered by an invalid write operation.
- `VK_DEVICE_FAULT_ADDRESS_TYPE_EXECUTE_INVALID_EXT` describes a page fault triggered by an attempt to execute non-executable memory.
- `VK_DEVICE_FAULT_ADDRESS_TYPE_INSTRUCTION_POINTER_UNKNOWN_EXT` specifies an instruction pointer value at the time the fault occurred. This may or may not be related to a fault.
- `VK_DEVICE_FAULT_ADDRESS_TYPE_INSTRUCTION_POINTER_INVALID_EXT` specifies an instruction pointer value associated with an invalid instruction fault.
- `VK_DEVICE_FAULT_ADDRESS_TYPE_INSTRUCTION_POINTER_FAULT_EXT` specifies an instruction pointer value associated with a fault.

**NOTE** The instruction pointer values recorded may not identify the specific instruction(s) that triggered the fault. The relationship between the instruction pointer reported and triggering instruction will be vendor-specific.

The `VkDeviceFaultVendorInfoEXT` structure is defined as:

```
// Provided by VK_EXT_device_fault
typedef struct VkDeviceFaultVendorInfoEXT {
    char        description[VK_MAX_DESCRIPTION_SIZE];
    uint64_t    vendorFaultCode;
    uint64_t    vendorFaultData;
} VkDeviceFaultVendorInfoEXT;
```

- `description` is an array of `VK_MAX_DESCRIPTION_SIZE` `char` containing a null-terminated UTF-8 string which is a human readable description of the fault.
- `vendorFaultCode` is the vendor-specific fault code for this fault.
- `vendorFaultData` is the vendor-specific fault data associated with this fault.

#### Vendor Binary Crash Dumps

Applications **can** store the vendor-specific binary crash dump data retrieved by calling vkGetDeviceFaultInfoEXT for later analysis using external tools.

However, the format of this data **may** depend on the vendor ID, device ID, driver version, and other details of the device. To enable external applications to identify the original device from which a crash dump was generated, the initial bytes written to VkDeviceFaultInfoEXT::pVendorBinaryData **must** begin with a valid crash dump header.

Version one of the crash dump header is defined as:

```
// Provided by VK_EXT_device_fault
typedef struct VkDeviceFaultVendorBinaryHeaderVersionOneEXT {
    uint32_t                                        headerSize;
    VkDeviceFaultVendorBinaryHeaderVersionEXT       headerVersion;
    uint32_t                                        vendorID;
    uint32_t                                        deviceID;
    uint32_t                                        driverVersion;
    uint8_t                                         pipelineCacheUUID[VK_UUID_SIZE];
    uint32_t                                        applicationNameOffset;
    uint32_t                                        applicationVersion;
    uint32_t                                        engineNameOffset;
    uint32_t                                        engineVersion;
    uint32_t                                        apiVersion;
} VkDeviceFaultVendorBinaryHeaderVersionOneEXT;
```

- headerSize is the length in bytes of the crash dump header.
- headerVersion is a VkDeviceFaultVendorBinaryHeaderVersionEXT enum value specifying the version of the header. A consumer of the crash dump **should** use the header version to interpret the remainder of the header.
- vendorID is the VkPhysicalDeviceProperties::vendorID of the implementation.
- deviceID is the VkPhysicalDeviceProperties::deviceID of the implementation.
- driverVersion is the VkPhysicalDeviceProperties::driverVersion of the implementation.
- pipelineCacheUUID is an array of VK_UUID_SIZE uint8_t values matching the VkPhysicalDeviceProperties::pipelineCacheUUID property of the implementation.
- applicationNameOffset is zero, or an offset from the base address of the crash dump header to a null-terminated UTF-8 string containing the name of the application. If applicationNameOffset is non-zero, this string **must** match the application name specified via VkApplicationInfo::pApplicationName during instance creation.
- applicationVersion **must** be zero or the value specified by VkApplicationInfo::applicationVersion during instance creation.
- engineNameOffset is zero, or an offset from the base address of the crash dump header to a null-terminated UTF-8 string containing the name of the engine (if any) used to create the application. If engineNameOffset is non-zero, this string **must** match the engine name specified

via VkApplicationInfo::pEngineName during instance creation.

- engineVersion **must** be zero or the value specified by VkApplicationInfo::engineVersion during instance creation.
- apiVersion **must** be zero or the value specified by VkApplicationInfo::apiVersion during instance creation.

Unlike most structures declared by the Vulkan API, all fields of this structure are written with the least significant byte first, regardless of host byte-order.

The C language specification does not define the packing of structure members. This layout assumes tight structure member packing, with members laid out in the order listed in the structure, and the intended size of the structure is 56 bytes. If a compiler produces code that diverges from that pattern, applications **must** employ another method to set values at the correct offsets.

### Valid Usage

- VUID-VkDeviceFaultVendorBinaryHeaderVersionOneEXT-headerSize-07340
  headerSize **must** be 56
- VUID-VkDeviceFaultVendorBinaryHeaderVersionOneEXT-headerVersion-07341
  headerVersion **must** be VK_DEVICE_FAULT_VENDOR_BINARY_HEADER_VERSION_ONE_EXT

### Valid Usage (Implicit)

- VUID-VkDeviceFaultVendorBinaryHeaderVersionOneEXT-headerVersion-parameter
  headerVersion **must** be a valid VkDeviceFaultVendorBinaryHeaderVersionEXT value

Possible values of the headerVersion value of the crash dump header are:

```
// Provided by VK_EXT_device_fault
typedef enum VkDeviceFaultVendorBinaryHeaderVersionEXT {
    VK_DEVICE_FAULT_VENDOR_BINARY_HEADER_VERSION_ONE_EXT = 1,
} VkDeviceFaultVendorBinaryHeaderVersionEXT;
```

- VK_DEVICE_FAULT_VENDOR_BINARY_HEADER_VERSION_ONE_EXT specifies version one of the binary crash dump header.

## 50.5. Active Tooling Information

Information about tools providing debugging, profiling, or similar services, active for a given physical device, can be obtained by calling:

```
// Provided by VK_EXT_tooling_info
VkResult vkGetPhysicalDeviceToolPropertiesEXT(
    VkPhysicalDevice                            physicalDevice,
    uint32_t*                                   pToolCount,
    VkPhysicalDeviceToolProperties*             pToolProperties);
```

- `physicalDevice` is the handle to the physical device to query for active tools.
- `pToolCount` is a pointer to an integer describing the number of tools active on `physicalDevice`.
- `pToolProperties` is either `NULL` or a pointer to an array of VkPhysicalDeviceToolProperties structures.

If `pToolProperties` is `NULL`, then the number of tools currently active on `physicalDevice` is returned in `pToolCount`. Otherwise, `pToolCount` **must** point to a variable set by the application to the number of elements in the `pToolProperties` array, and on return the variable is overwritten with the number of structures actually written to `pToolProperties`. If `pToolCount` is less than the number of currently active tools, at most `pToolCount` structures will be written.

The count and properties of active tools **may** change in response to events outside the scope of the specification. An application **should** assume these properties might change at any given time.

### Valid Usage (Implicit)

- VUID-vkGetPhysicalDeviceToolProperties-physicalDevice-parameter
  `physicalDevice` **must** be a valid VkPhysicalDevice handle
- VUID-vkGetPhysicalDeviceToolProperties-pToolCount-parameter
  `pToolCount` **must** be a valid pointer to a `uint32_t` value
- VUID-vkGetPhysicalDeviceToolProperties-pToolProperties-parameter
  If the value referenced by `pToolCount` is not `0`, and `pToolProperties` is not `NULL`, `pToolProperties` **must** be a valid pointer to an array of `pToolCount` VkPhysicalDeviceToolProperties structures

### Return Codes

**Success**

- `VK_SUCCESS`
- `VK_INCOMPLETE`

**Failure**

- `VK_ERROR_OUT_OF_HOST_MEMORY`

The VkPhysicalDeviceToolProperties structure is defined as:

```
typedef struct VkPhysicalDeviceToolProperties {
    VkStructureType       sType;
    void*                 pNext;
    char                  name[VK_MAX_EXTENSION_NAME_SIZE];
    char                  version[VK_MAX_EXTENSION_NAME_SIZE];
    VkToolPurposeFlags    purposes;
    char                  description[VK_MAX_DESCRIPTION_SIZE];
    char                  layer[VK_MAX_EXTENSION_NAME_SIZE];
} VkPhysicalDeviceToolProperties;
```

or the equivalent

```
// Provided by VK_EXT_tooling_info
typedef VkPhysicalDeviceToolProperties VkPhysicalDeviceToolPropertiesEXT;
```

- `sType` is a VkStructureType value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `name` is a null-terminated UTF-8 string containing the name of the tool.
- `version` is a null-terminated UTF-8 string containing the version of the tool.
- `purposes` is a bitmask of VkToolPurposeFlagBits which is populated with purposes supported by the tool.
- `description` is a null-terminated UTF-8 string containing a description of the tool.
- `layer` is a null-terminated UTF-8 string containing the name of the layer implementing the tool, if the tool is implemented in a layer - otherwise it **may** be an empty string.

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceToolProperties-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_TOOL_PROPERTIES`
- VUID-VkPhysicalDeviceToolProperties-pNext-pNext
  `pNext` **must** be `NULL`

Bits which **can** be set in VkPhysicalDeviceToolProperties::`purposes`, specifying the purposes of an active tool, are:

```
typedef enum VkToolPurposeFlagBits {
    VK_TOOL_PURPOSE_VALIDATION_BIT = 0x00000001,
    VK_TOOL_PURPOSE_PROFILING_BIT = 0x00000002,
    VK_TOOL_PURPOSE_TRACING_BIT = 0x00000004,
    VK_TOOL_PURPOSE_ADDITIONAL_FEATURES_BIT = 0x00000008,
    VK_TOOL_PURPOSE_MODIFYING_FEATURES_BIT = 0x00000010,
  // Provided by VK_EXT_debug_report with VK_EXT_tooling_info, VK_EXT_debug_utils with
VK_EXT_tooling_info
    VK_TOOL_PURPOSE_DEBUG_REPORTING_BIT_EXT = 0x00000020,
  // Provided by VK_EXT_debug_marker with VK_EXT_tooling_info, VK_EXT_debug_utils with
VK_EXT_tooling_info
    VK_TOOL_PURPOSE_DEBUG_MARKERS_BIT_EXT = 0x00000040,
    VK_TOOL_PURPOSE_VALIDATION_BIT_EXT = VK_TOOL_PURPOSE_VALIDATION_BIT,
    VK_TOOL_PURPOSE_PROFILING_BIT_EXT = VK_TOOL_PURPOSE_PROFILING_BIT,
    VK_TOOL_PURPOSE_TRACING_BIT_EXT = VK_TOOL_PURPOSE_TRACING_BIT,
    VK_TOOL_PURPOSE_ADDITIONAL_FEATURES_BIT_EXT =
VK_TOOL_PURPOSE_ADDITIONAL_FEATURES_BIT,
    VK_TOOL_PURPOSE_MODIFYING_FEATURES_BIT_EXT =
VK_TOOL_PURPOSE_MODIFYING_FEATURES_BIT,
} VkToolPurposeFlagBits;
```

or the equivalent

```
// Provided by VK_EXT_tooling_info
typedef VkToolPurposeFlagBits VkToolPurposeFlagBitsEXT;
```

- VK_TOOL_PURPOSE_VALIDATION_BIT specifies that the tool provides validation of API usage.
- VK_TOOL_PURPOSE_PROFILING_BIT specifies that the tool provides profiling of API usage.
- VK_TOOL_PURPOSE_TRACING_BIT specifies that the tool is capturing data about the application's API usage, including anything from simple logging to capturing data for later replay.
- VK_TOOL_PURPOSE_ADDITIONAL_FEATURES_BIT specifies that the tool provides additional API features/extensions on top of the underlying implementation.
- VK_TOOL_PURPOSE_MODIFYING_FEATURES_BIT specifies that the tool modifies the API features/limits/extensions presented to the application.
- VK_TOOL_PURPOSE_DEBUG_REPORTING_BIT_EXT specifies that the tool reports additional information to the application via callbacks specified by vkCreateDebugReportCallbackEXT or vkCreateDebugUtilsMessengerEXT
- VK_TOOL_PURPOSE_DEBUG_MARKERS_BIT_EXT specifies that the tool consumes debug markers or object debug annotation, queue labels, or command buffer labels

```
typedef VkFlags VkToolPurposeFlags;
```

or the equivalent

```
// Provided by VK_EXT_tooling_info
typedef VkToolPurposeFlags VkToolPurposeFlagsEXT;
```

VkToolPurposeFlags is a bitmask type for setting a mask of zero or more VkToolPurposeFlagBits.

## 50.6. Frame Boundary

The VkFrameBoundaryEXT structure is defined as:

```
// Provided by VK_EXT_frame_boundary
typedef struct VkFrameBoundaryEXT {
    VkStructureType            sType;
    const void*                pNext;
    VkFrameBoundaryFlagsEXT    flags;
    uint64_t                   frameID;
    uint32_t                   imageCount;
    const VkImage*             pImages;
    uint32_t                   bufferCount;
    const VkBuffer*            pBuffers;
    uint64_t                   tagName;
    size_t                     tagSize;
    const void*                pTag;
} VkFrameBoundaryEXT;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- flags is a bitmask of VkFrameBoundaryFlagBitsEXT that can flag the last submission of a frame identifier.
- frameID is the frame identifier.
- imageCount is the number of images that store frame results.
- pImages is a pointer to an array of VkImage objects with imageCount entries.
- bufferCount is the number of buffers the store the frame results.
- pBuffers is a pointer to an array of VkBuffer objects with bufferCount entries.
- tagName is a numerical identifier for tag data.
- tagSize is the number of bytes of tag data.
- pTag is a pointer to an array of tagSize bytes containing tag data.

The application **can** associate frame boundary information to a queue submission call by adding a VkFrameBoundaryEXT structure to the pNext chain of queue submission, VkPresentInfoKHR, or VkBindSparseInfo.

The frame identifier is used to associate one or more queue submission to a frame, it is thus meant to be unique within a frame lifetime, i.e. it is possible (but not recommended) to reuse frame

identifiers, as long as any two frames with any chance of having overlapping queue submissions (as in the example above) use two different frame identifiers.

| | |
|---|---|
| **NOTE** | Since the concept of frame is application-dependent, there is no way to validate the use of frame identifier. It is good practice to use a monotonically increasing counter as the frame identifier and not reuse identifiers between frames. |

The `pImages` and `pBuffers` arrays contain a list of images and buffers which store the "end result" of the frame. As the concept of frame is application-dependent, not all frames **may** produce their results in images or buffers, yet this is a sufficiently common case to be handled by `VkFrameBoundaryEXT`. Note that no extra information, such as image layout is being provided, since the images are meant to be used by tools which would already be tracking this required information. Having the possibility of passing a list of end-result images makes `VkFrameBoundaryEXT` as expressive as vkQueuePresentKHR, which is often the default frame boundary delimiter.

The application **can** also associate arbitrary extra information via tag data using `tagName`, `tagSize` and `pTag`. This extra information is typically tool-specific.

### Valid Usage (Implicit)

- VUID-VkFrameBoundaryEXT-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_FRAME_BOUNDARY_EXT`
- VUID-VkFrameBoundaryEXT-flags-parameter
  `flags` **must** be a valid combination of VkFrameBoundaryFlagBitsEXT values
- VUID-VkFrameBoundaryEXT-pImages-parameter
  If `imageCount` is not `0`, and `pImages` is not `NULL`, `pImages` **must** be a valid pointer to an array of `imageCount` valid VkImage handles
- VUID-VkFrameBoundaryEXT-pBuffers-parameter
  If `bufferCount` is not `0`, and `pBuffers` is not `NULL`, `pBuffers` **must** be a valid pointer to an array of `bufferCount` valid VkBuffer handles
- VUID-VkFrameBoundaryEXT-pTag-parameter
  If `tagSize` is not `0`, and `pTag` is not `NULL`, `pTag` **must** be a valid pointer to an array of `tagSize` bytes
- VUID-VkFrameBoundaryEXT-commonparent
  Both of the elements of `pBuffers`, and the elements of `pImages` that are valid handles of non-ignored parameters **must** have been created, allocated, or retrieved from the same VkDevice

The bit which **can** be set in VkFrameBoundaryEXT::`flags` is:

```
// Provided by VK_EXT_frame_boundary
typedef enum VkFrameBoundaryFlagBitsEXT {
    VK_FRAME_BOUNDARY_FRAME_END_BIT_EXT = 0x00000001,
} VkFrameBoundaryFlagBitsEXT;
```

- VK_FRAME_BOUNDARY_FRAME_END_BIT_EXT specifies that this queue submission is the last one for this frame, i.e. once this queue submission has terminated, then the work for this frame is completed.

Note that in the presence of timeline semaphores, the last queue submission might not be the last one to be submitted, as timeline semaphores allow for wait-before-signal submissions. In the context of frame boundary, the queue submission that should be done flagged as the last one is the one that is meant to be executed last, even if it **may** not be the last one to be submitted.

```
// Provided by VK_EXT_frame_boundary
typedef VkFlags VkFrameBoundaryFlagsEXT;
```

VkFrameBoundaryFlagsEXT is a bitmask type for setting a mask of zero or more VkFrameBoundaryFlagBitsEXT.

# Appendix A: Vulkan Environment for SPIR-V

Shaders for Vulkan are defined by the Khronos SPIR-V Specification as well as the Khronos SPIR-V Extended Instructions for GLSL Specification. This appendix defines additional SPIR-V requirements applying to Vulkan shaders.

## Versions and Formats

A Vulkan 1.1 implementation **must** support the 1.0, 1.1, 1.2, and 1.3 versions of SPIR-V and the 1.0 version of the SPIR-V Extended Instructions for GLSL. If the `VK_KHR_spirv_1_4` extension is enabled, the implementation **must** additionally support the 1.4 version of SPIR-V.

A SPIR-V module passed into vkCreateShaderModule is interpreted as a series of 32-bit words in host endianness, with literal strings packed as described in section 2.2 of the SPIR-V Specification. The first few words of the SPIR-V module **must** be a magic number and a SPIR-V version number, as described in section 2.3 of the SPIR-V Specification.

## Capabilities

The table below lists the set of SPIR-V capabilities that **may** be supported in Vulkan implementations. The application **must** not use any of these capabilities in SPIR-V passed to vkCreateShaderModule unless one of the following conditions is met for the VkDevice specified in the `device` parameter of vkCreateShaderModule:

- The corresponding field in the table is blank.
- Any corresponding Vulkan feature is enabled.
- Any corresponding Vulkan extension is enabled.
- Any corresponding Vulkan property is supported.
- The corresponding core version is supported (as returned by VkPhysicalDeviceProperties::`apiVersion`).

*Table 96. List of SPIR-V Capabilities and corresponding Vulkan features, extensions, or core version*

| **SPIR-V** `OpCapability` | **Vulkan feature, extension, or core version** |
|---|---|
| `Matrix` | VK_VERSION_1_0 |
| `Shader` | VK_VERSION_1_0 |
| `InputAttachment` | VK_VERSION_1_0 |
| `Sampled1D` | VK_VERSION_1_0 |

| SPIR-V OpCapability | Vulkan feature, extension, or core version |
|---|---|
| Image1D | VK_VERSION_1_0 |
| SampledBuffer | VK_VERSION_1_0 |
| ImageBuffer | VK_VERSION_1_0 |
| ImageQuery | VK_VERSION_1_0 |
| DerivativeControl | VK_VERSION_1_0 |
| Geometry | VkPhysicalDeviceFeatures::geometryShader |
| Tessellation | VkPhysicalDeviceFeatures::tessellationShader |
| Float64 | VkPhysicalDeviceFeatures::shaderFloat64 |
| Int64 | VkPhysicalDeviceFeatures::shaderInt64 |
| Int64Atomics | VkPhysicalDeviceVulkan12Features::shaderBufferInt64Atomics<br>VkPhysicalDeviceVulkan12Features::shaderSharedInt64Atomics<br>VkPhysicalDeviceShaderImageAtomicInt64FeaturesEXT::shaderImageInt64Atomics |
| AtomicFloat16AddEXT | VkPhysicalDeviceShaderAtomicFloat2FeaturesEXT::shaderBufferFloat16AtomicAdd<br>VkPhysicalDeviceShaderAtomicFloat2FeaturesEXT::shaderSharedFloat16AtomicAdd |
| AtomicFloat32AddEXT | VkPhysicalDeviceShaderAtomicFloatFeaturesEXT::shaderBufferFloat32AtomicAdd<br>VkPhysicalDeviceShaderAtomicFloatFeaturesEXT::shaderSharedFloat32AtomicAdd<br>VkPhysicalDeviceShaderAtomicFloatFeaturesEXT::shaderImageFloat32AtomicAdd |
| AtomicFloat64AddEXT | VkPhysicalDeviceShaderAtomicFloatFeaturesEXT::shaderBufferFloat64AtomicAdd<br>VkPhysicalDeviceShaderAtomicFloatFeaturesEXT::shaderSharedFloat64AtomicAdd |
| AtomicFloat16MinMaxEXT | VkPhysicalDeviceShaderAtomicFloat2FeaturesEXT::shaderBufferFloat16AtomicMinMax<br>VkPhysicalDeviceShaderAtomicFloat2FeaturesEXT::shaderSharedFloat16AtomicMinMax |
| AtomicFloat32MinMaxEXT | VkPhysicalDeviceShaderAtomicFloat2FeaturesEXT::shaderBufferFloat32AtomicMinMax<br>VkPhysicalDeviceShaderAtomicFloat2FeaturesEXT::shaderSharedFloat32AtomicMinMax<br>VkPhysicalDeviceShaderAtomicFloat2FeaturesEXT::shaderImageFloat32AtomicMinMax |

| SPIR-V OpCapability | Vulkan feature, extension, or core version |
| --- | --- |
| AtomicFloat64MinMaxEXT | VkPhysicalDeviceShaderAtomicFloat2FeaturesEXT::shaderBufferFloat64AtomicMinMax<br>VkPhysicalDeviceShaderAtomicFloat2FeaturesEXT::shaderSharedFloat64AtomicMinMax |
| AtomicFloat16VectorNV | VkPhysicalDeviceShaderAtomicFloat16VectorFeaturesNV::shaderFloat16VectorAtomics |
| Int64ImageEXT | VkPhysicalDeviceShaderImageAtomicInt64FeaturesEXT::shaderImageInt64Atomics |
| Int16 | VkPhysicalDeviceFeatures::shaderInt16 |
| TessellationPointSize | VkPhysicalDeviceFeatures::shaderTessellationAndGeometryPointSize |
| GeometryPointSize | VkPhysicalDeviceFeatures::shaderTessellationAndGeometryPointSize |
| ImageGatherExtended | VkPhysicalDeviceFeatures::shaderImageGatherExtended |
| StorageImageMultisample | VkPhysicalDeviceFeatures::shaderStorageImageMultisample |
| UniformBufferArrayDynamicIndexing | VkPhysicalDeviceFeatures::shaderUniformBufferArrayDynamicIndexing |
| SampledImageArrayDynamicIndexing | VkPhysicalDeviceFeatures::shaderSampledImageArrayDynamicIndexing |
| StorageBufferArrayDynamicIndexing | VkPhysicalDeviceFeatures::shaderStorageBufferArrayDynamicIndexing |
| StorageImageArrayDynamicIndexing | VkPhysicalDeviceFeatures::shaderStorageImageArrayDynamicIndexing |
| ClipDistance | VkPhysicalDeviceFeatures::shaderClipDistance |
| CullDistance | VkPhysicalDeviceFeatures::shaderCullDistance |
| ImageCubeArray | VkPhysicalDeviceFeatures::imageCubeArray |
| SampleRateShading | VkPhysicalDeviceFeatures::sampleRateShading |
| SparseResidency | VkPhysicalDeviceFeatures::shaderResourceResidency |
| MinLod | VkPhysicalDeviceFeatures::shaderResourceMinLod |

| SPIR-V OpCapability | Vulkan feature, extension, or core version |
| --- | --- |
| SampledCubeArray | VkPhysicalDeviceFeatures::imageCubeArray |
| ImageMSArray | VkPhysicalDeviceFeatures::shaderStorageImageMultisample |
| StorageImageExtendedFormats | VK_VERSION_1_0 |
| InterpolationFunction | VkPhysicalDeviceFeatures::sampleRateShading |
| StorageImageReadWithoutFormat | VkPhysicalDeviceFeatures::shaderStorageImageReadWithoutFormat<br>VK_KHR_format_feature_flags2 |
| StorageImageWriteWithoutFormat | VkPhysicalDeviceFeatures::shaderStorageImageWriteWithoutFormat<br>VK_KHR_format_feature_flags2 |
| MultiViewport | VkPhysicalDeviceFeatures::multiViewport |
| DrawParameters | VkPhysicalDeviceShaderDrawParametersFeatures::shaderDrawParameters<br>VK_KHR_shader_draw_parameters |
| MultiView | VkPhysicalDeviceMultiviewFeatures::multiview |
| DeviceGroup | VK_VERSION_1_1<br>VK_KHR_device_group |
| VariablePointersStorageBuffer | VkPhysicalDeviceVariablePointersFeatures::variablePointersStorageBuffer |
| VariablePointers | VkPhysicalDeviceVariablePointersFeatures::variablePointers |
| ShaderClockKHR | VK_KHR_shader_clock |
| StencilExportEXT | VK_EXT_shader_stencil_export |
| SubgroupBallotKHR | VK_EXT_shader_subgroup_ballot |
| SubgroupVoteKHR | VK_EXT_shader_subgroup_vote |
| ImageReadWriteLodAMD | VK_AMD_shader_image_load_store_lod |

| SPIR-V OpCapability | Vulkan feature, extension, or core version |
| --- | --- |
| ImageGatherBiasLodAMD | VK_AMD_texture_gather_bias_lod |
| FragmentMaskAMD | VK_AMD_shader_fragment_mask |
| SampleMaskOverrideCoverageNV | VK_NV_sample_mask_override_coverage |
| GeometryShaderPassthroughNV | VK_NV_geometry_shader_passthrough |
| ShaderViewportIndexLayerEXT | VK_EXT_shader_viewport_index_layer |
| ShaderViewportIndexLayerNV | VK_NV_viewport_array2 |
| ShaderViewportMaskNV | VK_NV_viewport_array2 |
| PerViewAttributesNV | VK_NVX_multiview_per_view_attributes |
| StorageBuffer16BitAccess | VkPhysicalDevice16BitStorageFeatures::storageBuffer16BitAccess |
| UniformAndStorageBuffer16BitAccess | VkPhysicalDevice16BitStorageFeatures::uniformAndStorageBuffer16BitAccess |
| StoragePushConstant16 | VkPhysicalDevice16BitStorageFeatures::storagePushConstant16 |
| StorageInputOutput16 | VkPhysicalDevice16BitStorageFeatures::storageInputOutput16 |
| GroupNonUniform | VK_SUBGROUP_FEATURE_BASIC_BIT |
| GroupNonUniformVote | VK_SUBGROUP_FEATURE_VOTE_BIT |
| GroupNonUniformArithmetic | VK_SUBGROUP_FEATURE_ARITHMETIC_BIT |
| GroupNonUniformBallot | VK_SUBGROUP_FEATURE_BALLOT_BIT |
| GroupNonUniformShuffle | VK_SUBGROUP_FEATURE_SHUFFLE_BIT |
| GroupNonUniformShuffleRelative | VK_SUBGROUP_FEATURE_SHUFFLE_RELATIVE_BIT |
| GroupNonUniformClustered | VK_SUBGROUP_FEATURE_CLUSTERED_BIT |

| SPIR-V OpCapability | Vulkan feature, extension, or core version |
|---|---|
| GroupNonUniformQuad | VK_SUBGROUP_FEATURE_QUAD_BIT |
| GroupNonUniformPartitionedNV | VK_SUBGROUP_FEATURE_PARTITIONED_BIT_NV |
| SampleMaskPostDepthCoverage | VK_EXT_post_depth_coverage |
| ShaderNonUniform | VK_EXT_descriptor_indexing |
| RuntimeDescriptorArray | VkPhysicalDeviceVulkan12Features::runtimeDescriptorArray |
| InputAttachmentArrayDynamicIndexing | VkPhysicalDeviceVulkan12Features::shaderInputAttachmentArrayDynamicIndexing |
| UniformTexelBufferArrayDynamicIndexing | VkPhysicalDeviceVulkan12Features::shaderUniformTexelBufferArrayDynamicIndexing |
| StorageTexelBufferArrayDynamicIndexing | VkPhysicalDeviceVulkan12Features::shaderStorageTexelBufferArrayDynamicIndexing |
| UniformBufferArrayNonUniformIndexing | VkPhysicalDeviceVulkan12Features::shaderUniformBufferArrayNonUniformIndexing |
| SampledImageArrayNonUniformIndexing | VkPhysicalDeviceVulkan12Features::shaderSampledImageArrayNonUniformIndexing |
| StorageBufferArrayNonUniformIndexing | VkPhysicalDeviceVulkan12Features::shaderStorageBufferArrayNonUniformIndexing |
| StorageImageArrayNonUniformIndexing | VkPhysicalDeviceVulkan12Features::shaderStorageImageArrayNonUniformIndexing |
| InputAttachmentArrayNonUniformIndexing | VkPhysicalDeviceVulkan12Features::shaderInputAttachmentArrayNonUniformIndexing |
| UniformTexelBufferArrayNonUniformIndexing | VkPhysicalDeviceVulkan12Features::shaderUniformTexelBufferArrayNonUniformIndexing |
| StorageTexelBufferArrayNonUniformIndexing | VkPhysicalDeviceVulkan12Features::shaderStorageTexelBufferArrayNonUniformIndexing |
| FragmentFullyCoveredEXT | VK_EXT_conservative_rasterization |
| Float16 | VkPhysicalDeviceVulkan12Features::shaderFloat16<br>VK_AMD_gpu_shader_half_float |
| Int8 | VkPhysicalDeviceVulkan12Features::shaderInt8 |

| SPIR-V OpCapability | Vulkan feature, extension, or core version |
| --- | --- |
| StorageBuffer8BitAccess | VkPhysicalDeviceVulkan12Features::storageBuffer8BitAccess |
| UniformAndStorageBuffer8BitAccess | VkPhysicalDeviceVulkan12Features::uniformAndStorageBuffer8BitAccess |
| StoragePushConstant8 | VkPhysicalDeviceVulkan12Features::storagePushConstant8 |
| VulkanMemoryModel | VkPhysicalDeviceVulkan12Features::vulkanMemoryModel |
| VulkanMemoryModelDeviceScope | VkPhysicalDeviceVulkan12Features::vulkanMemoryModelDeviceScope |
| DenormPreserve | VkPhysicalDeviceVulkan12Properties::shaderDenormPreserveFloat16<br>VkPhysicalDeviceVulkan12Properties::shaderDenormPreserveFloat32<br>VkPhysicalDeviceVulkan12Properties::shaderDenormPreserveFloat64 |
| DenormFlushToZero | VkPhysicalDeviceVulkan12Properties::shaderDenormFlushToZeroFloat16<br>VkPhysicalDeviceVulkan12Properties::shaderDenormFlushToZeroFloat32<br>VkPhysicalDeviceVulkan12Properties::shaderDenormFlushToZeroFloat64 |
| SignedZeroInfNanPreserve | VkPhysicalDeviceVulkan12Properties::shaderSignedZeroInfNanPreserveFloat16<br>VkPhysicalDeviceVulkan12Properties::shaderSignedZeroInfNanPreserveFloat32<br>VkPhysicalDeviceVulkan12Properties::shaderSignedZeroInfNanPreserveFloat64 |
| RoundingModeRTE | VkPhysicalDeviceVulkan12Properties::shaderRoundingModeRTEFloat16<br>VkPhysicalDeviceVulkan12Properties::shaderRoundingModeRTEFloat32<br>VkPhysicalDeviceVulkan12Properties::shaderRoundingModeRTEFloat64 |
| RoundingModeRTZ | VkPhysicalDeviceVulkan12Properties::shaderRoundingModeRTZFloat16<br>VkPhysicalDeviceVulkan12Properties::shaderRoundingModeRTZFloat32<br>VkPhysicalDeviceVulkan12Properties::shaderRoundingModeRTZFloat64 |
| ComputeDerivativeGroupQuadsNV | VkPhysicalDeviceComputeShaderDerivativesFeaturesNV::computeDerivativeGroupQuads |
| ComputeDerivativeGroupLinearNV | VkPhysicalDeviceComputeShaderDerivativesFeaturesNV::computeDerivativeGroupLinear |
| ComputeDerivativeGroupQuadsKHR | VkPhysicalDeviceComputeShaderDerivativesFeaturesKHR::computeDerivativeGroupQuads |
| ComputeDerivativeGroupLinearKHR | VkPhysicalDeviceComputeShaderDerivativesFeaturesKHR::computeDerivativeGroupLinear |

| SPIR-V OpCapability | Vulkan feature, extension, or core version |
|---|---|
| FragmentBarycentricNV | VkPhysicalDeviceFragmentShaderBarycentricFeaturesNV::fragmentShaderBarycentric |
| ImageFootprintNV | VkPhysicalDeviceShaderImageFootprintFeaturesNV::imageFootprint |
| ShadingRateNV | VkPhysicalDeviceShadingRateImageFeaturesNV::shadingRateImage |
| MeshShadingNV | VK_NV_mesh_shader |
| RayTracingKHR | VkPhysicalDeviceRayTracingPipelineFeaturesKHR::rayTracingPipeline |
| RayQueryKHR | VkPhysicalDeviceRayQueryFeaturesKHR::rayQuery |
| RayTraversalPrimitiveCullingKHR | VkPhysicalDeviceRayTracingPipelineFeaturesKHR::rayTraversalPrimitiveCulling<br>VkPhysicalDeviceRayQueryFeaturesKHR::rayQuery |
| RayCullMaskKHR | VkPhysicalDeviceRayTracingMaintenance1FeaturesKHR::rayTracingMaintenance1 |
| RayTracingNV | VK_NV_ray_tracing |
| RayTracingMotionBlurNV | VkPhysicalDeviceRayTracingMotionBlurFeaturesNV::rayTracingMotionBlur |
| TransformFeedback | VkPhysicalDeviceTransformFeedbackFeaturesEXT::transformFeedback |
| GeometryStreams | VkPhysicalDeviceTransformFeedbackFeaturesEXT::geometryStreams |
| FragmentDensityEXT | VkPhysicalDeviceFragmentDensityMapFeaturesEXT::fragmentDensityMap |
| PhysicalStorageBufferAddresses | VkPhysicalDeviceVulkan12Features::bufferDeviceAddress<br>VkPhysicalDeviceBufferDeviceAddressFeaturesEXT::bufferDeviceAddress |
| CooperativeMatrixNV | VkPhysicalDeviceCooperativeMatrixFeaturesNV::cooperativeMatrix |
| IntegerFunctions2INTEL | VkPhysicalDeviceShaderIntegerFunctions2FeaturesINTEL::shaderIntegerFunctions2 |
| ShaderSMBuiltinsNV | VkPhysicalDeviceShaderSMBuiltinsFeaturesNV::shaderSMBuiltins |
| FragmentShaderSampleInterlockEXT | VkPhysicalDeviceFragmentShaderInterlockFeaturesEXT::fragmentShaderSampleInterlock |

| SPIR-V OpCapability | Vulkan feature, extension, or core version |
| --- | --- |
| FragmentShaderPixelInterlockEXT | VkPhysicalDeviceFragmentShaderInterlockFeaturesEXT::fragmentShaderPixelInterlock |
| FragmentShaderShadingRateInterlockEXT | VkPhysicalDeviceFragmentShaderInterlockFeaturesEXT::fragmentShaderShadingRateInterlock<br>VkPhysicalDeviceShadingRateImageFeaturesNV::shadingRateImage |
| DemoteToHelperInvocationEXT | VkPhysicalDeviceVulkan13Features::shaderDemoteToHelperInvocation<br>VkPhysicalDeviceShaderDemoteToHelperInvocationFeaturesEXT::shaderDemoteToHelperInvocation |
| FragmentShadingRateKHR | VkPhysicalDeviceFragmentShadingRateFeaturesKHR::pipelineFragmentShadingRate<br>VkPhysicalDeviceFragmentShadingRateFeaturesKHR::primitiveFragmentShadingRate<br>VkPhysicalDeviceFragmentShadingRateFeaturesKHR::attachmentFragmentShadingRate |
| WorkgroupMemoryExplicitLayoutKHR | VkPhysicalDeviceWorkgroupMemoryExplicitLayoutFeaturesKHR::workgroupMemoryExplicitLayout |
| WorkgroupMemoryExplicitLayout8BitAccessKHR | VkPhysicalDeviceWorkgroupMemoryExplicitLayoutFeaturesKHR::workgroupMemoryExplicitLayout8BitAccess |
| WorkgroupMemoryExplicitLayout16BitAccessKHR | VkPhysicalDeviceWorkgroupMemoryExplicitLayoutFeaturesKHR::workgroupMemoryExplicitLayout16BitAccess |
| DotProductInputAllKHR | VkPhysicalDeviceVulkan13Features::shaderIntegerDotProduct<br>VkPhysicalDeviceShaderIntegerDotProductFeaturesKHR::shaderIntegerDotProduct |
| DotProductInput4x8BitKHR | VkPhysicalDeviceVulkan13Features::shaderIntegerDotProduct<br>VkPhysicalDeviceShaderIntegerDotProductFeaturesKHR::shaderIntegerDotProduct |
| DotProductInput4x8BitPackedKHR | VkPhysicalDeviceVulkan13Features::shaderIntegerDotProduct<br>VkPhysicalDeviceShaderIntegerDotProductFeaturesKHR::shaderIntegerDotProduct |
| DotProductKHR | VkPhysicalDeviceVulkan13Features::shaderIntegerDotProduct<br>VkPhysicalDeviceShaderIntegerDotProductFeaturesKHR::shaderIntegerDotProduct |
| FragmentBarycentricKHR | VkPhysicalDeviceFragmentShaderBarycentricFeaturesKHR::fragmentShaderBarycentric |
| TextureSampleWeightedQCOM | VkPhysicalDeviceImageProcessingFeaturesQCOM::textureSampleWeighted |
| TextureBoxFilterQCOM | VkPhysicalDeviceImageProcessingFeaturesQCOM::textureBoxFilter |

| SPIR-V OpCapability | Vulkan feature, extension, or core version |
| --- | --- |
| TextureBlockMatchQCOM | VkPhysicalDeviceImageProcessingFeaturesQCOM::textureBlockMatch |
| TextureBlockMatch2QCOM | VkPhysicalDeviceImageProcessing2FeaturesQCOM::textureBlockMatch2 |
| MeshShadingEXT | VK_EXT_mesh_shader |
| RayTracingOpacityMicromapEXT | VK_EXT_opacity_micromap |
| CoreBuiltinsARM | VkPhysicalDeviceShaderCoreBuiltinsFeaturesARM::shaderCoreBuiltins |
| ShaderInvocationReorderNV | VK_NV_ray_tracing_invocation_reorder |
| ClusterCullingShadingHUAWEI | VkPhysicalDeviceClusterCullingShaderFeaturesHUAWEI::clustercullingShader |
| RayTracingPositionFetchKHR | VkPhysicalDeviceRayTracingPositionFetchFeaturesKHR::rayTracingPositionFetch |
| RayQueryPositionFetchKHR | VkPhysicalDeviceRayTracingPositionFetchFeaturesKHR::rayTracingPositionFetch |
| TileImageColorReadAccessEXT | VkPhysicalDeviceShaderTileImageFeaturesEXT::shaderTileImageColorReadAccess |
| TileImageDepthReadAccessEXT | VkPhysicalDeviceShaderTileImageFeaturesEXT::shaderTileImageDepthReadAccess |
| TileImageStencilReadAccessEXT | VkPhysicalDeviceShaderTileImageFeaturesEXT::shaderTileImageStencilReadAccess |
| CooperativeMatrixKHR | VkPhysicalDeviceCooperativeMatrixFeaturesKHR::cooperativeMatrix |
| ShaderEnqueueAMDX | VkPhysicalDeviceShaderEnqueueFeaturesAMDX::shaderEnqueue |
| GroupNonUniformRotateKHR | VkPhysicalDeviceShaderSubgroupRotateFeaturesKHR::shaderSubgroupRotate |
| ExpectAssumeKHR | VkPhysicalDeviceShaderExpectAssumeFeaturesKHR::shaderExpectAssume |
| FloatControls2 | VkPhysicalDeviceShaderFloatControls2FeaturesKHR::shaderFloatControls2 |
| QuadControlKHR | VkPhysicalDeviceShaderQuadControlFeaturesKHR::shaderQuadControl |
| RawAccessChainsNV | VkPhysicalDeviceRawAccessChainsFeaturesNV::shaderRawAccessChains |

| SPIR-V OpCapability | Vulkan feature, extension, or core version |
|---|---|
| ReplicatedCompositesEXT | VkPhysicalDeviceShaderReplicatedCompositesFeaturesEXT::shaderReplicatedComposites |

The application **must** not pass a SPIR-V module containing any of the following to vkCreateShaderModule:

- any OpCapability not listed above,
- an unsupported capability, or
- a capability which corresponds to a Vulkan feature or extension which has not been enabled.

## SPIR-V Extensions

The following table lists SPIR-V extensions that implementations **may** support. The application **must** not pass a SPIR-V module to vkCreateShaderModule that uses the following SPIR-V extensions unless one of the following conditions is met for the VkDevice specified in the device parameter of vkCreateShaderModule:

- Any corresponding Vulkan extension is enabled.
- The corresponding core version is supported (as returned by VkPhysicalDeviceProperties::apiVersion).

*Table 97. List of SPIR-V Extensions and corresponding Vulkan extensions or core version*

| SPIR-V OpExtension | Vulkan extension or core version |
|---|---|
| SPV_KHR_variable_pointers | VK_VERSION_1_1<br>VK_KHR_variable_pointers |
| SPV_AMD_shader_explicit_vertex_parameter | VK_AMD_shader_explicit_vertex_parameter |
| SPV_AMD_gcn_shader | VK_AMD_gcn_shader |
| SPV_AMD_gpu_shader_half_float | VK_AMD_gpu_shader_half_float |
| SPV_AMD_gpu_shader_int16 | VK_AMD_gpu_shader_int16 |
| SPV_AMD_shader_ballot | VK_AMD_shader_ballot |
| SPV_AMD_shader_fragment_mask | VK_AMD_shader_fragment_mask |
| SPV_AMD_shader_image_load_store_lod | VK_AMD_shader_image_load_store_lod |

| **SPIR-V** OpExtension | **Vulkan extension or core version** |
| --- | --- |
| SPV_AMD_shader_trinary_minmax | VK_AMD_shader_trinary_minmax |
| SPV_AMD_texture_gather_bias_lod | VK_AMD_texture_gather_bias_lod |
| SPV_AMD_shader_early_and_late_fragment_tests | VK_AMD_shader_early_and_late_fragment_tests |
| SPV_KHR_shader_draw_parameters | VK_VERSION_1_1<br>VK_KHR_shader_draw_parameters |
| SPV_KHR_8bit_storage | VK_KHR_8bit_storage |
| SPV_KHR_16bit_storage | VK_VERSION_1_1<br>VK_KHR_16bit_storage |
| SPV_KHR_shader_clock | VK_KHR_shader_clock |
| SPV_KHR_float_controls | VK_KHR_shader_float_controls |
| SPV_KHR_storage_buffer_storage_class | VK_VERSION_1_1<br>VK_KHR_storage_buffer_storage_class |
| SPV_KHR_post_depth_coverage | VK_EXT_post_depth_coverage |
| SPV_EXT_shader_stencil_export | VK_EXT_shader_stencil_export |
| SPV_KHR_shader_ballot | VK_EXT_shader_subgroup_ballot |
| SPV_KHR_subgroup_vote | VK_EXT_shader_subgroup_vote |
| SPV_NV_sample_mask_override_coverage | VK_NV_sample_mask_override_coverage |
| SPV_NV_geometry_shader_passthrough | VK_NV_geometry_shader_passthrough |
| SPV_NV_mesh_shader | VK_NV_mesh_shader |
| SPV_NV_viewport_array2 | VK_NV_viewport_array2 |

| SPIR-V OpExtension | Vulkan extension or core version |
| --- | --- |
| SPV_NV_shader_subgroup_partitioned | VK_NV_shader_subgroup_partitioned |
| SPV_NV_shader_invocation_reorder | VK_NV_ray_tracing_invocation_reorder |
| SPV_EXT_shader_viewport_index_layer | VK_EXT_shader_viewport_index_layer |
| SPV_NVX_multiview_per_view_attributes | VK_NVX_multiview_per_view_attributes |
| SPV_EXT_descriptor_indexing | VK_EXT_descriptor_indexing |
| SPV_KHR_vulkan_memory_model | VK_KHR_vulkan_memory_model |
| SPV_NV_compute_shader_derivatives | VK_NV_compute_shader_derivatives |
| SPV_NV_fragment_shader_barycentric | VK_NV_fragment_shader_barycentric |
| SPV_NV_shader_image_footprint | VK_NV_shader_image_footprint |
| SPV_NV_shading_rate | VK_NV_shading_rate_image |
| SPV_NV_ray_tracing | VK_NV_ray_tracing |
| SPV_KHR_ray_tracing | VK_KHR_ray_tracing_pipeline |
| SPV_KHR_ray_query | VK_KHR_ray_query |
| SPV_KHR_ray_cull_mask | VK_KHR_ray_tracing_maintenance1 |
| SPV_GOOGLE_hlsl_functionality1 | VK_GOOGLE_hlsl_functionality1 |
| SPV_GOOGLE_user_type | VK_GOOGLE_user_type |
| SPV_GOOGLE_decorate_string | VK_GOOGLE_decorate_string |
| SPV_EXT_fragment_invocation_density | VK_EXT_fragment_density_map |
| SPV_KHR_physical_storage_buffer | VK_KHR_buffer_device_address |

| SPIR-V OpExtension | Vulkan extension or core version |
| --- | --- |
| SPV_EXT_physical_storage_buffer | VK_EXT_buffer_device_address |
| SPV_NV_cooperative_matrix | VK_NV_cooperative_matrix |
| SPV_NV_shader_sm_builtins | VK_NV_shader_sm_builtins |
| SPV_EXT_fragment_shader_interlock | VK_EXT_fragment_shader_interlock |
| SPV_EXT_demote_to_helper_invocation | VK_EXT_shader_demote_to_helper_invocation |
| SPV_KHR_fragment_shading_rate | VK_KHR_fragment_shading_rate |
| SPV_KHR_non_semantic_info | VK_KHR_shader_non_semantic_info |
| SPV_EXT_shader_image_int64 | VK_EXT_shader_image_atomic_int64 |
| SPV_KHR_terminate_invocation | VK_KHR_shader_terminate_invocation |
| SPV_KHR_multiview | VK_VERSION_1_1<br>VK_KHR_multiview |
| SPV_KHR_workgroup_memory_explicit_layout | VK_KHR_workgroup_memory_explicit_layout |
| SPV_EXT_shader_atomic_float_add | VK_EXT_shader_atomic_float |
| SPV_KHR_fragment_shader_barycentric | VK_KHR_fragment_shader_barycentric |
| SPV_KHR_subgroup_uniform_control_flow | VK_KHR_shader_subgroup_uniform_control_flow |
| SPV_EXT_shader_atomic_float_min_max | VK_EXT_shader_atomic_float2 |
| SPV_EXT_shader_atomic_float16_add | VK_EXT_shader_atomic_float2 |
| SPV_NV_shader_atomic_fp16_vector | VK_NV_shader_atomic_float16_vector |
| SPV_EXT_fragment_fully_covered | VK_EXT_conservative_rasterization |

| SPIR-V OpExtension | Vulkan extension or core version |
| --- | --- |
| SPV_KHR_integer_dot_product | VK_KHR_shader_integer_dot_product |
| SPV_INTEL_shader_integer_functions2 | VK_INTEL_shader_integer_functions2 |
| SPV_KHR_device_group | VK_VERSION_1_1<br>VK_KHR_device_group |
| SPV_QCOM_image_processing | VK_QCOM_image_processing |
| SPV_QCOM_image_processing2 | VK_QCOM_image_processing2 |
| SPV_EXT_mesh_shader | VK_EXT_mesh_shader |
| SPV_KHR_ray_tracing_position_fetch | VK_KHR_ray_tracing_position_fetch |
| SPV_EXT_shader_tile_image | VK_EXT_shader_tile_image |
| SPV_EXT_opacity_micromap | VK_EXT_opacity_micromap |
| SPV_KHR_cooperative_matrix | VK_KHR_cooperative_matrix |
| SPV_ARM_core_builtins | VK_ARM_shader_core_builtins |
| SPV_AMDX_shader_enqueue | VK_AMDX_shader_enqueue |
| SPV_HUAWEI_cluster_culling_shader | VK_HUAWEI_cluster_culling_shader |
| SPV_HUAWEI_subpass_shading | VK_HUAWEI_subpass_shading |
| SPV_NV_ray_tracing_motion_blur | VK_NV_ray_tracing_motion_blur |
| SPV_KHR_maximal_reconvergence | VK_KHR_shader_maximal_reconvergence |
| SPV_KHR_subgroup_rotate | VK_KHR_shader_subgroup_rotate |
| SPV_KHR_expect_assume | VK_KHR_shader_expect_assume |

| SPIR-V OpExtension | Vulkan extension or core version |
| --- | --- |
| SPV_KHR_float_controls2 | VK_KHR_shader_float_controls2 |
| SPV_KHR_quad_control | VK_KHR_shader_quad_control |
| SPV_NV_raw_access_chains | VK_NV_raw_access_chains |
| SPV_KHR_compute_shader_derivatives | VK_KHR_compute_shader_derivatives |
| SPV_EXT_replicated_composites | VK_EXT_shader_replicated_composites |
| SPV_KHR_relaxed_extended_instruction | VK_KHR_shader_relaxed_extended_instruction |

# Validation Rules Within a Module

A SPIR-V module passed to vkCreateShaderModule **must** conform to the following rules:

## Standalone SPIR-V Validation

The following rules **can** be validated with only the SPIR-V module itself. They do not depend on knowledge of the implementation and its capabilities or knowledge of runtime information, such as enabled features.

### Valid Usage

- VUID-StandaloneSpirv-None-04633
  Every entry point **must** have no return value and accept no arguments
- VUID-StandaloneSpirv-None-04634
  The static function-call graph for an entry point **must** not contain cycles; that is, static recursion is not allowed
- VUID-StandaloneSpirv-None-04635
  The Logical or PhysicalStorageBuffer64 addressing model **must** be selected
- VUID-StandaloneSpirv-None-04636
  Scope for execution **must** be limited to Workgroup or Subgroup
- VUID-StandaloneSpirv-None-04637
  If the Scope for execution is Workgroup, then it **must** only be used in the task, mesh, tessellation control, or compute Execution Model
- VUID-StandaloneSpirv-None-04638
  Scope for memory **must** be limited to Device, QueueFamily, Workgroup, ShaderCallKHR, Subgroup, or Invocation

- VUID-StandaloneSpirv-ExecutionModel-07320
  If the `Execution Model` is `TessellationControl`, and the `MemoryModel` is `GLSL450`, the `Scope` for memory **must** not be `Workgroup`
- VUID-StandaloneSpirv-None-07321
  If the `Scope` for memory is `Workgroup`, then it **must** only be used in the task, mesh, tessellation control, or compute `Execution Model`
- VUID-StandaloneSpirv-None-04640
  If the `Scope` for memory is `ShaderCallKHR`, then it **must** only be used in ray generation, intersection, closest hit, any-hit, miss, and callable `Execution Model`
- VUID-StandaloneSpirv-None-04641
  If the `Scope` for memory is `Invocation`, then memory semantics **must** be `None`
- VUID-StandaloneSpirv-None-04642
  `Scope` for group operations **must** be limited to `Subgroup`
- VUID-StandaloneSpirv-SubgroupVoteKHR-07951
  If none of the `SubgroupVoteKHR`, `GroupNonUniform`, or `SubgroupBallotKHR` capabilities are declared, `Scope` for memory **must** not be `Subgroup`
- VUID-StandaloneSpirv-None-04643
  `Storage Class` **must** be limited to `UniformConstant`, `Input`, `Uniform`, `Output`, `Workgroup`, `Private`, `Function`, `PushConstant`, `Image`, `StorageBuffer`, `RayPayloadKHR`, `IncomingRayPayloadKHR`, `HitAttributeKHR`, `CallableDataKHR`, `IncomingCallableDataKHR`, `ShaderRecordBufferKHR`, `PhysicalStorageBuffer`, or `TileImageEXT`
- VUID-StandaloneSpirv-None-04644
  If the `Storage Class` is `Output`, then it **must** not be used in the `GlCompute`, `RayGenerationKHR`, `IntersectionKHR`, `AnyHitKHR`, `ClosestHitKHR`, `MissKHR`, or `CallableKHR Execution Model`
- VUID-StandaloneSpirv-None-04645
  If the `Storage Class` is `Workgroup`, then it **must** only be used in the task, mesh, or compute `Execution Model`
- VUID-StandaloneSpirv-None-08720
  If the `Storage Class` is `TileImageEXT`, then it **must** only be used in the fragment execution model
- VUID-StandaloneSpirv-OpAtomicStore-04730
  `OpAtomicStore` **must** not use `Acquire`, `AcquireRelease`, or `SequentiallyConsistent` memory semantics
- VUID-StandaloneSpirv-OpAtomicLoad-04731
  `OpAtomicLoad` **must** not use `Release`, `AcquireRelease`, or `SequentiallyConsistent` memory semantics
- VUID-StandaloneSpirv-OpMemoryBarrier-04732
  `OpMemoryBarrier` **must** use one of `Acquire`, `Release`, `AcquireRelease`, or `SequentiallyConsistent` memory semantics
- VUID-StandaloneSpirv-OpMemoryBarrier-04733
  `OpMemoryBarrier` **must** include at least one `Storage Class`
- VUID-StandaloneSpirv-OpControlBarrier-04650

If the semantics for OpControlBarrier includes one of Acquire, Release, AcquireRelease, or SequentiallyConsistent memory semantics, then it **must** include at least one Storage Class

- VUID-StandaloneSpirv-OpVariable-04651
  Any OpVariable with an Initializer operand **must** have Output, Private, Function, or Workgroup as its Storage Class operand
- VUID-StandaloneSpirv-OpVariable-04734
  Any OpVariable with an Initializer operand and Workgroup as its Storage Class operand **must** use OpConstantNull as the initializer
- VUID-StandaloneSpirv-OpReadClockKHR-04652
  Scope for OpReadClockKHR **must** be limited to Subgroup or Device
- VUID-StandaloneSpirv-OriginLowerLeft-04653
  The OriginLowerLeft Execution Mode **must** not be used; fragment entry points **must** declare OriginUpperLeft
- VUID-StandaloneSpirv-PixelCenterInteger-04654
  The PixelCenterInteger Execution Mode **must** not be used (pixels are always centered at half-integer coordinates)
- VUID-StandaloneSpirv-UniformConstant-04655
  Any variable in the UniformConstant Storage Class **must** be typed as either OpTypeImage, OpTypeSampler, OpTypeSampledImage, OpTypeAccelerationStructureKHR, or an array of one of these types
- VUID-StandaloneSpirv-Uniform-06807
  Any variable in the Uniform or StorageBuffer Storage Class **must** be typed as OpTypeStruct or an array of this type
- VUID-StandaloneSpirv-PushConstant-06808
  Any variable in the PushConstant Storage Class **must** be typed as OpTypeStruct
- VUID-StandaloneSpirv-OpTypeImage-04656
  OpTypeImage **must** declare a scalar 32-bit float, 64-bit integer, or 32-bit integer type for the "Sampled Type" (RelaxedPrecision **can** be applied to a sampling instruction and to the variable holding the result of a sampling instruction)
- VUID-StandaloneSpirv-OpTypeImage-04657
  OpTypeImage **must** have a "Sampled" operand of 1 (sampled image) or 2 (storage image)
- VUID-StandaloneSpirv-OpTypeSampledImage-06671
  OpTypeSampledImage **must** have a OpTypeImage with a "Sampled" operand of 1 (sampled image)
- VUID-StandaloneSpirv-Image-04965
  The SPIR-V Type of the Image Format operand of an OpTypeImage **must** match the Sampled Type, as defined in Image Format and Type Matching
- VUID-StandaloneSpirv-OpImageTexelPointer-04658
  If an OpImageTexelPointer is used in an atomic operation, the image type of the image parameter to OpImageTexelPointer **must** have an image format of R64i, R64ui, R32f, R32i, or R32ui
- VUID-StandaloneSpirv-OpImageQuerySizeLod-04659

OpImageQuerySizeLod, OpImageQueryLod, and OpImageQueryLevels **must** only consume an "Image" operand whose type has its "Sampled" operand set to 1

- VUID-StandaloneSpirv-OpTypeImage-09638
  An OpTypeImage **must** not have a "Dim" operand of Rect
- VUID-StandaloneSpirv-OpTypeImage-06214
  An OpTypeImage with a "Dim" operand of SubpassData **must** have an "Arrayed" operand of 0 (non-arrayed) and a "Sampled" operand of 2 (storage image)
- VUID-StandaloneSpirv-SubpassData-04660
  The (u,v) coordinates used for a SubpassData **must** be the <id> of a constant vector (0,0).
- VUID-StandaloneSpirv-OpTypeImage-06924
  Objects of types OpTypeImage, OpTypeSampler, OpTypeSampledImage, OpTypeAccelerationStructureKHR, and arrays of these types **must** not be stored to or modified
- VUID-StandaloneSpirv-Uniform-06925
  Any variable in the Uniform Storage Class decorated as Block **must** not be stored to or modified
- VUID-StandaloneSpirv-Offset-04663
  Image operand Offset **must** only be used with OpImage*Gather instructions
- VUID-StandaloneSpirv-Offset-04865
  Any image instruction which uses an Offset, ConstOffset, or ConstOffsets image operand, **must** only consume a "Sampled Image" operand whose type has its "Sampled" operand set to 1
- VUID-StandaloneSpirv-OpImageGather-04664
  The "Component" operand of OpImageGather, and OpImageSparseGather **must** be the <id> of a constant instruction
- VUID-StandaloneSpirv-OpImage-04777
  OpImage*Dref* instructions **must** not consume an image whose Dim is 3D
- VUID-StandaloneSpirv-None-04667
  Structure types **must** not contain opaque types
- VUID-StandaloneSpirv-BuiltIn-04668
  Any BuiltIn decoration not listed in Built-In Variables **must** not be used
- VUID-StandaloneSpirv-OpEntryPoint-09658
  For a given OpEntryPoint, any BuiltIn decoration **must** not be used more than once by the Input interface.
- VUID-StandaloneSpirv-OpEntryPoint-09659
  For a given OpEntryPoint, any BuiltIn decoration **must** not be used more than once by the Output interface.
- VUID-StandaloneSpirv-Location-06672
  The Location or Component decorations **must** only be used with the Input, Output, RayPayloadKHR, IncomingRayPayloadKHR, HitAttributeKHR, HitObjectAttributeNV, CallableDataKHR, IncomingCallableDataKHR, or ShaderRecordBufferKHR storage classes
- VUID-StandaloneSpirv-Location-04915

The `Location` or `Component` decorations **must** not be used with `BuiltIn`

- VUID-StandaloneSpirv-Location-04916
  The `Location` decorations **must** be used on user-defined variables
- VUID-StandaloneSpirv-Location-04917
  If a user-defined variable is not a pointer to a `Block` decorated `OpTypeStruct`, then the `OpVariable` **must** have a `Location` decoration
- VUID-StandaloneSpirv-Location-04918
  If a user-defined variable has a `Location` decoration, and the variable is a pointer to a `OpTypeStruct`, then the members of that structure **must** not have `Location` decorations
- VUID-StandaloneSpirv-Location-04919
  If a user-defined variable does not have a `Location` decoration, and the variable is a pointer to a `Block` decorated `OpTypeStruct`, then each member of the struct **must** have a `Location` decoration
- VUID-StandaloneSpirv-Component-04920
  The `Component` decoration value **must** not be greater than 3
- VUID-StandaloneSpirv-Component-04921
  If the `Component` decoration is used on an `OpVariable` that has a `OpTypeVector` type with a `Component Type` with a `Width` that is less than or equal to 32, the sum of its `Component Count` and the `Component` decoration value **must** be less than or equal to 4
- VUID-StandaloneSpirv-Component-04922
  If the `Component` decoration is used on an `OpVariable` that has a `OpTypeVector` type with a `Component Type` with a `Width` that is equal to 64, the sum of two times its `Component Count` and the `Component` decoration value **must** be less than or equal to 4
- VUID-StandaloneSpirv-Component-04923
  The `Component` decorations value **must** not be 1 or 3 for scalar or two-component 64-bit data types
- VUID-StandaloneSpirv-Component-04924
  The `Component` decorations **must** not be used with any type that is not a scalar or vector, or an array of such a type
- VUID-StandaloneSpirv-Component-07703
  The `Component` decorations **must** not be used for a 64-bit vector type with more than two components
- VUID-StandaloneSpirv-Input-09557
  The pointers of any `Input` or `Output` Interface user-defined variables **must** not contain any `PhysicalStorageBuffer Storage Class` pointers
- VUID-StandaloneSpirv-GLSLShared-04669
  The `GLSLShared` and `GLSLPacked` decorations **must** not be used
- VUID-StandaloneSpirv-Flat-04670
  The `Flat`, `NoPerspective`, `Sample`, and `Centroid` decorations **must** only be used on variables with the `Output` or `Input Storage Class`
- VUID-StandaloneSpirv-Flat-06201
  The `Flat`, `NoPerspective`, `Sample`, and `Centroid` decorations **must** not be used on variables

with the `Output` storage class in a fragment shader

- VUID-StandaloneSpirv-Flat-06202
  The `Flat`, `NoPerspective`, `Sample`, and `Centroid` decorations **must** not be used on variables with the `Input` storage class in a vertex shader
- VUID-StandaloneSpirv-PerVertexKHR-06777
  The `PerVertexKHR` decoration **must** only be used on variables with the `Input Storage Class` in a fragment shader
- VUID-StandaloneSpirv-Flat-04744
  Any variable with integer or double-precision floating-point type and with `Input Storage Class` in a fragment shader, **must** be decorated `Flat`
- VUID-StandaloneSpirv-ViewportRelativeNV-04672
  The `ViewportRelativeNV` decoration **must** only be used on a variable decorated with `Layer` in the vertex, tessellation evaluation, or geometry shader stages
- VUID-StandaloneSpirv-ViewportRelativeNV-04673
  The `ViewportRelativeNV` decoration **must** not be used unless a variable decorated with one of `ViewportIndex` or `ViewportMaskNV` is also statically used by the same `OpEntryPoint`
- VUID-StandaloneSpirv-ViewportMaskNV-04674
  The `ViewportMaskNV` and `ViewportIndex` decorations **must** not both be statically used by one or more `OpEntryPoint`'s that form the pre-rasterization shader stages of a graphics pipeline
- VUID-StandaloneSpirv-FPRoundingMode-04675
  Rounding modes other than round-to-nearest-even and round-towards-zero **must** not be used for the `FPRoundingMode` decoration
- VUID-StandaloneSpirv-Invariant-04677
  Variables decorated with `Invariant` and variables with structure types that have any members decorated with `Invariant` **must** be in the `Output` or `Input Storage Class`, `Invariant` used on an `Input Storage Class` variable or structure member has no effect
- VUID-StandaloneSpirv-VulkanMemoryModel-04678
  If the `VulkanMemoryModel` capability is not declared, the `Volatile` decoration **must** be used on any variable declaration that includes one of the `SMIDNV`, `WarpIDNV`, `SubgroupSize`, `SubgroupLocalInvocationId`, `SubgroupEqMask`, `SubgroupGeMask`, `SubgroupGtMask`, `SubgroupLeMask`, or `SubgroupLtMask BuiltIn` decorations when used in the ray generation, closest hit, miss, intersection, or callable shaders, or with the `RayTmaxKHR Builtin` decoration when used in an intersection shader
- VUID-StandaloneSpirv-VulkanMemoryModel-04679
  If the `VulkanMemoryModel` capability is declared, the `OpLoad` instruction **must** use the `Volatile` memory semantics when it accesses into any variable that includes one of the `SMIDNV`, `WarpIDNV`, `SubgroupSize`, `SubgroupLocalInvocationId`, `SubgroupEqMask`, `SubgroupGeMask`, `SubgroupGtMask`, `SubgroupLeMask`, or `SubgroupLtMask BuiltIn` decorations when used in the ray generation, closest hit, miss, intersection, or callable shaders, or with the `RayTmaxKHR Builtin` decoration when used in an intersection shader
- VUID-StandaloneSpirv-OpTypeRuntimeArray-04680
  `OpTypeRuntimeArray` **must** only be used for:
  - the last member of a `Block`-decorated `OpTypeStruct` in `StorageBuffer` or

PhysicalStorageBuffer storage Storage Class
    - BufferBlock-decorated OpTypeStruct in the Uniform storage Storage Class
    - the outermost dimension of an arrayed variable in the StorageBuffer, Uniform, or UniformConstant storage Storage Class
    - variables in the NodePayloadAMDX storage Storage Class when the CoalescingAMDX Execution Mode is specified

- VUID-StandaloneSpirv-Function-04681
  A type *T* that is an array sized with a specialization constant **must** neither be, nor be contained in, the type *T2* of a variable *V*, unless either: a) *T* is equal to *T2*, b) *V* is declared in the Function, or Private Storage Class, c) *V* is a non-Block variable in the Workgroup Storage Class, or d) *V* is an interface variable with an additional level of arrayness, as described in interface matching, and *T* is the member type of the array type *T2*
- VUID-StandaloneSpirv-OpControlBarrier-04682
  If OpControlBarrier is used in ray generation, intersection, any-hit, closest hit, miss, fragment, vertex, tessellation evaluation, or geometry shaders, the execution Scope **must** be Subgroup
- VUID-StandaloneSpirv-LocalSize-06426
  For each compute shader entry point, either a LocalSize or LocalSizeId Execution Mode, or an object decorated with the WorkgroupSize decoration **must** be specified
- VUID-StandaloneSpirv-DerivativeGroupQuadsNV-04684
  For compute shaders using the DerivativeGroupQuadsNV execution mode, the first two dimensions of the local workgroup size **must** be a multiple of two
- VUID-StandaloneSpirv-DerivativeGroupLinearNV-04778
  For compute shaders using the DerivativeGroupLinearNV execution mode, the product of the dimensions of the local workgroup size **must** be a multiple of four
- VUID-StandaloneSpirv-DerivativeGroupQuadsKHR-10151
  For compute, mesh, or task shaders using the DerivativeGroupQuadsKHR execution mode, the first two dimensions of the local workgroup size **must** be a multiple of two
- VUID-StandaloneSpirv-DerivativeGroupLinearKHR-10152
  For compute, mesh, or task shaders using the DerivativeGroupLinearKHR execution mode, the product of the dimensions of the local workgroup size **must** be a multiple of four
- VUID-StandaloneSpirv-OpGroupNonUniformBallotBitCount-04685
  If OpGroupNonUniformBallotBitCount is used, the group operation **must** be limited to Reduce, InclusiveScan, or ExclusiveScan
- VUID-StandaloneSpirv-None-04686
  The *Pointer* operand of all atomic instructions **must** have a Storage Class limited to Uniform, Workgroup, Image, StorageBuffer, PhysicalStorageBuffer, or TaskPayloadWorkgroupEXT
- VUID-StandaloneSpirv-Offset-04687
  Output variables or block members decorated with Offset that have a 64-bit type, or a composite type containing a 64-bit type, **must** specify an Offset value aligned to a 8 byte boundary
- VUID-StandaloneSpirv-Offset-04689

The size of any output block containing any member decorated with `Offset` that is a 64-bit type **must** be a multiple of 8

- VUID-StandaloneSpirv-Offset-04690
  The first member of an output block specifying a `Offset` decoration **must** specify a `Offset` value that is aligned to an 8 byte boundary if that block contains any member decorated with `Offset` and is a 64-bit type
- VUID-StandaloneSpirv-Offset-04691
  Output variables or block members decorated with `Offset` that have a 32-bit type, or a composite type contains a 32-bit type, **must** specify an `Offset` value aligned to a 4 byte boundary
- VUID-StandaloneSpirv-Offset-04692
  Output variables, blocks or block members decorated with `Offset` **must** only contain base types that have components that are either 32-bit or 64-bit in size
- VUID-StandaloneSpirv-Offset-04716
  Only variables or block members in the output interface decorated with `Offset` **can** be captured for transform feedback, and those variables or block members **must** also be decorated with `XfbBuffer` and `XfbStride`, or inherit `XfbBuffer` and `XfbStride` decorations from a block containing them
- VUID-StandaloneSpirv-XfbBuffer-04693
  All variables or block members in the output interface of the entry point being compiled decorated with a specific `XfbBuffer` value **must** all be decorated with identical `XfbStride` values
- VUID-StandaloneSpirv-Stream-04694
  If any variables or block members in the output interface of the entry point being compiled are decorated with `Stream`, then all variables belonging to the same `XfbBuffer` **must** specify the same `Stream` value
- VUID-StandaloneSpirv-XfbBuffer-04696
  For any two variables or block members in the output interface of the entry point being compiled with the same `XfbBuffer` value, the ranges determined by the `Offset` decoration and the size of the type **must** not overlap
- VUID-StandaloneSpirv-XfbBuffer-04697
  All block members in the output interface of the entry point being compiled that are in the same block and have a declared or inherited `XfbBuffer` decoration **must** specify the same `XfbBuffer` value
- VUID-StandaloneSpirv-RayPayloadKHR-04698
  `RayPayloadKHR Storage Class` **must** only be used in ray generation, closest hit or miss shaders
- VUID-StandaloneSpirv-IncomingRayPayloadKHR-04699
  `IncomingRayPayloadKHR Storage Class` **must** only be used in closest hit, any-hit, or miss shaders
- VUID-StandaloneSpirv-IncomingRayPayloadKHR-04700
  There **must** be at most one variable with the `IncomingRayPayloadKHR Storage Class` in the input interface of an entry point

- VUID-StandaloneSpirv-HitAttributeKHR-04701
  `HitAttributeKHR Storage Class` **must** only be used in intersection, any-hit, or closest hit shaders
- VUID-StandaloneSpirv-HitAttributeKHR-04702
  There **must** be at most one variable with the `HitAttributeKHR Storage Class` in the input interface of an entry point
- VUID-StandaloneSpirv-HitAttributeKHR-04703
  A variable with `HitAttributeKHR Storage Class` **must** only be written to in an intersection shader
- VUID-StandaloneSpirv-CallableDataKHR-04704
  `CallableDataKHR Storage Class` **must** only be used in ray generation, closest hit, miss, and callable shaders
- VUID-StandaloneSpirv-IncomingCallableDataKHR-04705
  `IncomingCallableDataKHR Storage Class` **must** only be used in callable shaders
- VUID-StandaloneSpirv-IncomingCallableDataKHR-04706
  There **must** be at most one variable with the `IncomingCallableDataKHR Storage Class` in the input interface of an entry point
- VUID-StandaloneSpirv-ShaderRecordBufferKHR-07119
  `ShaderRecordBufferKHR Storage Class` **must** only be used in ray generation, intersection, any-hit, closest hit, callable, or miss shaders
- VUID-StandaloneSpirv-Base-07650
  The `Base` operand of `OpPtrAccessChain` **must** have a storage class of `Workgroup`, `StorageBuffer`, or `PhysicalStorageBuffer`
- VUID-StandaloneSpirv-Base-07651
  If the `Base` operand of `OpPtrAccessChain` has a `Workgroup Storage Class`, then the `VariablePointers` capability **must** be declared
- VUID-StandaloneSpirv-Base-07652
  If the `Base` operand of `OpPtrAccessChain` has a `StorageBuffer Storage Class`, then the `VariablePointers` or `VariablePointersStorageBuffer` capability **must** be declared
- VUID-StandaloneSpirv-PhysicalStorageBuffer64-04708
  If the `PhysicalStorageBuffer64` addressing model is enabled, all instructions that support memory access operands and that use a physical pointer **must** include the `Aligned` operand
- VUID-StandaloneSpirv-PhysicalStorageBuffer64-04709
  If the `PhysicalStorageBuffer64` addressing model is enabled, any access chain instruction that accesses into a `RowMajor` matrix **must** only be used as the `Pointer` operand to `OpLoad` or `OpStore`
- VUID-StandaloneSpirv-PhysicalStorageBuffer64-04710
  If the `PhysicalStorageBuffer64` addressing model is enabled, `OpConvertUToPtr` and `OpConvertPtrToU` **must** use an integer type whose `Width` is 64
- VUID-StandaloneSpirv-OpTypeForwardPointer-04711
  `OpTypeForwardPointer` **must** have a `Storage Class` of `PhysicalStorageBuffer`

- VUID-StandaloneSpirv-None-04745
  All block members in a variable with a `Storage Class` of `PushConstant` declared as an array **must** only be accessed by dynamically uniform indices
- VUID-StandaloneSpirv-OpVariable-06673
  There **must** not be more than one `OpVariable` in the `PushConstant Storage Class` listed in the `Interface` for each `OpEntryPoint`
- VUID-StandaloneSpirv-OpEntryPoint-06674
  Each `OpEntryPoint` **must** not statically use more than one `OpVariable` in the `PushConstant Storage Class`
- VUID-StandaloneSpirv-OpEntryPoint-08721
  Each `OpEntryPoint` **must** not have more than one `Input` variable assigned the same `Component` word inside a `Location` slot, either explicitly or implicitly
- VUID-StandaloneSpirv-OpEntryPoint-08722
  Each `OpEntryPoint` **must** not have more than one `Output` variable assigned the same `Component` word inside a `Location` slot, either explicitly or implicitly
- VUID-StandaloneSpirv-Result-04780
  The `Result Type` operand of any `OpImageRead` or `OpImageSparseRead` instruction **must** be a vector of four components
- VUID-StandaloneSpirv-Base-04781
  The `Base` operand of any `OpBitCount`, `OpBitReverse`, `OpBitFieldInsert`, `OpBitFieldSExtract`, or `OpBitFieldUExtract` instruction **must** be a 32-bit integer scalar or a vector of 32-bit integers
- VUID-StandaloneSpirv-PushConstant-06675
  Any variable in the `PushConstant` or `StorageBuffer` storage class **must** be decorated as `Block`
- VUID-StandaloneSpirv-Uniform-06676
  Any variable in the `Uniform Storage Class` **must** be decorated as `Block` or `BufferBlock`
- VUID-StandaloneSpirv-UniformConstant-06677
  Any variable in the `UniformConstant`, `StorageBuffer`, or `Uniform Storage Class` **must** be decorated with `DescriptorSet` and `Binding`
- VUID-StandaloneSpirv-InputAttachmentIndex-06678
  Variables decorated with `InputAttachmentIndex` **must** be in the `UniformConstant Storage Class`
- VUID-StandaloneSpirv-DescriptorSet-06491
  If a variable is decorated by `DescriptorSet` or `Binding`, the `Storage Class` **must** correspond to an entry in Shader Resource and Storage Class Correspondence
- VUID-StandaloneSpirv-Input-06778
  Variables with a `Storage Class` of `Input` in a fragment shader stage that are decorated with `PerVertexKHR` **must** be declared as arrays
- VUID-StandaloneSpirv-MeshEXT-07102
  The module **must** not contain both an entry point that uses the `TaskEXT` or `MeshEXT Execution Model` and an entry point that uses the `TaskNV` or `MeshNV Execution Model`
- VUID-StandaloneSpirv-MeshEXT-07106

In mesh shaders using the `MeshEXT Execution Model` `OpSetMeshOutputsEXT` **must** be called before any outputs are written

- VUID-StandaloneSpirv-MeshEXT-07107
  In mesh shaders using the `MeshEXT Execution Model` all variables declared as output **must** not be read from
- VUID-StandaloneSpirv-MeshEXT-07108
  In mesh shaders using the `MeshEXT Execution Model` for `OpSetMeshOutputsEXT` instructions, the "Vertex Count" and "Primitive Count" operands **must** not depend on `ViewIndex`
- VUID-StandaloneSpirv-MeshEXT-07109
  In mesh shaders using the `MeshEXT Execution Model` variables decorated with `PrimitivePointIndicesEXT`, `PrimitiveLineIndicesEXT`, or `PrimitiveTriangleIndicesEXT` declared as an array **must** not be accessed by indices that depend on `ViewIndex`
- VUID-StandaloneSpirv-MeshEXT-07110
  In mesh shaders using the `MeshEXT Execution Model` any values stored in variables decorated with `PrimitivePointIndicesEXT`, `PrimitiveLineIndicesEXT`, or `PrimitiveTriangleIndicesEXT` **must** not depend on `ViewIndex`
- VUID-StandaloneSpirv-MeshEXT-07111
  In mesh shaders using the `MeshEXT Execution Model` variables in workgroup or private `Storage Class` declared as or containing a composite type **must** not be accessed by indices that depend on `ViewIndex`
- VUID-StandaloneSpirv-MeshEXT-07330
  In mesh shaders using the `MeshEXT Execution Model` the `OutputVertices Execution Mode` **must** be greater than 0
- VUID-StandaloneSpirv-MeshEXT-07331
  In mesh shaders using the `MeshEXT Execution Model` the `OutputPrimitivesEXT Execution Mode` **must** be greater than 0
- VUID-StandaloneSpirv-Input-07290
  Variables with a `Storage Class` of `Input` or `Output` and a type of `OpTypeBool` **must** be decorated with the `BuiltIn` decoration
- VUID-StandaloneSpirv-TileImageEXT-08723
  The tile image variable declarations **must** obey the constraints on the `TileImageEXT Storage Class` and the `Location` decoration described in Fragment Tile Image Interface
- VUID-StandaloneSpirv-None-08724
  The `TileImageEXT Storage Class` **must** only be used for declaring tile image variables
- VUID-StandaloneSpirv-Pointer-08973
  The `Storage Class` of the `Pointer` operand to `OpCooperativeMatrixLoadKHR` or `OpCooperativeMatrixStoreKHR` **must** be limited to `Workgroup`, `StorageBuffer`, or `PhysicalStorageBuffer`
- VUID-StandaloneSpirv-UniformBufferArrayDynamicIndexing-10127
  If the `UniformBufferArrayDynamicIndexing` capability is not declared, and an instruction accesses memory through a uniform buffer, the uniform buffer through which that memory is accessed **must** be determined by constant integral expressions
- VUID-StandaloneSpirv-SampledImageArrayDynamicIndexing-10128

If the `SampledImageArrayDynamicIndexing` capability is not declared, and an instruction accesses memory through a sampled image or sampler, the sampled image or sampler through which that memory is accessed **must** be determined by constant integral expressions

- VUID-StandaloneSpirv-StorageBufferArrayDynamicIndexing-10129
  If the `StorageBufferArrayDynamicIndexing` capability is not declared, and an instruction accesses memory through a storage buffer, the storage buffer through which that memory is accessed **must** be determined by constant integral expressions
- VUID-StandaloneSpirv-StorageImageArrayDynamicIndexing-10130
  If the `StorageImageArrayDynamicIndexing` capability is not declared, and an instruction accesses memory through a storage image, the storage image through which that memory is accessed **must** be determined by constant integral expressions
- VUID-StandaloneSpirv-InputAttachmentArrayDynamicIndexing-10131
  If the `InputAttachmentArrayDynamicIndexing` capability is not declared, and an instruction accesses memory through an input attachment, the input attachmnet through which that memory is accessed **must** be determined by constant integral expressions
- VUID-StandaloneSpirv-UniformTexelBufferArrayDynamicIndexing-10132
  If the `UniformTexelBufferArrayDynamicIndexing` capability is declared, and an instruction accesses memory through a uniform texel buffer, the uniform texel buffer through which that memory is accessed **must** be determined by constant integral expressions
- VUID-StandaloneSpirv-StorageTexelBufferArrayDynamicIndexing-10133
  If the `StorageTexelBufferArrayDynamicIndexing` capability is declared, and an instruction accesses memory through a storage texel buffer, the storage texel buffer through which that memory is accessed **must** be determined by constant integral expressions

## Runtime SPIR-V Validation

The following rules **must** be validated at runtime. These rules depend on knowledge of the implementation and its capabilities and knowledge of runtime information, such as enabled features.

### Valid Usage

- VUID-RuntimeSpirv-vulkanMemoryModel-06265
  If `vulkanMemoryModel` is enabled and `vulkanMemoryModelDeviceScope` is not enabled, `Device` memory scope **must** not be used
- VUID-RuntimeSpirv-vulkanMemoryModel-06266
  If `vulkanMemoryModel` is not enabled, `QueueFamily` memory scope **must** not be used
- VUID-RuntimeSpirv-shaderSubgroupClock-06267
  If `shaderSubgroupClock` is not enabled, the `Subgroup` scope **must** not be used for `OpReadClockKHR`
- VUID-RuntimeSpirv-shaderDeviceClock-06268
  If `shaderDeviceClock` is not enabled, the `Device` scope **must** not be used for `OpReadClockKHR`

- VUID-RuntimeSpirv-None-09558
  If `dynamicRenderingLocalRead` is not enabled, any variable created with a “Type” of `OpTypeImage` that has a “Dim” operand of `SubpassData` **must** be decorated with `InputAttachmentIndex`
- VUID-RuntimeSpirv-OpTypeImage-09644
  Any variable declared as an `OpTypeArray` where the `Element Type` is an `OpTypeImage` with a “Dim” operand of `SubpassData` **must** be decorated with `InputAttachmentIndex`
- VUID-RuntimeSpirv-shaderStorageImageWriteWithoutFormat-07956
  If the VK_KHR_format_feature_flags2 extension is not enabled, and `shaderStorageImageWriteWithoutFormat` is not enabled, any variable created with a “Type” of `OpTypeImage` that has a “Sampled” operand of 2 and an “Image Format” operand of `Unknown` **must** be decorated with `NonWritable`
- VUID-RuntimeSpirv-shaderStorageImageReadWithoutFormat-07957
  If the VK_KHR_format_feature_flags2 extension is not enabled, and `shaderStorageImageReadWithoutFormat` is not enabled, any variable created with a “Type” of `OpTypeImage` that has a “Sampled” operand of 2 and an “Image Format” operand of `Unknown` **must** be decorated with `NonReadable`
- VUID-RuntimeSpirv-OpImageWrite-07112
  `OpImageWrite` to any `Image` whose `Image Format` is not `Unknown` **must** have the `Texel` operand contain at least as many components as the corresponding VkFormat as given in the SPIR-V Image Format compatibility table
- VUID-RuntimeSpirv-Location-06272
  The sum of `Location` and the number of locations the variable it decorates consumes **must** be less than or equal to the value for the matching `Execution Model` defined in Shader Input and Output Locations
- VUID-RuntimeSpirv-Location-06428
  The maximum number of storage buffers, storage images, and output `Location` decorated color attachments written to in the `Fragment Execution Model` **must** be less than or equal to `maxFragmentCombinedOutputResources`
- VUID-RuntimeSpirv-UniformBufferArrayNonUniformIndexing-10134
  If the `UniformBufferArrayNonUniformIndexing` capability is not declared, and an instruction accesses memory through a uniform buffer, the uniform buffer through which that memory is accessed **must** be dynamically uniform within the invocation group or subgroup
- VUID-RuntimeSpirv-SampledImageArrayNonUniformIndexing-10135
  If the `SampledImageArrayNonUniformIndexing` capability is not declared, and an instruction accesses memory through a sampled image or sampler, the sampled image or sampler through which that memory is accessed **must** be dynamically uniform within the invocation group or subgroup
- VUID-RuntimeSpirv-StorageBufferArrayNonUniformIndexing-10136
  If the `StorageBufferArrayNonUniformIndexing` capability is not declared, and an instruction accesses memory through a storage buffer, the storage buffer through which that memory is accessed **must** be dynamically uniform within the invocation group or subgroup
- VUID-RuntimeSpirv-StorageImageArrayNonUniformIndexing-10137

If the `StorageImageArrayNonUniformIndexing` capability is not declared, and an instruction accesses memory through a storage image, the storage image through which that memory is accessed **must** be dynamically uniform within the invocation group or subgroup

- VUID-RuntimeSpirv-InputAttachmentArrayNonUniformIndexing-10138
  If the `InputAttachmentArrayNonUniformIndexing` capability is not declared, and an instruction accesses memory through an input attachment, the input attachment through which that memory is accessed **must** be dynamically uniform within the invocation group or subgroup
- VUID-RuntimeSpirv-UniformTexelBufferArrayNonUniformIndexing-10139
  If the `UniformTexelBufferArrayNonUniformIndexing` capability is not declared, and an instruction accesses memory through a uniform texel buffer, the uniform texel buffer through which that memory is accessed **must** be dynamically uniform within the invocation group or subgroup
- VUID-RuntimeSpirv-StorageTexelBufferArrayNonUniformIndexing-10140
  If the `StorageTexelBufferArrayNonUniformIndexing` capability is not is not declared, and an instruction accesses memory through a storage texel buffer, the storage texel buffer through which that memory is accessed **must** be dynamically uniform within the invocation group or subgroup
- VUID-RuntimeSpirv-subgroupSize-10141
  If `subgroupSize` is 1, the `UniformBufferArrayNonUniformIndexing` capability is not declared, and an instruction accesses memory through a uniform buffer, the uniform buffer through which that memory is accessed **must** be dynamically uniform within the invocation group
- VUID-RuntimeSpirv-subgroupSize-10142
  If `subgroupSize` is 1, the `SampledImageArrayNonUniformIndexing` capability is not declared, and an instruction accesses memory through a sampled image or sampler, the sampled image or sampler through which that memory is accessed **must** be dynamically uniform within the invocation group
- VUID-RuntimeSpirv-subgroupSize-10143
  If `subgroupSize` is 1, the `StorageBufferArrayNonUniformIndexing` capability is not declared, and an instruction accesses memory through a storage buffer, the storage buffer through which that memory is accessed **must** be dynamically uniform within the invocation group
- VUID-RuntimeSpirv-subgroupSize-10144
  If `subgroupSize` is 1, the `StorageImageArrayNonUniformIndexing` capability is not declared, and an instruction accesses memory through a storage image, the storage image through which that memory is accessed **must** be dynamically uniform within the invocation group
- VUID-RuntimeSpirv-subgroupSize-10145
  If `subgroupSize` is 1, the `InputAttachmentArrayNonUniformIndexing` capability is not declared, and an instruction accesses memory through an input attachment, the input attachment through which that memory is accessed **must** be dynamically uniform within the invocation group
- VUID-RuntimeSpirv-subgroupSize-10146

If subgroupSize is 1, the UniformTexelBufferArrayNonUniformIndexing capability is not declared, and an instruction accesses memory through a uniform texel buffer, the uniform texel buffer through which that memory is accessed **must** be dynamically uniform within the invocation group

- VUID-RuntimeSpirv-subgroupSize-10147
  If subgroupSize is 1, the StorageTexelBufferArrayNonUniformIndexing capability is not is not declared, and an instruction accesses memory through a storage texel buffer, the storage texel buffer through which that memory is accessed **must** be dynamically uniform within the invocation group

- VUID-RuntimeSpirv-None-10148
  If an instruction accesses memory through any resource, subgroupSize is 1, and the resource through which that memory is accessed is not uniform within the invocation group, then the operand corresponding to that resource (e.g. the pointer or sampled image operand) **must** be decorated with NonUniform

- VUID-RuntimeSpirv-subgroupSize-10149
  If an instruction accesses memory through any resource, subgroupSize is greater than 1, and the resource through which that memory is accessed is not uniform within the invocation group, and not uniform within the subgroup, then the operand corresponding to that resource (e.g. the pointer or sampled image operand) **must** be decorated with NonUniform

- VUID-RuntimeSpirv-None-06275
  shaderSubgroupExtendedTypes **must** be enabled for group operations to use 8-bit integer, 16-bit integer, 64-bit integer, 16-bit floating-point, and vectors of these types

- VUID-RuntimeSpirv-None-06278
  shaderBufferInt64Atomics **must** be enabled for 64-bit integer atomic operations to be supported on a *Pointer* with a Storage Class of StorageBuffer or Uniform

- VUID-RuntimeSpirv-None-06279
  shaderSharedInt64Atomics **must** be enabled for 64-bit integer atomic operations to be supported on a *Pointer* with a Storage Class of Workgroup

- VUID-RuntimeSpirv-None-06284
  shaderBufferFloat32Atomics, or shaderBufferFloat32AtomicAdd, or shaderBufferFloat64Atomics, or shaderBufferFloat64AtomicAdd, or shaderBufferFloat16Atomics, or shaderBufferFloat16AtomicAdd, or shaderBufferFloat16AtomicMinMax, or shaderBufferFloat32AtomicMinMax, or shaderBufferFloat64AtomicMinMax, or shaderFloat16VectorAtomics **must** be enabled for floating-point atomic operations to be supported on a *Pointer* with a Storage Class of StorageBuffer

- VUID-RuntimeSpirv-None-06285
  shaderSharedFloat32Atomics, or shaderSharedFloat32AtomicAdd, or shaderSharedFloat64Atomics, or shaderSharedFloat64AtomicAdd, or shaderSharedFloat16Atomics, or shaderSharedFloat16AtomicAdd, or shaderSharedFloat16AtomicMinMax, or shaderSharedFloat32AtomicMinMax, or shaderSharedFloat64AtomicMinMax, or shaderFloat16VectorAtomics, **must** be enabled for floating-point atomic operations to be supported on a *Pointer* with a Storage Class of Workgroup

- VUID-RuntimeSpirv-None-06286
  `shaderImageFloat32Atomics`, or `shaderImageFloat32AtomicAdd`, or `shaderImageFloat32AtomicMinMax`, **must** be enabled for 32-bit floating-point atomic operations to be supported on a *Pointer* with a `Storage Class` of `Image`
- VUID-RuntimeSpirv-None-06287
  `sparseImageFloat32Atomics`, or `sparseImageFloat32AtomicAdd`, or `sparseImageFloat32AtomicMinMax`, **must** be enabled for 32-bit floating-point atomics to be supported on sparse images
- VUID-RuntimeSpirv-None-06288
  `shaderImageInt64Atomics` **must** be enabled for 64-bit integer atomic operations to be supported on a *Pointer* with a `Storage Class` of `Image`
- VUID-RuntimeSpirv-denormBehaviorIndependence-06289
  If `denormBehaviorIndependence` is `VK_SHADER_FLOAT_CONTROLS_INDEPENDENCE_32_BIT_ONLY`, then the entry point **must** use the same denormals `Execution Mode` for both 16-bit and 64-bit floating-point types
- VUID-RuntimeSpirv-denormBehaviorIndependence-06290
  If `denormBehaviorIndependence` is `VK_SHADER_FLOAT_CONTROLS_INDEPENDENCE_NONE`, then the entry point **must** use the same denormals `Execution Mode` for all floating-point types
- VUID-RuntimeSpirv-roundingModeIndependence-06291
  If `roundingModeIndependence` is `VK_SHADER_FLOAT_CONTROLS_INDEPENDENCE_32_BIT_ONLY`, then the entry point **must** use the same rounding `Execution Mode` for both 16-bit and 64-bit floating-point types
- VUID-RuntimeSpirv-roundingModeIndependence-06292
  If `roundingModeIndependence` is `VK_SHADER_FLOAT_CONTROLS_INDEPENDENCE_NONE`, then the entry point **must** use the same rounding `Execution Mode` for all floating-point types
- VUID-RuntimeSpirv-shaderSignedZeroInfNanPreserveFloat16-06293
  If `shaderSignedZeroInfNanPreserveFloat16` is `VK_FALSE`, then `SignedZeroInfNanPreserve` for 16-bit floating-point type **must** not be used
- VUID-RuntimeSpirv-shaderSignedZeroInfNanPreserveFloat32-06294
  If `shaderSignedZeroInfNanPreserveFloat32` is `VK_FALSE`, then `SignedZeroInfNanPreserve` for 32-bit floating-point type **must** not be used
- VUID-RuntimeSpirv-shaderSignedZeroInfNanPreserveFloat64-06295
  If `shaderSignedZeroInfNanPreserveFloat64` is `VK_FALSE`, then `SignedZeroInfNanPreserve` for 64-bit floating-point type **must** not be used
- VUID-RuntimeSpirv-shaderDenormPreserveFloat16-06296
  If `shaderDenormPreserveFloat16` is `VK_FALSE`, then `DenormPreserve` for 16-bit floating-point type **must** not be used
- VUID-RuntimeSpirv-shaderDenormPreserveFloat32-06297
  If `shaderDenormPreserveFloat32` is `VK_FALSE`, then `DenormPreserve` for 32-bit floating-point type **must** not be used
- VUID-RuntimeSpirv-shaderDenormPreserveFloat64-06298
  If `shaderDenormPreserveFloat64` is `VK_FALSE`, then `DenormPreserve` for 64-bit floating-point type **must** not be used

- VUID-RuntimeSpirv-shaderDenormFlushToZeroFloat16-06299
  If `shaderDenormFlushToZeroFloat16` is `VK_FALSE`, then `DenormFlushToZero` for 16-bit floating-point type **must** not be used
- VUID-RuntimeSpirv-shaderDenormFlushToZeroFloat32-06300
  If `shaderDenormFlushToZeroFloat32` is `VK_FALSE`, then `DenormFlushToZero` for 32-bit floating-point type **must** not be used
- VUID-RuntimeSpirv-shaderDenormFlushToZeroFloat64-06301
  If `shaderDenormFlushToZeroFloat64` is `VK_FALSE`, then `DenormFlushToZero` for 64-bit floating-point type **must** not be used
- VUID-RuntimeSpirv-shaderRoundingModeRTEFloat16-06302
  If `shaderRoundingModeRTEFloat16` is `VK_FALSE`, then `RoundingModeRTE` for 16-bit floating-point type **must** not be used
- VUID-RuntimeSpirv-shaderRoundingModeRTEFloat32-06303
  If `shaderRoundingModeRTEFloat32` is `VK_FALSE`, then `RoundingModeRTE` for 32-bit floating-point type **must** not be used
- VUID-RuntimeSpirv-shaderRoundingModeRTEFloat64-06304
  If `shaderRoundingModeRTEFloat64` is `VK_FALSE`, then `RoundingModeRTE` for 64-bit floating-point type **must** not be used
- VUID-RuntimeSpirv-shaderRoundingModeRTZFloat16-06305
  If `shaderRoundingModeRTZFloat16` is `VK_FALSE`, then `RoundingModeRTZ` for 16-bit floating-point type **must** not be used
- VUID-RuntimeSpirv-shaderRoundingModeRTZFloat32-06306
  If `shaderRoundingModeRTZFloat32` is `VK_FALSE`, then `RoundingModeRTZ` for 32-bit floating-point type **must** not be used
- VUID-RuntimeSpirv-shaderRoundingModeRTZFloat64-06307
  If `shaderRoundingModeRTZFloat64` is `VK_FALSE`, then `RoundingModeRTZ` for 64-bit floating-point type **must** not be used
- VUID-RuntimeSpirv-shaderSignedZeroInfNanPreserveFloat16-09559
  If `shaderSignedZeroInfNanPreserveFloat16` is `VK_FALSE` then any `FPFastMathDefault` execution mode with a type of 16-bit float **must** include the `NSZ`, `NotInf`, and `NotNaN` flags
- VUID-RuntimeSpirv-shaderSignedZeroInfNanPreserveFloat16-09560
  If `shaderSignedZeroInfNanPreserveFloat16` is `VK_FALSE` then any `FPFastMathMode` decoration on an instruction with result type or any operand type that includes a 16-bit float **must** include the `NSZ`, `NotInf`, and `NotNaN` flags
- VUID-RuntimeSpirv-shaderSignedZeroInfNanPreserveFloat32-09561
  If `shaderSignedZeroInfNanPreserveFloat32` is `VK_FALSE` then any `FPFastMathDefault` execution mode with a type of 32-bit float **must** include the `NSZ`, `NotInf`, and `NotNaN` flags
- VUID-RuntimeSpirv-shaderSignedZeroInfNanPreserveFloat32-09562
  If `shaderSignedZeroInfNanPreserveFloat32` is `VK_FALSE` then any `FPFastMathMode` decoration on an instruction with result type or any operand type that includes a 32-bit float **must** include the `NSZ`, `NotInf`, and `NotNaN` flags
- VUID-RuntimeSpirv-shaderSignedZeroInfNanPreserveFloat64-09563
  If `shaderSignedZeroInfNanPreserveFloat64` is `VK_FALSE` then any `FPFastMathDefault` execution

mode with a type of 64-bit float **must** include the NSZ, NotInf, and NotNaN flags

- VUID-RuntimeSpirv-shaderSignedZeroInfNanPreserveFloat64-09564
  If shaderSignedZeroInfNanPreserveFloat64 is VK_FALSE then any FPFastMathMode decoration on an instruction with result type or any operand type that includes a 64-bit float **must** include the NSZ, NotInf, and NotNaN flags
- VUID-RuntimeSpirv-Offset-06308
  The Offset plus size of the type of each variable, in the output interface of the entry point being compiled, decorated with XfbBuffer **must** not be greater than VkPhysicalDeviceTransformFeedbackPropertiesEXT::maxTransformFeedbackBufferDataSize
- VUID-RuntimeSpirv-XfbBuffer-06309
  For any given XfbBuffer value, define the buffer data size to be smallest number of bytes such that, for all outputs decorated with the same XfbBuffer value, the size of the output interface variable plus the Offset is less than or equal to the buffer data size. For a given Stream, the sum of all the buffer data sizes for all buffers writing to that stream the **must** not exceed VkPhysicalDeviceTransformFeedbackPropertiesEXT::maxTransformFeedbackStreamDataSize
- VUID-RuntimeSpirv-OpEmitStreamVertex-06310
  The Stream value to OpEmitStreamVertex and OpEndStreamPrimitive **must** be less than VkPhysicalDeviceTransformFeedbackPropertiesEXT::maxTransformFeedbackStreams
- VUID-RuntimeSpirv-transformFeedbackStreamsLinesTriangles-06311
  If the geometry shader emits to more than one vertex stream and VkPhysicalDeviceTransformFeedbackPropertiesEXT::transformFeedbackStreamsLinesTriangles is VK_FALSE, then Execution Mode **must** be OutputPoints
- VUID-RuntimeSpirv-Stream-06312
  The stream number value to Stream **must** be less than VkPhysicalDeviceTransformFeedbackPropertiesEXT::maxTransformFeedbackStreams
- VUID-RuntimeSpirv-XfbStride-06313
  The XFB Stride value to XfbStride **must** be less than or equal to VkPhysicalDeviceTransformFeedbackPropertiesEXT::maxTransformFeedbackBufferDataStride
- VUID-RuntimeSpirv-PhysicalStorageBuffer64-06314
  If the PhysicalStorageBuffer64 addressing model is enabled any load or store through a physical pointer type **must** be aligned to a multiple of the size of the largest scalar type in the pointed-to type
- VUID-RuntimeSpirv-PhysicalStorageBuffer64-06315
  If the PhysicalStorageBuffer64 addressing model is enabled the pointer value of a memory access instruction **must** be at least as aligned as specified by the Aligned memory access operand
- VUID-RuntimeSpirv-OpTypeCooperativeMatrixNV-06316
  For OpTypeCooperativeMatrixNV, the component type, scope, number of rows, and number of columns **must** match one of the matrices in any of the supported VkCooperativeMatrixPropertiesNV
- VUID-RuntimeSpirv-OpTypeCooperativeMatrixMulAddNV-10059

For OpTypeCooperativeMatrixMulAddNV, the operands **must** match a supported VkCooperativeMatrixPropertiesNV, such that:

- The type of A **must** have Rows match VkCooperativeMatrixPropertiesNV::MSize, Columns match VkCooperativeMatrixPropertiesNV::KSize, and ComponentType match VkCooperativeMatrixPropertiesNV::AType.
- The type of B **must** have Rows match VkCooperativeMatrixPropertiesNV::KSize, Columns match VkCooperativeMatrixPropertiesNV::NSize, and ComponentType match VkCooperativeMatrixPropertiesNV::BType.
- The type of C **must** have Rows match VkCooperativeMatrixPropertiesNV::MSize, Columns match VkCooperativeMatrixPropertiesNV::NSize, and ComponentType match VkCooperativeMatrixPropertiesNV::CType.
- The type of Result **must** have Rows match VkCooperativeMatrixPropertiesNV::MSize, Columns match VkCooperativeMatrixPropertiesNV::NSize, and ComponentType match VkCooperativeMatrixPropertiesNV::DType.
- The scope of all cooperative matrix operands **must** be VkScopeNV::VK_SCOPE_SUBGROUP_NV.
- If ComponentType of A, B, C, or Result is a signed integral type, the Signedness operand of the OpTypeInt must be 1.
- If ComponentType of A, B, C, or Result is an unsigned integral type, the Signedness operand of the OpTypeInt must be 0.

- VUID-RuntimeSpirv-OpTypeCooperativeMatrixNV-06322
  OpTypeCooperativeMatrixNV and OpCooperativeMatrix* instructions **must** not be used in shader stages not included in VkPhysicalDeviceCooperativeMatrixPropertiesNV::cooperativeMatrixSupportedStages
- VUID-RuntimeSpirv-OpTypeCooperativeMatrixKHR-08974
  For OpTypeCooperativeMatrixKHR, the component type, scope, number of rows, and number of columns **must** match one of the matrices in any of the supported VkCooperativeMatrixPropertiesKHR.
- VUID-RuntimeSpirv-OpCooperativeMatrixMulAddKHR-10060
  For OpCooperativeMatrixMulAddKHR, the operands **must** match a supported VkCooperativeMatrixPropertiesKHR, such that:
  - The type of A **must** have Rows match VkCooperativeMatrixPropertiesKHR::MSize, Columns match VkCooperativeMatrixPropertiesKHR::KSize, Use be MatrixAKHR, and ComponentType match VkCooperativeMatrixPropertiesKHR::AType.
  - The type of B **must** have Rows match VkCooperativeMatrixPropertiesKHR::KSize, Columns match VkCooperativeMatrixPropertiesKHR::NSize, Use be MatrixBKHR, and ComponentType match VkCooperativeMatrixPropertiesKHR::BType.
  - The type of C **must** have Rows match VkCooperativeMatrixPropertiesKHR::MSize, Columns match VkCooperativeMatrixPropertiesKHR::NSize, Use be MatrixAccumulatorKHR, and ComponentType match VkCooperativeMatrixPropertiesKHR::CType.
  - The type of Result **must** have Rows match VkCooperativeMatrixPropertiesKHR::MSize,

Columns match VkCooperativeMatrixPropertiesKHR::NSize, Use be MatrixAccumulatorKHR, and ComponentType match VkCooperativeMatrixPropertiesKHR::ResultType.

  - If and only if VkCooperativeMatrixPropertiesKHR::AType is a signed integer type, MatrixASignedComponents **must** be used.
  - If and only if VkCooperativeMatrixPropertiesKHR::BType is a signed integer type, MatrixBSignedComponents **must** be used.
  - If and only if VkCooperativeMatrixPropertiesKHR::CType is a signed integer type, MatrixCSignedComponents **must** be used.
  - If and only if VkCooperativeMatrixPropertiesKHR::ResultType is a signed integer type, MatrixResultSignedComponents **must** be used.
  - If and only if VkCooperativeMatrixPropertiesKHR::saturatingAccumulation is VK_TRUE, SaturatingAccumulationKHR **must** be used.
  - If and only if VkCooperativeMatrixPropertiesKHR::saturatingAccumulation is VK_FALSE, SaturatingAccumulationKHR **must** not be used.
  - The scope of all cooperative matrix operands **must** be VkScopeKHR::VK_SCOPE_SUBGROUP_KHR.

- VUID-RuntimeSpirv-cooperativeMatrixSupportedStages-08985
  OpTypeCooperativeMatrixKHR and OpCooperativeMatrix* instructions **must** not be used in shader stages not included in VkPhysicalDeviceCooperativeMatrixPropertiesKHR::cooperativeMatrixSupportedStages
- VUID-RuntimeSpirv-DescriptorSet-06323
  DescriptorSet and Binding decorations **must** obey the constraints on Storage Class, type, and descriptor type described in DescriptorSet and Binding Assignment
- VUID-RuntimeSpirv-OpCooperativeMatrixLoadNV-06324
  For OpCooperativeMatrixLoadNV and OpCooperativeMatrixStoreNV instructions, the Pointer and Stride operands **must** be aligned to at least the lesser of 16 bytes or the natural alignment of a row or column (depending on ColumnMajor) of the matrix (where the natural alignment is the number of columns/rows multiplied by the component size)
- VUID-RuntimeSpirv-MeshNV-07113
  For mesh shaders using the MeshNV Execution Model the OutputVertices OpExecutionMode **must** be less than or equal to VkPhysicalDeviceMeshShaderPropertiesNV::maxMeshOutputVertices
- VUID-RuntimeSpirv-MeshNV-07114
  For mesh shaders using the MeshNV Execution Model the OutputPrimitivesNV OpExecutionMode **must** be less than or equal to VkPhysicalDeviceMeshShaderPropertiesNV::maxMeshOutputPrimitives
- VUID-RuntimeSpirv-MeshEXT-07115
  For mesh shaders using the MeshEXT Execution Model the OutputVertices OpExecutionMode **must** be less than or equal to VkPhysicalDeviceMeshShaderPropertiesEXT::maxMeshOutputVertices
- VUID-RuntimeSpirv-MeshEXT-07332

For mesh shaders using the `MeshEXT` `Execution` `Model` the “Vertex Count” operand of `OpSetMeshOutputsEXT` **must** be less than or equal to `OutputVertices` `OpExecutionMode`

- VUID-RuntimeSpirv-MeshEXT-07116
  For mesh shaders using the `MeshEXT` `Execution` `Model` the `OutputPrimitivesEXT` `OpExecutionMode` **must** be less than or equal to VkPhysicalDeviceMeshShaderPropertiesEXT::`maxMeshOutputPrimitives`
- VUID-RuntimeSpirv-MeshEXT-07333
  For mesh shaders using the `MeshEXT` `Execution` `Model` the “Primitive Count” operand of `OpSetMeshOutputsEXT` **must** be less than or equal to `OutputPrimitivesEXT` `OpExecutionMode`
- VUID-RuntimeSpirv-TaskEXT-07117
  In task shaders using the `TaskEXT` `Execution` `Model` `OpEmitMeshTasksEXT` **must** be called exactly once under dynamically uniform conditions
- VUID-RuntimeSpirv-MeshEXT-07118
  In mesh shaders using the `MeshEXT` `Execution` `Model` `OpSetMeshOutputsEXT` **must** be called at most once under dynamically uniform conditions
- VUID-RuntimeSpirv-TaskEXT-07291
  In task shaders using the `TaskEXT` `Execution` `Model` the `x` size in `LocalSize` or `LocalSizeId` **must** be less than or equal to VkPhysicalDeviceMeshShaderPropertiesEXT::`maxTaskWorkGroupSize`[0]
- VUID-RuntimeSpirv-TaskEXT-07292
  In task shaders using the `TaskEXT` `Execution` `Model` the `y` size in `LocalSize` or `LocalSizeId` **must** be less than or equal to VkPhysicalDeviceMeshShaderPropertiesEXT::`maxTaskWorkGroupSize`[1]
- VUID-RuntimeSpirv-TaskEXT-07293
  In task shaders using the `TaskEXT` `Execution` `Model` the `z` size in `LocalSize` or `LocalSizeId` **must** be less than or equal to VkPhysicalDeviceMeshShaderPropertiesEXT::`maxTaskWorkGroupSize`[2]
- VUID-RuntimeSpirv-TaskEXT-07294
  In task shaders using the `TaskEXT` `Execution` `Model` the product of `x` size, `y` size, and `z` size in `LocalSize` or `LocalSizeId` **must** be less than or equal to VkPhysicalDeviceMeshShaderPropertiesEXT::`maxTaskWorkGroupInvocations`
- VUID-RuntimeSpirv-MeshEXT-07295
  For mesh shaders using the `MeshEXT` `Execution` `Model` the `x` size in `LocalSize` or `LocalSizeId` **must** be less than or equal to VkPhysicalDeviceMeshShaderPropertiesEXT::`maxMeshWorkGroupSize`[0]
- VUID-RuntimeSpirv-MeshEXT-07296
  For mesh shaders using the `MeshEXT` `Execution` `Model` the `y` size in `LocalSize` or `LocalSizeId` **must** be less than or equal to VkPhysicalDeviceMeshShaderPropertiesEXT::`maxMeshWorkGroupSize`[1]
- VUID-RuntimeSpirv-MeshEXT-07297
  For mesh shaders using the `MeshEXT` `Execution` `Model` the `z` size in `LocalSize` or `LocalSizeId` **must** be less than or equal to VkPhysicalDeviceMeshShaderPropertiesEXT::`maxMeshWorkGroupSize`[2]

- VUID-RuntimeSpirv-MeshEXT-07298
  For mesh shaders using the `MeshEXT Execution Model` the product of `x` size, `y` size, and `z` size in `LocalSize` or `LocalSizeId` **must** be less than or equal to VkPhysicalDeviceMeshShaderPropertiesEXT::`maxMeshWorkGroupInvocations`
- VUID-RuntimeSpirv-TaskEXT-07299
  In task shaders using the `TaskEXT Execution Model` the value of the "Group Count X" operand of `OpEmitMeshTasksEXT` **must** be less than or equal to VkPhysicalDeviceMeshShaderPropertiesEXT::`maxMeshWorkGroupCount`[0]
- VUID-RuntimeSpirv-TaskEXT-07300
  In task shaders using the `TaskEXT Execution Model` the value of the "Group Count Y" operand of `OpEmitMeshTasksEXT` **must** be less than or equal to VkPhysicalDeviceMeshShaderPropertiesEXT::`maxMeshWorkGroupCount`[1]
- VUID-RuntimeSpirv-TaskEXT-07301
  In task shaders using the `TaskEXT Execution Model` the value of the "Group Count Z" operand of `OpEmitMeshTasksEXT` **must** be less than or equal to VkPhysicalDeviceMeshShaderPropertiesEXT::`maxMeshWorkGroupCount`[2]
- VUID-RuntimeSpirv-TaskEXT-07302
  In task shaders using the `TaskEXT Execution Model` the product of the "Group Count" operands of `OpEmitMeshTasksEXT` **must** be less than or equal to VkPhysicalDeviceMeshShaderPropertiesEXT::`maxMeshWorkGroupTotalCount`
- VUID-RuntimeSpirv-maxMeshSharedMemorySize-08754
  The sum of size in bytes for variables and padding in the `Workgroup Storage Class` in the `MeshEXT Execution Model` **must** be less than or equal to `maxMeshSharedMemorySize`
- VUID-RuntimeSpirv-maxMeshPayloadAndSharedMemorySize-08755
  The sum of size in bytes for variables and padding in the `TaskPayloadWorkgroupEXT` or `Workgroup Storage Class` in the `MeshEXT Execution Model` **must** be less than or equal to `maxMeshPayloadAndSharedMemorySize`
- VUID-RuntimeSpirv-maxMeshOutputMemorySize-08756
  The sum of size in bytes for variables in the `Output Storage Class` in the `MeshEXT Execution Model` **must** be less than or equal to `maxMeshOutputMemorySize` according to the formula in Mesh Shader Output
- VUID-RuntimeSpirv-maxMeshPayloadAndOutputMemorySize-08757
  The sum of size in bytes for variables and in the `TaskPayloadWorkgroupEXT` or `Output Storage Class` in the `MeshEXT Execution Model` **must** be less than or equal to `maxMeshPayloadAndOutputMemorySize` according to the formula in Mesh Shader Output
- VUID-RuntimeSpirv-maxTaskPayloadSize-08758
  The sum of size in bytes for variables and in the `TaskPayloadWorkgroupEXT Storage Class` in the `TaskEXT Execution Model` **must** be less than or equal to `maxTaskPayloadSize`
- VUID-RuntimeSpirv-maxTaskSharedMemorySize-08759
  The sum of size in bytes for variables and padding in the `Workgroup Storage Class` in the `TaskEXT Execution Model` **must** be less than or equal to `maxTaskSharedMemorySize`
- VUID-RuntimeSpirv-maxTaskPayloadAndSharedMemorySize-08760
  The sum of size in bytes for variables and padding in the `TaskPayloadWorkgroupEXT` or

Workgroup Storage Class in the TaskEXT Execution Model **must** be less than or equal to maxTaskPayloadAndSharedMemorySize

- VUID-RuntimeSpirv-OpCooperativeMatrixLoadKHR-08986
  For OpCooperativeMatrixLoadKHR and OpCooperativeMatrixStoreKHR instructions, the Pointer and Stride operands **must** be aligned to at least the lesser of 16 bytes or the natural alignment of a row or column (depending on ColumnMajor) of the matrix (where the natural alignment is the number of columns/rows multiplied by the component size)
- VUID-RuntimeSpirv-shaderSampleRateInterpolationFunctions-06325
  If the VK_KHR_portability_subset extension is enabled, and VkPhysicalDevicePortabilitySubsetFeaturesKHR::shaderSampleRateInterpolationFunctions is VK_FALSE, then GLSL.std.450 fragment interpolation functions are not supported by the implementation and OpCapability **must** not be set to InterpolationFunction
- VUID-RuntimeSpirv-tessellationShader-06326
  If tessellationShader is enabled, and the VK_KHR_portability_subset extension is enabled, and VkPhysicalDevicePortabilitySubsetFeaturesKHR::tessellationIsolines is VK_FALSE, then OpExecutionMode **must** not be set to IsoLines
- VUID-RuntimeSpirv-tessellationShader-06327
  If tessellationShader is enabled, and the VK_KHR_portability_subset extension is enabled, and VkPhysicalDevicePortabilitySubsetFeaturesKHR::tessellationPointMode is VK_FALSE, then OpExecutionMode **must** not be set to PointMode
- VUID-RuntimeSpirv-storageBuffer8BitAccess-06328
  If storageBuffer8BitAccess is VK_FALSE, then objects containing an 8-bit integer element **must** not have Storage Class of StorageBuffer, ShaderRecordBufferKHR, or PhysicalStorageBuffer
- VUID-RuntimeSpirv-uniformAndStorageBuffer8BitAccess-06329
  If uniformAndStorageBuffer8BitAccess is VK_FALSE, then objects in the Uniform Storage Class with the Block decoration **must** not have an 8-bit integer member
- VUID-RuntimeSpirv-storagePushConstant8-06330
  If storagePushConstant8 is VK_FALSE, then objects containing an 8-bit integer element **must** not have Storage Class of PushConstant
- VUID-RuntimeSpirv-storageBuffer16BitAccess-06331
  If storageBuffer16BitAccess is VK_FALSE, then objects containing 16-bit integer or 16-bit floating-point elements **must** not have Storage Class of StorageBuffer, ShaderRecordBufferKHR, or PhysicalStorageBuffer
- VUID-RuntimeSpirv-uniformAndStorageBuffer16BitAccess-06332
  If uniformAndStorageBuffer16BitAccess is VK_FALSE, then objects in the Uniform Storage Class with the Block decoration **must** not have 16-bit integer or 16-bit floating-point members
- VUID-RuntimeSpirv-storagePushConstant16-06333
  If storagePushConstant16 is VK_FALSE, then objects containing 16-bit integer or 16-bit floating-point elements **must** not have Storage Class of PushConstant
- VUID-RuntimeSpirv-storageInputOutput16-06334
  If storageInputOutput16 is VK_FALSE, then objects containing 16-bit integer or 16-bit floating-point elements **must** not have Storage Class of Input or Output

- VUID-RuntimeSpirv-None-06337
  `shaderBufferFloat16Atomics`, or `shaderBufferFloat16AtomicAdd`, or `shaderBufferFloat16AtomicMinMax`, or `shaderSharedFloat16Atomics`, or `shaderSharedFloat16AtomicAdd`, or `shaderSharedFloat16AtomicMinMax` **must** be enabled for 16-bit floating-point atomic operations
- VUID-RuntimeSpirv-None-06338
  `shaderBufferFloat32Atomics`, or `shaderBufferFloat32AtomicAdd`, or `shaderSharedFloat32Atomics`, or `shaderSharedFloat32AtomicAdd`, or `shaderImageFloat32Atomics`, or `shaderImageFloat32AtomicAdd` or `shaderBufferFloat32AtomicMinMax`, or `shaderSharedFloat32AtomicMinMax`, or `shaderImageFloat32AtomicMinMax` **must** be enabled for 32-bit floating-point atomic operations
- VUID-RuntimeSpirv-None-06339
  `shaderBufferFloat64Atomics`, or `shaderBufferFloat64AtomicAdd`, or `shaderSharedFloat64Atomics`, or `shaderSharedFloat64AtomicAdd`, or `shaderBufferFloat64AtomicMinMax`, or `shaderSharedFloat64AtomicMinMax`, **must** be enabled for 64-bit floating-point atomic operations
- VUID-RuntimeSpirv-shaderFloat16VectorAtomics-09581
  `shaderFloat16VectorAtomics`, **must** be enabled for 16-bit floating-point, 2- and 4-component vector atomic operations to be supported
- VUID-RuntimeSpirv-NonWritable-06340
  If `fragmentStoresAndAtomics` is not enabled, then all storage image, storage texel buffer, and storage buffer variables in the fragment stage **must** be decorated with the `NonWritable` decoration
- VUID-RuntimeSpirv-NonWritable-06341
  If `vertexPipelineStoresAndAtomics` is not enabled, then all storage image, storage texel buffer, and storage buffer variables in the vertex, tessellation, and geometry stages **must** be decorated with the `NonWritable` decoration
- VUID-RuntimeSpirv-None-06342
  If `subgroupQuadOperationsInAllStages` is `VK_FALSE`, then quad subgroup operations **must** not be used except for in fragment and compute stages
- VUID-RuntimeSpirv-None-06343
  Group operations with subgroup scope **must** not be used if the shader stage is not in `subgroupSupportedStages`
- VUID-RuntimeSpirv-Offset-06344
  The first element of the `Offset` operand of `InterpolateAtOffset` **must** be greater than or equal to:
  $frag_{width} \times$ `minInterpolationOffset`
  where $frag_{width}$ is the width of the current fragment in pixels
- VUID-RuntimeSpirv-Offset-06345
  The first element of the `Offset` operand of `InterpolateAtOffset` **must** be less than or equal to
  $frag_{width} \times$ (`maxInterpolationOffset` + ULP ) - ULP
  where $frag_{width}$ is the width of the current fragment in pixels and ULP = 1 /

2^subPixelInterpolationOffsetBits^

- VUID-RuntimeSpirv-Offset-06346
  The second element of the Offset operand of InterpolateAtOffset **must** be greater than or equal to
  $frag_{height}$ × minInterpolationOffset
  where $frag_{height}$ is the height of the current fragment in pixels
- VUID-RuntimeSpirv-Offset-06347
  The second element of the Offset operand of InterpolateAtOffset **must** be less than or equal to
  $frag_{height}$ × (maxInterpolationOffset + ULP ) - ULP
  where $frag_{height}$ is the height of the current fragment in pixels and ULP = 1 / 2^subPixelInterpolationOffsetBits^
- VUID-RuntimeSpirv-OpRayQueryInitializeKHR-06348
  For OpRayQueryInitializeKHR instructions, all components of the RayOrigin and RayDirection operands **must** be finite floating-point values
- VUID-RuntimeSpirv-OpRayQueryInitializeKHR-06349
  For OpRayQueryInitializeKHR instructions, the RayTmin and RayTmax operands **must** be non-negative floating-point values
- VUID-RuntimeSpirv-OpRayQueryInitializeKHR-06350
  For OpRayQueryInitializeKHR instructions, the RayTmin operand **must** be less than or equal to the RayTmax operand
- VUID-RuntimeSpirv-OpRayQueryInitializeKHR-06351
  For OpRayQueryInitializeKHR instructions, RayOrigin, RayDirection, RayTmin, and RayTmax operands **must** not contain NaNs
- VUID-RuntimeSpirv-OpRayQueryInitializeKHR-06352
  For OpRayQueryInitializeKHR instructions, Acceleration Structure **must** be an acceleration structure built as a top-level acceleration structure
- VUID-RuntimeSpirv-OpRayQueryInitializeKHR-06889
  For OpRayQueryInitializeKHR instructions, the Rayflags operand **must** not contain both SkipTrianglesKHR and SkipAABBsKHR
- VUID-RuntimeSpirv-OpRayQueryInitializeKHR-06890
  For OpRayQueryInitializeKHR instructions, the Rayflags operand **must** not contain more than one of SkipTrianglesKHR, CullBackFacingTrianglesKHR, and CullFrontFacingTrianglesKHR
- VUID-RuntimeSpirv-OpRayQueryInitializeKHR-06891
  For OpRayQueryInitializeKHR instructions, the Rayflags operand **must** not contain more than one of OpaqueKHR, NoOpaqueKHR, CullOpaqueKHR, and CullNoOpaqueKHR
- VUID-RuntimeSpirv-OpRayQueryGenerateIntersectionKHR-06353
  For OpRayQueryGenerateIntersectionKHR instructions, Hit T **must** satisfy the condition RayTmin ≤ Hit T ≤ RayTmax, where RayTmin is equal to the value returned by OpRayQueryGetRayTMinKHR with the same ray query object, and RayTmax is equal to the value of OpRayQueryGetIntersectionTKHR for the current committed intersection with the same ray query object

- VUID-RuntimeSpirv-OpRayQueryGenerateIntersectionKHR-06354
  For `OpRayQueryGenerateIntersectionKHR` instructions, `Acceleration Structure` **must** not be built with `VK_BUILD_ACCELERATION_STRUCTURE_MOTION_BIT_NV` in `flags`
- VUID-RuntimeSpirv-flags-08761
  For `OpRayQueryGetIntersectionTriangleVertexPositionsKHR` instructions, `Acceleration Structure` **must** have been built with `VK_BUILD_ACCELERATION_STRUCTURE_ALLOW_DATA_ACCESS_KHR` in `flags`
- VUID-RuntimeSpirv-OpTraceRayKHR-06355
  For `OpTraceRayKHR` instructions, all components of the `RayOrigin` and `RayDirection` operands **must** be finite floating-point values
- VUID-RuntimeSpirv-OpTraceRayKHR-06356
  For `OpTraceRayKHR` instructions, the `RayTmin` and `RayTmax` operands **must** be non-negative floating-point values
- VUID-RuntimeSpirv-OpTraceRayKHR-06552
  For `OpTraceRayKHR` instructions, the `Rayflags` operand **must** not contain both `SkipTrianglesKHR` and `SkipAABBsKHR`
- VUID-RuntimeSpirv-OpTraceRayKHR-06892
  For `OpTraceRayKHR` instructions, the `Rayflags` operand **must** not contain more than one of `SkipTrianglesKHR`, `CullBackFacingTrianglesKHR`, and `CullFrontFacingTrianglesKHR`
- VUID-RuntimeSpirv-OpTraceRayKHR-06893
  For `OpTraceRayKHR` instructions, the `Rayflags` operand **must** not contain more than one of `OpaqueKHR`, `NoOpaqueKHR`, `CullOpaqueKHR`, and `CullNoOpaqueKHR`
- VUID-RuntimeSpirv-OpTraceRayKHR-06553
  For `OpTraceRayKHR` instructions, if the `Rayflags` operand contains `SkipTrianglesKHR`, the pipeline **must** not have been created with `VK_PIPELINE_CREATE_RAY_TRACING_SKIP_AABBS_BIT_KHR` set
- VUID-RuntimeSpirv-OpTraceRayKHR-06554
  For `OpTraceRayKHR` instructions, if the `Rayflags` operand contains `SkipAABBsKHR`, the pipeline **must** not have been created with `VK_PIPELINE_CREATE_RAY_TRACING_SKIP_TRIANGLES_BIT_KHR` set
- VUID-RuntimeSpirv-OpTraceRayKHR-06357
  For `OpTraceRayKHR` instructions, the `RayTmin` operand **must** be less than or equal to the `RayTmax` operand
- VUID-RuntimeSpirv-OpTraceRayKHR-06358
  For `OpTraceRayKHR` instructions, `RayOrigin`, `RayDirection`, `RayTmin`, and `RayTmax` operands **must** not contain NaNs
- VUID-RuntimeSpirv-OpTraceRayKHR-06359
  For `OpTraceRayKHR` instructions, `Acceleration Structure` **must** be an acceleration structure built as a top-level acceleration structure
- VUID-RuntimeSpirv-OpReportIntersectionKHR-06998
  The value of the “Hit Kind” operand of `OpReportIntersectionKHR` **must** be in the range [0,127]
- VUID-RuntimeSpirv-OpTraceRayKHR-06360

For OpTraceRayKHR instructions, if Acceleration Structure was built with VK_BUILD_ACCELERATION_STRUCTURE_MOTION_BIT_NV in flags, the pipeline **must** have been created with VK_PIPELINE_CREATE_RAY_TRACING_ALLOW_MOTION_BIT_NV set

- VUID-RuntimeSpirv-OpTraceRayMotionNV-06361
  For OpTraceRayMotionNV instructions, all components of the RayOrigin and RayDirection operands **must** be finite floating-point values
- VUID-RuntimeSpirv-OpTraceRayMotionNV-06362
  For OpTraceRayMotionNV instructions, the RayTmin and RayTmax operands **must** be non-negative floating-point values
- VUID-RuntimeSpirv-OpTraceRayMotionNV-06363
  For OpTraceRayMotionNV instructions, the RayTmin operand **must** be less than or equal to the RayTmax operand
- VUID-RuntimeSpirv-OpTraceRayMotionNV-06364
  For OpTraceRayMotionNV instructions, RayOrigin, RayDirection, RayTmin, and RayTmax operands **must** not contain NaNs
- VUID-RuntimeSpirv-OpTraceRayMotionNV-06365
  For OpTraceRayMotionNV instructions, Acceleration Structure **must** be an acceleration structure built as a top-level acceleration structure with VK_BUILD_ACCELERATION_STRUCTURE_MOTION_BIT_NV in flags
- VUID-RuntimeSpirv-OpTraceRayMotionNV-06366
  For OpTraceRayMotionNV instructions the time operand **must** be between 0.0 and 1.0
- VUID-RuntimeSpirv-OpTraceRayMotionNV-06367
  For OpTraceRayMotionNV instructions the pipeline **must** have been created with VK_PIPELINE_CREATE_RAY_TRACING_ALLOW_MOTION_BIT_NV set
- VUID-RuntimeSpirv-OpHitObjectTraceRayMotionNV-07704
  For OpHitObjectTraceRayMotionNV instructions, if Acceleration Structure was built with VK_BUILD_ACCELERATION_STRUCTURE_MOTION_BIT_NV in flags, the pipeline **must** have been created with VK_PIPELINE_CREATE_RAY_TRACING_ALLOW_MOTION_BIT_NV set
- VUID-RuntimeSpirv-OpHitObjectTraceRayNV-07705
  For OpHitObjectTraceRayNV and OpHitObjectTraceRayMotionNV instructions, all components of the RayOrigin and RayDirection operands **must** be finite floating-point values
- VUID-RuntimeSpirv-OpHitObjectTraceRayNV-07706
  For OpHitObjectTraceRayNV and OpHitObjectTraceRayMotionNV instructions, the RayTmin and RayTmax operands **must** be non-negative floating-point values
- VUID-RuntimeSpirv-OpHitObjectTraceRayNV-07707
  For OpHitObjectTraceRayNV and OpHitObjectTraceRayMotionNV instructions, the RayTmin operand **must** be less than or equal to the RayTmax operand
- VUID-RuntimeSpirv-OpHitObjectTraceRayNV-07708
  For OpHitObjectTraceRayNV and OpHitObjectTraceRayMotionNV instructions, RayOrigin, RayDirection, RayTmin, and RayTmax operands **must** not contain NaNs
- VUID-RuntimeSpirv-OpHitObjectTraceRayMotionNV-07709
  For OpHitObjectTraceRayMotionNV instructions, Acceleration Structure **must** be an acceleration structure built as a top-level acceleration structure with

VK_BUILD_ACCELERATION_STRUCTURE_MOTION_BIT_NV in flags

- VUID-RuntimeSpirv-OpHitObjectTraceRayNV-07710
  For OpHitObjectTraceRayNV and OpHitObjectTraceRayMotionNV instructions the time operand **must** be between 0.0 and 1.0
- VUID-RuntimeSpirv-OpHitObjectTraceRayMotionNV-07711
  For OpHitObjectTraceRayMotionNV instructions the pipeline **must** have been created with VK_PIPELINE_CREATE_RAY_TRACING_ALLOW_MOTION_BIT_NV set
- VUID-RuntimeSpirv-OpHitObjectTraceRayNV-07712
  For OpHitObjectTraceRayNV and OpHitObjectTraceRayMotionNV instructions, the Rayflags operand **must** not contain both SkipTrianglesKHR and SkipAABBsKHR
- VUID-RuntimeSpirv-OpHitObjectTraceRayNV-07713
  For OpHitObjectTraceRayNV and OpHitObjectTraceRayMotionNV instructions, the Rayflags operand **must** not contain more than one of SkipTrianglesKHR, CullBackFacingTrianglesKHR, and CullFrontFacingTrianglesKHR
- VUID-RuntimeSpirv-OpHitObjectTraceRayNV-07714
  For OpHitObjectTraceRayNV and OpHitObjectTraceRayMotionNV instructions, the Rayflags operand **must** not contain more than one of OpaqueKHR, NoOpaqueKHR, CullOpaqueKHR, and CullNoOpaqueKHR
- VUID-RuntimeSpirv-OpHitObjectTraceRayNV-07715
  For OpHitObjectTraceRayNV and OpHitObjectTraceRayMotionNV instructions, if the Rayflags operand contains SkipTrianglesKHR, the pipeline **must** not have been created with VK_PIPELINE_CREATE_RAY_TRACING_SKIP_AABBS_BIT_KHR set
- VUID-RuntimeSpirv-OpHitObjectTraceRayNV-07716
  For OpHitObjectTraceRayNV and OpHitObjectTraceRayMotionNV instructions, if the Rayflags operand contains SkipAABBsKHR, the pipeline **must** not have been created with VK_PIPELINE_CREATE_RAY_TRACING_SKIP_TRIANGLES_BIT_KHR set
- VUID-RuntimeSpirv-x-06429
  In compute shaders using the GLCompute Execution Model the x size in LocalSize or LocalSizeId **must** be less than or equal to VkPhysicalDeviceLimits::maxComputeWorkGroupSize[0]
- VUID-RuntimeSpirv-y-06430
  In compute shaders using the GLCompute Execution Model the y size in LocalSize or LocalSizeId **must** be less than or equal to VkPhysicalDeviceLimits::maxComputeWorkGroupSize[1]
- VUID-RuntimeSpirv-z-06431
  In compute shaders using the GLCompute Execution Model the z size in LocalSize or LocalSizeId **must** be less than or equal to VkPhysicalDeviceLimits::maxComputeWorkGroupSize[2]
- VUID-RuntimeSpirv-x-06432
  In compute shaders using the GLCompute Execution Model the product of x size, y size, and z size in LocalSize or LocalSizeId **must** be less than or equal to VkPhysicalDeviceLimits::maxComputeWorkGroupInvocations
- VUID-RuntimeSpirv-LocalSizeId-06434

If `Execution Mode LocalSizeId` is used, `maintenance4` **must** be enabled

- VUID-RuntimeSpirv-maintenance4-06817
  If `maintenance4` is not enabled, any `OpTypeVector` output interface variables **must** not have a higher `Component Count` than a matching `OpTypeVector` input interface variable
- VUID-RuntimeSpirv-OpEntryPoint-08743
  Any user-defined variables shared between the `OpEntryPoint` of two shader stages, and declared with `Input` as its `Storage Class` for the subsequent shader stage, **must** have all `Location` slots and `Component` words declared in the preceding shader stage's `OpEntryPoint` with `Output` as the `Storage Class`
- VUID-RuntimeSpirv-OpEntryPoint-07754
  Any user-defined variables between the `OpEntryPoint` of two shader stages **must** have the same type and width for each `Component`
- VUID-RuntimeSpirv-OpVariable-08746
  Any `OpVariable`, `Block`-decorated `OpTypeStruct`, or `Block`-decorated `OpTypeStruct` members shared between the `OpEntryPoint` of two shader stages **must** have matching decorations as defined in interface matching
- VUID-RuntimeSpirv-Workgroup-06530
  The sum of size in bytes for variables and padding in the `Workgroup Storage Class` in the `GLCompute Execution Model` **must** be less than or equal to `maxComputeSharedMemorySize`
- VUID-RuntimeSpirv-shaderZeroInitializeWorkgroupMemory-06372
  If `shaderZeroInitializeWorkgroupMemory` is not enabled, any `OpVariable` with `Workgroup` as its `Storage Class` **must** not have an `Initializer` operand
- VUID-RuntimeSpirv-OpImage-06376
  If an `OpImage*Gather` operation has an image operand of `Offset`, `ConstOffset`, or `ConstOffsets` the offset value **must** be greater than or equal to `minTexelGatherOffset`
- VUID-RuntimeSpirv-OpImage-06377
  If an `OpImage*Gather` operation has an image operand of `Offset`, `ConstOffset`, or `ConstOffsets` the offset value **must** be less than or equal to `maxTexelGatherOffset`
- VUID-RuntimeSpirv-OpImageSample-06435
  If an `OpImageSample*` or `OpImageFetch*` operation has an image operand of `ConstOffset` then the offset value **must** be greater than or equal to `minTexelOffset`
- VUID-RuntimeSpirv-OpImageSample-06436
  If an `OpImageSample*` or `OpImageFetch*` operation has an image operand of `ConstOffset` then the offset value **must** be less than or equal to `maxTexelOffset`
- VUID-RuntimeSpirv-samples-08725
  If an `OpTypeImage` has an `MS` operand 0, its bound image **must** have been created with VkImageCreateInfo::`samples` as `VK_SAMPLE_COUNT_1_BIT`
- VUID-RuntimeSpirv-samples-08726
  If an `OpTypeImage` has an `MS` operand 1, its bound image **must** not have been created with VkImageCreateInfo::`samples` as `VK_SAMPLE_COUNT_1_BIT`
- VUID-RuntimeSpirv-SampleRateShading-06378
  If the subpass description contains `VK_SUBPASS_DESCRIPTION_FRAGMENT_REGION_BIT_QCOM`, then the SPIR-V fragment shader Capability `SampleRateShading` **must** not be enabled

- VUID-RuntimeSpirv-SubgroupUniformControlFlowKHR-06379
  The `Execution Mode SubgroupUniformControlFlowKHR` **must** not be applied to an entry point unless `shaderSubgroupUniformControlFlow` is enabled, the corresponding shader stage bit is set in `subgroupSupportedStages`, and the entry point does not execute any *invocation repack instructions*
- VUID-RuntimeSpirv-shaderEarlyAndLateFragmentTests-06767
  If `shaderEarlyAndLateFragmentTests` is not enabled, the `EarlyAndLateFragmentTestsEXT Execution Mode` **must** not be used
- VUID-RuntimeSpirv-shaderEarlyAndLateFragmentTests-06768
  If `shaderEarlyAndLateFragmentTests` feature is not enabled, the `StencilRefUnchangedFrontEXT Execution Mode` **must** not be used
- VUID-RuntimeSpirv-shaderEarlyAndLateFragmentTests-06769
  If `shaderEarlyAndLateFragmentTests` is not enabled, the `StencilRefUnchangedBackEXT Execution Mode` **must** not be used
- VUID-RuntimeSpirv-shaderEarlyAndLateFragmentTests-06770
  If `shaderEarlyAndLateFragmentTests` is not enabled, the `StencilRefGreaterFrontEXT Execution Mode` **must** not be used
- VUID-RuntimeSpirv-shaderEarlyAndLateFragmentTests-06771
  If `shaderEarlyAndLateFragmentTests` is not enabled, the `StencilRefGreaterBackEXT Execution Mode` **must** not be used
- VUID-RuntimeSpirv-shaderEarlyAndLateFragmentTests-06772
  If `shaderEarlyAndLateFragmentTests` is not enabled, the `StencilRefLessFrontEXT Execution Mode` **must** not be used
- VUID-RuntimeSpirv-shaderEarlyAndLateFragmentTests-06773
  If `shaderEarlyAndLateFragmentTests` is not enabled, the `StencilRefLessBackEXT Execution Mode` **must** not be used
- VUID-RuntimeSpirv-OpImageWeightedSampleQCOM-06979
  If an `OpImageWeightedSampleQCOM` operation is used, then the `Texture Sampled Image` and `Weight Image` parameters **must** both be *dynamically uniform* for the quad
- VUID-RuntimeSpirv-OpImageWeightedSampleQCOM-06980
  If an `OpImageWeightedSampleQCOM` operation is used, then the `Weight Image` parameter **must** be of `Storage Class UniformConstant` and type `OpTypeImage` with `Depth`=0, `Dim=2D`, `Arrayed`=1, `MS`=0, and `Sampled`=1
- VUID-RuntimeSpirv-OpImageWeightedSampleQCOM-06981
  If an `OpImageWeightedSampleQCOM` operation is used, then the `Weight Image` parameter **must** be decorated with `WeightTextureQCOM`
- VUID-RuntimeSpirv-OpImageBlockMatchSADQCOM-06982
  If an `OpImageBlockMatchSADQCOM` or `OpImageBlockMatchSSDQCOM` operation is used, then the `target sampled image`, `reference sampled image`, and `Block Size` parameters **must** both be *dynamically uniform* for the quad
- VUID-RuntimeSpirv-OpImageBlockMatchSSDQCOM-06983
  If an `OpImageBlockMatchSSDQCOM` or `OpImageBlockMatchSADQCOM` operation is used, then `target sampled image` and `reference sampled image` parameters **must** be of storage class

UniformConstant and type OpTypeImage with Depth=0, Dim=2D, Arrayed=0, MS=0, and Sampled=1

- VUID-RuntimeSpirv-OpImageBlockMatchSSDQCOM-06984
  If an OpImageBlockMatchSSDQCOM or OpImageBlockMatchSADQCOM operation is used, then the target sampled image and reference sampled image parameters **must** be decorated with BlockMatchTextureQCOM
- VUID-RuntimeSpirv-OpImageBlockMatchSSDQCOM-06985
  If an OpImageBlockMatchSSDQCOM or OpImageBlockMatchSADQCOM operation is used, then target sampled image and reference sampled image parameters **must** have been created using an identical sampler object
- VUID-RuntimeSpirv-OpImageBlockMatchSSDQCOM-06986
  If an OpImageBlockMatchSSDQCOM or OpImageBlockMatchSADQCOM operation is used, then target sampled image and reference sampled image parameters **must** have been created with a sampler object with unnormalizedCoordinates equal to VK_TRUE
- VUID-RuntimeSpirv-OpImageBlockMatchSSDQCOM-06987
  If an OpImageBlockMatchSSDQCOM or OpImageBlockMatchSADQCOM operation is used, then target sampled image and reference sampled image parameters **must** have been created with a sampler object with unnormalizedCoordinates equal to VK_TRUE
- VUID-RuntimeSpirv-OpImageBlockMatchSSDQCOM-06988
  If an OpImageBlockMatchSSDQCOM or OpImageBlockMatchSADQCOM operation is used, then Block Size less than or equal to maxBlockMatchRegion
- VUID-RuntimeSpirv-OpImageBoxFilterQCOM-06989
  If an OpImageBoxFilterQCOM operation is used, then Box Size.y **must** be equal to or greater than 1.0 and less than or equal to maxBoxFilterBlockSize.height
- VUID-RuntimeSpirv-OpImageBoxFilterQCOM-06990
  If an OpImageBoxFilterQCOM operation is used, then Sampled Texture Image and Box Size parameters **must** be *dynamically uniform*
- VUID-RuntimeSpirv-OpEntryPoint-08727
  Each OpEntryPoint **must** not have more than one variable decorated with InputAttachmentIndex per image aspect of the attachment image bound to it, either explicitly or implicitly as described by input attachment interface
- VUID-RuntimeSpirv-shaderTileImageColorReadAccess-08728
  If shaderTileImageColorReadAccess is not enabled, OpColorAttachmentReadEXT operation **must** not be used
- VUID-RuntimeSpirv-shaderTileImageDepthReadAccess-08729
  If shaderTileImageDepthReadAccess is not enabled, OpDepthAttachmentReadEXT operation **must** not be used
- VUID-RuntimeSpirv-shaderTileImageStencilReadAccess-08730
  If shaderTileImageStencilReadAccess is not enabled, OpStencilAttachmentReadEXT operation **must** not be used
- VUID-RuntimeSpirv-minSampleShading-08731
  If sample shading is enabled and minSampleShading is 1.0, the sample operand of any OpColorAttachmentReadEXT, OpDepthAttachmentReadEXT, or OpStencilAttachmentReadEXT operation **must** evaluate to the value of the coverage index for any given fragment

invocation

- VUID-RuntimeSpirv-minSampleShading-08732
  If sample shading is enabled and any of the `OpColorAttachmentReadEXT`, `OpDepthAttachmentReadEXT`, or `OpStencilAttachmentReadEXT` operations are used, then `minSampleShading` **must** be 1.0
- VUID-RuntimeSpirv-MeshEXT-09218
  In mesh shaders using the `MeshEXT` or `MeshNV` `Execution Model` and the `OutputPoints` `Execution Mode`, if `maintenance5` is not enabled, and if the number of output points is greater than 0, a `PointSize` decorated variable **must** be written to for each output point
- VUID-RuntimeSpirv-maintenance5-09190
  If `maintenance5` is enabled and a `PointSize` decorated variable is written to, all execution paths **must** write to a `PointSize` decorated variable
- VUID-RuntimeSpirv-ShaderEnqueueAMDX-09191
  The `ShaderEnqueueAMDX` capability **must** only be used in shaders with the `GLCompute` execution model
- VUID-RuntimeSpirv-NodePayloadAMDX-09192
  Variables in the `NodePayloadAMDX` storage class **must** only be declared in the `GLCompute` execution model
- VUID-RuntimeSpirv-maxExecutionGraphShaderPayloadSize-09193
  Variables declared in the `NodePayloadAMDX` storage class **must** not be larger than the `maxExecutionGraphShaderPayloadSize` limit
- VUID-RuntimeSpirv-maxExecutionGraphShaderPayloadSize-09194
  Variables declared in the `NodeOutputPayloadAMDX` storage class **must** not be larger than the `maxExecutionGraphShaderPayloadSize` limit
- VUID-RuntimeSpirv-maxExecutionGraphShaderPayloadSize-09195
  For a given entry point, the sum of the size of any variable in the `NodePayloadAMDX` storage class, and the combined size of all statically initialized variables in the `NodeOutputPayloadAMDX` storage class **must** not be greater than `maxExecutionGraphShaderPayloadSize`
- VUID-RuntimeSpirv-maxExecutionGraphShaderPayloadCount-09196
  Shaders **must** not statically initialize more than `maxExecutionGraphShaderPayloadCount` variables in the `NodeOutputPayloadAMDX` storage class
- VUID-RuntimeSpirv-maxExecutionGraphShaderOutputNodes-09197
  Shaders **must** not include more than `maxExecutionGraphShaderOutputNodes` instances of `OpInitializeNodePayloadsAMDX`
- VUID-RuntimeSpirv-OpImageBlockMatchWindow-09219
  If a `OpImageBlockMatchWindow*QCOM` or `OpImageBlockMatchGather*QCOM` operation is used, then the `target sampled image`, `reference sampled image`, and `Block Size` parameters **must** both be *dynamically uniform* for the quad
- VUID-RuntimeSpirv-OpImageBlockMatchWindow-09220
  If a `OpImageBlockMatchWindow*QCOM` or `OpImageBlockMatchGather*QCOM` operation is used, then `target sampled image` and `reference sampled image` parameters **must** be of storage class `UniformConstant` and type `OpTypeImage` with `Depth`=0, `Dim`=`2D`, `Arrayed`=0, `MS`=0, and `Sampled`=1

- VUID-RuntimeSpirv-OpImageBlockMatchWindow-09221
  If a `OpImageBlockMatchWindow*QCOM` or `OpImageBlockMatchGather*QCOM` operation is used, then the `target sampled image` and `reference sampled image` parameters **must** be decorated with `BlockMatchTextureQCOM`
- VUID-RuntimeSpirv-OpImageBlockMatchWindow-09222
  If a `OpImageBlockMatchWindow*QCOM` or `OpImageBlockMatchGather*QCOM` operation is used, then `target sampled image` and `reference sampled image` parameters **must** have been created using an identical sampler object
- VUID-RuntimeSpirv-OpImageBlockMatchWindow-09223
  If a `OpImageBlockMatchWindow*QCOM` or `OpImageBlockMatchGather*QCOM` operation is used, then `target sampled image` and `reference sampled image` parameters **must** have been created with a sampler object with `unnormalizedCoordinates` equal to `VK_TRUE`
- VUID-RuntimeSpirv-OpImageBlockMatchWindow-09224
  If a `OpImageBlockMatchWindow*QCOM` or `OpImageBlockMatchGather*QCOM` operation is used, then `target sampled image` and `reference sampled image` parameters **must** have been created with sampler object with `unnormalizedCoordinates` equal to `VK_TRUE`
- VUID-RuntimeSpirv-maxBlockMatchRegion-09225
  If a `OpImageBlockMatchWindow*QCOM` or `OpImageBlockMatchGather*QCOM` operation is used, then `Block Size` less than or equal to `maxBlockMatchRegion`
- VUID-RuntimeSpirv-pNext-09226
  If a `OpImageBlockMatchWindow*QCOM` operation is used, then `target sampled image` **must** have been created using asampler object that included VkSamplerBlockMatchWindowCreateInfoQCOM in the `pNext` chain
- VUID-RuntimeSpirv-MaximallyReconvergesKHR-09565
  The execution mode `MaximallyReconvergesKHR` **must** not be applied to an entry point unless the entry point does not execute any *invocation repack instructions*
- VUID-RuntimeSpirv-shaderSubgroupRotateClustered-09566
  If `shaderSubgroupRotateClustered` is `VK_FALSE`, then the `ClusterSize` operand to `OpGroupNonUniformRotateKHR` **must** not be used
- VUID-RuntimeSpirv-protectedNoFault-09645
  If `protectedNoFault` is not supported, the `Storage Class` of the `PhysicalStorageBuffer` **must** not be used if the buffer being accessed is protected
- VUID-RuntimeSpirv-meshAndTaskShaderDerivatives-10153
  If `meshAndTaskShaderDerivatives` is `VK_FALSE`, the `DerivativeGroupLinearKHR` and `DerivativeGroupQuadsKHR` execution modes **must** not be used in the `MeshEXT`, `MeshNV`, `TaskEXT`, or `TaskNV Execution Model`

# Precision and Operation of SPIR-V Instructions

The following rules apply to half, single, and double-precision floating-point instructions:

- Positive and negative infinities and positive and negative zeros are generated as dictated by IEEE 754, but subject to the precisions allowed in the table below.

- Signaling NaNs are not required to be generated and exceptions are never raised. Signaling NaN **may** be converted to quiet NaNs values by any floating-point instruction.
- The set of operations OpPhi, OpSelect, OpFunctionCall, OpReturnValue, OpVectorExtractDynamic, OpVectorInsertDynamic, OpVectorShuffle, OpCompositeConstruct, OpCompositeExtract, OpCompositeInsert, OpTranspose, OpCopyObject, OpCopyLogical, OpCopyMemory, OpGroupNonUniformBroadcast, OpGroupNonUniformBroadcastFirst, OpGroupNonUniformShuffle, OpGroupNonUniformShuffleXor, OpGroupNonUniformShuffleUp, OpGroupNonUniformShuffleDown, OpGroupNonUniformQuadBroadcast, OpGroupNonUniformQuadSwap, OpSubgroupReadInvocationKHR, OpSubgroupFirstInvocationKHR, OpGroupNonUniformRotateKHR, OpCooperativeMatrixLoadKHR, OpCooperativeMatrixStoreKHR, OpCooperativeMatrixLoadNV, OpCooperativeMatrixStoreNV, OpAtomicLoad, OpAtomicStore, OpAtomicExchange, OpStore, and OpLoad are referred to as *bit-preserving operations*.
- The floating-point environment used for an instruction can be determined as follows:
  - If the SPIR-V specifies it explicitly using the FPFastMath decoration or FPFastMathDefault Execution Mode then that is used.
  - If the environment is not specified in the SPIR-V then it is determined as follows:
    - If the operation is not decorated NoContraction then the flags AllowContract, AllowReassoc, AllowRecip, and AllowTransform are assumed.
    - If any of the following conditions are true then the flags NSZ, NotInf, and NotNaN are assumed:
      - The entry point does not use the Execution Mode SignedZeroInfNanPreserve with a bit-width corresponding to one of the operands or to the result type.
      - The operation is not a bit-preserving operation and is not one of OpFConvert, OpFNegate, OpFAdd, OpFSub, OpFMul, OpIsNan, or OpIsInf.
      - The operation is an OpLoad from the Input Storage Class in the fragment shader stage.
- All bit-preserving operations and the following instructions **must** not flush denormalized values: OpConstant, OpConstantComposite, OpSpecConstant, OpSpecConstantComposite, and OpBitcast.
- Denormalized values are supported.
  - By default, any half, single, or double-precision denormalized value input into a shader or potentially generated by any instruction (except those listed above) or any extended instructions for GLSL in a shader **may** be flushed to zero.
  - If the entry point is declared with the DenormFlushToZero Execution Mode then for the affected instructions the denormalized result **must** be flushed to zero and the denormalized operands **may** be flushed to zero. Denormalized values obtained via unpacking an integer into a vector of values with smaller bit width and interpreting those values as floating-point numbers **must** be flushed to zero.
  - The following core SPIR-V instructions **must** respect the DenormFlushToZero Execution Mode: OpSpecConstantOp (with opcode OpFConvert), OpFConvert, OpFNegate, OpFAdd, OpFSub, OpFMul, OpFDiv, OpFRem, OpFMod, OpVectorTimesScalar, OpMatrixTimesScalar, OpVectorTimesMatrix, OpMatrixTimesVector, OpMatrixTimesMatrix, OpOuterProduct, OpDot; and the following extended instructions for GLSL: Round, RoundEven, Trunc, FAbs, Floor, Ceil, Fract, Radians, Degrees, Sin, Cos, Tan, Asin, Acos, Atan, Sinh, Cosh, Tanh, Asinh, Acosh, Atanh, Atan2, Pow, Exp, Log, Exp2, Log2,

Sqrt, InverseSqrt, Determinant, MatrixInverse, Modf, ModfStruct, FMin, FMax, FClamp, FMix, Step, SmoothStep, Fma, UnpackHalf2x16, Length, Distance, Cross, Normalize, FaceForward, Reflect, Refract, NMin, NMax, and NClamp.

- The following core SPIR-V instructions **must** respect the DenormPreserve Execution Mode: OpSpecConstantOp, OpFConvert, OpFNegate, OpFAdd, OpFSub, OpFMul, OpVectorTimesScalar, OpMatrixTimesScalar, OpVectorTimesMatrix, OpMatrixTimesVector, OpMatrixTimesMatrix, OpOuterProduct, OpDot, OpFOrdEqual, OpFUnordEqual, OpFOrdNotEqual, OpFUnordNotEqual, OpFOrdLessThan, OpFUnordLessThan, OpFOrdGreaterThan, OpFUnordGreaterThan, OpFOrdLessThanEqual, OpFUnordLessThanEqual, OpFOrdGreaterThanEqual, OpFUnordGreaterThanEqual; and the following extended instructions for GLSL: FAbs, FSign, Radians, Degrees, FMin, FMax, FClamp, FMix, Fma, PackHalf2x16, PackDouble2x32, UnpackHalf2x16, UnpackDouble2x32, NMin, NMax, and NClamp.

The precision of double-precision instructions is at least that of single precision.

The precision of individual operations is defined in Precision of Individual Operations. Subject to the constraints below, however, implementations **may** reorder or combine operations, resulting in expressions exhibiting different precisions than might be expected from the constituent operations.

## Evaluation of Expressions

Implementations **may** rearrange floating-point operations using any of the mathematical properties governing the expressions in precise arithmetic, even where the floating- point operations do not share these properties. This includes, but is not limited to, associativity and distributivity, and **may** involve a different number of rounding steps than would occur if the operations were not rearranged. In shaders that use the SignedZeroInfNanPreserve Execution Mode the values **must** be preserved if they are generated after any rearrangement but the Execution Mode does not change which rearrangements are valid. This rearrangement **can** be prevented for particular operations by using the NoContraction decoration.

**NOTE**

For example, in the absence of the NoContraction decoration implementations are allowed to implement a + b - a and $\frac{a \times b}{a}$ as b. The SignedZeroInfNanPreserve does not prevent these transformations, even though they may overflow to infinity or NaN when evaluated in floating-point.

If the NoContraction decoration is applied then operations may not be rearranged, so, for example, a + a - a must account for possible overflow to infinity. If infinities are not preserved then the expression may be replaced with a, since the replacement is exact when overflow does not occur and infinities may be replaced with undefined values. If both NoContraction and SignedZeroInfNanPreserve are used then the result must be infinity for sufficiently large a.

## Precision of Individual Operations

The precision of individual operations is defined either in terms of rounding (correctly rounded), as an error bound in ULP, or as inherited from a formula as follows:

*Correctly Rounded*

Operations described as “correctly rounded” will return the infinitely precise result, x, rounded so as to be representable in floating-point. The rounding mode is not specified, unless the entry point is declared with the `RoundingModeRTE` or the `RoundingModeRTZ Execution Mode`. These execution modes affect only correctly rounded SPIR-V instructions. These execution modes do not affect `OpQuantizeToF16` and `PackHalf2x16`. If the rounding mode is not specified then this rounding is implementation specific, subject to the following rules. If x is exactly representable then x will be returned. Otherwise, either the floating-point value closest to and no less than x or the value closest to and no greater than x will be returned.

*ULP*

Where an error bound of n ULP (units in the last place) is given, for an operation with infinitely precise result x the value returned **must** be in the range [x - n × ulp(x), x + n × ulp(x)]. The function ulp(x) is defined as follows:

> If there exist non-equal, finite floating-point numbers a and b such that a ≤ x ≤ b then ulp(x) is the minimum possible distance between such numbers, $ulp(x) = \min_{a,\ b}|b - a|$. If such numbers do not exist then ulp(x) is defined to be the difference between the two non-equal, finite floating-point numbers nearest to x.

Where the range of allowed return values includes any value of magnitude larger than that of the largest representable finite floating-point number, operations **may**, additionally, return either an infinity of the appropriate sign or the finite number with the largest magnitude of the appropriate sign. If the infinitely precise result of the operation is not mathematically defined then the value returned is undefined.

*Inherited From ...*

Where an operation’s precision is described as being inherited from a formula, the result returned **must** be at least as accurate as the result of computing an approximation to x using a formula equivalent to the given formula applied to the supplied inputs. Specifically, the formula given may be transformed using the mathematical associativity, commutativity and distributivity of the operators involved to yield an equivalent formula. The SPIR-V precision rules, when applied to each such formula and the given input values, define a range of permitted values. If NaN is one of the permitted values then the operation may return any result, otherwise let the largest permitted value in any of the ranges be $F_{max}$ and the smallest be $F_{min}$. The operation **must** return a value in the range [x - E, x + E] where $E = \max(|x - F_{\min}|, |x - F_{\max}|)$. If the entry point is declared with the `DenormFlushToZero` execution mode, then any intermediate denormal value(s) while evaluating the formula **may** be flushed to zero. Denormal final results **must** be flushed to zero. If the entry point is declared with the `DenormPreserve Execution Mode`, then denormals **must** be preserved throughout the formula.

For half- (16 bit) and single- (32 bit) precision instructions, precisions are **required** to be at least as follows:

*Table 98. Precision of core SPIR-V Instructions*

| Instruction | Single precision, unless decorated with RelaxedPrecision | Half precision |
|---|---|---|
| OpFNegate | Correct result. | |
| OpFAdd | Correctly rounded. | |
| OpFSub | Correctly rounded. | |
| OpFMul, OpVectorTimesScalar, OpMatrixTimesScalar | Correctly rounded. | |
| OpMatrixTimesVector | Inherited from $\sum_{j=0}^{n-1} M_{ij} \times v_j$. | |
| OpVectorTimesMatrix | Inherited from $\sum_{j=0}^{n-1} v_j \times M_{ji}$. | |
| OpMatrixTimesMatrix | Inherited from $\sum_{k=0}^{n-1} M_{ik} \times N_{kj}$. | |
| OpOuterProduct | Correctly rounded. | |
| OpDot(x, y) | Inherited from $\sum_{i=0}^{n-1} x_i \times y_i$. | |
| OpIsNan, OpIsInf | Correct Result. | |
| OpFOrdEqual, OpFUnordEqual | Correct result. | |
| OpFOrdNotEqual, OpFUnordNotEqual | Correct result. | |
| OpFOrdLessThan, OpFUnordLessThan | Correct result. | |
| OpFOrdGreaterThan, OpFUnordGreaterThan | Correct result. | |
| OpFOrdLessThanEqual, OpFUnordLessThanEqual | Correct result. | |
| OpFOrdGreaterThanEqual, OpFUnordGreaterThanEqual | Correct result. | |
| OpFDiv(x,y) | 2.5 ULP for \|y\| = 0 or \|y\| in the range $[2^{-126}, 2^{126}]$. | 2.5 ULP for \|y\| = 0 or \|y\| in the range $[2^{-14}, 2^{14}]$. |
| OpFRem(x,y) | Inherited from x - y × trunc(x/y). | |
| OpFMod(x,y) | Inherited from x - y × floor(x/y). | |
| OpQuantizeToF16 | Correctly rounded. | |
| conversions between types | Correctly rounded. | |

**NOTE** The OpFRem and OpFMod instructions use cheap approximations of remainder, and the error can be large due to the discontinuity in trunc() and floor(). This can produce mathematically unexpected results in some cases, such as FMod(x,x) computing x rather than 0, and can also cause the result to have a different sign than the infinitely precise result.

*Table 99. Precision of GLSL.std.450 Instructions*

| Instruction | Single precision, unless decorated with RelaxedPrecision | Half precision |
|---|---|---|
| `fma`() | Inherited from `OpFMul` followed by `OpFAdd`. | |
| `exp`(x), `exp2`(x) | $3 + 2 \times \lvert x \rvert$ ULP. | $1 + 2 \times \lvert x \rvert$ ULP. |
| `log`(), `log2`() | 3 ULP outside the range $[0.5, 2.0]$. Absolute error $< 2^{-21}$ inside the range $[0.5, 2.0]$. | 3 ULP outside the range $[0.5, 2.0]$. Absolute error $< 2^{-7}$ inside the range $[0.5, 2.0]$. |
| `pow`(x, y) | Inherited from `exp2`(y × `log2`(x)). | |
| `sqrt`() | Inherited from 1.0 / `inversesqrt`(). | |
| `inversesqrt`() | 2 ULP. | |
| `radians`(x) | Inherited from $x \times C_{\pi_180}$, where $C_{\pi_180}$ is a correctly rounded approximation to $\frac{\pi}{180}$. | |
| `degrees`(x) | Inherited from $x \times C_{180_\pi}$, where $C_{180_\pi}$ is a correctly rounded approximation to $\frac{180}{\pi}$. | |
| `sin`() | Absolute error $\leq 2^{-11}$ inside the range $[-\pi, \pi]$. | Absolute error $\leq 2^{-7}$ inside the range $[-\pi, \pi]$. |
| `cos`() | Absolute error $\leq 2^{-11}$ inside the range $[-\pi, \pi]$. | Absolute error $\leq 2^{-7}$ inside the range $[-\pi, \pi]$. |
| `tan`() | Inherited from $\frac{\sin()}{\cos()}$. | |
| `asin`(x) | Inherited from $\text{atan2}(x, sqrt(1.0 - x \times x))$. | |
| `acos`(x) | Inherited from $\text{atan2}(sqrt(1.0 - x \times x), x)$. | |
| `atan`(), `atan2`() | 4096 ULP | 5 ULP. |
| `sinh`(x) | Inherited from $(\exp(x) - \exp(-x)) \times 0.5$. | |
| `cosh`(x) | Inherited from $(\exp(x) + \exp(-x)) \times 0.5$. | |
| `tanh`() | Inherited from $\frac{\sinh()}{\cosh()}$. | |
| `asinh`(x) | Inherited from $\log(x + sqrt(x \times x + 1.0))$. | |
| `acosh`(x) | Inherited from $\log(x + sqrt(x \times x - 1.0))$. | |
| `atanh`(x) | Inherited from $\log(\frac{1.0 + x}{1.0 - x}) \times 0.5$. | |
| `frexp`() | Correctly rounded. | |
| `ldexp`() | Correctly rounded. | |
| `length`(x) | Inherited from $sqrt(dot(x, x))$. | |
| `distance`(x, y) | Inherited from $length(x - y)$. | |
| `cross`() | Inherited from `OpFSub`(`OpFMul`, `OpFMul`). | |
| `normalize`(x) | Inherited from $x \times inversesqrt(dot(x, x))$. | |
| `faceforward`(N, I, NRef) | Inherited from `dot`(NRef, I) < 0.0 ? N : -N. | |

| Instruction | Single precision, unless decorated with RelaxedPrecision | Half precision |
|---|---|---|
| reflect(x, y) | Inherited from x - 2.0 × dot(y, x) × y. | |
| refract(I, N, eta) | Inherited from k < 0.0 ? 0.0 : eta × I - (eta × dot(N, I) + sqrt(k)) × N, where k = 1 - eta × eta × (1.0 - dot(N, I) × dot(N, I)). | |
| round | Correctly rounded. | |
| roundEven | Correctly rounded. | |
| trunc | Correctly rounded. | |
| fabs | Correctly rounded. | |
| fsign | Correctly rounded. | |
| floor | Correctly rounded. | |
| ceil | Correctly rounded. | |
| fract | Correctly rounded. | |
| modf | Correctly rounded. | |
| fmin | Correctly rounded. | |
| fmax | Correctly rounded. | |
| fclamp | Correctly rounded. | |
| fmix(x, y, a) | Inherited from $x \times (1.0 - a) + y \times a$. | |
| step | Correctly rounded. | |
| smoothStep(edge0, edge1, x) | Inherited from $t \times t \times (3.0 - 2.0 \times t)$, where $t = clamp(\frac{x - edge0}{edge1 - edge0}, 0.0, 1.0)$. | |
| nmin | Correctly rounded. | |
| nmax | Correctly rounded. | |
| nclamp | Correctly rounded. | |
| packHalf2x16 | Correctly rounded. | |

GLSL.std.450 extended instructions specifically defined in terms of the above instructions inherit the above errors. GLSL.std.450 extended instructions not listed above and not defined in terms of the above have undefined precision.

For the OpSRem and OpSMod instructions, if either operand is negative the result is undefined.

**NOTE** While the OpSRem and OpSMod instructions are supported by the Vulkan environment, they require non-negative values and thus do not enable additional functionality beyond what OpUMod provides.

OpCooperativeMatrixMulAddNV performs its operations in an implementation-dependent order and internal precision.

`OpCooperativeMatrixMulAddKHR` performs its operations in an implementation-dependent order and internal precision.

## Signedness of SPIR-V Image Accesses

SPIR-V associates a signedness with all integer image accesses. This is required in certain parts of the SPIR-V and the Vulkan image access pipeline to ensure defined results. The signedness is determined from a combination of the access instruction's `Image Operands` and the underlying image's `Sampled Type` as follows:

1. If the instruction's `Image Operands` contains the `SignExtend` operand then the access is signed.
2. If the instruction's `Image Operands` contains the `ZeroExtend` operand then the access is unsigned.
3. Otherwise, the image accesses signedness matches that of the `Sampled Type` of the `OpTypeImage` being accessed.

## Image Format and Type Matching

When specifying the `Image Format` of an `OpTypeImage`, the converted bit width and type, as shown in the table below, **must** match the `Sampled Type`. The signedness **must** match the signedness of any access to the image.

**NOTE** Formatted accesses are always converted from a shader readable type to the resource's format or vice versa via Format Conversion for reads and Texel Output Format Conversion for writes. As such, the bit width and format below do not necessarily match 1:1 with what might be expected for some formats.

For a given `Image Format`, the `Sampled Type` **must** be the type described in the *Type* column of the below table, with its `Literal Width` set to that in the *Bit Width* column. Every access that is made to the image **must** have a signedness equal to that in the *Signedness* column (where applicable).

| Image Format | Type-Declaration instructions | Bit Width | Signedness |
|---|---|---|---|
| `Unknown` | Any | Any | Any |

| Image Format | Type-Declaration instructions | Bit Width | Signedness |
|---|---|---|---|
| `Rgba32f` | `OpTypeFloat` | 32 | N/A |
| `Rg32f` | | | |
| `R32f` | | | |
| `Rgba16f` | | | |
| `Rg16f` | | | |
| `R16f` | | | |
| `Rgba16` | | | |
| `Rg16` | | | |
| `R16` | | | |
| `Rgba16Snorm` | | | |
| `Rg16Snorm` | | | |
| `R16Snorm` | | | |
| `Rgb10A2` | | | |
| `R11fG11fB10f` | | | |
| `Rgba8` | | | |
| `Rg8` | | | |
| `R8` | | | |
| `Rgba8Snorm` | | | |
| `Rg8Snorm` | | | |
| `R8Snorm` | | | |

| Image Format | Type-Declaration instructions | Bit Width | Signedness |
|---|---|---|---|
| `Rgba32i` | `OpTypeInt` | 32 | 1 |
| `Rg32i` | | | |
| `R32i` | | | |
| `Rgba16i` | | | |
| `Rg16i` | | | |
| `R16i` | | | |
| `Rgba8i` | | | |
| `Rg8i` | | | |
| `R8i` | | | |
| `Rgba32ui` | | | 0 |
| `Rg32ui` | | | |
| `R32ui` | | | |
| `Rgba16ui` | | | |
| `Rg16ui` | | | |
| `R16ui` | | | |
| `Rgb10a2ui` | | | |
| `Rgba8ui` | | | |
| `Rg8ui` | | | |
| `R8ui` | | | |
| `R64i` | `OpTypeInt` | 64 | 1 |
| `R64ui` | | | 0 |

The *SPIR-V Type* is defined by an instruction in SPIR-V, declared with the Type-Declaration Instruction, Bit Width, and Signedness from above.

# Compatibility Between SPIR-V Image Dimensions and Vulkan ImageView Types

SPIR-V `Image Dim` values are compatible with VkImageView `viewType` values as defined below:

*Table 100. SPIR-V and Vulkan ImageView Dimension Compatibility*

| SPIR-V Image Dim | Compatible Vulkan ImageView viewTypes |
|---|---|
| 1D | `VK_IMAGE_VIEW_TYPE_1D`, `VK_IMAGE_VIEW_TYPE_1D_ARRAY` |
| 2D | `VK_IMAGE_VIEW_TYPE_2D`, `VK_IMAGE_VIEW_TYPE_2D_ARRAY` |
| 3D | `VK_IMAGE_VIEW_TYPE_3D` |
| Cube | `VK_IMAGE_VIEW_TYPE_CUBE`, `VK_IMAGE_VIEW_TYPE_CUBE_ARRAY` |

# Compatibility Between SPIR-V Image Formats and Vulkan Formats

SPIR-V `Image Format` values are compatible with VkFormat values as defined below:

*Table 101. SPIR-V and Vulkan Image Format Compatibility*

| SPIR-V Image Format | Compatible Vulkan Format |
|---|---|
| `Unknown` | Any |
| `R8` | `VK_FORMAT_R8_UNORM` |
| `R8Snorm` | `VK_FORMAT_R8_SNORM` |
| `R8ui` | `VK_FORMAT_R8_UINT` |
| `R8i` | `VK_FORMAT_R8_SINT` |
| `Rg8` | `VK_FORMAT_R8G8_UNORM` |
| `Rg8Snorm` | `VK_FORMAT_R8G8_SNORM` |
| `Rg8ui` | `VK_FORMAT_R8G8_UINT` |
| `Rg8i` | `VK_FORMAT_R8G8_SINT` |
| `Rgba8` | `VK_FORMAT_R8G8B8A8_UNORM` |
| `Rgba8Snorm` | `VK_FORMAT_R8G8B8A8_SNORM` |
| `Rgba8ui` | `VK_FORMAT_R8G8B8A8_UINT` |
| `Rgba8i` | `VK_FORMAT_R8G8B8A8_SINT` |
| `Rgb10A2` | `VK_FORMAT_A2B10G10R10_UNORM_PACK32` |
| `Rgb10a2ui` | `VK_FORMAT_A2B10G10R10_UINT_PACK32` |
| `R16` | `VK_FORMAT_R16_UNORM` |
| `R16Snorm` | `VK_FORMAT_R16_SNORM` |
| `R16ui` | `VK_FORMAT_R16_UINT` |
| `R16i` | `VK_FORMAT_R16_SINT` |
| `R16f` | `VK_FORMAT_R16_SFLOAT` |
| `Rg16` | `VK_FORMAT_R16G16_UNORM` |
| `Rg16Snorm` | `VK_FORMAT_R16G16_SNORM` |
| `Rg16ui` | `VK_FORMAT_R16G16_UINT` |
| `Rg16i` | `VK_FORMAT_R16G16_SINT` |
| `Rg16f` | `VK_FORMAT_R16G16_SFLOAT` |
| `Rgba16` | `VK_FORMAT_R16G16B16A16_UNORM` |
| `Rgba16Snorm` | `VK_FORMAT_R16G16B16A16_SNORM` |
| `Rgba16ui` | `VK_FORMAT_R16G16B16A16_UINT` |
| `Rgba16i` | `VK_FORMAT_R16G16B16A16_SINT` |
| `Rgba16f` | `VK_FORMAT_R16G16B16A16_SFLOAT` |
| `R32ui` | `VK_FORMAT_R32_UINT` |
| `R32i` | `VK_FORMAT_R32_SINT` |
| `R32f` | `VK_FORMAT_R32_SFLOAT` |

| SPIR-V Image Format | Compatible Vulkan Format |
|---|---|
| `Rg32ui` | `VK_FORMAT_R32G32_UINT` |
| `Rg32i` | `VK_FORMAT_R32G32_SINT` |
| `Rg32f` | `VK_FORMAT_R32G32_SFLOAT` |
| `Rgba32ui` | `VK_FORMAT_R32G32B32A32_UINT` |
| `Rgba32i` | `VK_FORMAT_R32G32B32A32_SINT` |
| `Rgba32f` | `VK_FORMAT_R32G32B32A32_SFLOAT` |
| `R64ui` | `VK_FORMAT_R64_UINT` |
| `R64i` | `VK_FORMAT_R64_SINT` |
| `R11fG11fB10f` | `VK_FORMAT_B10G11R11_UFLOAT_PACK32` |

# Ray Query Precision and Operation

The values returned by `OpRayQueryGetIntersectionTriangleVertexPositionsKHR` are transformed by the geometry transform, which is performed at standard floating-point precision, but without a specifically defined order of floating-point operations to perform the matrix multiplication.

# Appendix B: Memory Model

| | |
|---|---|
| **NOTE** | This memory model describes synchronizations provided by all implementations; however, some of the synchronizations defined require extra features to be supported by the implementation. See VkPhysicalDeviceVulkanMemoryModelFeatures. |

## Agent

*Operation* is a general term for any task that is executed on the system.

| | |
|---|---|
| **NOTE** | An operation is by definition something that is executed. Thus if an instruction is skipped due to control flow, it does not constitute an operation. |

Each operation is executed by a particular *agent*. Possible agents include each shader invocation, each host thread, and each fixed-function stage of the pipeline.

## Memory Location

A *memory location* identifies unique storage for 8 bits of data. Memory operations access a *set of memory locations* consisting of one or more memory locations at a time, e.g. an operation accessing a 32-bit integer in memory would read/write a set of four memory locations. Memory operations that access whole aggregates **may** access any padding bytes between elements or members, but no padding bytes at the end of the aggregate. Two sets of memory locations *overlap* if the intersection of their sets of memory locations is non-empty. A memory operation **must** not affect memory at a memory location not within its set of memory locations.

Memory locations for buffers and images are explicitly allocated in VkDeviceMemory objects, and are implicitly allocated for SPIR-V variables in each shader invocation.

Variables with `Workgroup` storage class that point to a block-decorated type share a set of memory locations.

## Allocation

The values stored in newly allocated memory locations are determined by a SPIR-V variable's initializer, if present, or else are undefined. At the time an allocation is created there have been no memory operations to any of its memory locations. The initialization is not considered to be a memory operation.

| | |
|---|---|
| **NOTE** | For tessellation control shader output variables, a consequence of initialization not being considered a memory operation is that some implementations may need to insert a barrier between the initialization of the output variables and any reads of those variables. |

# Memory Operation

For an operation A and memory location M:

- A *reads* M if and only if the data stored in M is an input to A.
- A *writes* M if and only if the data output from A is stored to M.
- A *accesses* M if and only if it either reads or writes (or both) M.

| | |
|---|---|
| **NOTE** | A write whose value is the same as what was already in those memory locations is still considered to be a write and has all the same effects. |

# Reference

A *reference* is an object that a particular agent **can** use to access a set of memory locations. On the host, a reference is a host virtual address. On the device, a reference is:

- The descriptor that a variable is bound to, for variables in Image, Uniform, or StorageBuffer storage classes. If the variable is an array (or array of arrays, etc.) then each element of the array **may** be a unique reference.
- The address range for a buffer in `PhysicalStorageBuffer` storage class, where the base of the address range is queried with vkGetBufferDeviceAddressKHR and the length of the range is the size of the buffer.
- A single common reference for all variables with `Workgroup` storage class that point to a block-decorated type.
- The variable itself for non-block-decorated type variables in `Workgroup` storage class.
- The variable itself for variables in other storage classes.

Two memory accesses through distinct references **may** require availability and visibility operations as defined below.

# Program-Order

A *dynamic instance* of an instruction is defined in SPIR-V (https://registry.khronos.org/spir-v/specs/unified1/SPIRV.html#DynamicInstance) as a way of referring to a particular execution of a static instruction. Program-order is an ordering on dynamic instances of instructions executed by a single shader invocation:

- (Basic block): If instructions A and B are in the same basic block, and A is listed in the module before B, then the n'th dynamic instance of A is program-ordered before the n'th dynamic instance of B.
- (Branch): The dynamic instance of a branch or switch instruction is program-ordered before the dynamic instance of the OpLabel instruction to which it transfers control.
- (Call entry): The dynamic instance of an `OpFunctionCall` instruction is program-ordered before the dynamic instances of the `OpFunctionParameter` instructions and the body of the called

function.

- (Call exit): The dynamic instance of the instruction following an `OpFunctionCall` instruction is program-ordered after the dynamic instance of the return instruction executed by the called function.
- (Transitive Closure): If dynamic instance A of any instruction is program-ordered before dynamic instance B of any instruction and B is program-ordered before dynamic instance C of any instruction then A is program-ordered before C.
- (Complete definition): No other dynamic instances are program-ordered.

For instructions executed on the host, the source language defines the program-order relation (e.g. as “sequenced-before”).

# Shader Call Related

Shader-call-related is an equivalence relation on invocations defined as the symmetric and transitive closure of:

- A is shader-call-related to B if A is created by an shader call instruction executed by B.

# Shader Call Order

Shader-call-order is a partial order on dynamic instances of instructions executed by invocations that are shader-call-related:

- (Program order): If dynamic instance A is program-ordered before B, then A is shader-call-ordered before B.
- (Shader call entry): If A is a dynamic instance of an shader call instruction and B is a dynamic instance executed by an invocation that is created by A, then A is shader-call-ordered before B.
- (Shader call exit): If A is a dynamic instance of an shader call instruction, B is the next dynamic instance executed by the same invocation, and C is a dynamic instance executed by an invocation that is created by A, then C is shader-call-ordered before B.
- (Transitive closure): If A is shader-call-ordered-before B and B is shader-call-ordered-before C, then A is shader-call-ordered-before C.
- (Complete definition): No other dynamic instances are shader-call-ordered.

# Scope

Atomic and barrier instructions include scopes which identify sets of shader invocations that **must** obey the requested ordering and atomicity rules of the operation, as defined below.

The various scopes are described in detail in the Shaders chapter.

# Atomic Operation

An *atomic operation* on the device is any SPIR-V operation whose name begins with `OpAtomic`. An atomic operation on the host is any operation performed with an std::atomic typed object.

Each atomic operation has a memory scope and a semantics. Informally, the scope determines which other agents it is atomic with respect to, and the semantics constrains its ordering against other memory accesses. Device atomic operations have explicit scopes and semantics. Each host atomic operation implicitly uses the `CrossDevice` scope, and uses a memory semantics equivalent to a C++ std::memory_order value of relaxed, acquire, release, acq_rel, or seq_cst.

Two atomic operations A and B are *potentially-mutually-ordered* if and only if all of the following are true:

- They access the same set of memory locations.
- They use the same reference.
- A is in the instance of B's memory scope.
- B is in the instance of A's memory scope.
- A and B are not the same operation (irreflexive).

Two atomic operations A and B are *mutually-ordered* if and only if they are potentially-mutually-ordered and any of the following are true:

- A and B are both device operations.
- A and B are both host operations.
- A is a device operation, B is a host operation, and the implementation supports concurrent host- and device-atomics.

**NOTE** If two atomic operations are not mutually-ordered, and if their sets of memory locations overlap, then each **must** be synchronized against the other as if they were non-atomic operations.

# Scoped Modification Order

For a given atomic write A, all atomic writes that are mutually-ordered with A occur in an order known as A's *scoped modification order*. A's scoped modification order relates no other operations.

**NOTE** Invocations outside the instance of A's memory scope **may** observe the values at A's set of memory locations becoming visible to it in an order that disagrees with the scoped modification order.

**NOTE** It is valid to have non-atomic operations or atomics in a different scope instance to the same set of memory locations, as long as they are synchronized against each other as if they were non-atomic (if they are not, it is treated as a data race). That means this definition of A's scoped modification order could include atomic

> operations that occur much later, after intervening non-atomics. That is a bit non-intuitive, but it helps to keep this definition simple and non-circular.

## Memory Semantics

Non-atomic memory operations, by default, **may** be observed by one agent in a different order than they were written by another agent.

Atomics and some synchronization operations include *memory semantics*, which are flags that constrain the order in which other memory accesses (including non-atomic memory accesses and availability and visibility operations) performed by the same agent **can** be observed by other agents, or **can** observe accesses by other agents.

Device instructions that include semantics are `OpAtomic*`, `OpControlBarrier`, `OpMemoryBarrier`, and `OpMemoryNamedBarrier`. Host instructions that include semantics are some std::atomic methods and memory fences.

SPIR-V supports the following memory semantics:

- Relaxed: No constraints on order of other memory accesses.
- Acquire: A memory read with this semantic performs an *acquire operation*. A memory barrier with this semantic is an *acquire barrier*.
- Release: A memory write with this semantic performs a *release operation*. A memory barrier with this semantic is a *release barrier*.
- AcquireRelease: A memory read-modify-write operation with this semantic performs both an acquire operation and a release operation, and inherits the limitations on ordering from both of those operations. A memory barrier with this semantic is both a release and acquire barrier.

**NOTE** SPIR-V does not support “consume” semantics on the device.

The memory semantics operand also includes *storage class semantics* which indicate which storage classes are constrained by the synchronization. SPIR-V storage class semantics include:

- UniformMemory
- WorkgroupMemory
- ImageMemory
- OutputMemory

Each SPIR-V memory operation accesses a single storage class. Semantics in synchronization operations can include a combination of storage classes.

The UniformMemory storage class semantic applies to accesses to memory in the PhysicalStorageBuffer, `ShaderRecordBufferKHR`, Uniform and StorageBuffer storage classes. The WorkgroupMemory storage class semantic applies to accesses to memory in the Workgroup storage class. The ImageMemory storage class semantic applies to accesses to memory in the Image storage class. The OutputMemory storage class semantic applies to accesses to memory in the Output

storage class.

| | |
|---|---|
| **NOTE** | Informally, these constraints limit how memory operations can be reordered, and these limits apply not only to the order of accesses as performed in the agent that executes the instruction, but also to the order the effects of writes become visible to all other agents within the same instance of the instruction's memory scope. |

| | |
|---|---|
| **NOTE** | Release and acquire operations in different threads **can** act as synchronization operations, to guarantee that writes that happened before the release are visible after the acquire. (This is not a formal definition, just an Informative forward reference.) |

| | |
|---|---|
| **NOTE** | The OutputMemory storage class semantic is only useful in tessellation control shaders, which is the only execution model where output variables are shared between invocations. |

The memory semantics operand **can** also include availability and visibility flags, which apply availability and visibility operations as described in availability and visibility. The availability/visibility flags are:

- MakeAvailable: Semantics **must** be Release or AcquireRelease. Performs an availability operation before the release operation or barrier.
- MakeVisible: Semantics **must** be Acquire or AcquireRelease. Performs a visibility operation after the acquire operation or barrier.

The specifics of these operations are defined in Availability and Visibility Semantics.

Host atomic operations **may** support a different list of memory semantics and synchronization operations, depending on the host architecture and source language.

# Release Sequence

After an atomic operation A performs a release operation on a set of memory locations M, the *release sequence headed by A* is the longest continuous subsequence of A's scoped modification order that consists of:

- the atomic operation A as its first element
- atomic read-modify-write operations on M by any agent

| | |
|---|---|
| **NOTE** | The atomics in the last bullet **must** be mutually-ordered with A by virtue of being in A's scoped modification order. |

| | |
|---|---|
| **NOTE** | This intentionally omits "atomic writes to M performed by the same agent that performed A", which is present in the corresponding C++ definition. |

# Synchronizes-With

*Synchronizes-with* is a relation between operations, where each operation is either an atomic operation or a memory barrier (aka fence on the host).

If A and B are atomic operations, then A synchronizes-with B if and only if all of the following are true:

- A performs a release operation
- B performs an acquire operation
- A and B are mutually-ordered
- B reads a value written by A or by an operation in the release sequence headed by A

`OpControlBarrier`, `OpMemoryBarrier`, and `OpMemoryNamedBarrier` are *memory barrier* instructions in SPIR-V.

If A is a release barrier and B is an atomic operation that performs an acquire operation, then A synchronizes-with B if and only if all of the following are true:

- there exists an atomic write X (with any memory semantics)
- A is program-ordered before X
- X and B are mutually-ordered
- B reads a value written by X or by an operation in the release sequence headed by X
  - If X is relaxed, it is still considered to head a hypothetical release sequence for this rule
- A and B are in the instance of each other's memory scopes
- X's storage class is in A's semantics.

If A is an atomic operation that performs a release operation and B is an acquire barrier, then A synchronizes-with B if and only if all of the following are true:

- there exists an atomic read X (with any memory semantics)
- X is program-ordered before B
- X and A are mutually-ordered
- X reads a value written by A or by an operation in the release sequence headed by A
- A and B are in the instance of each other's memory scopes
- X's storage class is in B's semantics.

If A is a release barrier and B is an acquire barrier, then A synchronizes-with B if all of the following are true:

- there exists an atomic write X (with any memory semantics)
- A is program-ordered before X
- there exists an atomic read Y (with any memory semantics)

- Y is program-ordered before B
- X and Y are mutually-ordered
- Y reads the value written by X or by an operation in the release sequence headed by X
  - If X is relaxed, it is still considered to head a hypothetical release sequence for this rule
- A and B are in the instance of each other's memory scopes
- X's and Y's storage class is in A's and B's semantics.
  - NOTE: X and Y **must** have the same storage class, because they are mutually ordered.

If A is a release barrier, B is an acquire barrier, and C is a control barrier (where A **can** equal C, and B **can** equal C), then A synchronizes-with B if all of the following are true:

- A is program-ordered before (or equals) C
- C is program-ordered before (or equals) B
- A and B are in the instance of each other's memory scopes
- A and B are in the instance of C's execution scope

**NOTE** This is similar to the barrier-barrier synchronization above, but with a control barrier filling the role of the relaxed atomics.

Let F be an ordering of fragment shader invocations, such that invocation $F_1$ is ordered before invocation $F_2$ if and only if $F_1$ and $F_2$ overlap as described in Fragment Shader Interlock and $F_1$ executes the interlocked code before $F_2$.

If A is an `OpEndInvocationInterlockEXT` instruction and B is an `OpBeginInvocationInterlockEXT` instruction, then A synchronizes-with B if the agent that executes A is ordered before the agent that executes B in F. A and B are both considered to have `FragmentInterlock` memory scope and semantics of UniformMemory and ImageMemory, and A is considered to have Release semantics and B is considered to have Acquire semantics.

**NOTE** `OpBeginInvocationInterlockEXT` and `OpBeginInvocationInterlockEXT` do not perform implicit availability or visibility operations. Usually, shaders using fragment shader interlock will declare the relevant resources as `coherent` to get implicit per-instruction availability and visibility operations.

If A is a release barrier and B is an acquire barrier, then A synchronizes-with B if all of the following are true:

- A is shader-call-ordered-before B
- A and B are in the instance of each other's memory scopes

No other release and acquire barriers synchronize-with each other.

## System-Synchronizes-With

*System-synchronizes-with* is a relation between arbitrary operations on the device or host. Certain operations system-synchronize-with each other, which informally means the first operation occurs before the second and that the synchronization is performed without using application-visible memory accesses.

If there is an execution dependency between two operations A and B, then the operation in the first synchronization scope system-synchronizes-with the operation in the second synchronization scope.

| | |
|---|---|
| **NOTE** | This covers all Vulkan synchronization primitives, including device operations executing before a synchronization primitive is signaled, wait operations happening before subsequent device operations, signal operations happening before host operations that wait on them, and host operations happening before vkQueueSubmit. The list is spread throughout the synchronization chapter, and is not repeated here. |

System-synchronizes-with implicitly includes all storage class semantics and has `CrossDevice` scope.

If A system-synchronizes-with B, we also say A is *system-synchronized-before* B and B is *system-synchronized-after* A.

## Private vs. Non-Private

By default, non-atomic memory operations are treated as *private*, meaning such a memory operation is not intended to be used for communication with other agents. Memory operations with the NonPrivatePointer/NonPrivateTexel bit set are treated as *non-private*, and are intended to be used for communication with other agents.

More precisely, for private memory operations to be Location-Ordered between distinct agents requires using system-synchronizes-with rather than shader-based synchronization. Private memory operations still obey program-order.

Atomic operations are always considered non-private.

## Inter-Thread-Happens-Before

Let SC be a non-empty set of storage class semantics. Then (using template syntax) operation A *inter-thread-happens-before*<SC> operation B if and only if any of the following is true:

- A system-synchronizes-with B
- A synchronizes-with B, and both A and B have all of SC in their semantics
- A is an operation on memory in a storage class in SC or that has all of SC in its semantics, B is a release barrier or release atomic with all of SC in its semantics, and A is program-ordered before B
- A is an acquire barrier or acquire atomic with all of SC in its semantics, B is an operation on

memory in a storage class in SC or that has all of SC in its semantics, and A is program-ordered before B

- A and B are both host operations and A inter-thread-happens-before B as defined in the host language specification
- A inter-thread-happens-before<SC> some X and X inter-thread-happens-before<SC> B

# Happens-Before

Operation A *happens-before* operation B if and only if any of the following is true:

- A is program-ordered before B
- A inter-thread-happens-before<SC> B for some set of storage classes SC

*Happens-after* is defined similarly.

| | |
|---|---|
| **NOTE** | Unlike C++, happens-before is not always sufficient for a write to be visible to a read. Additional availability and visibility operations **may** be required for writes to be visible-to other memory accesses. |

| | |
|---|---|
| **NOTE** | Happens-before is not transitive, but each of program-order and inter-thread-happens-before<SC> are transitive. These can be thought of as covering the "single-threaded" case and the "multi-threaded" case, and it is not necessary (and not valid) to form chains between the two. |

# Availability and Visibility

*Availability* and *visibility* are states of a write operation, which (informally) track how far the write has permeated the system, i.e. which agents and references are able to observe the write. Availability state is per *memory domain*. Visibility state is per (agent,reference) pair. Availability and visibility states are per-memory location for each write.

Memory domains are named according to the agents whose memory accesses use the domain. Domains used by shader invocations are organized hierarchically into multiple smaller memory domains which correspond to the different scopes. Each memory domain is considered the *dual* of a scope, and vice versa. The memory domains defined in Vulkan include:

- *host* - accessible by host agents
- *device* - accessible by all device agents for a particular device
- *shader* - accessible by shader agents for a particular device, corresponding to the `Device` scope
- *queue family instance* - accessible by shader agents in a single queue family, corresponding to the `QueueFamily` scope.
- *fragment interlock instance* - accessible by fragment shader agents that overlap, corresponding to the `FragmentInterlock` scope.
- *shader call instance* - accessible by shader agents that are shader-call-related, corresponding to

the `ShaderCallKHR` scope.

- *workgroup instance* - accessible by shader agents in the same workgroup, corresponding to the `Workgroup` scope.
- *subgroup instance* - accessible by shader agents in the same subgroup, corresponding to the `Subgroup` scope.

The memory domains are nested in the order listed above, except for shader call instance domain, with memory domains later in the list nested in the domains earlier in the list. The shader call instance domain is at an implementation-dependent location in the list, and is nested according to that location. The shader call instance domain is not broader than the queue family instance domain.

**NOTE** Memory domains do not correspond to storage classes or device-local and host-local VkDeviceMemory allocations, rather they indicate whether a write can be made visible only to agents in the same subgroup, same workgroup, overlapping fragment shader invocation, shader-call-related ray tracing invocation, in any shader invocation, or anywhere on the device, or host. The shader, queue family instance, fragment interlock instance, shader call instance, workgroup instance, and subgroup instance domains are only used for shader-based availability/visibility operations, in other cases writes can be made available from/visible to the shader via the device domain.

*Availability operations*, *visibility operations*, and *memory domain operations* alter the state of the write operations that happen-before them, and which are included in their *source scope* to be available or visible to their *destination scope*.

- For an availability operation, the source scope is a set of (agent,reference,memory location) tuples, and the destination scope is a set of memory domains.
- For a memory domain operation, the source scope is a memory domain and the destination scope is a memory domain.
- For a visibility operation, the source scope is a set of memory domains and the destination scope is a set of (agent,reference,memory location) tuples.

How the scopes are determined depends on the specific operation. Availability and memory domain operations expand the set of memory domains to which the write is available. Visibility operations expand the set of (agent,reference,memory location) tuples to which the write is visible.

Recall that availability and visibility states are per-memory location, and let W be a write operation to one or more locations performed by agent A via reference R. Let L be one of the locations written. (W,L) (the write W to L), is initially not available to any memory domain and only visible to (A,R,L). An availability operation AV that happens-after W and that includes (A,R,L) in its source scope makes (W,L) *available* to the memory domains in its destination scope.

A memory domain operation DOM that happens-after AV and for which (W,L) is available in the source scope makes (W,L) available in the destination memory domain.

A visibility operation VIS that happens-after AV (or DOM) and for which (W,L) is available in any

domain in the source scope makes (W,L) *visible* to all (agent,reference,L) tuples included in its destination scope.

If write $W_2$ happens-after W, and their sets of memory locations overlap, then W will not be available/visible to all agents/references for those memory locations that overlap (and future AV/DOM/VIS ops cannot revive W's write to those locations).

Availability, memory domain, and visibility operations are treated like other non-atomic memory accesses for the purpose of memory semantics, meaning they can be ordered by release-acquire sequences or memory barriers.

An *availability chain* is a sequence of availability operations to increasingly broad memory domains, where element N+1 of the chain is performed in the dual scope instance of the destination memory domain of element N and element N happens-before element N+1. An example is an availability operation with destination scope of the workgroup instance domain that happens-before an availability operation to the shader domain performed by an invocation in the same workgroup. An availability chain AVC that happens-after W and that includes (A,R,L) in the source scope makes (W,L) *available* to the memory domains in its final destination scope. An availability chain with a single element is just the availability operation.

Similarly, a *visibility chain* is a sequence of visibility operations from increasingly narrow memory domains, where element N of the chain is performed in the dual scope instance of the source memory domain of element N+1 and element N happens-before element N+1. An example is a visibility operation with source scope of the shader domain that happens-before a visibility operation with source scope of the workgroup instance domain performed by an invocation in the same workgroup. A visibility chain VISC that happens-after AVC (or DOM) and for which (W,L) is available in any domain in the source scope makes (W,L) *visible* to all (agent,reference,L) tuples included in its final destination scope. A visibility chain with a single element is just the visibility operation.

# Availability, Visibility, and Domain Operations

The following operations generate availability, visibility, and domain operations. When multiple availability/visibility/domain operations are described, they are system-synchronized-with each other in the order listed.

An operation that performs a memory dependency generates:

- If the source access mask includes `VK_ACCESS_HOST_WRITE_BIT`, then the dependency includes a memory domain operation from host domain to device domain.
- An availability operation with source scope of all writes in the first access scope of the dependency and a destination scope of the device domain.
- A visibility operation with source scope of the device domain and destination scope of the second access scope of the dependency.
- If the destination access mask includes `VK_ACCESS_HOST_READ_BIT` or `VK_ACCESS_HOST_WRITE_BIT`, then the dependency includes a memory domain operation from device domain to host domain.

vkFlushMappedMemoryRanges performs an availability operation, with a source scope of

(agents,references) = (all host threads, all mapped memory ranges passed to the command), and destination scope of the host domain.

vkInvalidateMappedMemoryRanges performs a visibility operation, with a source scope of the host domain and a destination scope of (agents,references) = (all host threads, all mapped memory ranges passed to the command).

vkQueueSubmit performs a memory domain operation from host to device, and a visibility operation with source scope of the device domain and destination scope of all agents and references on the device.

# Availability and Visibility Semantics

A memory barrier or atomic operation via agent A that includes MakeAvailable in its semantics performs an availability operation whose source scope includes agent A and all references in the storage classes in that instruction's storage class semantics, and all memory locations, and whose destination scope is a set of memory domains selected as specified below. The implicit availability operation is program-ordered between the barrier or atomic and all other operations program-ordered before the barrier or atomic.

A memory barrier or atomic operation via agent A that includes MakeVisible in its semantics performs a visibility operation whose source scope is a set of memory domains selected as specified below, and whose destination scope includes agent A and all references in the storage classes in that instruction's storage class semantics, and all memory locations. The implicit visibility operation is program-ordered between the barrier or atomic and all other operations program-ordered after the barrier or atomic.

The memory domains are selected based on the memory scope of the instruction as follows:

- `Device` scope uses the shader domain
- `QueueFamily` scope uses the queue family instance domain
- `FragmentInterlock` scope uses the fragment interlock instance domain
- `ShaderCallKHR` scope uses the shader call instance domain
- `Workgroup` scope uses the workgroup instance domain
- `Subgroup` uses the subgroup instance domain
- `Invocation` perform no availability/visibility operations.

When an availability operation performed by an agent A includes a memory domain D in its destination scope, where D corresponds to scope instance S, it also includes the memory domains that correspond to each smaller scope instance S' that is a subset of S and that includes A. Similarly for visibility operations.

# Per-Instruction Availability and Visibility Semantics

A memory write instruction that includes MakePointerAvailable, or an image write instruction that includes MakeTexelAvailable, performs an availability operation whose source scope includes the

agent and reference used to perform the write and the memory locations written by the instruction, and whose destination scope is a set of memory domains selected by the Scope operand specified in Availability and Visibility Semantics. The implicit availability operation is program-ordered between the write and all other operations program-ordered after the write.

A memory read instruction that includes MakePointerVisible, or an image read instruction that includes MakeTexelVisible, performs a visibility operation whose source scope is a set of memory domains selected by the Scope operand as specified in Availability and Visibility Semantics, and whose destination scope includes the agent and reference used to perform the read and the memory locations read by the instruction. The implicit visibility operation is program-ordered between read and all other operations program-ordered before the read.

| | |
|---|---|
| **NOTE** | Although reads with per-instruction visibility only perform visibility ops from the shader or fragment interlock instance or shader call instance or workgroup instance or subgroup instance domain, they will also see writes that were made visible via the device domain, i.e. those writes previously performed by non-shader agents and made visible via API commands. |

| | |
|---|---|
| **NOTE** | It is expected that all invocations in a subgroup execute on the same processor with the same path to memory, and thus availability and visibility operations with subgroup scope can be expected to be “free”. |

# Location-Ordered

Let X and Y be memory accesses to overlapping sets of memory locations M, where X != Y. Let $(A_X,R_X)$ be the agent and reference used for X, and $(A_Y,R_Y)$ be the agent and reference used for Y. For now, let “$\rightarrow$” denote happens-before and “$\rightarrow^{rcpo}$” denote the reflexive closure of program-ordered before.

If $D_1$ and $D_2$ are different memory domains, then let $DOM(D_1,D_2)$ be a memory domain operation from $D_1$ to $D_2$. Otherwise, let DOM(D,D) be a placeholder such that $X \rightarrow DOM(D,D) \rightarrow Y$ if and only if $X \rightarrow Y$.

X is *location-ordered* before Y for a location L in M if and only if any of the following is true:

- $A_X == A_Y$ and $R_X == R_Y$ and $X \rightarrow Y$
  - NOTE: this case means no availability/visibility ops are required when it is the same (agent,reference).
- X is a read, both X and Y are non-private, and $X \rightarrow Y$
- X is a read, and X (transitively) system-synchronizes with Y
- If $R_X == R_Y$ and $A_X$ and $A_Y$ access a common memory domain D (e.g. are in the same workgroup instance if D is the workgroup instance domain), and both X and Y are non-private:
  - X is a write, Y is a write, $AVC(A_X,R_X,D,L)$ is an availability chain making (X,L) available to domain D, and $X \rightarrow^{rcpo} AVC(A_X,R_X,D,L) \rightarrow Y$
  - X is a write, Y is a read, $AVC(A_X,R_X,D,L)$ is an availability chain making (X,L) available to domain D, $VISC(A_Y,R_Y,D,L)$ is a visibility chain making writes to L available in domain D

visible to Y, and $X \rightarrow^{rcpo} AVC(A_X,R_X,D,L) \rightarrow VISC(A_Y,R_Y,D,L) \rightarrow^{rcpo} Y$

  - If VkPhysicalDeviceVulkanMemoryModelFeatures::`vulkanMemoryModelAvailabilityVisibilityChains` is `VK_FALSE`, then AVC and VISC **must** each only have a single element in the chain, in each sub-bullet above.

- Let $D_X$ and $D_Y$ each be either the device domain or the host domain, depending on whether $A_X$ and $A_Y$ execute on the device or host:
  - X is a write and Y is a write, and $X \rightarrow AV(A_X,R_X,D_X,L) \rightarrow DOM(D_X,D_Y) \rightarrow Y$
  - X is a write and Y is a read, and $X \rightarrow AV(A_X,R_X,D_X,L) \rightarrow DOM(D_X,D_Y) \rightarrow VIS(A_Y,R_Y,D_Y,L) \rightarrow Y$

**NOTE** The final bullet (synchronization through device/host domain) requires API-level synchronization operations, since the device/host domains are not accessible via shader instructions. And “device domain” is not to be confused with “device scope”, which synchronizes through the “shader domain”.

# Data Race

Let X and Y be operations that access overlapping sets of memory locations M, where X != Y, and at least one of X and Y is a write, and X and Y are not mutually-ordered atomic operations. If there does not exist a location-ordered relation between X and Y for each location in M, then there is a *data race*.

Applications **must** ensure that no data races occur during the execution of their application.

**NOTE** Data races can only occur due to instructions that are actually executed. For example, an instruction skipped due to control flow must not contribute to a data race.

# Visible-To

Let X be a write and Y be a read whose sets of memory locations overlap, and let M be the set of memory locations that overlap. Let $M_2$ be a non-empty subset of M. Then X is *visible-to* Y for memory locations $M_2$ if and only if all of the following are true:

- X is location-ordered before Y for each location L in $M_2$.
- There does not exist another write Z to any location L in $M_2$ such that X is location-ordered before Z for location L and Z is location-ordered before Y for location L.

If X is visible-to Y, then Y reads the value written by X for locations $M_2$.

**NOTE** It is possible for there to be a write between X and Y that overwrites a subset of the memory locations, but the remaining memory locations ($M_2$) will still be visible-to Y.

# Acyclicity

*Reads-from* is a relation between operations, where the first operation is a write, the second operation is a read, and the second operation reads the value written by the first operation. *From-reads* is a relation between operations, where the first operation is a read, the second operation is a write, and the first operation reads a value written earlier than the second operation in the second operation's scoped modification order (or the first operation reads from the initial value, and the second operation is any write to the same locations).

Then the implementation **must** guarantee that no cycles exist in the union of the following relations:

- location-ordered
- scoped modification order (over all atomic writes)
- reads-from
- from-reads

| | |
|---|---|
| **NOTE** | This is a "consistency" axiom, which informally guarantees that sequences of operations cannot violate causality. |

### Scoped Modification Order Coherence

Let A and B be mutually-ordered atomic operations, where A is location-ordered before B. Then the following rules are a consequence of acyclicity:

- If A and B are both reads and A does not read the initial value, then the write that A takes its value from **must** be earlier in its own scoped modification order than (or the same as) the write that B takes its value from (no cycles between location-order, reads-from, and from-reads).
- If A is a read and B is a write and A does not read the initial value, then A **must** take its value from a write earlier than B in B's scoped modification order (no cycles between location-order, scope modification order, and reads-from).
- If A is a write and B is a read, then B **must** take its value from A or a write later than A in A's scoped modification order (no cycles between location-order, scoped modification order, and from-reads).
- If A and B are both writes, then A **must** be earlier than B in A's scoped modification order (no cycles between location-order and scoped modification order).
- If A is a write and B is a read-modify-write and B reads the value written by A, then B comes immediately after A in A's scoped modification order (no cycles between scoped modification order and from-reads).

# Shader I/O

If a shader invocation A in a shader stage other than `Vertex` performs a memory read operation X from an object in storage class `CallableDataKHR`, `IncomingCallableDataKHR`, `RayPayloadKHR`, `HitAttributeKHR`, `IncomingRayPayloadKHR`, or `Input`, then X is system-synchronized-after all writes to

the corresponding `CallableDataKHR`, `IncomingCallableDataKHR`, `RayPayloadKHR`, `HitAttributeKHR`, `IncomingRayPayloadKHR`, or `Output` storage variable(s) in the shader invocation(s) that contribute to generating invocation A, and those writes are all visible-to X.

| | |
|---|---|
| **NOTE** | It is not necessary for the upstream shader invocations to have completed execution, they only need to have generated the output that is being read. |

# Deallocation

A call to vkFreeMemory **must** happen-after all memory operations on all memory locations in that VkDeviceMemory object.

| | |
|---|---|
| **NOTE** | Normally, device memory operations in a given queue are synchronized with vkFreeMemory by having a host thread wait on a fence signaled by that queue, and the wait happens-before the call to vkFreeMemory on the host. |

The deallocation of SPIR-V variables is managed by the system and happens-after all operations on those variables.

# Descriptions (Informative)

This subsection offers more easily understandable consequences of the memory model for app/compiler developers.

Let SC be the storage class(es) specified by a release or acquire operation or barrier.

- An atomic write with release semantics must not be reordered against any read or write to SC that is program-ordered before it (regardless of the storage class the atomic is in).
- An atomic read with acquire semantics must not be reordered against any read or write to SC that is program-ordered after it (regardless of the storage class the atomic is in).
- Any write to SC program-ordered after a release barrier must not be reordered against any read or write to SC program-ordered before that barrier.
- Any read from SC program-ordered before an acquire barrier must not be reordered against any read or write to SC program-ordered after the barrier.

A control barrier (even if it has no memory semantics) must not be reordered against any memory barriers.

This memory model allows memory accesses with and without availability and visibility operations, as well as atomic operations, all to be performed on the same memory location. This is critical to allow it to reason about memory that is reused in multiple ways, e.g. across the lifetime of different shader invocations or draw calls. While GLSL (and legacy SPIR-V) applies the “coherent” decoration to variables (for historical reasons), this model treats each memory access instruction as having optional implicit availability/visibility operations. GLSL to SPIR-V compilers should map all (non-atomic) operations on a coherent variable to Make{Pointer,Texel}{Available}{Visible} flags in this model.

Atomic operations implicitly have availability/visibility operations, and the scope of those operations is taken from the atomic operation's scope.

# Tessellation Output Ordering

For SPIR-V that uses the Vulkan Memory Model, the `OutputMemory` storage class is used to synchronize accesses to tessellation control output variables. For legacy SPIR-V that does not enable the Vulkan Memory Model via `OpMemoryModel`, tessellation outputs can be ordered using a control barrier with no particular memory scope or semantics, as defined below.

Let X and Y be memory operations performed by shader invocations $A_X$ and $A_Y$. Operation X is *tessellation-output-ordered* before operation Y if and only if all of the following are true:

- There is a dynamic instance of an `OpControlBarrier` instruction C such that X is program-ordered before C in $A_X$ and C is program-ordered before Y in $A_Y$.
- $A_X$ and $A_Y$ are in the same instance of C's execution scope.

If shader invocations $A_X$ and $A_Y$ in the `TessellationControl` execution model execute memory operations X and Y, respectively, on the `Output` storage class, and X is tessellation-output-ordered before Y with a scope of `Workgroup`, then X is location-ordered before Y, and if X is a write and Y is a read then X is visible-to Y.

# Cooperative Matrix Memory Access

For each dynamic instance of a cooperative matrix load or store instruction (`OpCooperativeMatrixLoadNV` or `OpCooperativeMatrixStoreNV`), a single implementation-dependent invocation within the instance of the matrix's scope performs a non-atomic load or store (respectively) to each memory location that is defined to be accessed by the instruction.

# Appendix C: Compressed Image Formats

The compressed texture formats used by Vulkan are described in the specifically identified sections of the Khronos Data Format Specification, version 1.3.

Unless otherwise described, the quantities encoded in these compressed formats are treated as normalized, unsigned values.

Those formats listed as sRGB-encoded have in-memory representations of R, G and B components which are nonlinearly-encoded as R', G', and B'; any alpha component is unchanged. As part of filtering, the nonlinear R', G', and B' values are converted to linear R, G, and B components; any alpha component is unchanged. The conversion between linear and nonlinear encoding is performed as described in the “KHR_DF_TRANSFER_SRGB” section of the Khronos Data Format Specification.

# Block-Compressed Image Formats

BC1, BC2 and BC3 formats are described in “S3TC Compressed Texture Image Formats” chapter of the Khronos Data Format Specification. BC4 and BC5 are described in the “RGTC Compressed Texture Image Formats” chapter. BC6H and BC7 are described in the “BPTC Compressed Texture Image Formats” chapter.

*Table 102. Mapping of Vulkan BC formats to descriptions*

| VkFormat | Khronos Data Format Specification description |
|---|---|
| Formats described in the “S3TC Compressed Texture Image Formats” chapter | |
| VK_FORMAT_BC1_RGB_UNORM_BLOCK | BC1 with no alpha |
| VK_FORMAT_BC1_RGB_SRGB_BLOCK | BC1 with no alpha, sRGB-encoded |
| VK_FORMAT_BC1_RGBA_UNORM_BLOCK | BC1 with alpha |
| VK_FORMAT_BC1_RGBA_SRGB_BLOCK | BC1 with alpha, sRGB-encoded |
| VK_FORMAT_BC2_UNORM_BLOCK | BC2 |
| VK_FORMAT_BC2_SRGB_BLOCK | BC2, sRGB-encoded |
| VK_FORMAT_BC3_UNORM_BLOCK | BC3 |
| VK_FORMAT_BC3_SRGB_BLOCK | BC3, sRGB-encoded |
| Formats described in the “RGTC Compressed Texture Image Formats” chapter | |
| VK_FORMAT_BC4_UNORM_BLOCK | BC4 unsigned |
| VK_FORMAT_BC4_SNORM_BLOCK | BC4 signed |
| VK_FORMAT_BC5_UNORM_BLOCK | BC5 unsigned |
| VK_FORMAT_BC5_SNORM_BLOCK | BC5 signed |
| Formats described in the “BPTC Compressed Texture Image Formats” chapter | |
| VK_FORMAT_BC6H_UFLOAT_BLOCK | BC6H (unsigned version) |
| VK_FORMAT_BC6H_SFLOAT_BLOCK | BC6H (signed version) |
| VK_FORMAT_BC7_UNORM_BLOCK | BC7 |
| VK_FORMAT_BC7_SRGB_BLOCK | BC7, sRGB-encoded |

# ETC Compressed Image Formats

The following formats are described in the “ETC2 Compressed Texture Image Formats” chapter of the Khronos Data Format Specification.

*Table 103. Mapping of Vulkan ETC formats to descriptions*

| VkFormat | Khronos Data Format Specification description |
|---|---|
| `VK_FORMAT_ETC2_R8G8B8_UNORM_BLOCK` | RGB ETC2 |
| `VK_FORMAT_ETC2_R8G8B8_SRGB_BLOCK` | RGB ETC2 with sRGB encoding |
| `VK_FORMAT_ETC2_R8G8B8A1_UNORM_BLOCK` | RGB ETC2 with punch-through alpha |
| `VK_FORMAT_ETC2_R8G8B8A1_SRGB_BLOCK` | RGB ETC2 with punch-through alpha and sRGB |
| `VK_FORMAT_ETC2_R8G8B8A8_UNORM_BLOCK` | RGBA ETC2 |
| `VK_FORMAT_ETC2_R8G8B8A8_SRGB_BLOCK` | RGBA ETC2 with sRGB encoding |
| `VK_FORMAT_EAC_R11_UNORM_BLOCK` | Unsigned R11 EAC |
| `VK_FORMAT_EAC_R11_SNORM_BLOCK` | Signed R11 EAC |
| `VK_FORMAT_EAC_R11G11_UNORM_BLOCK` | Unsigned RG11 EAC |
| `VK_FORMAT_EAC_R11G11_SNORM_BLOCK` | Signed RG11 EAC |

# ASTC Compressed Image Formats

ASTC formats are described in the “ASTC Compressed Texture Image Formats” chapter of the Khronos Data Format Specification.

*Table 104. Mapping of Vulkan ASTC formats to descriptions*

| VkFormat | Compressed texel block dimensions | Requested mode |
|---|---|---|
| VK_FORMAT_ASTC_4x4_UNORM_BLOCK | 4 × 4 | Linear LDR |
| VK_FORMAT_ASTC_4x4_SRGB_BLOCK | 4 × 4 | sRGB |
| VK_FORMAT_ASTC_5x4_UNORM_BLOCK | 5 × 4 | Linear LDR |
| VK_FORMAT_ASTC_5x4_SRGB_BLOCK | 5 × 4 | sRGB |
| VK_FORMAT_ASTC_5x5_UNORM_BLOCK | 5 × 5 | Linear LDR |
| VK_FORMAT_ASTC_5x5_SRGB_BLOCK | 5 × 5 | sRGB |
| VK_FORMAT_ASTC_6x5_UNORM_BLOCK | 6 × 5 | Linear LDR |
| VK_FORMAT_ASTC_6x5_SRGB_BLOCK | 6 × 5 | sRGB |
| VK_FORMAT_ASTC_6x6_UNORM_BLOCK | 6 × 6 | Linear LDR |
| VK_FORMAT_ASTC_6x6_SRGB_BLOCK | 6 × 6 | sRGB |
| VK_FORMAT_ASTC_8x5_UNORM_BLOCK | 8 × 5 | Linear LDR |
| VK_FORMAT_ASTC_8x5_SRGB_BLOCK | 8 × 5 | sRGB |
| VK_FORMAT_ASTC_8x6_UNORM_BLOCK | 8 × 6 | Linear LDR |
| VK_FORMAT_ASTC_8x6_SRGB_BLOCK | 8 × 6 | sRGB |
| VK_FORMAT_ASTC_8x8_UNORM_BLOCK | 8 × 8 | Linear LDR |
| VK_FORMAT_ASTC_8x8_SRGB_BLOCK | 8 × 8 | sRGB |
| VK_FORMAT_ASTC_10x5_UNORM_BLOCK | 10 × 5 | Linear LDR |
| VK_FORMAT_ASTC_10x5_SRGB_BLOCK | 10 × 5 | sRGB |
| VK_FORMAT_ASTC_10x6_UNORM_BLOCK | 10 × 6 | Linear LDR |
| VK_FORMAT_ASTC_10x6_SRGB_BLOCK | 10 × 6 | sRGB |
| VK_FORMAT_ASTC_10x8_UNORM_BLOCK | 10 × 8 | Linear LDR |
| VK_FORMAT_ASTC_10x8_SRGB_BLOCK | 10 × 8 | sRGB |
| VK_FORMAT_ASTC_10x10_UNORM_BLOCK | 10 × 10 | Linear LDR |
| VK_FORMAT_ASTC_10x10_SRGB_BLOCK | 10 × 10 | sRGB |
| VK_FORMAT_ASTC_12x10_UNORM_BLOCK | 12 × 10 | Linear LDR |
| VK_FORMAT_ASTC_12x10_SRGB_BLOCK | 12 × 10 | sRGB |
| VK_FORMAT_ASTC_12x12_UNORM_BLOCK | 12 × 12 | Linear LDR |

| VkFormat | Compressed texel block dimensions | Requested mode |
|---|---|---|
| VK_FORMAT_ASTC_12x12_SRGB_BLOCK | 12 × 12 | sRGB |
| VK_FORMAT_ASTC_4x4_SFLOAT_BLOCK | 4 × 4 | HDR |
| VK_FORMAT_ASTC_5x4_SFLOAT_BLOCK | 5 × 4 | HDR |
| VK_FORMAT_ASTC_5x5_SFLOAT_BLOCK | 5 × 5 | HDR |
| VK_FORMAT_ASTC_6x5_SFLOAT_BLOCK | 6 × 5 | HDR |
| VK_FORMAT_ASTC_6x6_SFLOAT_BLOCK | 6 × 6 | HDR |
| VK_FORMAT_ASTC_8x5_SFLOAT_BLOCK | 8 × 5 | HDR |
| VK_FORMAT_ASTC_8x6_SFLOAT_BLOCK | 8 × 6 | HDR |
| VK_FORMAT_ASTC_8x8_SFLOAT_BLOCK | 8 × 8 | HDR |
| VK_FORMAT_ASTC_10x5_SFLOAT_BLOCK | 10 × 5 | HDR |
| VK_FORMAT_ASTC_10x6_SFLOAT_BLOCK | 10 × 6 | HDR |
| VK_FORMAT_ASTC_10x8_SFLOAT_BLOCK | 10 × 8 | HDR |
| VK_FORMAT_ASTC_10x10_SFLOAT_BLOCK | 10 × 10 | HDR |
| VK_FORMAT_ASTC_12x10_SFLOAT_BLOCK | 12 × 10 | HDR |
| VK_FORMAT_ASTC_12x12_SFLOAT_BLOCK | 12 × 12 | HDR |

ASTC textures containing HDR block encodings **should** be passed to the API using an ASTC SFLOAT texture format.

**NOTE** An HDR block in a texture passed using a LDR UNORM format will return the appropriate ASTC error color if the implementation supports only the ASTC LDR profile, but may result in either the error color or a decompressed HDR color if the implementation supports HDR decoding.

## ASTC Decode Mode

If the VK_EXT_astc_decode_mode extension is enabled, the decode mode is determined as follows:

*Table 105. Mapping of Vulkan ASTC decoding format to ASTC decoding modes*

| VkFormat | Decoding mode |
|---|---|
| VK_FORMAT_R16G16B16A16_SFLOAT | decode_float16 |
| VK_FORMAT_R8G8B8A8_UNORM | decode_unorm8 |
| VK_FORMAT_E5B9G9R9_UFLOAT_PACK32 | decode_rgb9e5 |

Otherwise, the ASTC decode mode is decode_float16.

Note that an implementation **may** use HDR mode when linear LDR mode is requested unless the decode mode is decode_unorm8.

# PVRTC Compressed Image Formats

PVRTC formats are described in the “PVRTC Compressed Texture Image Formats” chapter of the Khronos Data Format Specification.

*Table 106. Mapping of Vulkan PVRTC formats to descriptions*

| VkFormat | Compressed texel block dimensions | sRGB-encoded |
|---|---|---|
| VK_FORMAT_PVRTC1_2BPP_UNORM_BLOCK_IMG | 8 × 4 | No |
| VK_FORMAT_PVRTC1_4BPP_UNORM_BLOCK_IMG | 4 × 4 | No |
| VK_FORMAT_PVRTC2_2BPP_UNORM_BLOCK_IMG | 8 × 4 | No |
| VK_FORMAT_PVRTC2_4BPP_UNORM_BLOCK_IMG | 4 × 4 | No |
| VK_FORMAT_PVRTC1_2BPP_SRGB_BLOCK_IMG | 8 × 4 | Yes |
| VK_FORMAT_PVRTC1_4BPP_SRGB_BLOCK_IMG | 4 × 4 | Yes |
| VK_FORMAT_PVRTC2_2BPP_SRGB_BLOCK_IMG | 8 × 4 | Yes |
| VK_FORMAT_PVRTC2_4BPP_SRGB_BLOCK_IMG | 4 × 4 | Yes |

# Appendix D: Core Revisions (Informative)

New minor versions of the Vulkan API are defined periodically by the Khronos Vulkan Working Group. These consist of some amount of additional functionality added to the core API, potentially including both new functionality and functionality promoted from extensions.

It is possible to build the specification for earlier versions, but to aid readability of the latest versions, this appendix gives an overview of the changes as compared to earlier versions.

## Vulkan Version 1.1

Vulkan Version 1.1 promoted a number of key extensions into the core API:

- VK_KHR_16bit_storage
- VK_KHR_bind_memory2
- VK_KHR_dedicated_allocation
- VK_KHR_descriptor_update_template
- VK_KHR_device_group
- VK_KHR_device_group_creation
- VK_KHR_external_fence
- VK_KHR_external_fence_capabilities
- VK_KHR_external_memory
- VK_KHR_external_memory_capabilities
- VK_KHR_external_semaphore
- VK_KHR_external_semaphore_capabilities
- VK_KHR_get_memory_requirements2
- VK_KHR_get_physical_device_properties2
- VK_KHR_maintenance1
- VK_KHR_maintenance2
- VK_KHR_maintenance3
- VK_KHR_multiview
- VK_KHR_relaxed_block_layout
- VK_KHR_sampler_ycbcr_conversion

- VK_KHR_shader_draw_parameters
- VK_KHR_storage_buffer_storage_class
- VK_KHR_variable_pointers

All differences in behavior between these extensions and the corresponding Vulkan 1.1 functionality are summarized below.

### Differences Relative to VK_KHR_16bit_storage

If the VK_KHR_16bit_storage extension is not supported, support for the storageBuffer16BitAccess feature is optional. Support for this feature is defined by VkPhysicalDevice16BitStorageFeatures::storageBuffer16BitAccess when queried via vkGetPhysicalDeviceFeatures2.

### Differences Relative to VK_KHR_sampler_ycbcr_conversion

If the VK_KHR_sampler_ycbcr_conversion extension is not supported, support for the samplerYcbcrConversion feature is optional. Support for this feature is defined by VkPhysicalDeviceSamplerYcbcrConversionFeatures::samplerYcbcrConversion when queried via vkGetPhysicalDeviceFeatures2.

### Differences Relative to VK_KHR_shader_draw_parameters

If the VK_KHR_shader_draw_parameters extension is not supported, support for the SPV_KHR_shader_draw_parameters SPIR-V extension is optional. Support for this feature is defined by VkPhysicalDeviceShaderDrawParametersFeatures::shaderDrawParameters when queried via vkGetPhysicalDeviceFeatures2.

### Differences Relative to VK_KHR_variable_pointers

If the VK_KHR_variable_pointers extension is not supported, support for the variablePointersStorageBuffer feature is optional. Support for this feature is defined by VkPhysicalDeviceVariablePointersFeatures::variablePointersStorageBuffer when queried via vkGetPhysicalDeviceFeatures2.

### Additional Vulkan 1.1 Feature Support

In addition to the promoted extensions described above, Vulkan 1.1 added support for:

- The group operations and subgroup scope.
- The protected memory feature.
- A new command to enumerate the instance version: vkEnumerateInstanceVersion.
- The VkPhysicalDeviceShaderDrawParametersFeatures feature query struct (where the VK_KHR_shader_draw_parameters extension did not have one).

## New Macros

- VK_API_VERSION_1_1

## New Object Types

- VkDescriptorUpdateTemplate
- VkSamplerYcbcrConversion

## New Commands

- vkBindBufferMemory2
- vkBindImageMemory2
- vkCmdDispatchBase
- vkCmdSetDeviceMask
- vkCreateDescriptorUpdateTemplate
- vkCreateSamplerYcbcrConversion
- vkDestroyDescriptorUpdateTemplate
- vkDestroySamplerYcbcrConversion
- vkEnumerateInstanceVersion
- vkEnumeratePhysicalDeviceGroups
- vkGetBufferMemoryRequirements2
- vkGetDescriptorSetLayoutSupport
- vkGetDeviceGroupPeerMemoryFeatures
- vkGetDeviceQueue2
- vkGetImageMemoryRequirements2
- vkGetImageSparseMemoryRequirements2
- vkGetPhysicalDeviceExternalBufferProperties
- vkGetPhysicalDeviceExternalFenceProperties
- vkGetPhysicalDeviceExternalSemaphoreProperties
- vkGetPhysicalDeviceFeatures2
- vkGetPhysicalDeviceFormatProperties2
- vkGetPhysicalDeviceImageFormatProperties2
- vkGetPhysicalDeviceMemoryProperties2
- vkGetPhysicalDeviceProperties2
- vkGetPhysicalDeviceQueueFamilyProperties2
- vkGetPhysicalDeviceSparseImageFormatProperties2
- vkTrimCommandPool

- vkUpdateDescriptorSetWithTemplate

## New Structures

- VkBindBufferMemoryInfo
- VkBindImageMemoryInfo
- VkBufferMemoryRequirementsInfo2
- VkDescriptorSetLayoutSupport
- VkDescriptorUpdateTemplateCreateInfo
- VkDescriptorUpdateTemplateEntry
- VkDeviceQueueInfo2
- VkExternalBufferProperties
- VkExternalFenceProperties
- VkExternalMemoryProperties
- VkExternalSemaphoreProperties
- VkFormatProperties2
- VkImageFormatProperties2
- VkImageMemoryRequirementsInfo2
- VkImageSparseMemoryRequirementsInfo2
- VkInputAttachmentAspectReference
- VkMemoryRequirements2
- VkPhysicalDeviceExternalBufferInfo
- VkPhysicalDeviceExternalFenceInfo
- VkPhysicalDeviceExternalSemaphoreInfo
- VkPhysicalDeviceGroupProperties
- VkPhysicalDeviceImageFormatInfo2
- VkPhysicalDeviceMemoryProperties2
- VkPhysicalDeviceProperties2
- VkPhysicalDeviceSparseImageFormatInfo2
- VkQueueFamilyProperties2
- VkSamplerYcbcrConversionCreateInfo
- VkSparseImageFormatProperties2
- VkSparseImageMemoryRequirements2
- Extending VkBindBufferMemoryInfo:
  - VkBindBufferMemoryDeviceGroupInfo
- Extending VkBindImageMemoryInfo:

    - VkBindImageMemoryDeviceGroupInfo
    - VkBindImagePlaneMemoryInfo
- Extending VkBindSparseInfo:
    - VkDeviceGroupBindSparseInfo
- Extending VkBufferCreateInfo:
    - VkExternalMemoryBufferCreateInfo
- Extending VkCommandBufferBeginInfo:
    - VkDeviceGroupCommandBufferBeginInfo
- Extending VkDeviceCreateInfo:
    - VkDeviceGroupDeviceCreateInfo
    - VkPhysicalDeviceFeatures2
- Extending VkFenceCreateInfo:
    - VkExportFenceCreateInfo
- Extending VkImageCreateInfo:
    - VkExternalMemoryImageCreateInfo
- Extending VkImageFormatProperties2:
    - VkExternalImageFormatProperties
    - VkSamplerYcbcrConversionImageFormatProperties
- Extending VkImageMemoryRequirementsInfo2:
    - VkImagePlaneMemoryRequirementsInfo
- Extending VkImageViewCreateInfo:
    - VkImageViewUsageCreateInfo
- Extending VkMemoryAllocateInfo:
    - VkExportMemoryAllocateInfo
    - VkMemoryAllocateFlagsInfo
    - VkMemoryDedicatedAllocateInfo
- Extending VkMemoryRequirements2:
    - VkMemoryDedicatedRequirements
- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
    - VkPhysicalDevice16BitStorageFeatures
    - VkPhysicalDeviceMultiviewFeatures
    - VkPhysicalDeviceProtectedMemoryFeatures
    - VkPhysicalDeviceSamplerYcbcrConversionFeatures
    - VkPhysicalDeviceShaderDrawParameterFeatures
    - VkPhysicalDeviceShaderDrawParametersFeatures

    - VkPhysicalDeviceVariablePointerFeatures
    - VkPhysicalDeviceVariablePointersFeatures
- Extending VkPhysicalDeviceImageFormatInfo2:
    - VkPhysicalDeviceExternalImageFormatInfo
- Extending VkPhysicalDeviceProperties2:
    - VkPhysicalDeviceIDProperties
    - VkPhysicalDeviceMaintenance3Properties
    - VkPhysicalDeviceMultiviewProperties
    - VkPhysicalDevicePointClippingProperties
    - VkPhysicalDeviceProtectedMemoryProperties
    - VkPhysicalDeviceSubgroupProperties
- Extending VkPipelineTessellationStateCreateInfo:
    - VkPipelineTessellationDomainOriginStateCreateInfo
- Extending VkRenderPassBeginInfo, VkRenderingInfo:
    - VkDeviceGroupRenderPassBeginInfo
- Extending VkRenderPassCreateInfo:
    - VkRenderPassInputAttachmentAspectCreateInfo
    - VkRenderPassMultiviewCreateInfo
- Extending VkSamplerCreateInfo, VkImageViewCreateInfo:
    - VkSamplerYcbcrConversionInfo
- Extending VkSemaphoreCreateInfo:
    - VkExportSemaphoreCreateInfo
- Extending VkSubmitInfo:
    - VkDeviceGroupSubmitInfo
    - VkProtectedSubmitInfo

## New Enums

- VkChromaLocation
- VkDescriptorUpdateTemplateType
- VkDeviceQueueCreateFlagBits
- VkExternalFenceFeatureFlagBits
- VkExternalFenceHandleTypeFlagBits
- VkExternalMemoryFeatureFlagBits
- VkExternalMemoryHandleTypeFlagBits
- VkExternalSemaphoreFeatureFlagBits

- VkExternalSemaphoreHandleTypeFlagBits
- VkFenceImportFlagBits
- VkMemoryAllocateFlagBits
- VkPeerMemoryFeatureFlagBits
- VkPointClippingBehavior
- VkSamplerYcbcrModelConversion
- VkSamplerYcbcrRange
- VkSemaphoreImportFlagBits
- VkSubgroupFeatureFlagBits
- VkTessellationDomainOrigin

## New Bitmasks

- VkCommandPoolTrimFlags
- VkDescriptorUpdateTemplateCreateFlags
- VkExternalFenceFeatureFlags
- VkExternalFenceHandleTypeFlags
- VkExternalMemoryFeatureFlags
- VkExternalMemoryHandleTypeFlags
- VkExternalSemaphoreFeatureFlags
- VkExternalSemaphoreHandleTypeFlags
- VkFenceImportFlags
- VkMemoryAllocateFlags
- VkPeerMemoryFeatureFlags
- VkSemaphoreImportFlags
- VkSubgroupFeatureFlags

## New Enum Constants

- `VK_LUID_SIZE`
- `VK_MAX_DEVICE_GROUP_SIZE`
- `VK_QUEUE_FAMILY_EXTERNAL`
- Extending VkBufferCreateFlagBits:
  - `VK_BUFFER_CREATE_PROTECTED_BIT`
- Extending VkCommandPoolCreateFlagBits:
  - `VK_COMMAND_POOL_CREATE_PROTECTED_BIT`
- Extending VkDependencyFlagBits:

  - VK_DEPENDENCY_DEVICE_GROUP_BIT
  - VK_DEPENDENCY_VIEW_LOCAL_BIT
- Extending VkDeviceQueueCreateFlagBits:
  - VK_DEVICE_QUEUE_CREATE_PROTECTED_BIT
- Extending VkFormat:
  - VK_FORMAT_B10X6G10X6R10X6G10X6_422_UNORM_4PACK16
  - VK_FORMAT_B12X4G12X4R12X4G12X4_422_UNORM_4PACK16
  - VK_FORMAT_B16G16R16G16_422_UNORM
  - VK_FORMAT_B8G8R8G8_422_UNORM
  - VK_FORMAT_G10X6B10X6G10X6R10X6_422_UNORM_4PACK16
  - VK_FORMAT_G10X6_B10X6R10X6_2PLANE_420_UNORM_3PACK16
  - VK_FORMAT_G10X6_B10X6R10X6_2PLANE_422_UNORM_3PACK16
  - VK_FORMAT_G10X6_B10X6_R10X6_3PLANE_420_UNORM_3PACK16
  - VK_FORMAT_G10X6_B10X6_R10X6_3PLANE_422_UNORM_3PACK16
  - VK_FORMAT_G10X6_B10X6_R10X6_3PLANE_444_UNORM_3PACK16
  - VK_FORMAT_G12X4B12X4G12X4R12X4_422_UNORM_4PACK16
  - VK_FORMAT_G12X4_B12X4R12X4_2PLANE_420_UNORM_3PACK16
  - VK_FORMAT_G12X4_B12X4R12X4_2PLANE_422_UNORM_3PACK16
  - VK_FORMAT_G12X4_B12X4_R12X4_3PLANE_420_UNORM_3PACK16
  - VK_FORMAT_G12X4_B12X4_R12X4_3PLANE_422_UNORM_3PACK16
  - VK_FORMAT_G12X4_B12X4_R12X4_3PLANE_444_UNORM_3PACK16
  - VK_FORMAT_G16B16G16R16_422_UNORM
  - VK_FORMAT_G16_B16R16_2PLANE_420_UNORM
  - VK_FORMAT_G16_B16R16_2PLANE_422_UNORM
  - VK_FORMAT_G16_B16_R16_3PLANE_420_UNORM
  - VK_FORMAT_G16_B16_R16_3PLANE_422_UNORM
  - VK_FORMAT_G16_B16_R16_3PLANE_444_UNORM
  - VK_FORMAT_G8B8G8R8_422_UNORM
  - VK_FORMAT_G8_B8R8_2PLANE_420_UNORM
  - VK_FORMAT_G8_B8R8_2PLANE_422_UNORM
  - VK_FORMAT_G8_B8_R8_3PLANE_420_UNORM
  - VK_FORMAT_G8_B8_R8_3PLANE_422_UNORM
  - VK_FORMAT_G8_B8_R8_3PLANE_444_UNORM
  - VK_FORMAT_R10X6G10X6B10X6A10X6_UNORM_4PACK16
  - VK_FORMAT_R10X6G10X6_UNORM_2PACK16

    - VK_FORMAT_R10X6_UNORM_PACK16
    - VK_FORMAT_R12X4G12X4B12X4A12X4_UNORM_4PACK16
    - VK_FORMAT_R12X4G12X4_UNORM_2PACK16
    - VK_FORMAT_R12X4_UNORM_PACK16
- Extending VkFormatFeatureFlagBits:
    - VK_FORMAT_FEATURE_COSITED_CHROMA_SAMPLES_BIT
    - VK_FORMAT_FEATURE_DISJOINT_BIT
    - VK_FORMAT_FEATURE_MIDPOINT_CHROMA_SAMPLES_BIT
    - VK_FORMAT_FEATURE_SAMPLED_IMAGE_YCBCR_CONVERSION_CHROMA_RECONSTRUCTION_EXPLICIT_BIT
    - VK_FORMAT_FEATURE_SAMPLED_IMAGE_YCBCR_CONVERSION_CHROMA_RECONSTRUCTION_EXPLICIT_FORCEABLE_BIT
    - VK_FORMAT_FEATURE_SAMPLED_IMAGE_YCBCR_CONVERSION_LINEAR_FILTER_BIT
    - VK_FORMAT_FEATURE_SAMPLED_IMAGE_YCBCR_CONVERSION_SEPARATE_RECONSTRUCTION_FILTER_BIT
    - VK_FORMAT_FEATURE_TRANSFER_DST_BIT
    - VK_FORMAT_FEATURE_TRANSFER_SRC_BIT
- Extending VkImageAspectFlagBits:
    - VK_IMAGE_ASPECT_PLANE_0_BIT
    - VK_IMAGE_ASPECT_PLANE_1_BIT
    - VK_IMAGE_ASPECT_PLANE_2_BIT
- Extending VkImageCreateFlagBits:
    - VK_IMAGE_CREATE_2D_ARRAY_COMPATIBLE_BIT
    - VK_IMAGE_CREATE_ALIAS_BIT
    - VK_IMAGE_CREATE_BLOCK_TEXEL_VIEW_COMPATIBLE_BIT
    - VK_IMAGE_CREATE_DISJOINT_BIT
    - VK_IMAGE_CREATE_EXTENDED_USAGE_BIT
    - VK_IMAGE_CREATE_PROTECTED_BIT
    - VK_IMAGE_CREATE_SPLIT_INSTANCE_BIND_REGIONS_BIT
- Extending VkImageLayout:
    - VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_STENCIL_READ_ONLY_OPTIMAL
    - VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_STENCIL_ATTACHMENT_OPTIMAL
- Extending VkMemoryHeapFlagBits:
    - VK_MEMORY_HEAP_MULTI_INSTANCE_BIT
- Extending VkMemoryPropertyFlagBits:
    - VK_MEMORY_PROPERTY_PROTECTED_BIT
- Extending VkObjectType:

    - VK_OBJECT_TYPE_DESCRIPTOR_UPDATE_TEMPLATE
    - VK_OBJECT_TYPE_SAMPLER_YCBCR_CONVERSION
- Extending VkPipelineCreateFlagBits:
    - VK_PIPELINE_CREATE_DISPATCH_BASE
    - VK_PIPELINE_CREATE_DISPATCH_BASE_BIT
    - VK_PIPELINE_CREATE_VIEW_INDEX_FROM_DEVICE_INDEX_BIT
- Extending VkQueueFlagBits:
    - VK_QUEUE_PROTECTED_BIT
- Extending VkResult:
    - VK_ERROR_INVALID_EXTERNAL_HANDLE
    - VK_ERROR_OUT_OF_POOL_MEMORY
- Extending VkStructureType:
    - VK_STRUCTURE_TYPE_BIND_BUFFER_MEMORY_DEVICE_GROUP_INFO
    - VK_STRUCTURE_TYPE_BIND_BUFFER_MEMORY_INFO
    - VK_STRUCTURE_TYPE_BIND_IMAGE_MEMORY_DEVICE_GROUP_INFO
    - VK_STRUCTURE_TYPE_BIND_IMAGE_MEMORY_INFO
    - VK_STRUCTURE_TYPE_BIND_IMAGE_PLANE_MEMORY_INFO
    - VK_STRUCTURE_TYPE_BUFFER_MEMORY_REQUIREMENTS_INFO_2
    - VK_STRUCTURE_TYPE_DESCRIPTOR_SET_LAYOUT_SUPPORT
    - VK_STRUCTURE_TYPE_DESCRIPTOR_UPDATE_TEMPLATE_CREATE_INFO
    - VK_STRUCTURE_TYPE_DEVICE_GROUP_BIND_SPARSE_INFO
    - VK_STRUCTURE_TYPE_DEVICE_GROUP_COMMAND_BUFFER_BEGIN_INFO
    - VK_STRUCTURE_TYPE_DEVICE_GROUP_DEVICE_CREATE_INFO
    - VK_STRUCTURE_TYPE_DEVICE_GROUP_RENDER_PASS_BEGIN_INFO
    - VK_STRUCTURE_TYPE_DEVICE_GROUP_SUBMIT_INFO
    - VK_STRUCTURE_TYPE_DEVICE_QUEUE_INFO_2
    - VK_STRUCTURE_TYPE_EXPORT_FENCE_CREATE_INFO
    - VK_STRUCTURE_TYPE_EXPORT_MEMORY_ALLOCATE_INFO
    - VK_STRUCTURE_TYPE_EXPORT_SEMAPHORE_CREATE_INFO
    - VK_STRUCTURE_TYPE_EXTERNAL_BUFFER_PROPERTIES
    - VK_STRUCTURE_TYPE_EXTERNAL_FENCE_PROPERTIES
    - VK_STRUCTURE_TYPE_EXTERNAL_IMAGE_FORMAT_PROPERTIES
    - VK_STRUCTURE_TYPE_EXTERNAL_MEMORY_BUFFER_CREATE_INFO
    - VK_STRUCTURE_TYPE_EXTERNAL_MEMORY_IMAGE_CREATE_INFO
    - VK_STRUCTURE_TYPE_EXTERNAL_SEMAPHORE_PROPERTIES

- VK_STRUCTURE_TYPE_FORMAT_PROPERTIES_2
- VK_STRUCTURE_TYPE_IMAGE_FORMAT_PROPERTIES_2
- VK_STRUCTURE_TYPE_IMAGE_MEMORY_REQUIREMENTS_INFO_2
- VK_STRUCTURE_TYPE_IMAGE_PLANE_MEMORY_REQUIREMENTS_INFO
- VK_STRUCTURE_TYPE_IMAGE_SPARSE_MEMORY_REQUIREMENTS_INFO_2
- VK_STRUCTURE_TYPE_IMAGE_VIEW_USAGE_CREATE_INFO
- VK_STRUCTURE_TYPE_MEMORY_ALLOCATE_FLAGS_INFO
- VK_STRUCTURE_TYPE_MEMORY_DEDICATED_ALLOCATE_INFO
- VK_STRUCTURE_TYPE_MEMORY_DEDICATED_REQUIREMENTS
- VK_STRUCTURE_TYPE_MEMORY_REQUIREMENTS_2
- VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_16BIT_STORAGE_FEATURES
- VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTERNAL_BUFFER_INFO
- VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTERNAL_FENCE_INFO
- VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTERNAL_IMAGE_FORMAT_INFO
- VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTERNAL_SEMAPHORE_INFO
- VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_FEATURES_2
- VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_GROUP_PROPERTIES
- VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_ID_PROPERTIES
- VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_IMAGE_FORMAT_INFO_2
- VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MAINTENANCE_3_PROPERTIES
- VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MEMORY_PROPERTIES_2
- VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MULTIVIEW_FEATURES
- VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MULTIVIEW_PROPERTIES
- VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_POINT_CLIPPING_PROPERTIES
- VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PROPERTIES_2
- VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PROTECTED_MEMORY_FEATURES
- VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PROTECTED_MEMORY_PROPERTIES
- VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SAMPLER_YCBCR_CONVERSION_FEATURES
- VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_DRAW_PARAMETERS_FEATURES
- VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_DRAW_PARAMETER_FEATURES
- VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SPARSE_IMAGE_FORMAT_INFO_2
- VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SUBGROUP_PROPERTIES
- VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_VARIABLE_POINTERS_FEATURES
- VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_VARIABLE_POINTER_FEATURES
- VK_STRUCTURE_TYPE_PIPELINE_TESSELLATION_DOMAIN_ORIGIN_STATE_CREATE_INFO

- VK_STRUCTURE_TYPE_PROTECTED_SUBMIT_INFO
- VK_STRUCTURE_TYPE_QUEUE_FAMILY_PROPERTIES_2
- VK_STRUCTURE_TYPE_RENDER_PASS_INPUT_ATTACHMENT_ASPECT_CREATE_INFO
- VK_STRUCTURE_TYPE_RENDER_PASS_MULTIVIEW_CREATE_INFO
- VK_STRUCTURE_TYPE_SAMPLER_YCBCR_CONVERSION_CREATE_INFO
- VK_STRUCTURE_TYPE_SAMPLER_YCBCR_CONVERSION_IMAGE_FORMAT_PROPERTIES
- VK_STRUCTURE_TYPE_SAMPLER_YCBCR_CONVERSION_INFO
- VK_STRUCTURE_TYPE_SPARSE_IMAGE_FORMAT_PROPERTIES_2
- VK_STRUCTURE_TYPE_SPARSE_IMAGE_MEMORY_REQUIREMENTS_2

# Vulkan Version 1.0

Vulkan Version 1.0 was the initial release of the Vulkan API.

## New Macros

- VK_API_VERSION
- VK_API_VERSION_1_0
- VK_API_VERSION_MAJOR
- VK_API_VERSION_MINOR
- VK_API_VERSION_PATCH
- VK_API_VERSION_VARIANT
- VK_DEFINE_HANDLE
- VK_DEFINE_NON_DISPATCHABLE_HANDLE
- VK_HEADER_VERSION
- VK_HEADER_VERSION_COMPLETE
- VK_MAKE_API_VERSION
- VK_MAKE_VERSION
- VK_NULL_HANDLE
- VK_USE_64_BIT_PTR_DEFINES
- VK_VERSION_MAJOR
- VK_VERSION_MINOR
- VK_VERSION_PATCH

## New Base Types

- VkBool32
- VkDeviceAddress

- VkDeviceSize
- VkFlags
- VkSampleMask

## New Object Types

- VkBuffer
- VkBufferView
- VkCommandBuffer
- VkCommandPool
- VkDescriptorPool
- VkDescriptorSet
- VkDescriptorSetLayout
- VkDevice
- VkDeviceMemory
- VkEvent
- VkFence
- VkFramebuffer
- VkImage
- VkImageView
- VkInstance
- VkPhysicalDevice
- VkPipeline
- VkPipelineCache
- VkPipelineLayout
- VkQueryPool
- VkQueue
- VkRenderPass
- VkSampler
- VkSemaphore
- VkShaderModule

## New Commands

- vkAllocateCommandBuffers
- vkAllocateDescriptorSets
- vkAllocateMemory

- vkBeginCommandBuffer
- vkBindBufferMemory
- vkBindImageMemory
- vkCmdBeginQuery
- vkCmdBeginRenderPass
- vkCmdBindDescriptorSets
- vkCmdBindIndexBuffer
- vkCmdBindPipeline
- vkCmdBindVertexBuffers
- vkCmdBlitImage
- vkCmdClearAttachments
- vkCmdClearColorImage
- vkCmdClearDepthStencilImage
- vkCmdCopyBuffer
- vkCmdCopyBufferToImage
- vkCmdCopyImage
- vkCmdCopyImageToBuffer
- vkCmdCopyQueryPoolResults
- vkCmdDispatch
- vkCmdDispatchIndirect
- vkCmdDraw
- vkCmdDrawIndexed
- vkCmdDrawIndexedIndirect
- vkCmdDrawIndirect
- vkCmdEndQuery
- vkCmdEndRenderPass
- vkCmdExecuteCommands
- vkCmdFillBuffer
- vkCmdNextSubpass
- vkCmdPipelineBarrier
- vkCmdPushConstants
- vkCmdResetEvent
- vkCmdResetQueryPool
- vkCmdResolveImage
- vkCmdSetBlendConstants

- vkCmdSetDepthBias
- vkCmdSetDepthBounds
- vkCmdSetEvent
- vkCmdSetLineWidth
- vkCmdSetScissor
- vkCmdSetStencilCompareMask
- vkCmdSetStencilReference
- vkCmdSetStencilWriteMask
- vkCmdSetViewport
- vkCmdUpdateBuffer
- vkCmdWaitEvents
- vkCmdWriteTimestamp
- vkCreateBuffer
- vkCreateBufferView
- vkCreateCommandPool
- vkCreateComputePipelines
- vkCreateDescriptorPool
- vkCreateDescriptorSetLayout
- vkCreateDevice
- vkCreateEvent
- vkCreateFence
- vkCreateFramebuffer
- vkCreateGraphicsPipelines
- vkCreateImage
- vkCreateImageView
- vkCreateInstance
- vkCreatePipelineCache
- vkCreatePipelineLayout
- vkCreateQueryPool
- vkCreateRenderPass
- vkCreateSampler
- vkCreateSemaphore
- vkCreateShaderModule
- vkDestroyBuffer
- vkDestroyBufferView

- vkDestroyCommandPool
- vkDestroyDescriptorPool
- vkDestroyDescriptorSetLayout
- vkDestroyDevice
- vkDestroyEvent
- vkDestroyFence
- vkDestroyFramebuffer
- vkDestroyImage
- vkDestroyImageView
- vkDestroyInstance
- vkDestroyPipeline
- vkDestroyPipelineCache
- vkDestroyPipelineLayout
- vkDestroyQueryPool
- vkDestroyRenderPass
- vkDestroySampler
- vkDestroySemaphore
- vkDestroyShaderModule
- vkDeviceWaitIdle
- vkEndCommandBuffer
- vkEnumerateDeviceExtensionProperties
- vkEnumerateDeviceLayerProperties
- vkEnumerateInstanceExtensionProperties
- vkEnumerateInstanceLayerProperties
- vkEnumeratePhysicalDevices
- vkFlushMappedMemoryRanges
- vkFreeCommandBuffers
- vkFreeDescriptorSets
- vkFreeMemory
- vkGetBufferMemoryRequirements
- vkGetDeviceMemoryCommitment
- vkGetDeviceProcAddr
- vkGetDeviceQueue
- vkGetEventStatus
- vkGetFenceStatus

- vkGetImageMemoryRequirements
- vkGetImageSparseMemoryRequirements
- vkGetImageSubresourceLayout
- vkGetInstanceProcAddr
- vkGetPhysicalDeviceFeatures
- vkGetPhysicalDeviceFormatProperties
- vkGetPhysicalDeviceImageFormatProperties
- vkGetPhysicalDeviceMemoryProperties
- vkGetPhysicalDeviceProperties
- vkGetPhysicalDeviceQueueFamilyProperties
- vkGetPhysicalDeviceSparseImageFormatProperties
- vkGetPipelineCacheData
- vkGetQueryPoolResults
- vkGetRenderAreaGranularity
- vkInvalidateMappedMemoryRanges
- vkMapMemory
- vkMergePipelineCaches
- vkQueueBindSparse
- vkQueueSubmit
- vkQueueWaitIdle
- vkResetCommandBuffer
- vkResetCommandPool
- vkResetDescriptorPool
- vkResetEvent
- vkResetFences
- vkSetEvent
- vkUnmapMemory
- vkUpdateDescriptorSets
- vkWaitForFences

## New Structures

- VkAllocationCallbacks
- VkApplicationInfo
- VkAttachmentDescription
- VkAttachmentReference

- VkBaseInStructure
- VkBaseOutStructure
- VkBindSparseInfo
- VkBufferCopy
- VkBufferCreateInfo
- VkBufferImageCopy
- VkBufferMemoryBarrier
- VkBufferViewCreateInfo
- VkClearAttachment
- VkClearDepthStencilValue
- VkClearRect
- VkCommandBufferAllocateInfo
- VkCommandBufferBeginInfo
- VkCommandBufferInheritanceInfo
- VkCommandPoolCreateInfo
- VkComponentMapping
- VkComputePipelineCreateInfo
- VkCopyDescriptorSet
- VkDescriptorBufferInfo
- VkDescriptorImageInfo
- VkDescriptorPoolCreateInfo
- VkDescriptorPoolSize
- VkDescriptorSetAllocateInfo
- VkDescriptorSetLayoutBinding
- VkDescriptorSetLayoutCreateInfo
- VkDeviceCreateInfo
- VkDeviceQueueCreateInfo
- VkDispatchIndirectCommand
- VkDrawIndexedIndirectCommand
- VkDrawIndirectCommand
- VkEventCreateInfo
- VkExtensionProperties
- VkExtent2D
- VkExtent3D
- VkFenceCreateInfo

- VkFormatProperties
- VkFramebufferCreateInfo
- VkGraphicsPipelineCreateInfo
- VkImageBlit
- VkImageCopy
- VkImageCreateInfo
- VkImageFormatProperties
- VkImageMemoryBarrier
- VkImageResolve
- VkImageSubresource
- VkImageSubresourceLayers
- VkImageSubresourceRange
- VkImageViewCreateInfo
- VkInstanceCreateInfo
- VkLayerProperties
- VkMappedMemoryRange
- VkMemoryAllocateInfo
- VkMemoryBarrier
- VkMemoryHeap
- VkMemoryRequirements
- VkMemoryType
- VkOffset2D
- VkOffset3D
- VkPhysicalDeviceFeatures
- VkPhysicalDeviceLimits
- VkPhysicalDeviceMemoryProperties
- VkPhysicalDeviceProperties
- VkPhysicalDeviceSparseProperties
- VkPipelineCacheCreateInfo
- VkPipelineCacheHeaderVersionOne
- VkPipelineColorBlendAttachmentState
- VkPipelineColorBlendStateCreateInfo
- VkPipelineDepthStencilStateCreateInfo
- VkPipelineDynamicStateCreateInfo
- VkPipelineInputAssemblyStateCreateInfo

- VkPipelineMultisampleStateCreateInfo
- VkPipelineRasterizationStateCreateInfo
- VkPipelineShaderStageCreateInfo
- VkPipelineTessellationStateCreateInfo
- VkPipelineVertexInputStateCreateInfo
- VkPipelineViewportStateCreateInfo
- VkPushConstantRange
- VkQueryPoolCreateInfo
- VkQueueFamilyProperties
- VkRect2D
- VkRenderPassBeginInfo
- VkRenderPassCreateInfo
- VkSamplerCreateInfo
- VkSemaphoreCreateInfo
- VkSparseBufferMemoryBindInfo
- VkSparseImageFormatProperties
- VkSparseImageMemoryBind
- VkSparseImageMemoryBindInfo
- VkSparseImageMemoryRequirements
- VkSparseImageOpaqueMemoryBindInfo
- VkSparseMemoryBind
- VkSpecializationInfo
- VkSpecializationMapEntry
- VkStencilOpState
- VkSubmitInfo
- VkSubpassDependency
- VkSubpassDescription
- VkSubresourceLayout
- VkVertexInputAttributeDescription
- VkVertexInputBindingDescription
- VkViewport
- VkWriteDescriptorSet
- Extending VkBindDescriptorSetsInfoKHR, VkPushConstantsInfoKHR, VkPushDescriptorSetInfoKHR, VkPushDescriptorSetWithTemplateInfoKHR, VkSetDescriptorBufferOffsetsInfoEXT, VkBindDescriptorBufferEmbeddedSamplersInfoEXT:

    - VkPipelineLayoutCreateInfo
- Extending VkPipelineShaderStageCreateInfo:
    - VkShaderModuleCreateInfo

## New Unions

- VkClearColorValue
- VkClearValue

## New Function Pointers

- PFN_vkAllocationFunction
- PFN_vkFreeFunction
- PFN_vkInternalAllocationNotification
- PFN_vkInternalFreeNotification
- PFN_vkReallocationFunction
- PFN_vkVoidFunction

## New Enums

- VkAccessFlagBits
- VkAttachmentDescriptionFlagBits
- VkAttachmentLoadOp
- VkAttachmentStoreOp
- VkBlendFactor
- VkBlendOp
- VkBorderColor
- VkBufferCreateFlagBits
- VkBufferUsageFlagBits
- VkColorComponentFlagBits
- VkCommandBufferLevel
- VkCommandBufferResetFlagBits
- VkCommandBufferUsageFlagBits
- VkCommandPoolCreateFlagBits
- VkCommandPoolResetFlagBits
- VkCompareOp
- VkComponentSwizzle
- VkCullModeFlagBits

- VkDependencyFlagBits
- VkDescriptorPoolCreateFlagBits
- VkDescriptorSetLayoutCreateFlagBits
- VkDescriptorType
- VkDynamicState
- VkEventCreateFlagBits
- VkFenceCreateFlagBits
- VkFilter
- VkFormat
- VkFormatFeatureFlagBits
- VkFramebufferCreateFlagBits
- VkFrontFace
- VkImageAspectFlagBits
- VkImageCreateFlagBits
- VkImageLayout
- VkImageTiling
- VkImageType
- VkImageUsageFlagBits
- VkImageViewCreateFlagBits
- VkImageViewType
- VkIndexType
- VkInstanceCreateFlagBits
- VkInternalAllocationType
- VkLogicOp
- VkMemoryHeapFlagBits
- VkMemoryMapFlagBits
- VkMemoryPropertyFlagBits
- VkObjectType
- VkPhysicalDeviceType
- VkPipelineBindPoint
- VkPipelineCacheHeaderVersion
- VkPipelineCreateFlagBits
- VkPipelineShaderStageCreateFlagBits
- VkPipelineStageFlagBits
- VkPolygonMode

- VkPrimitiveTopology
- VkQueryControlFlagBits
- VkQueryPipelineStatisticFlagBits
- VkQueryResultFlagBits
- VkQueryType
- VkQueueFlagBits
- VkRenderPassCreateFlagBits
- VkResult
- VkSampleCountFlagBits
- VkSamplerAddressMode
- VkSamplerCreateFlagBits
- VkSamplerMipmapMode
- VkShaderStageFlagBits
- VkSharingMode
- VkSparseImageFormatFlagBits
- VkSparseMemoryBindFlagBits
- VkStencilFaceFlagBits
- VkStencilOp
- VkStructureType
- VkSubpassContents
- VkSubpassDescriptionFlagBits
- VkSystemAllocationScope
- VkVendorId
- VkVertexInputRate

## New Bitmasks

- VkAccessFlags
- VkAttachmentDescriptionFlags
- VkBufferCreateFlags
- VkBufferUsageFlags
- VkBufferViewCreateFlags
- VkColorComponentFlags
- VkCommandBufferResetFlags
- VkCommandBufferUsageFlags
- VkCommandPoolCreateFlags

- VkCommandPoolResetFlags
- VkCullModeFlags
- VkDependencyFlags
- VkDescriptorPoolCreateFlags
- VkDescriptorPoolResetFlags
- VkDescriptorSetLayoutCreateFlags
- VkDeviceCreateFlags
- VkDeviceQueueCreateFlags
- VkEventCreateFlags
- VkFenceCreateFlags
- VkFormatFeatureFlags
- VkFramebufferCreateFlags
- VkImageAspectFlags
- VkImageCreateFlags
- VkImageUsageFlags
- VkImageViewCreateFlags
- VkInstanceCreateFlags
- VkMemoryHeapFlags
- VkMemoryMapFlags
- VkMemoryPropertyFlags
- VkPipelineCacheCreateFlags
- VkPipelineColorBlendStateCreateFlags
- VkPipelineCreateFlags
- VkPipelineDepthStencilStateCreateFlags
- VkPipelineDynamicStateCreateFlags
- VkPipelineInputAssemblyStateCreateFlags
- VkPipelineLayoutCreateFlags
- VkPipelineMultisampleStateCreateFlags
- VkPipelineRasterizationStateCreateFlags
- VkPipelineShaderStageCreateFlags
- VkPipelineStageFlags
- VkPipelineTessellationStateCreateFlags
- VkPipelineVertexInputStateCreateFlags
- VkPipelineViewportStateCreateFlags
- VkQueryControlFlags

- VkQueryPipelineStatisticFlags
- VkQueryPoolCreateFlags
- VkQueryResultFlags
- VkQueueFlags
- VkRenderPassCreateFlags
- VkSampleCountFlags
- VkSamplerCreateFlags
- VkSemaphoreCreateFlags
- VkShaderModuleCreateFlags
- VkShaderStageFlags
- VkSparseImageFormatFlags
- VkSparseMemoryBindFlags
- VkStencilFaceFlags
- VkSubpassDescriptionFlags

## New Headers

- `vk_platform`

## New Enum Constants

- `VK_ATTACHMENT_UNUSED`
- `VK_FALSE`
- `VK_LOD_CLAMP_NONE`
- `VK_MAX_DESCRIPTION_SIZE`
- `VK_MAX_EXTENSION_NAME_SIZE`
- `VK_MAX_MEMORY_HEAPS`
- `VK_MAX_MEMORY_TYPES`
- `VK_MAX_PHYSICAL_DEVICE_NAME_SIZE`
- `VK_QUEUE_FAMILY_IGNORED`
- `VK_REMAINING_ARRAY_LAYERS`
- `VK_REMAINING_MIP_LEVELS`
- `VK_SUBPASS_EXTERNAL`
- `VK_TRUE`
- `VK_UUID_SIZE`
- `VK_WHOLE_SIZE`

# Appendix E: Layers & Extensions (Informative)

Extensions to the Vulkan API **can** be defined by authors, groups of authors, and the Khronos Vulkan Working Group. In order not to compromise the readability of the Vulkan Specification, the core Specification does not incorporate most extensions. The online Registry of extensions is available at URL

https://registry.khronos.org/vulkan/

and allows generating versions of the Specification incorporating different extensions.

Authors creating extensions and layers **must** follow the mandatory procedures described in the Vulkan Documentation and Extensions document when creating extensions and layers.

The remainder of this appendix documents a set of extensions chosen when this document was built. Versions of the Specification published in the Registry include:

- Core API + mandatory extensions required of all Vulkan implementations.
- Core API + all registered and published Khronos (`KHR`) extensions.
- Core API + all registered and published extensions.

Extensions are grouped as Khronos `KHR`, multivendor `EXT`, and then alphabetically by author ID. Within each group, extensions are listed in alphabetical order by their name.

## Extension Dependencies

Extensions which have dependencies on specific core versions or on other extensions will list such dependencies.

For core versions, the specified version **must** be supported at runtime. All extensions implicitly require support for Vulkan 1.0.

For a device extension, use of any device-level functionality defined by that extension requires that any extensions that extension depends on be enabled.

For any extension, use of any instance-level functionality defined by that extension requires only that any extensions that extension depends on be supported at runtime.

## Extension Interactions

Some extensions define APIs which are only supported when other extensions or core versions are supported at runtime. Such interactions are noted as “API Interactions”.

# List of Current Extensions

- VK_KHR_8bit_storage (promoted to core)
- VK_KHR_acceleration_structure
- VK_KHR_android_surface
- VK_KHR_buffer_device_address (promoted to core)
- VK_KHR_calibrated_timestamps
- VK_KHR_compute_shader_derivatives
- VK_KHR_cooperative_matrix
- VK_KHR_copy_commands2 (promoted to core)
- VK_KHR_create_renderpass2 (promoted to core)
- VK_KHR_deferred_host_operations
- VK_KHR_depth_stencil_resolve (promoted to core)
- VK_KHR_display
- VK_KHR_display_swapchain
- VK_KHR_draw_indirect_count (promoted to core)
- VK_KHR_driver_properties (promoted to core)
- VK_KHR_dynamic_rendering (promoted to core)
- VK_KHR_dynamic_rendering_local_read
- VK_KHR_external_fence_fd
- VK_KHR_external_fence_win32
- VK_KHR_external_memory_fd
- VK_KHR_external_memory_win32
- VK_KHR_external_semaphore_fd
- VK_KHR_external_semaphore_win32
- VK_KHR_format_feature_flags2 (promoted to core)
- VK_KHR_fragment_shader_barycentric
- VK_KHR_fragment_shading_rate
- VK_KHR_get_display_properties2
- VK_KHR_get_surface_capabilities2
- VK_KHR_global_priority
- VK_KHR_image_format_list (promoted to core)
- VK_KHR_imageless_framebuffer (promoted to core)
- VK_KHR_incremental_present
- VK_KHR_index_type_uint8

- VK_KHR_line_rasterization
- VK_KHR_load_store_op_none
- VK_KHR_maintenance4 (promoted to core)
- VK_KHR_maintenance5
- VK_KHR_maintenance6
- VK_KHR_maintenance7
- VK_KHR_map_memory2
- VK_KHR_performance_query
- VK_KHR_pipeline_binary
- VK_KHR_pipeline_executable_properties
- VK_KHR_pipeline_library
- VK_KHR_portability_enumeration
- VK_KHR_present_id
- VK_KHR_present_wait
- VK_KHR_push_descriptor
- VK_KHR_ray_query
- VK_KHR_ray_tracing_maintenance1
- VK_KHR_ray_tracing_pipeline
- VK_KHR_ray_tracing_position_fetch
- VK_KHR_sampler_mirror_clamp_to_edge (promoted to core)
- VK_KHR_separate_depth_stencil_layouts (promoted to core)
- VK_KHR_shader_atomic_int64 (promoted to core)
- VK_KHR_shader_clock
- VK_KHR_shader_expect_assume
- VK_KHR_shader_float16_int8 (promoted to core)
- VK_KHR_shader_float_controls (promoted to core)
- VK_KHR_shader_float_controls2
- VK_KHR_shader_integer_dot_product (promoted to core)
- VK_KHR_shader_maximal_reconvergence
- VK_KHR_shader_non_semantic_info (promoted to core)
- VK_KHR_shader_quad_control
- VK_KHR_shader_relaxed_extended_instruction
- VK_KHR_shader_subgroup_extended_types (promoted to core)
- VK_KHR_shader_subgroup_rotate
- VK_KHR_shader_subgroup_uniform_control_flow

- VK_KHR_shader_terminate_invocation (promoted to core)
- VK_KHR_shared_presentable_image
- VK_KHR_spirv_1_4 (promoted to core)
- VK_KHR_surface
- VK_KHR_surface_protected_capabilities
- VK_KHR_swapchain
- VK_KHR_swapchain_mutable_format
- VK_KHR_synchronization2 (promoted to core)
- VK_KHR_timeline_semaphore (promoted to core)
- VK_KHR_uniform_buffer_standard_layout (promoted to core)
- VK_KHR_vertex_attribute_divisor
- VK_KHR_video_decode_av1
- VK_KHR_video_decode_h264
- VK_KHR_video_decode_h265
- VK_KHR_video_decode_queue
- VK_KHR_video_encode_h264
- VK_KHR_video_encode_h265
- VK_KHR_video_encode_queue
- VK_KHR_video_maintenance1
- VK_KHR_video_queue
- VK_KHR_vulkan_memory_model (promoted to core)
- VK_KHR_wayland_surface
- VK_KHR_win32_keyed_mutex
- VK_KHR_win32_surface
- VK_KHR_workgroup_memory_explicit_layout
- VK_KHR_xcb_surface
- VK_KHR_xlib_surface
- VK_KHR_zero_initialize_workgroup_memory (promoted to core)
- VK_EXT_4444_formats (promoted to core)
- VK_EXT_acquire_drm_display
- VK_EXT_acquire_xlib_display
- VK_EXT_astc_decode_mode
- VK_EXT_attachment_feedback_loop_dynamic_state
- VK_EXT_attachment_feedback_loop_layout
- VK_EXT_blend_operation_advanced

- VK_EXT_border_color_swizzle
- VK_EXT_color_write_enable
- VK_EXT_conditional_rendering
- VK_EXT_conservative_rasterization
- VK_EXT_custom_border_color
- VK_EXT_debug_utils
- VK_EXT_depth_bias_control
- VK_EXT_depth_clamp_zero_one
- VK_EXT_depth_clip_control
- VK_EXT_depth_clip_enable
- VK_EXT_depth_range_unrestricted
- VK_EXT_descriptor_buffer
- VK_EXT_descriptor_indexing (promoted to core)
- VK_EXT_device_address_binding_report
- VK_EXT_device_fault
- VK_EXT_device_memory_report
- VK_EXT_direct_mode_display
- VK_EXT_directfb_surface
- VK_EXT_discard_rectangles
- VK_EXT_display_control
- VK_EXT_display_surface_counter
- VK_EXT_dynamic_rendering_unused_attachments
- VK_EXT_extended_dynamic_state (promoted to core)
- VK_EXT_extended_dynamic_state2 (promoted to core)
- VK_EXT_extended_dynamic_state3
- VK_EXT_external_memory_acquire_unmodified
- VK_EXT_external_memory_dma_buf
- VK_EXT_external_memory_host
- VK_EXT_filter_cubic
- VK_EXT_fragment_density_map
- VK_EXT_fragment_density_map2
- VK_EXT_fragment_shader_interlock
- VK_EXT_frame_boundary
- VK_EXT_full_screen_exclusive
- VK_EXT_graphics_pipeline_library

- VK_EXT_hdr_metadata
- VK_EXT_headless_surface
- VK_EXT_host_image_copy
- VK_EXT_host_query_reset (promoted to core)
- VK_EXT_image_2d_view_of_3d
- VK_EXT_image_compression_control
- VK_EXT_image_compression_control_swapchain
- VK_EXT_image_drm_format_modifier
- VK_EXT_image_robustness (promoted to core)
- VK_EXT_image_sliced_view_of_3d
- VK_EXT_image_view_min_lod
- VK_EXT_inline_uniform_block (promoted to core)
- VK_EXT_layer_settings
- VK_EXT_legacy_dithering
- VK_EXT_legacy_vertex_attributes
- VK_EXT_map_memory_placed
- VK_EXT_memory_budget
- VK_EXT_memory_priority
- VK_EXT_mesh_shader
- VK_EXT_metal_objects
- VK_EXT_metal_surface
- VK_EXT_multi_draw
- VK_EXT_multisampled_render_to_single_sampled
- VK_EXT_mutable_descriptor_type
- VK_EXT_nested_command_buffer
- VK_EXT_non_seamless_cube_map
- VK_EXT_opacity_micromap
- VK_EXT_pageable_device_local_memory
- VK_EXT_pci_bus_info
- VK_EXT_physical_device_drm
- VK_EXT_pipeline_creation_cache_control (promoted to core)
- VK_EXT_pipeline_creation_feedback (promoted to core)
- VK_EXT_pipeline_library_group_handles
- VK_EXT_pipeline_properties
- VK_EXT_pipeline_protected_access

- VK_EXT_pipeline_robustness
- VK_EXT_post_depth_coverage
- VK_EXT_primitive_topology_list_restart
- VK_EXT_primitives_generated_query
- VK_EXT_private_data (promoted to core)
- VK_EXT_provoking_vertex
- VK_EXT_queue_family_foreign
- VK_EXT_rasterization_order_attachment_access
- VK_EXT_rgba10x6_formats
- VK_EXT_robustness2
- VK_EXT_sample_locations
- VK_EXT_sampler_filter_minmax (promoted to core)
- VK_EXT_scalar_block_layout (promoted to core)
- VK_EXT_separate_stencil_usage (promoted to core)
- VK_EXT_shader_atomic_float
- VK_EXT_shader_atomic_float2
- VK_EXT_shader_demote_to_helper_invocation (promoted to core)
- VK_EXT_shader_image_atomic_int64
- VK_EXT_shader_module_identifier
- VK_EXT_shader_object
- VK_EXT_shader_replicated_composites
- VK_EXT_shader_stencil_export
- VK_EXT_shader_subgroup_ballot
- VK_EXT_shader_tile_image
- VK_EXT_shader_viewport_index_layer (promoted to core)
- VK_EXT_subgroup_size_control (promoted to core)
- VK_EXT_subpass_merge_feedback
- VK_EXT_surface_maintenance1
- VK_EXT_swapchain_colorspace
- VK_EXT_swapchain_maintenance1
- VK_EXT_texel_buffer_alignment (promoted to core)
- VK_EXT_texture_compression_astc_hdr (promoted to core)
- VK_EXT_tooling_info (promoted to core)
- VK_EXT_transform_feedback
- VK_EXT_validation_cache

- VK_EXT_vertex_input_dynamic_state
- VK_EXT_ycbcr_2plane_444_formats (promoted to core)
- VK_EXT_ycbcr_image_arrays
- VK_AMD_anti_lag
- VK_AMD_buffer_marker
- VK_AMD_device_coherent_memory
- VK_AMD_display_native_hdr
- VK_AMD_gcn_shader
- VK_AMD_memory_overallocation_behavior
- VK_AMD_mixed_attachment_samples
- VK_AMD_pipeline_compiler_control
- VK_AMD_rasterization_order
- VK_AMD_shader_ballot
- VK_AMD_shader_core_properties
- VK_AMD_shader_core_properties2
- VK_AMD_shader_early_and_late_fragment_tests
- VK_AMD_shader_explicit_vertex_parameter
- VK_AMD_shader_fragment_mask
- VK_AMD_shader_image_load_store_lod
- VK_AMD_shader_info
- VK_AMD_shader_trinary_minmax
- VK_AMD_texture_gather_bias_lod
- VK_ANDROID_external_format_resolve
- VK_ANDROID_external_memory_android_hardware_buffer
- VK_ARM_render_pass_striped
- VK_ARM_scheduling_controls
- VK_ARM_shader_core_builtins
- VK_ARM_shader_core_properties
- VK_FUCHSIA_buffer_collection
- VK_FUCHSIA_external_memory
- VK_FUCHSIA_external_semaphore
- VK_FUCHSIA_imagepipe_surface
- VK_GGP_frame_token
- VK_GGP_stream_descriptor_surface
- VK_GOOGLE_decorate_string

- VK_GOOGLE_display_timing
- VK_GOOGLE_hlsl_functionality1
- VK_GOOGLE_surfaceless_query
- VK_GOOGLE_user_type
- VK_HUAWEI_cluster_culling_shader
- VK_HUAWEI_invocation_mask
- VK_HUAWEI_subpass_shading
- VK_IMG_filter_cubic
- VK_IMG_relaxed_line_rasterization
- VK_INTEL_performance_query
- VK_INTEL_shader_integer_functions2
- VK_LUNARG_direct_driver_loading
- VK_MESA_image_alignment_control
- VK_MSFT_layered_driver
- VK_NN_vi_surface
- VK_NV_acquire_winrt_display
- VK_NV_clip_space_w_scaling
- VK_NV_command_buffer_inheritance
- VK_NV_cooperative_matrix
- VK_NV_copy_memory_indirect
- VK_NV_corner_sampled_image
- VK_NV_coverage_reduction_mode
- VK_NV_dedicated_allocation_image_aliasing
- VK_NV_descriptor_pool_overallocation
- VK_NV_device_diagnostic_checkpoints
- VK_NV_device_diagnostics_config
- VK_NV_device_generated_commands
- VK_NV_device_generated_commands_compute
- VK_NV_extended_sparse_address_space
- VK_NV_external_memory_rdma
- VK_NV_fill_rectangle
- VK_NV_fragment_coverage_to_color
- VK_NV_fragment_shading_rate_enums
- VK_NV_framebuffer_mixed_samples
- VK_NV_geometry_shader_passthrough

- VK_NV_inherited_viewport_scissor
- VK_NV_linear_color_attachment
- VK_NV_low_latency
- VK_NV_low_latency2
- VK_NV_memory_decompression
- VK_NV_mesh_shader
- VK_NV_optical_flow
- VK_NV_per_stage_descriptor_set
- VK_NV_present_barrier
- VK_NV_raw_access_chains
- VK_NV_ray_tracing_invocation_reorder
- VK_NV_ray_tracing_motion_blur
- VK_NV_ray_tracing_validation
- VK_NV_representative_fragment_test
- VK_NV_sample_mask_override_coverage
- VK_NV_scissor_exclusive
- VK_NV_shader_atomic_float16_vector
- VK_NV_shader_image_footprint
- VK_NV_shader_sm_builtins
- VK_NV_shader_subgroup_partitioned
- VK_NV_shading_rate_image
- VK_NV_viewport_array2
- VK_NV_viewport_swizzle
- VK_NVX_binary_import
- VK_NVX_image_view_handle
- VK_NVX_multiview_per_view_attributes
- VK_QCOM_filter_cubic_clamp
- VK_QCOM_filter_cubic_weights
- VK_QCOM_fragment_density_map_offset
- VK_QCOM_image_processing
- VK_QCOM_image_processing2
- VK_QCOM_multiview_per_view_render_areas
- VK_QCOM_multiview_per_view_viewports
- VK_QCOM_render_pass_shader_resolve
- VK_QCOM_render_pass_store_ops

- VK_QCOM_render_pass_transform
- VK_QCOM_rotated_copy_commands
- VK_QCOM_tile_properties
- VK_QCOM_ycbcr_degamma
- VK_QNX_external_memory_screen_buffer
- VK_QNX_screen_surface
- VK_SEC_amigo_profiling
- VK_VALVE_descriptor_set_host_mapping

## VK_KHR_8bit_storage

### Name String

`VK_KHR_8bit_storage`

### Extension Type

Device extension

### Registered Extension Number

178

### Revision

1

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
and
VK_KHR_storage_buffer_storage_class
or
Vulkan Version 1.1

### SPIR-V Dependencies

- SPV_KHR_8bit_storage

### Deprecation State

- *Promoted* to Vulkan 1.2

### Contact

- Alexander Galazin [GitHub]alegal-arm

### Other Extension Metadata

### Last Modified Date

2018-02-05

### Interactions and External Dependencies

- This extension provides API support for `GL_EXT_shader_16bit_storage`

### IP Status

No known IP claims.

### Contributors

- Alexander Galazin, Arm

### Description

The `VK_KHR_8bit_storage` extension allows use of 8-bit types in uniform and storage buffers, and

push constant blocks. This extension introduces several new optional features which map to SPIR-V capabilities and allow access to 8-bit data in `Block`-decorated objects in the `Uniform` and the `StorageBuffer` storage classes, and objects in the `PushConstant` storage class.

The `StorageBuffer8BitAccess` capability **must** be supported by all implementations of this extension. The other capabilities are optional.

#### Promotion to Vulkan 1.2

Functionality in this extension is included in core Vulkan 1.2, with the KHR suffix omitted. However, if Vulkan 1.2 is supported and this extension is not, the `StorageBuffer8BitAccess` capability is optional. The original type, enum and command names are still available as aliases of the core functionality.

#### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDevice8BitStorageFeaturesKHR

#### New Enum Constants

- `VK_KHR_8BIT_STORAGE_EXTENSION_NAME`
- `VK_KHR_8BIT_STORAGE_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_8BIT_STORAGE_FEATURES_KHR`

#### New SPIR-V Capabilities

- `StorageBuffer8BitAccess`
- `UniformAndStorageBuffer8BitAccess`
- `StoragePushConstant8`

#### Version History

- Revision 1, 2018-02-05 (Alexander Galazin)
  - Initial draft

## VK_KHR_acceleration_structure

#### Name String

`VK_KHR_acceleration_structure`

#### Extension Type

Device extension

#### Registered Extension Number

151

**Revision**

13

**Ratification Status**

Ratified

**Extension and Version Dependencies**

Vulkan Version 1.1
and
VK_EXT_descriptor_indexing
and
VK_KHR_buffer_device_address
or
Vulkan Version 1.2
and
VK_KHR_deferred_host_operations

**API Interactions**

- Interacts with VK_VERSION_1_2
- Interacts with VK_VERSION_1_3
- Interacts with VK_EXT_debug_report
- Interacts with VK_KHR_format_feature_flags2

**Contact**

- Daniel Koch [GitHub]dgkoch

**Other Extension Metadata**

**Last Modified Date**

2021-09-30

**Contributors**

- Samuel Bourasseau, Adobe
- Matthäus Chajdas, AMD
- Greg Grebe, AMD
- Nicolai Hähnle, AMD
- Tobias Hector, AMD
- Dave Oldcorn, AMD
- Skyler Saleh, AMD
- Mathieu Robart, Arm
- Marius Bjorge, Arm
- Tom Olson, Arm
- Sebastian Tafuri, EA

- Henrik Rydgard, Embark
- Juan Cañada, Epic Games
- Patrick Kelly, Epic Games
- Yuriy O'Donnell, Epic Games
- Michael Doggett, Facebook/Oculus
- Ricardo Garcia, Igalia
- Andrew Garrard, Imagination
- Don Scorgie, Imagination
- Dae Kim, Imagination
- Joshua Barczak, Intel
- Slawek Grajewski, Intel
- Jeff Bolz, NVIDIA
- Pascal Gautron, NVIDIA
- Daniel Koch, NVIDIA
- Christoph Kubisch, NVIDIA
- Ashwin Lele, NVIDIA
- Robert Stepinski, NVIDIA
- Martin Stich, NVIDIA
- Nuno Subtil, NVIDIA
- Eric Werness, NVIDIA
- Jon Leech, Khronos
- Jeroen van Schijndel, OTOY
- Juul Joosten, OTOY
- Alex Bourd, Qualcomm
- Roman Larionov, Qualcomm
- David McAllister, Qualcomm
- Lewis Gordon, Samsung
- Ralph Potter, Samsung
- Jasper Bekkers, Traverse Research
- Jesse Barker, Unity
- Baldur Karlsson, Valve

**Description**

In order to be efficient, rendering techniques such as ray tracing need a quick way to identify which primitives may be intersected by a ray traversing the geometries. Acceleration structures are the most common way to represent the geometry spatially sorted, in order to quickly identify such

potential intersections.

This extension adds new functionalities:

- Acceleration structure objects and build commands
- Structures to describe geometry inputs to acceleration structure builds
- Acceleration structure copy commands

### New Object Types

- VkAccelerationStructureKHR

### New Commands

- vkBuildAccelerationStructuresKHR
- vkCmdBuildAccelerationStructuresIndirectKHR
- vkCmdBuildAccelerationStructuresKHR
- vkCmdCopyAccelerationStructureKHR
- vkCmdCopyAccelerationStructureToMemoryKHR
- vkCmdCopyMemoryToAccelerationStructureKHR
- vkCmdWriteAccelerationStructuresPropertiesKHR
- vkCopyAccelerationStructureKHR
- vkCopyAccelerationStructureToMemoryKHR
- vkCopyMemoryToAccelerationStructureKHR
- vkCreateAccelerationStructureKHR
- vkDestroyAccelerationStructureKHR
- vkGetAccelerationStructureBuildSizesKHR
- vkGetAccelerationStructureDeviceAddressKHR
- vkGetDeviceAccelerationStructureCompatibilityKHR
- vkWriteAccelerationStructuresPropertiesKHR

### New Structures

- VkAabbPositionsKHR
- VkAccelerationStructureBuildGeometryInfoKHR
- VkAccelerationStructureBuildRangeInfoKHR
- VkAccelerationStructureBuildSizesInfoKHR
- VkAccelerationStructureCreateInfoKHR
- VkAccelerationStructureDeviceAddressInfoKHR
- VkAccelerationStructureGeometryAabbsDataKHR

- VkAccelerationStructureGeometryInstancesDataKHR
- VkAccelerationStructureGeometryKHR
- VkAccelerationStructureGeometryTrianglesDataKHR
- VkAccelerationStructureInstanceKHR
- VkAccelerationStructureVersionInfoKHR
- VkCopyAccelerationStructureInfoKHR
- VkCopyAccelerationStructureToMemoryInfoKHR
- VkCopyMemoryToAccelerationStructureInfoKHR
- VkTransformMatrixKHR
- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceAccelerationStructureFeaturesKHR
- Extending VkPhysicalDeviceProperties2:
  - VkPhysicalDeviceAccelerationStructurePropertiesKHR
- Extending VkWriteDescriptorSet:
  - VkWriteDescriptorSetAccelerationStructureKHR

### New Unions

- VkAccelerationStructureGeometryDataKHR
- VkDeviceOrHostAddressConstKHR
- VkDeviceOrHostAddressKHR

### New Enums

- VkAccelerationStructureBuildTypeKHR
- VkAccelerationStructureCompatibilityKHR
- VkAccelerationStructureCreateFlagBitsKHR
- VkAccelerationStructureTypeKHR
- VkBuildAccelerationStructureFlagBitsKHR
- VkBuildAccelerationStructureModeKHR
- VkCopyAccelerationStructureModeKHR
- VkGeometryFlagBitsKHR
- VkGeometryInstanceFlagBitsKHR
- VkGeometryTypeKHR

### New Bitmasks

- VkAccelerationStructureCreateFlagsKHR
- VkBuildAccelerationStructureFlagsKHR

- VkGeometryFlagsKHR
- VkGeometryInstanceFlagsKHR

### New Enum Constants

- VK_KHR_ACCELERATION_STRUCTURE_EXTENSION_NAME
- VK_KHR_ACCELERATION_STRUCTURE_SPEC_VERSION
- Extending VkAccessFlagBits:
  - VK_ACCESS_ACCELERATION_STRUCTURE_READ_BIT_KHR
  - VK_ACCESS_ACCELERATION_STRUCTURE_WRITE_BIT_KHR
- Extending VkBufferUsageFlagBits:
  - VK_BUFFER_USAGE_ACCELERATION_STRUCTURE_BUILD_INPUT_READ_ONLY_BIT_KHR
  - VK_BUFFER_USAGE_ACCELERATION_STRUCTURE_STORAGE_BIT_KHR
- Extending VkDescriptorType:
  - VK_DESCRIPTOR_TYPE_ACCELERATION_STRUCTURE_KHR
- Extending VkFormatFeatureFlagBits:
  - VK_FORMAT_FEATURE_ACCELERATION_STRUCTURE_VERTEX_BUFFER_BIT_KHR
- Extending VkIndexType:
  - VK_INDEX_TYPE_NONE_KHR
- Extending VkObjectType:
  - VK_OBJECT_TYPE_ACCELERATION_STRUCTURE_KHR
- Extending VkPipelineStageFlagBits:
  - VK_PIPELINE_STAGE_ACCELERATION_STRUCTURE_BUILD_BIT_KHR
- Extending VkQueryType:
  - VK_QUERY_TYPE_ACCELERATION_STRUCTURE_COMPACTED_SIZE_KHR
  - VK_QUERY_TYPE_ACCELERATION_STRUCTURE_SERIALIZATION_SIZE_KHR
- Extending VkStructureType:
  - VK_STRUCTURE_TYPE_ACCELERATION_STRUCTURE_BUILD_GEOMETRY_INFO_KHR
  - VK_STRUCTURE_TYPE_ACCELERATION_STRUCTURE_BUILD_SIZES_INFO_KHR
  - VK_STRUCTURE_TYPE_ACCELERATION_STRUCTURE_CREATE_INFO_KHR
  - VK_STRUCTURE_TYPE_ACCELERATION_STRUCTURE_DEVICE_ADDRESS_INFO_KHR
  - VK_STRUCTURE_TYPE_ACCELERATION_STRUCTURE_GEOMETRY_AABBS_DATA_KHR
  - VK_STRUCTURE_TYPE_ACCELERATION_STRUCTURE_GEOMETRY_INSTANCES_DATA_KHR
  - VK_STRUCTURE_TYPE_ACCELERATION_STRUCTURE_GEOMETRY_KHR
  - VK_STRUCTURE_TYPE_ACCELERATION_STRUCTURE_GEOMETRY_TRIANGLES_DATA_KHR
  - VK_STRUCTURE_TYPE_ACCELERATION_STRUCTURE_VERSION_INFO_KHR

- `VK_STRUCTURE_TYPE_COPY_ACCELERATION_STRUCTURE_INFO_KHR`
- `VK_STRUCTURE_TYPE_COPY_ACCELERATION_STRUCTURE_TO_MEMORY_INFO_KHR`
- `VK_STRUCTURE_TYPE_COPY_MEMORY_TO_ACCELERATION_STRUCTURE_INFO_KHR`
- `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_ACCELERATION_STRUCTURE_FEATURES_KHR`
- `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_ACCELERATION_STRUCTURE_PROPERTIES_KHR`
- `VK_STRUCTURE_TYPE_WRITE_DESCRIPTOR_SET_ACCELERATION_STRUCTURE_KHR`

If VK_EXT_debug_report is supported:

- Extending VkDebugReportObjectTypeEXT:
  - `VK_DEBUG_REPORT_OBJECT_TYPE_ACCELERATION_STRUCTURE_KHR_EXT`

If VK_KHR_format_feature_flags2 or Vulkan Version 1.3 is supported:

- Extending VkFormatFeatureFlagBits2:
  - `VK_FORMAT_FEATURE_2_ACCELERATION_STRUCTURE_VERTEX_BUFFER_BIT_KHR`

**Issues**

(1) How does this extension differ from VK_NV_ray_tracing?

**DISCUSSION**:

The following is a summary of the main functional differences between VK_KHR_acceleration_structure and VK_NV_ray_tracing:

- added acceleration structure serialization / deserialization (`VK_COPY_ACCELERATION_STRUCTURE_MODE_SERIALIZE_KHR`, `VK_COPY_ACCELERATION_STRUCTURE_MODE_DESERIALIZE_KHR`, vkCmdCopyAccelerationStructureToMemoryKHR, vkCmdCopyMemoryToAccelerationStructureKHR)
- document inactive primitives and instances
- added VkPhysicalDeviceAccelerationStructureFeaturesKHR structure
- added indirect and batched acceleration structure builds (vkCmdBuildAccelerationStructuresIndirectKHR)
- added host acceleration structure commands
- reworked geometry structures so they could be better shared between device, host, and indirect builds
- explicitly made VkAccelerationStructureKHR use device addresses
- added acceleration structure compatibility check function (vkGetDeviceAccelerationStructureCompatibilityKHR)
- add parameter for requesting memory requirements for host and/or device build
- added format feature for acceleration structure build vertex formats (`VK_FORMAT_FEATURE_ACCELERATION_STRUCTURE_VERTEX_BUFFER_BIT_KHR`)

(2) Can you give a more detailed comparison of differences and similarities between VK_NV_ray_tracing and VK_KHR_acceleration_structure?

**DISCUSSION**:

The following is a more detailed comparison of which commands, structures, and enums are aliased, changed, or removed.

- Aliased functionality — enums, structures, and commands that are considered equivalent:
  - VkGeometryTypeNV  VkGeometryTypeKHR
  - VkAccelerationStructureTypeNV  VkAccelerationStructureTypeKHR
  - VkCopyAccelerationStructureModeNV  VkCopyAccelerationStructureModeKHR
  - VkGeometryFlagsNV  VkGeometryFlagsKHR
  - VkGeometryFlagBitsNV  VkGeometryFlagBitsKHR
  - VkGeometryInstanceFlagsNV  VkGeometryInstanceFlagsKHR
  - VkGeometryInstanceFlagBitsNV  VkGeometryInstanceFlagBitsKHR
  - VkBuildAccelerationStructureFlagsNV  VkBuildAccelerationStructureFlagsKHR
  - VkBuildAccelerationStructureFlagBitsNV  VkBuildAccelerationStructureFlagBitsKHR
  - VkTransformMatrixNV  VkTransformMatrixKHR (added to VK_NV_ray_tracing for descriptive purposes)
  - VkAabbPositionsNV  VkAabbPositionsKHR (added to VK_NV_ray_tracing for descriptive purposes)
  - VkAccelerationStructureInstanceNV  VkAccelerationStructureInstanceKHR (added to VK_NV_ray_tracing for descriptive purposes)
- Changed enums, structures, and commands:
  - renamed `VK_GEOMETRY_INSTANCE_TRIANGLE_CULL_DISABLE_BIT_NV` → `VK_GEOMETRY_INSTANCE_TRIANGLE_FACING_CULL_DISABLE_BIT_KHR` in VkGeometryInstanceFlagBitsKHR
  - VkGeometryTrianglesNV → VkAccelerationStructureGeometryTrianglesDataKHR (device or host address instead of buffer+offset)
  - VkGeometryAABBNV → VkAccelerationStructureGeometryAabbsDataKHR (device or host address instead of buffer+offset)
  - VkGeometryDataNV → VkAccelerationStructureGeometryDataKHR (union of triangle/aabbs/instances)
  - VkGeometryNV → VkAccelerationStructureGeometryKHR (changed type of geometry)
  - VkAccelerationStructureCreateInfoNV → VkAccelerationStructureCreateInfoKHR (reshuffle geometry layout/information)
  - VkPhysicalDeviceRayTracingPropertiesNV → VkPhysicalDeviceAccelerationStructurePropertiesKHR (for acceleration structure properties, renamed `maxTriangleCount` to `maxPrimitiveCount`, added per stage and update after bind limits) and VkPhysicalDeviceRayTracingPipelinePropertiesKHR (for ray tracing

pipeline properties)

  - VkAccelerationStructureMemoryRequirementsInfoNV (deleted - replaced by allocating on top of VkBuffer)
  - VkWriteDescriptorSetAccelerationStructureNV → VkWriteDescriptorSetAccelerationStructureKHR (different acceleration structure type)
  - vkCreateAccelerationStructureNV → vkCreateAccelerationStructureKHR (device address, different geometry layout/information)
  - vkGetAccelerationStructureMemoryRequirementsNV (deleted - replaced by allocating on top of VkBuffer)
  - vkCmdBuildAccelerationStructureNV → vkCmdBuildAccelerationStructuresKHR (params moved to structs, layout differences)
  - vkCmdCopyAccelerationStructureNV → vkCmdCopyAccelerationStructureKHR (params to struct, extendable)
  - vkGetAccelerationStructureHandleNV → vkGetAccelerationStructureDeviceAddressKHR (device address instead of handle)
  - VkAccelerationStructureMemoryRequirementsTypeNV → size queries for scratch space moved to vkGetAccelerationStructureBuildSizesKHR
  - vkDestroyAccelerationStructureNV → vkDestroyAccelerationStructureKHR (different acceleration structure types)
  - vkCmdWriteAccelerationStructuresPropertiesNV → vkCmdWriteAccelerationStructuresPropertiesKHR (different acceleration structure types)
- Added enums, structures and commands:
  - `VK_GEOMETRY_TYPE_INSTANCES_KHR` to VkGeometryTypeKHR enum
  - `VK_COPY_ACCELERATION_STRUCTURE_MODE_SERIALIZE_KHR`, `VK_COPY_ACCELERATION_STRUCTURE_MODE_DESERIALIZE_KHR` to VkCopyAccelerationStructureModeKHR enum
  - VkPhysicalDeviceAccelerationStructureFeaturesKHR structure
  - VkAccelerationStructureBuildTypeKHR enum
  - VkBuildAccelerationStructureModeKHR enum
  - VkDeviceOrHostAddressKHR and VkDeviceOrHostAddressConstKHR unions
  - VkAccelerationStructureBuildRangeInfoKHR struct
  - VkAccelerationStructureGeometryInstancesDataKHR struct
  - VkAccelerationStructureDeviceAddressInfoKHR struct
  - VkAccelerationStructureVersionInfoKHR struct
  - VkStridedDeviceAddressRegionKHR struct
  - VkCopyAccelerationStructureToMemoryInfoKHR struct
  - VkCopyMemoryToAccelerationStructureInfoKHR struct
  - VkCopyAccelerationStructureInfoKHR struct

- vkBuildAccelerationStructuresKHR command (host build)
- vkCopyAccelerationStructureKHR command (host copy)
- vkCopyAccelerationStructureToMemoryKHR (host serialize)
- vkCopyMemoryToAccelerationStructureKHR (host deserialize)
- vkWriteAccelerationStructuresPropertiesKHR (host properties)
- vkCmdCopyAccelerationStructureToMemoryKHR (device serialize)
- vkCmdCopyMemoryToAccelerationStructureKHR (device deserialize)
- vkGetDeviceAccelerationStructureCompatibilityKHR (serialization)

(3) What are the changes between the public provisional (VK_KHR_ray_tracing v8) release and the internal provisional (VK_KHR_ray_tracing v9) release?

- added `geometryFlags` to `VkAccelerationStructureCreateGeometryTypeInfoKHR` (later reworked to obsolete this)
- added `minAccelerationStructureScratchOffsetAlignment` property to VkPhysicalDeviceRayTracingPropertiesKHR
- fix naming and return enum from vkGetDeviceAccelerationStructureCompatibilityKHR
  - renamed `VkAccelerationStructureVersionKHR` to VkAccelerationStructureVersionInfoKHR
  - renamed `VK_STRUCTURE_TYPE_ACCELERATION_STRUCTURE_VERSION_KHR` to `VK_STRUCTURE_TYPE_ACCELERATION_STRUCTURE_VERSION_INFO_KHR`
  - removed `VK_ERROR_INCOMPATIBLE_VERSION_KHR`
  - added VkAccelerationStructureCompatibilityKHR enum
  - remove return value from vkGetDeviceAccelerationStructureCompatibilityKHR and added return enum parameter
- Require Vulkan 1.1
- added creation time capture and replay flags
  - added VkAccelerationStructureCreateFlagBitsKHR and VkAccelerationStructureCreateFlagsKHR
  - renamed the `flags` member of VkAccelerationStructureCreateInfoKHR to `buildFlags` (later removed) and added the `createFlags` member
- change vkCmdBuildAccelerationStructuresIndirectKHR to use buffer device address for indirect parameter
- make `VK_KHR_deferred_host_operations` an interaction instead of a required extension (later went back on this)
- renamed `VkAccelerationStructureBuildOffsetInfoKHR` to VkAccelerationStructureBuildRangeInfoKHR
  - renamed the `ppOffsetInfos` parameter of vkCmdBuildAccelerationStructuresKHR to `ppBuildRangeInfos`
- Re-unify geometry description between build and create

  - remove VkAccelerationStructureCreateGeometryTypeInfoKHR and VK_STRUCTURE_TYPE_ACCELERATION_STRUCTURE_CREATE_GEOMETRY_TYPE_INFO_KHR
  - added VkAccelerationStructureCreateSizeInfoKHR structure (later removed)
  - change type of the pGeometryInfos member of VkAccelerationStructureCreateInfoKHR from VkAccelerationStructureCreateGeometryTypeInfoKHR to VkAccelerationStructureGeometryKHR (later removed)
  - added pCreateSizeInfos member to VkAccelerationStructureCreateInfoKHR (later removed)
- Fix ppGeometries ambiguity, add pGeometries
  - remove geometryArrayOfPointers member of VkAccelerationStructureBuildGeometryInfoKHR
  - disambiguate two meanings of ppGeometries by explicitly adding pGeometries to the VkAccelerationStructureBuildGeometryInfoKHR structure and require one of them be NULL
- added nullDescriptor support for acceleration structures
- changed the update member of VkAccelerationStructureBuildGeometryInfoKHR from a bool to the mode VkBuildAccelerationStructureModeKHR enum which allows future extensibility in update types
- Clarify deferred host ops for pipeline creation
  - VkDeferredOperationKHR is now a top-level parameter for vkBuildAccelerationStructuresKHR, vkCreateRayTracingPipelinesKHR, vkCopyAccelerationStructureToMemoryKHR, vkCopyAccelerationStructureKHR, and vkCopyMemoryToAccelerationStructureKHR
  - removed VkDeferredOperationInfoKHR structure
  - change deferred host creation/return parameter behavior such that the implementation can modify such parameters until the deferred host operation completes
  - VK_KHR_deferred_host_operations is required again
- Change acceleration structure build to always be sized
  - de-alias VkAccelerationStructureMemoryRequirementsTypeNV and VkAccelerationStructureMemoryRequirementsTypeKHR, and remove VkAccelerationStructureMemoryRequirementsTypeKHR
  - add vkGetAccelerationStructureBuildSizesKHR command and VkAccelerationStructureBuildSizesInfoKHR structure and VK_STRUCTURE_TYPE_ACCELERATION_STRUCTURE_BUILD_SIZES_INFO_KHR enum to query sizes for acceleration structures and scratch storage
  - move size queries for scratch space to vkGetAccelerationStructureBuildSizesKHR
  - remove compactedSize, buildFlags, maxGeometryCount, pGeometryInfos, pCreateSizeInfos members of VkAccelerationStructureCreateInfoKHR and add the size member
  - add maxVertex member to VkAccelerationStructureGeometryTrianglesDataKHR structure
  - remove VkAccelerationStructureCreateSizeInfoKHR structure

(4) What are the changes between the internal provisional (VK_KHR_ray_tracing v9) release and the

final (VK_KHR_acceleration_structure v11) release?

- refactor VK_KHR_ray_tracing into 3 extensions, enabling implementation flexibility and decoupling ray query support from ray pipelines:
  - VK_KHR_acceleration_structure (for acceleration structure operations)
  - VK_KHR_ray_tracing_pipeline (for ray tracing pipeline and shader stages)
  - VK_KHR_ray_query (for ray queries in existing shader stages)
- clarify buffer usage flags for ray tracing
  - VK_BUFFER_USAGE_RAY_TRACING_BIT_NV is left alone in VK_NV_ray_tracing (required on scratch and instanceData)
  - VK_BUFFER_USAGE_SHADER_BINDING_TABLE_BIT_KHR is added as an alias of VK_BUFFER_USAGE_RAY_TRACING_BIT_NV in VK_KHR_ray_tracing_pipeline and is required on shader binding table buffers
  - VK_BUFFER_USAGE_ACCELERATION_STRUCTURE_BUILD_INPUT_READ_ONLY_BIT_KHR is added in VK_KHR_acceleration_structure for all vertex, index, transform, aabb, and instance buffer data referenced by device build commands
  - VK_BUFFER_USAGE_STORAGE_BUFFER_BIT is used for scratchData
- add max primitive counts (ppMaxPrimitiveCounts) to vkCmdBuildAccelerationStructuresIndirectKHR
- Allocate acceleration structures from VkBuffers and add a mode to constrain the device address
  - de-alias VkBindAccelerationStructureMemoryInfoNV and vkBindAccelerationStructureMemoryNV, and remove VkBindAccelerationStructureMemoryInfoKHR, VkAccelerationStructureMemoryRequirementsInfoKHR, and vkGetAccelerationStructureMemoryRequirementsKHR
  - acceleration structures now take a VkBuffer and offset at creation time for memory placement
  - add a new VK_BUFFER_USAGE_ACCELERATION_STRUCTURE_STORAGE_BIT_KHR buffer usage for such buffers
  - add a new VK_ACCELERATION_STRUCTURE_TYPE_GENERIC_KHR acceleration structure type for layering
- move VK_GEOMETRY_TYPE_INSTANCES_KHR to main enum instead of being added via extension
- make build commands more consistent - all now build multiple acceleration structures and are named plurally (vkCmdBuildAccelerationStructuresIndirectKHR, vkCmdBuildAccelerationStructuresKHR, vkBuildAccelerationStructuresKHR)
- add interactions with VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT for acceleration structures, including a new feature (descriptorBindingAccelerationStructureUpdateAfterBind) and 3 new properties (maxPerStageDescriptorAccelerationStructures, maxPerStageDescriptorUpdateAfterBindAccelerationStructures, maxDescriptorSetUpdateAfterBindAccelerationStructures)
- extension is no longer provisional

- define synchronization requirements for builds, traces, and copies
- define synchronization requirements for AS build inputs and indirect build buffer

(5) What is `VK_ACCELERATION_STRUCTURE_TYPE_GENERIC_KHR` for?

**RESOLVED**: It is primarily intended for API layering. In DXR, the acceleration structure is basically just a buffer in a special layout, and you do not know at creation time whether it will be used as a top or bottom level acceleration structure. We thus added a generic acceleration structure type whose type is unknown at creation time, but is specified at build time instead. Applications which are written directly for Vulkan should not use it.

**Version History**

- Revision 1, 2019-12-05 (Members of the Vulkan Ray Tracing TSG)
  - Internal revisions (forked from VK_NV_ray_tracing)
- Revision 2, 2019-12-20 (Daniel Koch, Eric Werness)
  - Add const version of DeviceOrHostAddress (!3515)
  - Add VU to clarify that only handles in the current pipeline are valid (!3518)
  - Restore some missing VUs and add in-place update language (#1902, !3522)
  - rename VkAccelerationStructureInstanceKHR member from accelerationStructure to accelerationStructureReference to better match its type (!3523)
  - Allow VK_ERROR_INVALID_OPAQUE_CAPTURE_ADDRESS for pipeline creation if shader group handles cannot be reused (!3523)
  - update documentation for the VK_ERROR_INVALID_OPAQUE_CAPTURE_ADDRESS error code and add missing documentation for new return codes from VK_KHR_deferred_host_operations (!3523)
  - list new query types for VK_KHR_ray_tracing (!3523)
  - Fix VU statements for VkAccelerationStructureGeometryKHR referring to correct union members and update to use more current wording (!3523)
- Revision 3, 2020-01-10 (Daniel Koch, Jon Leech, Christoph Kubisch)
  - Fix 'instance of' and 'that/which contains/defines' markup issues (!3528)
  - factor out VK_KHR_pipeline_library as stand-alone extension (!3540)
  - Resolve Vulkan-hpp issues (!3543)
    - add missing require for VkGeometryInstanceFlagsKHR
    - de-alias VK_STRUCTURE_TYPE_ACCELERATION_STRUCTURE_CREATE_INFO_NV since the KHR structure is no longer equivalent
    - add len to pDataSize attribute for vkWriteAccelerationStructuresPropertiesKHR
- Revision 4, 2020-01-23 (Daniel Koch, Eric Werness)
  - Improve vkWriteAccelerationStructuresPropertiesKHR, add return value and VUs (#1947)
  - Clarify language to allow multiple raygen shaders (#1959)

  - Various editorial feedback (!3556)
  - Add language to help deal with looped self-intersecting fans (#1901)
  - Change vkCmdTraceRays{,Indirect}KHR args to pointers (!3559)
  - Add scratch address validation language (#1941, !3551)
  - Fix definition and add hierarchy information for shader call scope (#1977, !3571)
- Revision 5, 2020-02-04 (Eric Werness, Jeff Bolz, Daniel Koch)
  - remove vestigial accelerationStructureUUID (!3582)
  - update definition of repack instructions and improve memory model interactions (#1910, #1913, !3584)
  - Fix wrong sType for VkPhysicalDeviceRayTracingFeaturesKHR (#1988)
  - Use provisional SPIR-V capabilities (#1987)
  - require rayTraversalPrimitiveCulling if rayQuery is supported (#1927)
  - Miss shaders do not have object parameters (!3592)
  - Fix missing required types in XML (!3592)
  - clarify matching conditions for update (!3592)
  - add goal that host and device builds be similar (!3592)
  - clarify that `maxPrimitiveCount` limit should apply to triangles and AABBs (!3592)
  - Require alignment for instance arrayOfPointers (!3592)
  - Zero is a valid value for instance flags (!3592)
  - Add some alignment VUs that got lost in refactoring (!3592)
  - Recommend TMin epsilon rather than culling (!3592)
  - Get angle from dot product not cross product (!3592)
  - Clarify that AH can access the payload and attributes (!3592)
  - Match DXR behavior for inactive primitive definition (!3592)
  - Use a more generic term than degenerate for inactive to avoid confusion (!3592)
- Revision 6, 2020-02-20 (Daniel Koch)
  - fix some dangling NV references (#1996)
  - rename VkCmdTraceRaysIndirectCommandKHR to VkTraceRaysIndirectCommandKHR (!3607)
  - update contributor list (!3611)
  - use uint64_t instead of VkAccelerationStructureReferenceKHR in VkAccelerationStructureInstanceKHR (#2004)
- Revision 7, 2020-02-28 (Tobias Hector)
  - remove HitTKHR SPIR-V builtin (spirv/spirv-extensions#7)
- Revision 8, 2020-03-06 (Tobias Hector, Dae Kim, Daniel Koch, Jeff Bolz, Eric Werness)

  - explicitly state that Tmax is updated when new closest intersection is accepted (#2020,!3536)
  - Made references to min and max t values consistent (!3644)
  - finish enumerating differences relative to VK_NV_ray_tracing in issues (1) and (2) (#1974,!3642)
  - fix formatting in some math equations (!3642)
  - Restrict the Hit Kind operand of `OpReportIntersectionKHR` to 7-bits (spirv/spirv-extensions#8,!3646)
  - Say ray tracing '**should**' be watertight (#2008,!3631)
  - Clarify memory requirements for ray tracing buffers (#2005,!3649)
  - Add callable size limits (#1997,!3652)
- Revision 9, 2020-04-15 (Eric Werness, Daniel Koch, Tobias Hector, Joshua Barczak)
  - Add geometry flags to acceleration structure creation (!3672)
  - add build scratch memory alignment (minAccelerationStructureScratchOffsetAlignment) (#2065,!3725)
  - fix naming and return enum from vkGetDeviceAccelerationStructureCompatibilityKHR (#2051,!3726)
  - require SPIR-V 1.4 (#2096,!3777)
  - added creation time capture/replay flags (#2104,!3774)
  - require Vulkan 1.1 (#2133,!3806)
  - use device addresses instead of VkBuffers for ray tracing commands (#2074,!3815)
  - add interactions with Vulkan 1.2 and VK_KHR_vulkan_memory_model (#2133,!3830)
  - make VK_KHR_pipeline_library an interaction instead of required (#2045,#2108,!3830)
  - make VK_KHR_deferred_host_operations an interaction instead of required (#2045,!3830)
  - removed maxCallableSize and added explicit stack size management for ray pipelines (#1997,!3817,!3772,!3844)
  - improved documentation for VkAccelerationStructureVersionInfoKHR (#2135,3835)
  - rename VkAccelerationStructureBuildOffsetInfoKHR to VkAccelerationStructureBuildRangeInfoKHR (#2058,!3754)
  - Re-unify geometry description between build and create (!3754)
  - Fix ppGeometries ambiguity, add pGeometries (#2032,!3811)
  - add interactions with VK_EXT_robustness2 and allow nullDescriptor support for acceleration structures (#1920,!3848)
  - added future extensibility for AS updates (#2114,!3849)
  - Fix VU for dispatchrays and add a limit on the size of the full grid (#2160,!3851)
  - Add shaderGroupHandleAlignment property (#2180,!3875)
  - Clarify deferred host ops for pipeline creation (#2067,!3813)
  - Change acceleration structure build to always be sized

(#2131,#2197,#2198,!3854,!3883,!3880)

- Revision 10, 2020-07-03 (Mathieu Robart, Daniel Koch, Eric Werness, Tobias Hector)
  - Decomposition of the specification, from VK_KHR_ray_tracing to VK_KHR_acceleration_structure (#1918,!3912)
  - clarify buffer usage flags for ray tracing (#2181,!3939)
  - add max primitive counts to build indirect command (#2233,!3944)
  - Allocate acceleration structures from VkBuffers and add a mode to constrain the device address (#2131,!3936)
  - Move VK_GEOMETRY_TYPE_INSTANCES_KHR to main enum (#2243,!3952)
  - make build commands more consistent (#2247,!3958)
  - add interactions with UPDATE_AFTER_BIND (#2128,!3986)
  - correct and expand build command VUs (!4020)
  - fix copy command VUs (!4018)
  - added various alignment requirements (#2229,!3943)
  - fix valid usage for arrays of geometryCount items (#2198,!4010)
  - define what is allowed to change on RTAS updates and relevant VUs (#2177,!3961)
- Revision 11, 2020-11-12 (Eric Werness, Josh Barczak, Daniel Koch, Tobias Hector)
  - de-alias NV and KHR acceleration structure types and associated commands (#2271,!4035)
  - specify alignment for host copy commands (#2273,!4037)
  - document `VK_FORMAT_FEATURE_ACCELERATION_STRUCTURE_VERTEX_BUFFER_BIT_KHR`
  - specify that acceleration structures are non-linear (#2289,!4068)
  - add several missing VUs for strides, vertexFormat, and indexType (#2315,!4069)
  - restore VUs for VkAccelerationStructureBuildGeometryInfoKHR (#2337,!4098)
  - ban multi-instance memory for host operations (#2324,!4102)
  - allow dstAccelerationStructure to be null for vkGetAccelerationStructureBuildSizesKHR (#2330,!4111)
  - more build VU cleanup (#2138,#4130)
  - specify host endianness for AS serialization (#2261,!4136)
  - add invertible transform matrix VU (#1710,!4140)
  - require geometryCount to be 1 for TLAS builds (!4145)
  - improved validity conditions for build addresses (#4142)
  - add single statement SPIR-V VUs, build limit VUs (!4158)
  - document limits for vertex and aabb strides (#2390,!4184)
  - specify that `VK_PIPELINE_STAGE_ACCELERATION_STRUCTURE_BUILD_BIT_KHR` applies to AS copies (#2382,#4173)
  - define sync for AS build inputs and indirect buffer (#2407,!4208)

- Revision 12, 2021-08-06 (Samuel Bourasseau)
  - rename VK_GEOMETRY_INSTANCE_TRIANGLE_FRONT_COUNTERCLOCKWISE_BIT_KHR to VK_GEOMETRY_INSTANCE_TRIANGLE_FLIP_FACING_BIT_KHR (keep previous as alias).
  - Clarify description and add note.
- Revision 13, 2021-09-30 (Jon Leech)
  - Add interaction with `VK_KHR_format_feature_flags2` to `vk.xml`

## VK_KHR_android_surface

### Name String

`VK_KHR_android_surface`

### Extension Type

Instance extension

### Registered Extension Number

9

### Revision

6

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_surface

### Contact

- Jesse Hall [GitHub]critsec

### Other Extension Metadata

### Last Modified Date

2016-01-14

### IP Status

No known IP claims.

### Contributors

- Patrick Doane, Blizzard
- Faith Ekstrand, Intel
- Ian Elliott, LunarG
- Courtney Goeltzenleuchter, LunarG
- Jesse Hall, Google
- James Jones, NVIDIA

- Antoine Labour, Google
- Jon Leech, Khronos
- David Mao, AMD
- Norbert Nopper, Freescale
- Alon Or-bach, Samsung
- Daniel Rakos, AMD
- Graham Sellers, AMD
- Ray Smith, ARM
- Jeff Vigil, Qualcomm
- Chia-I Wu, LunarG

### Description

The `VK_KHR_android_surface` extension is an instance extension. It provides a mechanism to create a VkSurfaceKHR object (defined by the `VK_KHR_surface` extension) that refers to an ANativeWindow, Android's native surface type. The ANativeWindow represents the producer endpoint of any buffer queue, regardless of consumer endpoint. Common consumer endpoints for `ANativeWindows` are the system window compositor, video encoders, and application-specific compositors importing the images through a `SurfaceTexture`.

### New Base Types

- ANativeWindow

### New Commands

- vkCreateAndroidSurfaceKHR

### New Structures

- VkAndroidSurfaceCreateInfoKHR

### New Bitmasks

- VkAndroidSurfaceCreateFlagsKHR

### New Enum Constants

- `VK_KHR_ANDROID_SURFACE_EXTENSION_NAME`
- `VK_KHR_ANDROID_SURFACE_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_ANDROID_SURFACE_CREATE_INFO_KHR`

### Issues

1) Does Android need a way to query for compatibility between a particular physical device (and queue family?) and a specific Android display?

**RESOLVED**: No. Currently on Android, any physical device is expected to be able to present to the system compositor, and all queue families must support the necessary image layout transitions and synchronization operations.

### Version History

- Revision 1, 2015-09-23 (Jesse Hall)
  - Initial draft.
- Revision 2, 2015-10-26 (Ian Elliott)
  - Renamed from VK_EXT_KHR_android_surface to VK_KHR_android_surface.
- Revision 3, 2015-11-03 (Daniel Rakos)
  - Added allocation callbacks to surface creation function.
- Revision 4, 2015-11-10 (Jesse Hall)
  - Removed VK_ERROR_INVALID_ANDROID_WINDOW_KHR.
- Revision 5, 2015-11-28 (Daniel Rakos)
  - Updated the surface create function to take a pCreateInfo structure.
- Revision 6, 2016-01-14 (James Jones)
  - Moved VK_ERROR_NATIVE_WINDOW_IN_USE_KHR from the VK_KHR_android_surface to the VK_KHR_surface extension.

## VK_KHR_buffer_device_address

### Name String

`VK_KHR_buffer_device_address`

### Extension Type

Device extension

### Registered Extension Number

258

### Revision

1

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
and

VK_KHR_device_group

or

Vulkan Version 1.1

### SPIR-V Dependencies

- SPV_KHR_physical_storage_buffer

### Deprecation State

- *Promoted* to Vulkan 1.2

### Contact

- Jeff Bolz [GitHub]jeffbolznv

### Other Extension Metadata

### Last Modified Date

2019-06-24

### IP Status

No known IP claims.

### Interactions and External Dependencies

- This extension provides API support for `GL_EXT_buffer_reference` and `GL_EXT_buffer_reference2` and `GL_EXT_buffer_reference_uvec2`

### Contributors

- Jeff Bolz, NVIDIA
- Neil Henning, AMD
- Tobias Hector, AMD
- Faith Ekstrand, Intel
- Baldur Karlsson, Valve
- Jan-Harald Fredriksen, Arm

### Description

This extension allows the application to query a 64-bit buffer device address value for a buffer, which can be used to access the buffer memory via the `PhysicalStorageBuffer` storage class in the `GL_EXT_buffer_reference` GLSL extension and `SPV_KHR_physical_storage_buffer` SPIR-V extension.

Another way to describe this extension is that it adds “pointers to buffer memory in shaders”. By calling vkGetBufferDeviceAddressKHR with a `VkBuffer`, it will return a VkDeviceAddress value which represents the address of the start of the buffer.

vkGetBufferOpaqueCaptureAddressKHR and vkGetDeviceMemoryOpaqueCaptureAddressKHR allow opaque addresses for buffers and memory objects to be queried for the current process. A trace capture and replay tool can then supply these addresses to be used at replay time to match the addresses used when the trace was captured. To enable tools to insert these queries, new memory allocation flags must be specified for memory objects that will be bound to buffers accessed via the

`PhysicalStorageBuffer` storage class. **Note that this mechanism is intended only to support capture/replay tools, and is not recommended for use in other applications.**

### Promotion to Vulkan 1.2

All functionality in this extension is included in core Vulkan 1.2, with the KHR suffix omitted. However, if Vulkan 1.2 is supported and this extension is not, the `bufferDeviceAddress` feature is optional. The original type, enum and command names are still available as aliases of the core functionality.

### Promotion to Vulkan 1.3

Support for the `bufferDeviceAddress` feature is mandatory in Vulkan 1.3, regardless of whether this extension is supported.

### New Commands

- vkGetBufferDeviceAddressKHR
- vkGetBufferOpaqueCaptureAddressKHR
- vkGetDeviceMemoryOpaqueCaptureAddressKHR

### New Structures

- VkBufferDeviceAddressInfoKHR
- VkDeviceMemoryOpaqueCaptureAddressInfoKHR
- Extending VkBufferCreateInfo:
  - VkBufferOpaqueCaptureAddressCreateInfoKHR
- Extending VkMemoryAllocateInfo:
  - VkMemoryOpaqueCaptureAddressAllocateInfoKHR
- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceBufferDeviceAddressFeaturesKHR

### New Enum Constants

- `VK_KHR_BUFFER_DEVICE_ADDRESS_EXTENSION_NAME`
- `VK_KHR_BUFFER_DEVICE_ADDRESS_SPEC_VERSION`
- Extending VkBufferCreateFlagBits:
  - `VK_BUFFER_CREATE_DEVICE_ADDRESS_CAPTURE_REPLAY_BIT_KHR`
- Extending VkBufferUsageFlagBits:
  - `VK_BUFFER_USAGE_SHADER_DEVICE_ADDRESS_BIT_KHR`
- Extending VkMemoryAllocateFlagBits:
  - `VK_MEMORY_ALLOCATE_DEVICE_ADDRESS_BIT_KHR`
  - `VK_MEMORY_ALLOCATE_DEVICE_ADDRESS_CAPTURE_REPLAY_BIT_KHR`

- Extending VkResult:
  - VK_ERROR_INVALID_OPAQUE_CAPTURE_ADDRESS_KHR
- Extending VkStructureType:
  - VK_STRUCTURE_TYPE_BUFFER_DEVICE_ADDRESS_INFO_KHR
  - VK_STRUCTURE_TYPE_BUFFER_OPAQUE_CAPTURE_ADDRESS_CREATE_INFO_KHR
  - VK_STRUCTURE_TYPE_DEVICE_MEMORY_OPAQUE_CAPTURE_ADDRESS_INFO_KHR
  - VK_STRUCTURE_TYPE_MEMORY_OPAQUE_CAPTURE_ADDRESS_ALLOCATE_INFO_KHR
  - VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_BUFFER_DEVICE_ADDRESS_FEATURES_KHR

#### New SPIR-V Capabilities

- PhysicalStorageBufferAddresses

#### Version History

- Revision 1, 2019-06-24 (Jan-Harald Fredriksen)
  - Internal revisions based on VK_EXT_buffer_device_address

## VK_KHR_calibrated_timestamps

#### Name String

VK_KHR_calibrated_timestamps

#### Extension Type

Device extension

#### Registered Extension Number

544

#### Revision

1

#### Ratification Status

Ratified

#### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

#### Contact

- Daniel Rakos [GitHub]aqnuep

#### Extension Proposal

VK_EXT_calibrated_timestamps

### Other Extension Metadata

#### Last Modified Date

2023-07-12

#### IP Status

No known IP claims.

#### Contributors

- Matthaeus G. Chajdas, AMD
- Alan Harrison, AMD
- Derrick Owens, AMD
- Daniel Rakos, RasterGrid
- Faith Ekstrand, Intel
- Keith Packard, Valve

### Description

This extension provides an interface to query calibrated timestamps obtained quasi simultaneously from two time domains.

### New Commands

- vkGetCalibratedTimestampsKHR
- vkGetPhysicalDeviceCalibrateableTimeDomainsKHR

### New Structures

- VkCalibratedTimestampInfoKHR

### New Enums

- VkTimeDomainKHR

### New Enum Constants

- `VK_KHR_CALIBRATED_TIMESTAMPS_EXTENSION_NAME`
- `VK_KHR_CALIBRATED_TIMESTAMPS_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_CALIBRATED_TIMESTAMP_INFO_KHR`

### Version History

- Revision 1, 2023-07-12 (Daniel Rakos)
  - Initial draft.

# VK_KHR_compute_shader_derivatives

### Name String

`VK_KHR_compute_shader_derivatives`

### Extension Type

Device extension

### Registered Extension Number

512

### Revision

1

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2

### SPIR-V Dependencies

- SPV_KHR_compute_shader_derivatives

### Contact

- Jean-Noe Morissette [GitHub]MagicPoncho

### Extension Proposal

VK_KHR_compute_shader_derivatives

### Other Extension Metadata

### Last Modified Date

2024-06-26

### IP Status

No known IP claims.

### Interactions and External Dependencies

- This extension requires `SPV_KHR_compute_shader_derivatives`
- This extension provides API support for `GL_KHR_compute_shader_derivatives`

### Contributors

- Jean-Noe Morissette, Epic Games
- Daniel Koch, NVIDIA
- Pat Brown, NVIDIA
- Stu Smith, AMD
- Jan-Harald Fredriksen, Arm

- Tobias Hector, AMD
- Ralph Potter, Samsung
- Pan Gao, Huawei
- Samuel (Sheng-Wen) Huang, MediaTek
- Graeme Leese, Broadcom
- Hans-Kristian Arntzen, Valve
- Matthew Netsh, Qualcomm

### Description

This extension adds Vulkan support for the `SPV_KHR_compute_shader_derivatives` SPIR-V extension.

The SPIR-V extension provides two new execution modes, both of which allow execution models with defined workgroups to use built-ins that evaluate derivatives explicitly or implicitly. Derivatives will be computed via differencing over a 2x2 group of shader invocations. The `DerivativeGroupQuadsKHR` execution mode assembles shader invocations into 2x2 groups, where each group has x and y coordinates of the local invocation ID of the form (2m+{0,1}, 2n+{0,1}). The `DerivativeGroupLinearKHR` execution mode assembles shader invocations into 2x2 groups, where each group has local invocation index values of the form 4m+{0,1,2,3}.

The new execution modes are supported in compute shaders and optionally (see meshAndTaskShaderDerivatives) in mesh and task shaders.

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceComputeShaderDerivativesFeaturesKHR
- Extending VkPhysicalDeviceProperties2:
  - VkPhysicalDeviceComputeShaderDerivativesPropertiesKHR

### New Enum Constants

- `VK_KHR_COMPUTE_SHADER_DERIVATIVES_EXTENSION_NAME`
- `VK_KHR_COMPUTE_SHADER_DERIVATIVES_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_COMPUTE_SHADER_DERIVATIVES_FEATURES_KHR`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_COMPUTE_SHADER_DERIVATIVES_PROPERTIES_KHR`

### New SPIR-V Capability

- `ComputeDerivativeGroupQuadsKHR`
- `ComputeDerivativeGroupLinearKHR`

### Examples

None.

### Version History

- Revision 1, 2023-02-27 (Jean-Noe Morissette)
  - Initial draft
  - Add properties and clarify mesh and task support (Daniel Koch)

## VK_KHR_cooperative_matrix

### Name String

`VK_KHR_cooperative_matrix`

### Extension Type

Device extension

### Registered Extension Number

507

### Revision

2

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### SPIR-V Dependencies

- SPV_KHR_cooperative_matrix

### Contact

- Kevin Petit [GitHub]kpet

### Extension Proposal

VK_KHR_cooperative_matrix

### Other Extension Metadata

### Last Modified Date

2023-05-03

### Interactions and External Dependencies

- This extension provides API support for `GLSL_KHR_cooperative_matrix`

### Contributors

- Jeff Bolz, NVIDIA
- Markus Tavenrath, NVIDIA
- Daniel Koch, NVIDIA
- Kevin Petit, Arm Ltd.
- Boris Zanin, AMD

### Description

This extension adds support for using cooperative matrix types in SPIR-V. Cooperative matrix types are medium-sized matrices that are primarily supported in compute shaders, where the storage for the matrix is spread across all invocations in some scope (usually a subgroup) and those invocations cooperate to efficiently perform matrix multiplies.

Cooperative matrix types are defined by the `SPV_KHR_cooperative_matrix` SPIR-V extension and can be used with the `GLSL_KHR_cooperative_matrix` GLSL extension.

This extension includes support for enumerating the matrix types and dimensions that are supported by the implementation.

### New Commands

- vkGetPhysicalDeviceCooperativeMatrixPropertiesKHR

### New Structures

- VkCooperativeMatrixPropertiesKHR
- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceCooperativeMatrixFeaturesKHR
- Extending VkPhysicalDeviceProperties2:
  - VkPhysicalDeviceCooperativeMatrixPropertiesKHR

### New Enums

- VkComponentTypeKHR
- VkScopeKHR

### New Enum Constants

- `VK_KHR_COOPERATIVE_MATRIX_EXTENSION_NAME`
- `VK_KHR_COOPERATIVE_MATRIX_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_COOPERATIVE_MATRIX_PROPERTIES_KHR`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_COOPERATIVE_MATRIX_FEATURES_KHR`

- `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_COOPERATIVE_MATRIX_PROPERTIES_KHR`

### New SPIR-V Capabilities

- CooperativeMatrixKHR

### Issues

### Version History

- Revision 2, 2023-05-03 (Kevin Petit)
  - First KHR revision
- Revision 1, 2019-02-05 (Jeff Bolz)
  - NVIDIA vendor extension

## VK_KHR_copy_commands2

### Name String

`VK_KHR_copy_commands2`

### Extension Type

Device extension

### Registered Extension Number

338

### Revision

1

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### Deprecation State

- *Promoted* to Vulkan 1.3

### Contact

- Matthew Netsch [GitHub]mnetsch

### Other Extension Metadata

### Last Modified Date

2020-07-06

### Contributors

- Jeff Leger, Qualcomm
- Tobias Hector, AMD
- Jan-Harald Fredriksen, ARM
- Tom Olson, ARM

### Description

This extension provides extensible versions of the Vulkan buffer and image copy commands. The new commands are functionally identical to the core commands, except that their copy parameters are specified using extensible structures that can be used to pass extension-specific information.

The following extensible copy commands are introduced with this extension: vkCmdCopyBuffer2KHR, vkCmdCopyImage2KHR, vkCmdCopyBufferToImage2KHR, vkCmdCopyImageToBuffer2KHR, vkCmdBlitImage2KHR, and vkCmdResolveImage2KHR. Each command contains an `*Info2KHR` structure parameter that includes `sType`/`pNext` members. Lower level structures describing each region to be copied are also extended with `sType`/`pNext` members.

### New Commands

- vkCmdBlitImage2KHR
- vkCmdCopyBuffer2KHR
- vkCmdCopyBufferToImage2KHR
- vkCmdCopyImage2KHR
- vkCmdCopyImageToBuffer2KHR
- vkCmdResolveImage2KHR

### New Structures

- VkBlitImageInfo2KHR
- VkBufferCopy2KHR
- VkBufferImageCopy2KHR
- VkCopyBufferInfo2KHR
- VkCopyBufferToImageInfo2KHR
- VkCopyImageInfo2KHR
- VkCopyImageToBufferInfo2KHR
- VkImageBlit2KHR
- VkImageCopy2KHR
- VkImageResolve2KHR
- VkResolveImageInfo2KHR

### New Enum Constants

- `VK_KHR_COPY_COMMANDS_2_EXTENSION_NAME`
- `VK_KHR_COPY_COMMANDS_2_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_BLIT_IMAGE_INFO_2_KHR`
  - `VK_STRUCTURE_TYPE_BUFFER_COPY_2_KHR`
  - `VK_STRUCTURE_TYPE_BUFFER_IMAGE_COPY_2_KHR`
  - `VK_STRUCTURE_TYPE_COPY_BUFFER_INFO_2_KHR`
  - `VK_STRUCTURE_TYPE_COPY_BUFFER_TO_IMAGE_INFO_2_KHR`
  - `VK_STRUCTURE_TYPE_COPY_IMAGE_INFO_2_KHR`
  - `VK_STRUCTURE_TYPE_COPY_IMAGE_TO_BUFFER_INFO_2_KHR`
  - `VK_STRUCTURE_TYPE_IMAGE_BLIT_2_KHR`
  - `VK_STRUCTURE_TYPE_IMAGE_COPY_2_KHR`
  - `VK_STRUCTURE_TYPE_IMAGE_RESOLVE_2_KHR`
  - `VK_STRUCTURE_TYPE_RESOLVE_IMAGE_INFO_2_KHR`

### Promotion to Vulkan 1.3

Functionality in this extension is included in core Vulkan 1.3, with the KHR suffix omitted. The original type, enum and command names are still available as aliases of the core functionality.

### Version History

- Revision 1, 2020-07-06 (Jeff Leger)
  - Internal revisions

## VK_KHR_create_renderpass2

### Name String

`VK_KHR_create_renderpass2`

### Extension Type

Device extension

### Registered Extension Number

110

### Revision

1

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_multiview
and
VK_KHR_maintenance2
or
Vulkan Version 1.1

### Deprecation State

- *Promoted* to Vulkan 1.2

### Contact

- Tobias Hector [GitHub]tobias

### Other Extension Metadata

### Last Modified Date

2018-02-07

### Contributors

- Tobias Hector
- Jeff Bolz

### Description

This extension provides a new command to create render passes in a way that can be easily extended by other extensions through the substructures of render pass creation. The Vulkan 1.0 render pass creation sub-structures do not include `sType`/`pNext` members. Additionally, the render pass begin/next/end commands have been augmented with new extensible structures for passing additional subpass information.

The VkRenderPassMultiviewCreateInfo and VkInputAttachmentAspectReference structures that extended the original VkRenderPassCreateInfo are not accepted into the new creation functions, and instead their parameters are folded into this extension as follows:

- Elements of VkRenderPassMultiviewCreateInfo::`pViewMasks` are now specified in VkSubpassDescription2KHR::`viewMask`.
- Elements of VkRenderPassMultiviewCreateInfo::`pViewOffsets` are now specified in VkSubpassDependency2KHR::`viewOffset`.
- VkRenderPassMultiviewCreateInfo::`correlationMaskCount` and VkRenderPassMultiviewCreateInfo::`pCorrelationMasks` are directly specified in VkRenderPassCreateInfo2KHR.
- VkInputAttachmentAspectReference::`aspectMask` is now specified in the relevant input attachment reference in VkAttachmentReference2KHR::`aspectMask`

The details of these mappings are explained fully in the new structures.

### Promotion to Vulkan 1.2

All functionality in this extension is included in core Vulkan 1.2, with the KHR suffix omitted. The original type, enum and command names are still available as aliases of the core functionality.

### New Commands

- vkCmdBeginRenderPass2KHR
- vkCmdEndRenderPass2KHR
- vkCmdNextSubpass2KHR
- vkCreateRenderPass2KHR

### New Structures

- VkAttachmentDescription2KHR
- VkAttachmentReference2KHR
- VkRenderPassCreateInfo2KHR
- VkSubpassBeginInfoKHR
- VkSubpassDependency2KHR
- VkSubpassDescription2KHR
- VkSubpassEndInfoKHR

### New Enum Constants

- `VK_KHR_CREATE_RENDERPASS_2_EXTENSION_NAME`
- `VK_KHR_CREATE_RENDERPASS_2_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_ATTACHMENT_DESCRIPTION_2_KHR`
  - `VK_STRUCTURE_TYPE_ATTACHMENT_REFERENCE_2_KHR`
  - `VK_STRUCTURE_TYPE_RENDER_PASS_CREATE_INFO_2_KHR`
  - `VK_STRUCTURE_TYPE_SUBPASS_BEGIN_INFO_KHR`
  - `VK_STRUCTURE_TYPE_SUBPASS_DEPENDENCY_2_KHR`
  - `VK_STRUCTURE_TYPE_SUBPASS_DESCRIPTION_2_KHR`
  - `VK_STRUCTURE_TYPE_SUBPASS_END_INFO_KHR`

### Version History

- Revision 1, 2018-02-07 (Tobias Hector)
  - Internal revisions

## VK_KHR_deferred_host_operations

### Name String

`VK_KHR_deferred_host_operations`

### Extension Type

Device extension

### Registered Extension Number

269

### Revision

4

### Ratification Status

Ratified

### Extension and Version Dependencies

None

### Contact

- Josh Barczak [GitHub]jbarczak

### Other Extension Metadata

### Last Modified Date

2020-11-12

### IP Status

No known IP claims.

### Contributors

- Joshua Barczak, Intel
- Jeff Bolz, NVIDIA
- Daniel Koch, NVIDIA
- Slawek Grajewski, Intel
- Tobias Hector, AMD
- Yuriy O'Donnell, Epic
- Eric Werness, NVIDIA
- Baldur Karlsson, Valve
- Jesse Barker, Unity
- Contributors to VK_KHR_acceleration_structure, VK_KHR_ray_tracing_pipeline

### Description

The `VK_KHR_deferred_host_operations` extension defines the infrastructure and usage patterns for deferrable commands, but does not specify any commands as deferrable. This is left to additional dependent extensions. Commands **must** not be deferred unless the deferral is specifically allowed by another extension which depends on `VK_KHR_deferred_host_operations`.

### New Object Types

- VkDeferredOperationKHR

### New Commands

- vkCreateDeferredOperationKHR
- vkDeferredOperationJoinKHR
- vkDestroyDeferredOperationKHR
- vkGetDeferredOperationMaxConcurrencyKHR
- vkGetDeferredOperationResultKHR

### New Enum Constants

- `VK_KHR_DEFERRED_HOST_OPERATIONS_EXTENSION_NAME`
- `VK_KHR_DEFERRED_HOST_OPERATIONS_SPEC_VERSION`
- Extending VkObjectType:
  - `VK_OBJECT_TYPE_DEFERRED_OPERATION_KHR`
- Extending VkResult:
  - `VK_OPERATION_DEFERRED_KHR`
  - `VK_OPERATION_NOT_DEFERRED_KHR`
  - `VK_THREAD_DONE_KHR`
  - `VK_THREAD_IDLE_KHR`

### Code Examples

The following examples will illustrate the concept of deferrable operations using a hypothetical example. The command `vkDoSomethingExpensive` denotes a deferrable command.

The following example illustrates how a vulkan application might request deferral of an expensive operation:

```
// create a deferred operation
VkDeferredOperationKHR hOp;
VkResult result = vkCreateDeferredOperationKHR(device, pCallbacks, &hOp);
assert(result == VK_SUCCESS);

result = vkDoSomethingExpensive(device, hOp, ...);
```

```
assert( result == VK_OPERATION_DEFERRED_KHR );

// operation was deferred.  Execute it asynchronously
std::async::launch(
    [ hOp ] ( )
    {
        vkDeferredOperationJoinKHR(device, hOp);

        result = vkGetDeferredOperationResultKHR(device, hOp);

        // deferred operation is now complete.  'result' indicates success or failure

        vkDestroyDeferredOperationKHR(device, hOp, pCallbacks);
    }
);
```

The following example illustrates extracting concurrency from a single deferred operation:

```
// create a deferred operation
VkDeferredOperationKHR hOp;
VkResult result = vkCreateDeferredOperationKHR(device, pCallbacks, &hOp);
assert(result == VK_SUCCESS);

result = vkDoSomethingExpensive(device, hOp, ...);
assert( result == VK_OPERATION_DEFERRED_KHR );

// Query the maximum amount of concurrency and clamp to the desired maximum
uint32_t numLaunches = std::min(vkGetDeferredOperationMaxConcurrencyKHR(device, hOp),
maxThreads);

std::vector<std::future<void> > joins;

for (uint32_t i = 0; i < numLaunches; i++) {
  joins.emplace_back(std::async::launch(
    [ hOp ] ( )
    {
        vkDeferredOperationJoinKHR(device, hOp);
                // in a job system, a return of VK_THREAD_IDLE_KHR should queue
another
                // job, but it is not functionally required
    }
  ));
}

for (auto &f : joins) {
  f.get();
}

result = vkGetDeferredOperationResultKHR(device, hOp);
```

```
// deferred operation is now complete.  'result' indicates success or failure

vkDestroyDeferredOperationKHR(device, hOp, pCallbacks);
```

The following example shows a subroutine which guarantees completion of a deferred operation, in the presence of multiple worker threads, and returns the result of the operation.

```
VkResult FinishDeferredOperation(VkDeferredOperationKHR hOp)
{
    // Attempt to join the operation until the implementation indicates that we should
stop

    VkResult result = vkDeferredOperationJoinKHR(device, hOp);
    while( result == VK_THREAD_IDLE_KHR )
    {
        std::this_thread::yield();
        result = vkDeferredOperationJoinKHR(device, hOp);
    }

    switch( result )
    {
    case VK_SUCCESS:
        {
            // deferred operation has finished.  Query its result.
            result = vkGetDeferredOperationResultKHR(device, hOp);
        }
        break;

    case VK_THREAD_DONE_KHR:
        {
            // deferred operation is being wrapped up by another thread
            //  wait for that thread to finish
            do
            {
                std::this_thread::yield();
                result = vkGetDeferredOperationResultKHR(device, hOp);
            } while( result == VK_NOT_READY );
        }
        break;

    default:
        assert(false); // other conditions are illegal.
        break;
    }

    return result;
}
```

### Issues

1. Should this extension have a VkPhysicalDevice*FeaturesKHR structure?

**RESOLVED**: No. This extension does not add any functionality on its own and requires a dependent extension to actually enable functionality and thus there is no value in adding a feature structure. If necessary, any dependent extension could add a feature boolean if it wanted to indicate that it is adding optional deferral support.

### Version History

- Revision 1, 2019-12-05 (Josh Barczak, Daniel Koch)
  - Initial draft.
- Revision 2, 2020-03-06 (Daniel Koch, Tobias Hector)
  - Add missing VK_OBJECT_TYPE_DEFERRED_OPERATION_KHR enum
  - fix sample code
  - Clarified deferred operation parameter lifetimes (#2018,!3647)
- Revision 3, 2020-05-15 (Josh Barczak)
  - Clarify behavior of vkGetDeferredOperationMaxConcurrencyKHR, allowing it to return 0 if the operation is complete (#2036,!3850)
- Revision 4, 2020-11-12 (Tobias Hector, Daniel Koch)
  - Remove VkDeferredOperationInfoKHR and change return value semantics when deferred host operations are in use (#2067,3813)
  - clarify return value of vkGetDeferredOperationResultKHR (#2339,!4110)

## VK_KHR_depth_stencil_resolve

### Name String

`VK_KHR_depth_stencil_resolve`

### Extension Type

Device extension

### Registered Extension Number

200

### Revision

1

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_create_renderpass2
or

Vulkan Version 1.2

### Deprecation State

- *Promoted* to Vulkan 1.2

### Contact

- Jan-Harald Fredriksen [GitHub]janharald

### Other Extension Metadata

### Last Modified Date

2018-04-09

### Contributors

- Jan-Harald Fredriksen, Arm
- Andrew Garrard, Samsung Electronics
- Soowan Park, Samsung Electronics
- Jeff Bolz, NVIDIA
- Daniel Rakos, AMD

### Description

This extension adds support for automatically resolving multisampled depth/stencil attachments in a subpass in a similar manner as for color attachments.

Multisampled color attachments can be resolved at the end of a subpass by specifying `pResolveAttachments` entries corresponding to the `pColorAttachments` array entries. This does not allow for a way to map the resolve attachments to the depth/stencil attachment. The vkCmdResolveImage command does not allow for depth/stencil images. While there are other ways to resolve the depth/stencil attachment, they can give sub-optimal performance. Extending the `VkSubpassDescription2` in this extension allows an application to add a `pDepthStencilResolveAttachment`, that is similar to the color `pResolveAttachments`, that the `pDepthStencilAttachment` can be resolved into.

Depth and stencil samples are resolved to a single value based on the resolve mode. The set of possible resolve modes is defined in the VkResolveModeFlagBits enum. The `VK_RESOLVE_MODE_SAMPLE_ZERO_BIT` mode is the only mode that is required of all implementations (that support the extension or support Vulkan 1.2 or higher). Some implementations may also support averaging (the same as color sample resolve) or taking the minimum or maximum sample, which may be more suitable for depth/stencil resolve.

### Promotion to Vulkan 1.2

All functionality in this extension is included in core Vulkan 1.2, with the KHR suffix omitted. The original type, enum and command names are still available as aliases of the core functionality.

### New Structures

- Extending VkPhysicalDeviceProperties2:
  - VkPhysicalDeviceDepthStencilResolvePropertiesKHR
- Extending VkSubpassDescription2:
  - VkSubpassDescriptionDepthStencilResolveKHR

### New Enums

- VkResolveModeFlagBitsKHR

### New Bitmasks

- VkResolveModeFlagsKHR

### New Enum Constants

- `VK_KHR_DEPTH_STENCIL_RESOLVE_EXTENSION_NAME`
- `VK_KHR_DEPTH_STENCIL_RESOLVE_SPEC_VERSION`
- Extending VkResolveModeFlagBits:
  - `VK_RESOLVE_MODE_AVERAGE_BIT_KHR`
  - `VK_RESOLVE_MODE_MAX_BIT_KHR`
  - `VK_RESOLVE_MODE_MIN_BIT_KHR`
  - `VK_RESOLVE_MODE_NONE_KHR`
  - `VK_RESOLVE_MODE_SAMPLE_ZERO_BIT_KHR`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DEPTH_STENCIL_RESOLVE_PROPERTIES_KHR`
  - `VK_STRUCTURE_TYPE_SUBPASS_DESCRIPTION_DEPTH_STENCIL_RESOLVE_KHR`

### Version History

- Revision 1, 2018-04-09 (Jan-Harald Fredriksen)
  - Initial revision

## VK_KHR_display

### Name String

`VK_KHR_display`

### Extension Type

Instance extension

### Registered Extension Number

3

**Revision**

23

**Ratification Status**

Ratified

**Extension and Version Dependencies**

VK_KHR_surface

**Contact**

- James Jones [GitHub]cubanismo
- Norbert Nopper [GitHub]FslNopper

**Other Extension Metadata**

**Last Modified Date**

2017-03-13

**IP Status**

No known IP claims.

**Contributors**

- James Jones, NVIDIA
- Norbert Nopper, Freescale
- Jeff Vigil, Qualcomm
- Daniel Rakos, AMD

**Description**

This extension provides the API to enumerate displays and available modes on a given device.

**New Object Types**

- VkDisplayKHR
- VkDisplayModeKHR

**New Commands**

- vkCreateDisplayModeKHR
- vkCreateDisplayPlaneSurfaceKHR
- vkGetDisplayModePropertiesKHR
- vkGetDisplayPlaneCapabilitiesKHR
- vkGetDisplayPlaneSupportedDisplaysKHR
- vkGetPhysicalDeviceDisplayPlanePropertiesKHR
- vkGetPhysicalDeviceDisplayPropertiesKHR

### New Structures

- VkDisplayModeCreateInfoKHR
- VkDisplayModeParametersKHR
- VkDisplayModePropertiesKHR
- VkDisplayPlaneCapabilitiesKHR
- VkDisplayPlanePropertiesKHR
- VkDisplayPropertiesKHR
- VkDisplaySurfaceCreateInfoKHR

### New Enums

- VkDisplayPlaneAlphaFlagBitsKHR

### New Bitmasks

- VkDisplayModeCreateFlagsKHR
- VkDisplayPlaneAlphaFlagsKHR
- VkDisplaySurfaceCreateFlagsKHR
- VkSurfaceTransformFlagsKHR

### New Enum Constants

- `VK_KHR_DISPLAY_EXTENSION_NAME`
- `VK_KHR_DISPLAY_SPEC_VERSION`
- Extending VkObjectType:
  - `VK_OBJECT_TYPE_DISPLAY_KHR`
  - `VK_OBJECT_TYPE_DISPLAY_MODE_KHR`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_DISPLAY_MODE_CREATE_INFO_KHR`
  - `VK_STRUCTURE_TYPE_DISPLAY_SURFACE_CREATE_INFO_KHR`

### Issues

1) Which properties of a mode should be fixed in the mode information vs. settable in some other function when setting the mode? E.g., do we need to double the size of the mode pool to include both stereo and non-stereo modes? YUV and RGB scanout even if they both take RGB input images? BGR vs. RGB input? etc.

**RESOLVED**: Many modern displays support at most a handful of resolutions and timings natively. Other “modes” are expected to be supported using scaling hardware on the display engine or GPU. Other properties, such as rotation and mirroring should not require duplicating hardware modes just to express all combinations. Further, these properties may be implemented on a per-display or per-overlay granularity.

To avoid the exponential growth of modes as mutable properties are added, as was the case with `EGLConfig`/WGL pixel formats/`GLXFBConfig`, this specification should separate out hardware properties and configurable state into separate objects. Modes and overlay planes will express capabilities of the hardware, while a separate structure will allow applications to configure scaling, rotation, mirroring, color keys, LUT values, alpha masks, etc. for a given swapchain independent of the mode in use. Constraints on these settings will be established by properties of the immutable objects.

Note the resolution of this issue may affect issue 5 as well.

2) What properties of a display itself are useful?

**RESOLVED**: This issue is too broad. It was meant to prompt general discussion, but resolving this issue amounts to completing this specification. All interesting properties should be included. The issue will remain as a placeholder since removing it would make it hard to parse existing discussion notes that refer to issues by number.

3) How are multiple overlay planes within a display or mode enumerated?

**RESOLVED**: They are referred to by an index. Each display will report the number of overlay planes it contains.

4) Should swapchains be created relative to a mode or a display?

**RESOLVED**: When using this extension, swapchains are created relative to a mode and a plane. The mode implies the display object the swapchain will present to. If the specified mode is not the display's current mode, the new mode will be applied when the first image is presented to the swapchain, and the default operating system mode, if any, will be restored when the swapchain is destroyed.

5) Should users query generic ranges from displays and construct their own modes explicitly using those constraints rather than querying a fixed set of modes (Most monitors only have one real "mode" these days, even though many support relatively arbitrary scaling, either on the monitor side or in the GPU display engine, making "modes" something of a relic/compatibility construct).

**RESOLVED**: Expose both. Display information structures will expose a set of predefined modes, as well as any attributes necessary to construct a customized mode.

6) Is it fine if we return the display and display mode handles in the structure used to query their properties?

**RESOLVED**: Yes.

7) Is there a possibility that not all displays of a device work with all of the present queues of a device? If yes, how do we determine which displays work with which present queues?

**RESOLVED**: No known hardware has such limitations, but determining such limitations is supported automatically using the existing `VK_KHR_surface` and `VK_KHR_swapchain` query mechanisms.

8) Should all presentation need to be done relative to an overlay plane, or can a display mode + display be used alone to target an output?

**RESOLVED**: Require specifying a plane explicitly.

9) Should displays have an associated window system display, such as an `HDC` or `Display*`?

**RESOLVED**: No. Displays are independent of any windowing system in use on the system. Further, neither `HDC` nor `Display*` refer to a physical display object.

10) Are displays queried from a physical GPU or from a device instance?

**RESOLVED**: Developers prefer to query modes directly from the physical GPU so they can use display information as an input to their device selection algorithms prior to device creation. This avoids the need to create placeholder device instances to enumerate displays.

This preference must be weighed against the extra initialization that must be done by driver vendors prior to device instance creation to support this usage.

11) Should displays and/or modes be dispatchable objects? If functions are to take displays, overlays, or modes as their first parameter, they must be dispatchable objects as defined in Khronos bug 13529. If they are not added to the list of dispatchable objects, functions operating on them must take some higher-level object as their first parameter. There is no performance case against making them dispatchable objects, but they would be the first extension objects to be dispatchable.

**RESOLVED**: Do not make displays or modes dispatchable. They will dispatch based on their associated physical device.

12) Should hardware cursor capabilities be exposed?

**RESOLVED**: Defer. This could be a separate extension on top of the base WSI specs.

13) How many display objects should be enumerated for "tiled" display devices? There are ongoing design discussions among lower-level display API authors regarding how to expose displays if they are one physical display device to an end user, but may internally be implemented as two side-by-side displays using the same display engine (and sometimes cabling) resources as two physically separate display devices.

**RESOLVED**: Tiled displays will appear as a single display object in this API.

14) Should the raw EDID data be included in the display information?

**RESOLVED**: No. A future extension could be added which reports the EDID if necessary. This may be complicated by the outcome of issue 13.

15) Should min and max scaling factor capabilities of overlays be exposed?

**RESOLVED**: Yes. This is exposed indirectly by allowing applications to query the min/max position and extent of the source and destination regions from which image contents are fetched by the display engine when using a particular mode and overlay pair.

16) Should devices be able to expose planes that can be moved between displays? If so, how?

**RESOLVED**: Yes. Applications can determine which displays a given plane supports using

vkGetDisplayPlaneSupportedDisplaysKHR.

17) Should there be a way to destroy display modes? If so, does it support destroying “built in” modes?

**RESOLVED**: Not in this extension. A future extension could add this functionality.

18) What should the lifetime of display and built-in display mode objects be?

**RESOLVED**: The lifetime of the instance. These objects cannot be destroyed. A future extension may be added to expose a way to destroy these objects and/or support display hotplug.

19) Should persistent mode for smart panels be enabled/disabled at swapchain creation time, or on a per-present basis.

**RESOLVED**: On a per-present basis.

### Examples

| | |
|---|---|
| **NOTE** | The example code for the `VK_KHR_display` and `VK_KHR_display_swapchain` extensions was removed from the appendix after revision 1.0.43. The display enumeration example code was ported to the cube demo that is shipped with the official Khronos SDK, and is being kept up-to-date in that location (see: https://github.com/KhronosGroup/Vulkan-Tools/blob/main/cube/cube.c). |

### Version History

- Revision 1, 2015-02-24 (James Jones)
  - Initial draft
- Revision 2, 2015-03-12 (Norbert Nopper)
  - Added overlay enumeration for a display.
- Revision 3, 2015-03-17 (Norbert Nopper)
  - Fixed typos and namings as discussed in Bugzilla.
  - Reordered and grouped functions.
  - Added functions to query count of display, mode and overlay.
  - Added native display handle, which may be needed on some platforms to create a native Window.
- Revision 4, 2015-03-18 (Norbert Nopper)
  - Removed primary and virtualPostion members (see comment of James Jones in Bugzilla).
  - Added native overlay handle to information structure.
  - Replaced , with ; in struct.
- Revision 6, 2015-03-18 (Daniel Rakos)
  - Added WSI extension suffix to all items.

  - Made the whole API more “Vulkanish”.
  - Replaced all functions with a single vkGetDisplayInfoKHR function to better match the rest of the API.
  - Made the display, display mode, and overlay objects be first class objects, not subclasses of VkBaseObject as they do not support the common functions anyways.
  - Renamed *Info structures to *Properties.
  - Removed overlayIndex field from VkOverlayProperties as there is an implicit index already as a result of moving to a “Vulkanish” API.
  - Displays are not get through device, but through physical GPU to match the rest of the Vulkan API. Also this is something ISVs explicitly requested.
  - Added issue (6) and (7).
- Revision 7, 2015-03-25 (James Jones)
  - Added an issues section
  - Added rotation and mirroring flags
- Revision 8, 2015-03-25 (James Jones)
  - Combined the duplicate issues sections introduced in last change.
  - Added proposed resolutions to several issues.
- Revision 9, 2015-04-01 (Daniel Rakos)
  - Rebased extension against Vulkan 0.82.0
- Revision 10, 2015-04-01 (James Jones)
  - Added issues (10) and (11).
  - Added more straw-man issue resolutions, and cleaned up the proposed resolution for issue (4).
  - Updated the rotation and mirroring enums to have proper bitmask semantics.
- Revision 11, 2015-04-15 (James Jones)
  - Added proposed resolution for issues (1) and (2).
  - Added issues (12), (13), (14), and (15)
  - Removed pNativeHandle field from overlay structure.
  - Fixed small compilation errors in example code.
- Revision 12, 2015-07-29 (James Jones)
  - Rewrote the guts of the extension against the latest WSI swapchain specifications and the latest Vulkan API.
  - Address overlay planes by their index rather than an object handle and refer to them as “planes” rather than “overlays” to make it slightly clearer that even a display with no “overlays” still has at least one base “plane” that images can be displayed on.
  - Updated most of the issues.
  - Added an “extension type” section to the specification header.

    - Reused the VK_EXT_KHR_surface surface transform enumerations rather than redefining them here.
    - Updated the example code to use the new semantics.
- Revision 13, 2015-08-21 (Ian Elliott)
    - Renamed this extension and all of its enumerations, types, functions, etc. This makes it compliant with the proposed standard for Vulkan extensions.
    - Switched from “revision” to “version”, including use of the VK_MAKE_VERSION macro in the header file.
- Revision 14, 2015-09-01 (James Jones)
    - Restore single-field revision number.
- Revision 15, 2015-09-08 (James Jones)
    - Added alpha flags enum.
    - Added premultiplied alpha support.
- Revision 16, 2015-09-08 (James Jones)
    - Added description section to the spec.
    - Added issues 16 - 18.
- Revision 17, 2015-10-02 (James Jones)
    - Planes are now a property of the entire device rather than individual displays. This allows planes to be moved between multiple displays on devices that support it.
    - Added a function to create a VkSurfaceKHR object describing a display plane and mode to align with the new per-platform surface creation conventions.
    - Removed detailed mode timing data. It was agreed that the mode extents and refresh rate are sufficient for current use cases. Other information could be added back in as an extension if it is needed in the future.
    - Added support for smart/persistent/buffered display devices.
- Revision 18, 2015-10-26 (Ian Elliott)
    - Renamed from VK_EXT_KHR_display to VK_KHR_display.
- Revision 19, 2015-11-02 (James Jones)
    - Updated example code to match revision 17 changes.
- Revision 20, 2015-11-03 (Daniel Rakos)
    - Added allocation callbacks to creation functions.
- Revision 21, 2015-11-10 (Jesse Hall)
    - Added VK_DISPLAY_PLANE_ALPHA_OPAQUE_BIT_KHR, and use VkDisplayPlaneAlphaFlagBitsKHR for VkDisplayPlanePropertiesKHR::alphaMode instead of VkDisplayPlaneAlphaFlagsKHR, since it only represents one mode.
    - Added reserved flags bitmask to VkDisplayPlanePropertiesKHR.
    - Use VkSurfaceTransformFlagBitsKHR instead of obsolete VkSurfaceTransformKHR.

    - Renamed vkGetDisplayPlaneSupportedDisplaysKHR parameters for clarity.
- Revision 22, 2015-12-18 (James Jones)
    - Added missing “planeIndex” parameter to vkGetDisplayPlaneSupportedDisplaysKHR()
- Revision 23, 2017-03-13 (James Jones)
    - Closed all remaining issues. The specification and implementations have been shipping with the proposed resolutions for some time now.
    - Removed the sample code and noted it has been integrated into the official Vulkan SDK cube demo.

## VK_KHR_display_swapchain

### Name String

`VK_KHR_display_swapchain`

### Extension Type

Device extension

### Registered Extension Number

4

### Revision

10

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_swapchain
and
VK_KHR_display

### Contact

- James Jones [GitHub]cubanismo

### Other Extension Metadata

### Last Modified Date

2017-03-13

### IP Status

No known IP claims.

### Contributors

- James Jones, NVIDIA
- Jeff Vigil, Qualcomm

- Jesse Hall, Google

**Description**

This extension provides an API to create a swapchain directly on a device's display without any underlying window system.

**New Commands**

- vkCreateSharedSwapchainsKHR

**New Structures**

- Extending VkPresentInfoKHR:
  - VkDisplayPresentInfoKHR

**New Enum Constants**

- `VK_KHR_DISPLAY_SWAPCHAIN_EXTENSION_NAME`
- `VK_KHR_DISPLAY_SWAPCHAIN_SPEC_VERSION`
- Extending VkResult:
  - `VK_ERROR_INCOMPATIBLE_DISPLAY_KHR`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_DISPLAY_PRESENT_INFO_KHR`

**Issues**

1) Should swapchains sharing images each hold a reference to the images, or should it be up to the application to destroy the swapchains and images in an order that avoids the need for reference counting?

**RESOLVED**: Take a reference. The lifetime of presentable images is already complex enough.

2) Should the `srcRect` and `dstRect` parameters be specified as part of the presentation command, or at swapchain creation time?

**RESOLVED**: As part of the presentation command. This allows moving and scaling the image on the screen without the need to respecify the mode or create a new swapchain and presentable images.

3) Should `srcRect` and `dstRect` be specified as rects, or separate offset/extent values?

**RESOLVED**: As rects. Specifying them separately might make it easier for hardware to expose support for one but not the other, but in such cases applications must just take care to obey the reported capabilities and not use non-zero offsets or extents that require scaling, as appropriate.

4) How can applications create multiple swapchains that use the same images?

**RESOLVED**: By calling vkCreateSharedSwapchainsKHR.

An earlier resolution used vkCreateSwapchainKHR, chaining multiple VkSwapchainCreateInfoKHR structures through `pNext`. In order to allow each swapchain to also allow other extension structs, a level of indirection was used: VkSwapchainCreateInfoKHR::`pNext` pointed to a different structure, which had both `sType` and `pNext` members for additional extensions, and also had a pointer to the next VkSwapchainCreateInfoKHR structure. The number of swapchains to be created could only be found by walking this linked list of alternating structures, and the `pSwapchains` out parameter was reinterpreted to be an array of VkSwapchainKHR handles.

Another option considered was a method to specify a "shared" swapchain when creating a new swapchain, such that groups of swapchains using the same images could be built up one at a time. This was deemed unusable because drivers need to know all of the displays an image will be used on when determining which internal formats and layouts to use for that image.

### Examples

| | |
|---|---|
| **NOTE** | The example code for the `VK_KHR_display` and `VK_KHR_display_swapchain` extensions was removed from the appendix after revision 1.0.43. The display swapchain creation example code was ported to the cube demo that is shipped with the official Khronos SDK, and is being kept up-to-date in that location (see: https://github.com/KhronosGroup/Vulkan-Tools/blob/main/cube/cube.c). |

### Version History

- Revision 1, 2015-07-29 (James Jones)
  - Initial draft
- Revision 2, 2015-08-21 (Ian Elliott)
  - Renamed this extension and all of its enumerations, types, functions, etc. This makes it compliant with the proposed standard for Vulkan extensions.
  - Switched from "revision" to "version", including use of the VK_MAKE_VERSION macro in the header file.
- Revision 3, 2015-09-01 (James Jones)
  - Restore single-field revision number.
- Revision 4, 2015-09-08 (James Jones)
  - Allow creating multiple swapchains that share the same images using a single call to vkCreateSwapchainKHR().
- Revision 5, 2015-09-10 (Alon Or-bach)
  - Removed underscores from SWAP_CHAIN in two enums.
- Revision 6, 2015-10-02 (James Jones)
  - Added support for smart panels/buffered displays.
- Revision 7, 2015-10-26 (Ian Elliott)
  - Renamed from VK_EXT_KHR_display_swapchain to VK_KHR_display_swapchain.
- Revision 8, 2015-11-03 (Daniel Rakos)

    - Updated sample code based on the changes to VK_KHR_swapchain.
- Revision 9, 2015-11-10 (Jesse Hall)
    - Replaced VkDisplaySwapchainCreateInfoKHR with vkCreateSharedSwapchainsKHR, changing resolution of issue #4.
- Revision 10, 2017-03-13 (James Jones)
    - Closed all remaining issues. The specification and implementations have been shipping with the proposed resolutions for some time now.
    - Removed the sample code and noted it has been integrated into the official Vulkan SDK cube demo.

## VK_KHR_draw_indirect_count

### Name String

`VK_KHR_draw_indirect_count`

### Extension Type

Device extension

### Registered Extension Number

170

### Revision

1

### Ratification Status

Ratified

### Extension and Version Dependencies

None

### Deprecation State

- *Promoted* to Vulkan 1.2

### Contact

- Piers Daniell [GitHub]pdaniell-nv

### Other Extension Metadata

### Last Modified Date

2017-08-25

### IP Status

No known IP claims.

### Contributors

- Matthaeus G. Chajdas, AMD

- Derrick Owens, AMD
- Graham Sellers, AMD
- Daniel Rakos, AMD
- Dominik Witczak, AMD
- Piers Daniell, NVIDIA

#### Description

This extension is based on the `VK_AMD_draw_indirect_count` extension. This extension allows an application to source the number of draws for indirect drawing calls from a buffer.

Applications might want to do culling on the GPU via a compute shader prior to drawing. This enables the application to generate an arbitrary number of drawing commands and execute them without host intervention.

#### Promotion to Vulkan 1.2

All functionality in this extension is included in core Vulkan 1.2, with the KHR suffix omitted. However, if Vulkan 1.2 is supported and this extension is not, the commands vkCmdDrawIndirectCountKHR and vkCmdDrawIndexedIndirectCountKHR are optional. The original type, enum and command names are still available as aliases of the core functionality.

#### New Commands

- vkCmdDrawIndexedIndirectCountKHR
- vkCmdDrawIndirectCountKHR

#### New Enum Constants

- `VK_KHR_DRAW_INDIRECT_COUNT_EXTENSION_NAME`
- `VK_KHR_DRAW_INDIRECT_COUNT_SPEC_VERSION`

#### Version History

- Revision 1, 2017-08-25 (Piers Daniell)
  - Initial draft based on VK_AMD_draw_indirect_count

## VK_KHR_driver_properties

#### Name String

`VK_KHR_driver_properties`

#### Extension Type

Device extension

#### Registered Extension Number

197

**Revision**

1

**Ratification Status**

Ratified

**Extension and Version Dependencies**

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

**Deprecation State**

- *Promoted* to Vulkan 1.2

**Contact**

- Daniel Rakos [GitHub]drakos-amd

**Other Extension Metadata**

**Last Modified Date**

2018-04-11

**IP Status**

No known IP claims.

**Contributors**

- Baldur Karlsson
- Matthaeus G. Chajdas, AMD
- Piers Daniell, NVIDIA
- Alexander Galazin, Arm
- Jesse Hall, Google
- Daniel Rakos, AMD

**Description**

This extension provides a new physical device query which allows retrieving information about the driver implementation, allowing applications to determine which physical device corresponds to which particular vendor's driver, and which conformance test suite version the driver implementation is compliant with.

**Promotion to Vulkan 1.2**

All functionality in this extension is included in core Vulkan 1.2, with the KHR suffix omitted. The original type, enum and command names are still available as aliases of the core functionality.

### New Structures

- VkConformanceVersionKHR
- Extending VkPhysicalDeviceProperties2:
  - VkPhysicalDeviceDriverPropertiesKHR

### New Enums

- VkDriverIdKHR

### New Enum Constants

- `VK_KHR_DRIVER_PROPERTIES_EXTENSION_NAME`
- `VK_KHR_DRIVER_PROPERTIES_SPEC_VERSION`
- `VK_MAX_DRIVER_INFO_SIZE_KHR`
- `VK_MAX_DRIVER_NAME_SIZE_KHR`
- Extending VkDriverId:
  - `VK_DRIVER_ID_AMD_OPEN_SOURCE_KHR`
  - `VK_DRIVER_ID_AMD_PROPRIETARY_KHR`
  - `VK_DRIVER_ID_ARM_PROPRIETARY_KHR`
  - `VK_DRIVER_ID_BROADCOM_PROPRIETARY_KHR`
  - `VK_DRIVER_ID_GGP_PROPRIETARY_KHR`
  - `VK_DRIVER_ID_GOOGLE_SWIFTSHADER_KHR`
  - `VK_DRIVER_ID_IMAGINATION_PROPRIETARY_KHR`
  - `VK_DRIVER_ID_INTEL_OPEN_SOURCE_MESA_KHR`
  - `VK_DRIVER_ID_INTEL_PROPRIETARY_WINDOWS_KHR`
  - `VK_DRIVER_ID_MESA_RADV_KHR`
  - `VK_DRIVER_ID_NVIDIA_PROPRIETARY_KHR`
  - `VK_DRIVER_ID_QUALCOMM_PROPRIETARY_KHR`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DRIVER_PROPERTIES_KHR`

### Version History

- Revision 1, 2018-04-11 (Daniel Rakos)
  - Internal revisions

## VK_KHR_dynamic_rendering

### Name String

`VK_KHR_dynamic_rendering`

**Extension Type**

Device extension

**Registered Extension Number**

45

**Revision**

1

**Ratification Status**

Ratified

**Extension and Version Dependencies**

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1
and
VK_KHR_depth_stencil_resolve
or
Vulkan Version 1.2

**API Interactions**

- Interacts with VK_AMD_mixed_attachment_samples
- Interacts with VK_EXT_fragment_density_map
- Interacts with VK_KHR_fragment_shading_rate
- Interacts with VK_NVX_multiview_per_view_attributes
- Interacts with VK_NV_framebuffer_mixed_samples

**Deprecation State**

- *Promoted* to Vulkan 1.3

**Contact**

- Tobias Hector [GitHub]tobski

**Extension Proposal**

VK_KHR_dynamic_rendering

**Other Extension Metadata**

**Last Modified Date**

2021-10-06

**Contributors**

- Tobias Hector, AMD
- Arseny Kapoulkine, Roblox
- François Duranleau, Gameloft

- Stuart Smith, AMD
- Hai Nguyen, Google
- Jean-François Roy, Google
- Jeff Leger, Qualcomm
- Jan-Harald Fredriksen, Arm
- Piers Daniell, Nvidia
- James Fitzpatrick, Imagination
- Piotr Byszewski, Mobica
- Jesse Hall, Google
- Mike Blumenkrantz, Valve

**Description**

This extension allows applications to create single-pass render pass instances without needing to create render pass objects or framebuffers. Dynamic render passes can also span across multiple primary command buffers, rather than relying on secondary command buffers.

This extension also incorporates `VK_ATTACHMENT_STORE_OP_NONE_KHR` from `VK_QCOM_render_pass_store_ops`, enabling applications to avoid unnecessary synchronization when an attachment is not written during a render pass.

**New Commands**

- vkCmdBeginRenderingKHR
- vkCmdEndRenderingKHR

**New Structures**

- VkRenderingAttachmentInfoKHR
- VkRenderingInfoKHR
- Extending VkCommandBufferInheritanceInfo:
  - VkCommandBufferInheritanceRenderingInfoKHR
- Extending VkGraphicsPipelineCreateInfo:
  - VkPipelineRenderingCreateInfoKHR
- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceDynamicRenderingFeaturesKHR

If VK_AMD_mixed_attachment_samples is supported:

- Extending VkCommandBufferInheritanceInfo, VkGraphicsPipelineCreateInfo:
  - VkAttachmentSampleCountInfoAMD

If VK_EXT_fragment_density_map is supported:

- Extending VkRenderingInfo:
  - VkRenderingFragmentDensityMapAttachmentInfoEXT

If VK_KHR_fragment_shading_rate is supported:

- Extending VkRenderingInfo:
  - VkRenderingFragmentShadingRateAttachmentInfoKHR

If VK_NV_framebuffer_mixed_samples is supported:

- Extending VkCommandBufferInheritanceInfo, VkGraphicsPipelineCreateInfo:
  - VkAttachmentSampleCountInfoNV

If VK_NVX_multiview_per_view_attributes is supported:

- Extending VkCommandBufferInheritanceInfo, VkGraphicsPipelineCreateInfo, VkRenderingInfo:
  - VkMultiviewPerViewAttributesInfoNVX

**New Enums**

- VkRenderingFlagBitsKHR

**New Bitmasks**

- VkRenderingFlagsKHR

**New Enum Constants**

- `VK_KHR_DYNAMIC_RENDERING_EXTENSION_NAME`
- `VK_KHR_DYNAMIC_RENDERING_SPEC_VERSION`
- Extending VkAttachmentStoreOp:
  - `VK_ATTACHMENT_STORE_OP_NONE_KHR`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_COMMAND_BUFFER_INHERITANCE_RENDERING_INFO_KHR`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DYNAMIC_RENDERING_FEATURES_KHR`
  - `VK_STRUCTURE_TYPE_PIPELINE_RENDERING_CREATE_INFO_KHR`
  - `VK_STRUCTURE_TYPE_RENDERING_ATTACHMENT_INFO_KHR`
  - `VK_STRUCTURE_TYPE_RENDERING_INFO_KHR`

If VK_AMD_mixed_attachment_samples is supported:

- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_ATTACHMENT_SAMPLE_COUNT_INFO_AMD`

If VK_EXT_fragment_density_map is supported:

- Extending VkPipelineCreateFlagBits:
  - VK_PIPELINE_CREATE_RENDERING_FRAGMENT_DENSITY_MAP_ATTACHMENT_BIT_EXT
  - VK_PIPELINE_RASTERIZATION_STATE_CREATE_FRAGMENT_DENSITY_MAP_ATTACHMENT_BIT_EXT
- Extending VkStructureType:
  - VK_STRUCTURE_TYPE_RENDERING_FRAGMENT_DENSITY_MAP_ATTACHMENT_INFO_EXT

If VK_KHR_fragment_shading_rate is supported:

- Extending VkPipelineCreateFlagBits:
  - VK_PIPELINE_CREATE_RENDERING_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR
  - VK_PIPELINE_RASTERIZATION_STATE_CREATE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR
- Extending VkStructureType:
  - VK_STRUCTURE_TYPE_RENDERING_FRAGMENT_SHADING_RATE_ATTACHMENT_INFO_KHR

If VK_NV_framebuffer_mixed_samples is supported:

- Extending VkStructureType:
  - VK_STRUCTURE_TYPE_ATTACHMENT_SAMPLE_COUNT_INFO_NV

If VK_NVX_multiview_per_view_attributes is supported:

- Extending VkStructureType:
  - VK_STRUCTURE_TYPE_MULTIVIEW_PER_VIEW_ATTRIBUTES_INFO_NVX

#### Promotion to Vulkan 1.3

Functionality in this extension is included in core Vulkan 1.3, with the KHR suffix omitted. The original type, enum and command names are still available as aliases of the core functionality.

#### Version History

- Revision 1, 2021-10-06 (Tobias Hector)
  - Initial revision

## VK_KHR_dynamic_rendering_local_read

#### Name String

VK_KHR_dynamic_rendering_local_read

#### Extension Type

Device extension

#### Registered Extension Number

233

**Revision**

1

**Ratification Status**

Ratified

**Extension and Version Dependencies**

VK_KHR_dynamic_rendering
or
Vulkan Version 1.3

**Contact**

- Tobias Hector [GitHub]tobski

**Extension Proposal**

VK_KHR_dynamic_rendering_local_read

**Other Extension Metadata**

**Last Modified Date**

2023-11-03

**Contributors**

- Tobias Hector, AMD
- Hans-Kristian Arntzen, Valve
- Connor Abbott, Valve
- Pan Gao, Huawei
- Lionel Landwerlin, Intel
- Shahbaz Youssefi, Google
- Alyssa Rosenzweig, Valve
- Jan-Harald Fredriksen, Arm
- Mike Blumenkrantz, Valve
- Graeme Leese, Broadcom
- Piers Daniell, Nvidia
- Stuart Smith, AMD
- Daniel Story, Nintendo
- James Fitzpatrick, Imagination
- Piotr Byszewski, Mobica
- Spencer Fricke, LunarG
- Tom Olson, Arm
- Michal Pietrasiuk, Intel

- Matthew Netsch, Qualcomm
- Marty Johnson, Khronos
- Wyvern Wang, Huawei
- Jeff Bolz, Nvidia
- Samuel (Sheng-Wen) Huang, MediaTek

#### Description

This extension enables reads from attachments and resources written by previous fragment shaders within a dynamic render pass.

#### New Commands

- vkCmdSetRenderingAttachmentLocationsKHR
- vkCmdSetRenderingInputAttachmentIndicesKHR

#### New Structures

- Extending VkGraphicsPipelineCreateInfo, VkCommandBufferInheritanceInfo:
  - VkRenderingAttachmentLocationInfoKHR
  - VkRenderingInputAttachmentIndexInfoKHR
- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceDynamicRenderingLocalReadFeaturesKHR

#### New Enum Constants

- `VK_KHR_DYNAMIC_RENDERING_LOCAL_READ_EXTENSION_NAME`
- `VK_KHR_DYNAMIC_RENDERING_LOCAL_READ_SPEC_VERSION`
- Extending VkImageLayout:
  - `VK_IMAGE_LAYOUT_RENDERING_LOCAL_READ_KHR`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DYNAMIC_RENDERING_LOCAL_READ_FEATURES_KHR`
  - `VK_STRUCTURE_TYPE_RENDERING_ATTACHMENT_LOCATION_INFO_KHR`
  - `VK_STRUCTURE_TYPE_RENDERING_INPUT_ATTACHMENT_INDEX_INFO_KHR`

#### Version History

- Revision 1, 2023-11-03 (Tobias Hector)
  - Initial revision

## VK_KHR_external_fence_fd

### Name String

`VK_KHR_external_fence_fd`

### Extension Type

Device extension

### Registered Extension Number

116

### Revision

1

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_external_fence
or
Vulkan Version 1.1

### Contact

- Jesse Hall [GitHub]critsec

### Other Extension Metadata

### Last Modified Date

2017-05-08

### IP Status

No known IP claims.

### Contributors

- Jesse Hall, Google
- James Jones, NVIDIA
- Jeff Juliano, NVIDIA
- Cass Everitt, Oculus
- Contributors to `VK_KHR_external_semaphore_fd`

### Description

An application using external memory may wish to synchronize access to that memory using fences. This extension enables an application to export fence payload to and import fence payload from POSIX file descriptors.

### New Commands

- vkGetFenceFdKHR

- vkImportFenceFdKHR

#### New Structures

- VkFenceGetFdInfoKHR
- VkImportFenceFdInfoKHR

#### New Enum Constants

- `VK_KHR_EXTERNAL_FENCE_FD_EXTENSION_NAME`
- `VK_KHR_EXTERNAL_FENCE_FD_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_FENCE_GET_FD_INFO_KHR`
  - `VK_STRUCTURE_TYPE_IMPORT_FENCE_FD_INFO_KHR`

#### Issues

This extension borrows concepts, semantics, and language from `VK_KHR_external_semaphore_fd`. That extension's issues apply equally to this extension.

#### Version History

- Revision 1, 2017-05-08 (Jesse Hall)
  - Initial revision

### VK_KHR_external_fence_win32

#### Name String

`VK_KHR_external_fence_win32`

#### Extension Type

Device extension

#### Registered Extension Number

115

#### Revision

1

#### Ratification Status

Ratified

#### Extension and Version Dependencies

VK_KHR_external_fence

#### Contact

- Jesse Hall [GitHub]critsec

### Other Extension Metadata

#### Last Modified Date

2017-05-08

#### IP Status

No known IP claims.

#### Contributors

- Jesse Hall, Google
- James Jones, NVIDIA
- Jeff Juliano, NVIDIA
- Cass Everitt, Oculus
- Contributors to `VK_KHR_external_semaphore_win32`

#### Description

An application using external memory may wish to synchronize access to that memory using fences. This extension enables an application to export fence payload to and import fence payload from Windows handles.

#### New Commands

- vkGetFenceWin32HandleKHR
- vkImportFenceWin32HandleKHR

#### New Structures

- VkFenceGetWin32HandleInfoKHR
- VkImportFenceWin32HandleInfoKHR
- Extending VkFenceCreateInfo:
  - VkExportFenceWin32HandleInfoKHR

#### New Enum Constants

- `VK_KHR_EXTERNAL_FENCE_WIN32_EXTENSION_NAME`
- `VK_KHR_EXTERNAL_FENCE_WIN32_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_EXPORT_FENCE_WIN32_HANDLE_INFO_KHR`
  - `VK_STRUCTURE_TYPE_FENCE_GET_WIN32_HANDLE_INFO_KHR`
  - `VK_STRUCTURE_TYPE_IMPORT_FENCE_WIN32_HANDLE_INFO_KHR`

### Issues

This extension borrows concepts, semantics, and language from `VK_KHR_external_semaphore_win32`. That extension's issues apply equally to this extension.

1) Should D3D12 fence handle types be supported, like they are for semaphores?

**RESOLVED**: No. Doing so would require extending the fence signal and wait operations to provide values to signal / wait for, like `VkD3D12FenceSubmitInfoKHR` does. A D3D12 fence can be signaled by importing it into a VkSemaphore instead of a VkFence, and applications can check status or wait on the D3D12 fence using non-Vulkan APIs. The convenience of being able to do these operations on `VkFence` objects does not justify the extra API complexity.

### Version History

- Revision 1, 2017-05-08 (Jesse Hall)
  - Initial revision

## VK_KHR_external_memory_fd

### Name String

`VK_KHR_external_memory_fd`

### Extension Type

Device extension

### Registered Extension Number

75

### Revision

1

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_external_memory
or
Vulkan Version 1.1

### Contact

- James Jones [GitHub]cubanismo

### Other Extension Metadata

### Last Modified Date

2016-10-21

### IP Status

No known IP claims.

### Contributors

- James Jones, NVIDIA
- Jeff Juliano, NVIDIA

### Description

An application may wish to reference device memory in multiple Vulkan logical devices or instances, in multiple processes, and/or in multiple APIs. This extension enables an application to export POSIX file descriptor handles from Vulkan memory objects and to import Vulkan memory objects from POSIX file descriptor handles exported from other Vulkan memory objects or from similar resources in other APIs.

### New Commands

- vkGetMemoryFdKHR
- vkGetMemoryFdPropertiesKHR

### New Structures

- VkMemoryFdPropertiesKHR
- VkMemoryGetFdInfoKHR
- Extending VkMemoryAllocateInfo:
  - VkImportMemoryFdInfoKHR

### New Enum Constants

- `VK_KHR_EXTERNAL_MEMORY_FD_EXTENSION_NAME`
- `VK_KHR_EXTERNAL_MEMORY_FD_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_IMPORT_MEMORY_FD_INFO_KHR`
  - `VK_STRUCTURE_TYPE_MEMORY_FD_PROPERTIES_KHR`
  - `VK_STRUCTURE_TYPE_MEMORY_GET_FD_INFO_KHR`

### Issues

1) Does the application need to close the file descriptor returned by vkGetMemoryFdKHR?

**RESOLVED**: Yes, unless it is passed back in to a driver instance to import the memory. A successful get call transfers ownership of the file descriptor to the application, and a successful import transfers it back to the driver. Destroying the original memory object will not close the file descriptor or remove its reference to the underlying memory resource associated with it.

2) Do drivers ever need to expose multiple file descriptors per memory object?

**RESOLVED**: No. This would indicate there are actually multiple memory objects, rather than a single memory object.

3) How should the valid size and memory type for POSIX file descriptor memory handles created outside of Vulkan be specified?

**RESOLVED**: The valid memory types are queried directly from the external handle. The size will be specified by future extensions that introduce such external memory handle types.

### Version History

- Revision 1, 2016-10-21 (James Jones)
  - Initial revision

## VK_KHR_external_memory_win32

### Name String

`VK_KHR_external_memory_win32`

### Extension Type

Device extension

### Registered Extension Number

74

### Revision

1

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_external_memory
or
Vulkan Version 1.1

### Contact

- James Jones [GitHub]cubanismo

### Other Extension Metadata

### Last Modified Date

2016-10-21

### IP Status

No known IP claims.

### Contributors

- James Jones, NVIDIA
- Jeff Juliano, NVIDIA
- Carsten Rohde, NVIDIA

### Description

An application may wish to reference device memory in multiple Vulkan logical devices or instances, in multiple processes, and/or in multiple APIs. This extension enables an application to export Windows handles from Vulkan memory objects and to import Vulkan memory objects from Windows handles exported from other Vulkan memory objects or from similar resources in other APIs.

### New Commands

- vkGetMemoryWin32HandleKHR
- vkGetMemoryWin32HandlePropertiesKHR

### New Structures

- VkMemoryGetWin32HandleInfoKHR
- VkMemoryWin32HandlePropertiesKHR
- Extending VkMemoryAllocateInfo:
  - VkExportMemoryWin32HandleInfoKHR
  - VkImportMemoryWin32HandleInfoKHR

### New Enum Constants

- `VK_KHR_EXTERNAL_MEMORY_WIN32_EXTENSION_NAME`
- `VK_KHR_EXTERNAL_MEMORY_WIN32_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_EXPORT_MEMORY_WIN32_HANDLE_INFO_KHR`
  - `VK_STRUCTURE_TYPE_IMPORT_MEMORY_WIN32_HANDLE_INFO_KHR`
  - `VK_STRUCTURE_TYPE_MEMORY_GET_WIN32_HANDLE_INFO_KHR`
  - `VK_STRUCTURE_TYPE_MEMORY_WIN32_HANDLE_PROPERTIES_KHR`

### Issues

1) Do applications need to call `CloseHandle`() on the values returned from vkGetMemoryWin32HandleKHR when `handleType` is `VK_EXTERNAL_MEMORY_HANDLE_TYPE_OPAQUE_WIN32_BIT_KHR`?

**RESOLVED**: Yes. A successful get call transfers ownership of the handle to the application. Destroying the memory object will not destroy the handle or the handle's reference to the

underlying memory resource. Unlike file descriptor opaque handles, win32 opaque handle ownership can not be transferred back to a driver by an import operation.

2) Should the language regarding KMT/Windows 7 handles be moved to a separate extension so that it can be deprecated over time?

**RESOLVED**: No. Support for them can be deprecated by drivers if they choose, by no longer returning them in the supported handle types of the instance level queries.

3) How should the valid size and memory type for windows memory handles created outside of Vulkan be specified?

**RESOLVED**: The valid memory types are queried directly from the external handle. The size is determined by the associated image or buffer memory requirements for external handle types that require dedicated allocations, and by the size specified when creating the object from which the handle was exported for other external handle types.

### Version History

- Revision 1, 2016-10-21 (James Jones)
  - Initial revision

## VK_KHR_external_semaphore_fd

### Name String

`VK_KHR_external_semaphore_fd`

### Extension Type

Device extension

### Registered Extension Number

80

### Revision

1

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_external_semaphore
or
Vulkan Version 1.1

### Contact

- James Jones [GitHub]cubanismo

### Other Extension Metadata

#### Last Modified Date

2016-10-21

#### IP Status

No known IP claims.

#### Contributors

- Jesse Hall, Google
- James Jones, NVIDIA
- Jeff Juliano, NVIDIA
- Carsten Rohde, NVIDIA

### Description

An application using external memory may wish to synchronize access to that memory using semaphores. This extension enables an application to export semaphore payload to and import semaphore payload from POSIX file descriptors.

### New Commands

- vkGetSemaphoreFdKHR
- vkImportSemaphoreFdKHR

### New Structures

- VkImportSemaphoreFdInfoKHR
- VkSemaphoreGetFdInfoKHR

### New Enum Constants

- `VK_KHR_EXTERNAL_SEMAPHORE_FD_EXTENSION_NAME`
- `VK_KHR_EXTERNAL_SEMAPHORE_FD_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_IMPORT_SEMAPHORE_FD_INFO_KHR`
  - `VK_STRUCTURE_TYPE_SEMAPHORE_GET_FD_INFO_KHR`

### Issues

1) Does the application need to close the file descriptor returned by vkGetSemaphoreFdKHR?

**RESOLVED**: Yes, unless it is passed back in to a driver instance to import the semaphore. A successful get call transfers ownership of the file descriptor to the application, and a successful import transfers it back to the driver. Destroying the original semaphore object will not close the file descriptor or remove its reference to the underlying semaphore resource associated with it.

### Version History

- Revision 1, 2016-10-21 (Jesse Hall)
  - Initial revision

## VK_KHR_external_semaphore_win32

### Name String

`VK_KHR_external_semaphore_win32`

### Extension Type

Device extension

### Registered Extension Number

79

### Revision

1

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_external_semaphore

### Contact

- James Jones [GitHub]cubanismo

### Other Extension Metadata

### Last Modified Date

2016-10-21

### IP Status

No known IP claims.

### Contributors

- James Jones, NVIDIA
- Jeff Juliano, NVIDIA
- Carsten Rohde, NVIDIA

### Description

An application using external memory may wish to synchronize access to that memory using semaphores. This extension enables an application to export semaphore payload to and import semaphore payload from Windows handles.

### New Commands

- vkGetSemaphoreWin32HandleKHR
- vkImportSemaphoreWin32HandleKHR

### New Structures

- VkImportSemaphoreWin32HandleInfoKHR
- VkSemaphoreGetWin32HandleInfoKHR
- Extending VkSemaphoreCreateInfo:
  - VkExportSemaphoreWin32HandleInfoKHR
- Extending VkSubmitInfo:
  - VkD3D12FenceSubmitInfoKHR

### New Enum Constants

- `VK_KHR_EXTERNAL_SEMAPHORE_WIN32_EXTENSION_NAME`
- `VK_KHR_EXTERNAL_SEMAPHORE_WIN32_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_D3D12_FENCE_SUBMIT_INFO_KHR`
  - `VK_STRUCTURE_TYPE_EXPORT_SEMAPHORE_WIN32_HANDLE_INFO_KHR`
  - `VK_STRUCTURE_TYPE_IMPORT_SEMAPHORE_WIN32_HANDLE_INFO_KHR`
  - `VK_STRUCTURE_TYPE_SEMAPHORE_GET_WIN32_HANDLE_INFO_KHR`

### Issues

1) Do applications need to call `CloseHandle`() on the values returned from vkGetSemaphoreWin32HandleKHR when `handleType` is `VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_OPAQUE_WIN32_BIT_KHR`?

**RESOLVED**: Yes. A successful get call transfers ownership of the handle to the application. Destroying the semaphore object will not destroy the handle or the handle's reference to the underlying semaphore resource. Unlike file descriptor opaque handles, win32 opaque handle ownership can not be transferred back to a driver by an import operation.

2) Should the language regarding KMT/Windows 7 handles be moved to a separate extension so that it can be deprecated over time?

**RESOLVED**: No. Support for them can be deprecated by drivers if they choose, by no longer returning them in the supported handle types of the instance level queries.

3) Should applications be allowed to specify additional object attributes for shared handles?

**RESOLVED**: Yes. Applications will be allowed to provide similar attributes to those they would to any other handle creation API.

4) How do applications communicate the desired fence values to use with `D3D12_FENCE`-based Vulkan semaphores?

**RESOLVED**: There are a couple of options. The values for the signaled and reset states could be communicated up front when creating the object and remain static for the life of the Vulkan semaphore, or they could be specified using auxiliary structures when submitting semaphore signal and wait operations, similar to what is done with the keyed mutex extensions. The latter is more flexible and consistent with the keyed mutex usage, but the former is a much simpler API.

Since Vulkan tends to favor flexibility and consistency over simplicity, a new structure specifying D3D12 fence acquire and release values is added to the vkQueueSubmit function.

### Version History

- Revision 1, 2016-10-21 (James Jones)
  - Initial revision

## VK_KHR_format_feature_flags2

### Name String

`VK_KHR_format_feature_flags2`

### Extension Type

Device extension

### Registered Extension Number

361

### Revision

2

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### Deprecation State

- *Promoted* to Vulkan 1.3

### Contact

- Lionel Landwerlin [GitHub]llandwerlin

### Other Extension Metadata

### Last Modified Date

2021-07-01

### IP Status

No known IP claims.

### Contributors

- Lionel Landwerlin, Intel
- Faith Ekstrand, Intel
- Tobias Hector, AMD
- Spencer Fricke, Samsung Electronics
- Graeme Leese, Broadcom
- Jan-Harald Fredriksen, ARM

### Description

This extension adds a new VkFormatFeatureFlagBits2KHR 64bits format feature flag type to extend the existing VkFormatFeatureFlagBits which is limited to 31 flags. At the time of this writing 29 bits of VkFormatFeatureFlagBits are already used.

Because VkFormatProperties2 is already defined to extend the Vulkan 1.0 vkGetPhysicalDeviceFormatProperties command, this extension defines a new VkFormatProperties3KHR to extend the VkFormatProperties.

On top of replicating all the bits from VkFormatFeatureFlagBits, VkFormatFeatureFlagBits2KHR adds the following bits :

- `VK_FORMAT_FEATURE_2_STORAGE_READ_WITHOUT_FORMAT_BIT_KHR` and `VK_FORMAT_FEATURE_2_STORAGE_WRITE_WITHOUT_FORMAT_BIT_KHR` indicate that an implementation supports respectively reading and writing a given VkFormat through storage operations without specifying the format in the shader.
- `VK_FORMAT_FEATURE_2_SAMPLED_IMAGE_DEPTH_COMPARISON_BIT_KHR` indicates that an implementation supports depth comparison performed by `OpImage*Dref*` instructions on a given VkFormat. Previously the result of executing a `OpImage*Dref*` instruction on an image view, where the `format` was not one of the depth/stencil formats with a depth component, was undefined. This bit clarifies on which formats such instructions can be used.

Prior to version 2 of this extension, implementations exposing the `shaderStorageImageReadWithoutFormat` and `shaderStorageImageWriteWithoutFormat` features may not report `VK_FORMAT_FEATURE_2_STORAGE_READ_WITHOUT_FORMAT_BIT_KHR` and `VK_FORMAT_FEATURE_2_STORAGE_WRITE_WITHOUT_FORMAT_BIT_KHR` in VkFormatProperties3KHR::`bufferFeatures`. Despite this, buffer reads/writes are supported as intended by the original features.

### New Structures

- Extending VkFormatProperties2:
  - VkFormatProperties3KHR

### New Enums

- VkFormatFeatureFlagBits2KHR

### New Bitmasks

- VkFormatFeatureFlags2KHR

### New Enum Constants

- `VK_KHR_FORMAT_FEATURE_FLAGS_2_EXTENSION_NAME`
- `VK_KHR_FORMAT_FEATURE_FLAGS_2_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_FORMAT_PROPERTIES_3_KHR`

### Promotion to Vulkan 1.3

Functionality in this extension is included in core Vulkan 1.3, with the KHR suffix omitted. The original type, enum and command names are still available as aliases of the core functionality.

### Version History

- Revision 2, 2022-07-20 (Lionel Landwerlin)
  - Clarify that VK_FORMAT_FEATURE_2_STORAGE_(READ|WRITE)_WITHOUT_FORMAT_BIT also apply to buffer views.
- Revision 1, 2020-07-21 (Lionel Landwerlin)
  - Initial draft

## VK_KHR_fragment_shader_barycentric

### Name String

`VK_KHR_fragment_shader_barycentric`

### Extension Type

Device extension

### Registered Extension Number

323

### Revision

1

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2

or
Vulkan Version 1.1

**SPIR-V Dependencies**

- SPV_KHR_fragment_shader_barycentric

**Contact**

- Stu Smith

**Extension Proposal**

VK_KHR_fragment_shader_barycentric

**Other Extension Metadata**

**Last Modified Date**

2022-03-10

**IP Status**

No known IP claims.

**Interactions and External Dependencies**

- This extension provides API support for `GL_EXT_fragment_shader_barycentric`

**Contributors**

- Stu Smith, AMD
- Tobias Hector, AMD
- Graeme Leese, Broadcom
- Jan-Harald Fredriksen, Arm
- Slawek Grajewski, Intel
- Pat Brown, NVIDIA
- Hans-Kristian Arntzen, Valve
- Contributors to the VK_NV_fragment_shader_barycentric specification

**Description**

This extension is based on the `VK_NV_fragment_shader_barycentric` extension, and adds support for the following SPIR-V extension in Vulkan:

- `SPV_KHR_fragment_shader_barycentric`

The extension provides access to three additional fragment shader variable decorations in SPIR-V:

- `PerVertexKHR`, which indicates that a fragment shader input will not have interpolated values, but instead must be accessed with an extra array index that identifies one of the vertices of the primitive producing the fragment
- `BaryCoordKHR`, which indicates that the variable is a three-component floating-point vector holding barycentric weights for the fragment produced using perspective interpolation

- `BaryCoordNoPerspKHR`, which indicates that the variable is a three-component floating-point vector holding barycentric weights for the fragment produced using linear interpolation

When using GLSL source-based shader languages, the following variables from `GL_EXT_fragment_shader_barycentric` map to these SPIR-V built-in decorations:

- `in vec3 gl_BaryCoordEXT;` → `BaryCoordKHR`
- `in vec3 gl_BaryCoordNoPerspEXT;` → `BaryCoordNoPerspKHR`

GLSL variables declared using the `pervertexEXT` GLSL qualifier are expected to be decorated with `PerVertexKHR` in SPIR-V.

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceFragmentShaderBarycentricFeaturesKHR
- Extending VkPhysicalDeviceProperties2:
  - VkPhysicalDeviceFragmentShaderBarycentricPropertiesKHR

### New Enum Constants

- `VK_KHR_FRAGMENT_SHADER_BARYCENTRIC_EXTENSION_NAME`
- `VK_KHR_FRAGMENT_SHADER_BARYCENTRIC_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_FRAGMENT_SHADER_BARYCENTRIC_FEATURES_KHR`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_FRAGMENT_SHADER_BARYCENTRIC_PROPERTIES_KHR`

### New Built-In Variables

- `BaryCoordKHR`
- `BaryCoordNoPerspKHR`

### New SPIR-V Decorations

- `PerVertexKHR`

### New SPIR-V Capabilities

- `FragmentBarycentricKHR`

### Issues

1) What are the interactions with MSAA and how are `BaryCoordKHR` and `BaryCoordNoPerspKHR` interpolated?

**RESOLVED**: The inputs decorated with `BaryCoordKHR` or `BaryCoordNoPerspKHR` **may** also be decorated with the `Centroid` or `Sample` qualifiers to specify interpolation, like any other fragment shader input.

If `shaderSampleRateInterpolationFunctions` is enabled, the extended instructions InterpolateAtCentroid, InterpolateAtOffset, and InterpolateAtSample from the GLSL.std.450 **may** also be used with inputs decorated with `BaryCoordKHR` or `BaryCoordNoPerspKHR`.

### Version History

- Revision 1, 2022-03-10 (Stu Smith)
  - Initial revision

## VK_KHR_fragment_shading_rate

### Name String

`VK_KHR_fragment_shading_rate`

### Extension Type

Device extension

### Registered Extension Number

227

### Revision

2

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1
and
VK_KHR_create_renderpass2
or
Vulkan Version 1.2

### API Interactions

- Interacts with VK_VERSION_1_3
- Interacts with VK_KHR_format_feature_flags2

### SPIR-V Dependencies

- SPV_KHR_fragment_shading_rate

### Contact

- Tobias Hector [GitHub]tobski

### Extension Proposal

VK_KHR_fragment_shading_rate

### Other Extension Metadata

#### Last Modified Date

2021-09-30

#### Interactions and External Dependencies

- This extension provides API support for `GL_EXT_fragment_shading_rate`

#### Contributors

- Tobias Hector, AMD
- Guennadi Riguer, AMD
- Matthaeus Chajdas, AMD
- Pat Brown, Nvidia
- Matthew Netsch, Qualcomm
- Slawomir Grajewski, Intel
- Jan-Harald Fredriksen, Arm
- Jeff Bolz, Nvidia
- Arseny Kapoulkine, Roblox
- Contributors to the VK_NV_shading_rate_image specification
- Contributors to the VK_EXT_fragment_density_map specification

#### Description

This extension adds the ability to change the rate at which fragments are shaded. Rather than the usual single fragment invocation for each pixel covered by a primitive, multiple pixels can be shaded by a single fragment shader invocation.

Up to three methods are available to the application to change the fragment shading rate:

- Pipeline Fragment Shading Rate, which allows the specification of a rate per-draw.
- Primitive Fragment Shading Rate, which allows the specification of a rate per primitive, specified during shading.
- Attachment Fragment Shading Rate, which allows the specification of a rate per-region of the framebuffer, specified in a specialized image attachment.

Additionally, these rates can all be specified and combined in order to adjust the overall detail in the image at each point.

This functionality can be used to focus shading efforts where higher levels of detail are needed in some parts of a scene compared to others. This can be particularly useful in high resolution rendering, or for XR contexts.

This extension also adds support for the `SPV_KHR_fragment_shading_rate` extension which enables setting the primitive fragment shading rate, and allows querying the final shading rate from a fragment shader.

### New Commands

- vkCmdSetFragmentShadingRateKHR
- vkGetPhysicalDeviceFragmentShadingRatesKHR

### New Structures

- VkPhysicalDeviceFragmentShadingRateKHR
- Extending VkGraphicsPipelineCreateInfo:
  - VkPipelineFragmentShadingRateStateCreateInfoKHR
- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceFragmentShadingRateFeaturesKHR
- Extending VkPhysicalDeviceProperties2:
  - VkPhysicalDeviceFragmentShadingRatePropertiesKHR
- Extending VkSubpassDescription2:
  - VkFragmentShadingRateAttachmentInfoKHR

### New Enums

- VkFragmentShadingRateCombinerOpKHR

### New Enum Constants

- `VK_KHR_FRAGMENT_SHADING_RATE_EXTENSION_NAME`
- `VK_KHR_FRAGMENT_SHADING_RATE_SPEC_VERSION`
- Extending VkAccessFlagBits:
  - `VK_ACCESS_FRAGMENT_SHADING_RATE_ATTACHMENT_READ_BIT_KHR`
- Extending VkDynamicState:
  - `VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR`
- Extending VkFormatFeatureFlagBits:
  - `VK_FORMAT_FEATURE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR`
- Extending VkImageLayout:
  - `VK_IMAGE_LAYOUT_FRAGMENT_SHADING_RATE_ATTACHMENT_OPTIMAL_KHR`
- Extending VkImageUsageFlagBits:
  - `VK_IMAGE_USAGE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR`
- Extending VkPipelineStageFlagBits:
  - `VK_PIPELINE_STAGE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_FRAGMENT_SHADING_RATE_ATTACHMENT_INFO_KHR`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_FRAGMENT_SHADING_RATE_FEATURES_KHR`

    - VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_FRAGMENT_SHADING_RATE_KHR
    - VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_FRAGMENT_SHADING_RATE_PROPERTIES_KHR
    - VK_STRUCTURE_TYPE_PIPELINE_FRAGMENT_SHADING_RATE_STATE_CREATE_INFO_KHR

If VK_KHR_format_feature_flags2 or Vulkan Version 1.3 is supported:

- Extending VkFormatFeatureFlagBits2:
    - VK_FORMAT_FEATURE_2_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR

#### Version History

- Revision 1, 2020-05-06 (Tobias Hector)
    - Initial revision
- Revision 2, 2021-09-30 (Jon Leech)
    - Add interaction with VK_KHR_format_feature_flags2 to vk.xml

## VK_KHR_get_display_properties2

#### Name String

VK_KHR_get_display_properties2

#### Extension Type

Instance extension

#### Registered Extension Number

122

#### Revision

1

#### Ratification Status

Ratified

#### Extension and Version Dependencies

VK_KHR_display

#### Contact

- James Jones [GitHub]cubanismo

#### Other Extension Metadata

#### Last Modified Date

2017-02-21

#### IP Status

No known IP claims.

### Contributors

- Ian Elliott, Google
- James Jones, NVIDIA

### Description

This extension provides new queries for device display properties and capabilities that can be easily extended by other extensions, without introducing any further queries. This extension can be considered the `VK_KHR_display` equivalent of the `VK_KHR_get_physical_device_properties2` extension.

### New Commands

- vkGetDisplayModeProperties2KHR
- vkGetDisplayPlaneCapabilities2KHR
- vkGetPhysicalDeviceDisplayPlaneProperties2KHR
- vkGetPhysicalDeviceDisplayProperties2KHR

### New Structures

- VkDisplayModeProperties2KHR
- VkDisplayPlaneCapabilities2KHR
- VkDisplayPlaneInfo2KHR
- VkDisplayPlaneProperties2KHR
- VkDisplayProperties2KHR

### New Enum Constants

- `VK_KHR_GET_DISPLAY_PROPERTIES_2_EXTENSION_NAME`
- `VK_KHR_GET_DISPLAY_PROPERTIES_2_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_DISPLAY_MODE_PROPERTIES_2_KHR`
  - `VK_STRUCTURE_TYPE_DISPLAY_PLANE_CAPABILITIES_2_KHR`
  - `VK_STRUCTURE_TYPE_DISPLAY_PLANE_INFO_2_KHR`
  - `VK_STRUCTURE_TYPE_DISPLAY_PLANE_PROPERTIES_2_KHR`
  - `VK_STRUCTURE_TYPE_DISPLAY_PROPERTIES_2_KHR`

### Issues

1) What should this extension be named?

**RESOLVED**: `VK_KHR_get_display_properties2`. Other alternatives:

- `VK_KHR_display2`

- One extension, combined with `VK_KHR_surface_capabilites2`.

2) Should extensible input structs be added for these new functions:

**RESOLVED**:

- vkGetPhysicalDeviceDisplayProperties2KHR: No. The only current input is a VkPhysicalDevice. Other inputs would not make sense.
- vkGetPhysicalDeviceDisplayPlaneProperties2KHR: No. The only current input is a VkPhysicalDevice. Other inputs would not make sense.
- vkGetDisplayModeProperties2KHR: No. The only current inputs are a VkPhysicalDevice and a VkDisplayModeKHR. Other inputs would not make sense.

3) Should additional display query functions be extended?

**RESOLVED**:

- vkGetDisplayPlaneSupportedDisplaysKHR: No. Extensions should instead extend vkGetDisplayPlaneCapabilitiesKHR().

### Version History

- Revision 1, 2017-02-21 (James Jones)
  - Initial draft.

## VK_KHR_get_surface_capabilities2

### Name String

`VK_KHR_get_surface_capabilities2`

### Extension Type

Instance extension

### Registered Extension Number

120

### Revision

1

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_surface

### Contact

- James Jones [GitHub]cubanismo

### Other Extension Metadata

#### Last Modified Date

2017-02-27

#### IP Status

No known IP claims.

#### Contributors

- Ian Elliott, Google
- James Jones, NVIDIA
- Alon Or-bach, Samsung

#### Description

This extension provides new queries for device surface capabilities that can be easily extended by other extensions, without introducing any further queries. This extension can be considered the `VK_KHR_surface` equivalent of the `VK_KHR_get_physical_device_properties2` extension.

#### New Commands

- vkGetPhysicalDeviceSurfaceCapabilities2KHR
- vkGetPhysicalDeviceSurfaceFormats2KHR

#### New Structures

- VkPhysicalDeviceSurfaceInfo2KHR
- VkSurfaceCapabilities2KHR
- VkSurfaceFormat2KHR

#### New Enum Constants

- `VK_KHR_GET_SURFACE_CAPABILITIES_2_EXTENSION_NAME`
- `VK_KHR_GET_SURFACE_CAPABILITIES_2_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SURFACE_INFO_2_KHR`
  - `VK_STRUCTURE_TYPE_SURFACE_CAPABILITIES_2_KHR`
  - `VK_STRUCTURE_TYPE_SURFACE_FORMAT_2_KHR`

#### Issues

1) What should this extension be named?

**RESOLVED**: `VK_KHR_get_surface_capabilities2`. Other alternatives:

- `VK_KHR_surface2`

- One extension, combining a separate display-specific query extension.

2) Should additional WSI query functions be extended?

**RESOLVED**:

- vkGetPhysicalDeviceSurfaceCapabilitiesKHR: Yes. The need for this motivated the extension.
- vkGetPhysicalDeviceSurfaceSupportKHR: No. Currently only has boolean output. Extensions should instead extend vkGetPhysicalDeviceSurfaceCapabilities2KHR.
- vkGetPhysicalDeviceSurfaceFormatsKHR: Yes.
- vkGetPhysicalDeviceSurfacePresentModesKHR: No. Recent discussion concluded this introduced too much variability for applications to deal with. Extensions should instead extend vkGetPhysicalDeviceSurfaceCapabilities2KHR.
- vkGetPhysicalDeviceXlibPresentationSupportKHR: Not in this extension.
- vkGetPhysicalDeviceXcbPresentationSupportKHR: Not in this extension.
- vkGetPhysicalDeviceWaylandPresentationSupportKHR: Not in this extension.
- vkGetPhysicalDeviceWin32PresentationSupportKHR: Not in this extension.

### Version History

- Revision 1, 2017-02-27 (James Jones)
  - Initial draft.

## VK_KHR_global_priority

### Name String

`VK_KHR_global_priority`

### Extension Type

Device extension

### Registered Extension Number

189

### Revision

1

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### Contact

- Tobias Hector [GitHub]tobski

### Other Extension Metadata

### Last Modified Date

2021-10-22

### Contributors

- Tobias Hector, AMD
- Contributors to `VK_EXT_global_priority`
- Contributors to `VK_EXT_global_priority_query`

### Description

In Vulkan, users can specify device-scope queue priorities. In some cases it may be useful to extend this concept to a system-wide scope. This device extension allows applications to query the global queue priorities supported by a queue family, and then set a priority when creating queues. The default queue priority is `VK_QUEUE_GLOBAL_PRIORITY_MEDIUM_EXT`.

Implementations can report which global priority levels are treated differently by the implementation. It is intended primarily for use in system integration along with certain platform-specific priority enforcement rules.

The driver implementation will attempt to skew hardware resource allocation in favor of the higher-priority task. Therefore, higher-priority work may retain similar latency and throughput characteristics even if the system is congested with lower priority work.

The global priority level of a queue shall take precedence over the per-process queue priority (`VkDeviceQueueCreateInfo::pQueuePriorities`).

Abuse of this feature may result in starving the rest of the system from hardware resources. Therefore, the driver implementation may deny requests to acquire a priority above the default priority (`VK_QUEUE_GLOBAL_PRIORITY_MEDIUM_EXT`) if the caller does not have sufficient privileges. In this scenario `VK_ERROR_NOT_PERMITTED_EXT` is returned.

The driver implementation may fail the queue allocation request if resources required to complete the operation have been exhausted (either by the same process or a different process). In this scenario `VK_ERROR_INITIALIZATION_FAILED` is returned.

### New Structures

- Extending VkDeviceQueueCreateInfo:
  - VkDeviceQueueGlobalPriorityCreateInfoKHR
- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceGlobalPriorityQueryFeaturesKHR
- Extending VkQueueFamilyProperties2:

- VkQueueFamilyGlobalPriorityPropertiesKHR

#### New Enums

- VkQueueGlobalPriorityKHR

#### New Enum Constants

- `VK_KHR_GLOBAL_PRIORITY_EXTENSION_NAME`
- `VK_KHR_GLOBAL_PRIORITY_SPEC_VERSION`
- `VK_MAX_GLOBAL_PRIORITY_SIZE_KHR`
- Extending VkResult:
  - `VK_ERROR_NOT_PERMITTED_KHR`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_DEVICE_QUEUE_GLOBAL_PRIORITY_CREATE_INFO_KHR`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_GLOBAL_PRIORITY_QUERY_FEATURES_KHR`
  - `VK_STRUCTURE_TYPE_QUEUE_FAMILY_GLOBAL_PRIORITY_PROPERTIES_KHR`

#### Issues

1) Can we additionally query whether a caller is permitted to acquire a specific global queue priority in this extension?

**RESOLVED**: No. Whether a caller has enough privilege goes with the OS, and the Vulkan driver cannot really guarantee that the privilege will not change in between this query and the actual queue creation call.

2) If more than 1 queue using global priority is requested, is there a good way to know which queue is failing the device creation?

**RESOLVED**: No. There is not a good way at this moment, and it is also not quite actionable for the applications to know that because the information may not be accurate. Queue creation can fail because of runtime constraints like insufficient privilege or lack of resource, and the failure is not necessarily tied to that particular queue configuration requested.

#### Version History

- Revision 1, 2021-10-22 (Tobias Hector)
  - Initial draft

## VK_KHR_image_format_list

#### Name String

`VK_KHR_image_format_list`

**Extension Type**

Device extension

**Registered Extension Number**

148

**Revision**

1

**Ratification Status**

Ratified

**Extension and Version Dependencies**

None

**Deprecation State**

- *Promoted* to Vulkan 1.2

**Contact**

- Faith Ekstrand [GitHub]gfxstrand

**Other Extension Metadata**

**Last Modified Date**

2017-03-20

**IP Status**

No known IP claims.

**Contributors**

- Faith Ekstrand, Intel
- Jan-Harald Fredriksen, ARM
- Jeff Bolz, NVIDIA
- Jeff Leger, Qualcomm
- Neil Henning, Codeplay

**Description**

On some implementations, setting the `VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT` on image creation can cause access to that image to perform worse than an equivalent image created without `VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT` because the implementation does not know what view formats will be paired with the image.

This extension allows an application to provide the list of all formats that **can** be used with an image when it is created. The implementation may then be able to create a more efficient image that supports the subset of formats required by the application without having to support all formats in the format compatibility class of the image format.

#### Promotion to Vulkan 1.2

All functionality in this extension is included in core Vulkan 1.2, with the KHR suffix omitted. The original type, enum and command names are still available as aliases of the core functionality.

#### New Structures

- Extending VkImageCreateInfo, VkSwapchainCreateInfoKHR, VkPhysicalDeviceImageFormatInfo2:
  - VkImageFormatListCreateInfoKHR

#### New Enum Constants

- `VK_KHR_IMAGE_FORMAT_LIST_EXTENSION_NAME`
- `VK_KHR_IMAGE_FORMAT_LIST_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_IMAGE_FORMAT_LIST_CREATE_INFO_KHR`

#### Version History

- Revision 1, 2017-03-20 (Faith Ekstrand)
  - Initial revision

### VK_KHR_imageless_framebuffer

#### Name String

`VK_KHR_imageless_framebuffer`

#### Extension Type

Device extension

#### Registered Extension Number

109

#### Revision

1

#### Ratification Status

Ratified

#### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
and
VK_KHR_maintenance2
or
Vulkan Version 1.1
and

VK_KHR_image_format_list

or

Vulkan Version 1.2

### Deprecation State

- *Promoted* to Vulkan 1.2

### Contact

- Tobias Hector [GitHub]tobias

### Other Extension Metadata

### Last Modified Date

2018-12-14

### Contributors

- Tobias Hector
- Graham Wihlidal

### Description

This extension allows framebuffers to be created without the need for creating images first, allowing more flexibility in how they are used, and avoiding the need for many of the confusing compatibility rules.

Framebuffers are now created with a small amount of additional metadata about the image views that will be used in VkFramebufferAttachmentsCreateInfoKHR, and the actual image views are provided at render pass begin time via VkRenderPassAttachmentBeginInfoKHR.

### Promotion to Vulkan 1.2

All functionality in this extension is included in core Vulkan 1.2, with the KHR suffix omitted. The original type, enum and command names are still available as aliases of the core functionality.

### New Structures

- VkFramebufferAttachmentImageInfoKHR
- Extending VkFramebufferCreateInfo:
  - VkFramebufferAttachmentsCreateInfoKHR
- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceImagelessFramebufferFeaturesKHR
- Extending VkRenderPassBeginInfo:
  - VkRenderPassAttachmentBeginInfoKHR

### New Enum Constants

- `VK_KHR_IMAGELESS_FRAMEBUFFER_EXTENSION_NAME`

- VK_KHR_IMAGELESS_FRAMEBUFFER_SPEC_VERSION
- Extending VkFramebufferCreateFlagBits:
  - VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT_KHR
- Extending VkStructureType:
  - VK_STRUCTURE_TYPE_FRAMEBUFFER_ATTACHMENTS_CREATE_INFO_KHR
  - VK_STRUCTURE_TYPE_FRAMEBUFFER_ATTACHMENT_IMAGE_INFO_KHR
  - VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_IMAGELESS_FRAMEBUFFER_FEATURES_KHR
  - VK_STRUCTURE_TYPE_RENDER_PASS_ATTACHMENT_BEGIN_INFO_KHR

#### Version History

- Revision 1, 2018-12-14 (Tobias Hector)
  - Internal revisions

## VK_KHR_incremental_present

#### Name String

VK_KHR_incremental_present

#### Extension Type

Device extension

#### Registered Extension Number

85

#### Revision

2

#### Ratification Status

Ratified

#### Extension and Version Dependencies

VK_KHR_swapchain

#### Contact

- Ian Elliott [GitHub]ianelliottus

#### Other Extension Metadata

#### Last Modified Date

2016-11-02

#### IP Status

No known IP claims.

### Contributors

- Ian Elliott, Google
- Jesse Hall, Google
- Alon Or-bach, Samsung
- James Jones, NVIDIA
- Daniel Rakos, AMD
- Ray Smith, ARM
- Mika Isojarvi, Google
- Jeff Juliano, NVIDIA
- Jeff Bolz, NVIDIA

### Description

This device extension extends vkQueuePresentKHR, from the `VK_KHR_swapchain` extension, allowing an application to specify a list of rectangular, modified regions of each image to present. This should be used in situations where an application is only changing a small portion of the presentable images within a swapchain, since it enables the presentation engine to avoid wasting time presenting parts of the surface that have not changed.

This extension is leveraged from the `EGL_KHR_swap_buffers_with_damage` extension.

### New Structures

- VkPresentRegionKHR
- VkRectLayerKHR
- Extending VkPresentInfoKHR:
  - VkPresentRegionsKHR

### New Enum Constants

- `VK_KHR_INCREMENTAL_PRESENT_EXTENSION_NAME`
- `VK_KHR_INCREMENTAL_PRESENT_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PRESENT_REGIONS_KHR`

### Issues

1) How should we handle steroescopic-3D swapchains? We need to add a layer for each rectangle. One approach is to create another struct containing the VkRect2D plus layer, and have VkPresentRegionsKHR point to an array of that struct. Another approach is to have two parallel arrays, `pRectangles` and `pLayers`, where `pRectangles`[i] and `pLayers`[i] must be used together. Which approach should we use, and if the array of a new structure, what should that be called?

**RESOLVED**: Create a new structure, which is a VkRect2D plus a layer, and will be called

VkRectLayerKHR.

2) Where is the origin of the VkRectLayerKHR?

**RESOLVED**: The upper left corner of the presentable image(s) of the swapchain, per the definition of framebuffer coordinates.

3) Does the rectangular region, VkRectLayerKHR, specify pixels of the swapchain's image(s), or of the surface?

**RESOLVED**: Of the image(s). Some presentation engines may scale the pixels of a swapchain's image(s) to the size of the surface. The size of the swapchain's image(s) will be consistent, where the size of the surface may vary over time.

4) What if all of the rectangles for a given swapchain contain a width and/or height of zero?

**RESOLVED**: The application is indicating that no pixels changed since the last present. The presentation engine may use such a hint and not update any pixels for the swapchain. However, all other semantics of vkQueuePresentKHR must still be honored, including waiting for semaphores to signal.

5) When the swapchain is created with `VkSwapchainCreateInfoKHR::preTransform` set to a value other than `VK_SURFACE_TRANSFORM_IDENTITY_BIT_KHR`, should the rectangular region, VkRectLayerKHR, be transformed to align with the `preTransform`?

**RESOLVED**: No. The rectangular region in VkRectLayerKHR should not be transformed. As such, it may not align with the extents of the swapchain's image(s). It is the responsibility of the presentation engine to transform the rectangular region. This matches the behavior of the Android presentation engine, which set the precedent.

#### Version History

- Revision 1, 2016-11-02 (Ian Elliott)
  - Internal revisions
- Revision 2, 2021-03-18 (Ian Elliott)
  - Clarified alignment of rectangles for presentation engines that support transformed swapchains.

## VK_KHR_index_type_uint8

#### Name String

`VK_KHR_index_type_uint8`

#### Extension Type

Device extension

#### Registered Extension Number

534

### Revision

1

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### Contact

- Piers Daniell [GitHub]pdaniell-nv

### Other Extension Metadata

### Last Modified Date

2023-06-06

### IP Status

No known IP claims.

### Contributors

- Jeff Bolz, NVIDIA

### Description

This extension allows `uint8_t` indices to be used with vkCmdBindIndexBuffer.

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceIndexTypeUint8FeaturesKHR

### New Enum Constants

- `VK_KHR_INDEX_TYPE_UINT8_EXTENSION_NAME`
- `VK_KHR_INDEX_TYPE_UINT8_SPEC_VERSION`
- Extending VkIndexType:
  - `VK_INDEX_TYPE_UINT8_KHR`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_INDEX_TYPE_UINT8_FEATURES_KHR`

### Version History

- Revision 1, 2023-06-06 (Piers Daniell)

- Internal revisions

## VK_KHR_line_rasterization

### Name String

`VK_KHR_line_rasterization`

### Extension Type

Device extension

### Registered Extension Number

535

### Revision

1

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### Contact

- Piers Daniell [GitHub]pdaniell-nv

### Other Extension Metadata

### Last Modified Date

2023-06-08

### IP Status

No known IP claims.

### Contributors

- Jeff Bolz, NVIDIA
- Allen Jensen, NVIDIA
- Faith Ekstrand, Intel

### Description

This extension adds some line rasterization features that are commonly used in CAD applications and supported in other APIs like OpenGL. Bresenham-style line rasterization is supported, smooth rectangular lines (coverage to alpha) are supported, and stippled lines are supported for all three line rasterization modes.

### New Commands

- vkCmdSetLineStippleKHR

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceLineRasterizationFeaturesKHR
- Extending VkPhysicalDeviceProperties2:
  - VkPhysicalDeviceLineRasterizationPropertiesKHR
- Extending VkPipelineRasterizationStateCreateInfo:
  - VkPipelineRasterizationLineStateCreateInfoKHR

### New Enums

- VkLineRasterizationModeKHR

### New Enum Constants

- `VK_KHR_LINE_RASTERIZATION_EXTENSION_NAME`
- `VK_KHR_LINE_RASTERIZATION_SPEC_VERSION`
- Extending VkDynamicState:
  - `VK_DYNAMIC_STATE_LINE_STIPPLE_KHR`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_LINE_RASTERIZATION_FEATURES_KHR`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_LINE_RASTERIZATION_PROPERTIES_KHR`
  - `VK_STRUCTURE_TYPE_PIPELINE_RASTERIZATION_LINE_STATE_CREATE_INFO_KHR`

### Issues

1) Do we need to support Bresenham-style and smooth lines with more than one rasterization sample? i.e. the equivalent of glDisable(GL_MULTISAMPLE) in OpenGL when the framebuffer has more than one sample?

**RESOLVED**: Yes. For simplicity, Bresenham line rasterization carries forward a few restrictions from OpenGL, such as not supporting per-sample shading, alpha to coverage, or alpha to one.

### Version History

- Revision 1, 2019-05-09 (Jeff Bolz)
  - Initial draft

## VK_KHR_load_store_op_none

### Name String

`VK_KHR_load_store_op_none`

### Extension Type

Device extension

### Registered Extension Number

527

### Revision

1

### Ratification Status

Ratified

### Extension and Version Dependencies

None

### Contact

- Shahbaz Youssefi [GitHub]syoussefi

### Extension Proposal

VK_KHR_load_store_op_none

### Other Extension Metadata

### Last Modified Date

2023-05-16

### Contributors

- Shahbaz Youssefi, Google
- Bill Licea-Kane, Qualcomm Technologies, Inc.
- Tobias Hector, AMD

### Description

This extension provides `VK_ATTACHMENT_LOAD_OP_NONE_KHR` and `VK_ATTACHMENT_STORE_OP_NONE_KHR`, which are identically promoted from the `VK_EXT_load_store_op_none` extension.

### New Enum Constants

- `VK_KHR_LOAD_STORE_OP_NONE_EXTENSION_NAME`
- `VK_KHR_LOAD_STORE_OP_NONE_SPEC_VERSION`
- Extending VkAttachmentLoadOp:
  - `VK_ATTACHMENT_LOAD_OP_NONE_KHR`
- Extending VkAttachmentStoreOp:

- `VK_ATTACHMENT_STORE_OP_NONE_KHR`

| | |
|---|---|
| **NOTE** | While `VK_ATTACHMENT_STORE_OP_NONE` is part of Vulkan 1.3, this extension was not promoted to core either in whole or in part. This functionality was promoted from `VK_KHR_dynamic_rendering`. |

### Version History

- Revision 1, 2023-05-16 (Shahbaz Youssefi)
  - Initial revision, based on VK_EXT_load_store_op_none.

## VK_KHR_maintenance4

### Name String

`VK_KHR_maintenance4`

### Extension Type

Device extension

### Registered Extension Number

414

### Revision

2

### Ratification Status

Ratified

### Extension and Version Dependencies

Vulkan Version 1.1

### Deprecation State

- *Promoted* to Vulkan 1.3

### Contact

- Piers Daniell [GitHub]pdaniell-nv

### Other Extension Metadata

### Last Modified Date

2021-10-25

### Interactions and External Dependencies

- Requires SPIR-V 1.2 for `LocalSizeId`

### Contributors

- Lionel Duc, NVIDIA
- Faith Ekstrand, Intel

- Spencer Fricke, Samsung
- Tobias Hector, AMD
- Lionel Landwerlin, Intel
- Graeme Leese, Broadcom
- Tom Olson, Arm
- Stu Smith, AMD
- Yiwei Zhang, Google

### Description

`VK_KHR_maintenance4` adds a collection of minor features, none of which would warrant an entire extension of their own.

The new features are as follows:

- Allow the application to destroy their VkPipelineLayout object immediately after it was used to create another object. It is no longer necessary to keep its handle valid while the created object is in use.
- Add a new `maxBufferSize` implementation-defined limit for the maximum size `VkBuffer` that **can** be created.
- Add support for the SPIR-V 1.2 `LocalSizeId` execution mode, which can be used as an alternative to `LocalSize` to specify the local workgroup size with specialization constants.
- Add a guarantee that images created with identical creation parameters will always have the same alignment requirements.
- Add new vkGetDeviceBufferMemoryRequirementsKHR, vkGetDeviceImageMemoryRequirementsKHR, and vkGetDeviceImageSparseMemoryRequirementsKHR to allow the application to query the image memory requirements without having to create an image object and query it.
- Relax the requirement that push constants must be initialized before they are dynamically accessed.
- Relax the interface matching rules to allow a larger output vector to match with a smaller input vector, with additional values being discarded.
- Add a guarantee for buffer memory requirement that the size memory requirement is never greater than the result of aligning create size with the alignment memory requirement.

### New Commands

- vkGetDeviceBufferMemoryRequirementsKHR
- vkGetDeviceImageMemoryRequirementsKHR
- vkGetDeviceImageSparseMemoryRequirementsKHR

### New Structures

- VkDeviceBufferMemoryRequirementsKHR
- VkDeviceImageMemoryRequirementsKHR
- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceMaintenance4FeaturesKHR
- Extending VkPhysicalDeviceProperties2:
  - VkPhysicalDeviceMaintenance4PropertiesKHR

### New Enum Constants

- `VK_KHR_MAINTENANCE_4_EXTENSION_NAME`
- `VK_KHR_MAINTENANCE_4_SPEC_VERSION`
- Extending VkImageAspectFlagBits:
  - `VK_IMAGE_ASPECT_NONE_KHR`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_DEVICE_BUFFER_MEMORY_REQUIREMENTS_KHR`
  - `VK_STRUCTURE_TYPE_DEVICE_IMAGE_MEMORY_REQUIREMENTS_KHR`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MAINTENANCE_4_FEATURES_KHR`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MAINTENANCE_4_PROPERTIES_KHR`

### Promotion to Vulkan 1.3

Functionality in this extension is included in core Vulkan 1.3, with the KHR suffix omitted. The original type, enum and command names are still available as aliases of the core functionality.

### Issues

None.

### Version History

- Revision 1, 2021-08-18 (Piers Daniell)
  - Internal revisions
- Revision 2, 2021-10-25 (Yiwei Zhang)
  - More guarantees on buffer memory requirements

## VK_KHR_maintenance5

### Name String

`VK_KHR_maintenance5`

**Extension Type**

Device extension

**Registered Extension Number**

471

**Revision**

1

**Ratification Status**

Ratified

**Extension and Version Dependencies**

Vulkan Version 1.1
and
VK_KHR_dynamic_rendering
or
Vulkan Version 1.3

**API Interactions**

- Interacts with VK_VERSION_1_1
- Interacts with VK_VERSION_1_2
- Interacts with VK_VERSION_1_3
- Interacts with VK_EXT_attachment_feedback_loop_layout
- Interacts with VK_EXT_buffer_device_address
- Interacts with VK_EXT_conditional_rendering
- Interacts with VK_EXT_descriptor_buffer
- Interacts with VK_EXT_fragment_density_map
- Interacts with VK_EXT_graphics_pipeline_library
- Interacts with VK_EXT_opacity_micromap
- Interacts with VK_EXT_pipeline_creation_cache_control
- Interacts with VK_EXT_pipeline_protected_access
- Interacts with VK_EXT_transform_feedback
- Interacts with VK_KHR_acceleration_structure
- Interacts with VK_KHR_buffer_device_address
- Interacts with VK_KHR_device_group
- Interacts with VK_KHR_dynamic_rendering
- Interacts with VK_KHR_fragment_shading_rate
- Interacts with VK_KHR_pipeline_executable_properties
- Interacts with VK_KHR_pipeline_library

- Interacts with VK_KHR_ray_tracing_pipeline
- Interacts with VK_KHR_video_decode_queue
- Interacts with VK_KHR_video_encode_queue
- Interacts with VK_NV_device_generated_commands
- Interacts with VK_NV_displacement_micromap
- Interacts with VK_NV_ray_tracing
- Interacts with VK_NV_ray_tracing_motion_blur

**Contact**

- Stu Smith [GitHub]stu-s

**Extension Proposal**

VK_KHR_maintenance5

**Other Extension Metadata**

**Last Modified Date**

2023-05-02

**Interactions and External Dependencies**

**Contributors**

- Stu Smith, AMD
- Tobias Hector, AMD
- Shahbaz Youssefi, Google
- Slawomir Cygan, Intel
- Lionel Landwerlin, Intel
- James Fitzpatrick, Imagination Technologies
- Andrew Garrard, Imagination Technologies
- Ralph Potter, Samsung
- Pan Gao, Huawei
- Jan-Harald Fredriksen, ARM
- Jon Leech, Khronos
- Mike Blumenkrantz, Valve

**Description**

`VK_KHR_maintenance5` adds a collection of minor features, none of which would warrant an entire extension of their own.

The new features are as follows:

- A new `VK_FORMAT_A1B5G5R5_UNORM_PACK16_KHR` format

- A new VK_FORMAT_A8_UNORM_KHR format
- A property to indicate that multisample coverage operations are performed after sample counting in EarlyFragmentTests mode
- Relax VkBufferView creation requirements by allowing subsets of the associated VkBuffer usage using VkBufferUsageFlags2CreateInfoKHR
- A new command vkCmdBindIndexBuffer2KHR, allowing a range of memory to be bound as an index buffer
- vkGetDeviceProcAddr must return NULL for supported core functions beyond the version requested by the application.
- A property to indicate that the sample mask test is performed after sample counting in EarlyFragmentTests mode
- vkCmdBindVertexBuffers2 now supports using VK_WHOLE_SIZE in the pSizes parameter.
- A default size of 1.0 is used if PointSize is not written
- Shader modules are deprecated - applications can now pass VkShaderModuleCreateInfo as a chained struct to pipeline creation via VkPipelineShaderStageCreateInfo
- A function vkGetRenderingAreaGranularityKHR to query the optimal render area for a dynamic rendering instance.
- A property to indicate that depth/stencil texturing operations with VK_COMPONENT_SWIZZLE_ONE have defined behavior
- Add vkGetImageSubresourceLayout2KHR and a new function vkGetDeviceImageSubresourceLayoutKHR to allow the application to query the image memory layout without having to create an image object and query it.
- Allow VK_REMAINING_ARRAY_LAYERS as the layerCount member of VkImageSubresourceLayers
- Adds stronger guarantees for propagation of VK_ERROR_DEVICE_LOST return values
- A property to indicate whether PointSize controls the final rasterization of polygons if polygon mode is VK_POLYGON_MODE_POINT
- Two properties to indicate the non-strict line rasterization algorithm used
- Two new flags words VkPipelineCreateFlagBits2KHR and VkBufferUsageFlagBits2KHR
- Physical-device-level functions can now be called with any value in the valid range for a type beyond the defined enumerants, such that applications can avoid checking individual features, extensions, or versions before querying supported properties of a particular enumerant.
- Clarification that copies between images of any type are allowed, treating 1D images as 2D images with a height of 1.

### New Commands

- vkCmdBindIndexBuffer2KHR
- vkGetDeviceImageSubresourceLayoutKHR
- vkGetImageSubresourceLayout2KHR
- vkGetRenderingAreaGranularityKHR

### New Structures

- VkDeviceImageSubresourceInfoKHR
- VkImageSubresource2KHR
- VkRenderingAreaInfoKHR
- VkSubresourceLayout2KHR
- Extending VkBufferViewCreateInfo, VkBufferCreateInfo, VkPhysicalDeviceExternalBufferInfo, VkDescriptorBufferBindingInfoEXT:
  - VkBufferUsageFlags2CreateInfoKHR
- Extending VkComputePipelineCreateInfo, VkGraphicsPipelineCreateInfo, VkRayTracingPipelineCreateInfoNV, VkRayTracingPipelineCreateInfoKHR:
  - VkPipelineCreateFlags2CreateInfoKHR
- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceMaintenance5FeaturesKHR
- Extending VkPhysicalDeviceProperties2:
  - VkPhysicalDeviceMaintenance5PropertiesKHR

### New Enums

- VkBufferUsageFlagBits2KHR
- VkPipelineCreateFlagBits2KHR

### New Bitmasks

- VkBufferUsageFlags2KHR
- VkPipelineCreateFlags2KHR

### New Enum Constants

- `VK_KHR_MAINTENANCE_5_EXTENSION_NAME`
- `VK_KHR_MAINTENANCE_5_SPEC_VERSION`
- Extending VkFormat:
  - `VK_FORMAT_A1B5G5R5_UNORM_PACK16_KHR`
  - `VK_FORMAT_A8_UNORM_KHR`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_BUFFER_USAGE_FLAGS_2_CREATE_INFO_KHR`
  - `VK_STRUCTURE_TYPE_DEVICE_IMAGE_SUBRESOURCE_INFO_KHR`
  - `VK_STRUCTURE_TYPE_IMAGE_SUBRESOURCE_2_KHR`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MAINTENANCE_5_FEATURES_KHR`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MAINTENANCE_5_PROPERTIES_KHR`

- VK_STRUCTURE_TYPE_PIPELINE_CREATE_FLAGS_2_CREATE_INFO_KHR
- VK_STRUCTURE_TYPE_RENDERING_AREA_INFO_KHR
- VK_STRUCTURE_TYPE_SUBRESOURCE_LAYOUT_2_KHR

If VK_KHR_dynamic_rendering or Vulkan Version 1.3 and VK_EXT_fragment_density_map is supported:

- Extending VkPipelineCreateFlagBits2KHR:
  - VK_PIPELINE_CREATE_2_RENDERING_FRAGMENT_DENSITY_MAP_ATTACHMENT_BIT_EXT

If VK_KHR_dynamic_rendering or Vulkan Version 1.3 and VK_KHR_fragment_shading_rate is supported:

- Extending VkPipelineCreateFlagBits2KHR:
  - VK_PIPELINE_CREATE_2_RENDERING_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR

If VK_EXT_attachment_feedback_loop_layout is supported:

- Extending VkPipelineCreateFlagBits2KHR:
  - VK_PIPELINE_CREATE_2_COLOR_ATTACHMENT_FEEDBACK_LOOP_BIT_EXT
  - VK_PIPELINE_CREATE_2_DEPTH_STENCIL_ATTACHMENT_FEEDBACK_LOOP_BIT_EXT

If VK_EXT_conditional_rendering is supported:

- Extending VkBufferUsageFlagBits2KHR:
  - VK_BUFFER_USAGE_2_CONDITIONAL_RENDERING_BIT_EXT

If VK_EXT_descriptor_buffer is supported:

- Extending VkBufferUsageFlagBits2KHR:
  - VK_BUFFER_USAGE_2_PUSH_DESCRIPTORS_DESCRIPTOR_BUFFER_BIT_EXT
  - VK_BUFFER_USAGE_2_RESOURCE_DESCRIPTOR_BUFFER_BIT_EXT
  - VK_BUFFER_USAGE_2_SAMPLER_DESCRIPTOR_BUFFER_BIT_EXT
- Extending VkPipelineCreateFlagBits2KHR:
  - VK_PIPELINE_CREATE_2_DESCRIPTOR_BUFFER_BIT_EXT

If VK_EXT_graphics_pipeline_library is supported:

- Extending VkPipelineCreateFlagBits2KHR:
  - VK_PIPELINE_CREATE_2_LINK_TIME_OPTIMIZATION_BIT_EXT
  - VK_PIPELINE_CREATE_2_RETAIN_LINK_TIME_OPTIMIZATION_INFO_BIT_EXT

If VK_EXT_opacity_micromap is supported:

- Extending VkBufferUsageFlagBits2KHR:

    - VK_BUFFER_USAGE_2_MICROMAP_BUILD_INPUT_READ_ONLY_BIT_EXT
    - VK_BUFFER_USAGE_2_MICROMAP_STORAGE_BIT_EXT
- Extending VkPipelineCreateFlagBits2KHR:
    - VK_PIPELINE_CREATE_2_RAY_TRACING_OPACITY_MICROMAP_BIT_EXT

If VK_EXT_pipeline_protected_access is supported:

- Extending VkPipelineCreateFlagBits2KHR:
    - VK_PIPELINE_CREATE_2_NO_PROTECTED_ACCESS_BIT_EXT
    - VK_PIPELINE_CREATE_2_PROTECTED_ACCESS_ONLY_BIT_EXT

If VK_EXT_transform_feedback is supported:

- Extending VkBufferUsageFlagBits2KHR:
    - VK_BUFFER_USAGE_2_TRANSFORM_FEEDBACK_BUFFER_BIT_EXT
    - VK_BUFFER_USAGE_2_TRANSFORM_FEEDBACK_COUNTER_BUFFER_BIT_EXT

If VK_KHR_acceleration_structure is supported:

- Extending VkBufferUsageFlagBits2KHR:
    - VK_BUFFER_USAGE_2_ACCELERATION_STRUCTURE_BUILD_INPUT_READ_ONLY_BIT_KHR
    - VK_BUFFER_USAGE_2_ACCELERATION_STRUCTURE_STORAGE_BIT_KHR

If VK_KHR_pipeline_executable_properties is supported:

- Extending VkPipelineCreateFlagBits2KHR:
    - VK_PIPELINE_CREATE_2_CAPTURE_INTERNAL_REPRESENTATIONS_BIT_KHR
    - VK_PIPELINE_CREATE_2_CAPTURE_STATISTICS_BIT_KHR

If VK_KHR_pipeline_library is supported:

- Extending VkPipelineCreateFlagBits2KHR:
    - VK_PIPELINE_CREATE_2_LIBRARY_BIT_KHR

If VK_KHR_ray_tracing_pipeline is supported:

- Extending VkBufferUsageFlagBits2KHR:
    - VK_BUFFER_USAGE_2_SHADER_BINDING_TABLE_BIT_KHR
- Extending VkPipelineCreateFlagBits2KHR:
    - VK_PIPELINE_CREATE_2_RAY_TRACING_NO_NULL_ANY_HIT_SHADERS_BIT_KHR
    - VK_PIPELINE_CREATE_2_RAY_TRACING_NO_NULL_CLOSEST_HIT_SHADERS_BIT_KHR
    - VK_PIPELINE_CREATE_2_RAY_TRACING_NO_NULL_INTERSECTION_SHADERS_BIT_KHR
    - VK_PIPELINE_CREATE_2_RAY_TRACING_NO_NULL_MISS_SHADERS_BIT_KHR

    - VK_PIPELINE_CREATE_2_RAY_TRACING_SHADER_GROUP_HANDLE_CAPTURE_REPLAY_BIT_KHR
    - VK_PIPELINE_CREATE_2_RAY_TRACING_SKIP_AABBS_BIT_KHR
    - VK_PIPELINE_CREATE_2_RAY_TRACING_SKIP_TRIANGLES_BIT_KHR

If VK_KHR_video_decode_queue is supported:

- Extending VkBufferUsageFlagBits2KHR:
    - VK_BUFFER_USAGE_2_VIDEO_DECODE_DST_BIT_KHR
    - VK_BUFFER_USAGE_2_VIDEO_DECODE_SRC_BIT_KHR

If VK_KHR_video_encode_queue is supported:

- Extending VkBufferUsageFlagBits2KHR:
    - VK_BUFFER_USAGE_2_VIDEO_ENCODE_DST_BIT_KHR
    - VK_BUFFER_USAGE_2_VIDEO_ENCODE_SRC_BIT_KHR

If VK_NV_device_generated_commands is supported:

- Extending VkPipelineCreateFlagBits2KHR:
    - VK_PIPELINE_CREATE_2_INDIRECT_BINDABLE_BIT_NV

If VK_NV_displacement_micromap is supported:

- Extending VkPipelineCreateFlagBits2KHR:
    - VK_PIPELINE_CREATE_2_RAY_TRACING_DISPLACEMENT_MICROMAP_BIT_NV

If VK_NV_ray_tracing is supported:

- Extending VkBufferUsageFlagBits2KHR:
    - VK_BUFFER_USAGE_2_RAY_TRACING_BIT_NV
- Extending VkPipelineCreateFlagBits2KHR:
    - VK_PIPELINE_CREATE_2_DEFER_COMPILE_BIT_NV

If VK_NV_ray_tracing_motion_blur is supported:

- Extending VkPipelineCreateFlagBits2KHR:
    - VK_PIPELINE_CREATE_2_RAY_TRACING_ALLOW_MOTION_BIT_NV

If Vulkan Version 1.1 or VK_KHR_device_group is supported:

- Extending VkPipelineCreateFlagBits2KHR:
    - VK_PIPELINE_CREATE_2_DISPATCH_BASE_BIT_KHR
    - VK_PIPELINE_CREATE_2_VIEW_INDEX_FROM_DEVICE_INDEX_BIT_KHR

If Vulkan Version 1.2 or VK_KHR_buffer_device_address or VK_EXT_buffer_device_address is supported:

- Extending VkBufferUsageFlagBits2KHR:
  - `VK_BUFFER_USAGE_2_SHADER_DEVICE_ADDRESS_BIT_KHR`

If Vulkan Version 1.3 or VK_EXT_pipeline_creation_cache_control is supported:

- Extending VkPipelineCreateFlagBits2KHR:
  - `VK_PIPELINE_CREATE_2_EARLY_RETURN_ON_FAILURE_BIT_KHR`
  - `VK_PIPELINE_CREATE_2_FAIL_ON_PIPELINE_COMPILE_REQUIRED_BIT_KHR`

#### Issues

None.

#### Version History

- Revision 1, 2022-12-12 (Stu Smith)
  - Initial revision

## VK_KHR_maintenance6

#### Name String

`VK_KHR_maintenance6`

#### Extension Type

Device extension

#### Registered Extension Number

546

#### Revision

1

#### Ratification Status

Ratified

#### Extension and Version Dependencies

Vulkan Version 1.1

#### API Interactions

- Interacts with VK_EXT_descriptor_buffer
- Interacts with VK_KHR_push_descriptor

#### Contact

- Jon Leech [GitHub]oddhack

#### Extension Proposal

VK_KHR_maintenance6

### Other Extension Metadata

#### Last Modified Date

2023-08-03

#### Interactions and External Dependencies

- Interacts with `VK_EXT_robustness2`

#### Contributors

- Jon Leech, Khronos
- Stu Smith, AMD
- Mike Blumenkrantz, Valve
- Ralph Potter, Samsung
- James Fitzpatrick, Imagination Technologies
- Piers Daniell, NVIDIA
- Daniel Story, Nintendo

### Description

VK_KHR_maintenance6 adds a collection of minor features, none of which would warrant an entire extension of their own.

The new features are as follows:

- VkBindMemoryStatusKHR may be included in the `pNext` chain of VkBindBufferMemoryInfo and VkBindImageMemoryInfo, allowing applications to identify individual resources for which memory binding failed during calls to vkBindBufferMemory2 and vkBindImageMemory2.
- A new property `fragmentShadingRateClampCombinerInputs` to indicate if an implementation clamps the inputs to fragment shading rate combiner operations.
- VK_NULL_HANDLE is allowed to be used when binding an index buffer, instead of a valid VkBuffer handle. When the `nullDescriptor` feature is enabled, every index fetched results in a value of zero.
- A new property `maxCombinedImageSamplerDescriptorCount` to indicate the maximum number of descriptors needed for any of the formats that require a sampler $Y'C_BC_R$ conversion supported by the implementation.
- A new property `blockTexelViewCompatibleMultipleLayers` indicating whether `VK_IMAGE_CREATE_BLOCK_TEXEL_VIEW_COMPATIBLE_BIT` is allowed to be used with `layerCount` > 1
- `pNext` extensible *2 versions of all descriptor binding commands.

### New Commands

- vkCmdBindDescriptorSets2KHR
- vkCmdPushConstants2KHR

If VK_EXT_descriptor_buffer is supported:

- vkCmdBindDescriptorBufferEmbeddedSamplers2EXT
- vkCmdSetDescriptorBufferOffsets2EXT

If VK_KHR_push_descriptor is supported:

- vkCmdPushDescriptorSet2KHR
- vkCmdPushDescriptorSetWithTemplate2KHR

**New Structures**

- VkBindDescriptorSetsInfoKHR
- VkPushConstantsInfoKHR
- Extending VkBindBufferMemoryInfo, VkBindImageMemoryInfo:
  - VkBindMemoryStatusKHR
- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceMaintenance6FeaturesKHR
- Extending VkPhysicalDeviceProperties2:
  - VkPhysicalDeviceMaintenance6PropertiesKHR

If VK_EXT_descriptor_buffer is supported:

- VkBindDescriptorBufferEmbeddedSamplersInfoEXT
- VkSetDescriptorBufferOffsetsInfoEXT

If VK_KHR_push_descriptor is supported:

- VkPushDescriptorSetInfoKHR
- VkPushDescriptorSetWithTemplateInfoKHR

**New Enum Constants**

- `VK_KHR_MAINTENANCE_6_EXTENSION_NAME`
- `VK_KHR_MAINTENANCE_6_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_BIND_DESCRIPTOR_SETS_INFO_KHR`
  - `VK_STRUCTURE_TYPE_BIND_MEMORY_STATUS_KHR`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MAINTENANCE_6_FEATURES_KHR`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MAINTENANCE_6_PROPERTIES_KHR`
  - `VK_STRUCTURE_TYPE_PUSH_CONSTANTS_INFO_KHR`

If VK_EXT_descriptor_buffer is supported:

- Extending VkStructureType:

- `VK_STRUCTURE_TYPE_BIND_DESCRIPTOR_BUFFER_EMBEDDED_SAMPLERS_INFO_EXT`
- `VK_STRUCTURE_TYPE_SET_DESCRIPTOR_BUFFER_OFFSETS_INFO_EXT`

If VK_KHR_push_descriptor is supported:

- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PUSH_DESCRIPTOR_SET_INFO_KHR`
  - `VK_STRUCTURE_TYPE_PUSH_DESCRIPTOR_SET_WITH_TEMPLATE_INFO_KHR`

#### Issues

None.

#### Version History

- Revision 1, 2023-08-01 (Jon Leech)
  - Initial revision

## VK_KHR_maintenance7

#### Name String

`VK_KHR_maintenance7`

#### Extension Type

Device extension

#### Registered Extension Number

563

#### Revision

1

#### Ratification Status

Ratified

#### Extension and Version Dependencies

Vulkan Version 1.1

#### Contact

- Mike Blumenkrantz [GitHub]zmike

#### Extension Proposal

VK_KHR_maintenance7

#### Other Extension Metadata

#### Last Modified Date

2024-01-30

### Interactions and External Dependencies

### Contributors

- Mike Blumenkrantz, Valve
- Hans-Kristian Arntzen, Valve
- Pan Gao, Huawei
- Tobias Hector, AMD
- Jon Leech, Khronos
- Daniel Story, Nintendo
- Shahbaz Youssefi, Google
- Yiwei Zhang, Google
- Matthew Netsch, Qualcomm

### Description

VK_KHR_maintenance7 adds a collection of minor features, none of which would warrant an entire extension of their own.

The proposed new features are as follows:

- Add a property query to determine if a framebuffer writes to depth or stencil aspect does not trigger a write access in the sibling aspect. For example, this allows sampling stencil aspect as a texture while rendering to the sibling depth attachment and vice-versa given appropriate image layouts.
- Add a way to query information regarding the underlying devices in environments where the Vulkan implementation is provided through layered implementations. For example, running on Mesa/Venus, driver ID is returned as `VK_DRIVER_ID_MESA_VENUS`, but it can be necessary to know what the real driver under the hood is. The new VkPhysicalDeviceLayeredApiPropertiesKHR structure can be used to gather information regarding layers underneath the top-level physical device.
- Promote `VK_RENDERING_CONTENTS_INLINE_BIT_EXT` and `VK_SUBPASS_CONTENTS_INLINE_AND_SECONDARY_COMMAND_BUFFERS_EXT` to KHR
- Add a limit to report the maximum total count of dynamic uniform buffers and dynamic storage buffers that can be included in a pipeline layout.
- Require that for an unsigned integer query, the 32-bit result value **must** be equal to the 32 least significant bits of the equivalent 64-bit result value.
- Add query for robust access support when using fragment shading rate attachments

### New Structures

- VkPhysicalDeviceLayeredApiPropertiesKHR
- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceMaintenance7FeaturesKHR

- Extending VkPhysicalDeviceLayeredApiPropertiesKHR:
  - VkPhysicalDeviceLayeredApiVulkanPropertiesKHR
- Extending VkPhysicalDeviceProperties2:
  - VkPhysicalDeviceLayeredApiPropertiesListKHR
  - VkPhysicalDeviceMaintenance7PropertiesKHR

### New Enums

- VkPhysicalDeviceLayeredApiKHR

### New Enum Constants

- `VK_KHR_MAINTENANCE_7_EXTENSION_NAME`
- `VK_KHR_MAINTENANCE_7_SPEC_VERSION`
- Extending VkRenderingFlagBits:
  - `VK_RENDERING_CONTENTS_INLINE_BIT_KHR`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_LAYERED_API_PROPERTIES_KHR`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_LAYERED_API_PROPERTIES_LIST_KHR`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_LAYERED_API_VULKAN_PROPERTIES_KHR`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MAINTENANCE_7_FEATURES_KHR`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MAINTENANCE_7_PROPERTIES_KHR`
- Extending VkSubpassContents:
  - `VK_SUBPASS_CONTENTS_INLINE_AND_SECONDARY_COMMAND_BUFFERS_KHR`

### Issues

None.

### Version History

- Revision 1, 2024-01-30 (Jon Leech)
  - Initial revision

## VK_KHR_map_memory2

### Name String

`VK_KHR_map_memory2`

### Extension Type

Device extension

### Registered Extension Number

272

### Revision

1

### Ratification Status

Ratified

### Extension and Version Dependencies

None

### Contact

- Faith Ekstrand [GitHub]gfxstrand

### Extension Proposal

VK_KHR_map_memory2

### Other Extension Metadata

### Last Modified Date

2023-03-14

### Interactions and External Dependencies

- None

### Contributors

- Faith Ekstrand, Collabora
- Tobias Hector, AMD

### Description

This extension provides extensible versions of the Vulkan memory map and unmap commands. The new commands are functionally identical to the core commands, except that their parameters are specified using extensible structures that can be used to pass extension-specific information.

### New Commands

- vkMapMemory2KHR
- vkUnmapMemory2KHR

### New Structures

- VkMemoryMapInfoKHR
- VkMemoryUnmapInfoKHR

### New Enums

- VkMemoryUnmapFlagBitsKHR

### New Bitmasks

- VkMemoryUnmapFlagsKHR

### New Enum Constants

- `VK_KHR_MAP_MEMORY_2_EXTENSION_NAME`
- `VK_KHR_MAP_MEMORY_2_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_MEMORY_MAP_INFO_KHR`
  - `VK_STRUCTURE_TYPE_MEMORY_UNMAP_INFO_KHR`

### Version History

- Revision 0, 2022-08-03 (Faith Ekstrand)
  - Internal revisions
- Revision 1, 2023-03-14
  - Public release

## VK_KHR_performance_query

### Name String

`VK_KHR_performance_query`

### Extension Type

Device extension

### Registered Extension Number

117

### Revision

1

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### Special Use

- Developer tools

### Contact

- Alon Or-bach [GitHub]alonorbach

### Other Extension Metadata

#### Last Modified Date

2019-10-08

#### IP Status

No known IP claims.

#### Contributors

- Jesse Barker, Unity Technologies
- Kenneth Benzie, Codeplay
- Jan-Harald Fredriksen, ARM
- Jeff Leger, Qualcomm
- Jesse Hall, Google
- Tobias Hector, AMD
- Neil Henning, Codeplay
- Baldur Karlsson
- Lionel Landwerlin, Intel
- Peter Lohrmann, AMD
- Alon Or-bach, Samsung
- Daniel Rakos, AMD
- Niklas Smedberg, Unity Technologies
- Igor Ostrowski, Intel

#### Description

The `VK_KHR_performance_query` extension adds a mechanism to allow querying of performance counters for use in applications and by profiling tools.

Each queue family **may** expose counters that **can** be enabled on a queue of that family. We extend VkQueryType to add a new query type for performance queries, and chain a structure on VkQueryPoolCreateInfo to specify the performance queries to enable.

#### New Commands

- vkAcquireProfilingLockKHR
- vkEnumeratePhysicalDeviceQueueFamilyPerformanceQueryCountersKHR
- vkGetPhysicalDeviceQueueFamilyPerformanceQueryPassesKHR
- vkReleaseProfilingLockKHR

#### New Structures

- VkAcquireProfilingLockInfoKHR

- VkPerformanceCounterDescriptionKHR
- VkPerformanceCounterKHR
- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDevicePerformanceQueryFeaturesKHR
- Extending VkPhysicalDeviceProperties2:
  - VkPhysicalDevicePerformanceQueryPropertiesKHR
- Extending VkQueryPoolCreateInfo:
  - VkQueryPoolPerformanceCreateInfoKHR
- Extending VkSubmitInfo, VkSubmitInfo2:
  - VkPerformanceQuerySubmitInfoKHR

### New Unions

- VkPerformanceCounterResultKHR

### New Enums

- VkAcquireProfilingLockFlagBitsKHR
- VkPerformanceCounterDescriptionFlagBitsKHR
- VkPerformanceCounterScopeKHR
- VkPerformanceCounterStorageKHR
- VkPerformanceCounterUnitKHR

### New Bitmasks

- VkAcquireProfilingLockFlagsKHR
- VkPerformanceCounterDescriptionFlagsKHR

### New Enum Constants

- `VK_KHR_PERFORMANCE_QUERY_EXTENSION_NAME`
- `VK_KHR_PERFORMANCE_QUERY_SPEC_VERSION`
- Extending VkQueryType:
  - `VK_QUERY_TYPE_PERFORMANCE_QUERY_KHR`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_ACQUIRE_PROFILING_LOCK_INFO_KHR`
  - `VK_STRUCTURE_TYPE_PERFORMANCE_COUNTER_DESCRIPTION_KHR`
  - `VK_STRUCTURE_TYPE_PERFORMANCE_COUNTER_KHR`
  - `VK_STRUCTURE_TYPE_PERFORMANCE_QUERY_SUBMIT_INFO_KHR`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PERFORMANCE_QUERY_FEATURES_KHR`

- VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PERFORMANCE_QUERY_PROPERTIES_KHR
- VK_STRUCTURE_TYPE_QUERY_POOL_PERFORMANCE_CREATE_INFO_KHR

**Issues**

1) Should this extension include a mechanism to begin a query in command buffer *A* and end the query in command buffer *B*?

**RESOLVED** No - queries are tied to command buffer creation and thus have to be encapsulated within a single command buffer.

2) Should this extension include a mechanism to begin and end queries globally on the queue, not using the existing command buffer commands?

**RESOLVED** No - for the same reasoning as the resolution of 1).

3) Should this extension expose counters that require multiple passes?

**RESOLVED** Yes - users should re-submit a command buffer with the same commands in it multiple times, specifying the pass to count as the query parameter in VkPerformanceQuerySubmitInfoKHR.

4) How to handle counters across parallel workloads?

**RESOLVED** In the spirit of Vulkan, a counter description flag VK_PERFORMANCE_COUNTER_DESCRIPTION_CONCURRENTLY_IMPACTED_BIT_KHR denotes that the accuracy of a counter result is affected by parallel workloads.

5) How to handle secondary command buffers?

**RESOLVED** Secondary command buffers inherit any counter pass index specified in the parent primary command buffer. Note: this is no longer an issue after change from issue 10 resolution

6) What commands does the profiling lock have to be held for?

**RESOLVED** For any command buffer that is being queried with a performance query pool, the profiling lock **must** be held while that command buffer is in the *recording*, *executable*, or *pending state*.

7) Should we support vkCmdCopyQueryPoolResults?

**RESOLVED** Yes.

8) Should we allow performance queries to interact with multiview?

**RESOLVED** Yes, but the performance queries must be performed once for each pass per view.

9) Should a queryCount > 1 be usable for performance queries?

**RESOLVED** Yes. Some vendors will have costly performance counter query pool creation, and would rather if a certain set of counters were to be used multiple times that a queryCount > 1 can be used to amortize the instantiation cost.

10) Should we introduce an indirect mechanism to set the counter pass index?

**RESOLVED** Specify the counter pass index at submit time instead, to avoid requiring re-recording of command buffers when multiple counter passes are needed.

### Examples

The following example shows how to find what performance counters a queue family supports, setup a query pool to record these performance counters, how to add the query pool to the command buffer to record information, and how to get the results from the query pool.

```
// A previously created physical device
VkPhysicalDevice physicalDevice;

// One of the queue families our device supports
uint32_t queueFamilyIndex;

uint32_t counterCount;

// Get the count of counters supported
vkEnumeratePhysicalDeviceQueueFamilyPerformanceQueryCountersKHR(
  physicalDevice,
  queueFamilyIndex,
  &counterCount,
  NULL,
  NULL);

VkPerformanceCounterKHR* counters =
  malloc(sizeof(VkPerformanceCounterKHR) * counterCount);
VkPerformanceCounterDescriptionKHR* counterDescriptions =
  malloc(sizeof(VkPerformanceCounterDescriptionKHR) * counterCount);

// Get the counters supported
vkEnumeratePhysicalDeviceQueueFamilyPerformanceQueryCountersKHR(
  physicalDevice,
  queueFamilyIndex,
  &counterCount,
  counters,
  counterDescriptions);

// Try to enable the first 8 counters
uint32_t enabledCounters[8];

const uint32_t enabledCounterCount = min(counterCount, 8));

for (uint32_t i = 0; i < enabledCounterCount; i++) {
  enabledCounters[i] = i;
}

// A previously created device that had the performanceCounterQueryPools feature
```

```
// set to VK_TRUE
VkDevice device;

VkQueryPoolPerformanceCreateInfoKHR performanceQueryCreateInfo = {
  .sType = VK_STRUCTURE_TYPE_QUERY_POOL_PERFORMANCE_CREATE_INFO_KHR,
  .pNext = NULL,

  // Specify the queue family that this performance query is performed on
  .queueFamilyIndex = queueFamilyIndex,

  // The number of counters to enable
  .counterIndexCount = enabledCounterCount,

  // The array of indices of counters to enable
  .pCounterIndices = enabledCounters
};


// Get the number of passes our counters will require.
uint32_t numPasses;

vkGetPhysicalDeviceQueueFamilyPerformanceQueryPassesKHR(
  physicalDevice,
  &performanceQueryCreateInfo,
  &numPasses);

VkQueryPoolCreateInfo queryPoolCreateInfo = {
  .sType = VK_STRUCTURE_TYPE_QUERY_POOL_CREATE_INFO,
  .pNext = &performanceQueryCreateInfo,
  .flags = 0,
  // Using our new query type here
  .queryType = VK_QUERY_TYPE_PERFORMANCE_QUERY_KHR,
  .queryCount = 1,
  .pipelineStatistics = 0
};

VkQueryPool queryPool;

VkResult result = vkCreateQueryPool(
  device,
  &queryPoolCreateInfo,
  NULL,
  &queryPool);

assert(VK_SUCCESS == result);

// A queue from queueFamilyIndex
VkQueue queue;

// A command buffer we want to record counters on
VkCommandBuffer commandBuffer;
```

```
VkCommandBufferBeginInfo commandBufferBeginInfo = {
  .sType = VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO,
  .pNext = NULL,
  .flags = 0,
  .pInheritanceInfo = NULL
};

VkAcquireProfilingLockInfoKHR lockInfo = {
  .sType = VK_STRUCTURE_TYPE_ACQUIRE_PROFILING_LOCK_INFO_KHR,
  .pNext = NULL,
  .flags = 0,
  .timeout = UINT64_MAX // Wait forever for the lock
};

// Acquire the profiling lock before we record command buffers
// that will use performance queries

result = vkAcquireProfilingLockKHR(device, &lockInfo);

assert(VK_SUCCESS == result);

result = vkBeginCommandBuffer(commandBuffer, &commandBufferBeginInfo);

assert(VK_SUCCESS == result);

vkCmdResetQueryPool(
  commandBuffer,
  queryPool,
  0,
  1);

vkCmdBeginQuery(
  commandBuffer,
  queryPool,
  0,
  0);

// Perform the commands you want to get performance information on
// ...

// Perform a barrier to ensure all previous commands were complete before
// ending the query
vkCmdPipelineBarrier(commandBuffer,
  VK_PIPELINE_STAGE_BOTTOM_OF_PIPE_BIT,
  VK_PIPELINE_STAGE_BOTTOM_OF_PIPE_BIT,
  0,
  0,
  NULL,
  0,
  NULL,
```

```
  0,
  NULL);

vkCmdEndQuery(
  commandBuffer,
  queryPool,
  0);

result = vkEndCommandBuffer(commandBuffer);

assert(VK_SUCCESS == result);

for (uint32_t counterPass = 0; counterPass < numPasses; counterPass++) {

  VkPerformanceQuerySubmitInfoKHR performanceQuerySubmitInfo = {
    VK_STRUCTURE_TYPE_PERFORMANCE_QUERY_SUBMIT_INFO_KHR,
    NULL,
    counterPass
  };


  // Submit the command buffer and wait for its completion
  // ...
}

// Release the profiling lock after the command buffer is no longer in the
// pending state.
vkReleaseProfilingLockKHR(device);

result = vkResetCommandBuffer(commandBuffer, 0);

assert(VK_SUCCESS == result);

// Create an array to hold the results of all counters
VkPerformanceCounterResultKHR* recordedCounters = malloc(
  sizeof(VkPerformanceCounterResultKHR) * enabledCounterCount);

result = vkGetQueryPoolResults(
  device,
  queryPool,
  0,
  1,
  sizeof(VkPerformanceCounterResultKHR) * enabledCounterCount,
  recordedCounters,
  sizeof(VkPerformanceCounterResultKHR) * enabledCounterCount,
  NULL);

// recordedCounters is filled with our counters, we will look at one for posterity
switch (counters[0].storage) {
  case VK_PERFORMANCE_COUNTER_STORAGE_INT32:
    // use recordCounters[0].int32 to get at the counter result!
```

```
    break;
  case VK_PERFORMANCE_COUNTER_STORAGE_INT64:
    // use recordCounters[0].int64 to get at the counter result!
    break;
  case VK_PERFORMANCE_COUNTER_STORAGE_UINT32:
    // use recordCounters[0].uint32 to get at the counter result!
    break;
  case VK_PERFORMANCE_COUNTER_STORAGE_UINT64:
    // use recordCounters[0].uint64 to get at the counter result!
    break;
  case VK_PERFORMANCE_COUNTER_STORAGE_FLOAT32:
    // use recordCounters[0].float32 to get at the counter result!
    break;
  case VK_PERFORMANCE_COUNTER_STORAGE_FLOAT64:
    // use recordCounters[0].float64 to get at the counter result!
    break;
}
```

### Version History

- Revision 1, 2019-10-08

## VK_KHR_pipeline_binary

### Name String

`VK_KHR_pipeline_binary`

### Extension Type

Device extension

### Registered Extension Number

484

### Revision

1

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_maintenance5

### Contact

- Stu Smith [GitHub]stu-s

### Extension Proposal

VK_KHR_pipeline_binary

### Other Extension Metadata

#### Last Modified Date

2024-07-01

#### Contributors

- Stu Smith, AMD
- Tobias Hector, AMD
- Alan Harrison, AMD
- Maciej Jesionowski, AMD
- Younggwan Kim, Arm
- Jan-Harald Fredriksen, Arm
- Ting Wei, Arm
- Chris Glover, Google
- Shahbaz Youssefi, Google
- Jakub Kuderski, Google
- Piotr Byszewski, Mobica
- Piers Daniell, NVIDIA
- Ralph Potter, Samsung
- Matthew Netsch, Qualcomm
- Hans-Kristian Arntzen, Valve
- Samuel Pitoiset, Valve
- Tatsuyuki Ishi, Valve

#### Description

This extension provides a method to obtain binary data associated with individual pipelines such that applications can manage caching themselves instead of using VkPipelineCache objects.

#### New Object Types

- VkPipelineBinaryKHR

#### New Commands

- vkCreatePipelineBinariesKHR
- vkDestroyPipelineBinaryKHR
- vkGetPipelineBinaryDataKHR
- vkGetPipelineKeyKHR
- vkReleaseCapturedPipelineDataKHR

### New Structures

- VkPipelineBinaryCreateInfoKHR
- VkPipelineBinaryDataInfoKHR
- VkPipelineBinaryDataKHR
- VkPipelineBinaryHandlesInfoKHR
- VkPipelineBinaryKeyKHR
- VkPipelineBinaryKeysAndDataKHR
- VkPipelineCreateInfoKHR
- VkReleaseCapturedPipelineDataInfoKHR
- Extending VkDeviceCreateInfo:
  - VkDevicePipelineBinaryInternalCacheControlKHR
- Extending VkGraphicsPipelineCreateInfo, VkComputePipelineCreateInfo, VkRayTracingPipelineCreateInfoKHR:
  - VkPipelineBinaryInfoKHR
- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDevicePipelineBinaryFeaturesKHR
- Extending VkPhysicalDeviceProperties2:
  - VkPhysicalDevicePipelineBinaryPropertiesKHR

### New Enum Constants

- `VK_KHR_PIPELINE_BINARY_EXTENSION_NAME`
- `VK_KHR_PIPELINE_BINARY_SPEC_VERSION`
- `VK_MAX_PIPELINE_BINARY_KEY_SIZE_KHR`
- Extending VkObjectType:
  - `VK_OBJECT_TYPE_PIPELINE_BINARY_KHR`
- Extending VkPipelineCreateFlagBits2KHR:
  - `VK_PIPELINE_CREATE_2_CAPTURE_DATA_BIT_KHR`
- Extending VkResult:
  - `VK_ERROR_NOT_ENOUGH_SPACE_KHR`
  - `VK_PIPELINE_BINARY_MISSING_KHR`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_DEVICE_PIPELINE_BINARY_INTERNAL_CACHE_CONTROL_KHR`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PIPELINE_BINARY_FEATURES_KHR`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PIPELINE_BINARY_PROPERTIES_KHR`
  - `VK_STRUCTURE_TYPE_PIPELINE_BINARY_CREATE_INFO_KHR`

- VK_STRUCTURE_TYPE_PIPELINE_BINARY_DATA_INFO_KHR
- VK_STRUCTURE_TYPE_PIPELINE_BINARY_HANDLES_INFO_KHR
- VK_STRUCTURE_TYPE_PIPELINE_BINARY_INFO_KHR
- VK_STRUCTURE_TYPE_PIPELINE_BINARY_KEY_KHR
- VK_STRUCTURE_TYPE_PIPELINE_CREATE_INFO_KHR
- VK_STRUCTURE_TYPE_RELEASE_CAPTURED_PIPELINE_DATA_INFO_KHR

### Version History

- Revision 1, 2021-12-10 (Chris Glover)
  - Initial draft.

## VK_KHR_pipeline_executable_properties

### Name String

VK_KHR_pipeline_executable_properties

### Extension Type

Device extension

### Registered Extension Number

270

### Revision

1

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### Special Use

- Developer tools

### Contact

- Faith Ekstrand [GitHub]gfxstrand

### Other Extension Metadata

### Last Modified Date

2019-05-28

**IP Status**

No known IP claims.

**Interactions and External Dependencies**

**Contributors**

- Faith Ekstrand, Intel
- Ian Romanick, Intel
- Kenneth Graunke, Intel
- Baldur Karlsson, Valve
- Jesse Hall, Google
- Jeff Bolz, Nvidia
- Piers Daniel, Nvidia
- Tobias Hector, AMD
- Jan-Harald Fredriksen, ARM
- Tom Olson, ARM
- Daniel Koch, Nvidia
- Spencer Fricke, Samsung

**Description**

When a pipeline is created, its state and shaders are compiled into zero or more device-specific executables, which are used when executing commands against that pipeline. This extension adds a mechanism to query properties and statistics about the different executables produced by the pipeline compilation process. This is intended to be used by debugging and performance tools to allow them to provide more detailed information to the user. Certain compile time shader statistics provided through this extension may be useful to developers for debugging or performance analysis.

**New Commands**

- vkGetPipelineExecutableInternalRepresentationsKHR
- vkGetPipelineExecutablePropertiesKHR
- vkGetPipelineExecutableStatisticsKHR

**New Structures**

- VkPipelineExecutableInfoKHR
- VkPipelineExecutableInternalRepresentationKHR
- VkPipelineExecutablePropertiesKHR
- VkPipelineExecutableStatisticKHR
- VkPipelineInfoKHR

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDevicePipelineExecutablePropertiesFeaturesKHR

#### New Unions

- VkPipelineExecutableStatisticValueKHR

#### New Enums

- VkPipelineExecutableStatisticFormatKHR

#### New Enum Constants

- `VK_KHR_PIPELINE_EXECUTABLE_PROPERTIES_EXTENSION_NAME`
- `VK_KHR_PIPELINE_EXECUTABLE_PROPERTIES_SPEC_VERSION`
- Extending VkPipelineCreateFlagBits:
  - `VK_PIPELINE_CREATE_CAPTURE_INTERNAL_REPRESENTATIONS_BIT_KHR`
  - `VK_PIPELINE_CREATE_CAPTURE_STATISTICS_BIT_KHR`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PIPELINE_EXECUTABLE_PROPERTIES_FEATURES_KHR`
  - `VK_STRUCTURE_TYPE_PIPELINE_EXECUTABLE_INFO_KHR`
  - `VK_STRUCTURE_TYPE_PIPELINE_EXECUTABLE_INTERNAL_REPRESENTATION_KHR`
  - `VK_STRUCTURE_TYPE_PIPELINE_EXECUTABLE_PROPERTIES_KHR`
  - `VK_STRUCTURE_TYPE_PIPELINE_EXECUTABLE_STATISTIC_KHR`
  - `VK_STRUCTURE_TYPE_PIPELINE_INFO_KHR`

#### Issues

1) What should we call the pieces of the pipeline which are produced by the compilation process and about which you can query properties and statistics?

**RESOLVED**: Call them “executables”. The name “binary” was used in early drafts of the extension but it was determined that “pipeline binary” could have a fairly broad meaning (such as a binary serialized form of an entire pipeline) and was too big of a namespace for the very specific needs of this extension.

#### Version History

- Revision 1, 2019-05-28 (Faith Ekstrand)
  - Initial draft

## VK_KHR_pipeline_library

### Name String

`VK_KHR_pipeline_library`

### Extension Type

Device extension

### Registered Extension Number

291

### Revision

1

### Ratification Status

Ratified

### Extension and Version Dependencies

None

### Contact

- Christoph Kubisch [GitHub]pixeljetstream

### Other Extension Metadata

### Last Modified Date

2020-01-08

### IP Status

No known IP claims.

### Contributors

- See contributors to `VK_KHR_ray_tracing_pipeline`

### Description

A pipeline library is a special pipeline that cannot be bound, instead it defines a set of shaders and shader groups which can be linked into other pipelines. This extension defines the infrastructure for pipeline libraries, but does not specify the creation or usage of pipeline libraries. This is left to additional dependent extensions.

### New Structures

- Extending VkGraphicsPipelineCreateInfo:
  - VkPipelineLibraryCreateInfoKHR

### New Enum Constants

- `VK_KHR_PIPELINE_LIBRARY_EXTENSION_NAME`
- `VK_KHR_PIPELINE_LIBRARY_SPEC_VERSION`

- Extending VkPipelineCreateFlagBits:
  - `VK_PIPELINE_CREATE_LIBRARY_BIT_KHR`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PIPELINE_LIBRARY_CREATE_INFO_KHR`

#### Version History

- Revision 1, 2020-01-08 (Christoph Kubisch)
  - Initial draft.

## VK_KHR_portability_enumeration

#### Name String

`VK_KHR_portability_enumeration`

#### Extension Type

Instance extension

#### Registered Extension Number

395

#### Revision

1

#### Ratification Status

Ratified

#### Extension and Version Dependencies

None

#### Contact

- Charles Giessen [GitHub]charles-lunarg

#### Other Extension Metadata

#### Last Modified Date

2021-06-02

#### IP Status

No known IP claims.

#### Interactions and External Dependencies

- Interacts with `VK_KHR_portability_subset`

#### Contributors

- Lenny Komow, LunarG
- Charles Giessen, LunarG

### Description

This extension allows applications to control whether devices that expose the `VK_KHR_portability_subset` extension are included in the results of physical device enumeration. Since devices which support the `VK_KHR_portability_subset` extension are not fully conformant Vulkan implementations, the Vulkan loader does not report those devices unless the application explicitly asks for them. This prevents applications which may not be aware of non-conformant devices from accidentally using them, as any device which supports the `VK_KHR_portability_subset` extension mandates that the extension must be enabled if that device is used.

This extension is implemented in the loader.

### New Enum Constants

- `VK_KHR_PORTABILITY_ENUMERATION_EXTENSION_NAME`
- `VK_KHR_PORTABILITY_ENUMERATION_SPEC_VERSION`
- Extending VkInstanceCreateFlagBits:
  - `VK_INSTANCE_CREATE_ENUMERATE_PORTABILITY_BIT_KHR`

### Version History

- Revision 1, 2021-06-02 (Lenny Komow)
  - Initial version

## VK_KHR_present_id

### Name String

`VK_KHR_present_id`

### Extension Type

Device extension

### Registered Extension Number

295

### Revision

1

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_swapchain
and
VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### Contact

- Keith Packard [GitHub]keithp

### Other Extension Metadata

### Last Modified Date

2019-05-15

### IP Status

No known IP claims.

### Contributors

- Keith Packard, Valve
- Ian Elliott, Google
- Alon Or-bach, Samsung

### Description

This device extension allows an application that uses the `VK_KHR_swapchain` extension to provide an identifier for present operations on a swapchain. An application **can** use this to reference specific present operations in other extensions.

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDevicePresentIdFeaturesKHR
- Extending VkPresentInfoKHR:
  - VkPresentIdKHR

### New Enum Constants

- `VK_KHR_PRESENT_ID_EXTENSION_NAME`
- `VK_KHR_PRESENT_ID_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PRESENT_ID_FEATURES_KHR`
  - `VK_STRUCTURE_TYPE_PRESENT_ID_KHR`

### Issues

None.

### Examples

### Version History

- Revision 1, 2019-05-15 (Keith Packard)

- Initial version

## VK_KHR_present_wait

### Name String

`VK_KHR_present_wait`

### Extension Type

Device extension

### Registered Extension Number

249

### Revision

1

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_swapchain
and
VK_KHR_present_id

### Contact

- Keith Packard [GitHub]keithp

### Other Extension Metadata

### Last Modified Date

2019-05-15

### IP Status

No known IP claims.

### Contributors

- Keith Packard, Valve
- Ian Elliott, Google
- Tobias Hector, AMD
- Daniel Stone, Collabora

### Description

This device extension allows an application that uses the `VK_KHR_swapchain` extension to wait for present operations to complete. An application can use this to monitor and control the pacing of the application by managing the number of outstanding images yet to be presented.

### New Commands

- vkWaitForPresentKHR

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDevicePresentWaitFeaturesKHR

### New Enum Constants

- `VK_KHR_PRESENT_WAIT_EXTENSION_NAME`
- `VK_KHR_PRESENT_WAIT_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PRESENT_WAIT_FEATURES_KHR`

### Issues

1) When does the wait finish?

**RESOLVED**. The wait will finish when the present is visible to the user. There is no requirement for any precise timing relationship between the presentation of the image to the user, but implementations **should** signal the wait as close as possible to the presentation of the first pixel in the new image to the user.

2) Should this use fences or other existing synchronization mechanism.

**RESOLVED**. Because display and rendering are often implemented in separate drivers, this extension will provide a separate synchronization API.

3) Should this extension share present identification with other extensions?

**RESOLVED**. Yes. A new extension, VK_KHR_present_id, should be created to provide a shared structure for presentation identifiers.

4) What happens when presentations complete out of order wrt calls to vkQueuePresent? This could happen if the semaphores for the presentations were ready out of order.

**OPTION A**: Require that when a PresentId is set that the driver ensure that images are always presented in the order of calls to vkQueuePresent.

**OPTION B**: Finish both waits when the earliest present completes. This will complete the later present wait earlier than the actual presentation. This should be the easiest to implement as the driver need only track the largest present ID completed. This is also the 'natural' consequence of interpreting the existing vkWaitForPresentKHR specificationn.

**OPTION C**: Finish both waits when both have completed. This will complete the earlier presentation later than the actual presentation time. This is allowed by the current specification as there is no precise timing requirement for when the presentId value is updated. This requires slightly more complexity in the driver as it will need to track all outstanding presentId values.

### Examples

### Version History

- Revision 1, 2019-02-19 (Keith Packard)
  - Initial version

## VK_KHR_push_descriptor

### Name String

`VK_KHR_push_descriptor`

### Extension Type

Device extension

### Registered Extension Number

81

### Revision

2

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### API Interactions

- Interacts with VK_VERSION_1_1
- Interacts with VK_KHR_descriptor_update_template

### Contact

- Jeff Bolz [GitHub]jeffbolznv

### Other Extension Metadata

### Last Modified Date

2017-09-12

### IP Status

No known IP claims.

### Contributors

- Jeff Bolz, NVIDIA
- Michael Worcester, Imagination Technologies

### Description

This extension allows descriptors to be written into the command buffer, while the implementation is responsible for managing their memory. Push descriptors may enable easier porting from older APIs and in some cases can be more efficient than writing descriptors into descriptor sets.

### New Commands

- vkCmdPushDescriptorSetKHR

If Vulkan Version 1.1 or VK_KHR_descriptor_update_template is supported:

- vkCmdPushDescriptorSetWithTemplateKHR

### New Structures

- Extending VkPhysicalDeviceProperties2:
  - VkPhysicalDevicePushDescriptorPropertiesKHR

### New Enum Constants

- `VK_KHR_PUSH_DESCRIPTOR_EXTENSION_NAME`
- `VK_KHR_PUSH_DESCRIPTOR_SPEC_VERSION`
- Extending VkDescriptorSetLayoutCreateFlagBits:
  - `VK_DESCRIPTOR_SET_LAYOUT_CREATE_PUSH_DESCRIPTOR_BIT_KHR`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PUSH_DESCRIPTOR_PROPERTIES_KHR`

If Vulkan Version 1.1 or VK_KHR_descriptor_update_template is supported:

- Extending VkDescriptorUpdateTemplateType:
  - `VK_DESCRIPTOR_UPDATE_TEMPLATE_TYPE_PUSH_DESCRIPTORS_KHR`

### Version History

- Revision 1, 2016-10-15 (Jeff Bolz)
  - Internal revisions
- Revision 2, 2017-09-12 (Tobias Hector)
  - Added interactions with Vulkan 1.1

## VK_KHR_ray_query

### Name String

`VK_KHR_ray_query`

**Extension Type**

Device extension

**Registered Extension Number**

349

**Revision**

1

**Ratification Status**

Ratified

**Extension and Version Dependencies**

VK_KHR_spirv_1_4
and
VK_KHR_acceleration_structure

**SPIR-V Dependencies**

- SPV_KHR_ray_query

**Contact**

- Daniel Koch [GitHub]dgkoch

**Other Extension Metadata**

**Last Modified Date**

2020-11-12

**Interactions and External Dependencies**

- This extension provides API support for `GLSL_EXT_ray_query`

**Contributors**

- Matthäus Chajdas, AMD
- Greg Grebe, AMD
- Nicolai Hähnle, AMD
- Tobias Hector, AMD
- Dave Oldcorn, AMD
- Skyler Saleh, AMD
- Mathieu Robart, Arm
- Marius Bjorge, Arm
- Tom Olson, Arm
- Sebastian Tafuri, EA
- Henrik Rydgard, Embark
- Juan Cañada, Epic Games

- Patrick Kelly, Epic Games
- Yuriy O'Donnell, Epic Games
- Michael Doggett, Facebook/Oculus
- Andrew Garrard, Imagination
- Don Scorgie, Imagination
- Dae Kim, Imagination
- Joshua Barczak, Intel
- Slawek Grajewski, Intel
- Jeff Bolz, NVIDIA
- Pascal Gautron, NVIDIA
- Daniel Koch, NVIDIA
- Christoph Kubisch, NVIDIA
- Ashwin Lele, NVIDIA
- Robert Stepinski, NVIDIA
- Martin Stich, NVIDIA
- Nuno Subtil, NVIDIA
- Eric Werness, NVIDIA
- Jon Leech, Khronos
- Jeroen van Schijndel, OTOY
- Juul Joosten, OTOY
- Alex Bourd, Qualcomm
- Roman Larionov, Qualcomm
- David McAllister, Qualcomm
- Spencer Fricke, Samsung
- Lewis Gordon, Samsung
- Ralph Potter, Samsung
- Jasper Bekkers, Traverse Research
- Jesse Barker, Unity
- Baldur Karlsson, Valve

**Description**

Rasterization has been the dominant method to produce interactive graphics, but increasing performance of graphics hardware has made ray tracing a viable option for interactive rendering. Being able to integrate ray tracing with traditional rasterization makes it easier for applications to incrementally add ray traced effects to existing applications or to do hybrid approaches with rasterization for primary visibility and ray tracing for secondary queries.

Ray queries are available to all shader types, including graphics, compute and ray tracing pipelines. Ray queries are not able to launch additional shaders, instead returning traversal results to the calling shader.

This extension adds support for the following SPIR-V extension in Vulkan:

- `SPV_KHR_ray_query`

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceRayQueryFeaturesKHR

### New Enum Constants

- `VK_KHR_RAY_QUERY_EXTENSION_NAME`
- `VK_KHR_RAY_QUERY_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_RAY_QUERY_FEATURES_KHR`

### New SPIR-V Capabilities

- `RayQueryKHR`
- `RayTraversalPrimitiveCullingKHR`

### Sample Code

Example of ray query in a GLSL shader, illustrating how to use ray queries to determine whether a given position (at ray origin) is in shadow or not, by tracing a ray towards the light, and checking for any intersections with geometry occluding the light.

```
rayQueryEXT rq;

rayQueryInitializeEXT(rq, accStruct, gl_RayFlagsTerminateOnFirstHitEXT, cullMask,
origin, tMin, direction, tMax);

// Traverse the acceleration structure and store information about the first
intersection (if any)
rayQueryProceedEXT(rq);

if (rayQueryGetIntersectionTypeEXT(rq, true) ==
gl_RayQueryCommittedIntersectionNoneEXT) {
    // Not in shadow
}
```

### Issues

(1) What are the changes between the public provisional (VK_KHR_ray_tracing v8) release and the final (VK_KHR_acceleration_structure v11 / VK_KHR_ray_query v1) release?

- refactor VK_KHR_ray_tracing into 3 extensions, enabling implementation flexibility and decoupling ray query support from ray pipelines:
  - `VK_KHR_acceleration_structure` (for acceleration structure operations)
  - `VK_KHR_ray_tracing_pipeline` (for ray tracing pipeline and shader stages)
  - `VK_KHR_ray_query` (for ray queries in existing shader stages)
- Update SPIRV capabilities to use `RayQueryKHR`
- extension is no longer provisional

### Version History

- Revision 1, 2020-11-12 (Mathieu Robart, Daniel Koch, Andrew Garrard)
  - Decomposition of the specification, from VK_KHR_ray_tracing to VK_KHR_ray_query (#1918,!3912)
  - update to use `RayQueryKHR` SPIR-V capability
  - add numerical limits for ray parameters (#2235,!3960)
  - relax formula for ray intersection candidate determination (#2322,!4080)
  - restrict traces to TLAS (#2239,!4141)
  - require `HitT` to be in ray interval for `OpRayQueryGenerateIntersectionKHR` (#2359,!4146)
  - add ray query shader stages for AS read bit (#2407,!4203)

## VK_KHR_ray_tracing_maintenance1

### Name String

`VK_KHR_ray_tracing_maintenance1`

### Extension Type

Device extension

### Registered Extension Number

387

### Revision

1

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_acceleration_structure

**API Interactions**

- Interacts with VK_VERSION_1_3
- Interacts with VK_KHR_ray_tracing_pipeline
- Interacts with VK_KHR_synchronization2

**SPIR-V Dependencies**

- SPV_KHR_ray_cull_mask

**Contact**

- Daniel Koch [GitHub]dgkoch

### Other Extension Metadata

**Last Modified Date**

2022-02-21

**Interactions and External Dependencies**

- This extension provides API support for `GLSL_EXT_ray_cull_mask`
- Interacts with `VK_KHR_ray_tracing_pipeline`
- Interacts with `VK_KHR_synchronization2`

**Contributors**

- Stu Smith, AMD
- Tobias Hector, AMD
- Marius Bjorge, Arm
- Tom Olson, Arm
- Yuriy O'Donnell, Epic Games
- Yunpeng Zhu, Huawei
- Andrew Garrard, Imagination
- Dae Kim, Imagination
- Joshua Barczak, Intel
- Lionel Landwerlin, Intel
- Daniel Koch, NVIDIA
- Eric Werness, NVIDIA
- Spencer Fricke, Samsung

### Description

`VK_KHR_ray_tracing_maintenance1` adds a collection of minor ray tracing features, none of which would warrant an entire extension of their own.

The new features are as follows:

- Adds support for the `SPV_KHR_ray_cull_mask` SPIR-V extension in Vulkan. This extension provides

access to built-in `CullMaskKHR` shader variable which contains the value of the `OpTrace* Cull Mask` parameter. This new shader variable is accessible in the intersection, any-hit, closest hit and miss shader stages.

- Adds support for a new pipeline stage and access mask built on top of `VK_KHR_synchronization2`:
  - `VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_COPY_BIT_KHR` to specify execution of acceleration structure copy commands
  - `VK_ACCESS_2_SHADER_BINDING_TABLE_READ_BIT_KHR` to specify read access to a shader binding table in any shader pipeline stage
- Adds two new acceleration structure query parameters:
  - `VK_QUERY_TYPE_ACCELERATION_STRUCTURE_SIZE_KHR` to query the acceleration structure size on the device timeline
  - `VK_QUERY_TYPE_ACCELERATION_STRUCTURE_SERIALIZATION_BOTTOM_LEVEL_POINTERS_KHR` to query the number of bottom level acceleration structure pointers for serialization
- Adds an optional new indirect ray tracing dispatch command, vkCmdTraceRaysIndirect2KHR, which sources the shader binding table parameters as well as the dispatch dimensions from the device. The `rayTracingPipelineTraceRaysIndirect2` feature indicates whether this functionality is supported.

#### New Commands

If VK_KHR_ray_tracing_pipeline is supported:

- vkCmdTraceRaysIndirect2KHR

#### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceRayTracingMaintenance1FeaturesKHR

If VK_KHR_ray_tracing_pipeline is supported:

- VkTraceRaysIndirectCommand2KHR

#### New Enum Constants

- `VK_KHR_RAY_TRACING_MAINTENANCE_1_EXTENSION_NAME`
- `VK_KHR_RAY_TRACING_MAINTENANCE_1_SPEC_VERSION`
- Extending VkQueryType:
  - `VK_QUERY_TYPE_ACCELERATION_STRUCTURE_SERIALIZATION_BOTTOM_LEVEL_POINTERS_KHR`
  - `VK_QUERY_TYPE_ACCELERATION_STRUCTURE_SIZE_KHR`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_RAY_TRACING_MAINTENANCE_1_FEATURES_KHR`

If VK_KHR_synchronization2 or Vulkan Version 1.3 and VK_KHR_ray_tracing_pipeline is supported:

- Extending VkAccessFlagBits2:
  - `VK_ACCESS_2_SHADER_BINDING_TABLE_READ_BIT_KHR`

If VK_KHR_synchronization2 or Vulkan Version 1.3 is supported:

- Extending VkPipelineStageFlagBits2:
  - `VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_COPY_BIT_KHR`

#### New Built-In Variables

- `CullMaskKHR`

#### New SPIR-V Capabilities

- `RayCullMaskKHR`

#### Issues

None Yet!

#### Version History

- Revision 1, 2022-02-21 (Members of the Vulkan Ray Tracing TSG)
  - internal revisions

### VK_KHR_ray_tracing_pipeline

#### Name String

`VK_KHR_ray_tracing_pipeline`

#### Extension Type

Device extension

#### Registered Extension Number

348

#### Revision

1

#### Ratification Status

Ratified

#### Extension and Version Dependencies

VK_KHR_spirv_1_4
and
VK_KHR_acceleration_structure

### API Interactions

- Interacts with VK_KHR_ray_query

### SPIR-V Dependencies

- SPV_KHR_ray_tracing

### Contact

- Daniel Koch [GitHub]dgkoch

## Other Extension Metadata

### Last Modified Date

2020-11-12

### Interactions and External Dependencies

- This extension provides API support for `GLSL_EXT_ray_tracing`
- This extension interacts with Vulkan 1.2 and `VK_KHR_vulkan_memory_model`, adding the shader-call-related relation of invocations, shader-call-order partial order of dynamic instances of instructions, and the `ShaderCallKHR` scope.
- This extension interacts with `VK_KHR_pipeline_library`, enabling pipeline libraries to be used with ray tracing pipelines and enabling usage of VkRayTracingPipelineInterfaceCreateInfoKHR.

### Contributors

- Matthäus Chajdas, AMD
- Greg Grebe, AMD
- Nicolai Hähnle, AMD
- Tobias Hector, AMD
- Dave Oldcorn, AMD
- Skyler Saleh, AMD
- Mathieu Robart, Arm
- Marius Bjorge, Arm
- Tom Olson, Arm
- Sebastian Tafuri, EA
- Henrik Rydgard, Embark
- Juan Cañada, Epic Games
- Patrick Kelly, Epic Games
- Yuriy O'Donnell, Epic Games
- Michael Doggett, Facebook/Oculus
- Andrew Garrard, Imagination
- Don Scorgie, Imagination
- Dae Kim, Imagination

- Joshua Barczak, Intel
- Slawek Grajewski, Intel
- Jeff Bolz, NVIDIA
- Pascal Gautron, NVIDIA
- Daniel Koch, NVIDIA
- Christoph Kubisch, NVIDIA
- Ashwin Lele, NVIDIA
- Robert Stepinski, NVIDIA
- Martin Stich, NVIDIA
- Nuno Subtil, NVIDIA
- Eric Werness, NVIDIA
- Jon Leech, Khronos
- Jeroen van Schijndel, OTOY
- Juul Joosten, OTOY
- Alex Bourd, Qualcomm
- Roman Larionov, Qualcomm
- David McAllister, Qualcomm
- Spencer Fricke, Samsung
- Lewis Gordon, Samsung
- Ralph Potter, Samsung
- Jasper Bekkers, Traverse Research
- Jesse Barker, Unity
- Baldur Karlsson, Valve

**Description**

Rasterization has been the dominant method to produce interactive graphics, but increasing performance of graphics hardware has made ray tracing a viable option for interactive rendering. Being able to integrate ray tracing with traditional rasterization makes it easier for applications to incrementally add ray traced effects to existing applications or to do hybrid approaches with rasterization for primary visibility and ray tracing for secondary queries.

To enable ray tracing, this extension adds a few different categories of new functionality:

- A new ray tracing pipeline type with new shader domains: ray generation, intersection, any-hit, closest hit, miss, and callable
- A shader binding indirection table to link shader groups with acceleration structure items
- Ray tracing commands which initiate the ray pipeline traversal and invocation of the various new shader domains depending on which traversal conditions are met

This extension adds support for the following SPIR-V extension in Vulkan:

- `SPV_KHR_ray_tracing`

### New Commands

- vkCmdSetRayTracingPipelineStackSizeKHR
- vkCmdTraceRaysIndirectKHR
- vkCmdTraceRaysKHR
- vkCreateRayTracingPipelinesKHR
- vkGetRayTracingCaptureReplayShaderGroupHandlesKHR
- vkGetRayTracingShaderGroupHandlesKHR
- vkGetRayTracingShaderGroupStackSizeKHR

### New Structures

- VkRayTracingPipelineCreateInfoKHR
- VkRayTracingPipelineInterfaceCreateInfoKHR
- VkRayTracingShaderGroupCreateInfoKHR
- VkStridedDeviceAddressRegionKHR
- VkTraceRaysIndirectCommandKHR
- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceRayTracingPipelineFeaturesKHR
- Extending VkPhysicalDeviceProperties2:
  - VkPhysicalDeviceRayTracingPipelinePropertiesKHR

### New Enums

- VkRayTracingShaderGroupTypeKHR
- VkShaderGroupShaderKHR

### New Enum Constants

- `VK_KHR_RAY_TRACING_PIPELINE_EXTENSION_NAME`
- `VK_KHR_RAY_TRACING_PIPELINE_SPEC_VERSION`
- `VK_SHADER_UNUSED_KHR`
- Extending VkBufferUsageFlagBits:
  - `VK_BUFFER_USAGE_SHADER_BINDING_TABLE_BIT_KHR`
- Extending VkDynamicState:
  - `VK_DYNAMIC_STATE_RAY_TRACING_PIPELINE_STACK_SIZE_KHR`
- Extending VkPipelineBindPoint:

    - VK_PIPELINE_BIND_POINT_RAY_TRACING_KHR
- Extending VkPipelineCreateFlagBits:
    - VK_PIPELINE_CREATE_RAY_TRACING_NO_NULL_ANY_HIT_SHADERS_BIT_KHR
    - VK_PIPELINE_CREATE_RAY_TRACING_NO_NULL_CLOSEST_HIT_SHADERS_BIT_KHR
    - VK_PIPELINE_CREATE_RAY_TRACING_NO_NULL_INTERSECTION_SHADERS_BIT_KHR
    - VK_PIPELINE_CREATE_RAY_TRACING_NO_NULL_MISS_SHADERS_BIT_KHR
    - VK_PIPELINE_CREATE_RAY_TRACING_SHADER_GROUP_HANDLE_CAPTURE_REPLAY_BIT_KHR
    - VK_PIPELINE_CREATE_RAY_TRACING_SKIP_AABBS_BIT_KHR
    - VK_PIPELINE_CREATE_RAY_TRACING_SKIP_TRIANGLES_BIT_KHR
- Extending VkPipelineStageFlagBits:
    - VK_PIPELINE_STAGE_RAY_TRACING_SHADER_BIT_KHR
- Extending VkShaderStageFlagBits:
    - VK_SHADER_STAGE_ANY_HIT_BIT_KHR
    - VK_SHADER_STAGE_CALLABLE_BIT_KHR
    - VK_SHADER_STAGE_CLOSEST_HIT_BIT_KHR
    - VK_SHADER_STAGE_INTERSECTION_BIT_KHR
    - VK_SHADER_STAGE_MISS_BIT_KHR
    - VK_SHADER_STAGE_RAYGEN_BIT_KHR
- Extending VkStructureType:
    - VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_RAY_TRACING_PIPELINE_FEATURES_KHR
    - VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_RAY_TRACING_PIPELINE_PROPERTIES_KHR
    - VK_STRUCTURE_TYPE_RAY_TRACING_PIPELINE_CREATE_INFO_KHR
    - VK_STRUCTURE_TYPE_RAY_TRACING_PIPELINE_INTERFACE_CREATE_INFO_KHR
    - VK_STRUCTURE_TYPE_RAY_TRACING_SHADER_GROUP_CREATE_INFO_KHR

### New or Modified Built-In Variables

- LaunchIdKHR
- LaunchSizeKHR
- WorldRayOriginKHR
- WorldRayDirectionKHR
- ObjectRayOriginKHR
- ObjectRayDirectionKHR
- RayTminKHR
- RayTmaxKHR
- InstanceCustomIndexKHR

- `InstanceId`
- `ObjectToWorldKHR`
- `WorldToObjectKHR`
- `HitKindKHR`
- `IncomingRayFlagsKHR`
- `RayGeometryIndexKHR`
- (modified)`PrimitiveId`

**New SPIR-V Capabilities**

- `RayTracingKHR`
- `RayTraversalPrimitiveCullingKHR`

**Issues**

(1) How does this extension differ from VK_NV_ray_tracing?

**DISCUSSION**:

The following is a summary of the main functional differences between VK_KHR_ray_tracing_pipeline and VK_NV_ray_tracing:

- added support for indirect ray tracing (vkCmdTraceRaysIndirectKHR)
- uses SPV_KHR_ray_tracing instead of SPV_NV_ray_tracing
  - refer to KHR SPIR-V enums instead of NV SPIR-V enums (which are functionally equivalent and aliased to the same values).
  - added `RayGeometryIndexKHR` built-in
- removed vkCompileDeferredNV compilation functionality and replaced with deferred host operations interactions for ray tracing
- added VkPhysicalDeviceRayTracingPipelineFeaturesKHR structure
- extended VkPhysicalDeviceRayTracingPipelinePropertiesKHR structure
  - renamed `maxRecursionDepth` to `maxRayRecursionDepth` and it has a minimum of 1 instead of 31
  - require `shaderGroupHandleSize` to be 32 bytes
  - added `maxRayDispatchInvocationCount`, `shaderGroupHandleAlignment` and `maxRayHitAttributeSize`
- reworked geometry structures so they could be better shared between device, host, and indirect builds
- changed SBT parameters to a structure and added size (VkStridedDeviceAddressRegionKHR)
- add parameter for requesting memory requirements for host and/or device build
- added pipeline library support for ray tracing
- added watertightness guarantees

- added no-null-shader pipeline flags (`VK_PIPELINE_CREATE_RAY_TRACING_NO_NULL_*_SHADERS_BIT_KHR`)
- added memory model interactions with ray tracing and define how subgroups work and can be repacked

(2) Can you give a more detailed comparison of differences and similarities between VK_NV_ray_tracing and VK_KHR_ray_tracing_pipeline?

**DISCUSSION**:

The following is a more detailed comparison of which commands, structures, and enums are aliased, changed, or removed.

- Aliased functionality — enums, structures, and commands that are considered equivalent:
  - VkRayTracingShaderGroupTypeNV  VkRayTracingShaderGroupTypeKHR
  - vkGetRayTracingShaderGroupHandlesNV  vkGetRayTracingShaderGroupHandlesKHR
- Changed enums, structures, and commands:
  - VkRayTracingShaderGroupCreateInfoNV → VkRayTracingShaderGroupCreateInfoKHR (added `pShaderGroupCaptureReplayHandle`)
  - VkRayTracingPipelineCreateInfoNV → VkRayTracingPipelineCreateInfoKHR (changed type of `pGroups`, added `libraries`, `pLibraryInterface`, and `pDynamicState`)
  - VkPhysicalDeviceRayTracingPropertiesNV → VkPhysicalDeviceRayTracingPropertiesKHR (renamed `maxTriangleCount` to `maxPrimitiveCount`, added `shaderGroupHandleCaptureReplaySize`)
  - vkCmdTraceRaysNV → vkCmdTraceRaysKHR (params to struct)
  - vkCreateRayTracingPipelinesNV → vkCreateRayTracingPipelinesKHR (different struct, changed functionality)
- Added enums, structures and commands:
  - `VK_PIPELINE_CREATE_RAY_TRACING_NO_NULL_ANY_HIT_SHADERS_BIT_KHR` `VK_PIPELINE_CREATE_RAY_TRACING_NO_NULL_CLOSEST_HIT_SHADERS_BIT_KHR`, `VK_PIPELINE_CREATE_RAY_TRACING_NO_NULL_MISS_SHADERS_BIT_KHR`, `VK_PIPELINE_CREATE_RAY_TRACING_NO_NULL_INTERSECTION_SHADERS_BIT_KHR`, `VK_PIPELINE_CREATE_RAY_TRACING_SKIP_TRIANGLES_BIT_KHR`, `VK_PIPELINE_CREATE_RAY_TRACING_SKIP_AABBS_BIT_KHR` to VkPipelineCreateFlagBits
  - VkPhysicalDeviceRayTracingPipelineFeaturesKHR structure
  - VkDeviceOrHostAddressKHR and VkDeviceOrHostAddressConstKHR unions
  - VkPipelineLibraryCreateInfoKHR struct
  - VkRayTracingPipelineInterfaceCreateInfoKHR struct
  - VkStridedDeviceAddressRegionKHR struct
  - vkCmdTraceRaysIndirectKHR command and VkTraceRaysIndirectCommandKHR struct
  - vkGetRayTracingCaptureReplayShaderGroupHandlesKHR (shader group capture/replay)
  - vkCmdSetRayTracingPipelineStackSizeKHR and vkGetRayTracingShaderGroupStackSizeKHR commands for stack size control

- Functionality removed:
  - VK_PIPELINE_CREATE_DEFER_COMPILE_BIT_NV
  - vkCompileDeferredNV command (replaced with VK_KHR_deferred_host_operations)

(3) What are the changes between the public provisional (VK_KHR_ray_tracing v8) release and the internal provisional (VK_KHR_ray_tracing v9) release?

- Require Vulkan 1.1 and SPIR-V 1.4
- Added interactions with Vulkan 1.2 and VK_KHR_vulkan_memory_model
- added creation time capture and replay flags
  - added VK_PIPELINE_CREATE_RAY_TRACING_SHADER_GROUP_HANDLE_CAPTURE_REPLAY_BIT_KHR to VkPipelineCreateFlagBits
- replace VkStridedBufferRegionKHR with VkStridedDeviceAddressRegionKHR and change vkCmdTraceRaysKHR, vkCmdTraceRaysIndirectKHR, to take these for the shader binding table and use device addresses instead of buffers.
- require the shader binding table buffers to have the VK_BUFFER_USAGE_RAY_TRACING_BIT_KHR set
- make VK_KHR_pipeline_library an interaction instead of required extension
- rename the libraries member of VkRayTracingPipelineCreateInfoKHR to pLibraryInfo and make it a pointer
- make VK_KHR_deferred_host_operations an interaction instead of a required extension (later went back on this)
- added explicit stack size management for ray tracing pipelines
  - removed the maxCallableSize member of VkRayTracingPipelineInterfaceCreateInfoKHR
  - added the pDynamicState member to VkRayTracingPipelineCreateInfoKHR
  - added VK_DYNAMIC_STATE_RAY_TRACING_PIPELINE_STACK_SIZE_KHR dynamic state for ray tracing pipelines
  - added vkGetRayTracingShaderGroupStackSizeKHR and vkCmdSetRayTracingPipelineStackSizeKHR commands
  - added VkShaderGroupShaderKHR enum
- Added maxRayDispatchInvocationCount limit to VkPhysicalDeviceRayTracingPipelinePropertiesKHR
- Added shaderGroupHandleAlignment property to VkPhysicalDeviceRayTracingPipelinePropertiesKHR
- Added maxRayHitAttributeSize property to VkPhysicalDeviceRayTracingPipelinePropertiesKHR
- Clarify deferred host ops for pipeline creation
  - VkDeferredOperationKHR is now a top-level parameter for vkCreateRayTracingPipelinesKHR
  - removed VkDeferredOperationInfoKHR structure
  - change deferred host creation/return parameter behavior such that the implementation can

modify such parameters until the deferred host operation completes

  - `VK_KHR_deferred_host_operations` is required again

(4) What are the changes between the internal provisional (VK_KHR_ray_tracing v9) release and the final (VK_KHR_acceleration_structure v11 / VK_KHR_ray_tracing_pipeline v1) release?

- refactor VK_KHR_ray_tracing into 3 extensions, enabling implementation flexibility and decoupling ray query support from ray pipelines:
  - `VK_KHR_acceleration_structure` (for acceleration structure operations)
  - `VK_KHR_ray_tracing_pipeline` (for ray tracing pipeline and shader stages)
  - `VK_KHR_ray_query` (for ray queries in existing shader stages)
- Require `Volatile` for the following builtins in the ray generation, closest hit, miss, intersection, and callable shader stages:
  - `SubgroupSize`, `SubgroupLocalInvocationId`, `SubgroupEqMask`, `SubgroupGeMask`, `SubgroupGtMask`, `SubgroupLeMask`, `SubgroupLtMask`
  - `SMIDNV`, `WarpIDNV`
- clarify buffer usage flags for ray tracing
  - `VK_BUFFER_USAGE_SHADER_BINDING_TABLE_BIT_KHR` is added as an alias of `VK_BUFFER_USAGE_RAY_TRACING_BIT_NV` and is required on shader binding table buffers
  - `VK_BUFFER_USAGE_STORAGE_BUFFER_BIT` is used in `VK_KHR_acceleration_structure` for `scratchData`
- rename `maxRecursionDepth` to `maxRayPipelineRecursionDepth` (pipeline creation) and `maxRayRecursionDepth` (limit) to reduce confusion
- Add queryable `maxRayHitAttributeSize` limit and rename members of VkRayTracingPipelineInterfaceCreateInfoKHR to `maxPipelineRayPayloadSize` and `maxPipelineRayHitAttributeSize` for clarity
- Update SPIRV capabilities to use `RayTracingKHR`
- extension is no longer provisional
- define synchronization requirements for indirect trace rays and indirect buffer

(5) This extension adds gl_InstanceID for the intersection, any-hit, and closest hit shaders, but in KHR_vulkan_glsl, gl_InstanceID is replaced with gl_InstanceIndex. Which should be used for Vulkan in this extension?

**RESOLVED**: This extension uses gl_InstanceID and maps it to `InstanceId` in SPIR-V. It is acknowledged that this is different than other shader stages in Vulkan. There are two main reasons for the difference here:

- symmetry with gl_PrimitiveID which is also available in these shaders
- there is no “baseInstance” relevant for these shaders, and so ID makes it more obvious that this is zero-based.

(6) Why is `VK_KHR_pipeline_library` an interaction instead of a required dependency, particularly when the “Feature Requirements” section says it is required to be supported anyhow?

**RESOLVED**: If `VK_KHR_pipeline_library` were a required extension dependency, then every application would need to enable the extension whether or not they actually want to use the pipeline library functionality. Developers found this to be annoying and unfriendly behavior. We do wish to require all **implementations** to support it though, and thus it is listed in the feature requirements section.

### Sample Code

Example ray generation GLSL shader

```
#version 450 core
#extension GL_EXT_ray_tracing : require
layout(set = 0, binding = 0, rgba8) uniform image2D image;
layout(set = 0, binding = 1) uniform accelerationStructureEXT as;
layout(location = 0) rayPayloadEXT float payload;

void main()
{
   vec4 col = vec4(0, 0, 0, 1);

   vec3 origin = vec3(float(gl_LaunchIDEXT.x)/float(gl_LaunchSizeEXT.x), float
(gl_LaunchIDEXT.y)/float(gl_LaunchSizeEXT.y), 1.0);
   vec3 dir = vec3(0.0, 0.0, -1.0);

   traceRayEXT(as, 0, 0xff, 0, 1, 0, origin, 0.0, dir, 1000.0, 0);

   col.y = payload;

   imageStore(image, ivec2(gl_LaunchIDEXT.xy), col);
}
```

### Version History

- Revision 1, 2020-11-12 (Mathieu Robart, Daniel Koch, Eric Werness, Tobias Hector)
  - Decomposition of the specification, from VK_KHR_ray_tracing to VK_KHR_ray_tracing_pipeline (#1918,!3912)
  - require certain subgroup and sm_shader_builtin shader builtins to be decorated as volatile in the ray generation, closest hit, miss, intersection, and callable stages (#1924,!3903,!3954)
  - clarify buffer usage flags for ray tracing (#2181,!3939)
  - rename maxRecursionDepth to maxRayPipelineRecursionDepth and maxRayRecursionDepth (#2203,!3937)
  - add queryable maxRayHitAttributeSize and rename members of VkRayTracingPipelineInterfaceCreateInfoKHR (#2102,!3966)
  - update to use `RayTracingKHR` SPIR-V capability
  - add VUs for matching hit group type against geometry type (#2245,!3994)

- require `RayTMaxKHR` be volatile in intersection shaders (#2268,!4030)
- add numerical limits for ray parameters (#2235,!3960)
- fix SBT indexing rules for device addresses (#2308,!4079)
- relax formula for ray intersection candidate determination (#2322,!4080)
- add more details on `ShaderRecordBufferKHR` variables (#2230,!4083)
- clarify valid bits for `InstanceCustomIndexKHR` (GLSL/GLSL#19,!4128)
- allow at most one `IncomingRayPayloadKHR`, `IncomingCallableDataKHR`, and `HitAttributeKHR` (!4129)
- add minimum for maxShaderGroupStride (#2353,!4131)
- require VK_KHR_pipeline_library extension to be supported (#2348,!4135)
- clarify meaning of 'geometry index' (#2272,!4137)
- restrict traces to TLAS (#2239,!4141)
- add note about maxPipelineRayPayloadSize (#2383,!4172)
- do not require raygen shader in pipeline libraries (!4185)
- define sync for indirect trace rays and indirect buffer (#2407,!4208)

## VK_KHR_ray_tracing_position_fetch

### Name String

`VK_KHR_ray_tracing_position_fetch`

### Extension Type

Device extension

### Registered Extension Number

482

### Revision

1

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_acceleration_structure

### SPIR-V Dependencies

- SPV_KHR_ray_tracing_position_fetch

### Contact

- Eric Werness

### Extension Proposal

VK_KHR_ray_tracing_position_fetch

### Other Extension Metadata

#### Last Modified Date

2023-02-17

#### Interactions and External Dependencies

- This extension provides API support for `GLSL_EXT_ray_tracing_position_fetch`
- Interacts with `VK_KHR_ray_tracing_pipeline`
- Interacts with `VK_KHR_ray_query`

#### Contributors

- Eric Werness, NVIDIA
- Stu Smith, AMD
- Yuriy O'Donnell, Epic Games
- Ralph Potter, Samsung
- Joshua Barczak, Intel
- Lionel Landwerlin, Intel
- Andrew Garrard, Imagination Technologies
- Alex Bourd, Qualcomm
- Yunpeng Zhu, Huawei Technologies
- Marius Bjorge, Arm
- Daniel Koch, NVIDIA

### Description

`VK_KHR_ray_tracing_position_fetch` adds the ability to fetch the vertex positions in the shader from a hit triangle as stored in the acceleration structure.

An application adds `VK_BUILD_ACCELERATION_STRUCTURE_ALLOW_DATA_ACCESS_KHR` to the acceleration structure at build time. Then, if the hit is a triangle geometry, the shader (any-hit or closest hit for ray pipelines or using ray query) **can** fetch the three, three-component vertex positions in object space, of the triangle which was hit.

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceRayTracingPositionFetchFeaturesKHR

### New Enum Constants

- `VK_KHR_RAY_TRACING_POSITION_FETCH_EXTENSION_NAME`
- `VK_KHR_RAY_TRACING_POSITION_FETCH_SPEC_VERSION`
- Extending VkBuildAccelerationStructureFlagBitsKHR:

    - `VK_BUILD_ACCELERATION_STRUCTURE_ALLOW_DATA_ACCESS_KHR`
- Extending VkStructureType:
    - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_RAY_TRACING_POSITION_FETCH_FEATURES_KHR`

#### New Built-In Variables

- `HitTriangleVertexPositionsKHR`

#### New SPIR-V Capabilities

- RayTracingPositionFetchKHR
- RayQueryPositionFetchKHR

#### Issues

None Yet!

#### Version History

- Revision 1, 2023-02-17 (Eric Werness)
    - internal revisions

## VK_KHR_sampler_mirror_clamp_to_edge

#### Name String

`VK_KHR_sampler_mirror_clamp_to_edge`

#### Extension Type

Device extension

#### Registered Extension Number

15

#### Revision

3

#### Ratification Status

Ratified

#### Extension and Version Dependencies

None

#### Deprecation State

- *Promoted* to Vulkan 1.2

#### Contact

- Tobias Hector [GitHub]tobski

### Other Extension Metadata

#### Last Modified Date

2019-08-17

#### Contributors

- Tobias Hector, Imagination Technologies
- Jon Leech, Khronos

#### Description

`VK_KHR_sampler_mirror_clamp_to_edge` extends the set of sampler address modes to include an additional mode (`VK_SAMPLER_ADDRESS_MODE_MIRROR_CLAMP_TO_EDGE`) that effectively uses a texture map twice as large as the original image in which the additional half of the new image is a mirror image of the original image.

This new mode relaxes the need to generate images whose opposite edges match by using the original image to generate a matching "mirror image". This mode allows the texture to be mirrored only once in the negative s, t, and r directions.

#### Promotion to Vulkan 1.2

All functionality in this extension is included in core Vulkan 1.2. However, if Vulkan 1.2 is supported and this extension is not, the VkSamplerAddressMode `VK_SAMPLER_ADDRESS_MODE_MIRROR_CLAMP_TO_EDGE` is optional. Since the original extension did not use an author suffix on the enum `VK_SAMPLER_ADDRESS_MODE_MIRROR_CLAMP_TO_EDGE`, it is used by both core and extension implementations.

#### New Enum Constants

- `VK_KHR_SAMPLER_MIRROR_CLAMP_TO_EDGE_EXTENSION_NAME`
- `VK_KHR_SAMPLER_MIRROR_CLAMP_TO_EDGE_SPEC_VERSION`
- Extending VkSamplerAddressMode:
  - `VK_SAMPLER_ADDRESS_MODE_MIRROR_CLAMP_TO_EDGE`
  - `VK_SAMPLER_ADDRESS_MODE_MIRROR_CLAMP_TO_EDGE_KHR`

#### Example

Creating a sampler with the new address mode in each dimension

```
VkSamplerCreateInfo createInfo =
{
    .sType = VK_STRUCTURE_TYPE_SAMPLER_CREATE_INFO,
    // Other members set to application-desired values
};

createInfo.addressModeU = VK_SAMPLER_ADDRESS_MODE_MIRROR_CLAMP_TO_EDGE;
createInfo.addressModeV = VK_SAMPLER_ADDRESS_MODE_MIRROR_CLAMP_TO_EDGE;
```

```
    createInfo.addressModeW = VK_SAMPLER_ADDRESS_MODE_MIRROR_CLAMP_TO_EDGE;

    VkSampler sampler;
    VkResult result = vkCreateSampler(
        device,
        &createInfo,
        &sampler);
```

#### Issues

1) Why are both KHR and core versions of the `VK_SAMPLER_ADDRESS_MODE_MIRROR_CLAMP_TO_EDGE` token present?

**RESOLVED**: This functionality was intended to be required in Vulkan 1.0. We realized shortly before public release that not all implementations could support it, and moved the functionality into an optional extension, but did not apply the KHR extension suffix. Adding a KHR-suffixed alias of the non-suffixed enum has been done to comply with our own naming rules.

In a related change, before spec revision 1.1.121 this extension was hardwiring into the spec Makefile so it was always included with the Specification, even in the core-only versions. This has now been reverted, and it is treated as any other extension.

#### Version History

- Revision 1, 2016-02-16 (Tobias Hector)
  - Initial draft
- Revision 2, 2019-08-14 (Jon Leech)
  - Add KHR-suffixed alias of non-suffixed enum.
- Revision 3, 2019-08-17 (Jon Leech)
  - Add an issue explaining the reason for the extension API not being suffixed with KHR.

## VK_KHR_separate_depth_stencil_layouts

#### Name String

`VK_KHR_separate_depth_stencil_layouts`

#### Extension Type

Device extension

#### Registered Extension Number

242

#### Revision

1

#### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1
and
VK_KHR_create_renderpass2
or
Vulkan Version 1.2

### Deprecation State

- *Promoted* to Vulkan 1.2

### Contact

- Piers Daniell [GitHub]pdaniell-nv

### Other Extension Metadata

### Last Modified Date

2019-06-25

### Contributors

- Daniel Koch, NVIDIA
- Jeff Bolz, NVIDIA
- Jesse Barker, Unity
- Tobias Hector, AMD

### Description

This extension allows image memory barriers for depth/stencil images to have just one of the `VK_IMAGE_ASPECT_DEPTH_BIT` or `VK_IMAGE_ASPECT_STENCIL_BIT` aspect bits set, rather than require both. This allows their layouts to be set independently. To support depth/stencil images with different layouts for the depth and stencil aspects, the depth/stencil attachment interface has been updated to support a separate layout for stencil.

### Promotion to Vulkan 1.2

All functionality in this extension is included in core Vulkan 1.2, with the KHR suffix omitted. The original type, enum and command names are still available as aliases of the core functionality.

### New Structures

- Extending VkAttachmentDescription2:
  - VkAttachmentDescriptionStencilLayoutKHR
- Extending VkAttachmentReference2:
  - VkAttachmentReferenceStencilLayoutKHR
- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:

- VkPhysicalDeviceSeparateDepthStencilLayoutsFeaturesKHR

### New Enum Constants

- VK_KHR_SEPARATE_DEPTH_STENCIL_LAYOUTS_EXTENSION_NAME
- VK_KHR_SEPARATE_DEPTH_STENCIL_LAYOUTS_SPEC_VERSION
- Extending VkImageLayout:
  - VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL_KHR
  - VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL_KHR
  - VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL_KHR
  - VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL_KHR
- Extending VkStructureType:
  - VK_STRUCTURE_TYPE_ATTACHMENT_DESCRIPTION_STENCIL_LAYOUT_KHR
  - VK_STRUCTURE_TYPE_ATTACHMENT_REFERENCE_STENCIL_LAYOUT_KHR
  - VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SEPARATE_DEPTH_STENCIL_LAYOUTS_FEATURES_KHR

### Version History

- Revision 1, 2019-06-25 (Piers Daniell)
  - Internal revisions

## VK_KHR_shader_atomic_int64

### Name String

VK_KHR_shader_atomic_int64

### Extension Type

Device extension

### Registered Extension Number

181

### Revision

1

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### Deprecation State

- *Promoted* to Vulkan 1.2

### Contact

- Aaron Hagan [GitHub]ahagan

### Other Extension Metadata

### Last Modified Date

2018-07-05

### Interactions and External Dependencies

- This extension provides API support for `GL_ARB_gpu_shader_int64` and `GL_EXT_shader_atomic_int64`

### Contributors

- Aaron Hagan, AMD
- Daniel Rakos, AMD
- Jeff Bolz, NVIDIA
- Neil Henning, Codeplay

### Description

This extension advertises the SPIR-V **Int64Atomics** capability for Vulkan, which allows a shader to contain 64-bit atomic operations on signed and unsigned integers. The supported operations include OpAtomicMin, OpAtomicMax, OpAtomicAnd, OpAtomicOr, OpAtomicXor, OpAtomicAdd, OpAtomicExchange, and OpAtomicCompareExchange.

### Promotion to Vulkan 1.2

All functionality in this extension is included in core Vulkan 1.2, with the KHR suffix omitted. However, if Vulkan 1.2 is supported and this extension is not, the `shaderBufferInt64Atomics` capability is optional. The original type, enum and command names are still available as aliases of the core functionality.

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceShaderAtomicInt64FeaturesKHR

### New Enum Constants

- `VK_KHR_SHADER_ATOMIC_INT64_EXTENSION_NAME`
- `VK_KHR_SHADER_ATOMIC_INT64_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_ATOMIC_INT64_FEATURES_KHR`

### New SPIR-V Capabilities

- `Int64Atomics`

### Version History

- Revision 1, 2018-07-05 (Aaron Hagan)
  - Internal revisions

## VK_KHR_shader_clock

### Name String

`VK_KHR_shader_clock`

### Extension Type

Device extension

### Registered Extension Number

182

### Revision

1

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### SPIR-V Dependencies

- SPV_KHR_shader_clock

### Contact

- Aaron Hagan [GitHub]ahagan

### Other Extension Metadata

### Last Modified Date

2019-4-25

### IP Status

No known IP claims.

### Interactions and External Dependencies

- This extension provides API support for `GL_ARB_shader_clock` and `GL_EXT_shader_realtime_clock`

### Contributors

- Aaron Hagan, AMD
- Daniel Koch, NVIDIA

### Description

This extension advertises the SPIR-V `ShaderClockKHR` capability for Vulkan, which allows a shader to query a real-time or monotonically incrementing counter at the subgroup level or across the device level. The two valid SPIR-V scopes for `OpReadClockKHR` are `Subgroup` and `Device`.

When using GLSL source-based shading languages, the `clockRealtime*EXT`() timing functions map to the `OpReadClockKHR` instruction with a scope of `Device`, and the `clock*ARB`() timing functions map to the `OpReadClockKHR` instruction with a scope of `Subgroup`.

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceShaderClockFeaturesKHR

### New Enum Constants

- `VK_KHR_SHADER_CLOCK_EXTENSION_NAME`
- `VK_KHR_SHADER_CLOCK_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_CLOCK_FEATURES_KHR`

### New SPIR-V Capabilities

- `ShaderClockKHR`

### Version History

- Revision 1, 2019-4-25 (Aaron Hagan)
  - Initial revision

## VK_KHR_shader_expect_assume

### Name String

`VK_KHR_shader_expect_assume`

### Extension Type

Device extension

### Registered Extension Number

545

### Revision

1

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### SPIR-V Dependencies

- SPV_KHR_expect_assume

### Contact

- Kevin Petit [GitHub]kpet

### Extension Proposal

VK_KHR_shader_expect_assume

### Other Extension Metadata

### Last Modified Date

2023-12-06

### IP Status

No known IP claims.

### Contributors

- Kevin Petit, Arm
- Tobias Hector, AMD
- James Fitzpatrick, Imagination Technologies

### Description

This extension allows the use of the `SPV_KHR_expect_assume` extension in SPIR-V shader modules which enables SPIR-V producers to provide optimization hints to the Vulkan implementation.

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceShaderExpectAssumeFeaturesKHR

### New Enum Constants

- `VK_KHR_SHADER_EXPECT_ASSUME_EXTENSION_NAME`
- `VK_KHR_SHADER_EXPECT_ASSUME_SPEC_VERSION`
- Extending VkStructureType:

- `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_EXPECT_ASSUME_FEATURES_KHR`

### New SPIR-V Capabilities

- ExpectAssumeKHR

### Version History

- Revision 1, 2023-12-06 (Kevin Petit)
  - Initial revision

## VK_KHR_shader_float16_int8

### Name String

`VK_KHR_shader_float16_int8`

### Extension Type

Device extension

### Registered Extension Number

83

### Revision

1

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### Deprecation State

- *Promoted* to Vulkan 1.2

### Contact

- Alexander Galazin [GitHub]alegal-arm

### Other Extension Metadata

### Last Modified Date

2018-03-07

### Interactions and External Dependencies

- This extension interacts with `VK_KHR_8bit_storage`
- This extension interacts with `VK_KHR_16bit_storage`
- This extension interacts with `VK_KHR_shader_float_controls`

- This extension provides API support for `GL_EXT_shader_explicit_arithmetic_types`

**IP Status**

No known IP claims.

**Contributors**

- Alexander Galazin, Arm
- Jan-Harald Fredriksen, Arm
- Jeff Bolz, NVIDIA
- Graeme Leese, Broadcom
- Daniel Rakos, AMD

**Description**

The `VK_KHR_shader_float16_int8` extension allows use of 16-bit floating-point types and 8-bit integer types in shaders for arithmetic operations.

It introduces two new optional features `shaderFloat16` and `shaderInt8` which directly map to the `Float16` and the `Int8` SPIR-V capabilities. The `VK_KHR_shader_float16_int8` extension also specifies precision requirements for half-precision floating-point SPIR-V operations. This extension does not enable use of 8-bit integer types or 16-bit floating-point types in any shader input and output interfaces and therefore does not supersede the `VK_KHR_8bit_storage` or `VK_KHR_16bit_storage` extensions.

**Promotion to Vulkan 1.2**

All functionality in this extension is included in core Vulkan 1.2, with the KHR suffix omitted. However, if Vulkan 1.2 is supported and this extension is not, both the `shaderFloat16` and `shaderInt8` capabilities are optional. The original type, enum and command names are still available as aliases of the core functionality.

**New Structures**

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceFloat16Int8FeaturesKHR
  - VkPhysicalDeviceShaderFloat16Int8FeaturesKHR

**New Enum Constants**

- `VK_KHR_SHADER_FLOAT16_INT8_EXTENSION_NAME`
- `VK_KHR_SHADER_FLOAT16_INT8_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_FLOAT16_INT8_FEATURES_KHR`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_FLOAT16_INT8_FEATURES_KHR`

### Version History

- Revision 1, 2018-03-07 (Alexander Galazin)
  - Initial draft

## VK_KHR_shader_float_controls

### Name String

`VK_KHR_shader_float_controls`

### Extension Type

Device extension

### Registered Extension Number

198

### Revision

4

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### SPIR-V Dependencies

- SPV_KHR_float_controls

### Deprecation State

- *Promoted* to Vulkan 1.2

### Contact

- Alexander Galazin [GitHub]alegal-arm

### Other Extension Metadata

### Last Modified Date

2018-09-11

### IP Status

No known IP claims.

### Contributors

- Alexander Galazin, Arm
- Jan-Harald Fredriksen, Arm
- Jeff Bolz, NVIDIA

- Graeme Leese, Broadcom
- Daniel Rakos, AMD

### Description

The `VK_KHR_shader_float_controls` extension enables efficient use of floating-point computations through the ability to query and override the implementation's default behavior for rounding modes, denormals, signed zero, and infinity.

### Promotion to Vulkan 1.2

All functionality in this extension is included in core Vulkan 1.2, with the KHR suffix omitted. The original type, enum and command names are still available as aliases of the core functionality.

### New Structures

- Extending VkPhysicalDeviceProperties2:
  - VkPhysicalDeviceFloatControlsPropertiesKHR

### New Enums

- VkShaderFloatControlsIndependenceKHR

### New Enum Constants

- `VK_KHR_SHADER_FLOAT_CONTROLS_EXTENSION_NAME`
- `VK_KHR_SHADER_FLOAT_CONTROLS_SPEC_VERSION`
- Extending VkShaderFloatControlsIndependence:
  - `VK_SHADER_FLOAT_CONTROLS_INDEPENDENCE_32_BIT_ONLY_KHR`
  - `VK_SHADER_FLOAT_CONTROLS_INDEPENDENCE_ALL_KHR`
  - `VK_SHADER_FLOAT_CONTROLS_INDEPENDENCE_NONE_KHR`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_FLOAT_CONTROLS_PROPERTIES_KHR`

### New SPIR-V Capabilities

- `DenormPreserve`
- `DenormFlushToZero`
- `SignedZeroInfNanPreserve`
- `RoundingModeRTE`
- `RoundingModeRTZ`

### Issues

1) Which instructions must flush denorms?

**RESOLVED**: Only floating-point conversion, floating-point arithmetic, floating-point relational (except `OpIsNaN`, `OpIsInf`), and floating-point GLSL.std.450 extended instructions must flush denormals.

2) What is the denorm behavior for intermediate results?

**RESOLVED**: When a SPIR-V instruction is implemented as a sequence of other instructions:

- in the `DenormFlushToZero` execution mode, the intermediate instructions may flush denormals, the final result of the sequence **must** not be denormal.
- in the `DenormPreserve` execution mode, denormals must be preserved throughout the whole sequence.

3) Do denorm and rounding mode controls apply to `OpSpecConstantOp`?

**RESOLVED**: Yes, except when the opcode is `OpQuantizeToF16`.

4) The SPIR-V specification says that `OpConvertFToU` and `OpConvertFToS` unconditionally round towards zero. Do the rounding mode controls specified through the execution modes apply to them?

**RESOLVED**: No, these instructions unconditionally round towards zero.

5) Do any of the "Pack" GLSL.std.450 instructions count as conversion instructions and have the rounding mode applied?

**RESOLVED**: No, only instructions listed in "section 3.32.11. Conversion Instructions" of the SPIR-V specification count as conversion instructions.

6) When using inf/nan-ignore mode, what is expected of `OpIsNan` and `OpIsInf`?

**RESOLVED**: These instructions must always accurately detect inf/nan if it is passed to them.

**Version 4 API Incompatibility**

The original versions of `VK_KHR_shader_float_controls` shipped with booleans named "separateDenormSettings" and "separateRoundingModeSettings", which at first glance could have indicated "they can all be set independently, or not". However the spec language as written indicated that the 32-bit value could always be set independently, and only the 16- and 64-bit controls needed to be the same if these values were `VK_FALSE`.

As a result of this slight disparity, and lack of test coverage for this facet of the extension, we ended up with two different behaviors in the wild, where some implementations worked as written, and others worked based on the naming. As these are hard limits in hardware with reasons for exposure as written, it was not possible to standardize on a single way to make this work within the existing API.

No known users of this part of the extension exist in the wild, and as such the Vulkan WG took the unusual step of retroactively changing the once boolean value into a tri-state enum, breaking source compatibility. This was however done in such a way as to retain ABI compatibility, in case any code using this did exist; with the numerical values 0 and 1 retaining their original specified

meaning, and a new value signifying the additional “all need to be set together” state. If any applications exist today, compiled binaries will continue to work as written in most cases, but will need changes before the code can be recompiled.

### Version History

- Revision 4, 2019-06-18 (Tobias Hector)
  - Modified settings restrictions, see Version 4 API incompatibility
- Revision 3, 2018-09-11 (Alexander Galazin)
  - Minor restructuring
- Revision 2, 2018-04-17 (Alexander Galazin)
  - Added issues and resolutions
- Revision 1, 2018-04-11 (Alexander Galazin)
  - Initial draft

## VK_KHR_shader_float_controls2

### Name String

`VK_KHR_shader_float_controls2`

### Extension Type

Device extension

### Registered Extension Number

529

### Revision

1

### Ratification Status

Ratified

### Extension and Version Dependencies

Vulkan Version 1.1
and
VK_KHR_shader_float_controls

### SPIR-V Dependencies

- SPV_KHR_float_controls2

### Contact

- Graeme Leese [GitHub]gnl21

### Extension Proposal

VK_KHR_shader_float_controls2

### Other Extension Metadata

#### Last Modified Date

2023-05-16

#### Interactions and External Dependencies

- This extension requires `SPV_KHR_float_controls2`.

#### Contributors

- Graeme Leese, Broadcom

#### Description

This extension enables use of the more expressive fast floating-point math flags in the SPV_KHR_float_controls2 extension. These flags give finer- grained control over which optimizations compilers may apply, potentially speeding up execution while retaining correct results.

The extension also adds control over the fast-math modes to the GLSL extended instruction set, making these operations more consistent with SPIR-V and allowing their use in situations where floating-point conformance is important.

#### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceShaderFloatControls2FeaturesKHR

#### New Enum Constants

- `VK_KHR_SHADER_FLOAT_CONTROLS_2_EXTENSION_NAME`
- `VK_KHR_SHADER_FLOAT_CONTROLS_2_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_FLOAT_CONTROLS_2_FEATURES_KHR`

#### New SPIR-V Capabilities

- FloatControls2

#### Version History

- Revision 1, 2023-05-16 (Graeme Leese)
  - Initial draft

## VK_KHR_shader_integer_dot_product

#### Name String

`VK_KHR_shader_integer_dot_product`

**Extension Type**

Device extension

**Registered Extension Number**

281

**Revision**

1

**Ratification Status**

Ratified

**Extension and Version Dependencies**

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

**SPIR-V Dependencies**

- SPV_KHR_integer_dot_product

**Deprecation State**

- *Promoted* to Vulkan 1.3

**Contact**

- Kevin Petit [GitHub]kpet

**Extension Proposal**

VK_KHR_shader_integer_dot_product

**Other Extension Metadata**

**Last Modified Date**

2021-06-16

**Interactions and External Dependencies**

- This extension interacts with `VK_KHR_shader_float16_int8`.

**IP Status**

No known IP claims.

**Contributors**

- Kévin Petit, Arm Ltd.
- Jeff Bolz, NVidia
- Spencer Fricke, Samsung
- Jesse Hall, Google
- John Kessenich, Google
- Graeme Leese, Broadcom

- Einar Hov, Arm Ltd.
- Stuart Brady, Arm Ltd.
- Pablo Cascon, Arm Ltd.
- Tobias Hector, AMD
- Jeff Leger, Qualcomm
- Ruihao Zhang, Qualcomm
- Pierre Boudier, NVidia
- Jon Leech, The Khronos Group
- Tom Olson, Arm Ltd.

### Description

This extension adds support for the integer dot product SPIR-V instructions defined in SPV_KHR_integer_dot_product. These instructions are particularly useful for neural network inference and training but find uses in other general-purpose compute applications as well.

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceShaderIntegerDotProductFeaturesKHR
- Extending VkPhysicalDeviceProperties2:
  - VkPhysicalDeviceShaderIntegerDotProductPropertiesKHR

### New Enum Constants

- `VK_KHR_SHADER_INTEGER_DOT_PRODUCT_EXTENSION_NAME`
- `VK_KHR_SHADER_INTEGER_DOT_PRODUCT_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_INTEGER_DOT_PRODUCT_FEATURES_KHR`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_INTEGER_DOT_PRODUCT_PROPERTIES_KHR`

### Promotion to Vulkan 1.3

Functionality in this extension is included in core Vulkan 1.3, with the KHR suffix omitted. The original type, enum and command names are still available as aliases of the core functionality.

### New SPIR-V Capabilities

- `DotProductInputAllKHR`
- `DotProductInput4x8BitKHR`
- `DotProductInput4x8BitPackedKHR`
- `DotProductKHR`

### Version History

- Revision 1, 2021-06-16 (Kévin Petit)
  - Initial revision

## VK_KHR_shader_maximal_reconvergence

### Name String

`VK_KHR_shader_maximal_reconvergence`

### Extension Type

Device extension

### Registered Extension Number

435

### Revision

1

### Ratification Status

Ratified

### Extension and Version Dependencies

Vulkan Version 1.1

### SPIR-V Dependencies

- SPV_KHR_maximal_reconvergence

### Contact

- Alan Baker [GitHub]alan-baker

### Extension Proposal

VK_KHR_shader_maximal_reconvergence

### Other Extension Metadata

### Last Modified Date

2021-11-12

### IP Status

No known IP claims.

### Interactions and External Dependencies

- Requires SPIR-V 1.3.
- This extension requires `SPV_KHR_maximal_reconvergence`

### Contributors

- Alan Baker, Google

### Description

This extension allows the use of the `SPV_KHR_maximal_reconvergence` SPIR-V extension in shader modules. `SPV_KHR_maximal_reconvergence` provides stronger guarantees that diverged subgroups will reconverge. These guarantees should match shader author intuition about divergence and reconvergence of invocations based on the structure of the code in the HLL.

Developers should utilize this extension if they require stronger guarantees about reconvergence than either the core spec or SPV_KHR_subgroup_uniform_control_flow. This extension will define the rules that govern how invocations diverge and reconverge in a way that should match developer intuition. It allows robust programs to be written relying on subgroup operations and other tangled instructions.

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceShaderMaximalReconvergenceFeaturesKHR

### New Enum Constants

- `VK_KHR_SHADER_MAXIMAL_RECONVERGENCE_EXTENSION_NAME`
- `VK_KHR_SHADER_MAXIMAL_RECONVERGENCE_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_MAXIMAL_RECONVERGENCE_FEATURES_KHR`

### Version History

- Revision 1, 2021-11-12 (Alan Baker)
  - Internal draft version

## VK_KHR_shader_non_semantic_info

### Name String

`VK_KHR_shader_non_semantic_info`

### Extension Type

Device extension

### Registered Extension Number

294

### Revision

1

### Ratification Status

Ratified

#### Extension and Version Dependencies

None

#### SPIR-V Dependencies

- SPV_KHR_non_semantic_info

#### Deprecation State

- *Promoted* to Vulkan 1.3

#### Contact

- Baldur Karlsson [GitHub]baldurk

#### Other Extension Metadata

#### Last Modified Date

2019-10-16

#### IP Status

No known IP claims.

#### Contributors

- Baldur Karlsson, Valve

#### Description

This extension allows the use of the `SPV_KHR_non_semantic_info` extension in SPIR-V shader modules.

#### New Enum Constants

- `VK_KHR_SHADER_NON_SEMANTIC_INFO_EXTENSION_NAME`
- `VK_KHR_SHADER_NON_SEMANTIC_INFO_SPEC_VERSION`

#### Promotion to Vulkan 1.3

Functionality in this extension is included in core Vulkan 1.3 Because the extension has no API controlling its functionality, this results only in a change to the SPIR-V Extensions table.

#### Version History

- Revision 1, 2019-10-16 (Baldur Karlsson)
  - Initial revision

## VK_KHR_shader_quad_control

#### Name String

`VK_KHR_shader_quad_control`

#### Extension Type

Device extension

**Registered Extension Number**

236

**Revision**

1

**Ratification Status**

Ratified

**Extension and Version Dependencies**

Vulkan Version 1.1
and
VK_KHR_vulkan_memory_model
and
VK_KHR_shader_maximal_reconvergence

**SPIR-V Dependencies**

- SPV_KHR_quad_control

**Contact**

- Tobias Hector [GitHub]tobski

**Extension Proposal**

VK_KHR_shader_quad_control

**Other Extension Metadata**

**Last Modified Date**

2023-11-01

**IP Status**

No known IP claims.

**Contributors**

- Tobias Hector, AMD
- Bill Licea-Kane, Qualcomm
- Graeme Leese, Broadcom
- Jan-Harald Fredriksen, Arm
- Nicolai Hähnle, AMD
- Jeff Bolz, NVidia
- Alan Baker, Google
- Hans-Kristian Arntzen, Valve

**Description**

This extension adds new quad any/all operations, requires that derivatives are well-defined in

quad-uniform control flow, and adds the ability to require helper invocations participate in group operations.

#### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceShaderQuadControlFeaturesKHR

#### New Enum Constants

- `VK_KHR_SHADER_QUAD_CONTROL_EXTENSION_NAME`
- `VK_KHR_SHADER_QUAD_CONTROL_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_QUAD_CONTROL_FEATURES_KHR`

#### New SPIR-V Capabilities

- QuadControlKHR

#### Version History

- Revision 1, 2023-11-01 (Tobias Hector)
  - Initial draft

## VK_KHR_shader_relaxed_extended_instruction

#### Name String

`VK_KHR_shader_relaxed_extended_instruction`

#### Extension Type

Device extension

#### Registered Extension Number

559

#### Revision

1

#### Ratification Status

Ratified

#### Extension and Version Dependencies

None

#### SPIR-V Dependencies

- SPV_KHR_relaxed_extended_instruction

### Contact

- Nathan Gauër [GitHub]Keenuts

### Extension Proposal

VK_KHR_shader_relaxed_extended_instruction

### Other Extension Metadata

### Last Modified Date

2024-01-24

### IP Status

No known IP claims.

### Contributors

- Alan Baker, Google LLC
- Nathan Gauër, Google LLC

### Description

This extension allows the use of the `SPV_KHR_relaxed_extended_instruction` extension in SPIR-V shader modules.

It adds a new SPIR-V instruction, which allows some usage of forward references in non-semantic instruction sets. This extensions interacts with the `SPV_KHR_non_semantic_info` extension, hence with `VK_KHR_shader_non_semantic_info`.

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceShaderRelaxedExtendedInstructionFeaturesKHR

### New Enum Constants

- `VK_KHR_SHADER_RELAXED_EXTENDED_INSTRUCTION_EXTENSION_NAME`
- `VK_KHR_SHADER_RELAXED_EXTENDED_INSTRUCTION_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_RELAXED_EXTENDED_INSTRUCTION_FEATURES_KHR`

### Version History

- Revision 1, 2024-01-24 (Nathan Gauër)
  - Initial revision

## VK_KHR_shader_subgroup_extended_types

**Name String**

`VK_KHR_shader_subgroup_extended_types`

**Extension Type**

Device extension

**Registered Extension Number**

176

**Revision**

1

**Ratification Status**

Ratified

**Extension and Version Dependencies**

Vulkan Version 1.1

**Deprecation State**

- *Promoted* to Vulkan 1.2

**Contact**

- Neil Henning [GitHub]sheredom

**Other Extension Metadata**

**Last Modified Date**

2019-01-08

**IP Status**

No known IP claims.

**Interactions and External Dependencies**

- This extension provides API support for `GLSL_EXT_shader_subgroup_extended_types`

**Contributors**

- Jeff Bolz, NVIDIA
- Jan-Harald Fredriksen, Arm
- Neil Henning, AMD
- Daniel Koch, NVIDIA
- Jeff Leger, Qualcomm
- Graeme Leese, Broadcom
- David Neto, Google
- Daniel Rakos, AMD

### Description

This extension enables the Non Uniform Group Operations in SPIR-V to support 8-bit integer, 16-bit integer, 64-bit integer, 16-bit floating-point, and vectors of these types.

### Promotion to Vulkan 1.2

All functionality in this extension is included in core Vulkan 1.2, with the KHR suffix omitted. The original type, enum and command names are still available as aliases of the core functionality.

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceShaderSubgroupExtendedTypesFeaturesKHR

### New Enum Constants

- `VK_KHR_SHADER_SUBGROUP_EXTENDED_TYPES_EXTENSION_NAME`
- `VK_KHR_SHADER_SUBGROUP_EXTENDED_TYPES_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_SUBGROUP_EXTENDED_TYPES_FEATURES_KHR`

### Version History

- Revision 1, 2019-01-08 (Neil Henning)
  - Initial draft

## VK_KHR_shader_subgroup_rotate

### Name String

`VK_KHR_shader_subgroup_rotate`

### Extension Type

Device extension

### Registered Extension Number

417

### Revision

2

### Ratification Status

Ratified

### Extension and Version Dependencies

None

**SPIR-V Dependencies**

- SPV_KHR_subgroup_rotate

**Contact**

- Kevin Petit [GitHub]kpet

**Extension Proposal**

VK_KHR_shader_subgroup_rotate

**Last Modified Date**

2024-01-29

**IP Status**

No known IP claims.

**Contributors**

- Kévin Petit, Arm Ltd.
- Tobias Hector, AMD
- John Leech, Khronos
- Matthew Netsch, Qualcomm
- Jan-Harald Fredriksen, Arm Ltd.
- Graeme Leese, Broadcom
- Tom Olson, Arm Ltd.
- Spencer Fricke, LunarG Inc.

This extension adds support for the subgroup rotate instruction defined in SPV_KHR_subgroup_rotate.

**New Structures**

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceShaderSubgroupRotateFeaturesKHR

**New Enum Constants**

- `VK_KHR_SHADER_SUBGROUP_ROTATE_EXTENSION_NAME`
- `VK_KHR_SHADER_SUBGROUP_ROTATE_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_SUBGROUP_ROTATE_FEATURES_KHR`
- Extending VkSubgroupFeatureFlagBits:
  - `VK_SUBGROUP_FEATURE_ROTATE_BIT_KHR`
  - `VK_SUBGROUP_FEATURE_ROTATE_CLUSTERED_BIT_KHR`

### New SPIR-V Capabilities

- GroupNonUniformRotateKHR

### Version History

- Revision 2, 2024-01-29 (Kévin Petit)
  - Add `VK_SUBGROUP_FEATURE_ROTATE_BIT_KHR` and `VK_SUBGROUP_FEATURE_ROTATE_CLUSTERED_BIT_KHR`
- Revision 1, 2023-06-20 (Kévin Petit)
  - Initial revision

## VK_KHR_shader_subgroup_uniform_control_flow

### Name String

`VK_KHR_shader_subgroup_uniform_control_flow`

### Extension Type

Device extension

### Registered Extension Number

324

### Revision

1

### Ratification Status

Ratified

### Extension and Version Dependencies

Vulkan Version 1.1

### SPIR-V Dependencies

- SPV_KHR_subgroup_uniform_control_flow

### Contact

- Alan Baker [GitHub]alan-baker

### Other Extension Metadata

### Last Modified Date

2020-08-27

### IP Status

No known IP claims.

### Interactions and External Dependencies

- Requires SPIR-V 1.3.
- This extension provides API support for `GL_EXT_subgroupuniform_qualifier`

### Contributors

- Alan Baker, Google
- Jeff Bolz, NVIDIA

### Description

This extension allows the use of the `SPV_KHR_subgroup_uniform_control_flow` SPIR-V extension in shader modules. `SPV_KHR_subgroup_uniform_control_flow` provides stronger guarantees that diverged subgroups will reconverge.

Developers should utilize this extension if they use subgroup operations to reduce the work performed by a uniform subgroup. This extension will guarantee that uniform subgroup will reconverge in the same manner as invocation groups (see “Uniform Control Flow” in the Khronos SPIR-V Specification).

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceShaderSubgroupUniformControlFlowFeaturesKHR

### New Enum Constants

- `VK_KHR_SHADER_SUBGROUP_UNIFORM_CONTROL_FLOW_EXTENSION_NAME`
- `VK_KHR_SHADER_SUBGROUP_UNIFORM_CONTROL_FLOW_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_SUBGROUP_UNIFORM_CONTROL_FLOW_FEATURES_KHR`

### Version History

- Revision 1, 2020-08-27 (Alan Baker)
  - Internal draft version

## VK_KHR_shader_terminate_invocation

### Name String

`VK_KHR_shader_terminate_invocation`

### Extension Type

Device extension

### Registered Extension Number

216

### Revision

1

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### SPIR-V Dependencies

- SPV_KHR_terminate_invocation

### Deprecation State

- *Promoted* to Vulkan 1.3

### Contact

- Jesse Hall [GitHub]critsec

### Other Extension Metadata

### Last Modified Date

2020-08-11

### IP Status

No known IP claims.

### Contributors

- Alan Baker, Google
- Jeff Bolz, NVIDIA
- Jesse Hall, Google
- Ralph Potter, Samsung
- Tom Olson, Arm

### Description

This extension adds Vulkan support for the `SPV_KHR_terminate_invocation` SPIR-V extension. That SPIR-V extension provides a new instruction, `OpTerminateInvocation`, which causes a shader invocation to immediately terminate and sets the coverage of shaded samples to `0`; only previously executed instructions will have observable effects. The `OpTerminateInvocation` instruction, along with the `OpDemoteToHelperInvocation` instruction from the `VK_EXT_shader_demote_to_helper_invocation` extension, together replace the `OpKill` instruction, which could behave like either of these instructions. `OpTerminateInvocation` provides the behavior required by the GLSL `discard` statement, and should be used when available by GLSL compilers and applications that need the GLSL `discard` behavior.

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:

- VkPhysicalDeviceShaderTerminateInvocationFeaturesKHR

#### New Enum Constants

- VK_KHR_SHADER_TERMINATE_INVOCATION_EXTENSION_NAME
- VK_KHR_SHADER_TERMINATE_INVOCATION_SPEC_VERSION
- Extending VkStructureType:
  - VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_TERMINATE_INVOCATION_FEATURES_KHR

#### Promotion to Vulkan 1.3

Functionality in this extension is included in core Vulkan 1.3, with the KHR suffix omitted. The original type, enum and command names are still available as aliases of the core functionality.

#### Version History

- Revision 1, 2020-08-11 (Jesse Hall)

### VK_KHR_shared_presentable_image

#### Name String

VK_KHR_shared_presentable_image

#### Extension Type

Device extension

#### Registered Extension Number

112

#### Revision

1

#### Ratification Status

Ratified

#### Extension and Version Dependencies

VK_KHR_swapchain
and
VK_KHR_get_surface_capabilities2
and
VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

#### Contact

- Alon Or-bach [GitHub]alonorbach

### Other Extension Metadata

#### Last Modified Date

2017-03-20

#### IP Status

No known IP claims.

#### Contributors

- Alon Or-bach, Samsung Electronics
- Ian Elliott, Google
- Jesse Hall, Google
- Pablo Ceballos, Google
- Chris Forbes, Google
- Jeff Juliano, NVIDIA
- James Jones, NVIDIA
- Daniel Rakos, AMD
- Tobias Hector, Imagination Technologies
- Graham Connor, Imagination Technologies
- Michael Worcester, Imagination Technologies
- Cass Everitt, Oculus
- Johannes Van Waveren, Oculus

### Description

This extension extends `VK_KHR_swapchain` to enable creation of a shared presentable image. This allows the application to use the image while the presention engine is accessing it, in order to reduce the latency between rendering and presentation.

### New Commands

- vkGetSwapchainStatusKHR

### New Structures

- Extending VkSurfaceCapabilities2KHR:
  - VkSharedPresentSurfaceCapabilitiesKHR

### New Enum Constants

- `VK_KHR_SHARED_PRESENTABLE_IMAGE_EXTENSION_NAME`
- `VK_KHR_SHARED_PRESENTABLE_IMAGE_SPEC_VERSION`
- Extending VkImageLayout:

    - `VK_IMAGE_LAYOUT_SHARED_PRESENT_KHR`
- Extending VkPresentModeKHR:
    - `VK_PRESENT_MODE_SHARED_CONTINUOUS_REFRESH_KHR`
    - `VK_PRESENT_MODE_SHARED_DEMAND_REFRESH_KHR`
- Extending VkStructureType:
    - `VK_STRUCTURE_TYPE_SHARED_PRESENT_SURFACE_CAPABILITIES_KHR`

**Issues**

1) Should we allow a Vulkan WSI swapchain to toggle between normal usage and shared presentation usage?

**RESOLVED**: No. WSI swapchains are typically recreated with new properties instead of having their properties changed. This can also save resources, assuming that fewer images are needed for shared presentation, and assuming that most VR applications do not need to switch between normal and shared usage.

2) Should we have a query for determining how the presentation engine refresh is triggered?

**RESOLVED**: Yes. This is done via which presentation modes a surface supports.

3) Should the object representing a shared presentable image be an extension of a VkSwapchainKHR or a separate object?

**RESOLVED**: Extension of a swapchain due to overlap in creation properties and to allow common functionality between shared and normal presentable images and swapchains.

4) What should we call the extension and the new structures it creates?

**RESOLVED**: Shared presentable image / shared present.

5) Should the `minImageCount` and `presentMode` values of the VkSwapchainCreateInfoKHR be ignored, or required to be compatible values?

**RESOLVED**: `minImageCount` must be set to 1, and `presentMode` should be set to either `VK_PRESENT_MODE_SHARED_DEMAND_REFRESH_KHR` or `VK_PRESENT_MODE_SHARED_CONTINUOUS_REFRESH_KHR`.

6) What should the layout of the shared presentable image be?

**RESOLVED**: After acquiring the shared presentable image, the application must transition it to the `VK_IMAGE_LAYOUT_SHARED_PRESENT_KHR` layout prior to it being used. After this initial transition, any image usage that was requested during swapchain creation **can** be performed on the image without layout transitions being performed.

7) Do we need a new API for the trigger to refresh new content?

**RESOLVED**: vkQueuePresentKHR to act as API to trigger a refresh, as will allow combination with other compatible extensions to vkQueuePresentKHR.

8) How should an application detect a VK_ERROR_OUT_OF_DATE_KHR error on a swapchain using the VK_PRESENT_MODE_SHARED_CONTINUOUS_REFRESH_KHR present mode?

**RESOLVED**: Introduce vkGetSwapchainStatusKHR to allow applications to query the status of a swapchain using a shared presentation mode.

9) What should subsequent calls to vkQueuePresentKHR for VK_PRESENT_MODE_SHARED_CONTINUOUS_REFRESH_KHR swapchains be defined to do?

**RESOLVED**: State that implementations may use it as a hint for updated content.

10) Can the ownership of a shared presentable image be transferred to a different queue?

**RESOLVED**: No. It is not possible to transfer ownership of a shared presentable image obtained from a swapchain created using VK_SHARING_MODE_EXCLUSIVE after it has been presented.

11) How should vkQueueSubmit behave if a command buffer uses an image from a VK_ERROR_OUT_OF_DATE_KHR swapchain?

**RESOLVED**: vkQueueSubmit is expected to return the VK_ERROR_DEVICE_LOST error.

12) Can Vulkan provide any guarantee on the order of rendering, to enable beam chasing?

**RESOLVED**: This could be achieved via use of render passes to ensure strip rendering.

#### Version History

- Revision 1, 2017-03-20 (Alon Or-bach)
  - Internal revisions

## VK_KHR_spirv_1_4

#### Name String

VK_KHR_spirv_1_4

#### Extension Type

Device extension

#### Registered Extension Number

237

#### Revision

1

#### Ratification Status

Ratified

#### Extension and Version Dependencies

Vulkan Version 1.1
and

VK_KHR_shader_float_controls

### Deprecation State

- *Promoted* to Vulkan 1.2

### Contact

- Jesse Hall [GitHub]critsec

### Other Extension Metadata

### Last Modified Date

2019-04-01

### IP Status

No known IP claims.

### Interactions and External Dependencies

- Requires SPIR-V 1.4.

### Contributors

- Alexander Galazin, Arm
- David Neto, Google
- Jesse Hall, Google
- John Kessenich, Google
- Neil Henning, AMD
- Tom Olson, Arm

### Description

This extension allows the use of SPIR-V 1.4 shader modules. SPIR-V 1.4's new features primarily make it an easier target for compilers from high-level languages, rather than exposing new hardware functionality.

SPIR-V 1.4 incorporates features that are also available separately as extensions. SPIR-V 1.4 shader modules do not need to enable those extensions with the `OpExtension` opcode, since they are integral parts of SPIR-V 1.4.

SPIR-V 1.4 introduces new floating-point execution mode capabilities, also available via `SPV_KHR_float_controls`. Implementations are not required to support all of these new capabilities; support can be queried using VkPhysicalDeviceFloatControlsPropertiesKHR from the `VK_KHR_shader_float_controls` extension.

### Promotion to Vulkan 1.2

All functionality in this extension is included in core Vulkan 1.2, with the KHR suffix omitted. The original type, enum and command names are still available as aliases of the core functionality.

### New Enum Constants

- `VK_KHR_SPIRV_1_4_EXTENSION_NAME`
- `VK_KHR_SPIRV_1_4_SPEC_VERSION`

### Issues

1. Should we have an extension specific to this SPIR-V version, or add a version-generic query for SPIR-V version? SPIR-V 1.4 does not need any other API changes.

**RESOLVED**: Just expose SPIR-V 1.4.

Most new SPIR-V versions introduce optionally-required capabilities or have implementation-defined limits, and would need more API and specification changes specific to that version to make them available in Vulkan. For example, to support the subgroup capabilities added in SPIR-V 1.3 required introducing VkPhysicalDeviceSubgroupProperties to allow querying the supported group operation categories, maximum supported subgroup size, etc. While we could expose the parts of a new SPIR-V version that do not need accompanying changes generically, we will still end up writing extensions specific to each version for the remaining parts. Thus the generic mechanism will not reduce future spec-writing effort. In addition, making it clear which parts of a future version are supported by the generic mechanism and which cannot be used without specific support would be difficult to get right ahead of time.

2. Can different stages of the same pipeline use shaders with different SPIR-V versions?

**RESOLVED**: Yes.

Mixing SPIR-V versions 1.0-1.3 in the same pipeline has not been disallowed, so it would be inconsistent to disallow mixing 1.4 with previous versions. SPIR-V 1.4 does not introduce anything that should cause new difficulties here.

3. Must Vulkan extensions corresponding to SPIR-V extensions that were promoted to core in 1.4 be enabled in order to use that functionality in a SPIR-V 1.4 module?

**RESOLVED**: No, with caveats.

The SPIR-V 1.4 module does not need to declare the SPIR-V extensions, since the functionality is now part of core, so there is no need to enable the Vulkan extension that allows SPIR-V modules to declare the SPIR-V extension. However, when the functionality that is now core in SPIR-V 1.4 is optionally supported, the query for support is provided by a Vulkan extension, and that query can only be used if the extension is enabled.

This applies to any SPIR-V version; specifically for SPIR-V 1.4 this only applies to the functionality from `SPV_KHR_float_controls`, which was made available in Vulkan by `VK_KHR_shader_float_controls`. Even though the extension was promoted in SPIR-V 1.4, the capabilities are still optional in implementations that support `VK_KHR_spirv_1_4`.

A SPIR-V 1.4 module does not need to enable `SPV_KHR_float_controls` in order to use the capabilities, so if the application has *a priori* knowledge that the implementation supports the capabilities, it does not need to enable `VK_KHR_shader_float_controls`. However, if it does not have this knowledge

and has to query for support at runtime, it must enable `VK_KHR_shader_float_controls` in order to use VkPhysicalDeviceFloatControlsPropertiesKHR.

### Version History

- Revision 1, 2019-04-01 (Jesse Hall)
  - Internal draft versions

## VK_KHR_surface

### Name String

`VK_KHR_surface`

### Extension Type

Instance extension

### Registered Extension Number

1

### Revision

25

### Ratification Status

Ratified

### Extension and Version Dependencies

None

### Contact

- James Jones [GitHub]cubanismo
- Ian Elliott [GitHub]ianelliottus

### Other Extension Metadata

### Last Modified Date

2016-08-25

### IP Status

No known IP claims.

### Contributors

- Patrick Doane, Blizzard
- Ian Elliott, LunarG
- Jesse Hall, Google
- James Jones, NVIDIA
- David Mao, AMD

- Norbert Nopper, Freescale
- Alon Or-bach, Samsung
- Daniel Rakos, AMD
- Graham Sellers, AMD
- Jeff Vigil, Qualcomm
- Chia-I Wu, LunarG
- Faith Ekstrand, Intel

### Description

The `VK_KHR_surface` extension is an instance extension. It introduces VkSurfaceKHR objects, which abstract native platform surface or window objects for use with Vulkan. It also provides a way to determine whether a queue family in a physical device supports presenting to particular surface.

Separate extensions for each platform provide the mechanisms for creating VkSurfaceKHR objects, but once created they may be used in this and other platform-independent extensions, in particular the `VK_KHR_swapchain` extension.

### New Object Types

- VkSurfaceKHR

### New Commands

- vkDestroySurfaceKHR
- vkGetPhysicalDeviceSurfaceCapabilitiesKHR
- vkGetPhysicalDeviceSurfaceFormatsKHR
- vkGetPhysicalDeviceSurfacePresentModesKHR
- vkGetPhysicalDeviceSurfaceSupportKHR

### New Structures

- VkSurfaceCapabilitiesKHR
- VkSurfaceFormatKHR

### New Enums

- VkColorSpaceKHR
- VkCompositeAlphaFlagBitsKHR
- VkPresentModeKHR
- VkSurfaceTransformFlagBitsKHR

### New Bitmasks

- VkCompositeAlphaFlagsKHR

### New Enum Constants

- `VK_KHR_SURFACE_EXTENSION_NAME`
- `VK_KHR_SURFACE_SPEC_VERSION`
- Extending VkObjectType:
  - `VK_OBJECT_TYPE_SURFACE_KHR`
- Extending VkResult:
  - `VK_ERROR_NATIVE_WINDOW_IN_USE_KHR`
  - `VK_ERROR_SURFACE_LOST_KHR`

### Examples

| | |
|---|---|
| **NOTE** | The example code for the `VK_KHR_surface` and `VK_KHR_swapchain` extensions was removed from the appendix after revision 1.0.29. This WSI example code was ported to the cube demo that is shipped with the official Khronos SDK, and is being kept up-to-date in that location (see: https://github.com/KhronosGroup/Vulkan-Tools/blob/main/cube/cube.c). |

### Issues

1) Should this extension include a method to query whether a physical device supports presenting to a specific window or native surface on a given platform?

**RESOLVED**: Yes. Without this, applications would need to create a device instance to determine whether a particular window can be presented to. Knowing that a device supports presentation to a platform in general is not sufficient, as a single machine might support multiple seats, or instances of the platform that each use different underlying physical devices. Additionally, on some platforms, such as the X Window System, different drivers and devices might be used for different windows depending on which section of the desktop they exist on.

2) Should the vkGetPhysicalDeviceSurfaceCapabilitiesKHR, vkGetPhysicalDeviceSurfaceFormatsKHR, and vkGetPhysicalDeviceSurfacePresentModesKHR functions be in this extension and operate on physical devices, rather than being in `VK_KHR_swapchain` (i.e. device extension) and being dependent on VkDevice?

**RESOLVED**: Yes. While it might be useful to depend on `VkDevice` (and therefore on enabled extensions and features) for the queries, Vulkan was released only with the VkPhysicalDevice versions. Many cases can be resolved by a Valid Usage statement, and/or by a separate `pNext` chain version of the query struct specific to a given extension or parameters, via extensible versions of the queries: vkGetPhysicalDeviceSurfacePresentModes2EXT, vkGetPhysicalDeviceSurfaceCapabilities2KHR, and vkGetPhysicalDeviceSurfaceFormats2KHR.

3) Should Vulkan support Xlib or XCB as the API for accessing the X Window System platform?

**RESOLVED**: Both. XCB is a more modern and efficient API, but Xlib usage is deeply ingrained in many applications and likely will remain in use for the foreseeable future. Not all drivers necessarily need to support both, but including both as options in the core specification will probably encourage support, which should in turn ease adoption of the Vulkan API in older codebases. Additionally, the performance improvements possible with XCB likely will not have a measurable impact on the performance of Vulkan presentation and other minimal window system interactions defined here.

4) Should the GBM platform be included in the list of platform enums?

**RESOLVED**: Deferred, and will be addressed with a platform-specific extension to be written in the future.

**Version History**

- Revision 1, 2015-05-20 (James Jones)
  - Initial draft, based on LunarG KHR spec, other KHR specs, patches attached to bugs.
- Revision 2, 2015-05-22 (Ian Elliott)
  - Created initial Description section.
  - Removed query for whether a platform requires the use of a queue for presentation, since it was decided that presentation will always be modeled as being part of the queue.
  - Fixed typos and other minor mistakes.
- Revision 3, 2015-05-26 (Ian Elliott)
  - Improved the Description section.
- Revision 4, 2015-05-27 (James Jones)
  - Fixed compilation errors in example code.
- Revision 5, 2015-06-01 (James Jones)
  - Added issues 1 and 2 and made related spec updates.
- Revision 6, 2015-06-01 (James Jones)
  - Merged the platform type mappings table previously removed from VK_KHR_swapchain with the platform description table in this spec.
  - Added issues 3 and 4 documenting choices made when building the initial list of native platforms supported.
- Revision 7, 2015-06-11 (Ian Elliott)
  - Updated table 1 per input from the KHR TSG.
  - Updated issue 4 (GBM) per discussion with Daniel Stone. He will create a platform-specific extension sometime in the future.
- Revision 8, 2015-06-17 (James Jones)
  - Updated enum-extending values using new convention.
  - Fixed the value of VK_SURFACE_PLATFORM_INFO_TYPE_SUPPORTED_KHR.

- Revision 9, 2015-06-17 (James Jones)
  - Rebased on Vulkan API version 126.
- Revision 10, 2015-06-18 (James Jones)
  - Marked issues 2 and 3 resolved.
- Revision 11, 2015-06-23 (Ian Elliott)
  - Examples now show use of function pointers for extension functions.
  - Eliminated extraneous whitespace.
- Revision 12, 2015-07-07 (Daniel Rakos)
  - Added error section describing when each error is expected to be reported.
  - Replaced the term “queue node index” with “queue family index” in the spec as that is the agreed term to be used in the latest version of the core header and spec.
  - Replaced bool32_t with VkBool32.
- Revision 13, 2015-08-06 (Daniel Rakos)
  - Updated spec against latest core API header version.
- Revision 14, 2015-08-20 (Ian Elliott)
  - Renamed this extension and all of its enumerations, types, functions, etc. This makes it compliant with the proposed standard for Vulkan extensions.
  - Switched from “revision” to “version”, including use of the VK_MAKE_VERSION macro in the header file.
  - Did miscellaneous cleanup, etc.
- Revision 15, 2015-08-20 (Ian Elliott—porting a 2015-07-29 change from James Jones)
  - Moved the surface transform enums here from VK_WSI_swapchain so they could be reused by VK_WSI_display.
- Revision 16, 2015-09-01 (James Jones)
  - Restore single-field revision number.
- Revision 17, 2015-09-01 (James Jones)
  - Fix example code compilation errors.
- Revision 18, 2015-09-26 (Jesse Hall)
  - Replaced VkSurfaceDescriptionKHR with the VkSurfaceKHR object, which is created via layered extensions. Added VkDestroySurfaceKHR.
- Revision 19, 2015-09-28 (Jesse Hall)
  - Renamed from VK_EXT_KHR_swapchain to VK_EXT_KHR_surface.
- Revision 20, 2015-09-30 (Jeff Vigil)
  - Add error result VK_ERROR_SURFACE_LOST_KHR.
- Revision 21, 2015-10-15 (Daniel Rakos)
  - Updated the resolution of issue #2 and include the surface capability queries in this

extension.

  - Renamed SurfaceProperties to SurfaceCapabilities as it better reflects that the values returned are the capabilities of the surface on a particular device.
  - Other minor cleanup and consistency changes.
- Revision 22, 2015-10-26 (Ian Elliott)
  - Renamed from VK_EXT_KHR_surface to VK_KHR_surface.
- Revision 23, 2015-11-03 (Daniel Rakos)
  - Added allocation callbacks to vkDestroySurfaceKHR.
- Revision 24, 2015-11-10 (Jesse Hall)
  - Removed VkSurfaceTransformKHR. Use VkSurfaceTransformFlagBitsKHR instead.
  - Rename VkSurfaceCapabilitiesKHR member maxImageArraySize to maxImageArrayLayers.
- Revision 25, 2016-01-14 (James Jones)
  - Moved VK_ERROR_NATIVE_WINDOW_IN_USE_KHR from the VK_KHR_android_surface to the VK_KHR_surface extension.
- 2016-08-23 (Ian Elliott)
  - Update the example code, to not have so many characters per line, and to split out a new example to show how to obtain function pointers.
- 2016-08-25 (Ian Elliott)
  - A note was added at the beginning of the example code, stating that it will be removed from future versions of the appendix.

## VK_KHR_surface_protected_capabilities

### Name String

`VK_KHR_surface_protected_capabilities`

### Extension Type

Instance extension

### Registered Extension Number

240

### Revision

1

### Ratification Status

Ratified

### Extension and Version Dependencies

Vulkan Version 1.1
and
VK_KHR_get_surface_capabilities2

### Contact

- Sandeep Shinde [GitHub]sashinde

### Other Extension Metadata

### Last Modified Date

2018-12-18

### IP Status

No known IP claims.

### Contributors

- Sandeep Shinde, NVIDIA
- James Jones, NVIDIA
- Daniel Koch, NVIDIA

### Description

This extension extends VkSurfaceCapabilities2KHR, providing applications a way to query whether swapchains **can** be created with the `VK_SWAPCHAIN_CREATE_PROTECTED_BIT_KHR` flag set.

Vulkan 1.1 added (optional) support for protect memory and protected resources including buffers (`VK_BUFFER_CREATE_PROTECTED_BIT`), images (`VK_IMAGE_CREATE_PROTECTED_BIT`), and swapchains (`VK_SWAPCHAIN_CREATE_PROTECTED_BIT_KHR`). However, on implementations which support multiple windowing systems, not all window systems **may** be able to provide a protected display path.

This extension provides a way to query if a protected swapchain created for a surface (and thus a specific windowing system) **can** be displayed on screen. It extends the existing VkSurfaceCapabilities2KHR structure with a new VkSurfaceProtectedCapabilitiesKHR structure from which the application **can** obtain information about support for protected swapchain creation through vkGetPhysicalDeviceSurfaceCapabilities2KHR.

### New Structures

- Extending VkSurfaceCapabilities2KHR:
  - VkSurfaceProtectedCapabilitiesKHR

### New Enum Constants

- `VK_KHR_SURFACE_PROTECTED_CAPABILITIES_EXTENSION_NAME`
- `VK_KHR_SURFACE_PROTECTED_CAPABILITIES_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_SURFACE_PROTECTED_CAPABILITIES_KHR`

### Version History

- Revision 1, 2018-12-18 (Sandeep Shinde, Daniel Koch)

- Internal revisions.

## VK_KHR_swapchain

### Name String

`VK_KHR_swapchain`

### Extension Type

Device extension

### Registered Extension Number

2

### Revision

70

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_surface

### API Interactions

- Interacts with VK_VERSION_1_1

### Contact

- James Jones [GitHub]cubanismo
- Ian Elliott [GitHub]ianelliottus

### Other Extension Metadata

### Last Modified Date

2017-10-06

### IP Status

No known IP claims.

### Interactions and External Dependencies

- Interacts with Vulkan 1.1

### Contributors

- Patrick Doane, Blizzard
- Ian Elliott, LunarG
- Jesse Hall, Google
- Mathias Heyer, NVIDIA
- James Jones, NVIDIA
- David Mao, AMD

- Norbert Nopper, Freescale
- Alon Or-bach, Samsung
- Daniel Rakos, AMD
- Graham Sellers, AMD
- Jeff Vigil, Qualcomm
- Chia-I Wu, LunarG
- Faith Ekstrand, Intel
- Matthaeus G. Chajdas, AMD
- Ray Smith, ARM

#### Description

The `VK_KHR_swapchain` extension is the device-level companion to the `VK_KHR_surface` extension. It introduces VkSwapchainKHR objects, which provide the ability to present rendering results to a surface.

#### New Object Types

- VkSwapchainKHR

#### New Commands

- vkAcquireNextImageKHR
- vkCreateSwapchainKHR
- vkDestroySwapchainKHR
- vkGetSwapchainImagesKHR
- vkQueuePresentKHR

If Vulkan Version 1.1 is supported:

- vkAcquireNextImage2KHR
- vkGetDeviceGroupPresentCapabilitiesKHR
- vkGetDeviceGroupSurfacePresentModesKHR
- vkGetPhysicalDevicePresentRectanglesKHR

#### New Structures

- VkPresentInfoKHR
- VkSwapchainCreateInfoKHR

If Vulkan Version 1.1 is supported:

- VkAcquireNextImageInfoKHR

- VkDeviceGroupPresentCapabilitiesKHR
- Extending VkBindImageMemoryInfo:
  - VkBindImageMemorySwapchainInfoKHR
- Extending VkImageCreateInfo:
  - VkImageSwapchainCreateInfoKHR
- Extending VkPresentInfoKHR:
  - VkDeviceGroupPresentInfoKHR
- Extending VkSwapchainCreateInfoKHR:
  - VkDeviceGroupSwapchainCreateInfoKHR

**New Enums**

- VkSwapchainCreateFlagBitsKHR

If Vulkan Version 1.1 is supported:

- VkDeviceGroupPresentModeFlagBitsKHR

**New Bitmasks**

- VkSwapchainCreateFlagsKHR

If Vulkan Version 1.1 is supported:

- VkDeviceGroupPresentModeFlagsKHR

**New Enum Constants**

- `VK_KHR_SWAPCHAIN_EXTENSION_NAME`
- `VK_KHR_SWAPCHAIN_SPEC_VERSION`
- Extending VkImageLayout:
  - `VK_IMAGE_LAYOUT_PRESENT_SRC_KHR`
- Extending VkObjectType:
  - `VK_OBJECT_TYPE_SWAPCHAIN_KHR`
- Extending VkResult:
  - `VK_ERROR_OUT_OF_DATE_KHR`
  - `VK_SUBOPTIMAL_KHR`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PRESENT_INFO_KHR`
  - `VK_STRUCTURE_TYPE_SWAPCHAIN_CREATE_INFO_KHR`

If Vulkan Version 1.1 is supported:

- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_ACQUIRE_NEXT_IMAGE_INFO_KHR`
  - `VK_STRUCTURE_TYPE_BIND_IMAGE_MEMORY_SWAPCHAIN_INFO_KHR`
  - `VK_STRUCTURE_TYPE_DEVICE_GROUP_PRESENT_CAPABILITIES_KHR`
  - `VK_STRUCTURE_TYPE_DEVICE_GROUP_PRESENT_INFO_KHR`
  - `VK_STRUCTURE_TYPE_DEVICE_GROUP_SWAPCHAIN_CREATE_INFO_KHR`
  - `VK_STRUCTURE_TYPE_IMAGE_SWAPCHAIN_CREATE_INFO_KHR`
- Extending VkSwapchainCreateFlagBitsKHR:
  - `VK_SWAPCHAIN_CREATE_PROTECTED_BIT_KHR`
  - `VK_SWAPCHAIN_CREATE_SPLIT_INSTANCE_BIND_REGIONS_BIT_KHR`

**Issues**

1) Does this extension allow the application to specify the memory backing of the presentable images?

**RESOLVED**: No. Unlike standard images, the implementation will allocate the memory backing of the presentable image.

2) What operations are allowed on presentable images?

**RESOLVED**: This is determined by the image usage flags specified when creating the presentable image's swapchain.

3) Does this extension support MSAA presentable images?

**RESOLVED**: No. Presentable images are always single-sampled. Multi-sampled rendering must use regular images. To present the rendering results the application must manually resolve the multi-sampled image to a single-sampled presentable image prior to presentation.

4) Does this extension support stereo/multi-view presentable images?

**RESOLVED**: Yes. The number of views associated with a presentable image is determined by the `imageArrayLayers` specified when creating a swapchain. All presentable images in a given swapchain use the same array size.

5) Are the layers of stereo presentable images half-sized?

**RESOLVED**: No. The image extents always match those requested by the application.

6) Do the "present" and "acquire next image" commands operate on a queue? If not, do they need to include explicit semaphore objects to interlock them with queue operations?

**RESOLVED**: The present command operates on a queue. The image ownership operation it represents happens in order with other operations on the queue, so no explicit semaphore object is required to synchronize its actions.

Applications may want to acquire the next image in separate threads from those in which they manage their queue, or in multiple threads. To make such usage easier, the acquire next image command takes a semaphore to signal as a method of explicit synchronization. The application must later queue a wait for this semaphore before queuing execution of any commands using the image.

7) Does vkAcquireNextImageKHR block if no images are available?

**RESOLVED**: The command takes a timeout parameter. Special values for the timeout are 0, which makes the call a non-blocking operation, and `UINT64_MAX`, which blocks indefinitely. Values in between will block for up to the specified time. The call will return when an image becomes available or an error occurs. It may, but is not required to, return before the specified timeout expires if the swapchain becomes out of date.

8) Can multiple presents be queued using one vkQueuePresentKHR call?

**RESOLVED**: Yes. VkPresentInfoKHR contains a list of swapchains and corresponding image indices that will be presented. When supported, all presentations queued with a single vkQueuePresentKHR call will be applied atomically as one operation. The same swapchain must not appear in the list more than once. Later extensions may provide applications stronger guarantees of atomicity for such present operations, and/or allow them to query whether atomic presentation of a particular group of swapchains is possible.

9) How do the presentation and acquire next image functions notify the application the targeted surface has changed?

**RESOLVED**: Two new result codes are introduced for this purpose:

- `VK_SUBOPTIMAL_KHR` - Presentation will still succeed, subject to the window resize behavior, but the swapchain is no longer configured optimally for the surface it targets. Applications should query updated surface information and recreate their swapchain at the next convenient opportunity.
- `VK_ERROR_OUT_OF_DATE_KHR` - Failure. The swapchain is no longer compatible with the surface it targets. The application must query updated surface information and recreate the swapchain before presentation will succeed.

These can be returned by both vkAcquireNextImageKHR and vkQueuePresentKHR.

10) Does the vkAcquireNextImageKHR command return a semaphore to the application via an output parameter, or accept a semaphore to signal from the application as an object handle parameter?

**RESOLVED**: Accept a semaphore to signal as an object handle. This avoids the need to specify whether the application must destroy the semaphore or whether it is owned by the swapchain, and if the latter, what its lifetime is and whether it can be reused for other operations once it is received from vkAcquireNextImageKHR.

11) What types of swapchain queuing behavior should be exposed? Options include swap interval specification, mailbox/most recent vs. FIFO queue management, targeting specific vertical blank intervals or absolute times for a given present operation, and probably others. For some of these,

whether they are specified at swapchain creation time or as per-present parameters needs to be decided as well.

**RESOLVED**: The base swapchain extension will expose 3 possible behaviors (of which, FIFO will always be supported):

- Immediate present: Does not wait for vertical blanking period to update the current image, likely resulting in visible tearing. No internal queue is used. Present requests are applied immediately.
- Mailbox queue: Waits for the next vertical blanking period to update the current image. No tearing should be observed. An internal single-entry queue is used to hold pending presentation requests. If the queue is full when a new presentation request is received, the new request replaces the existing entry, and any images associated with the prior entry become available for reuse by the application.
- FIFO queue: Waits for the next vertical blanking period to update the current image. No tearing should be observed. An internal queue containing `numSwapchainImages` - 1 entries is used to hold pending presentation requests. New requests are appended to the end of the queue, and one request is removed from the beginning of the queue and processed during each vertical blanking period in which the queue is non-empty

Not all surfaces will support all of these modes, so the modes supported will be returned using a surface information query. All surfaces must support the FIFO queue mode. Applications must choose one of these modes up front when creating a swapchain. Switching modes can be accomplished by recreating the swapchain.

12) Can `VK_PRESENT_MODE_MAILBOX_KHR` provide non-blocking guarantees for vkAcquireNextImageKHR? If so, what is the proper criteria?

**RESOLVED**: Yes. The difficulty is not immediately obvious here. Naively, if at least 3 images are requested, mailbox mode should always have an image available for the application if the application does not own any images when the call to vkAcquireNextImageKHR was made. However, some presentation engines may have more than one “current” image, and would still need to block in some cases. The right requirement appears to be that if the application allocates the surface’s minimum number of images + 1 then it is guaranteed non-blocking behavior when it does not currently own any images.

13) Is there a way to create and initialize a new swapchain for a surface that has generated a `VK_SUBOPTIMAL_KHR` return code while still using the old swapchain?

**RESOLVED**: Not as part of this specification. This could be useful to allow the application to create an “optimal” replacement swapchain and rebuild all its command buffers using it in a background thread at a low priority while continuing to use the “suboptimal” swapchain in the main thread. It could probably use the same “atomic replace” semantics proposed for recreating direct-to-device swapchains without incurring a mode switch. However, after discussion, it was determined some platforms probably could not support concurrent swapchains for the same surface though, so this will be left out of the base KHR extensions. A future extension could add this for platforms where it is supported.

14) Should there be a special value for VkSurfaceCapabilitiesKHR::`maxImageCount` to indicate there

are no practical limits on the number of images in a swapchain?

**RESOLVED**: Yes. There will often be cases where there is no practical limit to the number of images in a swapchain other than the amount of available resources (i.e., memory) in the system. Trying to derive a hard limit from things like memory size is prone to failure. It is better in such cases to leave it to applications to figure such soft limits out via trial/failure iterations.

15) Should there be a special value for VkSurfaceCapabilitiesKHR::`currentExtent` to indicate the size of the platform surface is undefined?

**RESOLVED**: Yes. On some platforms (Wayland, for example), the surface size is defined by the images presented to it rather than the other way around.

16) Should there be a special value for VkSurfaceCapabilitiesKHR::`maxImageExtent` to indicate there is no practical limit on the surface size?

**RESOLVED**: No. It seems unlikely such a system would exist. 0 could be used to indicate the platform places no limits on the extents beyond those imposed by Vulkan for normal images, but this query could just as easily return those same limits, so a special "unlimited" value does not seem useful for this field.

17) How should surface rotation and mirroring be exposed to applications? How do they specify rotation and mirroring transforms applied prior to presentation?

**RESOLVED**: Applications can query both the supported and current transforms of a surface. Both are specified relative to the device's "natural" display rotation and direction. The supported transforms indicate which orientations the presentation engine accepts images in. For example, a presentation engine that does not support transforming surfaces as part of presentation, and which is presenting to a surface that is displayed with a 90-degree rotation, would return only one supported transform bit: `VK_SURFACE_TRANSFORM_ROTATE_90_BIT_KHR`. Applications must transform their rendering by the transform they specify when creating the swapchain in `preTransform` field.

18) Can surfaces ever not support `VK_MIRROR_NONE`? Can they support vertical and horizontal mirroring simultaneously? Relatedly, should `VK_MIRROR_NONE`[_BIT] be zero, or bit one, and should applications be allowed to specify multiple pre and current mirror transform bits, or exactly one?

**RESOLVED**: Since some platforms may not support presenting with a transform other than the native window's current transform, and prerotation/mirroring are specified relative to the device's natural rotation and direction, rather than relative to the surface's current rotation and direction, it is necessary to express lack of support for no mirroring. To allow this, the `MIRROR_NONE` enum must occupy a bit in the flags. Since `MIRROR_NONE` must be a bit in the bitmask rather than a bitmask with no values set, allowing more than one bit to be set in the bitmask would make it possible to describe undefined transforms such as `VK_MIRROR_NONE_BIT | VK_MIRROR_HORIZONTAL_BIT`, or a transform that includes both "no mirroring" and "horizontal mirroring" simultaneously. Therefore, it is desirable to allow specifying all supported mirroring transforms using only one bit. The question then becomes, should there be a `VK_MIRROR_HORIZONTAL_AND_VERTICAL_BIT` to represent a simultaneous horizontal and vertical mirror transform? However, such a transform is equivalent to a 180 degree rotation, so presentation engines and applications that wish to support or use such a transform can express it through rotation instead. Therefore, 3 exclusive bits are sufficient to express all needed mirroring transforms.

19) Should support for sRGB be required?

**RESOLVED**: In the advent of UHD and HDR display devices, proper color space information is vital to the display pipeline represented by the swapchain. The application can discover the supported format/color-space pairs and select a pair most suited to its rendering needs. Currently only the sRGB color space is supported, future extensions may provide support for more color spaces. See issues 23 and 24.

20) Is there a mechanism to modify or replace an existing swapchain with one targeting the same surface?

**RESOLVED**: Yes. This is described above in the text.

21) Should there be a way to set prerotation and mirroring using native APIs when presenting using a Vulkan swapchain?

**RESOLVED**: Yes. The transforms that can be expressed in this extension are a subset of those possible on native platforms. If a platform exposes a method to specify the transform of presented images for a given surface using native methods and exposes more transforms or other properties for surfaces than Vulkan supports, it might be impossible, difficult, or inconvenient to set some of those properties using Vulkan KHR extensions and some using the native interfaces. To avoid overwriting properties set using native commands when presenting using a Vulkan swapchain, the application can set the pretransform to “inherit”, in which case the current native properties will be used, or if none are available, a platform-specific default will be used. Platforms that do not specify a reasonable default or do not provide native mechanisms to specify such transforms should not include the inherit bits in the `supportedTransforms` bitmask they return in VkSurfaceCapabilitiesKHR.

22) Should the content of presentable images be clipped by objects obscuring their target surface?

**RESOLVED**: Applications can choose which behavior they prefer. Allowing the content to be clipped could enable more efficient presentation methods on some platforms, but some applications might rely on the content of presentable images to perform techniques such as partial updates or motion blurs.

23) What is the purpose of specifying a VkColorSpaceKHR along with VkFormat when creating a swapchain?

**RESOLVED**: While Vulkan itself is color space agnostic (e.g. even the meaning of R, G, B and A can be freely defined by the rendering application), the swapchain eventually will have to present the images on a display device with specific color reproduction characteristics. If any color space transformations are necessary before an image can be displayed, the color space of the presented image must be known to the swapchain. A swapchain will only support a restricted set of color format and -space pairs. This set can be discovered via vkGetPhysicalDeviceSurfaceFormatsKHR. As it can be expected that most display devices support the sRGB color space, at least one format/color-space pair has to be exposed, where the color space is `VK_COLOR_SPACE_SRGB_NONLINEAR_KHR`.

24) How are sRGB formats and the sRGB color space related?

**RESOLVED**: While Vulkan exposes a number of SRGB texture formats, using such formats does not

guarantee working in a specific color space. It merely means that the hardware can directly support applying the non-linear transfer functions defined by the sRGB standard color space when reading from or writing to images of those formats. Still, it is unlikely that a swapchain will expose a `*_SRGB` format along with any color space other than `VK_COLOR_SPACE_SRGB_NONLINEAR_KHR`.

On the other hand, non-`*_SRGB` formats will be very likely exposed in pair with a SRGB color space. This means, the hardware will not apply any transfer function when reading from or writing to such images, yet they will still be presented on a device with sRGB display characteristics. In this case the application is responsible for applying the transfer function, for instance by using shader math.

25) How are the lifetimes of surfaces and swapchains targeting them related?

**RESOLVED**: A surface must outlive any swapchains targeting it. A VkSurfaceKHR owns the binding of the native window to the Vulkan driver.

26) How can the application control the way the alpha component of swapchain images is treated by the presentation engine during compositing?

**RESOLVED**: We should add new enum values to allow the application to negotiate with the presentation engine on how to treat image alpha values during the compositing process. Since not all platforms can practically control this through the Vulkan driver, a value of `VK_COMPOSITE_ALPHA_INHERIT_BIT_KHR` is provided like for surface transforms.

27) Is vkCreateSwapchainKHR the right function to return `VK_ERROR_NATIVE_WINDOW_IN_USE_KHR`, or should the various platform-specific VkSurfaceKHR factory functions catch this error earlier?

**RESOLVED**: For most platforms, the VkSurfaceKHR structure is a simple container holding the data that identifies a native window or other object representing a surface on a particular platform. For the surface factory functions to return this error, they would likely need to register a reference on the native objects with the native display server somehow, and ensure no other such references exist. Surfaces were not intended to be that heavyweight.

Swapchains are intended to be the objects that directly manipulate native windows and communicate with the native presentation mechanisms. Swapchains will already need to communicate with the native display server to negotiate allocation and/or presentation of presentable images for a native surface. Therefore, it makes more sense for swapchain creation to be the point at which native object exclusivity is enforced. Platforms may choose to enforce further restrictions on the number of VkSurfaceKHR objects that may be created for the same native window if such a requirement makes sense on a particular platform, but a global requirement is only sensible at the swapchain level.

**Examples**

| | |
|---|---|
| **NOTE** | The example code for the `VK_KHR_surface` and `VK_KHR_swapchain` extensions was removed from the appendix after revision 1.0.29. This WSI example code was ported to the cube demo that is shipped with the official Khronos SDK, and is being kept up-to-date in that location (see: https://github.com/KhronosGroup/Vulkan-Tools/blob/main/cube/cube.c). |

**Version History**

- Revision 1, 2015-05-20 (James Jones)
  - Initial draft, based on LunarG KHR spec, other KHR specs, patches attached to bugs.
- Revision 2, 2015-05-22 (Ian Elliott)
  - Made many agreed-upon changes from 2015-05-21 KHR TSG meeting. This includes using only a queue for presentation, and having an explicit function to acquire the next image.
  - Fixed typos and other minor mistakes.
- Revision 3, 2015-05-26 (Ian Elliott)
  - Improved the Description section.
  - Added or resolved issues that were found in improving the Description. For example, pSurfaceDescription is used consistently, instead of sometimes using pSurface.
- Revision 4, 2015-05-27 (James Jones)
  - Fixed some grammatical errors and typos
  - Filled in the description of imageUseFlags when creating a swapchain.
  - Added a description of swapInterval.
  - Replaced the paragraph describing the order of operations on a queue for image ownership and presentation.
- Revision 5, 2015-05-27 (James Jones)
  - Imported relevant issues from the (abandoned) vk_wsi_persistent_swapchain_images extension.
  - Added issues 6 and 7, regarding behavior of the acquire next image and present commands with respect to queues.
  - Updated spec language and examples to align with proposed resolutions to issues 6 and 7.
- Revision 6, 2015-05-27 (James Jones)
  - Added issue 8, regarding atomic presentation of multiple swapchains
  - Updated spec language and examples to align with proposed resolution to issue 8.
- Revision 7, 2015-05-27 (James Jones)
  - Fixed compilation errors in example code, and made related spec fixes.
- Revision 8, 2015-05-27 (James Jones)
  - Added issue 9, and the related VK_SUBOPTIMAL_KHR result code.
  - Renamed VK_OUT_OF_DATE_KHR to VK_ERROR_OUT_OF_DATE_KHR.
- Revision 9, 2015-05-27 (James Jones)
  - Added inline proposed resolutions (marked with [JRJ]) to some XXX questions/issues. These should be moved to the issues section in a subsequent update if the proposals are adopted.
- Revision 10, 2015-05-28 (James Jones)
  - Converted vkAcquireNextImageKHR back to a non-queue operation that uses a

VkSemaphore object for explicit synchronization.
    - Added issue 10 to determine whether vkAcquireNextImageKHR generates or returns semaphores, or whether it operates on a semaphore provided by the application.
- Revision 11, 2015-05-28 (James Jones)
    - Marked issues 6, 7, and 8 resolved.
    - Renamed VkSurfaceCapabilityPropertiesKHR to VkSurfacePropertiesKHR to better convey the mutable nature of the information it contains.
- Revision 12, 2015-05-28 (James Jones)
    - Added issue 11 with a proposed resolution, and the related issue 12.
    - Updated various sections of the spec to match the proposed resolution to issue 11.
- Revision 13, 2015-06-01 (James Jones)
    - Moved some structures to VK_EXT_KHR_swap_chain to resolve the specification's issues 1 and 2.
- Revision 14, 2015-06-01 (James Jones)
    - Added code for example 4 demonstrating how an application might make use of the two different present and acquire next image KHR result codes.
    - Added issue 13.
- Revision 15, 2015-06-01 (James Jones)
    - Added issues 14 - 16 and related spec language.
    - Fixed some spelling errors.
    - Added language describing the meaningful return values for vkAcquireNextImageKHR and vkQueuePresentKHR.
- Revision 16, 2015-06-02 (James Jones)
    - Added issues 17 and 18, as well as related spec language.
    - Removed some erroneous text added by mistake in the last update.
- Revision 17, 2015-06-15 (Ian Elliott)
    - Changed special value from "-1" to "0" so that the data types can be unsigned.
- Revision 18, 2015-06-15 (Ian Elliott)
    - Clarified the values of VkSurfacePropertiesKHR::minImageCount and the timeout parameter of the vkAcquireNextImageKHR function.
- Revision 19, 2015-06-17 (James Jones)
    - Misc. cleanup. Removed resolved inline issues and fixed typos.
    - Fixed clarification of VkSurfacePropertiesKHR::minImageCount made in version 18.
    - Added a brief "Image Ownership" definition to the list of terms used in the spec.
- Revision 20, 2015-06-17 (James Jones)
    - Updated enum-extending values using new convention.

- Revision 21, 2015-06-17 (James Jones)
  - Added language describing how to use VK_IMAGE_LAYOUT_PRESENT_SOURCE_KHR.
  - Cleaned up an XXX comment regarding the description of which queues vkQueuePresentKHR can be used on.
- Revision 22, 2015-06-17 (James Jones)
  - Rebased on Vulkan API version 126.
- Revision 23, 2015-06-18 (James Jones)
  - Updated language for issue 12 to read as a proposed resolution.
  - Marked issues 11, 12, 13, 16, and 17 resolved.
  - Temporarily added links to the relevant bugs under the remaining unresolved issues.
  - Added issues 19 and 20 as well as proposed resolutions.
- Revision 24, 2015-06-19 (Ian Elliott)
  - Changed special value for VkSurfacePropertiesKHR::currentExtent back to “-1” from “0”. This value will never need to be unsigned, and “0” is actually a legal value.
- Revision 25, 2015-06-23 (Ian Elliott)
  - Examples now show use of function pointers for extension functions.
  - Eliminated extraneous whitespace.
- Revision 26, 2015-06-25 (Ian Elliott)
  - Resolved Issues 9 & 10 per KHR TSG meeting.
- Revision 27, 2015-06-25 (James Jones)
  - Added oldSwapchain member to VkSwapchainCreateInfoKHR.
- Revision 28, 2015-06-25 (James Jones)
  - Added the “inherit” bits to the rotation and mirroring flags and the associated issue 21.
- Revision 29, 2015-06-25 (James Jones)
  - Added the “clipped” flag to VkSwapchainCreateInfoKHR, and the associated issue 22.
  - Specified that presenting an image does not modify it.
- Revision 30, 2015-06-25 (James Jones)
  - Added language to the spec that clarifies the behavior of vkCreateSwapchainKHR() when the oldSwapchain field of VkSwapchainCreateInfoKHR is not NULL.
- Revision 31, 2015-06-26 (Ian Elliott)
  - Example of new VkSwapchainCreateInfoKHR members, “oldSwapchain” and “clipped”.
  - Example of using VkSurfacePropertiesKHR::{min|max}ImageCount to set VkSwapchainCreateInfoKHR::minImageCount.
  - Rename vkGetSurfaceInfoKHR()'s 4th parameter to “pDataSize”, for consistency with other functions.
  - Add macro with C-string name of extension (just to header file).

- Revision 32, 2015-06-26 (James Jones)
  - Minor adjustments to the language describing the behavior of “oldSwapchain”
  - Fixed the version date on my previous two updates.
- Revision 33, 2015-06-26 (Jesse Hall)
  - Add usage flags to VkSwapchainCreateInfoKHR
- Revision 34, 2015-06-26 (Ian Elliott)
  - Rename vkQueuePresentKHR()'s 2nd parameter to “pPresentInfo”, for consistency with other functions.
- Revision 35, 2015-06-26 (Faith Ekstrand)
  - Merged the VkRotationFlagBitsKHR and VkMirrorFlagBitsKHR enums into a single VkSurfaceTransformFlagBitsKHR enum.
- Revision 36, 2015-06-26 (Faith Ekstrand)
  - Added a VkSurfaceTransformKHR enum that is not a bitmask. Each value in VkSurfaceTransformKHR corresponds directly to one of the bits in VkSurfaceTransformFlagBitsKHR so transforming from one to the other is easy. Having a separate enum means that currentTransform and preTransform are now unambiguous by definition.
- Revision 37, 2015-06-29 (Ian Elliott)
  - Corrected one of the signatures of vkAcquireNextImageKHR, which had the last two parameters switched from what it is elsewhere in the specification and header files.
- Revision 38, 2015-06-30 (Ian Elliott)
  - Corrected a typo in description of the vkGetSwapchainInfoKHR() function.
  - Corrected a typo in header file comment for VkPresentInfoKHR::sType.
- Revision 39, 2015-07-07 (Daniel Rakos)
  - Added error section describing when each error is expected to be reported.
  - Replaced bool32_t with VkBool32.
- Revision 40, 2015-07-10 (Ian Elliott)
  - Updated to work with version 138 of the `vulkan.h` header. This includes declaring the VkSwapchainKHR type using the new VK_DEFINE_NONDISP_HANDLE macro, and no longer extending VkObjectType (which was eliminated).
- Revision 41 2015-07-09 (Mathias Heyer)
  - Added color space language.
- Revision 42, 2015-07-10 (Daniel Rakos)
  - Updated query mechanism to reflect the convention changes done in the core spec.
  - Removed “queue” from the name of VK_STRUCTURE_TYPE_QUEUE_PRESENT_INFO_KHR to be consistent with the established naming convention.
  - Removed reference to the no longer existing VkObjectType enum.

- Revision 43, 2015-07-17 (Daniel Rakos)
  - Added support for concurrent sharing of swapchain images across queue families.
  - Updated sample code based on recent changes
- Revision 44, 2015-07-27 (Ian Elliott)
  - Noted that support for VK_PRESENT_MODE_FIFO_KHR is required. That is ICDs may optionally support IMMEDIATE and MAILBOX, but must support FIFO.
- Revision 45, 2015-08-07 (Ian Elliott)
  - Corrected a typo in spec file (type and variable name had wrong case for the imageColorSpace member of the VkSwapchainCreateInfoKHR struct).
  - Corrected a typo in header file (last parameter in PFN_vkGetSurfacePropertiesKHR was missing “KHR” at the end of type: VkSurfacePropertiesKHR).
- Revision 46, 2015-08-20 (Ian Elliott)
  - Renamed this extension and all of its enumerations, types, functions, etc. This makes it compliant with the proposed standard for Vulkan extensions.
  - Switched from “revision” to “version”, including use of the VK_MAKE_VERSION macro in the header file.
  - Made improvements to several descriptions.
  - Changed the status of several issues from PROPOSED to RESOLVED, leaving no unresolved issues.
  - Resolved several TODOs, did miscellaneous cleanup, etc.
- Revision 47, 2015-08-20 (Ian Elliott—porting a 2015-07-29 change from James Jones)
  - Moved the surface transform enums to VK_WSI_swapchain so they could be reused by VK_WSI_display.
- Revision 48, 2015-09-01 (James Jones)
  - Various minor cleanups.
- Revision 49, 2015-09-01 (James Jones)
  - Restore single-field revision number.
- Revision 50, 2015-09-01 (James Jones)
  - Update Example #4 to include code that illustrates how to use the oldSwapchain field.
- Revision 51, 2015-09-01 (James Jones)
  - Fix example code compilation errors.
- Revision 52, 2015-09-08 (Matthaeus G. Chajdas)
  - Corrected a typo.
- Revision 53, 2015-09-10 (Alon Or-bach)
  - Removed underscore from SWAP_CHAIN left in VK_STRUCTURE_TYPE_SWAPCHAIN_CREATE_INFO_KHR.
- Revision 54, 2015-09-11 (Jesse Hall)

    - Described the execution and memory coherence requirements for image transitions to and from VK_IMAGE_LAYOUT_PRESENT_SOURCE_KHR.
- Revision 55, 2015-09-11 (Ray Smith)
    - Added errors for destroying and binding memory to presentable images
- Revision 56, 2015-09-18 (James Jones)
    - Added fence argument to vkAcquireNextImageKHR
    - Added example of how to meter a host thread based on presentation rate.
- Revision 57, 2015-09-26 (Jesse Hall)
    - Replace VkSurfaceDescriptionKHR with VkSurfaceKHR.
    - Added issue 25 with agreed resolution.
- Revision 58, 2015-09-28 (Jesse Hall)
    - Renamed from VK_EXT_KHR_device_swapchain to VK_EXT_KHR_swapchain.
- Revision 59, 2015-09-29 (Ian Elliott)
    - Changed vkDestroySwapchainKHR() to return void.
- Revision 60, 2015-10-01 (Jeff Vigil)
    - Added error result VK_ERROR_SURFACE_LOST_KHR.
- Revision 61, 2015-10-05 (Faith Ekstrand)
    - Added the VkCompositeAlpha enum and corresponding structure fields.
- Revision 62, 2015-10-12 (Daniel Rakos)
    - Added VK_PRESENT_MODE_FIFO_RELAXED_KHR.
- Revision 63, 2015-10-15 (Daniel Rakos)
    - Moved surface capability queries to VK_EXT_KHR_surface.
- Revision 64, 2015-10-26 (Ian Elliott)
    - Renamed from VK_EXT_KHR_swapchain to VK_KHR_swapchain.
- Revision 65, 2015-10-28 (Ian Elliott)
    - Added optional pResult member to VkPresentInfoKHR, so that per-swapchain results can be obtained from vkQueuePresentKHR().
- Revision 66, 2015-11-03 (Daniel Rakos)
    - Added allocation callbacks to create and destroy functions.
    - Updated resource transition language.
    - Updated sample code.
- Revision 67, 2015-11-10 (Jesse Hall)
    - Add reserved flags bitmask to VkSwapchainCreateInfoKHR.
    - Modify naming and member ordering to match API style conventions, and so the VkSwapchainCreateInfoKHR image property members mirror corresponding VkImageCreateInfo members but with an 'image' prefix.

    - Make VkPresentInfoKHR::pResults non-const; it is an output array parameter.
    - Make pPresentInfo parameter to vkQueuePresentKHR const.
- Revision 68, 2016-04-05 (Ian Elliott)
    - Moved the “validity” include for vkAcquireNextImage to be in its proper place, after the prototype and list of parameters.
    - Clarified language about presentable images, including how they are acquired, when applications can and cannot use them, etc. As part of this, removed language about “ownership” of presentable images, and replaced it with more-consistent language about presentable images being “acquired” by the application.
- 2016-08-23 (Ian Elliott)
    - Update the example code, to use the final API command names, to not have so many characters per line, and to split out a new example to show how to obtain function pointers. This code is more similar to the LunarG “cube” demo program.
- 2016-08-25 (Ian Elliott)
    - A note was added at the beginning of the example code, stating that it will be removed from future versions of the appendix.
- Revision 69, 2017-09-07 (Tobias Hector)
    - Added interactions with Vulkan 1.1
- Revision 70, 2017-10-06 (Ian Elliott)
    - Corrected interactions with Vulkan 1.1

## VK_KHR_swapchain_mutable_format

### Name String

`VK_KHR_swapchain_mutable_format`

### Extension Type

Device extension

### Registered Extension Number

201

### Revision

1

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_swapchain
and
  VK_KHR_maintenance2
  or

Vulkan Version 1.1

and

VK_KHR_image_format_list

or

Vulkan Version 1.2

### Contact

- Daniel Rakos [GitHub]drakos-amd

### Other Extension Metadata

### Last Modified Date

2018-03-28

### IP Status

No known IP claims.

### Contributors

- Faith Ekstrand, Intel
- Jan-Harald Fredriksen, ARM
- Jesse Hall, Google
- Daniel Rakos, AMD
- Ray Smith, ARM

### Description

This extension allows processing of swapchain images as different formats to that used by the window system, which is particularly useful for switching between sRGB and linear RGB formats.

It adds a new swapchain creation flag that enables creating image views from presentable images with a different format than the one used to create the swapchain.

### New Enum Constants

- `VK_KHR_SWAPCHAIN_MUTABLE_FORMAT_EXTENSION_NAME`
- `VK_KHR_SWAPCHAIN_MUTABLE_FORMAT_SPEC_VERSION`
- Extending VkSwapchainCreateFlagBitsKHR:
  - `VK_SWAPCHAIN_CREATE_MUTABLE_FORMAT_BIT_KHR`

### Issues

1) Are there any new capabilities needed?

**RESOLVED**: No. It is expected that all implementations exposing this extension support swapchain image format mutability.

2) Do we need a separate `VK_SWAPCHAIN_CREATE_EXTENDED_USAGE_BIT_KHR`?

**RESOLVED**: No. This extension requires `VK_KHR_maintenance2` and presentable images of swapchains created with `VK_SWAPCHAIN_CREATE_MUTABLE_FORMAT_BIT_KHR` are created internally in a way equivalent to specifying both `VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT` and `VK_IMAGE_CREATE_EXTENDED_USAGE_BIT_KHR`.

3) Do we need a separate structure to allow specifying an image format list for swapchains?

**RESOLVED**: No. We simply use the same VkImageFormatListCreateInfoKHR structure introduced by `VK_KHR_image_format_list`. The structure is required to be included in the `pNext` chain of VkSwapchainCreateInfoKHR for swapchains created with `VK_SWAPCHAIN_CREATE_MUTABLE_FORMAT_BIT_KHR`.

### Version History

- Revision 1, 2018-03-28 (Daniel Rakos)
  - Internal revisions.

## VK_KHR_synchronization2

### Name String

`VK_KHR_synchronization2`

### Extension Type

Device extension

### Registered Extension Number

315

### Revision

1

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### API Interactions

- Interacts with VK_AMD_buffer_marker
- Interacts with VK_EXT_blend_operation_advanced
- Interacts with VK_EXT_conditional_rendering
- Interacts with VK_EXT_fragment_density_map
- Interacts with VK_EXT_mesh_shader
- Interacts with VK_EXT_transform_feedback
- Interacts with VK_KHR_acceleration_structure

- Interacts with VK_KHR_fragment_shading_rate
- Interacts with VK_KHR_ray_tracing_pipeline
- Interacts with VK_NV_device_diagnostic_checkpoints
- Interacts with VK_NV_device_generated_commands
- Interacts with VK_NV_mesh_shader
- Interacts with VK_NV_ray_tracing
- Interacts with VK_NV_shading_rate_image

**Deprecation State**

- *Promoted* to Vulkan 1.3

**Contact**

- Tobias Hector [GitHub]tobski

**Other Extension Metadata**

**Last Modified Date**

2020-12-03

**Interactions and External Dependencies**

- Interacts with `VK_KHR_create_renderpass2`

**Contributors**

- Tobias Hector

**Description**

This extension modifies the original core synchronization APIs to simplify the interface and improve usability of these APIs. It also adds new pipeline stage and access flag types that extend into the 64-bit range, as we have run out within the 32-bit range. The new flags are identical to the old values within the 32-bit range, with new stages and bits beyond that.

Pipeline stages and access flags are now specified together in memory barrier structures, making the connection between the two more obvious. Additionally, scoping the pipeline stages into the barrier structs allows the use of the `MEMORY_READ` and `MEMORY_WRITE` flags without sacrificing precision. The per-stage access flags should be used to disambiguate specific accesses in a given stage or set of stages - for instance, between uniform reads and sampling operations.

Layout transitions have been simplified as well; rather than requiring a different set of layouts for depth/stencil/color attachments, there are generic `VK_IMAGE_LAYOUT_ATTACHMENT_OPTIMAL_KHR` and `VK_IMAGE_LAYOUT_READ_ONLY_OPTIMAL_KHR` layouts which are contextually applied based on the image format. For example, for a depth format image, `VK_IMAGE_LAYOUT_READ_ONLY_OPTIMAL_KHR` is equivalent to `VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL_KHR`. `VK_IMAGE_LAYOUT_READ_ONLY_OPTIMAL_KHR` also functionally replaces `VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL`.

Events are now more efficient, because they include memory dependency information when you set them on the device. Previously, this information was only known when waiting on an event, so the dependencies could not be satisfied until the wait occurred. That sometimes meant stalling the

pipeline when the wait occurred. The new API provides enough information for implementations to satisfy these dependencies in parallel with other tasks.

Queue submission has been changed to wrap command buffers and semaphores in extensible structures, which incorporate changes from Vulkan 1.1, `VK_KHR_device_group`, and `VK_KHR_timeline_semaphore`. This also adds a pipeline stage to the semaphore signal operation, mirroring the existing pipeline stage specification for wait operations.

Other miscellaneous changes include:

- Events can now be specified as interacting only with the device, allowing more efficient access to the underlying object.
- Image memory barriers that do not perform an image layout transition can be specified by setting `oldLayout` equal to `newLayout`.
  - E.g. the old and new layout can both be set to `VK_IMAGE_LAYOUT_UNDEFINED`, without discarding data in the image.
- Queue family ownership transfer parameters are simplified in some cases.
- Extensions with commands or functions with a VkPipelineStageFlags or VkPipelineStageFlagBits parameter have had those APIs replaced with equivalents using VkPipelineStageFlags2KHR.
- The new event and barrier interfaces are now more extensible for future changes.
- Relevant pipeline stage masks can now be specified as empty with the new `VK_PIPELINE_STAGE_NONE_KHR` and `VK_PIPELINE_STAGE_2_NONE_KHR` values.
- VkMemoryBarrier2KHR can be chained to VkSubpassDependency2, overriding the original 32-bit stage and access masks.

**New Base Types**

- VkFlags64

**New Commands**

- vkCmdPipelineBarrier2KHR
- vkCmdResetEvent2KHR
- vkCmdSetEvent2KHR
- vkCmdWaitEvents2KHR
- vkCmdWriteTimestamp2KHR
- vkQueueSubmit2KHR

If VK_AMD_buffer_marker is supported:

- vkCmdWriteBufferMarker2AMD

If VK_NV_device_diagnostic_checkpoints is supported:

- vkGetQueueCheckpointData2NV

### New Structures

- VkBufferMemoryBarrier2KHR
- VkCommandBufferSubmitInfoKHR
- VkDependencyInfoKHR
- VkImageMemoryBarrier2KHR
- VkSemaphoreSubmitInfoKHR
- VkSubmitInfo2KHR
- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceSynchronization2FeaturesKHR
- Extending VkSubpassDependency2:
  - VkMemoryBarrier2KHR

If VK_NV_device_diagnostic_checkpoints is supported:

- VkCheckpointData2NV
- Extending VkQueueFamilyProperties2:
  - VkQueueFamilyCheckpointProperties2NV

### New Enums

- VkAccessFlagBits2KHR
- VkPipelineStageFlagBits2KHR
- VkSubmitFlagBitsKHR

### New Bitmasks

- VkAccessFlags2KHR
- VkPipelineStageFlags2KHR
- VkSubmitFlagsKHR

### New Enum Constants

- `VK_KHR_SYNCHRONIZATION_2_EXTENSION_NAME`
- `VK_KHR_SYNCHRONIZATION_2_SPEC_VERSION`
- Extending VkAccessFlagBits:
  - `VK_ACCESS_NONE_KHR`
- Extending VkEventCreateFlagBits:
  - `VK_EVENT_CREATE_DEVICE_ONLY_BIT_KHR`
- Extending VkImageLayout:
  - `VK_IMAGE_LAYOUT_ATTACHMENT_OPTIMAL_KHR`

    - VK_IMAGE_LAYOUT_READ_ONLY_OPTIMAL_KHR
- Extending VkPipelineStageFlagBits:
    - VK_PIPELINE_STAGE_NONE_KHR
- Extending VkStructureType:
    - VK_STRUCTURE_TYPE_BUFFER_MEMORY_BARRIER_2_KHR
    - VK_STRUCTURE_TYPE_COMMAND_BUFFER_SUBMIT_INFO_KHR
    - VK_STRUCTURE_TYPE_DEPENDENCY_INFO_KHR
    - VK_STRUCTURE_TYPE_IMAGE_MEMORY_BARRIER_2_KHR
    - VK_STRUCTURE_TYPE_MEMORY_BARRIER_2_KHR
    - VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SYNCHRONIZATION_2_FEATURES_KHR
    - VK_STRUCTURE_TYPE_SEMAPHORE_SUBMIT_INFO_KHR
    - VK_STRUCTURE_TYPE_SUBMIT_INFO_2_KHR

If VK_EXT_blend_operation_advanced is supported:

- Extending VkAccessFlagBits2:
    - VK_ACCESS_2_COLOR_ATTACHMENT_READ_NONCOHERENT_BIT_EXT

If VK_EXT_conditional_rendering is supported:

- Extending VkAccessFlagBits2:
    - VK_ACCESS_2_CONDITIONAL_RENDERING_READ_BIT_EXT
- Extending VkPipelineStageFlagBits2:
    - VK_PIPELINE_STAGE_2_CONDITIONAL_RENDERING_BIT_EXT

If VK_EXT_fragment_density_map is supported:

- Extending VkAccessFlagBits2:
    - VK_ACCESS_2_FRAGMENT_DENSITY_MAP_READ_BIT_EXT
- Extending VkPipelineStageFlagBits2:
    - VK_PIPELINE_STAGE_2_FRAGMENT_DENSITY_PROCESS_BIT_EXT

If VK_EXT_mesh_shader is supported:

- Extending VkPipelineStageFlagBits2:
    - VK_PIPELINE_STAGE_2_MESH_SHADER_BIT_EXT
    - VK_PIPELINE_STAGE_2_TASK_SHADER_BIT_EXT

If VK_EXT_transform_feedback is supported:

- Extending VkAccessFlagBits2:
    - VK_ACCESS_2_TRANSFORM_FEEDBACK_COUNTER_READ_BIT_EXT

    - VK_ACCESS_2_TRANSFORM_FEEDBACK_COUNTER_WRITE_BIT_EXT
    - VK_ACCESS_2_TRANSFORM_FEEDBACK_WRITE_BIT_EXT
- Extending VkPipelineStageFlagBits2:
    - VK_PIPELINE_STAGE_2_TRANSFORM_FEEDBACK_BIT_EXT

If VK_KHR_acceleration_structure is supported:

- Extending VkAccessFlagBits2:
    - VK_ACCESS_2_ACCELERATION_STRUCTURE_READ_BIT_KHR
    - VK_ACCESS_2_ACCELERATION_STRUCTURE_WRITE_BIT_KHR
- Extending VkPipelineStageFlagBits2:
    - VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_BUILD_BIT_KHR

If VK_KHR_fragment_shading_rate is supported:

- Extending VkAccessFlagBits2:
    - VK_ACCESS_2_FRAGMENT_SHADING_RATE_ATTACHMENT_READ_BIT_KHR
- Extending VkPipelineStageFlagBits2:
    - VK_PIPELINE_STAGE_2_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR

If VK_KHR_ray_tracing_pipeline is supported:

- Extending VkPipelineStageFlagBits2:
    - VK_PIPELINE_STAGE_2_RAY_TRACING_SHADER_BIT_KHR

If VK_NV_device_diagnostic_checkpoints is supported:

- Extending VkStructureType:
    - VK_STRUCTURE_TYPE_CHECKPOINT_DATA_2_NV
    - VK_STRUCTURE_TYPE_QUEUE_FAMILY_CHECKPOINT_PROPERTIES_2_NV

If VK_NV_device_generated_commands is supported:

- Extending VkAccessFlagBits2:
    - VK_ACCESS_2_COMMAND_PREPROCESS_READ_BIT_NV
    - VK_ACCESS_2_COMMAND_PREPROCESS_WRITE_BIT_NV
- Extending VkPipelineStageFlagBits2:
    - VK_PIPELINE_STAGE_2_COMMAND_PREPROCESS_BIT_NV

If VK_NV_mesh_shader is supported:

- Extending VkPipelineStageFlagBits2:
    - VK_PIPELINE_STAGE_2_MESH_SHADER_BIT_NV

- `VK_PIPELINE_STAGE_2_TASK_SHADER_BIT_NV`

If VK_NV_ray_tracing is supported:

- Extending VkAccessFlagBits2:
  - `VK_ACCESS_2_ACCELERATION_STRUCTURE_READ_BIT_NV`
  - `VK_ACCESS_2_ACCELERATION_STRUCTURE_WRITE_BIT_NV`
- Extending VkPipelineStageFlagBits2:
  - `VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_BUILD_BIT_NV`
  - `VK_PIPELINE_STAGE_2_RAY_TRACING_SHADER_BIT_NV`

If VK_NV_shading_rate_image is supported:

- Extending VkAccessFlagBits2:
  - `VK_ACCESS_2_SHADING_RATE_IMAGE_READ_BIT_NV`
- Extending VkPipelineStageFlagBits2:
  - `VK_PIPELINE_STAGE_2_SHADING_RATE_IMAGE_BIT_NV`

#### Promotion to Vulkan 1.3

Functionality in this extension is included in core Vulkan 1.3, with the KHR suffix omitted. The original type, enum and command names are still available as aliases of the core functionality.

#### Examples

See https://github.com/KhronosGroup/Vulkan-Docs/wiki/Synchronization-Examples

#### Version History

- Revision 1, 2020-12-03 (Tobias Hector)
  - Internal revisions

## VK_KHR_timeline_semaphore

#### Name String

`VK_KHR_timeline_semaphore`

#### Extension Type

Device extension

#### Registered Extension Number

208

#### Revision

2

**Ratification Status**

Ratified

**Extension and Version Dependencies**

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

**Deprecation State**

- *Promoted* to Vulkan 1.2

**Contact**

- Faith Ekstrand [GitHub]gfxstrand

**Other Extension Metadata**

**Last Modified Date**

2019-06-12

**IP Status**

No known IP claims.

**Interactions and External Dependencies**

- This extension interacts with `VK_KHR_external_semaphore_capabilities`
- This extension interacts with `VK_KHR_external_semaphore`
- This extension interacts with `VK_KHR_external_semaphore_win32`

**Contributors**

- Jeff Bolz, NVIDIA
- Yuriy O'Donnell, Epic Games
- Faith Ekstrand, Intel
- Jesse Hall, Google
- James Jones, NVIDIA
- Jeff Juliano, NVIDIA
- Daniel Rakos, AMD
- Ray Smith, Arm

**Description**

This extension introduces a new type of semaphore that has an integer payload identifying a point in a timeline. Such timeline semaphores support the following operations:

- Host query - A host operation that allows querying the payload of the timeline semaphore.
- Host wait - A host operation that allows a blocking wait for a timeline semaphore to reach a specified value.

- Host signal - A host operation that allows advancing the timeline semaphore to a specified value.
- Device wait - A device operation that allows waiting for a timeline semaphore to reach a specified value.
- Device signal - A device operation that allows advancing the timeline semaphore to a specified value.

#### Promotion to Vulkan 1.2

All functionality in this extension is included in core Vulkan 1.2, with the KHR suffix omitted. The original type, enum and command names are still available as aliases of the core functionality.

#### New Commands

- vkGetSemaphoreCounterValueKHR
- vkSignalSemaphoreKHR
- vkWaitSemaphoresKHR

#### New Structures

- VkSemaphoreSignalInfoKHR
- VkSemaphoreWaitInfoKHR
- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceTimelineSemaphoreFeaturesKHR
- Extending VkPhysicalDeviceProperties2:
  - VkPhysicalDeviceTimelineSemaphorePropertiesKHR
- Extending VkSemaphoreCreateInfo, VkPhysicalDeviceExternalSemaphoreInfo:
  - VkSemaphoreTypeCreateInfoKHR
- Extending VkSubmitInfo, VkBindSparseInfo:
  - VkTimelineSemaphoreSubmitInfoKHR

#### New Enums

- VkSemaphoreTypeKHR
- VkSemaphoreWaitFlagBitsKHR

#### New Bitmasks

- VkSemaphoreWaitFlagsKHR

#### New Enum Constants

- `VK_KHR_TIMELINE_SEMAPHORE_EXTENSION_NAME`
- `VK_KHR_TIMELINE_SEMAPHORE_SPEC_VERSION`

- Extending VkSemaphoreType:
  - VK_SEMAPHORE_TYPE_BINARY_KHR
  - VK_SEMAPHORE_TYPE_TIMELINE_KHR
- Extending VkSemaphoreWaitFlagBits:
  - VK_SEMAPHORE_WAIT_ANY_BIT_KHR
- Extending VkStructureType:
  - VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_TIMELINE_SEMAPHORE_FEATURES_KHR
  - VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_TIMELINE_SEMAPHORE_PROPERTIES_KHR
  - VK_STRUCTURE_TYPE_SEMAPHORE_SIGNAL_INFO_KHR
  - VK_STRUCTURE_TYPE_SEMAPHORE_TYPE_CREATE_INFO_KHR
  - VK_STRUCTURE_TYPE_SEMAPHORE_WAIT_INFO_KHR
  - VK_STRUCTURE_TYPE_TIMELINE_SEMAPHORE_SUBMIT_INFO_KHR

**Issues**

1) Do we need a new object type for this?

**RESOLVED**: No, we just introduce a new type of semaphore object, as VK_KHR_external_semaphore_win32 already uses semaphores as the destination for importing D3D12 fence objects, which are semantically close/identical to the proposed synchronization primitive.

2) What type of payload the new synchronization primitive has?

**RESOLVED**: A 64-bit unsigned integer that can only be set to strictly increasing values by signal operations and is not changed by wait operations.

3) Does the new synchronization primitive have the same signal-before-wait requirement as the existing semaphores do?

**RESOLVED**: No. Timeline semaphores support signaling and waiting entirely asynchronously. It is the responsibility of the application to avoid deadlock.

4) Does the new synchronization primitive allow resetting its payload?

**RESOLVED**: No, allowing the payload value to “go backwards” is problematic. Applications looking for reset behavior should create a new instance of the synchronization primitive instead.

5) How do we enable host waits on the synchronization primitive?

**RESOLVED**: Both a non-blocking query of the current payload value of the synchronization primitive, and a blocking wait operation are provided.

6) How do we enable device waits and signals on the synchronization primitive?

**RESOLVED**: Similar to VK_KHR_external_semaphore_win32, this extension introduces a new structure that can be chained to VkSubmitInfo to specify the values signaled semaphores should be set to,

and the values waited semaphores need to reach.

7) Can the new synchronization primitive be used to synchronize presentation and swapchain image acquisition operations?

**RESOLVED**: Some implementations may have problems with supporting that directly, thus it is not allowed in this extension.

8) Do we want to support external sharing of the new synchronization primitive type?

**RESOLVED**: Yes. Timeline semaphore specific external sharing capabilities can be queried using vkGetPhysicalDeviceExternalSemaphoreProperties by chaining the new VkSemaphoreTypeCreateInfoKHR structure to its `pExternalSemaphoreInfo` structure. This allows having a different set of external semaphore handle types supported for timeline semaphores vs. binary semaphores.

9) Do we need to add a host signal operation for the new synchronization primitive type?

**RESOLVED**: Yes. This helps in situations where one host thread submits a workload but another host thread has the information on when the workload is ready to be executed.

10) How should the new synchronization primitive interact with the ordering requirements of the original `VkSemaphore`?

**RESOLVED**: Prior to calling any command which **may** cause a wait operation on a binary semaphore, the application **must** ensure that the semaphore signal operation that has been submitted for execution and any semaphore signal operations on which it depends (if any) **must** have also been submitted for execution.

11) Should we have separate feature bits for different sub-features of timeline semaphores?

**RESOLVED**: No. The only feature which cannot be supported universally is timeline semaphore import/export. For import/export, the application is already required to query available external handle types via vkGetPhysicalDeviceExternalSemaphoreProperties and provide the semaphore type by adding a VkSemaphoreTypeCreateInfoKHR structure to the `pNext` chain of VkPhysicalDeviceExternalSemaphoreInfo so no new feature bit is required.

### Version History

- Revision 1, 2018-05-10 (Faith Ekstrand)
  - Initial version
- Revision 2, 2019-06-12 (Faith Ekstrand)
  - Added an initialValue parameter to timeline semaphore creation

## VK_KHR_uniform_buffer_standard_layout

### Name String

`VK_KHR_uniform_buffer_standard_layout`

### Extension Type

Device extension

### Registered Extension Number

254

### Revision

1

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### Deprecation State

- *Promoted* to Vulkan 1.2

### Contact

- Graeme Leese [GitHub]gnl21

### Other Extension Metadata

### Last Modified Date

2019-01-25

### Contributors

- Graeme Leese, Broadcom
- Jeff Bolz, NVIDIA
- Tobias Hector, AMD
- Faith Ekstrand, Intel
- Neil Henning, AMD

### Description

This extension enables tighter array and struct packing to be used with uniform buffers.

It modifies the alignment rules for uniform buffers, allowing for tighter packing of arrays and structures. This allows, for example, the std430 layout, as defined in GLSL to be supported in uniform buffers.

### Promotion to Vulkan 1.2

All functionality in this extension is included in core Vulkan 1.2, with the KHR suffix omitted. The original type, enum and command names are still available as aliases of the core functionality.

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceUniformBufferStandardLayoutFeaturesKHR

### New Enum Constants

- `VK_KHR_UNIFORM_BUFFER_STANDARD_LAYOUT_EXTENSION_NAME`
- `VK_KHR_UNIFORM_BUFFER_STANDARD_LAYOUT_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_UNIFORM_BUFFER_STANDARD_LAYOUT_FEATURES_KHR`

### Version History

- Revision 1, 2019-01-25 (Graeme Leese)
  - Initial draft

## VK_KHR_vertex_attribute_divisor

### Name String

`VK_KHR_vertex_attribute_divisor`

### Extension Type

Device extension

### Registered Extension Number

526

### Revision

1

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### Contact

- Shahbaz Youssefi [GitHub]syoussefi

### Extension Proposal

VK_KHR_vertex_attribute_divisor

### Other Extension Metadata

#### Last Modified Date

2023-09-20

#### IP Status

No known IP claims.

#### Contributors

- Shahbaz Youssefi, Google
- Contributors to `VK_EXT_vertex_attribute_divisor`

#### Description

This extension is based on the `VK_EXT_vertex_attribute_divisor` extension. The only difference is the new property `supportsNonZeroFirstInstance`, which indicates support for non-zero values in `firstInstance`. This allows the extension to be supported on implementations that have traditionally only supported OpenGL ES.

#### New Structures

- VkVertexInputBindingDivisorDescriptionKHR
- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceVertexAttributeDivisorFeaturesKHR
- Extending VkPhysicalDeviceProperties2:
  - VkPhysicalDeviceVertexAttributeDivisorPropertiesKHR
- Extending VkPipelineVertexInputStateCreateInfo:
  - VkPipelineVertexInputDivisorStateCreateInfoKHR

#### New Enum Constants

- `VK_KHR_VERTEX_ATTRIBUTE_DIVISOR_EXTENSION_NAME`
- `VK_KHR_VERTEX_ATTRIBUTE_DIVISOR_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_VERTEX_ATTRIBUTE_DIVISOR_FEATURES_KHR`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_VERTEX_ATTRIBUTE_DIVISOR_PROPERTIES_KHR`
  - `VK_STRUCTURE_TYPE_PIPELINE_VERTEX_INPUT_DIVISOR_STATE_CREATE_INFO_KHR`

#### Version History

- Revision 1, 2023-09-20 (Shahbaz Youssefi)
  - First Version, based on `VK_EXT_vertex_attribute_divisor`

## VK_KHR_video_decode_av1

**Name String**

`VK_KHR_video_decode_av1`

**Extension Type**

Device extension

**Registered Extension Number**

513

**Revision**

1

**Ratification Status**

Ratified

**Extension and Version Dependencies**

VK_KHR_video_decode_queue

**Contact**

- Daniel Rakos [GitHub]aqnuep

**Extension Proposal**

VK_KHR_video_decode_av1

**Other Extension Metadata**

**Last Modified Date**

2024-01-02

**IP Status**

No known IP claims.

**Contributors**

- Ahmed Abdelkhalek, AMD
- Benjamin Cheng, AMD
- Ho Hin Lau, AMD
- Lynne Iribarren, Independent
- David Airlie, Red Hat, Inc.
- Ping Liu, Intel
- Srinath Kumarapuram, NVIDIA
- Vassili Nikolaev, NVIDIA
- Tony Zlatinski, NVIDIA
- Charlie Turner, Igalia
- Daniel Almeida, Collabora

- Nicolas Dufresne, Collabora
- Daniel Rakos, RasterGrid

### Description

This extension builds upon the `VK_KHR_video_decode_queue` extension by adding support for decoding elementary video stream sequences compliant with the AV1 video compression standard.

### New Structures

- Extending VkVideoCapabilitiesKHR:
  - VkVideoDecodeAV1CapabilitiesKHR
- Extending VkVideoDecodeInfoKHR:
  - VkVideoDecodeAV1PictureInfoKHR
- Extending VkVideoProfileInfoKHR, VkQueryPoolCreateInfo:
  - VkVideoDecodeAV1ProfileInfoKHR
- Extending VkVideoReferenceSlotInfoKHR:
  - VkVideoDecodeAV1DpbSlotInfoKHR
- Extending VkVideoSessionParametersCreateInfoKHR:
  - VkVideoDecodeAV1SessionParametersCreateInfoKHR

### New Enum Constants

- `VK_KHR_VIDEO_DECODE_AV1_EXTENSION_NAME`
- `VK_KHR_VIDEO_DECODE_AV1_SPEC_VERSION`
- `VK_MAX_VIDEO_AV1_REFERENCES_PER_FRAME_KHR`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_VIDEO_DECODE_AV1_CAPABILITIES_KHR`
  - `VK_STRUCTURE_TYPE_VIDEO_DECODE_AV1_DPB_SLOT_INFO_KHR`
  - `VK_STRUCTURE_TYPE_VIDEO_DECODE_AV1_PICTURE_INFO_KHR`
  - `VK_STRUCTURE_TYPE_VIDEO_DECODE_AV1_PROFILE_INFO_KHR`
  - `VK_STRUCTURE_TYPE_VIDEO_DECODE_AV1_SESSION_PARAMETERS_CREATE_INFO_KHR`
- Extending VkVideoCodecOperationFlagBitsKHR:
  - `VK_VIDEO_CODEC_OPERATION_DECODE_AV1_BIT_KHR`

### Version History

- Revision 1, 2024-01-02 (Daniel Rakos)
  - Internal revisions

## VK_KHR_video_decode_h264

### Name String

`VK_KHR_video_decode_h264`

### Extension Type

Device extension

### Registered Extension Number

41

### Revision

9

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_video_decode_queue

### Contact

- peter.fang@amd.com

### Extension Proposal

VK_KHR_video_decode_h264

### Other Extension Metadata

### Last Modified Date

2023-12-05

### IP Status

No known IP claims.

### Contributors

- Ahmed Abdelkhalek, AMD
- Chunbo Chen, Intel
- HoHin Lau, AMD
- Jake Beju, AMD
- Peter Fang, AMD
- Ping Liu, Intel
- Srinath Kumarapuram, NVIDIA
- Tony Zlatinski, NVIDIA
- Daniel Rakos, RasterGrid

### Description

This extension builds upon the `VK_KHR_video_decode_queue` extension by adding support for decoding elementary video stream sequences compliant with the H.264/AVC video compression standard.

| | |
|---|---|
| **NOTE** | This extension was promoted to `KHR` from the provisional extension `VK_EXT_video_decode_h264`. |

### New Structures

- Extending VkVideoCapabilitiesKHR:
  - VkVideoDecodeH264CapabilitiesKHR
- Extending VkVideoDecodeInfoKHR:
  - VkVideoDecodeH264PictureInfoKHR
- Extending VkVideoProfileInfoKHR, VkQueryPoolCreateInfo:
  - VkVideoDecodeH264ProfileInfoKHR
- Extending VkVideoReferenceSlotInfoKHR:
  - VkVideoDecodeH264DpbSlotInfoKHR
- Extending VkVideoSessionParametersCreateInfoKHR:
  - VkVideoDecodeH264SessionParametersCreateInfoKHR
- Extending VkVideoSessionParametersUpdateInfoKHR:
  - VkVideoDecodeH264SessionParametersAddInfoKHR

### New Enums

- VkVideoDecodeH264PictureLayoutFlagBitsKHR

### New Bitmasks

- VkVideoDecodeH264PictureLayoutFlagsKHR

### New Enum Constants

- `VK_KHR_VIDEO_DECODE_H264_EXTENSION_NAME`
- `VK_KHR_VIDEO_DECODE_H264_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_VIDEO_DECODE_H264_CAPABILITIES_KHR`
  - `VK_STRUCTURE_TYPE_VIDEO_DECODE_H264_DPB_SLOT_INFO_KHR`
  - `VK_STRUCTURE_TYPE_VIDEO_DECODE_H264_PICTURE_INFO_KHR`
  - `VK_STRUCTURE_TYPE_VIDEO_DECODE_H264_PROFILE_INFO_KHR`
  - `VK_STRUCTURE_TYPE_VIDEO_DECODE_H264_SESSION_PARAMETERS_ADD_INFO_KHR`
  - `VK_STRUCTURE_TYPE_VIDEO_DECODE_H264_SESSION_PARAMETERS_CREATE_INFO_KHR`

- Extending VkVideoCodecOperationFlagBitsKHR:
  - `VK_VIDEO_CODEC_OPERATION_DECODE_H264_BIT_KHR`

**Version History**

- Revision 1, 2018-6-11 (Peter Fang)
  - Initial draft
- Revision 2, March 29 2021 (Tony Zlatinski)
  - Spec and API Updates
- Revision 3, August 1 2021 (Srinath Kumarapuram)
  - Rename `VkVideoDecodeH264FieldLayoutFlagsEXT` to `VkVideoDecodeH264PictureLayoutFlagsEXT`, `VkVideoDecodeH264FieldLayoutFlagBitsEXT` to `VkVideoDecodeH264PictureLayoutFlagBitsEXT` (along with the names of enumerants it defines), and `VkVideoDecodeH264ProfileEXT.fieldLayout` to `VkVideoDecodeH264ProfileEXT.pictureLayout`, following Vulkan naming conventions.
- Revision 4, 2022-03-16 (Ahmed Abdelkhalek)
  - Relocate Std header version reporting/requesting from this extension to VK_KHR_video_queue extension.
  - Remove the now empty VkVideoDecodeH264SessionCreateInfoEXT.
- Revision 5, 2022-03-31 (Ahmed Abdelkhalek)
  - Use type StdVideoH264Level for VkVideoDecodeH264Capabilities.maxLevel
- Revision 6, 2022-08-09 (Daniel Rakos)
  - Rename `VkVideoDecodeH264ProfileEXT` to `VkVideoDecodeH264ProfileInfoEXT`
  - Rename `VkVideoDecodeH264MvcEXT` to `VkVideoDecodeH264MvcInfoEXT`
- Revision 7, 2022-09-18 (Daniel Rakos)
  - Change type of `VkVideoDecodeH264ProfileInfoEXT::pictureLayout` to `VkVideoDecodeH264PictureLayoutFlagBitsEXT`
  - Remove MVC support and related `VkVideoDecodeH264MvcInfoEXT` structure
  - Rename `spsStdCount`, `pSpsStd`, `ppsStdCount`, and `pPpsStd` to `stdSPSCount`, `pStdSPSs`, `stdPPSCount`, and `pStdPPSs`, respectively, in `VkVideoDecodeH264SessionParametersAddInfoEXT`
  - Rename `maxSpsStdCount` and `maxPpsStdCount` to `maxStdSPSCount` and `maxStdPPSCount`, respectively, in `VkVideoDecodeH264SessionParametersCreateInfoEXT`
  - Rename `slicesCount` and `pSlicesDataOffsets` to `sliceCount` and `pSliceOffsets`, respectively, in `VkVideoDecodeH264PictureInfoEXT`
- Revision 8, 2022-09-29 (Daniel Rakos)
  - Change extension from `EXT` to `KHR`
  - Extension is no longer provisional
- Revision 9, 2023-12-05 (Daniel Rakos)

- Condition reference picture setup based on the value of `StdVideoDecodeH264PictureInfo::flags.is_reference`

## VK_KHR_video_decode_h265

### Name String

`VK_KHR_video_decode_h265`

### Extension Type

Device extension

### Registered Extension Number

188

### Revision

8

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_video_decode_queue

### Contact

- peter.fang@amd.com

### Extension Proposal

VK_KHR_video_decode_h265

### Other Extension Metadata

### Last Modified Date

2023-12-05

### IP Status

No known IP claims.

### Contributors

- Ahmed Abdelkhalek, AMD
- HoHin Lau, AMD
- Jake Beju, AMD
- Peter Fang, AMD
- Ping Liu, Intel
- Srinath Kumarapuram, NVIDIA
- Tony Zlatinski, NVIDIA
- Daniel Rakos, RasterGrid

### Description

This extension builds upon the `VK_KHR_video_decode_queue` extension by adding support for decoding elementary video stream sequences compliant with the H.265/HEVC video compression standard.

| | |
|---|---|
| **NOTE** | This extension was promoted to `KHR` from the provisional extension `VK_EXT_video_decode_h265`. |

### New Structures

- Extending VkVideoCapabilitiesKHR:
  - VkVideoDecodeH265CapabilitiesKHR
- Extending VkVideoDecodeInfoKHR:
  - VkVideoDecodeH265PictureInfoKHR
- Extending VkVideoProfileInfoKHR, VkQueryPoolCreateInfo:
  - VkVideoDecodeH265ProfileInfoKHR
- Extending VkVideoReferenceSlotInfoKHR:
  - VkVideoDecodeH265DpbSlotInfoKHR
- Extending VkVideoSessionParametersCreateInfoKHR:
  - VkVideoDecodeH265SessionParametersCreateInfoKHR
- Extending VkVideoSessionParametersUpdateInfoKHR:
  - VkVideoDecodeH265SessionParametersAddInfoKHR

### New Enum Constants

- `VK_KHR_VIDEO_DECODE_H265_EXTENSION_NAME`
- `VK_KHR_VIDEO_DECODE_H265_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_VIDEO_DECODE_H265_CAPABILITIES_KHR`
  - `VK_STRUCTURE_TYPE_VIDEO_DECODE_H265_DPB_SLOT_INFO_KHR`
  - `VK_STRUCTURE_TYPE_VIDEO_DECODE_H265_PICTURE_INFO_KHR`
  - `VK_STRUCTURE_TYPE_VIDEO_DECODE_H265_PROFILE_INFO_KHR`
  - `VK_STRUCTURE_TYPE_VIDEO_DECODE_H265_SESSION_PARAMETERS_ADD_INFO_KHR`
  - `VK_STRUCTURE_TYPE_VIDEO_DECODE_H265_SESSION_PARAMETERS_CREATE_INFO_KHR`
- Extending VkVideoCodecOperationFlagBitsKHR:
  - `VK_VIDEO_CODEC_OPERATION_DECODE_H265_BIT_KHR`

### Version History

- Revision 1, 2018-6-11 (Peter Fang)

    - Initial draft
- Revision 1.6, March 29 2021 (Tony Zlatinski)
    - Spec and API updates.
- Revision 2, 2022-03-16 (Ahmed Abdelkhalek)
    - Relocate Std header version reporting/requesting from this extension to VK_KHR_video_queue extension.
    - Remove the now empty VkVideoDecodeH265SessionCreateInfoEXT.
- Revision 3, 2022-03-31 (Ahmed Abdelkhalek)
    - Use type StdVideoH265Level for VkVideoDecodeH265Capabilities.maxLevel
- Revision 4, 2022-08-09 (Daniel Rakos)
    - Rename `VkVideoDecodeH265ProfileEXT` to `VkVideoDecodeH265ProfileInfoEXT`
- Revision 5, 2022-09-18 (Daniel Rakos)
    - Rename `vpsStdCount`, `pVpsStd`, `spsStdCount`, `pSpsStd`, `ppsStdCount`, and `pPpsStd` to `stdVPSCount`, `pStdVPSs`, `stdSPSCount`, `pStdSPSs`, `stdPPSCount`, and `pStdPPSs`, respectively, in `VkVideoDecodeH265SessionParametersAddInfoEXT`
    - Rename `maxVpsStdCount`, `maxSpsStdCount`, and `maxPpsStdCount` to `maxStdVPSCount`, `maxStdSPSCount` and `maxStdPPSCount`, respectively, in `VkVideoDecodeH265SessionParametersCreateInfoEXT`
    - Rename `slicesCount` and `pSlicesDataOffsets` to `sliceCount` and `pSliceOffsets`, respectively, in `VkVideoDecodeH265PictureInfoEXT`
- Revision 6, 2022-11-14 (Daniel Rakos)
    - Rename `slice` to `sliceSegment` in the APIs for better clarity
- Revision 7, 2022-11-14 (Daniel Rakos)
    - Change extension from `EXT` to `KHR`
    - Extension is no longer provisional
- Revision 8, 2023-12-05 (Daniel Rakos)
    - Condition reference picture setup based on the value of `StdVideoDecodeH265PictureInfo::flags.IsReference`

## VK_KHR_video_decode_queue

### Name String

`VK_KHR_video_decode_queue`

### Extension Type

Device extension

### Registered Extension Number

25

**Revision**

8

**Ratification Status**

Ratified

**Extension and Version Dependencies**

VK_KHR_video_queue
and
VK_KHR_synchronization2
or
Vulkan Version 1.3

**API Interactions**

- Interacts with VK_VERSION_1_3
- Interacts with VK_KHR_format_feature_flags2

**Contact**

- jake.beju@amd.com

**Extension Proposal**

VK_KHR_video_decode_queue

**Other Extension Metadata**

**Last Modified Date**

2023-12-05

**IP Status**

No known IP claims.

**Contributors**

- Ahmed Abdelkhalek, AMD
- Jake Beju, AMD
- Olivier Lapicque, NVIDIA
- Peter Fang, AMD
- Piers Daniell, NVIDIA
- Srinath Kumarapuram, NVIDIA
- Tony Zlatinski, NVIDIA
- Daniel Rakos, RasterGrid

**Description**

This extension builds upon the `VK_KHR_video_queue` extension by adding common APIs specific to video decoding and thus enabling implementations to expose queue families supporting video decode operations.

More specifically, it adds video decode specific capabilities and a new command buffer command that allows recording video decode operations against a video session.

This extension is to be used in conjunction with other codec specific video decode extensions that enable decoding video sequences of specific video compression standards.

#### New Commands

- vkCmdDecodeVideoKHR

#### New Structures

- VkVideoDecodeInfoKHR
- Extending VkVideoCapabilitiesKHR:
  - VkVideoDecodeCapabilitiesKHR
- Extending VkVideoProfileInfoKHR, VkQueryPoolCreateInfo:
  - VkVideoDecodeUsageInfoKHR

#### New Enums

- VkVideoDecodeCapabilityFlagBitsKHR
- VkVideoDecodeUsageFlagBitsKHR

#### New Bitmasks

- VkVideoDecodeCapabilityFlagsKHR
- VkVideoDecodeFlagsKHR
- VkVideoDecodeUsageFlagsKHR

#### New Enum Constants

- `VK_KHR_VIDEO_DECODE_QUEUE_EXTENSION_NAME`
- `VK_KHR_VIDEO_DECODE_QUEUE_SPEC_VERSION`
- Extending VkAccessFlagBits2:
  - `VK_ACCESS_2_VIDEO_DECODE_READ_BIT_KHR`
  - `VK_ACCESS_2_VIDEO_DECODE_WRITE_BIT_KHR`
- Extending VkBufferUsageFlagBits:
  - `VK_BUFFER_USAGE_VIDEO_DECODE_DST_BIT_KHR`
  - `VK_BUFFER_USAGE_VIDEO_DECODE_SRC_BIT_KHR`
- Extending VkFormatFeatureFlagBits:
  - `VK_FORMAT_FEATURE_VIDEO_DECODE_DPB_BIT_KHR`
  - `VK_FORMAT_FEATURE_VIDEO_DECODE_OUTPUT_BIT_KHR`
- Extending VkImageLayout:

    - VK_IMAGE_LAYOUT_VIDEO_DECODE_DPB_KHR
    - VK_IMAGE_LAYOUT_VIDEO_DECODE_DST_KHR
    - VK_IMAGE_LAYOUT_VIDEO_DECODE_SRC_KHR
- Extending VkImageUsageFlagBits:
    - VK_IMAGE_USAGE_VIDEO_DECODE_DPB_BIT_KHR
    - VK_IMAGE_USAGE_VIDEO_DECODE_DST_BIT_KHR
    - VK_IMAGE_USAGE_VIDEO_DECODE_SRC_BIT_KHR
- Extending VkPipelineStageFlagBits2:
    - VK_PIPELINE_STAGE_2_VIDEO_DECODE_BIT_KHR
- Extending VkQueueFlagBits:
    - VK_QUEUE_VIDEO_DECODE_BIT_KHR
- Extending VkStructureType:
    - VK_STRUCTURE_TYPE_VIDEO_DECODE_CAPABILITIES_KHR
    - VK_STRUCTURE_TYPE_VIDEO_DECODE_INFO_KHR
    - VK_STRUCTURE_TYPE_VIDEO_DECODE_USAGE_INFO_KHR

If VK_KHR_format_feature_flags2 or Vulkan Version 1.3 is supported:

- Extending VkFormatFeatureFlagBits2:
    - VK_FORMAT_FEATURE_2_VIDEO_DECODE_DPB_BIT_KHR
    - VK_FORMAT_FEATURE_2_VIDEO_DECODE_OUTPUT_BIT_KHR

**Version History**

- Revision 1, 2018-6-11 (Peter Fang)
    - Initial draft
- Revision 1.5, Nov 09 2018 (Tony Zlatinski)
    - API Updates
- Revision 1.6, Jan 08 2020 (Tony Zlatinski)
    - API unify with the video_encode_queue spec
- Revision 1.7, March 29 2021 (Tony Zlatinski)
    - Spec and API updates.
- Revision 2, September 30 2021 (Jon Leech)
    - Add interaction with VK_KHR_format_feature_flags2 to vk.xml
- Revision 3, 2022-02-25 (Ahmed Abdelkhalek)
    - Add VkVideoDecodeCapabilitiesKHR with new flags to report support for decode DPB and output coinciding in the same image, or in distinct images.
- Revision 4, 2022-03-31 (Ahmed Abdelkhalek)

    - Remove redundant VkVideoDecodeInfoKHR.coded{Offset|Extent}
- Revision 5, 2022-07-18 (Daniel Rakos)
    - Remove `VkVideoDecodeFlagBitsKHR` as it contains no defined flags for now
- Revision 6, 2022-08-12 (Daniel Rakos)
    - Add VkVideoDecodeUsageInfoKHR structure and related flags
- Revision 7, 2022-09-29 (Daniel Rakos)
    - Extension is no longer provisional
- Revision 8, 2023-12-05 (Daniel Rakos)
    - Require the specification of a reconstructed picture in all cases, except when the video session was created with no DPB slots to match shipping implementations
    - Make DPB slot activation behavior codec-specific to continue allowing application control over reference picture setup now that a reconstructed picture is always mandatory

## VK_KHR_video_encode_h264

### Name String

`VK_KHR_video_encode_h264`

### Extension Type

Device extension

### Registered Extension Number

39

### Revision

14

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_video_encode_queue

### Contact

- Ahmed Abdelkhalek [GitHub]aabdelkh

### Extension Proposal

VK_KHR_video_encode_h264

### Other Extension Metadata

### Last Modified Date

2023-12-05

### IP Status

No known IP claims.

### Contributors

- Ahmed Abdelkhalek, AMD
- George Hao, AMD
- Jake Beju, AMD
- Peter Fang, AMD
- Ping Liu, Intel
- Srinath Kumarapuram, NVIDIA
- Tony Zlatinski, NVIDIA
- Ravi Chaudhary, NVIDIA
- Yang Liu, AMD
- Daniel Rakos, RasterGrid
- Aidan Fabius, Core Avionics & Industrial Inc.
- Lynne Iribarren, Independent

### Description

This extension builds upon the `VK_KHR_video_encode_queue` extension by adding support for encoding elementary video stream sequences compliant with the H.264/AVC video compression standard.

| | |
|---|---|
| **NOTE** | This extension was promoted to `KHR` from the provisional extension `VK_EXT_video_encode_h264`. |

### New Structures

- VkVideoEncodeH264FrameSizeKHR
- VkVideoEncodeH264NaluSliceInfoKHR
- VkVideoEncodeH264QpKHR
- Extending VkVideoBeginCodingInfoKHR:
  - VkVideoEncodeH264GopRemainingFrameInfoKHR
- Extending VkVideoCapabilitiesKHR:
  - VkVideoEncodeH264CapabilitiesKHR
- Extending VkVideoCodingControlInfoKHR, VkVideoBeginCodingInfoKHR:
  - VkVideoEncodeH264RateControlInfoKHR
- Extending VkVideoEncodeInfoKHR:
  - VkVideoEncodeH264PictureInfoKHR

- Extending VkVideoEncodeQualityLevelPropertiesKHR:
  - VkVideoEncodeH264QualityLevelPropertiesKHR
- Extending VkVideoEncodeRateControlLayerInfoKHR:
  - VkVideoEncodeH264RateControlLayerInfoKHR
- Extending VkVideoEncodeSessionParametersFeedbackInfoKHR:
  - VkVideoEncodeH264SessionParametersFeedbackInfoKHR
- Extending VkVideoEncodeSessionParametersGetInfoKHR:
  - VkVideoEncodeH264SessionParametersGetInfoKHR
- Extending VkVideoProfileInfoKHR, VkQueryPoolCreateInfo:
  - VkVideoEncodeH264ProfileInfoKHR
- Extending VkVideoReferenceSlotInfoKHR:
  - VkVideoEncodeH264DpbSlotInfoKHR
- Extending VkVideoSessionCreateInfoKHR:
  - VkVideoEncodeH264SessionCreateInfoKHR
- Extending VkVideoSessionParametersCreateInfoKHR:
  - VkVideoEncodeH264SessionParametersCreateInfoKHR
- Extending VkVideoSessionParametersUpdateInfoKHR:
  - VkVideoEncodeH264SessionParametersAddInfoKHR

### New Enums

- VkVideoEncodeH264CapabilityFlagBitsKHR
- VkVideoEncodeH264RateControlFlagBitsKHR
- VkVideoEncodeH264StdFlagBitsKHR

### New Bitmasks

- VkVideoEncodeH264CapabilityFlagsKHR
- VkVideoEncodeH264RateControlFlagsKHR
- VkVideoEncodeH264StdFlagsKHR

### New Enum Constants

- `VK_KHR_VIDEO_ENCODE_H264_EXTENSION_NAME`
- `VK_KHR_VIDEO_ENCODE_H264_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_VIDEO_ENCODE_H264_CAPABILITIES_KHR`
  - `VK_STRUCTURE_TYPE_VIDEO_ENCODE_H264_DPB_SLOT_INFO_KHR`
  - `VK_STRUCTURE_TYPE_VIDEO_ENCODE_H264_GOP_REMAINING_FRAME_INFO_KHR`

- VK_STRUCTURE_TYPE_VIDEO_ENCODE_H264_NALU_SLICE_INFO_KHR
- VK_STRUCTURE_TYPE_VIDEO_ENCODE_H264_PICTURE_INFO_KHR
- VK_STRUCTURE_TYPE_VIDEO_ENCODE_H264_PROFILE_INFO_KHR
- VK_STRUCTURE_TYPE_VIDEO_ENCODE_H264_QUALITY_LEVEL_PROPERTIES_KHR
- VK_STRUCTURE_TYPE_VIDEO_ENCODE_H264_RATE_CONTROL_INFO_KHR
- VK_STRUCTURE_TYPE_VIDEO_ENCODE_H264_RATE_CONTROL_LAYER_INFO_KHR
- VK_STRUCTURE_TYPE_VIDEO_ENCODE_H264_SESSION_CREATE_INFO_KHR
- VK_STRUCTURE_TYPE_VIDEO_ENCODE_H264_SESSION_PARAMETERS_ADD_INFO_KHR
- VK_STRUCTURE_TYPE_VIDEO_ENCODE_H264_SESSION_PARAMETERS_CREATE_INFO_KHR
- VK_STRUCTURE_TYPE_VIDEO_ENCODE_H264_SESSION_PARAMETERS_FEEDBACK_INFO_KHR
- VK_STRUCTURE_TYPE_VIDEO_ENCODE_H264_SESSION_PARAMETERS_GET_INFO_KHR

- Extending VkVideoCodecOperationFlagBitsKHR:
  - VK_VIDEO_CODEC_OPERATION_ENCODE_H264_BIT_KHR

**Version History**

- Revision 0, 2018-7-23 (Ahmed Abdelkhalek)
  - Initial draft
- Revision 0.5, 2020-02-13 (Tony Zlatinski)
  - General Spec cleanup
  - Added DPB structures
  - Change the VCL frame encode structure
  - Added a common Non-VCL Picture Paramarameters structure
- Revision 1, 2021-03-29 (Tony Zlatinski)
  - Spec and API updates
- Revision 2, August 1 2021 (Srinath Kumarapuram)
  - Rename VkVideoEncodeH264CapabilitiesFlagsEXT to VkVideoEncodeH264CapabilityFlagsEXT and VkVideoEncodeH264CapabilitiesFlagsEXT to VkVideoEncodeH264CapabilityFlagsEXT, following Vulkan naming conventions.
- Revision 3, 2021-12-08 (Ahmed Abdelkhalek)
  - Rate control updates
- Revision 4, 2022-02-04 (Ahmed Abdelkhalek)
  - Align VkVideoEncodeH264VclFrameInfoEXT structure to similar one in VK_EXT_video_encode_h265 extension
- Revision 5, 2022-02-10 (Ahmed Abdelkhalek)
  - Updates to encode capability interface
- Revision 6, 2022-03-16 (Ahmed Abdelkhalek)

  - Relocate Std header version reporting/requesting from this extension to VK_KHR_video_queue extension.
  - Remove redundant maxPictureSizeInMbs from VkVideoEncodeH264SessionCreateInfoEXT.
  - Remove the now empty VkVideoEncodeH264SessionCreateInfoEXT.
- Revision 7, 2022-04-06 (Ahmed Abdelkhalek)
  - Add capability flag to report support to use B frame in L1 reference list.
  - Add capability flag to report support for disabling SPS direct_8x8_inference_flag.
- Revision 8, 2022-07-18 (Daniel Rakos)
  - Replace `VkVideoEncodeH264RateControlStructureFlagBitsEXT` bit enum with `VkVideoEncodeH264RateControlStructureEXT` enum
  - Rename `VkVideoEncodeH264ProfileEXT` to `VkVideoEncodeH264ProfileInfoEXT`
  - Rename `VkVideoEncodeH264ReferenceListsEXT` to `VkVideoEncodeH264ReferenceListsInfoEXT`
  - Rename `VkVideoEncodeH264EmitPictureParametersEXT` to `VkVideoEncodeH264EmitPictureParametersInfoEXT`
  - Rename `VkVideoEncodeH264NaluSliceEXT` to `VkVideoEncodeH264NaluSliceInfoEXT`
- Revision 9, 2022-09-18 (Daniel Rakos)
  - Rename `spsStdCount`, `pSpsStd`, `ppsStdCount`, and `pPpsStd` to `stdSPSCount`, `pStdSPSs`, `stdPPSCount`, and `pStdPPSs`, respectively, in `VkVideoEncodeH264SessionParametersAddInfoEXT`
  - Rename `maxSpsStdCount` and `maxPpsStdCount` to `maxStdSPSCount` and `maxStdPPSCount`, respectively, in `VkVideoEncodeH264SessionParametersCreateInfoEXT`
- Revision 10, 2023-03-06 (Daniel Rakos)
  - Removed `VkVideoEncodeH264EmitPictureParametersInfoEXT`
  - Changed member types in `VkVideoEncodeH264CapabilitiesEXT` and `VkVideoEncodeH264ReferenceListsInfoEXT` from `uint8_t` to `uint32_t`
  - Changed the type of `VkVideoEncodeH264RateControlInfoEXT::temporalLayerCount` and `VkVideoEncodeH264RateControlLayerInfoEXT::temporalLayerId` from `uint8_t` to `uint32_t`
  - Removed `VkVideoEncodeH264InputModeFlagsEXT` and `VkVideoEncodeH264OutputModeFlagsEXT` as we only support frame-in-frame-out mode for now
  - Rename `pCurrentPictureInfo` in `VkVideoEncodeH264VclFrameInfoEXT` to `pStdPictureInfo`
  - Rename `pSliceHeaderStd` in `VkVideoEncodeH264NaluSliceInfoEXT` to `pStdSliceHeader`
  - Rename `pReferenceFinalLists` in `VkVideoEncodeH264VclFrameInfoEXT` and `VkVideoEncodeH264NaluSliceInfoEXT` to `pStdReferenceFinalLists`
  - Removed the `slotIndex` member of `VkVideoEncodeH264DpbSlotInfoEXT` and changed it to be chained to `VkVideoReferenceSlotInfoKHR`
  - Replaced `VkVideoEncodeH264ReferenceListsInfoEXT` with the new Video Std header structure `StdVideoEncodeH264ReferenceLists` that also includes data previously part of the now removed `StdVideoEncodeH264RefMemMgmtCtrlOperations` structure
  - Added new capability flag

VK_VIDEO_ENCODE_H264_CAPABILITY_DIFFERENT_REFERENCE_FINAL_LISTS_BIT_EXT

- Revision 11, 2023-05-22 (Daniel Rakos)
  - Renamed VkVideoEncodeH264VclFrameInfoEXT to VkVideoEncodeH264PictureInfoEXT
  - Added VkVideoEncodeH264PictureInfoEXT::generatePrefixNalu and VK_VIDEO_ENCODE_H264_CAPABILITY_GENERATE_PREFIX_NALU_BIT_EXT to enable the generation of H.264 prefix NALUs when supported by the implementation
  - Removed VkVideoEncodeH264RateControlLayerInfoEXT::temporalLayerId
  - Added expectDyadicTemporalLayerPattern capability
  - Added the VkVideoEncodeH264SessionParametersGetInfoEXT structure to identify the H.264 parameter sets to retrieve encoded parameter data for, and the VkVideoEncodeH264SessionParametersFeedbackInfoEXT structure to retrieve H.264 parameter set override information when using the new vkGetEncodedVideoSessionParametersKHR command
  - Added VkVideoEncodeH264NaluSliceInfoEXT::constantQp to specify per-slice constant QP when rate control mode is VK_VIDEO_ENCODE_RATE_CONTROL_MODE_DISABLED_BIT_KHR
  - Added VkVideoEncodeH264QualityLevelPropertiesEXT for retrieving H.264 specific quality level recommendations
  - Replaced VkVideoEncodeH264RateControlStructureEXT enum with the flags type VkVideoEncodeH264RateControlFlagsEXT and bits defined in VkVideoEncodeH264RateControlFlagBitsEXT and added HRD compliance flag
  - Removed useInitialRcQp and initialRcQp members of VkVideoEncodeH264RateControlLayerInfoEXT
  - Added prefersGopRemainingFrames and requiresGopRemainingFrames, and the new VkVideoEncodeH264GopRemainingFrameInfoEXT structure to allow specifying remaining frames of each type in the rate control GOP
  - Added maxTemporalLayers, maxQp, and minQp capabilities
  - Added maxLevelIdc capability and new VkVideoEncodeH264SessionCreateInfoEXT structure to specify upper bounds on the H.264 level of the produced video bitstream
  - Moved capability flags specific to codec syntax restrictions from VkVideoEncodeH264CapabilityFlagsEXT to the new VkVideoEncodeH264StdFlagsEXT which is now included as a separate stdSyntaxFlags member in VkVideoEncodeH264CapabilitiesEXT
  - Removed codec syntax override values from VkVideoEncodeH264CapabilitiesEXT
  - Removed VkVideoEncodeH264NaluSliceInfoEXT::mbCount and VK_VIDEO_ENCODE_H264_CAPABILITY_SLICE_MB_COUNT_BIT_EXT
  - Replaced VK_VIDEO_ENCODE_H264_CAPABILITY_MULTIPLE_SLICES_PER_FRAME_BIT_EXT with the new maxSliceCount capability
  - Removed capability flag VK_VIDEO_ENCODE_H264_CAPABILITY_DIFFERENT_REFERENCE_FINAL_LISTS_BIT_EXT and removed pStdReferenceFinalLists members from the VkVideoEncodeH264PictureInfoEXT and VkVideoEncodeH264NaluSliceInfoEXT structures as reference lists info is now included in pStdPictureInfo

    - Added capability flag `VK_VIDEO_ENCODE_H264_CAPABILITY_B_FRAME_IN_L0_LIST_BIT_EXT`
- Revision 12, 2023-07-19 (Daniel Rakos)
    - Added video std capability flags `VK_VIDEO_ENCODE_H264_STD_SLICE_QP_DELTA_BIT_EXT` and `VK_VIDEO_ENCODE_H264_STD_DIFFERENT_SLICE_QP_DELTA_BIT_EXT`
    - Fixed optionality of the array members of `VkVideoEncodeH264SessionParametersAddInfoEXT`
    - Fixed optionality of `VkVideoEncodeH264RateControlInfoEXT::flags`
- Revision 13, 2023-09-04 (Daniel Rakos)
    - Change extension from `EXT` to `KHR`
    - Extension is no longer provisional
- Revision 14, 2023-12-05 (Daniel Rakos)
    - Condition reference picture setup based on the value of `StdVideoEncodeH264PictureInfo::flags.is_reference`

## VK_KHR_video_encode_h265

### Name String

`VK_KHR_video_encode_h265`

### Extension Type

Device extension

### Registered Extension Number

40

### Revision

14

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_video_encode_queue

### Contact

- Ahmed Abdelkhalek [GitHub]aabdelkh

### Extension Proposal

VK_KHR_video_encode_h265

### Other Extension Metadata

### Last Modified Date

2023-12-05

### IP Status

No known IP claims.

### Contributors

- Ahmed Abdelkhalek, AMD
- George Hao, AMD
- Jake Beju, AMD
- Chunbo Chen, Intel
- Ping Liu, Intel
- Srinath Kumarapuram, NVIDIA
- Tony Zlatinski, NVIDIA
- Ravi Chaudhary, NVIDIA
- Daniel Rakos, RasterGrid
- Aidan Fabius, Core Avionics & Industrial Inc.
- Lynne Iribarren, Independent

### Description

This extension builds upon the `VK_KHR_video_encode_queue` extension by adding support for encoding elementary video stream sequences compliant with the H.265/HEVC video compression standard.

| | |
|---|---|
| **NOTE** | This extension was promoted to `KHR` from the provisional extension `VK_EXT_video_encode_h265`. |

### New Structures

- VkVideoEncodeH265FrameSizeKHR
- VkVideoEncodeH265NaluSliceSegmentInfoKHR
- VkVideoEncodeH265QpKHR
- Extending VkVideoBeginCodingInfoKHR:
  - VkVideoEncodeH265GopRemainingFrameInfoKHR
- Extending VkVideoCapabilitiesKHR:
  - VkVideoEncodeH265CapabilitiesKHR
- Extending VkVideoCodingControlInfoKHR, VkVideoBeginCodingInfoKHR:
  - VkVideoEncodeH265RateControlInfoKHR
- Extending VkVideoEncodeInfoKHR:
  - VkVideoEncodeH265PictureInfoKHR
- Extending VkVideoEncodeQualityLevelPropertiesKHR:

    - VkVideoEncodeH265QualityLevelPropertiesKHR
- Extending VkVideoEncodeRateControlLayerInfoKHR:
    - VkVideoEncodeH265RateControlLayerInfoKHR
- Extending VkVideoEncodeSessionParametersFeedbackInfoKHR:
    - VkVideoEncodeH265SessionParametersFeedbackInfoKHR
- Extending VkVideoEncodeSessionParametersGetInfoKHR:
    - VkVideoEncodeH265SessionParametersGetInfoKHR
- Extending VkVideoProfileInfoKHR, VkQueryPoolCreateInfo:
    - VkVideoEncodeH265ProfileInfoKHR
- Extending VkVideoReferenceSlotInfoKHR:
    - VkVideoEncodeH265DpbSlotInfoKHR
- Extending VkVideoSessionCreateInfoKHR:
    - VkVideoEncodeH265SessionCreateInfoKHR
- Extending VkVideoSessionParametersCreateInfoKHR:
    - VkVideoEncodeH265SessionParametersCreateInfoKHR
- Extending VkVideoSessionParametersUpdateInfoKHR:
    - VkVideoEncodeH265SessionParametersAddInfoKHR

### New Enums

- VkVideoEncodeH265CapabilityFlagBitsKHR
- VkVideoEncodeH265CtbSizeFlagBitsKHR
- VkVideoEncodeH265RateControlFlagBitsKHR
- VkVideoEncodeH265StdFlagBitsKHR
- VkVideoEncodeH265TransformBlockSizeFlagBitsKHR

### New Bitmasks

- VkVideoEncodeH265CapabilityFlagsKHR
- VkVideoEncodeH265CtbSizeFlagsKHR
- VkVideoEncodeH265RateControlFlagsKHR
- VkVideoEncodeH265StdFlagsKHR
- VkVideoEncodeH265TransformBlockSizeFlagsKHR

### New Enum Constants

- `VK_KHR_VIDEO_ENCODE_H265_EXTENSION_NAME`
- `VK_KHR_VIDEO_ENCODE_H265_SPEC_VERSION`
- Extending VkStructureType:

- VK_STRUCTURE_TYPE_VIDEO_ENCODE_H265_CAPABILITIES_KHR
- VK_STRUCTURE_TYPE_VIDEO_ENCODE_H265_DPB_SLOT_INFO_KHR
- VK_STRUCTURE_TYPE_VIDEO_ENCODE_H265_GOP_REMAINING_FRAME_INFO_KHR
- VK_STRUCTURE_TYPE_VIDEO_ENCODE_H265_NALU_SLICE_SEGMENT_INFO_KHR
- VK_STRUCTURE_TYPE_VIDEO_ENCODE_H265_PICTURE_INFO_KHR
- VK_STRUCTURE_TYPE_VIDEO_ENCODE_H265_PROFILE_INFO_KHR
- VK_STRUCTURE_TYPE_VIDEO_ENCODE_H265_QUALITY_LEVEL_PROPERTIES_KHR
- VK_STRUCTURE_TYPE_VIDEO_ENCODE_H265_RATE_CONTROL_INFO_KHR
- VK_STRUCTURE_TYPE_VIDEO_ENCODE_H265_RATE_CONTROL_LAYER_INFO_KHR
- VK_STRUCTURE_TYPE_VIDEO_ENCODE_H265_SESSION_CREATE_INFO_KHR
- VK_STRUCTURE_TYPE_VIDEO_ENCODE_H265_SESSION_PARAMETERS_ADD_INFO_KHR
- VK_STRUCTURE_TYPE_VIDEO_ENCODE_H265_SESSION_PARAMETERS_CREATE_INFO_KHR
- VK_STRUCTURE_TYPE_VIDEO_ENCODE_H265_SESSION_PARAMETERS_FEEDBACK_INFO_KHR
- VK_STRUCTURE_TYPE_VIDEO_ENCODE_H265_SESSION_PARAMETERS_GET_INFO_KHR

- Extending VkVideoCodecOperationFlagBitsKHR:
  - VK_VIDEO_CODEC_OPERATION_ENCODE_H265_BIT_KHR

**Version History**

- Revision 0, 2019-11-14 (Ahmed Abdelkhalek)
  - Initial draft
- Revision 0.5, 2020-02-13 (Tony Zlatinski)
  - General Spec cleanup
  - Added DPB structures
  - Change the VCL frame encode structure
  - Added a common Non-VCL Picture Paramarameters structure
- Revision 2, Oct 10 2021 (Srinath Kumarapuram)
  - Vulkan Video Encode h.265 update and spec edits
- Revision 3, 2021-12-08 (Ahmed Abdelkhalek)
  - Rate control updates
- Revision 4, 2022-01-11 (Ahmed Abdelkhalek)
  - Replace occurrences of “slice” by “slice segment” and rename structures/enums to reflect this.
- Revision 5, 2022-02-10 (Ahmed Abdelkhalek)
  - Updates to encode capability interface
- Revision 6, 2022-03-16 (Ahmed Abdelkhalek)

  - Relocate Std header version reporting/requesting from this extension to VK_KHR_video_queue extension.
  - Remove the now empty VkVideoEncodeH265SessionCreateInfoEXT.
- Revision 7, 2022-03-24 (Ahmed Abdelkhalek)
  - Add capability flags to report support to disable transform skip and support to use B frame in L1 reference list.
- Revision 8, 2022-07-18 (Daniel Rakos)
  - Replace `VkVideoEncodeH265RateControlStructureFlagBitsEXT` bit enum with `VkVideoEncodeH265RateControlStructureEXT` enum
  - Rename `VkVideoEncodeH265ProfileEXT` to `VkVideoEncodeH265ProfileInfoEXT`
  - Rename `VkVideoEncodeH265ReferenceListsEXT` to `VkVideoEncodeH265ReferenceListsInfoEXT`
  - Rename `VkVideoEncodeH265EmitPictureParametersEXT` to `VkVideoEncodeH265EmitPictureParametersInfoEXT`
  - Rename `VkVideoEncodeH265NaluSliceSegmentEXT` to `VkVideoEncodeH265NaluSliceSegmentInfoEXT`
- Revision 9, 2022-09-18 (Daniel Rakos)
  - Rename `vpsStdCount`, `pVpsStd`, `spsStdCount`, `pSpsStd`, `ppsStdCount`, and `pPpsStd` to `stdVPSCount`, `pStdVPSs`, `stdSPSCount`, `pStdSPSs`, `stdPPSCount`, and `pStdPPSs`, respectively, in `VkVideoEncodeH265SessionParametersAddInfoEXT`
  - Rename `maxVpsStdCount`, `maxSpsStdCount`, and `maxPpsStdCount` to `maxStdVPSCount`, `maxStdSPSCount` and `maxStdPPSCount`, respectively, in `VkVideoEncodeH265SessionParametersCreateInfoEXT`
- Revision 10, 2023-03-06 (Daniel Rakos)
  - Removed `VkVideoEncodeH265EmitPictureParametersInfoEXT`
  - Changed member types in `VkVideoEncodeH265CapabilitiesEXT` and `VkVideoEncodeH265ReferenceListsInfoEXT` from `uint8_t` to `uint32_t`
  - Changed the type of `VkVideoEncodeH265RateControlInfoEXT::subLayerCount` and `VkVideoEncodeH265RateControlLayerInfoEXT::temporalId` from `uint8_t` to `uint32_t`
  - Removed `VkVideoEncodeH265InputModeFlagsEXT` and `VkVideoEncodeH265OutputModeFlagsEXT` as we only support frame-in-frame-out mode for now
  - Rename `pCurrentPictureInfo` in `VkVideoEncodeH265VclFrameInfoEXT` to `pStdPictureInfo`
  - Rename `pSliceSegmentHeaderStd` in `VkVideoEncodeH265NaluSliceSegmentInfoEXT` to `pStdSliceSegmentHeader`
  - Rename `pReferenceFinalLists` in `VkVideoEncodeH265VclFrameInfoEXT` and `VkVideoEncodeH265NaluSliceSegmentInfoEXT` to `pStdReferenceFinalLists`
  - Removed the `slotIndex` member of `VkVideoEncodeH265DpbSlotInfoEXT` and changed it to be chained to `VkVideoReferenceSlotInfoKHR`
  - Replaced `VkVideoEncodeH265ReferenceListsInfoEXT` with the new Video Std header structure `StdVideoEncodeH265ReferenceLists`
  - Added new capability flag

VK_VIDEO_ENCODE_H265_CAPABILITY_DIFFERENT_REFERENCE_FINAL_LISTS_BIT_EXT

- Revision 11, 2023-05-26 (Daniel Rakos)
  - Renamed VkVideoEncodeH265VclFrameInfoEXT to VkVideoEncodeH265PictureInfoEXT
  - Removed VkVideoEncodeH265RateControlLayerInfoEXT::temporalId
  - Added expectDyadicTemporalSubLayerPattern capability
  - Added the VkVideoEncodeH265SessionParametersGetInfoEXT structure to identify the H.265 parameter sets to retrieve encoded parameter data for, and the VkVideoEncodeH265SessionParametersFeedbackInfoEXT structure to retrieve H.265 parameter set override information when using the new vkGetEncodedVideoSessionParametersKHR command
  - Added VkVideoEncodeH265NaluSliceSegmentInfoEXT::constantQp to specify per-slice segment constant QP when rate control mode is VK_VIDEO_ENCODE_RATE_CONTROL_MODE_DISABLED_BIT_KHR
  - Added VkVideoEncodeH265QualityLevelPropertiesEXT for retrieving H.265 specific quality level recommendations
  - Replaced VkVideoEncodeH265RateControlStructureEXT enum with the flags type VkVideoEncodeH265RateControlFlagsEXT and bits defined in VkVideoEncodeH265RateControlFlagBitsEXT and added HRD compliance flag
  - Removed useInitialRcQp and initialRcQp members of VkVideoEncodeH265RateControlLayerInfoEXT
  - Added prefersGopRemainingFrames and requiresGopRemainingFrames, and the new VkVideoEncodeH265GopRemainingFrameInfoEXT structure to allow specifying remaining frames of each type in the rate control GOP
  - Renamed maxSubLayersCount capability to maxSubLayerCount
  - Added maxQp, and minQp capabilities
  - Added maxLevelIdc capability and new VkVideoEncodeH265SessionCreateInfoEXT structure to specify upper bounds on the H.265 level of the produced video bitstream
  - Moved capability flags specific to codec syntax restrictions from VkVideoEncodeH265CapabilityFlagsEXT to the new VkVideoEncodeH265StdFlagsEXT which is now included as a separate stdSyntaxFlags member in VkVideoEncodeH265CapabilitiesEXT
  - Added std prefix to codec syntax capabilities in VkVideoEncodeH265CapabilitiesEXT
  - Removed VkVideoEncodeH265NaluSliceSegmentInfoEXT::ctbCount and VK_VIDEO_ENCODE_H265_CAPABILITY_SLICE_SEGMENT_CTB_COUNT_BIT_EXT
  - Replaced VK_VIDEO_ENCODE_H265_CAPABILITY_MULTIPLE_SLICE_SEGMENTS_PER_FRAME_BIT_EXT with the new maxSliceSegmentCount capability
  - Added maxTiles capability
  - Removed codec syntax min/max capabilities from VkVideoEncodeH265CapabilitiesEXT
  - Removed capability flag VK_VIDEO_ENCODE_H265_CAPABILITY_DIFFERENT_REFERENCE_FINAL_LISTS_BIT_EXT and removed pStdReferenceFinalLists members from the VkVideoEncodeH265PictureInfoEXT and VkVideoEncodeH265NaluSliceSegmentInfoEXT structures as reference lists info is now included

in `pStdPictureInfo`
    - Added capability flag `VK_VIDEO_ENCODE_H265_CAPABILITY_B_FRAME_IN_L0_LIST_BIT_EXT`
- Revision 12, 2023-07-19 (Daniel Rakos)
    - Added video std capability flags `VK_VIDEO_ENCODE_H265_STD_SLICE_QP_DELTA_BIT_EXT` and `VK_VIDEO_ENCODE_H265_STD_DIFFERENT_SLICE_QP_DELTA_BIT_EXT`
    - Fixed optionality of the array members of `VkVideoEncodeH265SessionParametersAddInfoEXT`
    - Fixed optionality of `VkVideoEncodeH265RateControlInfoEXT::flags`
- Revision 13, 2023-09-04 (Daniel Rakos)
    - Change extension from `EXT` to `KHR`
    - Extension is no longer provisional
- Revision 14, 2023-12-05 (Daniel Rakos)
    - Condition reference picture setup based on the value of `StdVideoEncodeH265PictureInfo::flags.is_reference`

## VK_KHR_video_encode_queue

### Name String

`VK_KHR_video_encode_queue`

### Extension Type

Device extension

### Registered Extension Number

300

### Revision

12

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_video_queue
and
VK_KHR_synchronization2
or
Vulkan Version 1.3

### API Interactions

- Interacts with VK_VERSION_1_3
- Interacts with VK_KHR_format_feature_flags2

### Contact

- Ahmed Abdelkhalek [GitHub]aabdelkh

**Extension Proposal**

VK_KHR_video_encode_queue

**Other Extension Metadata**

**Last Modified Date**

2023-12-05

**IP Status**

No known IP claims.

**Contributors**

- Ahmed Abdelkhalek, AMD
- Damien Kessler, NVIDIA
- George Hao, AMD
- Jake Beju, AMD
- Peter Fang, AMD
- Piers Daniell, NVIDIA
- Srinath Kumarapuram, NVIDIA
- Thomas J. Meier, NVIDIA
- Tony Zlatinski, NVIDIA
- Ravi Chaudhary, NVIDIA
- Yang Liu, AMD
- Daniel Rakos, RasterGrid
- Ping Liu, Intel
- Aidan Fabius, Core Avionics & Industrial Inc.
- Lynne Iribarren, Independent

**Description**

This extension builds upon the `VK_KHR_video_queue` extension by adding common APIs specific to video encoding and thus enabling implementations to expose queue families supporting video encode operations.

More specifically, it adds video encode specific capabilities and a new command buffer command that allows recording video encode operations against a video session.

This extension is to be used in conjunction with other codec specific video encode extensions that enable encoding video sequences of specific video compression standards.

**New Commands**

- vkCmdEncodeVideoKHR

- vkGetEncodedVideoSessionParametersKHR
- vkGetPhysicalDeviceVideoEncodeQualityLevelPropertiesKHR

### New Structures

- VkPhysicalDeviceVideoEncodeQualityLevelInfoKHR
- VkVideoEncodeInfoKHR
- VkVideoEncodeQualityLevelPropertiesKHR
- VkVideoEncodeRateControlLayerInfoKHR
- VkVideoEncodeSessionParametersFeedbackInfoKHR
- VkVideoEncodeSessionParametersGetInfoKHR
- Extending VkQueryPoolCreateInfo:
  - VkQueryPoolVideoEncodeFeedbackCreateInfoKHR
- Extending VkVideoCapabilitiesKHR:
  - VkVideoEncodeCapabilitiesKHR
- Extending VkVideoCodingControlInfoKHR, VkVideoBeginCodingInfoKHR:
  - VkVideoEncodeRateControlInfoKHR
- Extending VkVideoCodingControlInfoKHR, VkVideoSessionParametersCreateInfoKHR:
  - VkVideoEncodeQualityLevelInfoKHR
- Extending VkVideoProfileInfoKHR, VkQueryPoolCreateInfo:
  - VkVideoEncodeUsageInfoKHR

### New Enums

- VkVideoEncodeCapabilityFlagBitsKHR
- VkVideoEncodeContentFlagBitsKHR
- VkVideoEncodeFeedbackFlagBitsKHR
- VkVideoEncodeRateControlModeFlagBitsKHR
- VkVideoEncodeTuningModeKHR
- VkVideoEncodeUsageFlagBitsKHR

### New Bitmasks

- VkVideoEncodeCapabilityFlagsKHR
- VkVideoEncodeContentFlagsKHR
- VkVideoEncodeFeedbackFlagsKHR
- VkVideoEncodeFlagsKHR
- VkVideoEncodeRateControlFlagsKHR
- VkVideoEncodeRateControlModeFlagsKHR

- VkVideoEncodeUsageFlagsKHR

### New Enum Constants

- VK_KHR_VIDEO_ENCODE_QUEUE_EXTENSION_NAME
- VK_KHR_VIDEO_ENCODE_QUEUE_SPEC_VERSION
- Extending VkAccessFlagBits2:
  - VK_ACCESS_2_VIDEO_ENCODE_READ_BIT_KHR
  - VK_ACCESS_2_VIDEO_ENCODE_WRITE_BIT_KHR
- Extending VkBufferUsageFlagBits:
  - VK_BUFFER_USAGE_VIDEO_ENCODE_DST_BIT_KHR
  - VK_BUFFER_USAGE_VIDEO_ENCODE_SRC_BIT_KHR
- Extending VkFormatFeatureFlagBits:
  - VK_FORMAT_FEATURE_VIDEO_ENCODE_DPB_BIT_KHR
  - VK_FORMAT_FEATURE_VIDEO_ENCODE_INPUT_BIT_KHR
- Extending VkImageLayout:
  - VK_IMAGE_LAYOUT_VIDEO_ENCODE_DPB_KHR
  - VK_IMAGE_LAYOUT_VIDEO_ENCODE_DST_KHR
  - VK_IMAGE_LAYOUT_VIDEO_ENCODE_SRC_KHR
- Extending VkImageUsageFlagBits:
  - VK_IMAGE_USAGE_VIDEO_ENCODE_DPB_BIT_KHR
  - VK_IMAGE_USAGE_VIDEO_ENCODE_DST_BIT_KHR
  - VK_IMAGE_USAGE_VIDEO_ENCODE_SRC_BIT_KHR
- Extending VkPipelineStageFlagBits2:
  - VK_PIPELINE_STAGE_2_VIDEO_ENCODE_BIT_KHR
- Extending VkQueryResultStatusKHR:
  - VK_QUERY_RESULT_STATUS_INSUFFICIENT_BITSTREAM_BUFFER_RANGE_KHR
- Extending VkQueryType:
  - VK_QUERY_TYPE_VIDEO_ENCODE_FEEDBACK_KHR
- Extending VkQueueFlagBits:
  - VK_QUEUE_VIDEO_ENCODE_BIT_KHR
- Extending VkResult:
  - VK_ERROR_INVALID_VIDEO_STD_PARAMETERS_KHR
- Extending VkStructureType:
  - VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_VIDEO_ENCODE_QUALITY_LEVEL_INFO_KHR
  - VK_STRUCTURE_TYPE_QUERY_POOL_VIDEO_ENCODE_FEEDBACK_CREATE_INFO_KHR

    - VK_STRUCTURE_TYPE_VIDEO_ENCODE_CAPABILITIES_KHR
    - VK_STRUCTURE_TYPE_VIDEO_ENCODE_INFO_KHR
    - VK_STRUCTURE_TYPE_VIDEO_ENCODE_QUALITY_LEVEL_INFO_KHR
    - VK_STRUCTURE_TYPE_VIDEO_ENCODE_QUALITY_LEVEL_PROPERTIES_KHR
    - VK_STRUCTURE_TYPE_VIDEO_ENCODE_RATE_CONTROL_INFO_KHR
    - VK_STRUCTURE_TYPE_VIDEO_ENCODE_RATE_CONTROL_LAYER_INFO_KHR
    - VK_STRUCTURE_TYPE_VIDEO_ENCODE_SESSION_PARAMETERS_FEEDBACK_INFO_KHR
    - VK_STRUCTURE_TYPE_VIDEO_ENCODE_SESSION_PARAMETERS_GET_INFO_KHR
    - VK_STRUCTURE_TYPE_VIDEO_ENCODE_USAGE_INFO_KHR
- Extending VkVideoCodingControlFlagBitsKHR:
    - VK_VIDEO_CODING_CONTROL_ENCODE_QUALITY_LEVEL_BIT_KHR
    - VK_VIDEO_CODING_CONTROL_ENCODE_RATE_CONTROL_BIT_KHR
- Extending VkVideoSessionCreateFlagBitsKHR:
    - VK_VIDEO_SESSION_CREATE_ALLOW_ENCODE_PARAMETER_OPTIMIZATIONS_BIT_KHR

If VK_KHR_format_feature_flags2 or Vulkan Version 1.3 is supported:

- Extending VkFormatFeatureFlagBits2:
    - VK_FORMAT_FEATURE_2_VIDEO_ENCODE_DPB_BIT_KHR
    - VK_FORMAT_FEATURE_2_VIDEO_ENCODE_INPUT_BIT_KHR

**Version History**

- Revision 1, 2018-07-23 (Ahmed Abdelkhalek)
    - Initial draft
- Revision 1.1, 10/29/2019 (Tony Zlatinski)
    - Updated the reserved spec tokens and renamed VkVideoEncoderKHR to VkVideoSessionKHR
- Revision 1.6, Jan 08 2020 (Tony Zlatinski)
    - API unify with the video_decode_queue spec
- Revision 2, March 29 2021 (Tony Zlatinski)
    - Spec and API updates.
- Revision 3, 2021-09-30 (Jon Leech)
    - Add interaction with VK_KHR_format_feature_flags2 to vk.xml
- Revision 4, 2022-02-10 (Ahmed Abdelkhalek)
    - Updates to encode capability interface
- Revision 5, 2022-03-31 (Ahmed Abdelkhalek)
    - Remove redundant VkVideoEncodeInfoKHR.codedExtent

- Revision 6, 2022-07-18 (Daniel Rakos)
  - Remove `VkVideoEncodeRateControlFlagBitsKHR` and `VkVideoEncodeFlagBitsKHR` as they contain no defined flags for now
  - Add `VK_VIDEO_CODING_CONTROL_ENCODE_RATE_CONTROL_BIT_KHR` and `VK_VIDEO_CODING_CONTROL_ENCODE_RATE_CONTROL_LAYER_BIT_KHR` to indicate rate control and rate control layer change requests, respectively, in video coding control operations
- Revision 7, 2022-08-12 (Daniel Rakos)
  - Add VkVideoEncodeUsageInfoKHR structure and related flags
- Revision 8, 2023-03-06 (Daniel Rakos)
  - Replace `VK_QUERY_TYPE_VIDEO_ENCODE_BITSTREAM_BUFFER_RANGE_KHR` queries with more generic `VK_QUERY_TYPE_VIDEO_ENCODE_FEEDBACK_KHR` queries that can be extended in the future with more feedback values
  - Rename `dstBitstreamBuffer`, `dstBitstreamBufferOffset`, and `dstBitstreamBufferMaxRange` in `VkVideoEncodeInfoKHR` to `dstBuffer`, `dstBufferOffset`, and `dstBufferRange`, respectively, for consistency with the naming convention in the video decode extensions
  - Change the type of `rateControlLayerCount` and `qualityLevelCount` in `VkVideoEncodeCapabilitiesKHR` from `uint8_t` to `uint32_t` and rename them to `maxRateControlLayers` and `maxQualityLevels`, respectively
  - Change the type of `averageBitrate` and `maxBitrate` in `VkVideoEncodeRateControlLayerInfoKHR` from `uint32_t` to `uint64_t`
  - Fixed the definition of rate control flag bits and added the new `VK_VIDEO_ENCODE_RATE_CONTROL_MODE_DEFAULT_KHR` constant to indicate implementation-specific automatic rate control
  - Change the type of `VkVideoEncodeRateControlInfoKHR::layerCount` from `uint8_t` to `uint32_t`
  - Rename `pLayerConfigs` to `pLayers` in `VkVideoEncodeRateControlInfoKHR`
- Revision 9, 2023-03-28 (Daniel Rakos)
  - Removed `VK_VIDEO_CODING_CONTROL_ENCODE_RATE_CONTROL_LAYER_BIT_KHR` and the ability to change the state of individual rate control layers
  - Added new `VK_VIDEO_ENCODE_FEEDBACK_BITSTREAM_HAS_OVERRIDES_BIT_KHR` flag to video encode feedback queries
  - Added new video session create flag `VK_VIDEO_SESSION_CREATE_ALLOW_ENCODE_PARAMETER_OPTIMIZATIONS_BIT_KHR` to opt-in to video session and encoding parameter optimizations
  - Added the `vkGetEncodedVideoSessionParametersKHR` command to enable retrieving encoded video session parameter data
  - Moved `virtualBufferSizeInMs` and `initialVirtualBufferSizeInMs` from `VkVideoEncodeRateControlLayerInfoKHR` to `VkVideoEncodeRateControlInfoKHR`
  - Added `maxBitrate` capability
  - Renamed `inputImageDataFillAlignment` capability to `encodeInputPictureGranularity` to better reflect its purpose

    - Added new `vkGetPhysicalDeviceVideoEncodeQualityLevelPropertiesKHR` command and related structures to enable querying recommended settings for video encode quality levels
    - Added `VK_VIDEO_CODING_CONTROL_ENCODE_QUALITY_LEVEL_BIT_KHR` flag and `VkVideoEncodeQualityLevelInfoKHR` structure to allow controlling video encode quality level and removed `qualityLevel` from the encode operation parameters
- Revision 10, 2023-07-19 (Daniel Rakos)
    - Added `VK_QUERY_RESULT_STATUS_INSUFFICIENT_BITSTREAM_BUFFER_RANGE_KHR` query result status code and the related capability flag `VK_VIDEO_ENCODE_CAPABILITY_INSUFFICIENT_BITSTREAM_BUFFER_RANGE_DETECTION_BIT_KHR`
- Revision 11, 2023-09-04 (Daniel Rakos)
    - Extension is no longer provisional
- Revision 12, 2023-12-05 (Daniel Rakos)
    - Require the specification of a reconstructed picture in all cases, except when the video session was created with no DPB slots to match shipping implementations
    - Make DPB slot activation behavior codec-specific to continue allowing application control over reference picture setup now that a reconstructed picture is always mandatory

## VK_KHR_video_maintenance1

### Name String

`VK_KHR_video_maintenance1`

### Extension Type

Device extension

### Registered Extension Number

516

### Revision

1

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_video_queue

### Contact

- Daniel Rakos [GitHub]aqnuep

### Extension Proposal

VK_KHR_video_maintenance1

### Other Extension Metadata

#### Last Modified Date

2023-07-27

#### IP Status

No known IP claims.

#### Contributors

- Ahmed Abdelkhalek, AMD
- Aidan Fabius, Core Avionics & Industrial Inc.
- Ping Liu, Intel
- Lynne Iribarren, Independent
- Srinath Kumarapuram, NVIDIA
- Tony Zlatinski, NVIDIA
- Daniel Rakos, RasterGrid

#### Description

`VK_KHR_video_maintenance1` adds a collection of minor video coding features, none of which would warrant an entire extension of their own.

The new features are as follows:

- Allow creating buffers that can be used in video coding operations, independent of the used video profile, using the new buffer creation flag `VK_BUFFER_CREATE_VIDEO_PROFILE_INDEPENDENT_BIT_KHR`.
- Allow creating images that can be used as decode output or encode input pictures, independent of the used video profile, using the new image creation flag `VK_IMAGE_CREATE_VIDEO_PROFILE_INDEPENDENT_BIT_KHR`.
- Allow specifying queries used by video coding operations as part of the video coding command parameters, instead of using begin/end query when the video session is created using the new video session creation flag `VK_VIDEO_SESSION_CREATE_INLINE_QUERIES_BIT_KHR`.

#### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceVideoMaintenance1FeaturesKHR
- Extending VkVideoDecodeInfoKHR, VkVideoEncodeInfoKHR:
  - VkVideoInlineQueryInfoKHR

#### New Enum Constants

- `VK_KHR_VIDEO_MAINTENANCE_1_EXTENSION_NAME`
- `VK_KHR_VIDEO_MAINTENANCE_1_SPEC_VERSION`

- Extending VkBufferCreateFlagBits:
  - `VK_BUFFER_CREATE_VIDEO_PROFILE_INDEPENDENT_BIT_KHR`
- Extending VkImageCreateFlagBits:
  - `VK_IMAGE_CREATE_VIDEO_PROFILE_INDEPENDENT_BIT_KHR`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_VIDEO_MAINTENANCE_1_FEATURES_KHR`
  - `VK_STRUCTURE_TYPE_VIDEO_INLINE_QUERY_INFO_KHR`
- Extending VkVideoSessionCreateFlagBitsKHR:
  - `VK_VIDEO_SESSION_CREATE_INLINE_QUERIES_BIT_KHR`

### Version History

- Revision 1, 2023-07-27 (Daniel Rakos)
  - internal revisions

## VK_KHR_video_queue

### Name String

`VK_KHR_video_queue`

### Extension Type

Device extension

### Registered Extension Number

24

### Revision

8

### Ratification Status

Ratified

### Extension and Version Dependencies

Vulkan Version 1.1
and
VK_KHR_synchronization2
or
Vulkan Version 1.3

### Contact

- Tony Zlatinski [GitHub]tzlatinski

### Extension Proposal

VK_KHR_video_queue

### Other Extension Metadata

#### Last Modified Date

2022-09-29

#### IP Status

No known IP claims.

#### Contributors

- Ahmed Abdelkhalek, AMD
- George Hao, AMD
- Jake Beju, AMD
- Piers Daniell, NVIDIA
- Srinath Kumarapuram, NVIDIA
- Tobias Hector, AMD
- Tony Zlatinski, NVIDIA
- Daniel Rakos, RasterGrid

#### Description

This extension provides common APIs to enable exposing queue families with support for video codec operations by introducing the following new object types and related functionalities:

- Video session objects that represent and maintain the state needed to perform video codec operations.
- Video session parameters objects that act as a container for codec specific parameters.

In addition, it also introduces query commands that allow applications to determine video coding related capabilities, and command buffer commands that enable recording video coding operations against a video session.

This extension is to be used in conjunction with other extensions that enable specific video coding operations.

#### New Object Types

- VkVideoSessionKHR
- VkVideoSessionParametersKHR

#### New Commands

- vkBindVideoSessionMemoryKHR
- vkCmdBeginVideoCodingKHR
- vkCmdControlVideoCodingKHR
- vkCmdEndVideoCodingKHR

- vkCreateVideoSessionKHR
- vkCreateVideoSessionParametersKHR
- vkDestroyVideoSessionKHR
- vkDestroyVideoSessionParametersKHR
- vkGetPhysicalDeviceVideoCapabilitiesKHR
- vkGetPhysicalDeviceVideoFormatPropertiesKHR
- vkGetVideoSessionMemoryRequirementsKHR
- vkUpdateVideoSessionParametersKHR

**New Structures**

- VkBindVideoSessionMemoryInfoKHR
- VkPhysicalDeviceVideoFormatInfoKHR
- VkVideoBeginCodingInfoKHR
- VkVideoCapabilitiesKHR
- VkVideoCodingControlInfoKHR
- VkVideoEndCodingInfoKHR
- VkVideoFormatPropertiesKHR
- VkVideoPictureResourceInfoKHR
- VkVideoReferenceSlotInfoKHR
- VkVideoSessionCreateInfoKHR
- VkVideoSessionMemoryRequirementsKHR
- VkVideoSessionParametersCreateInfoKHR
- VkVideoSessionParametersUpdateInfoKHR
- Extending VkPhysicalDeviceImageFormatInfo2, VkPhysicalDeviceVideoFormatInfoKHR, VkImageCreateInfo, VkBufferCreateInfo:
  - VkVideoProfileListInfoKHR
- Extending VkQueryPoolCreateInfo:
  - VkVideoProfileInfoKHR
- Extending VkQueueFamilyProperties2:
  - VkQueueFamilyQueryResultStatusPropertiesKHR
  - VkQueueFamilyVideoPropertiesKHR

**New Enums**

- VkQueryResultStatusKHR
- VkVideoCapabilityFlagBitsKHR
- VkVideoChromaSubsamplingFlagBitsKHR

- VkVideoCodecOperationFlagBitsKHR
- VkVideoCodingControlFlagBitsKHR
- VkVideoComponentBitDepthFlagBitsKHR
- VkVideoSessionCreateFlagBitsKHR

### New Bitmasks

- VkVideoBeginCodingFlagsKHR
- VkVideoCapabilityFlagsKHR
- VkVideoChromaSubsamplingFlagsKHR
- VkVideoCodecOperationFlagsKHR
- VkVideoCodingControlFlagsKHR
- VkVideoComponentBitDepthFlagsKHR
- VkVideoEndCodingFlagsKHR
- VkVideoSessionCreateFlagsKHR
- VkVideoSessionParametersCreateFlagsKHR

### New Enum Constants

- `VK_KHR_VIDEO_QUEUE_EXTENSION_NAME`
- `VK_KHR_VIDEO_QUEUE_SPEC_VERSION`
- Extending VkObjectType:
  - `VK_OBJECT_TYPE_VIDEO_SESSION_KHR`
  - `VK_OBJECT_TYPE_VIDEO_SESSION_PARAMETERS_KHR`
- Extending VkQueryResultFlagBits:
  - `VK_QUERY_RESULT_WITH_STATUS_BIT_KHR`
- Extending VkQueryType:
  - `VK_QUERY_TYPE_RESULT_STATUS_ONLY_KHR`
- Extending VkResult:
  - `VK_ERROR_IMAGE_USAGE_NOT_SUPPORTED_KHR`
  - `VK_ERROR_VIDEO_PICTURE_LAYOUT_NOT_SUPPORTED_KHR`
  - `VK_ERROR_VIDEO_PROFILE_CODEC_NOT_SUPPORTED_KHR`
  - `VK_ERROR_VIDEO_PROFILE_FORMAT_NOT_SUPPORTED_KHR`
  - `VK_ERROR_VIDEO_PROFILE_OPERATION_NOT_SUPPORTED_KHR`
  - `VK_ERROR_VIDEO_STD_VERSION_NOT_SUPPORTED_KHR`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_BIND_VIDEO_SESSION_MEMORY_INFO_KHR`

- VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_VIDEO_FORMAT_INFO_KHR
- VK_STRUCTURE_TYPE_QUEUE_FAMILY_QUERY_RESULT_STATUS_PROPERTIES_KHR
- VK_STRUCTURE_TYPE_QUEUE_FAMILY_VIDEO_PROPERTIES_KHR
- VK_STRUCTURE_TYPE_VIDEO_BEGIN_CODING_INFO_KHR
- VK_STRUCTURE_TYPE_VIDEO_CAPABILITIES_KHR
- VK_STRUCTURE_TYPE_VIDEO_CODING_CONTROL_INFO_KHR
- VK_STRUCTURE_TYPE_VIDEO_END_CODING_INFO_KHR
- VK_STRUCTURE_TYPE_VIDEO_FORMAT_PROPERTIES_KHR
- VK_STRUCTURE_TYPE_VIDEO_PICTURE_RESOURCE_INFO_KHR
- VK_STRUCTURE_TYPE_VIDEO_PROFILE_INFO_KHR
- VK_STRUCTURE_TYPE_VIDEO_PROFILE_LIST_INFO_KHR
- VK_STRUCTURE_TYPE_VIDEO_REFERENCE_SLOT_INFO_KHR
- VK_STRUCTURE_TYPE_VIDEO_SESSION_CREATE_INFO_KHR
- VK_STRUCTURE_TYPE_VIDEO_SESSION_MEMORY_REQUIREMENTS_KHR
- VK_STRUCTURE_TYPE_VIDEO_SESSION_PARAMETERS_CREATE_INFO_KHR
- VK_STRUCTURE_TYPE_VIDEO_SESSION_PARAMETERS_UPDATE_INFO_KHR

**Version History**

- Revision 0.1, 2019-11-21 (Tony Zlatinski)
  - Initial draft
- Revision 0.2, 2019-11-27 (Tony Zlatinski)
  - Make vulkan video core common between decode and encode
- Revision 1, March 29 2021 (Tony Zlatinski)
  - Spec and API updates.
- Revision 2, August 1 2021 (Srinath Kumarapuram)
  - Rename VkVideoCapabilitiesFlagBitsKHR to VkVideoCapabilityFlagBitsKHR (along with the names of enumerants it defines) and VkVideoCapabilitiesFlagsKHR to VkVideoCapabilityFlagsKHR, following Vulkan naming conventions.
- Revision 3, 2022-03-16 (Ahmed Abdelkhalek)
  - Relocate Std header version reporting/requesting from codec-operation specific extensions to this extension.
  - Make Std header versions codec-operation specific instead of only codec-specific.
- Revision 4, 2022-05-30 (Daniel Rakos)
  - Refactor the video format query APIs and related language
  - Extend VkResult with video-specific error codes
- Revision 5, 2022-08-11 (Daniel Rakos)

  - Add VkVideoSessionParametersCreateFlagsKHR
  - Remove VkVideoCodingQualityPresetFlagsKHR
  - Rename VkQueueFamilyQueryResultStatusProperties2KHR to VkQueueFamilyQueryResultStatusPropertiesKHR
  - Rename VkVideoQueueFamilyProperties2KHR to VkQueueFamilyVideoPropertiesKHR
  - Rename VkVideoProfileKHR to VkVideoProfileInfoKHR
  - Rename VkVideoProfilesKHR to VkVideoProfileListInfoKHR
  - Rename VkVideoGetMemoryPropertiesKHR to VkVideoSessionMemoryRequirementsKHR
  - Rename VkVideoBindMemoryKHR to VkBindVideoSessionMemoryInfoKHR
  - Fix pNext constness of VkPhysicalDeviceVideoFormatInfoKHR and VkVideoSessionMemoryRequirementsKHR
  - Fix incorrectly named value enums in bit enum types VkVideoCodecOperationFlagBitsKHR and VkVideoChromaSubsamplingFlagBitsKHR
  - Remove unnecessary default values from VkVideoSessionCreateFlagBitsKHR and VkVideoCodingControlFlagBitsKHR
  - Eliminate nested pointer in VkVideoSessionMemoryRequirementsKHR
  - Rename VkVideoPictureResourceKHR to VkVideoPictureResourceInfoKHR
  - Rename VkVideoReferenceSlotKHR to VkVideoReferenceSlotInfoKHR
- Revision 6, 2022-09-18 (Daniel Rakos)
  - Rename the maxReferencePicturesSlotsCount and maxReferencePicturesActiveCount fields of VkVideoCapabilitiesKHR and VkVideoSessionCreateInfoKHR to maxDpbSlots and maxActiveReferencePictures, respectively, to clarify their meaning
  - Rename capabilityFlags to flags in VkVideoCapabilitiesKHR
  - Rename videoPictureExtentGranularity to pictureAccessGranularity in VkVideoCapabilitiesKHR
  - Rename minExtent and maxExtent to minCodedExtent and maxCodedExtent, respectively, in VkVideoCapabilitiesKHR
  - Rename referencePicturesFormat to referencePictureFormat in VkVideoSessionCreateInfoKHR
- Revision 7, 2022-09-26 (Daniel Rakos)
  - Change type of VkVideoReferenceSlotInfoKHR::slotIndex from int8_t to int32_t
- Revision 8, 2022-09-29 (Daniel Rakos)
  - Extension is no longer provisional

## VK_KHR_vulkan_memory_model

### Name String

VK_KHR_vulkan_memory_model

### Extension Type

Device extension

### Registered Extension Number

212

### Revision

3

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### SPIR-V Dependencies

- SPV_KHR_vulkan_memory_model

### Deprecation State

- *Promoted* to Vulkan 1.2

### Contact

- Jeff Bolz [GitHub]jeffbolznv

### Other Extension Metadata

### Last Modified Date

2018-12-10

### IP Status

No known IP claims.

### Contributors

- Jeff Bolz, NVIDIA
- Alan Baker, Google
- Tobias Hector, AMD
- David Neto, Google
- Robert Simpson, Qualcomm Technologies, Inc.
- Brian Sumner, AMD

### Description

The VK_KHR_vulkan_memory_model extension allows use of the features guarded by the `VulkanMemoryModel`, `VulkanMemoryModelDeviceScope`, and `VulkanMemoryModelAvailabilityVisibilityChains` capabilities in shader modules. The Vulkan

Memory Model formally defines how to synchronize memory accesses to the same memory locations performed by multiple shader invocations.

| | |
|---|---|
| **NOTE** | Version 3 of the spec added a member (`vulkanMemoryModelAvailabilityVisibilityChains`) to VkPhysicalDeviceVulkanMemoryModelFeaturesKHR, which is an incompatible change from version 2. |

#### Promotion to Vulkan 1.2

All functionality in this extension is included in core Vulkan 1.2, with the KHR suffix omitted. However, if Vulkan 1.2 is supported and this extension is not, the `vulkanMemoryModel` capability is optional. The original type, enum and command names are still available as aliases of the core functionality.

#### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceVulkanMemoryModelFeaturesKHR

#### New Enum Constants

- `VK_KHR_VULKAN_MEMORY_MODEL_EXTENSION_NAME`
- `VK_KHR_VULKAN_MEMORY_MODEL_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_VULKAN_MEMORY_MODEL_FEATURES_KHR`

#### New SPIR-V Capabilities

- `VulkanMemoryModelKHR`

#### Version History

- Revision 1, 2018-06-24 (Jeff Bolz)
  - Initial draft
- Revision 3, 2018-12-10 (Jeff Bolz)
  - Add vulkanMemoryModelAvailabilityVisibilityChains member to the VkPhysicalDeviceVulkanMemoryModelFeaturesKHR structure.

## VK_KHR_wayland_surface

#### Name String

`VK_KHR_wayland_surface`

#### Extension Type

Instance extension

**Registered Extension Number**

7

**Revision**

6

**Ratification Status**

Ratified

**Extension and Version Dependencies**

VK_KHR_surface

**Contact**

- Jesse Hall [GitHub]critsec
- Ian Elliott [GitHub]ianelliottus

**Other Extension Metadata**

**Last Modified Date**

2015-11-28

**IP Status**

No known IP claims.

**Contributors**

- Patrick Doane, Blizzard
- Faith Ekstrand, Intel
- Ian Elliott, LunarG
- Courtney Goeltzenleuchter, LunarG
- Jesse Hall, Google
- James Jones, NVIDIA
- Antoine Labour, Google
- Jon Leech, Khronos
- David Mao, AMD
- Norbert Nopper, Freescale
- Alon Or-bach, Samsung
- Daniel Rakos, AMD
- Graham Sellers, AMD
- Ray Smith, ARM
- Jeff Vigil, Qualcomm
- Chia-I Wu, LunarG

### Description

The `VK_KHR_wayland_surface` extension is an instance extension. It provides a mechanism to create a VkSurfaceKHR object (defined by the `VK_KHR_surface` extension) that refers to a Wayland `wl_surface`, as well as a query to determine support for rendering to a Wayland compositor.

### New Commands

- vkCreateWaylandSurfaceKHR
- vkGetPhysicalDeviceWaylandPresentationSupportKHR

### New Structures

- VkWaylandSurfaceCreateInfoKHR

### New Bitmasks

- VkWaylandSurfaceCreateFlagsKHR

### New Enum Constants

- `VK_KHR_WAYLAND_SURFACE_EXTENSION_NAME`
- `VK_KHR_WAYLAND_SURFACE_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_WAYLAND_SURFACE_CREATE_INFO_KHR`

### Issues

1) Does Wayland need a way to query for compatibility between a particular physical device and a specific Wayland display? This would be a more general query than vkGetPhysicalDeviceSurfaceSupportKHR: if the Wayland-specific query returned `VK_TRUE` for a (VkPhysicalDevice, `struct wl_display*`) pair, then the physical device could be assumed to support presentation to any VkSurfaceKHR for surfaces on the display.

**RESOLVED**: Yes. vkGetPhysicalDeviceWaylandPresentationSupportKHR was added to address this issue.

2) Should we require surfaces created with vkCreateWaylandSurfaceKHR to support the `VK_PRESENT_MODE_MAILBOX_KHR` present mode?

**RESOLVED**: Yes. Wayland is an inherently mailbox window system and mailbox support is required for some Wayland compositor interactions to work as expected. While handling these interactions may be possible with `VK_PRESENT_MODE_FIFO_KHR`, it is much more difficult to do without deadlock and requiring all Wayland applications to be able to support implementations which only support `VK_PRESENT_MODE_FIFO_KHR` would be an onerous restriction on application developers.

### Version History

- Revision 1, 2015-09-23 (Jesse Hall)

    - Initial draft, based on the previous contents of VK_EXT_KHR_swapchain (later renamed VK_EXT_KHR_surface).
- Revision 2, 2015-10-02 (James Jones)
    - Added vkGetPhysicalDeviceWaylandPresentationSupportKHR() to resolve issue #1.
    - Adjusted wording of issue #1 to match the agreed-upon solution.
    - Renamed “window” parameters to “surface” to match Wayland conventions.
- Revision 3, 2015-10-26 (Ian Elliott)
    - Renamed from VK_EXT_KHR_wayland_surface to VK_KHR_wayland_surface.
- Revision 4, 2015-11-03 (Daniel Rakos)
    - Added allocation callbacks to vkCreateWaylandSurfaceKHR.
- Revision 5, 2015-11-28 (Daniel Rakos)
    - Updated the surface create function to take a pCreateInfo structure.
- Revision 6, 2017-02-08 (Faith Ekstrand)
    - Added the requirement that implementations support `VK_PRESENT_MODE_MAILBOX_KHR`.
    - Added wording about interactions between vkQueuePresentKHR and the Wayland requests sent to the compositor.

## VK_KHR_win32_keyed_mutex

### Name String

`VK_KHR_win32_keyed_mutex`

### Extension Type

Device extension

### Registered Extension Number

76

### Revision

1

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_external_memory_win32

### Contact

- Carsten Rohde [GitHub]crohde

### Other Extension Metadata

#### Last Modified Date

2016-10-21

#### IP Status

No known IP claims.

#### Contributors

- James Jones, NVIDIA
- Jeff Juliano, NVIDIA
- Carsten Rohde, NVIDIA

#### Description

Applications that wish to import Direct3D 11 memory objects into the Vulkan API may wish to use the native keyed mutex mechanism to synchronize access to the memory between Vulkan and Direct3D. This extension provides a way for an application to access the keyed mutex associated with an imported Vulkan memory object when submitting command buffers to a queue.

#### New Structures

- Extending VkSubmitInfo, VkSubmitInfo2:
  - VkWin32KeyedMutexAcquireReleaseInfoKHR

#### New Enum Constants

- `VK_KHR_WIN32_KEYED_MUTEX_EXTENSION_NAME`
- `VK_KHR_WIN32_KEYED_MUTEX_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_WIN32_KEYED_MUTEX_ACQUIRE_RELEASE_INFO_KHR`

#### Version History

- Revision 1, 2016-10-21 (James Jones)
  - Initial revision

### VK_KHR_win32_surface

#### Name String

`VK_KHR_win32_surface`

#### Extension Type

Instance extension

#### Registered Extension Number

10

**Revision**

6

**Ratification Status**

Ratified

**Extension and Version Dependencies**

VK_KHR_surface

**Contact**

- Jesse Hall [GitHub]critsec
- Ian Elliott [GitHub]ianelliottus

**Other Extension Metadata**

**Last Modified Date**

2017-04-24

**IP Status**

No known IP claims.

**Contributors**

- Patrick Doane, Blizzard
- Faith Ekstrand, Intel
- Ian Elliott, LunarG
- Courtney Goeltzenleuchter, LunarG
- Jesse Hall, Google
- James Jones, NVIDIA
- Antoine Labour, Google
- Jon Leech, Khronos
- David Mao, AMD
- Norbert Nopper, Freescale
- Alon Or-bach, Samsung
- Daniel Rakos, AMD
- Graham Sellers, AMD
- Ray Smith, ARM
- Jeff Vigil, Qualcomm
- Chia-I Wu, LunarG

**Description**

The `VK_KHR_win32_surface` extension is an instance extension. It provides a mechanism to create a

VkSurfaceKHR object (defined by the `VK_KHR_surface` extension) that refers to a Win32 `HWND`, as well as a query to determine support for rendering to the windows desktop.

### New Commands

- vkCreateWin32SurfaceKHR
- vkGetPhysicalDeviceWin32PresentationSupportKHR

### New Structures

- VkWin32SurfaceCreateInfoKHR

### New Bitmasks

- VkWin32SurfaceCreateFlagsKHR

### New Enum Constants

- `VK_KHR_WIN32_SURFACE_EXTENSION_NAME`
- `VK_KHR_WIN32_SURFACE_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_WIN32_SURFACE_CREATE_INFO_KHR`

### Issues

1) Does Win32 need a way to query for compatibility between a particular physical device and a specific screen? Compatibility between a physical device and a window generally only depends on what screen the window is on. However, there is not an obvious way to identify a screen without already having a window on the screen.

**RESOLVED**: No. While it may be useful, there is not a clear way to do this on Win32. However, a method was added to query support for presenting to the windows desktop as a whole.

2) If a native window object (`HWND`) is used by one graphics API, and then is later used by a different graphics API (one of which is Vulkan), can these uses interfere with each other?

**RESOLVED**: Yes.

Uses of a window object by multiple graphics APIs results in undefined behavior. Such behavior may succeed when using one Vulkan implementation but fail when using a different Vulkan implementation. Potential failures include:

- Creating then destroying a flip presentation model DXGI swapchain on a window object can prevent vkCreateSwapchainKHR from succeeding on the same window object.
- Creating then destroying a VkSwapchainKHR on a window object can prevent creation of a bitblt model DXGI swapchain on the same window object.
- Creating then destroying a VkSwapchainKHR on a window object can effectively `SetPixelFormat` to a different format than the format chosen by an OpenGL application.

- Creating then destroying a VkSwapchainKHR on a window object on one VkPhysicalDevice can prevent vkCreateSwapchainKHR from succeeding on the same window object, but on a different VkPhysicalDevice that is associated with a different Vulkan ICD.

In all cases the problem can be worked around by creating a new window object.

Technical details include:

- Creating a DXGI swapchain over a window object can alter the object for the remainder of its lifetime. The alteration persists even after the DXGI swapchain has been destroyed. This alteration can make it impossible for a conformant Vulkan implementation to create a VkSwapchainKHR over the same window object. Mention of this alteration can be found in the remarks section of the MSDN documentation for `DXGI_SWAP_EFFECT`.
- Calling GDI's `SetPixelFormat` (needed by OpenGL's WGL layer) on a window object alters the object for the remainder of its lifetime. The MSDN documentation for `SetPixelFormat` explains that a window object's pixel format can be set only one time.
- Creating a VkSwapchainKHR over a window object can alter the object for its remaining lifetime. Either of the above alterations may occur as a side effect of vkCreateSwapchainKHR.

#### Version History

- Revision 1, 2015-09-23 (Jesse Hall)
  - Initial draft, based on the previous contents of VK_EXT_KHR_swapchain (later renamed VK_EXT_KHR_surface).
- Revision 2, 2015-10-02 (James Jones)
  - Added presentation support query for win32 desktops.
- Revision 3, 2015-10-26 (Ian Elliott)
  - Renamed from VK_EXT_KHR_win32_surface to VK_KHR_win32_surface.
- Revision 4, 2015-11-03 (Daniel Rakos)
  - Added allocation callbacks to vkCreateWin32SurfaceKHR.
- Revision 5, 2015-11-28 (Daniel Rakos)
  - Updated the surface create function to take a pCreateInfo structure.
- Revision 6, 2017-04-24 (Jeff Juliano)
  - Add issue 2 addressing reuse of a native window object in a different Graphics API, or by a different Vulkan ICD.

## VK_KHR_workgroup_memory_explicit_layout

#### Name String

`VK_KHR_workgroup_memory_explicit_layout`

#### Extension Type

Device extension

### Registered Extension Number

337

### Revision

1

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### SPIR-V Dependencies

- SPV_KHR_workgroup_memory_explicit_layout

### Contact

- Caio Marcelo de Oliveira Filho [GitHub]cmarcelo

### Other Extension Metadata

### Last Modified Date

2020-06-01

### IP Status

No known IP claims.

### Interactions and External Dependencies

- This extension provides API support for `GL_EXT_shared_memory_block`

### Contributors

- Caio Marcelo de Oliveira Filho, Intel
- Jeff Bolz, NVIDIA
- Graeme Leese, Broadcom
- Faith Ekstrand, Intel
- Daniel Koch, NVIDIA

### Description

This extension adds Vulkan support for the `SPV_KHR_workgroup_memory_explicit_layout` SPIR-V extension, which allows shaders to explicitly define the layout of `Workgroup` storage class memory and create aliases between variables from that storage class in a compute shader.

The aliasing feature allows different “views” on the same data, so the shader can bulk copy data from another storage class using one type (e.g. an array of large vectors), and then use the data with a more specific type. It also enables reducing the amount of workgroup memory consumed by allowing the shader to alias data whose lifetimes do not overlap.

The explicit layout support and some form of aliasing is also required for layering OpenCL on top of Vulkan.

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceWorkgroupMemoryExplicitLayoutFeaturesKHR

### New Enum Constants

- `VK_KHR_WORKGROUP_MEMORY_EXPLICIT_LAYOUT_EXTENSION_NAME`
- `VK_KHR_WORKGROUP_MEMORY_EXPLICIT_LAYOUT_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_WORKGROUP_MEMORY_EXPLICIT_LAYOUT_FEATURES_KHR`

### New SPIR-V Capabilities

- `WorkgroupMemoryExplicitLayoutKHR`
- `WorkgroupMemoryExplicitLayout8BitAccessKHR`
- `WorkgroupMemoryExplicitLayout16BitAccessKHR`

### Version History

- Revision 1, 2020-06-01 (Caio Marcelo de Oliveira Filho)
  - Initial version

## VK_KHR_xcb_surface

### Name String

`VK_KHR_xcb_surface`

### Extension Type

Instance extension

### Registered Extension Number

6

### Revision

6

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_surface

### Contact

- Jesse Hall [GitHub]critsec
- Ian Elliott [GitHub]ianelliottus

### Other Extension Metadata

### Last Modified Date

2015-11-28

### IP Status

No known IP claims.

### Contributors

- Patrick Doane, Blizzard
- Faith Ekstrand, Intel
- Ian Elliott, LunarG
- Courtney Goeltzenleuchter, LunarG
- Jesse Hall, Google
- James Jones, NVIDIA
- Antoine Labour, Google
- Jon Leech, Khronos
- David Mao, AMD
- Norbert Nopper, Freescale
- Alon Or-bach, Samsung
- Daniel Rakos, AMD
- Graham Sellers, AMD
- Ray Smith, ARM
- Jeff Vigil, Qualcomm
- Chia-I Wu, LunarG

### Description

The `VK_KHR_xcb_surface` extension is an instance extension. It provides a mechanism to create a VkSurfaceKHR object (defined by the `VK_KHR_surface` extension) that refers to an X11 `Window`, using the XCB client-side library, as well as a query to determine support for rendering via XCB.

### New Commands

- vkCreateXcbSurfaceKHR
- vkGetPhysicalDeviceXcbPresentationSupportKHR

### New Structures

- VkXcbSurfaceCreateInfoKHR

### New Bitmasks

- VkXcbSurfaceCreateFlagsKHR

### New Enum Constants

- `VK_KHR_XCB_SURFACE_EXTENSION_NAME`
- `VK_KHR_XCB_SURFACE_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_XCB_SURFACE_CREATE_INFO_KHR`

### Issues

1) Does XCB need a way to query for compatibility between a particular physical device and a specific screen? This would be a more general query than vkGetPhysicalDeviceSurfaceSupportKHR: If it returned `VK_TRUE`, then the physical device could be assumed to support presentation to any window on that screen.

**RESOLVED**: Yes, this is needed for toolkits that want to create a VkDevice before creating a window. To ensure the query is reliable, it must be made against a particular X visual rather than the screen in general.

### Version History

- Revision 1, 2015-09-23 (Jesse Hall)
  - Initial draft, based on the previous contents of VK_EXT_KHR_swapchain (later renamed VK_EXT_KHR_surface).
- Revision 2, 2015-10-02 (James Jones)
  - Added presentation support query for an (xcb_connection_t*, xcb_visualid_t) pair.
  - Removed “root” parameter from CreateXcbSurfaceKHR(), as it is redundant when a window on the same screen is specified as well.
  - Adjusted wording of issue #1 and added agreed upon resolution.
- Revision 3, 2015-10-14 (Ian Elliott)
  - Removed “root” parameter from CreateXcbSurfaceKHR() in one more place.
- Revision 4, 2015-10-26 (Ian Elliott)
  - Renamed from VK_EXT_KHR_xcb_surface to VK_KHR_xcb_surface.
- Revision 5, 2015-10-23 (Daniel Rakos)
  - Added allocation callbacks to vkCreateXcbSurfaceKHR.
- Revision 6, 2015-11-28 (Daniel Rakos)
  - Updated the surface create function to take a pCreateInfo structure.

## VK_KHR_xlib_surface

### Name String

`VK_KHR_xlib_surface`

### Extension Type

Instance extension

### Registered Extension Number

5

### Revision

6

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_surface

### Contact

- Jesse Hall [GitHub]critsec
- Ian Elliott [GitHub]ianelliottus

### Other Extension Metadata

### Last Modified Date

2015-11-28

### IP Status

No known IP claims.

### Contributors

- Patrick Doane, Blizzard
- Faith Ekstrand, Intel
- Ian Elliott, LunarG
- Courtney Goeltzenleuchter, LunarG
- Jesse Hall, Google
- James Jones, NVIDIA
- Antoine Labour, Google
- Jon Leech, Khronos
- David Mao, AMD
- Norbert Nopper, Freescale
- Alon Or-bach, Samsung

- Daniel Rakos, AMD
- Graham Sellers, AMD
- Ray Smith, ARM
- Jeff Vigil, Qualcomm
- Chia-I Wu, LunarG

### Description

The `VK_KHR_xlib_surface` extension is an instance extension. It provides a mechanism to create a VkSurfaceKHR object (defined by the `VK_KHR_surface` extension) that refers to an X11 `Window`, using the Xlib client-side library, as well as a query to determine support for rendering via Xlib.

### New Commands

- vkCreateXlibSurfaceKHR
- vkGetPhysicalDeviceXlibPresentationSupportKHR

### New Structures

- VkXlibSurfaceCreateInfoKHR

### New Bitmasks

- VkXlibSurfaceCreateFlagsKHR

### New Enum Constants

- `VK_KHR_XLIB_SURFACE_EXTENSION_NAME`
- `VK_KHR_XLIB_SURFACE_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_XLIB_SURFACE_CREATE_INFO_KHR`

### Issues

1) Does X11 need a way to query for compatibility between a particular physical device and a specific screen? This would be a more general query than vkGetPhysicalDeviceSurfaceSupportKHR; if it returned `VK_TRUE`, then the physical device could be assumed to support presentation to any window on that screen.

**RESOLVED**: Yes, this is needed for toolkits that want to create a VkDevice before creating a window. To ensure the query is reliable, it must be made against a particular X visual rather than the screen in general.

### Version History

- Revision 1, 2015-09-23 (Jesse Hall)
  - Initial draft, based on the previous contents of VK_EXT_KHR_swapchain (later renamed

VK_EXT_KHR_surface).

- Revision 2, 2015-10-02 (James Jones)
  - Added presentation support query for (Display*, VisualID) pair.
  - Removed “root” parameter from CreateXlibSurfaceKHR(), as it is redundant when a window on the same screen is specified as well.
  - Added appropriate X errors.
  - Adjusted wording of issue #1 and added agreed upon resolution.
- Revision 3, 2015-10-14 (Ian Elliott)
  - Renamed this extension from VK_EXT_KHR_x11_surface to VK_EXT_KHR_xlib_surface.
- Revision 4, 2015-10-26 (Ian Elliott)
  - Renamed from VK_EXT_KHR_xlib_surface to VK_KHR_xlib_surface.
- Revision 5, 2015-11-03 (Daniel Rakos)
  - Added allocation callbacks to vkCreateXlibSurfaceKHR.
- Revision 6, 2015-11-28 (Daniel Rakos)
  - Updated the surface create function to take a pCreateInfo structure.

## VK_KHR_zero_initialize_workgroup_memory

### Name String

`VK_KHR_zero_initialize_workgroup_memory`

### Extension Type

Device extension

### Registered Extension Number

326

### Revision

1

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### Deprecation State

- *Promoted* to Vulkan 1.3

### Contact

- Alan Baker [GitHub]alan-baker

#### Other Extension Metadata

#### Last Modified Date

2020-11-18

#### IP Status

No known IP claims.

#### Contributors

- Alan Baker, Google
- Jeff Bolz, Nvidia
- Faith Ekstrand, Intel

#### Description

This extension allows the use of a null constant initializer on shader Workgroup memory variables, allowing implementations to expose any special hardware or instructions they may have. Zero initialization is commonly used by applications running untrusted content (e.g. web browsers) as way of defeating memory-scraping attacks.

#### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceZeroInitializeWorkgroupMemoryFeaturesKHR

#### New Enum Constants

- `VK_KHR_ZERO_INITIALIZE_WORKGROUP_MEMORY_EXTENSION_NAME`
- `VK_KHR_ZERO_INITIALIZE_WORKGROUP_MEMORY_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_ZERO_INITIALIZE_WORKGROUP_MEMORY_FEATURES_KHR`

#### Promotion to Vulkan 1.3

Functionality in this extension is included in core Vulkan 1.3, with the KHR suffix omitted. The original type, enum and command names are still available as aliases of the core functionality.

#### Version History

- Revision 1, 2020-11-18 (Alan Baker)
  - Internal draft version

## VK_EXT_4444_formats

#### Name String

`VK_EXT_4444_formats`

### Extension Type

Device extension

### Registered Extension Number

341

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### Deprecation State

- *Promoted* to Vulkan 1.3

### Contact

- Joshua Ashton [GitHub]Joshua-Ashton

### Other Extension Metadata

### Last Modified Date

2020-07-28

### IP Status

No known IP claims.

### Contributors

- Joshua Ashton, Valve
- Faith Ekstrand, Intel

### Description

This extension defines the `VK_FORMAT_A4R4G4B4_UNORM_PACK16_EXT` and `VK_FORMAT_A4B4G4R4_UNORM_PACK16_EXT` formats which are defined in other current graphics APIs.

This extension may be useful for building translation layers for those APIs or for porting applications that use these formats without having to resort to swizzles.

When VK_EXT_custom_border_color is used, these formats are not subject to the same restrictions for border color without format as with VK_FORMAT_B4G4R4A4_UNORM_PACK16.

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:

- VkPhysicalDevice4444FormatsFeaturesEXT

#### New Enum Constants

- `VK_EXT_4444_FORMATS_EXTENSION_NAME`
- `VK_EXT_4444_FORMATS_SPEC_VERSION`
- Extending VkFormat:
  - `VK_FORMAT_A4B4G4R4_UNORM_PACK16_EXT`
  - `VK_FORMAT_A4R4G4B4_UNORM_PACK16_EXT`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_4444_FORMATS_FEATURES_EXT`

#### Promotion to Vulkan 1.3

This extension has been partially promoted. The format enumerants introduced by the extension are included in core Vulkan 1.3, with the EXT suffix omitted. However, runtime support for these formats is optional in core Vulkan 1.3, while if this extension is supported, runtime support is mandatory. The feature structure is not promoted. The original enum names are still available as aliases of the core functionality.

#### Version History

- Revision 1, 2020-07-04 (Joshua Ashton)
  - Initial draft

## VK_EXT_acquire_drm_display

#### Name String

`VK_EXT_acquire_drm_display`

#### Extension Type

Instance extension

#### Registered Extension Number

286

#### Revision

1

#### Ratification Status

Not ratified

#### Extension and Version Dependencies

VK_EXT_direct_mode_display

### Contact

- Drew DeVault sir@cmpwn.com

### Other Extension Metadata

### Last Modified Date

2021-06-09

### IP Status

No known IP claims.

### Contributors

- Simon Zeni, Status Holdings, Ltd.

### Description

This extension allows an application to take exclusive control of a display using the Direct Rendering Manager (DRM) interface. When acquired, the display will be under full control of the application until the display is either released or the connector is unplugged.

### New Commands

- vkAcquireDrmDisplayEXT
- vkGetDrmDisplayEXT

### New Enum Constants

- `VK_EXT_ACQUIRE_DRM_DISPLAY_EXTENSION_NAME`
- `VK_EXT_ACQUIRE_DRM_DISPLAY_SPEC_VERSION`

### Issues

None.

### Version History

- Revision 1, 2021-05-11 (Simon Zeni)
  - Initial draft

## VK_EXT_acquire_xlib_display

### Name String

`VK_EXT_acquire_xlib_display`

### Extension Type

Instance extension

**Registered Extension Number**

90

**Revision**

1

**Ratification Status**

Not ratified

**Extension and Version Dependencies**

VK_EXT_direct_mode_display

**Contact**

- James Jones [GitHub]cubanismo

**Other Extension Metadata**

**Last Modified Date**

2016-12-13

**IP Status**

No known IP claims.

**Contributors**

- Dave Airlie, Red Hat
- Pierre Boudier, NVIDIA
- James Jones, NVIDIA
- Damien Leone, NVIDIA
- Pierre-Loup Griffais, Valve
- Liam Middlebrook, NVIDIA
- Daniel Vetter, Intel

**Description**

This extension allows an application to take exclusive control on a display currently associated with an X11 screen. When control is acquired, the display will be deassociated from the X11 screen until control is released or the specified display connection is closed. Essentially, the X11 screen will behave as if the monitor has been unplugged until control is released.

**New Commands**

- vkAcquireXlibDisplayEXT
- vkGetRandROutputDisplayEXT

### New Enum Constants

- `VK_EXT_ACQUIRE_XLIB_DISPLAY_EXTENSION_NAME`
- `VK_EXT_ACQUIRE_XLIB_DISPLAY_SPEC_VERSION`

### Issues

1) Should vkAcquireXlibDisplayEXT take an RandR display ID, or a Vulkan display handle as input?

**RESOLVED**: A Vulkan display handle. Otherwise there would be no way to specify handles to displays that had been prevented from being included in the X11 display list by some native platform or vendor-specific mechanism.

2) How does an application figure out which RandR display corresponds to a Vulkan display?

**RESOLVED**: A new function, vkGetRandROutputDisplayEXT, is introduced for this purpose.

3) Should vkGetRandROutputDisplayEXT be part of this extension, or a general Vulkan / RandR or Vulkan / Xlib extension?

**RESOLVED**: To avoid yet another extension, include it in this extension.

### Version History

- Revision 1, 2016-12-13 (James Jones)
  - Initial draft

## VK_EXT_astc_decode_mode

### Name String

`VK_EXT_astc_decode_mode`

### Extension Type

Device extension

### Registered Extension Number

68

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### Contact

- Jan-Harald Fredriksen [GitHub]janharaldfredriksen-arm

### Other Extension Metadata

### Last Modified Date

2018-08-07

### Contributors

- Jan-Harald Fredriksen, Arm

### Description

The existing specification requires that low dynamic range (LDR) ASTC textures are decompressed to FP16 values per component. In many cases, decompressing LDR textures to a lower precision intermediate result gives acceptable image quality. Source material for LDR textures is typically authored as 8-bit UNORM values, so decoding to FP16 values adds little value. On the other hand, reducing precision of the decoded result reduces the size of the decompressed data, potentially improving texture cache performance and saving power.

The goal of this extension is to enable this efficiency gain on existing ASTC texture data. This is achieved by giving the application the ability to select the intermediate decoding precision.

Three decoding options are provided:

- Decode to `VK_FORMAT_R16G16B16A16_SFLOAT` precision: This is the default, and matches the required behavior in the core API.
- Decode to `VK_FORMAT_R8G8B8A8_UNORM` precision: This is provided as an option in LDR mode.
- Decode to `VK_FORMAT_E5B9G9R9_UFLOAT_PACK32` precision: This is provided as an option in both LDR and HDR mode. In this mode, negative values cannot be represented and are clamped to zero. The alpha component is ignored, and the results are as if alpha was 1.0. This decode mode is optional and support can be queried via the physical device properties.

### New Structures

- Extending VkImageViewCreateInfo:
  - VkImageViewASTCDecodeModeEXT
- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceASTCDecodeFeaturesEXT

### New Enum Constants

- `VK_EXT_ASTC_DECODE_MODE_EXTENSION_NAME`
- `VK_EXT_ASTC_DECODE_MODE_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_IMAGE_VIEW_ASTC_DECODE_MODE_EXT`

- VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_ASTC_DECODE_FEATURES_EXT

**Issues**

1) Are implementations allowed to decode at a higher precision than what is requested?

```
RESOLUTION: No.
If we allow this, then this extension could be exposed on all
implementations that support ASTC.
But developers would have no way of knowing what precision was actually
used, and thus whether the image quality is sufficient at reduced
precision.
```

2) Should the decode mode be image view state and/or sampler state?

```
RESOLUTION: Image view state only.
Some implementations treat the different decode modes as different
texture formats.
```

**Example**

Create an image view that decodes to VK_FORMAT_R8G8B8A8_UNORM precision:

```
VkImageViewASTCDecodeModeEXT decodeMode =
{
    .sType = VK_STRUCTURE_TYPE_IMAGE_VIEW_ASTC_DECODE_MODE_EXT,
    .pNext = NULL,
    .decodeMode = VK_FORMAT_R8G8B8A8_UNORM
};

VkImageViewCreateInfo createInfo =
{
    .sType = VK_STRUCTURE_TYPE_IMAGE_VIEW_CREATE_INFO,
    .pNext = &decodeMode,
    // flags, image, viewType set to application-desired values
    .format = VK_FORMAT_ASTC_8x8_UNORM_BLOCK,
    // components, subresourceRange set to application-desired values
};

VkImageView imageView;
VkResult result = vkCreateImageView(
    device,
    &createInfo,
    NULL,
    &imageView);
```

### Version History

- Revision 1, 2018-08-07 (Jan-Harald Fredriksen)
  - Initial revision

## VK_EXT_attachment_feedback_loop_dynamic_state

### Name String

`VK_EXT_attachment_feedback_loop_dynamic_state`

### Extension Type

Device extension

### Registered Extension Number

525

### Revision

1

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1
and
VK_EXT_attachment_feedback_loop_layout

### Contact

- Mike Blumenkrantz [GitHub]zmike

### Extension Proposal

VK_EXT_attachment_feedback_loop_dynamic_state

### Other Extension Metadata

### Last Modified Date

2023-04-28

### IP Status

No known IP claims.

### Contributors

- Mike Blumenkrantz, Valve
- Daniel Story, Nintendo
- Stu Smith, AMD

- Samuel Pitoiset, Valve
- Ricardo Garcia, Igalia

#### Description

This extension adds support for setting attachment feedback loops dynamically on command buffers.

#### New Commands

- vkCmdSetAttachmentFeedbackLoopEnableEXT

#### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceAttachmentFeedbackLoopDynamicStateFeaturesEXT

#### New Enum Constants

- `VK_EXT_ATTACHMENT_FEEDBACK_LOOP_DYNAMIC_STATE_EXTENSION_NAME`
- `VK_EXT_ATTACHMENT_FEEDBACK_LOOP_DYNAMIC_STATE_SPEC_VERSION`
- Extending VkDynamicState:
  - `VK_DYNAMIC_STATE_ATTACHMENT_FEEDBACK_LOOP_ENABLE_EXT`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_ATTACHMENT_FEEDBACK_LOOP_DYNAMIC_STATE_FEATURES_EXT`

#### Version History

- Revision 1, 2023-04-28 (Mike Blumenkrantz)
  - Initial revision

## VK_EXT_attachment_feedback_loop_layout

#### Name String

`VK_EXT_attachment_feedback_loop_layout`

#### Extension Type

Device extension

#### Registered Extension Number

340

#### Revision

2

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### Contact

- Joshua Ashton [GitHub]Joshua-Ashton

### Extension Proposal

VK_EXT_attachment_feedback_loop_layout

### Other Extension Metadata

### Last Modified Date

2022-04-04

### IP Status

No known IP claims.

### Contributors

- Joshua Ashton, Valve
- Faith Ekstrand, Collabora
- Bas Nieuwenhuizen, Google
- Samuel Iglesias Gonsálvez, Igalia
- Ralph Potter, Samsung
- Jan-Harald Fredriksen, Arm
- Ricardo Garcia, Igalia

### Description

This extension adds a new image layout, `VK_IMAGE_LAYOUT_ATTACHMENT_FEEDBACK_LOOP_OPTIMAL_EXT`, which allows applications to have an image layout in which they are able to both render to and sample/fetch from the same subresource of an image in a given render pass.

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceAttachmentFeedbackLoopLayoutFeaturesEXT

### New Enum Constants

- `VK_EXT_ATTACHMENT_FEEDBACK_LOOP_LAYOUT_EXTENSION_NAME`
- `VK_EXT_ATTACHMENT_FEEDBACK_LOOP_LAYOUT_SPEC_VERSION`

- Extending VkDependencyFlagBits:
  - `VK_DEPENDENCY_FEEDBACK_LOOP_BIT_EXT`
- Extending VkImageLayout:
  - `VK_IMAGE_LAYOUT_ATTACHMENT_FEEDBACK_LOOP_OPTIMAL_EXT`
- Extending VkImageUsageFlagBits:
  - `VK_IMAGE_USAGE_ATTACHMENT_FEEDBACK_LOOP_BIT_EXT`
- Extending VkPipelineCreateFlagBits:
  - `VK_PIPELINE_CREATE_COLOR_ATTACHMENT_FEEDBACK_LOOP_BIT_EXT`
  - `VK_PIPELINE_CREATE_DEPTH_STENCIL_ATTACHMENT_FEEDBACK_LOOP_BIT_EXT`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_ATTACHMENT_FEEDBACK_LOOP_LAYOUT_FEATURES_EXT`

#### Version History

- Revision 2, 2022-04-04 (Joshua Ashton)
  - Renamed from VALVE to EXT.
- Revision 1, 2021-03-09 (Joshua Ashton)
  - Initial draft.

## VK_EXT_blend_operation_advanced

#### Name String

`VK_EXT_blend_operation_advanced`

#### Extension Type

Device extension

#### Registered Extension Number

149

#### Revision

2

#### Ratification Status

Ratified

#### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

#### Contact

- Jeff Bolz [GitHub]jeffbolznv

### Other Extension Metadata

### Last Modified Date

2017-06-12

### Contributors

- Jeff Bolz, NVIDIA

### Description

This extension adds a number of “advanced” blending operations that **can** be used to perform new color blending operations, many of which are more complex than the standard blend modes provided by unextended Vulkan. This extension requires different styles of usage, depending on the level of hardware support and the enabled features:

- If VkPhysicalDeviceBlendOperationAdvancedFeaturesEXT::`advancedBlendCoherentOperations` is `VK_FALSE`, the new blending operations are supported, but a memory dependency **must** separate each advanced blend operation on a given sample. `VK_ACCESS_COLOR_ATTACHMENT_READ_NONCOHERENT_BIT_EXT` is used to synchronize reads using advanced blend operations.
- If VkPhysicalDeviceBlendOperationAdvancedFeaturesEXT::`advancedBlendCoherentOperations` is `VK_TRUE`, advanced blend operations obey primitive order just like basic blend operations.

In unextended Vulkan, the set of blending operations is limited, and **can** be expressed very simply. The `VK_BLEND_OP_MIN` and `VK_BLEND_OP_MAX` blend operations simply compute component-wise minimums or maximums of source and destination color components. The `VK_BLEND_OP_ADD`, `VK_BLEND_OP_SUBTRACT`, and `VK_BLEND_OP_REVERSE_SUBTRACT` modes multiply the source and destination colors by source and destination factors and either add the two products together or subtract one from the other. This limited set of operations supports many common blending operations but precludes the use of more sophisticated transparency and blending operations commonly available in many dedicated imaging APIs.

This extension provides a number of new “advanced” blending operations. Unlike traditional blending operations using `VK_BLEND_OP_ADD`, these blending equations do not use source and destination factors specified by VkBlendFactor. Instead, each blend operation specifies a complete equation based on the source and destination colors. These new blend operations are used for both RGB and alpha components; they **must** not be used to perform separate RGB and alpha blending (via different values of color and alpha VkBlendOp).

These blending operations are performed using premultiplied colors, where RGB colors **can** be considered premultiplied or non-premultiplied by alpha, according to the `srcPremultiplied` and `dstPremultiplied` members of VkPipelineColorBlendAdvancedStateCreateInfoEXT. If a color is considered non-premultiplied, the (R,G,B) color components are multiplied by the alpha component prior to blending. For non-premultiplied color components in the range [0,1], the corresponding premultiplied color component would have values in the range [0 × A, 1 × A].

Many of these advanced blending equations are formulated where the result of blending source and destination colors with partial coverage have three separate contributions: from the portions covered by both the source and the destination, from the portion covered only by the source, and

from the portion covered only by the destination. The blend parameter VkPipelineColorBlendAdvancedStateCreateInfoEXT::blendOverlap **can** be used to specify a correlation between source and destination pixel coverage. If set to VK_BLEND_OVERLAP_CONJOINT_EXT, the source and destination are considered to have maximal overlap, as would be the case if drawing two objects on top of each other. If set to VK_BLEND_OVERLAP_DISJOINT_EXT, the source and destination are considered to have minimal overlap, as would be the case when rendering a complex polygon tessellated into individual non-intersecting triangles. If set to VK_BLEND_OVERLAP_UNCORRELATED_EXT, the source and destination coverage are assumed to have no spatial correlation within the pixel.

In addition to the coherency issues on implementations not supporting advancedBlendCoherentOperations, this extension has several limitations worth noting. First, the new blend operations have a limit on the number of color attachments they **can** be used with, as indicated by VkPhysicalDeviceBlendOperationAdvancedPropertiesEXT::advancedBlendMaxColorAttachments. Additionally, blending precision **may** be limited to 16-bit floating-point, which **may** result in a loss of precision and dynamic range for framebuffer formats with 32-bit floating-point components, and in a loss of precision for formats with 12- and 16-bit signed or unsigned normalized integer components.

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceBlendOperationAdvancedFeaturesEXT
- Extending VkPhysicalDeviceProperties2:
  - VkPhysicalDeviceBlendOperationAdvancedPropertiesEXT
- Extending VkPipelineColorBlendStateCreateInfo:
  - VkPipelineColorBlendAdvancedStateCreateInfoEXT

### New Enums

- VkBlendOverlapEXT

### New Enum Constants

- VK_EXT_BLEND_OPERATION_ADVANCED_EXTENSION_NAME
- VK_EXT_BLEND_OPERATION_ADVANCED_SPEC_VERSION
- Extending VkAccessFlagBits:
  - VK_ACCESS_COLOR_ATTACHMENT_READ_NONCOHERENT_BIT_EXT
- Extending VkBlendOp:
  - VK_BLEND_OP_BLUE_EXT
  - VK_BLEND_OP_COLORBURN_EXT
  - VK_BLEND_OP_COLORDODGE_EXT
  - VK_BLEND_OP_CONTRAST_EXT
  - VK_BLEND_OP_DARKEN_EXT

- VK_BLEND_OP_DIFFERENCE_EXT
- VK_BLEND_OP_DST_ATOP_EXT
- VK_BLEND_OP_DST_EXT
- VK_BLEND_OP_DST_IN_EXT
- VK_BLEND_OP_DST_OUT_EXT
- VK_BLEND_OP_DST_OVER_EXT
- VK_BLEND_OP_EXCLUSION_EXT
- VK_BLEND_OP_GREEN_EXT
- VK_BLEND_OP_HARDLIGHT_EXT
- VK_BLEND_OP_HARDMIX_EXT
- VK_BLEND_OP_HSL_COLOR_EXT
- VK_BLEND_OP_HSL_HUE_EXT
- VK_BLEND_OP_HSL_LUMINOSITY_EXT
- VK_BLEND_OP_HSL_SATURATION_EXT
- VK_BLEND_OP_INVERT_EXT
- VK_BLEND_OP_INVERT_OVG_EXT
- VK_BLEND_OP_INVERT_RGB_EXT
- VK_BLEND_OP_LIGHTEN_EXT
- VK_BLEND_OP_LINEARBURN_EXT
- VK_BLEND_OP_LINEARDODGE_EXT
- VK_BLEND_OP_LINEARLIGHT_EXT
- VK_BLEND_OP_MINUS_CLAMPED_EXT
- VK_BLEND_OP_MINUS_EXT
- VK_BLEND_OP_MULTIPLY_EXT
- VK_BLEND_OP_OVERLAY_EXT
- VK_BLEND_OP_PINLIGHT_EXT
- VK_BLEND_OP_PLUS_CLAMPED_ALPHA_EXT
- VK_BLEND_OP_PLUS_CLAMPED_EXT
- VK_BLEND_OP_PLUS_DARKER_EXT
- VK_BLEND_OP_PLUS_EXT
- VK_BLEND_OP_RED_EXT
- VK_BLEND_OP_SCREEN_EXT
- VK_BLEND_OP_SOFTLIGHT_EXT
- VK_BLEND_OP_SRC_ATOP_EXT
- VK_BLEND_OP_SRC_EXT

    - VK_BLEND_OP_SRC_IN_EXT
    - VK_BLEND_OP_SRC_OUT_EXT
    - VK_BLEND_OP_SRC_OVER_EXT
    - VK_BLEND_OP_VIVIDLIGHT_EXT
    - VK_BLEND_OP_XOR_EXT
    - VK_BLEND_OP_ZERO_EXT
- Extending VkStructureType:
    - VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_BLEND_OPERATION_ADVANCED_FEATURES_EXT
    - VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_BLEND_OPERATION_ADVANCED_PROPERTIES_EXT
    - VK_STRUCTURE_TYPE_PIPELINE_COLOR_BLEND_ADVANCED_STATE_CREATE_INFO_EXT

#### Issues

None.

#### Version History

- Revision 1, 2017-06-12 (Jeff Bolz)
    - Internal revisions
- Revision 2, 2017-06-12 (Jeff Bolz)
    - Internal revisions

## VK_EXT_border_color_swizzle

#### Name String

VK_EXT_border_color_swizzle

#### Extension Type

Device extension

#### Registered Extension Number

412

#### Revision

1

#### Ratification Status

Not ratified

#### Extension and Version Dependencies

VK_EXT_custom_border_color

#### Special Uses

- OpenGL / ES support

- D3D support

**Contact**

- Piers Daniell [GitHub]pdaniell-nv

**Other Extension Metadata**

**Last Modified Date**

2021-10-12

**IP Status**

No known IP claims.

**Contributors**

- Graeme Leese, Broadcom
- Jan-Harald Fredriksen, Arm
- Ricardo Garcia, Igalia
- Shahbaz Youssefi, Google
- Stu Smith, AMD

**Description**

After the publication of VK_EXT_custom_border_color, it was discovered that some implementations had undefined behavior when combining a sampler that uses a custom border color with image views whose component mapping is not the identity mapping.

Since VK_EXT_custom_border_color has already shipped, this new extension VK_EXT_border_color_swizzle was created to define the interaction between custom border colors and non-identity image view swizzles, and provide a work-around for implementations that must pre-swizzle the sampler border color to match the image view component mapping it is combined with.

This extension also defines the behavior between samplers with an opaque black border color and image views with a non-identity component swizzle, which was previously left undefined.

**New Structures**

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceBorderColorSwizzleFeaturesEXT
- Extending VkSamplerCreateInfo:
  - VkSamplerBorderColorComponentMappingCreateInfoEXT

**New Enum Constants**

- `VK_EXT_BORDER_COLOR_SWIZZLE_EXTENSION_NAME`
- `VK_EXT_BORDER_COLOR_SWIZZLE_SPEC_VERSION`

- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_BORDER_COLOR_SWIZZLE_FEATURES_EXT`
  - `VK_STRUCTURE_TYPE_SAMPLER_BORDER_COLOR_COMPONENT_MAPPING_CREATE_INFO_EXT`

#### Issues

None.

#### Version History

- Revision 1, 2021-10-12 (Piers Daniell)
  - Internal revisions.

## VK_EXT_color_write_enable

#### Name String

`VK_EXT_color_write_enable`

#### Extension Type

Device extension

#### Registered Extension Number

382

#### Revision

1

#### Ratification Status

Ratified

#### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

#### Contact

- Sharif Elcott [GitHub]selcott

#### Other Extension Metadata

#### Last Modified Date

2020-02-25

#### IP Status

No known IP claims.

#### Contributors

- Sharif Elcott, Google

- Tobias Hector, AMD
- Piers Daniell, NVIDIA

### Description

This extension allows for selectively enabling and disabling writes to output color attachments via a pipeline dynamic state.

The intended use cases for this new state are mostly identical to those of colorWriteMask, such as selectively disabling writes to avoid feedback loops between subpasses or bandwidth savings for unused outputs. By making the state dynamic, one additional benefit is the ability to reduce pipeline counts and pipeline switching via shaders that write a superset of the desired data of which subsets are selected dynamically. The reason for a new state, colorWriteEnable, rather than making colorWriteMask dynamic is that, on many implementations, the more flexible per-component semantics of the colorWriteMask state cannot be made dynamic in a performant manner.

### New Commands

- vkCmdSetColorWriteEnableEXT

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceColorWriteEnableFeaturesEXT
- Extending VkPipelineColorBlendStateCreateInfo:
  - VkPipelineColorWriteCreateInfoEXT

### New Enum Constants

- `VK_EXT_COLOR_WRITE_ENABLE_EXTENSION_NAME`
- `VK_EXT_COLOR_WRITE_ENABLE_SPEC_VERSION`
- Extending VkDynamicState:
  - `VK_DYNAMIC_STATE_COLOR_WRITE_ENABLE_EXT`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_COLOR_WRITE_ENABLE_FEATURES_EXT`
  - `VK_STRUCTURE_TYPE_PIPELINE_COLOR_WRITE_CREATE_INFO_EXT`

### Version History

- Revision 1, 2020-01-25 (Sharif Elcott)
  - Internal revisions

## VK_EXT_conditional_rendering

### Name String

`VK_EXT_conditional_rendering`

### Extension Type

Device extension

### Registered Extension Number

82

### Revision

2

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### Contact

- Vikram Kushwaha [GitHub]vkushwaha

### Other Extension Metadata

### Last Modified Date

2018-05-21

### IP Status

No known IP claims.

### Contributors

- Vikram Kushwaha, NVIDIA
- Daniel Rakos, AMD
- Jesse Hall, Google
- Jeff Bolz, NVIDIA
- Piers Daniell, NVIDIA
- Stuart Smith, Imagination Technologies

### Description

This extension allows the execution of one or more rendering commands to be conditional on a value in buffer memory. This may help an application reduce the latency by conditionally discarding rendering commands without application intervention. The conditional rendering commands are limited to draws, compute dispatches and clearing attachments within a conditional

rendering block.

### New Commands

- vkCmdBeginConditionalRenderingEXT
- vkCmdEndConditionalRenderingEXT

### New Structures

- VkConditionalRenderingBeginInfoEXT
- Extending VkCommandBufferInheritanceInfo:
  - VkCommandBufferInheritanceConditionalRenderingInfoEXT
- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceConditionalRenderingFeaturesEXT

### New Enums

- VkConditionalRenderingFlagBitsEXT

### New Bitmasks

- VkConditionalRenderingFlagsEXT

### New Enum Constants

- `VK_EXT_CONDITIONAL_RENDERING_EXTENSION_NAME`
- `VK_EXT_CONDITIONAL_RENDERING_SPEC_VERSION`
- Extending VkAccessFlagBits:
  - `VK_ACCESS_CONDITIONAL_RENDERING_READ_BIT_EXT`
- Extending VkBufferUsageFlagBits:
  - `VK_BUFFER_USAGE_CONDITIONAL_RENDERING_BIT_EXT`
- Extending VkPipelineStageFlagBits:
  - `VK_PIPELINE_STAGE_CONDITIONAL_RENDERING_BIT_EXT`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_COMMAND_BUFFER_INHERITANCE_CONDITIONAL_RENDERING_INFO_EXT`
  - `VK_STRUCTURE_TYPE_CONDITIONAL_RENDERING_BEGIN_INFO_EXT`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_CONDITIONAL_RENDERING_FEATURES_EXT`

### Issues

1) Should conditional rendering affect copy and blit commands?

**RESOLVED**: Conditional rendering should not affect copies and blits.

2) Should secondary command buffers be allowed to execute while conditional rendering is active in the primary command buffer?

**RESOLVED**: The rendering commands in secondary command buffer will be affected by an active conditional rendering in primary command buffer if the `conditionalRenderingEnable` is set to `VK_TRUE`. Conditional rendering **must** not be active in the primary command buffer if `conditionalRenderingEnable` is `VK_FALSE`.

### Examples

None.

### Version History

- Revision 1, 2018-04-19 (Vikram Kushwaha)
  - First Version
- Revision 2, 2018-05-21 (Vikram Kushwaha)
  - Add new pipeline stage, access flags and limit conditional rendering to a subpass or entire render pass.

## VK_EXT_conservative_rasterization

### Name String

`VK_EXT_conservative_rasterization`

### Extension Type

Device extension

### Registered Extension Number

102

### Revision

1

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### SPIR-V Dependencies

- SPV_EXT_fragment_fully_covered

### Contact

- Piers Daniell [GitHub]pdaniell-nv

### Other Extension Metadata

### Last Modified Date

2020-06-09

### Interactions and External Dependencies

- This extension requires `SPV_EXT_fragment_fully_covered` if the `VkPhysicalDeviceConservativeRasterizationPropertiesEXT::fullyCoveredFragmentShaderInputVariable` feature is used.
- This extension requires `SPV_KHR_post_depth_coverage`if the `VkPhysicalDeviceConservativeRasterizationPropertiesEXT::conservativeRasterizationPostDepthCoverage` feature is used.
- This extension provides API support for `GL_NV_conservative_raster_underestimation` if the `VkPhysicalDeviceConservativeRasterizationPropertiesEXT::fullyCoveredFragmentShaderInputVariable` feature is used.

### Contributors

- Daniel Koch, NVIDIA
- Daniel Rakos, AMD
- Jeff Bolz, NVIDIA
- Slawomir Grajewski, Intel
- Stu Smith, Imagination Technologies

### Description

This extension adds a new rasterization mode called conservative rasterization. There are two modes of conservative rasterization; overestimation and underestimation.

When overestimation is enabled, if any part of the primitive, including its edges, covers any part of the rectangular pixel area, including its sides, then a fragment is generated with all coverage samples turned on. This extension allows for some variation in implementations by accounting for differences in overestimation, where the generating primitive size is increased at each of its edges by some sub-pixel amount to further increase conservative pixel coverage. Implementations can allow the application to specify an extra overestimation beyond the base overestimation the implementation already does. It also allows implementations to either cull degenerate primitives or rasterize them.

When underestimation is enabled, fragments are only generated if the rectangular pixel area is fully covered by the generating primitive. If supported by the implementation, when a pixel rectangle is fully covered the fragment shader input variable builtin called FullyCoveredEXT is set to true. The shader variable works in either overestimation or underestimation mode.

Implementations can process degenerate triangles and lines by either discarding them or generating conservative fragments for them. Degenerate triangles are those that end up with zero area after the rasterizer quantizes them to the fixed-point pixel grid. Degenerate lines are those with zero length after quantization.

### New Structures

- Extending VkPhysicalDeviceProperties2:
  - VkPhysicalDeviceConservativeRasterizationPropertiesEXT
- Extending VkPipelineRasterizationStateCreateInfo:
  - VkPipelineRasterizationConservativeStateCreateInfoEXT

### New Enums

- VkConservativeRasterizationModeEXT

### New Bitmasks

- VkPipelineRasterizationConservativeStateCreateFlagsEXT

### New Enum Constants

- `VK_EXT_CONSERVATIVE_RASTERIZATION_EXTENSION_NAME`
- `VK_EXT_CONSERVATIVE_RASTERIZATION_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_CONSERVATIVE_RASTERIZATION_PROPERTIES_EXT`
  - `VK_STRUCTURE_TYPE_PIPELINE_RASTERIZATION_CONSERVATIVE_STATE_CREATE_INFO_EXT`

### New Built-In Variables

- FullyCoveredEXT

### New SPIR-V Capabilities

- FragmentFullyCoveredEXT

### Version History

- Revision 1.1, 2020-09-06 (Piers Daniell)
  - Add missing SPIR-V and GLSL dependencies.
- Revision 1, 2017-08-28 (Piers Daniell)
  - Internal revisions

## VK_EXT_custom_border_color

### Name String

`VK_EXT_custom_border_color`

### Extension Type

Device extension

**Registered Extension Number**

288

**Revision**

12

**Ratification Status**

Ratified

**Extension and Version Dependencies**

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

**Special Uses**

- OpenGL / ES support
- D3D support

**Contact**

- Liam Middlebrook [GitHub]liam-middlebrook

**Other Extension Metadata**

**Last Modified Date**

2020-04-16

**IP Status**

No known IP claims.

**Contributors**

- Joshua Ashton, Valve
- Hans-Kristian Arntzen, Valve
- Philip Rebohle, Valve
- Liam Middlebrook, NVIDIA
- Jeff Bolz, NVIDIA
- Tobias Hector, AMD
- Faith Ekstrand, Intel
- Spencer Fricke, Samsung Electronics
- Graeme Leese, Broadcom
- Jesse Hall, Google
- Jan-Harald Fredriksen, ARM
- Tom Olson, ARM
- Stuart Smith, Imagination Technologies

- Donald Scorgie, Imagination Technologies
- Alex Walters, Imagination Technologies
- Peter Quayle, Imagination Technologies

### Description

This extension provides cross-vendor functionality to specify a custom border color for use when the sampler address mode `VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_BORDER` is used.

To create a sampler which uses a custom border color set VkSamplerCreateInfo::`borderColor` to one of:

- `VK_BORDER_COLOR_FLOAT_CUSTOM_EXT`
- `VK_BORDER_COLOR_INT_CUSTOM_EXT`

When `VK_BORDER_COLOR_FLOAT_CUSTOM_EXT` or `VK_BORDER_COLOR_INT_CUSTOM_EXT` is used, applications must provide a VkSamplerCustomBorderColorCreateInfoEXT in the `pNext` chain for VkSamplerCreateInfo.

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceCustomBorderColorFeaturesEXT
- Extending VkPhysicalDeviceProperties2:
  - VkPhysicalDeviceCustomBorderColorPropertiesEXT
- Extending VkSamplerCreateInfo:
  - VkSamplerCustomBorderColorCreateInfoEXT

### New Enum Constants

- `VK_EXT_CUSTOM_BORDER_COLOR_EXTENSION_NAME`
- `VK_EXT_CUSTOM_BORDER_COLOR_SPEC_VERSION`
- Extending VkBorderColor:
  - `VK_BORDER_COLOR_FLOAT_CUSTOM_EXT`
  - `VK_BORDER_COLOR_INT_CUSTOM_EXT`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_CUSTOM_BORDER_COLOR_FEATURES_EXT`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_CUSTOM_BORDER_COLOR_PROPERTIES_EXT`
  - `VK_STRUCTURE_TYPE_SAMPLER_CUSTOM_BORDER_COLOR_CREATE_INFO_EXT`

### Issues

1) Should VkClearColorValue be used for the border color value, or should we have our own struct/union? Do we need to specify the type of the input values for the components? This is more

of a concern if VkClearColorValue is used here because it provides a union of float,int,uint types.

**RESOLVED**: Will reuse existing VkClearColorValue structure in order to easily take advantage of float,int,uint borderColor types.

2) For hardware which supports a limited number of border colors what happens if that number is exceeded? Should this be handled by the driver unbeknownst to the application? In Revision 1 we had solved this issue using a new Object type, however that may have lead to additional system resource consumption which would otherwise not be required.

**RESOLVED**: Added `VkPhysicalDeviceCustomBorderColorPropertiesEXT::maxCustomBorderColorSamplers` for tracking implementation-specific limit, and Valid Usage statement handling overflow.

3) Should this be supported for immutable samplers at all, or by a feature bit? Some implementations may not be able to support custom border colors on immutable samplers — is it worthwhile enabling this to work on them for implementations that can support it, or forbidding it entirely.

**RESOLVED**: Samplers created with a custom border color are forbidden from being immutable. This resolves concerns for implementations where the custom border color is an index to a LUT instead of being directly embedded into sampler state.

4) Should UINT and SINT (unsigned integer and signed integer) border color types be separated or should they be combined into one generic INT (integer) type?

**RESOLVED**: Separating these does not make much sense as the existing fixed border color types do not have this distinction, and there is no reason in hardware to do so. This separation would also create unnecessary work and considerations for the application.

**Version History**

- Revision 1, 2019-10-10 (Joshua Ashton)
  - Internal revisions.
- Revision 2, 2019-10-11 (Liam Middlebrook)
  - Remove VkCustomBorderColor object and associated functions
  - Add issues concerning HW limitations for custom border color count
- Revision 3, 2019-10-12 (Joshua Ashton)
  - Re-expose the limits for the maximum number of unique border colors
  - Add extra details about border color tracking
  - Fix typos
- Revision 4, 2019-10-12 (Joshua Ashton)
  - Changed maxUniqueCustomBorderColors to a uint32_t from a VkDeviceSize
- Revision 5, 2019-10-14 (Liam Middlebrook)
  - Added features bit
- Revision 6, 2019-10-15 (Joshua Ashton)

    - Type-ize VK_BORDER_COLOR_CUSTOM
    - Fix const-ness on `pNext` of VkSamplerCustomBorderColorCreateInfoEXT
- Revision 7, 2019-11-26 (Liam Middlebrook)
    - Renamed maxUniqueCustomBorderColors to maxCustomBorderColors
- Revision 8, 2019-11-29 (Joshua Ashton)
    - Renamed borderColor member of VkSamplerCustomBorderColorCreateInfoEXT to customBorderColor
- Revision 9, 2020-02-19 (Joshua Ashton)
    - Renamed maxCustomBorderColors to maxCustomBorderColorSamplers
- Revision 10, 2020-02-21 (Joshua Ashton)
    - Added format to VkSamplerCustomBorderColorCreateInfoEXT and feature bit
- Revision 11, 2020-04-07 (Joshua Ashton)
    - Dropped UINT/SINT border color differences, consolidated types
- Revision 12, 2020-04-16 (Joshua Ashton)
    - Renamed VK_BORDER_COLOR_CUSTOM_FLOAT_EXT to VK_BORDER_COLOR_FLOAT_CUSTOM_EXT for consistency

## VK_EXT_debug_utils

### Name String

`VK_EXT_debug_utils`

### Extension Type

Instance extension

### Registered Extension Number

129

### Revision

2

### Ratification Status

Not ratified

### Extension and Version Dependencies

None

### Special Use

- Debugging tools

### Contact

- Mark Young [GitHub]marky-lunarg

### Other Extension Metadata

#### Last Modified Date

2020-04-03

#### Revision

2

#### IP Status

No known IP claims.

#### Dependencies

- This extension is written against version 1.0 of the Vulkan API.
- Requires VkObjectType

#### Contributors

- Mark Young, LunarG
- Baldur Karlsson
- Ian Elliott, Google
- Courtney Goeltzenleuchter, Google
- Karl Schultz, LunarG
- Mark Lobodzinski, LunarG
- Mike Schuchardt, LunarG
- Jaakko Konttinen, AMD
- Dan Ginsburg, Valve Software
- Rolando Olivares, Epic Games
- Dan Baker, Oxide Games
- Kyle Spagnoli, NVIDIA
- Jon Ashburn, LunarG
- Piers Daniell, NVIDIA

### Description

Due to the nature of the Vulkan interface, there is very little error information available to the developer and application. By using the `VK_EXT_debug_utils` extension, developers **can** obtain more information. When combined with validation layers, even more detailed feedback on the application's use of Vulkan will be provided.

This extension provides the following capabilities:

- The ability to create a debug messenger which will pass along debug messages to an application supplied callback.
- The ability to identify specific Vulkan objects using a name or tag to improve tracking.

- The ability to identify specific sections within a `VkQueue` or `VkCommandBuffer` using labels to aid organization and offline analysis in external tools.

The main difference between this extension and `VK_EXT_debug_report` and `VK_EXT_debug_marker` is that those extensions use VkDebugReportObjectTypeEXT to identify objects. This extension uses the core VkObjectType in place of VkDebugReportObjectTypeEXT. The primary reason for this move is that no future object type handle enumeration values will be added to VkDebugReportObjectTypeEXT since the creation of VkObjectType.

In addition, this extension combines the functionality of both `VK_EXT_debug_report` and `VK_EXT_debug_marker` by allowing object name and debug markers (now called labels) to be returned to the application's callback function. This should assist in clarifying the details of a debug message including: what objects are involved and potentially which location within a VkQueue or VkCommandBuffer the message occurred.

#### New Object Types

- VkDebugUtilsMessengerEXT

#### New Commands

- vkCmdBeginDebugUtilsLabelEXT
- vkCmdEndDebugUtilsLabelEXT
- vkCmdInsertDebugUtilsLabelEXT
- vkCreateDebugUtilsMessengerEXT
- vkDestroyDebugUtilsMessengerEXT
- vkQueueBeginDebugUtilsLabelEXT
- vkQueueEndDebugUtilsLabelEXT
- vkQueueInsertDebugUtilsLabelEXT
- vkSetDebugUtilsObjectNameEXT
- vkSetDebugUtilsObjectTagEXT
- vkSubmitDebugUtilsMessageEXT

#### New Structures

- VkDebugUtilsLabelEXT
- VkDebugUtilsMessengerCallbackDataEXT
- VkDebugUtilsObjectTagInfoEXT
- Extending VkInstanceCreateInfo:
  - VkDebugUtilsMessengerCreateInfoEXT
- Extending VkPipelineShaderStageCreateInfo:
  - VkDebugUtilsObjectNameInfoEXT

### New Function Pointers

- PFN_vkDebugUtilsMessengerCallbackEXT

### New Enums

- VkDebugUtilsMessageSeverityFlagBitsEXT
- VkDebugUtilsMessageTypeFlagBitsEXT

### New Bitmasks

- VkDebugUtilsMessageSeverityFlagsEXT
- VkDebugUtilsMessageTypeFlagsEXT
- VkDebugUtilsMessengerCallbackDataFlagsEXT
- VkDebugUtilsMessengerCreateFlagsEXT

### New Enum Constants

- `VK_EXT_DEBUG_UTILS_EXTENSION_NAME`
- `VK_EXT_DEBUG_UTILS_SPEC_VERSION`
- Extending VkObjectType:
  - `VK_OBJECT_TYPE_DEBUG_UTILS_MESSENGER_EXT`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_DEBUG_UTILS_LABEL_EXT`
  - `VK_STRUCTURE_TYPE_DEBUG_UTILS_MESSENGER_CALLBACK_DATA_EXT`
  - `VK_STRUCTURE_TYPE_DEBUG_UTILS_MESSENGER_CREATE_INFO_EXT`
  - `VK_STRUCTURE_TYPE_DEBUG_UTILS_OBJECT_NAME_INFO_EXT`
  - `VK_STRUCTURE_TYPE_DEBUG_UTILS_OBJECT_TAG_INFO_EXT`

## Examples

### Example 1

`VK_EXT_debug_utils` allows an application to register multiple callbacks with any Vulkan component wishing to report debug information. Some callbacks may log the information to a file, others may cause a debug break point or other application-defined behavior. An application **can** register callbacks even when no validation layers are enabled, but they will only be called for loader and, if implemented, driver events.

To capture events that occur while creating or destroying an instance an application **can** link a VkDebugUtilsMessengerCreateInfoEXT structure to the `pNext` element of the VkInstanceCreateInfo structure given to vkCreateInstance.

Example uses: Create three callback objects. One will log errors and warnings to the debug console using Windows `OutputDebugString`. The second will cause the debugger to break at that callback

when an error happens and the third will log warnings to stdout.

```
    extern VkInstance instance;
    VkResult res;
    VkDebugUtilsMessengerEXT cb1, cb2, cb3;

    // Must call extension functions through a function pointer:
    PFN_vkCreateDebugUtilsMessengerEXT pfnCreateDebugUtilsMessengerEXT =
(PFN_vkCreateDebugUtilsMessengerEXT)vkGetInstanceProcAddr(instance,
"vkCreateDebugUtilsMessengerEXT");
    PFN_vkDestroyDebugUtilsMessengerEXT pfnDestroyDebugUtilsMessengerEXT =
(PFN_vkDestroyDebugUtilsMessengerEXT)vkGetInstanceProcAddr(instance,
"vkDestroyDebugUtilsMessengerEXT");

    VkDebugUtilsMessengerCreateInfoEXT callback1 = {
        .sType = VK_STRUCTURE_TYPE_DEBUG_UTILS_MESSENGER_CREATE_INFO_EXT,
        .pNext = NULL,
        .flags = 0,
        .messageSeverity = VK_DEBUG_UTILS_MESSAGE_SEVERITY_ERROR_BIT_EXT |
                           VK_DEBUG_UTILS_MESSAGE_SEVERITY_WARNING_BIT_EXT,
        .messageType= VK_DEBUG_UTILS_MESSAGE_TYPE_GENERAL_BIT_EXT |
                      VK_DEBUG_UTILS_MESSAGE_TYPE_VALIDATION_BIT_EXT,
        .pfnUserCallback = myOutputDebugString,
        .pUserData = NULL
    };
    res = pfnCreateDebugUtilsMessengerEXT(instance, &callback1, NULL, &cb1);
    if (res != VK_SUCCESS) {
       // Do error handling for VK_ERROR_OUT_OF_MEMORY
    }

    callback1.messageSeverity = VK_DEBUG_UTILS_MESSAGE_SEVERITY_ERROR_BIT_EXT;
    callback1.pfnUserCallback = myDebugBreak;
    callback1.pUserData = NULL;
    res = pfnCreateDebugUtilsMessengerEXT(instance, &callback1, NULL, &cb2);
    if (res != VK_SUCCESS) {
       // Do error handling for VK_ERROR_OUT_OF_MEMORY
    }

    VkDebugUtilsMessengerCreateInfoEXT callback3 = {
        .sType = VK_STRUCTURE_TYPE_DEBUG_UTILS_MESSENGER_CREATE_INFO_EXT,
        .pNext = NULL,
        .flags = 0,
        .messageSeverity = VK_DEBUG_UTILS_MESSAGE_SEVERITY_WARNING_BIT_EXT,
        .messageType = VK_DEBUG_UTILS_MESSAGE_TYPE_GENERAL_BIT_EXT |
                       VK_DEBUG_UTILS_MESSAGE_TYPE_VALIDATION_BIT_EXT,
        .pfnUserCallback = mystdOutLogger,
        .pUserData = NULL
    };
    res = pfnCreateDebugUtilsMessengerEXT(instance, &callback3, NULL, &cb3);
    if (res != VK_SUCCESS) {
```

```
        // Do error handling for VK_ERROR_OUT_OF_MEMORY
    }

    ...

    // Remove callbacks when cleaning up
    pfnDestroyDebugUtilsMessengerEXT(instance, cb1, NULL);
    pfnDestroyDebugUtilsMessengerEXT(instance, cb2, NULL);
    pfnDestroyDebugUtilsMessengerEXT(instance, cb3, NULL);
```

**Example 2**

Associate a name with an image, for easier debugging in external tools or with validation layers that can print a friendly name when referring to objects in error messages.

```
    extern VkInstance instance;
    extern VkDevice device;
    extern VkImage image;

    // Must call extension functions through a function pointer:
    PFN_vkSetDebugUtilsObjectNameEXT pfnSetDebugUtilsObjectNameEXT =
(PFN_vkSetDebugUtilsObjectNameEXT)vkGetInstanceProcAddr(instance,
"vkSetDebugUtilsObjectNameEXT");

    // Set a name on the image
    const VkDebugUtilsObjectNameInfoEXT imageNameInfo =
    {
        .sType = VK_STRUCTURE_TYPE_DEBUG_UTILS_OBJECT_NAME_INFO_EXT,
        .pNext = NULL,
        .objectType = VK_OBJECT_TYPE_IMAGE,
        .objectHandle = (uint64_t)image,
        .pObjectName = "Brick Diffuse Texture",
    };

    pfnSetDebugUtilsObjectNameEXT(device, &imageNameInfo);

    // A subsequent error might print:
    //   Image 'Brick Diffuse Texture' (0xc0dec0dedeadbeef) is used in a
    //   command buffer with no memory bound to it.
```

**Example 3**

Annotating regions of a workload with naming information so that offline analysis tools can display a more usable visualization of the commands submitted.

```
    extern VkInstance instance;
    extern VkCommandBuffer commandBuffer;

    // Must call extension functions through a function pointer:
```

```
    PFN_vkQueueBeginDebugUtilsLabelEXT pfnQueueBeginDebugUtilsLabelEXT =
(PFN_vkQueueBeginDebugUtilsLabelEXT)vkGetInstanceProcAddr(instance,
"vkQueueBeginDebugUtilsLabelEXT");
    PFN_vkQueueEndDebugUtilsLabelEXT pfnQueueEndDebugUtilsLabelEXT =
(PFN_vkQueueEndDebugUtilsLabelEXT)vkGetInstanceProcAddr(instance,
"vkQueueEndDebugUtilsLabelEXT");
    PFN_vkCmdBeginDebugUtilsLabelEXT pfnCmdBeginDebugUtilsLabelEXT =
(PFN_vkCmdBeginDebugUtilsLabelEXT)vkGetInstanceProcAddr(instance,
"vkCmdBeginDebugUtilsLabelEXT");
    PFN_vkCmdEndDebugUtilsLabelEXT pfnCmdEndDebugUtilsLabelEXT =
(PFN_vkCmdEndDebugUtilsLabelEXT)vkGetInstanceProcAddr(instance,
"vkCmdEndDebugUtilsLabelEXT");
    PFN_vkCmdInsertDebugUtilsLabelEXT pfnCmdInsertDebugUtilsLabelEXT =
(PFN_vkCmdInsertDebugUtilsLabelEXT)vkGetInstanceProcAddr(instance,
"vkCmdInsertDebugUtilsLabelEXT");

    // Describe the area being rendered
    const VkDebugUtilsLabelEXT houseLabel =
    {
        .sType = VK_STRUCTURE_TYPE_DEBUG_UTILS_LABEL_EXT,
        .pNext = NULL,
        .pLabelName = "Brick House",
        .color = { 1.0f, 0.0f, 0.0f, 1.0f },
    };

    // Start an annotated group of calls under the 'Brick House' name
    pfnCmdBeginDebugUtilsLabelEXT(commandBuffer, &houseLabel);
    {
        // A mutable structure for each part being rendered
        VkDebugUtilsLabelEXT housePartLabel =
        {
            .sType = VK_STRUCTURE_TYPE_DEBUG_UTILS_LABEL_EXT,
            .pNext = NULL,
            .pLabelName = NULL,
            .color = { 0.0f, 0.0f, 0.0f, 0.0f },
        };

        // Set the name and insert the marker
        housePartLabel.pLabelName = "Walls";
        pfnCmdInsertDebugUtilsLabelEXT(commandBuffer, &housePartLabel);

        // Insert the drawcall for the walls
        vkCmdDrawIndexed(commandBuffer, 1000, 1, 0, 0, 0);

        // Insert a recursive region for two sets of windows
        housePartLabel.pLabelName = "Windows";
        pfnCmdBeginDebugUtilsLabelEXT(commandBuffer, &housePartLabel);
        {
            vkCmdDrawIndexed(commandBuffer, 75, 6, 1000, 0, 0);
            vkCmdDrawIndexed(commandBuffer, 100, 2, 1450, 0, 0);
        }
```

```
        pfnCmdEndDebugUtilsLabelEXT(commandBuffer);

        housePartLabel.pLabelName = "Front Door";
        pfnCmdInsertDebugUtilsLabelEXT(commandBuffer, &housePartLabel);

        vkCmdDrawIndexed(commandBuffer, 350, 1, 1650, 0, 0);

        housePartLabel.pLabelName = "Roof";
        pfnCmdInsertDebugUtilsLabelEXT(commandBuffer, &housePartLabel);

        vkCmdDrawIndexed(commandBuffer, 500, 1, 2000, 0, 0);
    }
    // End the house annotation started above
    pfnCmdEndDebugUtilsLabelEXT(commandBuffer);

    // Do other work

    vkEndCommandBuffer(commandBuffer);

    // Describe the queue being used
    const VkDebugUtilsLabelEXT queueLabel =
    {
        .sType = VK_STRUCTURE_TYPE_DEBUG_UTILS_LABEL_EXT,
        .pNext = NULL,
        .pLabelName = "Main Render Work",
        .color = { 0.0f, 1.0f, 0.0f, 1.0f },
    };

    // Identify the queue label region
    pfnQueueBeginDebugUtilsLabelEXT(queue, &queueLabel);

    // Submit the work for the main render thread
    const VkCommandBuffer cmd_bufs[] = {commandBuffer};
    VkSubmitInfo submit_info =
    {
        .sType = VK_STRUCTURE_TYPE_SUBMIT_INFO,
        .pNext = NULL,
        .waitSemaphoreCount = 0,
        .pWaitSemaphores = NULL,
        .pWaitDstStageMask = NULL,
        .commandBufferCount = 1,
        .pCommandBuffers = cmd_bufs,
        .signalSemaphoreCount = 0,
        .pSignalSemaphores = NULL
    };
    vkQueueSubmit(queue, 1, &submit_info, fence);

    // End the queue label region
    pfnQueueEndDebugUtilsLabelEXT(queue);
```

### Issues

1) Should we just name this extension `VK_EXT_debug_report2`

**RESOLVED**: No. There is enough additional changes to the structures to break backwards compatibility. So, a new name was decided that would not indicate any interaction with the previous extension.

2) Will validation layers immediately support all the new features.

**RESOLVED**: Not immediately. As one can imagine, there is a lot of work involved with converting the validation layer logging over to the new functionality. Basic logging, as seen in the origin `VK_EXT_debug_report` extension will be made available immediately. However, adding the labels and object names will take time. Since the priority for Khronos at this time is to continue focusing on Valid Usage statements, it may take a while before the new functionality is fully exposed.

3) If the validation layers will not expose the new functionality immediately, then what is the point of this extension?

**RESOLVED**: We needed a replacement for `VK_EXT_debug_report` because the VkDebugReportObjectTypeEXT enumeration will no longer be updated and any new objects will need to be debugged using the new functionality provided by this extension.

4) Should this extension be split into two separate parts (1 extension that is an instance extension providing the callback functionality, and another device extension providing the general debug marker and annotation functionality)?

**RESOLVED**: No, the functionality for this extension is too closely related. If we did split up the extension, where would the structures and enums live, and how would you define that the device behavior in the instance extension is really only valid if the device extension is enabled, and the functionality is passed in. It is cleaner to just define this all as an instance extension, plus it allows the application to enable all debug functionality provided with one enable string during vkCreateInstance.

### Version History

- Revision 1, 2017-09-14 (Mark Young and all listed Contributors)
  - Initial draft, based on `VK_EXT_debug_report` and `VK_EXT_debug_marker` in addition to previous feedback supplied from various companies including Valve, Epic, and Oxide games.
- Revision 2, 2020-04-03 (Mark Young and Piers Daniell)
  - Updated to allow either `NULL` or an empty string to be passed in for `pObjectName` in `VkDebugUtilsObjectNameInfoEXT`, because the loader and various drivers support `NULL` already.

## VK_EXT_depth_bias_control

### Name String

`VK_EXT_depth_bias_control`

**Extension Type**

Device extension

**Registered Extension Number**

284

**Revision**

1

**Ratification Status**

Ratified

**Extension and Version Dependencies**

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

**Special Use**

- D3D support

**Contact**

- Joshua Ashton [GitHub]Joshua-Ashton

**Extension Proposal**

VK_EXT_depth_bias_control

**Other Extension Metadata**

**Last Modified Date**

2023-02-15

**IP Status**

No known IP claims.

**Contributors**

- Joshua Ashton, VALVE
- Hans-Kristian Arntzen, VALVE
- Mike Blumenkrantz, VALVE
- Georg Lehmann, VALVE
- Piers Daniell, NVIDIA
- Lionel Landwerlin, INTEL
- Tobias Hector, AMD
- Ricardo Garcia, IGALIA
- Jan-Harald Fredriksen, ARM
- Shahbaz Youssefi, GOOGLE

- Tom Olson, ARM

### Description

This extension adds a new structure, `VkDepthBiasRepresentationInfoEXT`, that can be added to a `pNext` chain of `VkPipelineRasterizationStateCreateInfo` and allows setting the scaling and representation of depth bias for a pipeline.

This state can also be set dynamically by using the new structure mentioned above in combination with the new `vkCmdSetDepthBias2EXT` command.

### New Commands

- vkCmdSetDepthBias2EXT

### New Structures

- VkDepthBiasInfoEXT
- Extending VkDepthBiasInfoEXT, VkPipelineRasterizationStateCreateInfo:
  - VkDepthBiasRepresentationInfoEXT
- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceDepthBiasControlFeaturesEXT

### New Enums

- VkDepthBiasRepresentationEXT

### New Enum Constants

- `VK_EXT_DEPTH_BIAS_CONTROL_EXTENSION_NAME`
- `VK_EXT_DEPTH_BIAS_CONTROL_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_DEPTH_BIAS_INFO_EXT`
  - `VK_STRUCTURE_TYPE_DEPTH_BIAS_REPRESENTATION_INFO_EXT`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DEPTH_BIAS_CONTROL_FEATURES_EXT`

### Version History

- Revision 1, 2022-09-22 (Joshua Ashton)
  - Initial draft.

## VK_EXT_depth_clamp_zero_one

### Name String

`VK_EXT_depth_clamp_zero_one`

### Extension Type

Device extension

### Registered Extension Number

422

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### Contact

- Graeme Leese [GitHub]gnl21

### Other Extension Metadata

### Last Modified Date

2021-07-29

### Contributors

- Graeme Leese, Broadcom

### Description

This extension gives defined behavior to fragment depth values which end up outside the conventional [0, 1] range. It can be used to ensure portability in edge cases of features like depthBias. The particular behavior is chosen to match OpenGL to aid porting or emulation.

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceDepthClampZeroOneFeaturesEXT

### New Enum Constants

- `VK_EXT_DEPTH_CLAMP_ZERO_ONE_EXTENSION_NAME`
- `VK_EXT_DEPTH_CLAMP_ZERO_ONE_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DEPTH_CLAMP_ZERO_ONE_FEATURES_EXT`

### Version History

- Revision 1, 2021-07-29 (Graeme Leese)
  - Internal revisions

## VK_EXT_depth_clip_control

### Name String

`VK_EXT_depth_clip_control`

### Extension Type

Device extension

### Registered Extension Number

356

### Revision

1

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### Special Use

- OpenGL / ES support

### Contact

- Shahbaz Youssefi [GitHub]syoussefi

### Other Extension Metadata

### Last Modified Date

2021-11-09

### Contributors

- Spencer Fricke, Samsung Electronics
- Shahbaz Youssefi, Google
- Ralph Potter, Samsung Electronics

### Description

This extension allows the application to use the OpenGL depth range in NDC, i.e. with depth in range [-1, 1], as opposed to Vulkan's default of [0, 1]. The purpose of this extension is to allow efficient layering of OpenGL over Vulkan, by avoiding emulation in the pre-rasterization shader

stages. This emulation, which effectively duplicates gl_Position but with a different depth value, costs ALU and consumes shader output components that the implementation may not have to spare to meet OpenGL minimum requirements.

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceDepthClipControlFeaturesEXT
- Extending VkPipelineViewportStateCreateInfo:
  - VkPipelineViewportDepthClipControlCreateInfoEXT

### New Enum Constants

- `VK_EXT_DEPTH_CLIP_CONTROL_EXTENSION_NAME`
- `VK_EXT_DEPTH_CLIP_CONTROL_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DEPTH_CLIP_CONTROL_FEATURES_EXT`
  - `VK_STRUCTURE_TYPE_PIPELINE_VIEWPORT_DEPTH_CLIP_CONTROL_CREATE_INFO_EXT`

### Issues

1) Should this extension include an origin control option to match GL_LOWER_LEFT found in ARB_clip_control?

**RESOLVED**: No. The fix for porting over the origin is a simple y-axis flip. The depth clip control is a much harder problem to solve than what this extension is aimed to solve. Adding an equivalent to GL_LOWER_LEFT would require more testing.

2) Should this pipeline state be dynamic?

**RESOLVED**: Yes. The purpose of this extension is to emulate the OpenGL depth range, which is expected to be globally fixed (in case of OpenGL ES) or very infrequently changed (with `glClipControl` in OpenGL).

3) Should the control provided in this extension be an enum that could be extended in the future?

**RESOLVED**: No. It is highly unlikely that the depth range is changed to anything other than [0, 1] in the future. Should that happen a new extension will be required to extend such an enum, and that extension might as well add a new struct to chain to VkPipelineViewportStateCreateInfo::`pNext` instead.

### Version History

- Revision 0, 2020-10-01 (Spencer Fricke)
  - Internal revisions
- Revision 1, 2020-11-26 (Shahbaz Youssefi)

- Language fixes

## VK_EXT_depth_clip_enable

### Name String

`VK_EXT_depth_clip_enable`

### Extension Type

Device extension

### Registered Extension Number

103

### Revision

1

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### Special Use

- D3D support

### Contact

- Piers Daniell [GitHub]pdaniell-nv

### Other Extension Metadata

### Last Modified Date

2018-12-20

### Contributors

- Daniel Rakos, AMD
- Henri Verbeet, CodeWeavers
- Jeff Bolz, NVIDIA
- Philip Rebohle, DXVK
- Tobias Hector, AMD

### Description

This extension allows the depth clipping operation, that is normally implicitly controlled by VkPipelineRasterizationStateCreateInfo::`depthClampEnable`, to instead be controlled explicitly by VkPipelineRasterizationDepthClipStateCreateInfoEXT::`depthClipEnable`.

This is useful for translating DX content which assumes depth clamping is always enabled, but depth clip can be controlled by the DepthClipEnable rasterization state (D3D12_RASTERIZER_DESC).

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceDepthClipEnableFeaturesEXT
- Extending VkPipelineRasterizationStateCreateInfo:
  - VkPipelineRasterizationDepthClipStateCreateInfoEXT

### New Bitmasks

- VkPipelineRasterizationDepthClipStateCreateFlagsEXT

### New Enum Constants

- `VK_EXT_DEPTH_CLIP_ENABLE_EXTENSION_NAME`
- `VK_EXT_DEPTH_CLIP_ENABLE_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DEPTH_CLIP_ENABLE_FEATURES_EXT`
  - `VK_STRUCTURE_TYPE_PIPELINE_RASTERIZATION_DEPTH_CLIP_STATE_CREATE_INFO_EXT`

### Version History

- Revision 1, 2018-12-20 (Piers Daniell)
  - Internal revisions

## VK_EXT_depth_range_unrestricted

### Name String

`VK_EXT_depth_range_unrestricted`

### Extension Type

Device extension

### Registered Extension Number

14

### Revision

1

### Ratification Status

Ratified

### Extension and Version Dependencies

None

### Contact

- Piers Daniell [GitHub]pdaniell-nv

### Other Extension Metadata

### Last Modified Date

2017-06-22

### Contributors

- Daniel Koch, NVIDIA
- Jeff Bolz, NVIDIA

### Description

This extension removes the VkViewport `minDepth` and `maxDepth` restrictions that the values must be between `0.0` and `1.0`, inclusive. It also removes the same restriction on VkPipelineDepthStencilStateCreateInfo `minDepthBounds` and `maxDepthBounds`. Finally it removes the restriction on the `depth` value in VkClearDepthStencilValue.

### New Enum Constants

- `VK_EXT_DEPTH_RANGE_UNRESTRICTED_EXTENSION_NAME`
- `VK_EXT_DEPTH_RANGE_UNRESTRICTED_SPEC_VERSION`

### Issues

1) How do VkViewport `minDepth` and `maxDepth` values outside of the `0.0` to `1.0` range interact with Primitive Clipping?

**RESOLVED**: The behavior described in Primitive Clipping still applies. If depth clamping is disabled the depth values are still clipped to $0 \leq z_c \leq w_c$ before the viewport transform. If depth clamping is enabled the above equation is ignored and the depth values are instead clamped to the VkViewport `minDepth` and `maxDepth` values, which in the case of this extension can be outside of the `0.0` to `1.0` range.

2) What happens if a resulting depth fragment is outside of the `0.0` to `1.0` range and the depth buffer is fixed-point rather than floating-point?

**RESOLVED**: This situation can also arise without this extension (when fragment shaders replace depth values, for example), and this extension does not change the behavior, which is defined in the Depth Test section of the Fragment Operations chapter.

### Version History

- Revision 1, 2017-06-22 (Piers Daniell)
  - Internal revisions

# VK_EXT_descriptor_buffer

## Name String

`VK_EXT_descriptor_buffer`

## Extension Type

Device extension

## Registered Extension Number

317

## Revision

1

## Ratification Status

Not ratified

## Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1
and
VK_KHR_buffer_device_address
and
VK_EXT_descriptor_indexing
or
Vulkan Version 1.2
and
VK_KHR_synchronization2
or
Vulkan Version 1.3

## API Interactions

- Interacts with VK_KHR_acceleration_structure
- Interacts with VK_NV_ray_tracing

## Contact

- Tobias Hector [GitHub]tobski

## Extension Proposal

VK_EXT_descriptor_buffer

## Other Extension Metadata

## Last Modified Date

2021-06-07

### IP Status

No known IP claims.

### Contributors

- Tobias Hector, AMD
- Stu Smith, AMD
- Maciej Jesionowski, AMD
- Boris Zanin, AMD
- Hans-Kristian Arntzen, Valve
- Connor Abbott, Valve
- Baldur Karlsson, Valve
- Mike Blumenkrantz, Valve
- Graeme Leese, Broadcom
- Jan-Harald Fredriksen, Arm
- Rodrigo Locatti, NVIDIA
- Jeff Bolz, NVIDIA
- Piers Daniell, NVIDIA
- Jeff Leger, QUALCOMM
- Lionel Landwerlin, Intel
- Slawomir Grajewski, Intel

### Description

This extension introduces new commands to put shader-accessible descriptors directly in memory, making the management of descriptor data more explicit.

### New Commands

- vkCmdBindDescriptorBufferEmbeddedSamplersEXT
- vkCmdBindDescriptorBuffersEXT
- vkCmdSetDescriptorBufferOffsetsEXT
- vkGetBufferOpaqueCaptureDescriptorDataEXT
- vkGetDescriptorEXT
- vkGetDescriptorSetLayoutBindingOffsetEXT
- vkGetDescriptorSetLayoutSizeEXT
- vkGetImageOpaqueCaptureDescriptorDataEXT
- vkGetImageViewOpaqueCaptureDescriptorDataEXT
- vkGetSamplerOpaqueCaptureDescriptorDataEXT

If VK_KHR_acceleration_structure or VK_NV_ray_tracing is supported:

- vkGetAccelerationStructureOpaqueCaptureDescriptorDataEXT

### New Structures

- VkBufferCaptureDescriptorDataInfoEXT
- VkDescriptorAddressInfoEXT
- VkDescriptorBufferBindingInfoEXT
- VkDescriptorGetInfoEXT
- VkImageCaptureDescriptorDataInfoEXT
- VkImageViewCaptureDescriptorDataInfoEXT
- VkSamplerCaptureDescriptorDataInfoEXT
- Extending VkBufferCreateInfo, VkImageCreateInfo, VkImageViewCreateInfo, VkSamplerCreateInfo, VkAccelerationStructureCreateInfoKHR, VkAccelerationStructureCreateInfoNV:
  - VkOpaqueCaptureDescriptorDataCreateInfoEXT
- Extending VkDescriptorBufferBindingInfoEXT:
  - VkDescriptorBufferBindingPushDescriptorBufferHandleEXT
- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceDescriptorBufferFeaturesEXT
- Extending VkPhysicalDeviceProperties2:
  - VkPhysicalDeviceDescriptorBufferDensityMapPropertiesEXT
  - VkPhysicalDeviceDescriptorBufferPropertiesEXT

If VK_KHR_acceleration_structure or VK_NV_ray_tracing is supported:

- VkAccelerationStructureCaptureDescriptorDataInfoEXT

### New Unions

- VkDescriptorDataEXT

### New Enum Constants

- `VK_EXT_DESCRIPTOR_BUFFER_EXTENSION_NAME`
- `VK_EXT_DESCRIPTOR_BUFFER_SPEC_VERSION`
- Extending VkAccelerationStructureCreateFlagBitsKHR:
  - `VK_ACCELERATION_STRUCTURE_CREATE_DESCRIPTOR_BUFFER_CAPTURE_REPLAY_BIT_EXT`
- Extending VkAccessFlagBits2:
  - `VK_ACCESS_2_DESCRIPTOR_BUFFER_READ_BIT_EXT`

- Extending VkBufferCreateFlagBits:
  - VK_BUFFER_CREATE_DESCRIPTOR_BUFFER_CAPTURE_REPLAY_BIT_EXT
- Extending VkBufferUsageFlagBits:
  - VK_BUFFER_USAGE_PUSH_DESCRIPTORS_DESCRIPTOR_BUFFER_BIT_EXT
  - VK_BUFFER_USAGE_RESOURCE_DESCRIPTOR_BUFFER_BIT_EXT
  - VK_BUFFER_USAGE_SAMPLER_DESCRIPTOR_BUFFER_BIT_EXT
- Extending VkDescriptorSetLayoutCreateFlagBits:
  - VK_DESCRIPTOR_SET_LAYOUT_CREATE_DESCRIPTOR_BUFFER_BIT_EXT
  - VK_DESCRIPTOR_SET_LAYOUT_CREATE_EMBEDDED_IMMUTABLE_SAMPLERS_BIT_EXT
- Extending VkImageCreateFlagBits:
  - VK_IMAGE_CREATE_DESCRIPTOR_BUFFER_CAPTURE_REPLAY_BIT_EXT
- Extending VkImageViewCreateFlagBits:
  - VK_IMAGE_VIEW_CREATE_DESCRIPTOR_BUFFER_CAPTURE_REPLAY_BIT_EXT
- Extending VkPipelineCreateFlagBits:
  - VK_PIPELINE_CREATE_DESCRIPTOR_BUFFER_BIT_EXT
- Extending VkSamplerCreateFlagBits:
  - VK_SAMPLER_CREATE_DESCRIPTOR_BUFFER_CAPTURE_REPLAY_BIT_EXT
- Extending VkStructureType:
  - VK_STRUCTURE_TYPE_BUFFER_CAPTURE_DESCRIPTOR_DATA_INFO_EXT
  - VK_STRUCTURE_TYPE_DESCRIPTOR_ADDRESS_INFO_EXT
  - VK_STRUCTURE_TYPE_DESCRIPTOR_BUFFER_BINDING_INFO_EXT
  - VK_STRUCTURE_TYPE_DESCRIPTOR_BUFFER_BINDING_PUSH_DESCRIPTOR_BUFFER_HANDLE_EXT
  - VK_STRUCTURE_TYPE_DESCRIPTOR_GET_INFO_EXT
  - VK_STRUCTURE_TYPE_IMAGE_CAPTURE_DESCRIPTOR_DATA_INFO_EXT
  - VK_STRUCTURE_TYPE_IMAGE_VIEW_CAPTURE_DESCRIPTOR_DATA_INFO_EXT
  - VK_STRUCTURE_TYPE_OPAQUE_CAPTURE_DESCRIPTOR_DATA_CREATE_INFO_EXT
  - VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DESCRIPTOR_BUFFER_DENSITY_MAP_PROPERTIES_EXT
  - VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DESCRIPTOR_BUFFER_FEATURES_EXT
  - VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DESCRIPTOR_BUFFER_PROPERTIES_EXT
  - VK_STRUCTURE_TYPE_SAMPLER_CAPTURE_DESCRIPTOR_DATA_INFO_EXT

If VK_KHR_acceleration_structure or VK_NV_ray_tracing is supported:

- Extending VkStructureType:
  - VK_STRUCTURE_TYPE_ACCELERATION_STRUCTURE_CAPTURE_DESCRIPTOR_DATA_INFO_EXT

### Version History

- Revision 1, 2021-06-07 (Stu Smith)
  - Initial revision

## VK_EXT_descriptor_indexing

### Name String

`VK_EXT_descriptor_indexing`

### Extension Type

Device extension

### Registered Extension Number

162

### Revision

2

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
and
VK_KHR_maintenance3
or
Vulkan Version 1.1

### SPIR-V Dependencies

- SPV_EXT_descriptor_indexing

### Deprecation State

- *Promoted* to Vulkan 1.2

### Contact

- Jeff Bolz [GitHub]jeffbolznv

### Other Extension Metadata

### Last Modified Date

2017-10-02

### Interactions and External Dependencies

- This extension provides API support for `GL_EXT_nonuniform_qualifier`

### Contributors

- Jeff Bolz, NVIDIA
- Daniel Rakos, AMD

- Slawomir Grajewski, Intel
- Tobias Hector, Imagination Technologies

### Description

This extension adds several small features which together enable applications to create large descriptor sets containing substantially all of their resources, and selecting amongst those resources with dynamic (non-uniform) indexes in the shader. There are feature enables and SPIR-V capabilities for non-uniform descriptor indexing in the shader, and non-uniform indexing in the shader requires use of a new `NonUniformEXT` decoration defined in the `SPV_EXT_descriptor_indexing` SPIR-V extension. There are descriptor set layout binding creation flags enabling several features:

- Descriptors can be updated after they are bound to a command buffer, such that the execution of the command buffer reflects the most recent update to the descriptors.
- Descriptors that are not used by any pending command buffers can be updated, which enables writing new descriptors for frame N+1 while frame N is executing.
- Relax the requirement that all descriptors in a binding that is "statically used" must be valid, such that descriptors that are not accessed by a submission need not be valid and can be updated while that submission is executing.
- The final binding in a descriptor set layout can have a variable size (and unsized arrays of resources are allowed in the `GL_EXT_nonuniform_qualifier` and `SPV_EXT_descriptor_indexing` extensions).

Note that it is valid for multiple descriptor arrays in a shader to use the same set and binding number, as long as they are all compatible with the descriptor type in the pipeline layout. This means a single array binding in the descriptor set can serve multiple texture dimensionalities, or an array of buffer descriptors can be used with multiple different block layouts.

There are new descriptor set layout and descriptor pool creation flags that are required to opt in to the update-after-bind functionality, and there are separate `maxPerStage*` and `maxDescriptorSet*` limits that apply to these descriptor set layouts which **may** be much higher than the pre-existing limits. The old limits only count descriptors in non-updateAfterBind descriptor set layouts, and the new limits count descriptors in all descriptor set layouts in the pipeline layout.

### New Structures

- Extending VkDescriptorSetAllocateInfo:
  - VkDescriptorSetVariableDescriptorCountAllocateInfoEXT
- Extending VkDescriptorSetLayoutCreateInfo:
  - VkDescriptorSetLayoutBindingFlagsCreateInfoEXT
- Extending VkDescriptorSetLayoutSupport:
  - VkDescriptorSetVariableDescriptorCountLayoutSupportEXT
- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceDescriptorIndexingFeaturesEXT

- Extending VkPhysicalDeviceProperties2:
  - VkPhysicalDeviceDescriptorIndexingPropertiesEXT

### New Enums

- VkDescriptorBindingFlagBitsEXT

### New Bitmasks

- VkDescriptorBindingFlagsEXT

### New Enum Constants

- `VK_EXT_DESCRIPTOR_INDEXING_EXTENSION_NAME`
- `VK_EXT_DESCRIPTOR_INDEXING_SPEC_VERSION`
- Extending VkDescriptorBindingFlagBits:
  - `VK_DESCRIPTOR_BINDING_PARTIALLY_BOUND_BIT_EXT`
  - `VK_DESCRIPTOR_BINDING_UPDATE_AFTER_BIND_BIT_EXT`
  - `VK_DESCRIPTOR_BINDING_UPDATE_UNUSED_WHILE_PENDING_BIT_EXT`
  - `VK_DESCRIPTOR_BINDING_VARIABLE_DESCRIPTOR_COUNT_BIT_EXT`
- Extending VkDescriptorPoolCreateFlagBits:
  - `VK_DESCRIPTOR_POOL_CREATE_UPDATE_AFTER_BIND_BIT_EXT`
- Extending VkDescriptorSetLayoutCreateFlagBits:
  - `VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT_EXT`
- Extending VkResult:
  - `VK_ERROR_FRAGMENTATION_EXT`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_DESCRIPTOR_SET_LAYOUT_BINDING_FLAGS_CREATE_INFO_EXT`
  - `VK_STRUCTURE_TYPE_DESCRIPTOR_SET_VARIABLE_DESCRIPTOR_COUNT_ALLOCATE_INFO_EXT`
  - `VK_STRUCTURE_TYPE_DESCRIPTOR_SET_VARIABLE_DESCRIPTOR_COUNT_LAYOUT_SUPPORT_EXT`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DESCRIPTOR_INDEXING_FEATURES_EXT`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DESCRIPTOR_INDEXING_PROPERTIES_EXT`

### Promotion to Vulkan 1.2

Functionality in this extension is included in core Vulkan 1.2, with the EXT suffix omitted. However, if Vulkan 1.2 is supported and this extension is not, the `descriptorIndexing` capability is optional. The original type, enum and command names are still available as aliases of the core functionality.

### Version History

- Revision 1, 2017-07-26 (Jeff Bolz)

    - Internal revisions
- Revision 2, 2017-10-02 (Jeff Bolz)
    - ???

## VK_EXT_device_address_binding_report

### Name String

`VK_EXT_device_address_binding_report`

### Extension Type

Device extension

### Registered Extension Number

355

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1
and
VK_EXT_debug_utils

### Special Uses

- Debugging tools
- Developer tools

### Contact

- Ralph Potter [GitLab]r_potter

### Other Extension Metadata

### Last Modified Date

2020-11-23

### Interactions and External Dependencies

- This extension requires `VK_EXT_debug_utils`

### Contributors

- Ralph Potter, Samsung
- Spencer Fricke, Samsung
- Jan-Harald Fredriksen, ARM

- Andrew Ellem, Google
- Alex Walters, IMG
- Jeff Bolz, NVIDIA

### Description

This extension enables applications to track the binding of regions of the GPU virtual address space, and to associate those regions with Vulkan objects. This extension is primarily intended to aid in crash postmortem, where applications may wish to map a faulting GPU address to a Vulkan object.

For example, a page fault triggered by accessing an address located within a region of the GPU virtual address space that was previously reported as bound and then unbound may indicate a use-after-free error. Similarly, faults generated by accessing virtual addresses outside the limits of a bound region of GPU virtual address space may indicate indexing beyond the bounds of a resource.

### New Structures

- Extending VkDebugUtilsMessengerCallbackDataEXT:
  - VkDeviceAddressBindingCallbackDataEXT
- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceAddressBindingReportFeaturesEXT

### New Enums

- VkDeviceAddressBindingFlagBitsEXT
- VkDeviceAddressBindingTypeEXT

### New Bitmasks

- VkDeviceAddressBindingFlagsEXT

### New Enum Constants

- `VK_EXT_DEVICE_ADDRESS_BINDING_REPORT_EXTENSION_NAME`
- `VK_EXT_DEVICE_ADDRESS_BINDING_REPORT_SPEC_VERSION`
- Extending VkDebugUtilsMessageTypeFlagBitsEXT:
  - `VK_DEBUG_UTILS_MESSAGE_TYPE_DEVICE_ADDRESS_BINDING_BIT_EXT`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_DEVICE_ADDRESS_BINDING_CALLBACK_DATA_EXT`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_ADDRESS_BINDING_REPORT_FEATURES_EXT`

### Issues

1.) Should this extend VK_EXT_debug_utils or VK_EXT_device_memory_report?

**RESOLVED**: Extend VK_EXT_debug_utils. VK_EXT_device_memory_report focuses on memory

allocations and would not normally trigger callbacks in all of the situations where VK_EXT_device_address_binding_report is expected to.

2.) Should this extension cover all Vulkan object types, or only resources such as buffers and images?

**RESOLVED**: The extension covers all Vulkan objects, and is not restricted to objects backed by VkDeviceMemory objects.

3.) Should reallocation be identified explicitly, or as an unbind/bind pair?

**RESOLVED**: Reallocation should be represented as an unbind/bind pair.

4.) Can multiple Vulkan objects share an overlapping virtual address range?

**RESOLVED**: Yes. This can be expected to occur due to resources aliasing.

5.) Can a single Vulkan object be associated with multiple virtual address ranges concurrently?

**RESOLVED**: Yes. These should be reported via multiple calls to the reporting callback.

6.) Should the virtual address ranges associated with internal allocations such as memory pools be reported?

**RESOLVED**: Virtual address ranges associated with internal allocations should only be reported when they become associated with a specific Vulkan object. In the case of internal pool allocations, a bind event should be reported when resources from the pool are assigned to a Vulkan object, and an unbind event should be reported when those resources are returned to the pool. Implementations should not report the binding or unbinding of virtual address ranges for which there are no related API objects visible to the application developer.

7.) Can an implementation report binding a virtual address range at VkImage or VkImageView creation, rather than in response to vkBindImageMemory?

**RESOLVED**: Yes. Virtual address range binding should be reported at the appropriate point at which it occurs within the implementation. This extension does not mandate when that should occur, and applications should anticipate receiving callback events at any point after registering callbacks.

8.) Can reporting of binding/unbinding be deferred until a resource is referenced by an executing command buffer?

**RESOLVED**: Changes to the virtual address ranges associated with a Vulkan object should be reported as close as possible to where they occur within the implementation. If virtual address binding is deferred, then the callback should also be deferred to match.

9.) Do bind/unbind callbacks have to form matched pairs? Can a large region be bound, and then subregions unbound, resulting in fragmentation?

**RESOLVED**: Splitting of virtual address regions, and unmatched bind/unbind callbacks may occur. Developers should anticipate that sparse memory may exhibit this behavior.

10.) The specification mandates that a callback **must** be triggered whenever a GPU virtual address range associated with any Vulkan object is bound or unbound. Do we need queries or properties indicating which Vulkan objects will report binding modifications?

**RESOLVED**: No. This extension is not intended to mandate how and when implementations bind virtual ranges to objects. Adding queries or properties would constrain implementations, which might otherwise vary how virtual address bindings occur based on usage.

11.) Should vkAllocateMemory and vkFreeMemory trigger reporting callbacks?

**RESOLVED**: If an implementation binds a GPU virtual address range when vkAllocateMemory is called, then the callbacks must be triggered associating the virtual address range with the VkDeviceMemory object. If the device memory is subsequently bound to a buffer or image via vkBind*Memory, the callbacks should be triggered a second time, reporting the association between virtual address range and the buffer/image.

### Version History

- Revision 1, 2020-09-23 (Ralph Potter)
  - Internal revisions

## VK_EXT_device_fault

### Name String

`VK_EXT_device_fault`

### Extension Type

Device extension

### Registered Extension Number

342

### Revision

2

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### Contact

- Ralph Potter [GitLab]r_potter

### Extension Proposal

VK_EXT_device_fault

### Other Extension Metadata

#### Last Modified Date

2021-03-10

#### IP Status

No known IP claims.

#### Contributors

- Ralph Potter, Samsung
- Stuart Smith, AMD
- Jan-Harald Fredriksen, ARM
- Mark Bellamy, ARM
- Andrew Ellem, Google
- Alex Walters, IMG
- Jeff Bolz, NVIDIA
- Baldur Karlsson, Valve

#### Description

Device loss can be triggered by a variety of issues, including invalid API usage, implementation errors, or hardware failures.

This extension introduces a new command: vkGetDeviceFaultInfoEXT, which may be called subsequent to a `VK_ERROR_DEVICE_LOST` error code having been returned by the implementation. This command allows developers to query for additional information on GPU faults which may have caused device loss, and to generate binary crash dumps, which may be loaded into external tools for further diagnosis.

#### New Commands

- vkGetDeviceFaultInfoEXT

#### New Structures

- VkDeviceFaultAddressInfoEXT
- VkDeviceFaultCountsEXT
- VkDeviceFaultInfoEXT
- VkDeviceFaultVendorBinaryHeaderVersionOneEXT
- VkDeviceFaultVendorInfoEXT
- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceFaultFeaturesEXT

### New Enums

- VkDeviceFaultAddressTypeEXT
- VkDeviceFaultVendorBinaryHeaderVersionEXT

### New Enum Constants

- `VK_EXT_DEVICE_FAULT_EXTENSION_NAME`
- `VK_EXT_DEVICE_FAULT_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_DEVICE_FAULT_COUNTS_EXT`
  - `VK_STRUCTURE_TYPE_DEVICE_FAULT_INFO_EXT`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_FAULT_FEATURES_EXT`

### Version History

- Revision 2, 2023-04-05 (Ralph Potter)
  - Restored two missing members to the XML definition of VkDeviceFaultVendorBinaryHeaderVersionOneEXT. No functional change to the specification.
- Revision 1, 2020-10-19 (Ralph Potter)
  - Initial revision

## VK_EXT_device_memory_report

### Name String

`VK_EXT_device_memory_report`

### Extension Type

Device extension

### Registered Extension Number

285

### Revision

2

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

**Special Use**

- Developer tools

**Contact**

- Yiwei Zhang [GitHub]zhangyiwei

**Other Extension Metadata**

**Last Modified Date**

2021-01-06

**IP Status**

No known IP claims.

**Contributors**

- Yiwei Zhang, Google
- Jesse Hall, Google

**Description**

This device extension allows registration of device memory event callbacks upon device creation, so that applications or middleware can obtain detailed information about memory usage and how memory is associated with Vulkan objects. This extension exposes the actual underlying device memory usage, including allocations that are not normally visible to the application, such as memory consumed by vkCreateGraphicsPipelines. It is intended primarily for use by debug tooling rather than for production applications.

**New Structures**

- VkDeviceMemoryReportCallbackDataEXT
- Extending VkDeviceCreateInfo:
  - VkDeviceDeviceMemoryReportCreateInfoEXT
- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceDeviceMemoryReportFeaturesEXT

**New Function Pointers**

- PFN_vkDeviceMemoryReportCallbackEXT

**New Enums**

- VkDeviceMemoryReportEventTypeEXT

**New Bitmasks**

- VkDeviceMemoryReportFlagsEXT

### New Enum Constants

- `VK_EXT_DEVICE_MEMORY_REPORT_EXTENSION_NAME`
- `VK_EXT_DEVICE_MEMORY_REPORT_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_DEVICE_DEVICE_MEMORY_REPORT_CREATE_INFO_EXT`
  - `VK_STRUCTURE_TYPE_DEVICE_MEMORY_REPORT_CALLBACK_DATA_EXT`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DEVICE_MEMORY_REPORT_FEATURES_EXT`

### Issues

1) Should this be better expressed as an extension to VK_EXT_debug_utils and its general-purpose messenger construct?

**RESOLVED**: No. The intended lifecycle is quite different. We want to make this extension tied to the device's lifecycle. Each ICD just handles its own implementation of this extension, and this extension will only be directly exposed from the ICD. So we can avoid the extra implementation complexity used to accommodate the flexibility of `VK_EXT_debug_utils` extension.

2) Can we extend and use the existing internal allocation callbacks instead of adding the new callback structure in this extension?

**RESOLVED**: No. Our memory reporting layer that combines this information with other memory information it collects directly (e.g. bindings of resources to VkDeviceMemory) would have to intercept all entry points that take a VkAllocationCallbacks parameter and inject its own `pfnInternalAllocation` and `pfnInternalFree`. That may be doable for the extensions we know about, but not for ones we do not. The proposal would work fine in the face of most unknown extensions. But even for ones we know about, since apps can provide a different set of callbacks and userdata and those can be retained by the driver and used later (esp. for pool object, but not just those), we would have to dynamically allocate the interception trampoline every time. That is getting to be an unreasonably large amount of complexity and (possibly) overhead.

We are interested in both alloc/free and import/unimport. The latter is fairly important for tracking (and avoiding double-counting) of swapchain images (still true with "native swapchains" based on external memory) and media/camera interop. Though we might be able to handle this with additional VkInternalAllocationType values, for import/export we do want to be able to tie this to the external resource, which is one thing that the `memoryObjectId` is for.

The internal alloc/free callbacks are not extensible except via new VkInternalAllocationType values. The VkDeviceMemoryReportCallbackDataEXT in this extension is extensible. That was deliberate: there is a real possibility we will want to get extra information in the future. As one example, currently this reports only physical allocations, but we believe there are interesting cases for tracking how populated that VA region is.

The callbacks are clearly specified as only callable within the context of a call from the application into Vulkan. We believe there are some cases where drivers can allocate device memory asynchronously. This was one of the sticky issues that derailed the internal device memory allocation reporting design (which is essentially what this extension is trying to do) leading up to

1.0.

VkAllocationCallbacks is described in a section called “Host memory” and the intro to it is very explicitly about host memory. The other callbacks are all inherently about host memory. But this extension is very focused on device memory.

3) Should the callback be reporting which heap is used?

**RESOLVED**: Yes. It is important for non-UMA systems to have all the device memory allocations attributed to the corresponding device memory heaps. For internally-allocated device memory, `heapIndex` will always correspond to an advertised heap, rather than having a magic value indicating a non-advertised heap. Drivers can advertise heaps that do not have any corresponding memory types if they need to.

4) Should we use an array of callback for the layers to intercept instead of chaining multiple of the VkDeviceDeviceMemoryReportCreateInfoEXT structures in the `pNext` of VkDeviceCreateInfo?

**RESOLVED** No. The pointer to the VkDeviceDeviceMemoryReportCreateInfoEXT structure itself is const and you cannot just cast it away. Thus we cannot update the callback array inside the structure. In addition, we cannot drop this `pNext` chain either, so making a copy of this whole structure does not work either.

5) Should we track bulk allocations shared among multiple objects?

**RESOLVED** No. Take the shader heap as an example. Some implementations will let multiple VkPipeline objects share the same shader heap. We are not asking the implementation to report `VK_OBJECT_TYPE_PIPELINE` along with a VK_NULL_HANDLE for this bulk allocation. Instead, this bulk allocation is considered as a layer below what this extension is interested in. Later, when the actual VkPipeline objects are created by suballocating from the bulk allocation, we ask the implementation to report the valid handles of the VkPipeline objects along with the actual suballocated sizes and different `memoryObjectId`.

6) Can we require the callbacks to be always called in the same thread with the Vulkan commands?

**RESOLVED** No. Some implementations might choose to multiplex work from multiple application threads into a single backend thread and perform JIT allocations as a part of that flow. Since this behavior is theoretically legit, we cannot require the callbacks to be always called in the same thread with the Vulkan commands, and the note is to remind the applications to handle this case properly.

7) Should we add an additional “allocation failed” event type with things like size and heap index reported?

**RESOLVED** Yes. This fits in well with the callback infrastructure added in this extension, and implementation touches the same code and has the same overheads as the rest of the extension. It could help debugging things like getting a `VK_ERROR_OUT_OF_HOST_MEMORY` error when ending a command buffer. Right now the allocation failure could have happened anywhere during recording, and a callback would be really useful to understand where and why.

### Version History

- Revision 1, 2020-08-26 (Yiwei Zhang)
  - Initial version
- Revision 2, 2021-01-06 (Yiwei Zhang)
  - Minor description update

## VK_EXT_direct_mode_display

### Name String

`VK_EXT_direct_mode_display`

### Extension Type

Instance extension

### Registered Extension Number

89

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_display

### Contact

- James Jones [GitHub]cubanismo

### Other Extension Metadata

### Last Modified Date

2016-12-13

### IP Status

No known IP claims.

### Contributors

- Pierre Boudier, NVIDIA
- James Jones, NVIDIA
- Damien Leone, NVIDIA
- Pierre-Loup Griffais, Valve
- Liam Middlebrook, NVIDIA

### Description

This is extension, along with related platform extensions, allows applications to take exclusive control of displays associated with a native windowing system. This is especially useful for virtual reality applications that wish to hide HMDs (head mounted displays) from the native platform's display management system, desktop, and/or other applications.

### New Commands

- vkReleaseDisplayEXT

### New Enum Constants

- `VK_EXT_DIRECT_MODE_DISPLAY_EXTENSION_NAME`
- `VK_EXT_DIRECT_MODE_DISPLAY_SPEC_VERSION`

### Issues

1) Should this extension and its related platform-specific extensions leverage `VK_KHR_display`, or provide separate equivalent interfaces.

**RESOLVED**: Use `VK_KHR_display` concepts and objects. `VK_KHR_display` can be used to enumerate all displays on the system, including those attached to/in use by a window system or native platform, but `VK_KHR_display_swapchain` will fail to create a swapchain on in-use displays. This extension and its platform-specific children will allow applications to grab in-use displays away from window systems and/or native platforms, allowing them to be used with `VK_KHR_display_swapchain`.

2) Are separate calls needed to acquire displays and enable direct mode?

**RESOLVED**: No, these operations happen in one combined command. Acquiring a display puts it into direct mode.

### Version History

- Revision 1, 2016-12-13 (James Jones)
  - Initial draft

## VK_EXT_directfb_surface

### Name String

`VK_EXT_directfb_surface`

### Extension Type

Instance extension

### Registered Extension Number

347

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_surface

### Contact

- Nicolas Caramelli [GitHub]caramelli

### Other Extension Metadata

### Last Modified Date

2020-06-16

### IP Status

No known IP claims.

### Contributors

- Nicolas Caramelli

### Description

The `VK_EXT_directfb_surface` extension is an instance extension. It provides a mechanism to create a VkSurfaceKHR object (defined by the `VK_KHR_surface` extension) that refers to a DirectFB `IDirectFBSurface`, as well as a query to determine support for rendering via DirectFB.

### New Commands

- vkCreateDirectFBSurfaceEXT
- vkGetPhysicalDeviceDirectFBPresentationSupportEXT

### New Structures

- VkDirectFBSurfaceCreateInfoEXT

### New Bitmasks

- VkDirectFBSurfaceCreateFlagsEXT

### New Enum Constants

- `VK_EXT_DIRECTFB_SURFACE_EXTENSION_NAME`
- `VK_EXT_DIRECTFB_SURFACE_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_DIRECTFB_SURFACE_CREATE_INFO_EXT`

### Version History

- Revision 1, 2020-06-16 (Nicolas Caramelli)
  - Initial version

## VK_EXT_discard_rectangles

### Name String

`VK_EXT_discard_rectangles`

### Extension Type

Device extension

### Registered Extension Number

100

### Revision

2

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### Contact

- Piers Daniell [GitHub]pdaniell-nv

### Other Extension Metadata

### Last Modified Date

2023-01-18

### Interactions and External Dependencies

- Interacts with `VK_KHR_device_group`
- Interacts with Vulkan 1.1

### Contributors

- Daniel Koch, NVIDIA
- Jeff Bolz, NVIDIA

### Description

This extension provides additional orthogonally aligned “discard rectangles” specified in framebuffer-space coordinates that restrict rasterization of all points, lines and triangles.

From zero to an implementation-dependent limit (specified by `maxDiscardRectangles`) number of discard rectangles can be operational at once. When one or more discard rectangles are active, rasterized fragments can either survive if the fragment is within any of the operational discard rectangles (`VK_DISCARD_RECTANGLE_MODE_INCLUSIVE_EXT` mode) or be rejected if the fragment is within any of the operational discard rectangles (`VK_DISCARD_RECTANGLE_MODE_EXCLUSIVE_EXT` mode).

These discard rectangles operate orthogonally to the existing scissor test functionality. The discard rectangles can be different for each physical device in a device group by specifying the device mask and setting discard rectangle dynamic state.

Version 2 of this extension introduces new dynamic states `VK_DYNAMIC_STATE_DISCARD_RECTANGLE_ENABLE_EXT` and `VK_DYNAMIC_STATE_DISCARD_RECTANGLE_MODE_EXT`, and the corresponding functions vkCmdSetDiscardRectangleEnableEXT and vkCmdSetDiscardRectangleModeEXT. Applications that use these dynamic states must ensure the implementation advertises at least `specVersion` `2` of this extension.

### New Commands

- vkCmdSetDiscardRectangleEXT
- vkCmdSetDiscardRectangleEnableEXT
- vkCmdSetDiscardRectangleModeEXT

### New Structures

- Extending VkGraphicsPipelineCreateInfo:
  - VkPipelineDiscardRectangleStateCreateInfoEXT
- Extending VkPhysicalDeviceProperties2:
  - VkPhysicalDeviceDiscardRectanglePropertiesEXT

### New Enums

- VkDiscardRectangleModeEXT

### New Bitmasks

- VkPipelineDiscardRectangleStateCreateFlagsEXT

### New Enum Constants

- `VK_EXT_DISCARD_RECTANGLES_EXTENSION_NAME`
- `VK_EXT_DISCARD_RECTANGLES_SPEC_VERSION`
- Extending VkDynamicState:
  - `VK_DYNAMIC_STATE_DISCARD_RECTANGLE_ENABLE_EXT`
  - `VK_DYNAMIC_STATE_DISCARD_RECTANGLE_EXT`
  - `VK_DYNAMIC_STATE_DISCARD_RECTANGLE_MODE_EXT`
- Extending VkStructureType:

- `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DISCARD_RECTANGLE_PROPERTIES_EXT`
- `VK_STRUCTURE_TYPE_PIPELINE_DISCARD_RECTANGLE_STATE_CREATE_INFO_EXT`

### Version History

- Revision 2, 2023-01-18 (Piers Daniell)
  - Add dynamic states for discard rectangle enable/disable and mode.
- Revision 1, 2016-12-22 (Piers Daniell)
  - Internal revisions

## VK_EXT_display_control

### Name String

`VK_EXT_display_control`

### Extension Type

Device extension

### Registered Extension Number

92

### Revision

1

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_EXT_display_surface_counter
and
VK_KHR_swapchain

### Contact

- James Jones [GitHub]cubanismo

### Other Extension Metadata

### Last Modified Date

2016-12-13

### IP Status

No known IP claims.

### Contributors

- Pierre Boudier, NVIDIA
- James Jones, NVIDIA

- Damien Leone, NVIDIA
- Pierre-Loup Griffais, Valve
- Daniel Vetter, Intel

### Description

This extension defines a set of utility functions for use with the `VK_KHR_display` and `VK_KHR_display_swapchain` extensions.

### New Commands

- vkDisplayPowerControlEXT
- vkGetSwapchainCounterEXT
- vkRegisterDeviceEventEXT
- vkRegisterDisplayEventEXT

### New Structures

- VkDeviceEventInfoEXT
- VkDisplayEventInfoEXT
- VkDisplayPowerInfoEXT
- Extending VkSwapchainCreateInfoKHR:
  - VkSwapchainCounterCreateInfoEXT

### New Enums

- VkDeviceEventTypeEXT
- VkDisplayEventTypeEXT
- VkDisplayPowerStateEXT

### New Enum Constants

- `VK_EXT_DISPLAY_CONTROL_EXTENSION_NAME`
- `VK_EXT_DISPLAY_CONTROL_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_DEVICE_EVENT_INFO_EXT`
  - `VK_STRUCTURE_TYPE_DISPLAY_EVENT_INFO_EXT`
  - `VK_STRUCTURE_TYPE_DISPLAY_POWER_INFO_EXT`
  - `VK_STRUCTURE_TYPE_SWAPCHAIN_COUNTER_CREATE_INFO_EXT`

### Issues

1) Should this extension add an explicit “WaitForVsync” API or a fence signaled at vsync that the

application can wait on?

**RESOLVED**: A fence. A separate API could later be provided that allows exporting the fence to a native object that could be inserted into standard run loops on POSIX and Windows systems.

2) Should callbacks be added for a vsync event, or in general to monitor events in Vulkan?

**RESOLVED**: No, fences should be used. Some events are generated by interrupts which are managed in the kernel. In order to use a callback provided by the application, drivers would need to have the userspace driver spawn threads that would wait on the kernel event, and hence the callbacks could be difficult for the application to synchronize with its other work given they would arrive on a foreign thread.

3) Should vblank or scanline events be exposed?

**RESOLVED**: Vblank events. Scanline events could be added by a separate extension, but the latency of processing an interrupt and waking up a userspace event is high enough that the accuracy of a scanline event would be rather low. Further, per-scanline interrupts are not supported by all hardware.

#### Version History

- Revision 1, 2016-12-13 (James Jones)
  - Initial draft

## VK_EXT_display_surface_counter

#### Name String

`VK_EXT_display_surface_counter`

#### Extension Type

Instance extension

#### Registered Extension Number

91

#### Revision

1

#### Ratification Status

Ratified

#### Extension and Version Dependencies

VK_KHR_display

#### Contact

- James Jones [GitHub]cubanismo

### Other Extension Metadata

#### Last Modified Date

2016-12-13

#### IP Status

No known IP claims.

#### Contributors

- Pierre Boudier, NVIDIA
- James Jones, NVIDIA
- Damien Leone, NVIDIA
- Pierre-Loup Griffais, Valve
- Daniel Vetter, Intel

### Description

This extension defines a vertical blanking period counter associated with display surfaces. It provides a mechanism to query support for such a counter from a VkSurfaceKHR object.

### New Commands

- vkGetPhysicalDeviceSurfaceCapabilities2EXT

### New Structures

- VkSurfaceCapabilities2EXT

### New Enums

- VkSurfaceCounterFlagBitsEXT

### New Bitmasks

- VkSurfaceCounterFlagsEXT

### New Enum Constants

- `VK_EXT_DISPLAY_SURFACE_COUNTER_EXTENSION_NAME`
- `VK_EXT_DISPLAY_SURFACE_COUNTER_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_SURFACE_CAPABILITIES2_EXT`
  - `VK_STRUCTURE_TYPE_SURFACE_CAPABILITIES_2_EXT`

### Version History

- Revision 1, 2016-12-13 (James Jones)

- Initial draft

## VK_EXT_dynamic_rendering_unused_attachments

### Name String

`VK_EXT_dynamic_rendering_unused_attachments`

### Extension Type

Device extension

### Registered Extension Number

500

### Revision

1

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1
and
VK_KHR_dynamic_rendering
or
Vulkan Version 1.3

### Contact

- Piers Daniell [GitHub]pdaniell-nv

### Extension Proposal

VK_EXT_dynamic_rendering_unused_attachments

### Other Extension Metadata

### Last Modified Date

2023-05-22

### IP Status

No known IP claims.

### Contributors

- Daniel Story, Nintendo
- Hans-Kristian Arntzen, Valve
- Jan-Harald Fredriksen, Arm
- James Fitzpatrick, Imagination Technologies

- Pan Gao, Huawei Technologies
- Ricardo Garcia, Igalia
- Stu Smith, AMD

#### Description

This extension lifts some restrictions in the `VK_KHR_dynamic_rendering` extension to allow render pass instances and bound pipelines within those render pass instances to have an unused attachment specified in one but not the other. It also allows pipelines to use different formats in a render pass as long the attachment is NULL.

#### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceDynamicRenderingUnusedAttachmentsFeaturesEXT

#### New Enum Constants

- `VK_EXT_DYNAMIC_RENDERING_UNUSED_ATTACHMENTS_EXTENSION_NAME`
- `VK_EXT_DYNAMIC_RENDERING_UNUSED_ATTACHMENTS_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DYNAMIC_RENDERING_UNUSED_ATTACHMENTS_FEATURES_EXT`

#### Issues

None.

#### Version History

- Revision 1, 2023-05-22 (Piers Daniell)
  - Internal revisions

## VK_EXT_extended_dynamic_state

#### Name String

`VK_EXT_extended_dynamic_state`

#### Extension Type

Device extension

#### Registered Extension Number

268

#### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### Deprecation State

- *Promoted* to Vulkan 1.3

### Contact

- Piers Daniell [GitHub]pdaniell-nv

### Other Extension Metadata

### Last Modified Date

2019-12-09

### IP Status

No known IP claims.

### Contributors

- Dan Ginsburg, Valve Corporation
- Graeme Leese, Broadcom
- Hans-Kristian Arntzen, Valve Corporation
- Jan-Harald Fredriksen, Arm Limited
- Faith Ekstrand, Intel
- Jeff Bolz, NVIDIA
- Jesse Hall, Google
- Philip Rebohle, Valve Corporation
- Stuart Smith, Imagination Technologies
- Tobias Hector, AMD

### Description

This extension adds some more dynamic state to support applications that need to reduce the number of pipeline state objects they compile and bind.

### New Commands

- vkCmdBindVertexBuffers2EXT
- vkCmdSetCullModeEXT
- vkCmdSetDepthBoundsTestEnableEXT

- vkCmdSetDepthCompareOpEXT
- vkCmdSetDepthTestEnableEXT
- vkCmdSetDepthWriteEnableEXT
- vkCmdSetFrontFaceEXT
- vkCmdSetPrimitiveTopologyEXT
- vkCmdSetScissorWithCountEXT
- vkCmdSetStencilOpEXT
- vkCmdSetStencilTestEnableEXT
- vkCmdSetViewportWithCountEXT

#### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceExtendedDynamicStateFeaturesEXT

#### New Enum Constants

- `VK_EXT_EXTENDED_DYNAMIC_STATE_EXTENSION_NAME`
- `VK_EXT_EXTENDED_DYNAMIC_STATE_SPEC_VERSION`
- Extending VkDynamicState:
  - `VK_DYNAMIC_STATE_CULL_MODE_EXT`
  - `VK_DYNAMIC_STATE_DEPTH_BOUNDS_TEST_ENABLE_EXT`
  - `VK_DYNAMIC_STATE_DEPTH_COMPARE_OP_EXT`
  - `VK_DYNAMIC_STATE_DEPTH_TEST_ENABLE_EXT`
  - `VK_DYNAMIC_STATE_DEPTH_WRITE_ENABLE_EXT`
  - `VK_DYNAMIC_STATE_FRONT_FACE_EXT`
  - `VK_DYNAMIC_STATE_PRIMITIVE_TOPOLOGY_EXT`
  - `VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT_EXT`
  - `VK_DYNAMIC_STATE_STENCIL_OP_EXT`
  - `VK_DYNAMIC_STATE_STENCIL_TEST_ENABLE_EXT`
  - `VK_DYNAMIC_STATE_VERTEX_INPUT_BINDING_STRIDE_EXT`
  - `VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT_EXT`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTENDED_DYNAMIC_STATE_FEATURES_EXT`

#### Promotion to Vulkan 1.3

This extension has been partially promoted. All dynamic state enumerants and commands in this extension are included in core Vulkan 1.3, with the EXT suffix omitted. The feature structure is not

promoted. Extension interfaces that were promoted remain available as aliases of the core functionality.

#### Issues

1) Why are the values of `pStrides` in vkCmdBindVertexBuffers2EXT limited to be between 0 and the maximum extent of the binding, when this restriction is not present for the same static state?

Implementing these edge cases adds overhead to some implementations that would require significant cost when calling this function, and the intention is that this state should be more or less free to change.

VK_EXT_vertex_input_dynamic_state allows the stride to be changed freely when supported via vkCmdSetVertexInputEXT.

#### Version History

- Revision 1, 2019-12-09 (Piers Daniell)
  - Internal revisions

## VK_EXT_extended_dynamic_state2

#### Name String

`VK_EXT_extended_dynamic_state2`

#### Extension Type

Device extension

#### Registered Extension Number

378

#### Revision

1

#### Ratification Status

Not ratified

#### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

#### Deprecation State

- *Promoted* to Vulkan 1.3

#### Contact

- Vikram Kushwaha [GitHub]vkushwaha-nv

### Other Extension Metadata

#### Last Modified Date

2021-04-12

#### IP Status

No known IP claims.

#### Contributors

- Vikram Kushwaha, NVIDIA
- Piers Daniell, NVIDIA
- Jeff Bolz, NVIDIA

#### Description

This extension adds some more dynamic state to support applications that need to reduce the number of pipeline state objects they compile and bind.

#### New Commands

- vkCmdSetDepthBiasEnableEXT
- vkCmdSetLogicOpEXT
- vkCmdSetPatchControlPointsEXT
- vkCmdSetPrimitiveRestartEnableEXT
- vkCmdSetRasterizerDiscardEnableEXT

#### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceExtendedDynamicState2FeaturesEXT

#### New Enum Constants

- `VK_EXT_EXTENDED_DYNAMIC_STATE_2_EXTENSION_NAME`
- `VK_EXT_EXTENDED_DYNAMIC_STATE_2_SPEC_VERSION`
- Extending VkDynamicState:
  - `VK_DYNAMIC_STATE_DEPTH_BIAS_ENABLE_EXT`
  - `VK_DYNAMIC_STATE_LOGIC_OP_EXT`
  - `VK_DYNAMIC_STATE_PATCH_CONTROL_POINTS_EXT`
  - `VK_DYNAMIC_STATE_PRIMITIVE_RESTART_ENABLE_EXT`
  - `VK_DYNAMIC_STATE_RASTERIZER_DISCARD_ENABLE_EXT`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTENDED_DYNAMIC_STATE_2_FEATURES_EXT`

### Promotion to Vulkan 1.3

This extension has been partially promoted. The dynamic state enumerants `VK_DYNAMIC_STATE_DEPTH_BIAS_ENABLE_EXT`, `VK_DYNAMIC_STATE_PRIMITIVE_RESTART_ENABLE_EXT`, and `VK_DYNAMIC_STATE_RASTERIZER_DISCARD_ENABLE_EXT`; and the corresponding commands in this extension are included in core Vulkan 1.3, with the EXT suffix omitted. The enumerants and commands for dynamic logic operation and patch control points are not promoted, nor is the feature structure. Extension interfaces that were promoted remain available as aliases of the core functionality.

### Version History

- Revision 1, 2021-04-12 (Vikram Kushwaha)
  - Internal revisions

## VK_EXT_extended_dynamic_state3

### Name String

`VK_EXT_extended_dynamic_state3`

### Extension Type

Device extension

### Registered Extension Number

456

### Revision

2

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### API Interactions

- Interacts with VK_VERSION_1_1
- Interacts with VK_EXT_blend_operation_advanced
- Interacts with VK_EXT_conservative_rasterization
- Interacts with VK_EXT_depth_clip_control
- Interacts with VK_EXT_depth_clip_enable
- Interacts with VK_EXT_line_rasterization
- Interacts with VK_EXT_provoking_vertex
- Interacts with VK_EXT_sample_locations

- Interacts with VK_EXT_transform_feedback
- Interacts with VK_KHR_maintenance2
- Interacts with VK_NV_clip_space_w_scaling
- Interacts with VK_NV_coverage_reduction_mode
- Interacts with VK_NV_fragment_coverage_to_color
- Interacts with VK_NV_framebuffer_mixed_samples
- Interacts with VK_NV_representative_fragment_test
- Interacts with VK_NV_shading_rate_image
- Interacts with VK_NV_viewport_swizzle

**Contact**

- Piers Daniell [GitHub]pdaniell-nv

**Extension Proposal**

VK_EXT_extended_dynamic_state3

### Other Extension Metadata

**Last Modified Date**

2022-09-02

**IP Status**

No known IP claims.

**Contributors**

- Daniel Story, Nintendo
- Jamie Madill, Google
- Jan-Harald Fredriksen, Arm
- Faith Ekstrand, Collabora
- Mike Blumenkrantz, Valve
- Ricardo Garcia, Igalia
- Samuel Pitoiset, Valve
- Shahbaz Youssefi, Google
- Stu Smith, AMD
- Tapani Pälli, Intel

### Description

This extension adds almost all of the remaining pipeline state as dynamic state to help applications further reduce the number of monolithic pipelines they need to create and bind.

**New Commands**

- vkCmdSetAlphaToCoverageEnableEXT
- vkCmdSetAlphaToOneEnableEXT
- vkCmdSetColorBlendEnableEXT
- vkCmdSetColorBlendEquationEXT
- vkCmdSetColorWriteMaskEXT
- vkCmdSetDepthClampEnableEXT
- vkCmdSetLogicOpEnableEXT
- vkCmdSetPolygonModeEXT
- vkCmdSetRasterizationSamplesEXT
- vkCmdSetSampleMaskEXT

If VK_EXT_blend_operation_advanced is supported:

- vkCmdSetColorBlendAdvancedEXT

If VK_EXT_conservative_rasterization is supported:

- vkCmdSetConservativeRasterizationModeEXT
- vkCmdSetExtraPrimitiveOverestimationSizeEXT

If VK_EXT_depth_clip_control is supported:

- vkCmdSetDepthClipNegativeOneToOneEXT

If VK_EXT_depth_clip_enable is supported:

- vkCmdSetDepthClipEnableEXT

If VK_EXT_line_rasterization is supported:

- vkCmdSetLineRasterizationModeEXT
- vkCmdSetLineStippleEnableEXT

If VK_EXT_provoking_vertex is supported:

- vkCmdSetProvokingVertexModeEXT

If VK_EXT_sample_locations is supported:

- vkCmdSetSampleLocationsEnableEXT

If VK_EXT_transform_feedback is supported:

- vkCmdSetRasterizationStreamEXT

If VK_KHR_maintenance2 or Vulkan Version 1.1 is supported:

- vkCmdSetTessellationDomainOriginEXT

If VK_NV_clip_space_w_scaling is supported:

- vkCmdSetViewportWScalingEnableNV

If VK_NV_coverage_reduction_mode is supported:

- vkCmdSetCoverageReductionModeNV

If VK_NV_fragment_coverage_to_color is supported:

- vkCmdSetCoverageToColorEnableNV
- vkCmdSetCoverageToColorLocationNV

If VK_NV_framebuffer_mixed_samples is supported:

- vkCmdSetCoverageModulationModeNV
- vkCmdSetCoverageModulationTableEnableNV
- vkCmdSetCoverageModulationTableNV

If VK_NV_representative_fragment_test is supported:

- vkCmdSetRepresentativeFragmentTestEnableNV

If VK_NV_shading_rate_image is supported:

- vkCmdSetShadingRateImageEnableNV

If VK_NV_viewport_swizzle is supported:

- vkCmdSetViewportSwizzleNV

**New Structures**

- VkColorBlendAdvancedEXT
- VkColorBlendEquationEXT
- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceExtendedDynamicState3FeaturesEXT
- Extending VkPhysicalDeviceProperties2:
  - VkPhysicalDeviceExtendedDynamicState3PropertiesEXT

**New Enum Constants**

- `VK_EXT_EXTENDED_DYNAMIC_STATE_3_EXTENSION_NAME`
- `VK_EXT_EXTENDED_DYNAMIC_STATE_3_SPEC_VERSION`
- Extending VkDynamicState:

    - VK_DYNAMIC_STATE_ALPHA_TO_COVERAGE_ENABLE_EXT
    - VK_DYNAMIC_STATE_ALPHA_TO_ONE_ENABLE_EXT
    - VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT
    - VK_DYNAMIC_STATE_COLOR_BLEND_EQUATION_EXT
    - VK_DYNAMIC_STATE_COLOR_WRITE_MASK_EXT
    - VK_DYNAMIC_STATE_DEPTH_CLAMP_ENABLE_EXT
    - VK_DYNAMIC_STATE_LOGIC_OP_ENABLE_EXT
    - VK_DYNAMIC_STATE_POLYGON_MODE_EXT
    - VK_DYNAMIC_STATE_RASTERIZATION_SAMPLES_EXT
    - VK_DYNAMIC_STATE_SAMPLE_MASK_EXT
- Extending VkStructureType:
    - VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTENDED_DYNAMIC_STATE_3_FEATURES_EXT
    - VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTENDED_DYNAMIC_STATE_3_PROPERTIES_EXT

If VK_EXT_blend_operation_advanced is supported:

- Extending VkDynamicState:
    - VK_DYNAMIC_STATE_COLOR_BLEND_ADVANCED_EXT

If VK_EXT_conservative_rasterization is supported:

- Extending VkDynamicState:
    - VK_DYNAMIC_STATE_CONSERVATIVE_RASTERIZATION_MODE_EXT
    - VK_DYNAMIC_STATE_EXTRA_PRIMITIVE_OVERESTIMATION_SIZE_EXT

If VK_EXT_depth_clip_control is supported:

- Extending VkDynamicState:
    - VK_DYNAMIC_STATE_DEPTH_CLIP_NEGATIVE_ONE_TO_ONE_EXT

If VK_EXT_depth_clip_enable is supported:

- Extending VkDynamicState:
    - VK_DYNAMIC_STATE_DEPTH_CLIP_ENABLE_EXT

If VK_EXT_line_rasterization is supported:

- Extending VkDynamicState:
    - VK_DYNAMIC_STATE_LINE_RASTERIZATION_MODE_EXT
    - VK_DYNAMIC_STATE_LINE_STIPPLE_ENABLE_EXT

If VK_EXT_provoking_vertex is supported:

- Extending VkDynamicState:
  - VK_DYNAMIC_STATE_PROVOKING_VERTEX_MODE_EXT

If VK_EXT_sample_locations is supported:

- Extending VkDynamicState:
  - VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT

If VK_EXT_transform_feedback is supported:

- Extending VkDynamicState:
  - VK_DYNAMIC_STATE_RASTERIZATION_STREAM_EXT

If VK_KHR_maintenance2 or Vulkan Version 1.1 is supported:

- Extending VkDynamicState:
  - VK_DYNAMIC_STATE_TESSELLATION_DOMAIN_ORIGIN_EXT

If VK_NV_clip_space_w_scaling is supported:

- Extending VkDynamicState:
  - VK_DYNAMIC_STATE_VIEWPORT_W_SCALING_ENABLE_NV

If VK_NV_coverage_reduction_mode is supported:

- Extending VkDynamicState:
  - VK_DYNAMIC_STATE_COVERAGE_REDUCTION_MODE_NV

If VK_NV_fragment_coverage_to_color is supported:

- Extending VkDynamicState:
  - VK_DYNAMIC_STATE_COVERAGE_TO_COLOR_ENABLE_NV
  - VK_DYNAMIC_STATE_COVERAGE_TO_COLOR_LOCATION_NV

If VK_NV_framebuffer_mixed_samples is supported:

- Extending VkDynamicState:
  - VK_DYNAMIC_STATE_COVERAGE_MODULATION_MODE_NV
  - VK_DYNAMIC_STATE_COVERAGE_MODULATION_TABLE_ENABLE_NV
  - VK_DYNAMIC_STATE_COVERAGE_MODULATION_TABLE_NV

If VK_NV_representative_fragment_test is supported:

- Extending VkDynamicState:
  - VK_DYNAMIC_STATE_REPRESENTATIVE_FRAGMENT_TEST_ENABLE_NV

If VK_NV_shading_rate_image is supported:

- Extending VkDynamicState:
  - `VK_DYNAMIC_STATE_SHADING_RATE_IMAGE_ENABLE_NV`

If VK_NV_viewport_swizzle is supported:

- Extending VkDynamicState:
  - `VK_DYNAMIC_STATE_VIEWPORT_SWIZZLE_NV`

#### Issues

1) What about the VkPipelineMultisampleStateCreateInfo state `sampleShadingEnable` and `minSampleShading`?

**UNRESOLVED**

- `sampleShadingEnable` and `minSampleShading` are required when compiling the fragment shader, and it is not meaningful to set them dynamically since they always need to match the fragment shader state, so this hardware state may as well just come from the pipeline with the fragment shader.

#### Version History

- Revision 2, 2022-07-18 (Piers Daniell)
  - Added rasterizationSamples
- Revision 1, 2022-05-18 (Piers Daniell)
  - Internal revisions

## VK_EXT_external_memory_acquire_unmodified

#### Name String

`VK_EXT_external_memory_acquire_unmodified`

#### Extension Type

Device extension

#### Registered Extension Number

454

#### Revision

1

#### Ratification Status

Ratified

#### Extension and Version Dependencies

VK_KHR_external_memory
or
Vulkan Version 1.1

**Contact**

- Lina Versace [GitHub]versalinyaa

**Extension Proposal**

VK_EXT_external_memory_acquire_unmodified

**Other Extension Metadata**

**Last Modified Date**

2023-03-09

**Contributors**

- Lina Versace, Google
- Chia-I Wu, Google
- James Jones, NVIDIA
- Yiwei Zhang, Google

**Description**

A memory barrier **may** have a performance penalty when acquiring ownership of a subresource range from an external queue family. This extension provides API that **may** reduce the performance penalty if ownership of the subresource range was previously released to the external queue family and if the resource's memory has remained unmodified between the release and acquire operations.

**New Structures**

- Extending VkBufferMemoryBarrier, VkBufferMemoryBarrier2, VkImageMemoryBarrier, VkImageMemoryBarrier2:
  - VkExternalMemoryAcquireUnmodifiedEXT

**New Enum Constants**

- `VK_EXT_EXTERNAL_MEMORY_ACQUIRE_UNMODIFIED_EXTENSION_NAME`
- `VK_EXT_EXTERNAL_MEMORY_ACQUIRE_UNMODIFIED_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_EXTERNAL_MEMORY_ACQUIRE_UNMODIFIED_EXT`

**Version History**

- Revision 1, 2023-03-09 (Lina Versace)
  - Initial revision

## VK_EXT_external_memory_dma_buf

### Name String

`VK_EXT_external_memory_dma_buf`

### Extension Type

Device extension

### Registered Extension Number

126

### Revision

1

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_external_memory_fd

### Contact

- Lina Versace [GitHub]versalinyaa

### Other Extension Metadata

### Last Modified Date

2017-10-10

### IP Status

No known IP claims.

### Contributors

- Lina Versace, Google
- James Jones, NVIDIA
- Faith Ekstrand, Intel

### Description

A `dma_buf` is a type of file descriptor, defined by the Linux kernel, that allows sharing memory across kernel device drivers and across processes. This extension enables applications to import a `dma_buf` as VkDeviceMemory, to export VkDeviceMemory as a `dma_buf`, and to create VkBuffer objects that **can** be bound to that memory.

### New Enum Constants

- `VK_EXT_EXTERNAL_MEMORY_DMA_BUF_EXTENSION_NAME`
- `VK_EXT_EXTERNAL_MEMORY_DMA_BUF_SPEC_VERSION`
- Extending VkExternalMemoryHandleTypeFlagBits:
  - `VK_EXTERNAL_MEMORY_HANDLE_TYPE_DMA_BUF_BIT_EXT`

### Issues

1) How does the application, when creating a VkImage that it intends to bind to `dma_buf` VkDeviceMemory containing an externally produced image, specify the memory layout (such as row pitch and DRM format modifier) of the VkImage? In other words, how does the application achieve behavior comparable to that provided by `EGL_EXT_image_dma_buf_import` and `EGL_EXT_image_dma_buf_import_modifiers` ?

**RESOLVED**: Features comparable to those in `EGL_EXT_image_dma_buf_import` and `EGL_EXT_image_dma_buf_import_modifiers` will be provided by an extension layered atop this one.

2) Without the ability to specify the memory layout of external `dma_buf` images, how is this extension useful?

**RESOLVED**: This extension provides exactly one new feature: the ability to import/export between `dma_buf` and VkDeviceMemory. This feature, together with features provided by `VK_KHR_external_memory_fd`, is sufficient to bind a VkBuffer to `dma_buf`.

### Version History

- Revision 1, 2017-10-10 (Lina Versace)
  - Squashed internal revisions

## VK_EXT_external_memory_host

### Name String

`VK_EXT_external_memory_host`

### Extension Type

Device extension

### Registered Extension Number

179

### Revision

1

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_external_memory
or
Vulkan Version 1.1

### Contact

- Daniel Rakos [GitHub]drakos-amd

### Other Extension Metadata

#### Last Modified Date

2017-11-10

#### IP Status

No known IP claims.

#### Contributors

- Jaakko Konttinen, AMD
- David Mao, AMD
- Daniel Rakos, AMD
- Tobias Hector, Imagination Technologies
- Faith Ekstrand, Intel
- James Jones, NVIDIA

### Description

This extension enables an application to import host allocations and host mapped foreign device memory to Vulkan memory objects.

### New Commands

- vkGetMemoryHostPointerPropertiesEXT

### New Structures

- VkMemoryHostPointerPropertiesEXT
- Extending VkMemoryAllocateInfo:
  - VkImportMemoryHostPointerInfoEXT
- Extending VkPhysicalDeviceProperties2:
  - VkPhysicalDeviceExternalMemoryHostPropertiesEXT

### New Enum Constants

- `VK_EXT_EXTERNAL_MEMORY_HOST_EXTENSION_NAME`
- `VK_EXT_EXTERNAL_MEMORY_HOST_SPEC_VERSION`
- Extending VkExternalMemoryHandleTypeFlagBits:
  - `VK_EXTERNAL_MEMORY_HANDLE_TYPE_HOST_ALLOCATION_BIT_EXT`
  - `VK_EXTERNAL_MEMORY_HANDLE_TYPE_HOST_MAPPED_FOREIGN_MEMORY_BIT_EXT`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_IMPORT_MEMORY_HOST_POINTER_INFO_EXT`
  - `VK_STRUCTURE_TYPE_MEMORY_HOST_POINTER_PROPERTIES_EXT`

- `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTERNAL_MEMORY_HOST_PROPERTIES_EXT`

**Issues**

1) What memory type has to be used to import host pointers?

**RESOLVED**: Depends on the implementation. Applications have to use the new vkGetMemoryHostPointerPropertiesEXT command to query the supported memory types for a particular host pointer. The reported memory types may include memory types that come from a memory heap that is otherwise not usable for regular memory object allocation and thus such a heap's size may be zero.

2) Can the application still access the contents of the host allocation after importing?

**RESOLVED**: Yes. However, usual synchronization requirements apply.

3) Can the application free the host allocation?

**RESOLVED**: No, it violates valid usage conditions. Using the memory object imported from a host allocation that is already freed thus results in undefined behavior.

4) Is vkMapMemory expected to return the same host address which was specified when importing it to the memory object?

**RESOLVED**: No. Implementations are allowed to return the same address but it is not required. Some implementations might return a different virtual mapping of the allocation, although the same physical pages will be used.

5) Is there any limitation on the alignment of the host pointer and/or size?

**RESOLVED**: Yes. Both the address and the size have to be an integer multiple of `minImportedHostPointerAlignment`. In addition, some platforms and foreign devices may have additional restrictions.

6) Can the same host allocation be imported multiple times into a given physical device?

**RESOLVED**: No, at least not guaranteed by this extension. Some platforms do not allow locking the same physical pages for device access multiple times, so attempting to do it may result in undefined behavior.

7) Does this extension support exporting the new handle type?

**RESOLVED**: No.

8) Should we include the possibility to import host mapped foreign device memory using this API?

**RESOLVED**: Yes, through a separate handle type. Implementations are still allowed to support only one of the handle types introduced by this extension by not returning import support for a particular handle type as returned in VkExternalMemoryPropertiesKHR.

### Version History

- Revision 1, 2017-11-10 (Daniel Rakos)
  - Internal revisions

## VK_EXT_filter_cubic

### Name String

`VK_EXT_filter_cubic`

### Extension Type

Device extension

### Registered Extension Number

171

### Revision

3

### Ratification Status

Ratified

### Extension and Version Dependencies

None

### Contact

- Matthew Netsch [GitHub]mnetsch

### Other Extension Metadata

### Last Modified Date

2019-12-13

### Contributors

- Bill Licea-Kane, Qualcomm Technologies, Inc.
- Andrew Garrard, Samsung
- Daniel Koch, NVIDIA
- Donald Scorgie, Imagination Technologies
- Graeme Leese, Broadcom
- Jan-Harald Fredriksen, ARM
- Jeff Leger, Qualcomm Technologies, Inc.
- Tobias Hector, AMD
- Tom Olson, ARM
- Stuart Smith, Imagination Technologies

### Description

`VK_EXT_filter_cubic` extends `VK_IMG_filter_cubic`.

It documents cubic filtering of other image view types. It adds new structures that **can** be added to the `pNext` chain of VkPhysicalDeviceImageFormatInfo2 and VkImageFormatProperties2 that **can** be used to determine which image types and which image view types support cubic filtering.

### New Structures

- Extending VkImageFormatProperties2:
  - VkFilterCubicImageViewImageFormatPropertiesEXT
- Extending VkPhysicalDeviceImageFormatInfo2:
  - VkPhysicalDeviceImageViewImageFormatInfoEXT

### New Enum Constants

- `VK_EXT_FILTER_CUBIC_EXTENSION_NAME`
- `VK_EXT_FILTER_CUBIC_SPEC_VERSION`
- Extending VkFilter:
  - `VK_FILTER_CUBIC_EXT`
- Extending VkFormatFeatureFlagBits:
  - `VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_CUBIC_BIT_EXT`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_FILTER_CUBIC_IMAGE_VIEW_IMAGE_FORMAT_PROPERTIES_EXT`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_IMAGE_VIEW_IMAGE_FORMAT_INFO_EXT`

### Version History

- Revision 3, 2019-12-13 (wwlk)
  - Delete requirement to cubic filter the formats USCALED_PACKED32, SSCALED_PACKED32, UINT_PACK32, and SINT_PACK32 (cut/paste error)
- Revision 2, 2019-06-05 (wwlk)
  - Clarify 1D optional
- Revision 1, 2019-01-24 (wwlk)
  - Initial version

## VK_EXT_fragment_density_map

### Name String

`VK_EXT_fragment_density_map`

**Extension Type**

Device extension

**Registered Extension Number**

219

**Revision**

2

**Ratification Status**

Not ratified

**Extension and Version Dependencies**

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

**API Interactions**

- Interacts with VK_VERSION_1_3
- Interacts with VK_KHR_format_feature_flags2

**SPIR-V Dependencies**

- SPV_EXT_fragment_invocation_density

**Contact**

- Matthew Netsch [GitHub]mnetsch

**Other Extension Metadata**

**Last Modified Date**

2021-09-30

**Interactions and External Dependencies**

- This extension provides API support for `GL_EXT_fragment_invocation_density`

**Contributors**

- Matthew Netsch, Qualcomm Technologies, Inc.
- Robert VanReenen, Qualcomm Technologies, Inc.
- Jonathan Wicks, Qualcomm Technologies, Inc.
- Tate Hornbeck, Qualcomm Technologies, Inc.
- Sam Holmes, Qualcomm Technologies, Inc.
- Jeff Leger, Qualcomm Technologies, Inc.
- Jan-Harald Fredriksen, ARM
- Jeff Bolz, NVIDIA
- Pat Brown, NVIDIA

- Daniel Rakos, AMD
- Piers Daniell, NVIDIA

### Description

This extension allows an application to specify areas of the render target where the fragment shader may be invoked fewer times. These fragments are broadcasted out to multiple pixels to cover the render target.

The primary use of this extension is to reduce workloads in areas where lower quality may not be perceived such as the distorted edges of a lens or the periphery of a user's gaze.

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceFragmentDensityMapFeaturesEXT
- Extending VkPhysicalDeviceProperties2:
  - VkPhysicalDeviceFragmentDensityMapPropertiesEXT
- Extending VkRenderPassCreateInfo, VkRenderPassCreateInfo2:
  - VkRenderPassFragmentDensityMapCreateInfoEXT

### New Enum Constants

- `VK_EXT_FRAGMENT_DENSITY_MAP_EXTENSION_NAME`
- `VK_EXT_FRAGMENT_DENSITY_MAP_SPEC_VERSION`
- Extending VkAccessFlagBits:
  - `VK_ACCESS_FRAGMENT_DENSITY_MAP_READ_BIT_EXT`
- Extending VkFormatFeatureFlagBits:
  - `VK_FORMAT_FEATURE_FRAGMENT_DENSITY_MAP_BIT_EXT`
- Extending VkImageCreateFlagBits:
  - `VK_IMAGE_CREATE_SUBSAMPLED_BIT_EXT`
- Extending VkImageLayout:
  - `VK_IMAGE_LAYOUT_FRAGMENT_DENSITY_MAP_OPTIMAL_EXT`
- Extending VkImageUsageFlagBits:
  - `VK_IMAGE_USAGE_FRAGMENT_DENSITY_MAP_BIT_EXT`
- Extending VkImageViewCreateFlagBits:
  - `VK_IMAGE_VIEW_CREATE_FRAGMENT_DENSITY_MAP_DYNAMIC_BIT_EXT`
- Extending VkPipelineStageFlagBits:
  - `VK_PIPELINE_STAGE_FRAGMENT_DENSITY_PROCESS_BIT_EXT`
- Extending VkSamplerCreateFlagBits:

    - VK_SAMPLER_CREATE_SUBSAMPLED_BIT_EXT
    - VK_SAMPLER_CREATE_SUBSAMPLED_COARSE_RECONSTRUCTION_BIT_EXT
- Extending VkStructureType:
    - VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_FRAGMENT_DENSITY_MAP_FEATURES_EXT
    - VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_FRAGMENT_DENSITY_MAP_PROPERTIES_EXT
    - VK_STRUCTURE_TYPE_RENDER_PASS_FRAGMENT_DENSITY_MAP_CREATE_INFO_EXT

If VK_KHR_format_feature_flags2 or Vulkan Version 1.3 is supported:

- Extending VkFormatFeatureFlagBits2:
    - VK_FORMAT_FEATURE_2_FRAGMENT_DENSITY_MAP_BIT_EXT

#### New or Modified Built-In Variables

- FragInvocationCountEXT
- FragSizeEXT

#### New SPIR-V Capabilities

- FragmentDensityEXT

#### Version History

- Revision 1, 2018-09-25 (Matthew Netsch)
    - Initial version
- Revision 2, 2021-09-30 (Jon Leech)
    - Add interaction with VK_KHR_format_feature_flags2 to vk.xml

## VK_EXT_fragment_density_map2

#### Name String

VK_EXT_fragment_density_map2

#### Extension Type

Device extension

#### Registered Extension Number

333

#### Revision

1

#### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_EXT_fragment_density_map

### Contact

- Matthew Netsch [GitHub]mnetsch

### Other Extension Metadata

### Last Modified Date

2020-06-16

### Interactions and External Dependencies

- Interacts with Vulkan 1.1

### Contributors

- Matthew Netsch, Qualcomm Technologies, Inc.
- Jonathan Tinkham, Qualcomm Technologies, Inc.
- Jonathan Wicks, Qualcomm Technologies, Inc.
- Jan-Harald Fredriksen, ARM

### Description

This extension adds additional features and properties to `VK_EXT_fragment_density_map` in order to reduce fragment density map host latency as well as improved queries for subsampled sampler implementation-dependent behavior.

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceFragmentDensityMap2FeaturesEXT
- Extending VkPhysicalDeviceProperties2:
  - VkPhysicalDeviceFragmentDensityMap2PropertiesEXT

### New Enum Constants

- `VK_EXT_FRAGMENT_DENSITY_MAP_2_EXTENSION_NAME`
- `VK_EXT_FRAGMENT_DENSITY_MAP_2_SPEC_VERSION`
- Extending VkImageViewCreateFlagBits:
  - `VK_IMAGE_VIEW_CREATE_FRAGMENT_DENSITY_MAP_DEFERRED_BIT_EXT`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_FRAGMENT_DENSITY_MAP_2_FEATURES_EXT`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_FRAGMENT_DENSITY_MAP_2_PROPERTIES_EXT`

### Version History

- Revision 1, 2020-06-16 (Matthew Netsch)
  - Initial version

## VK_EXT_fragment_shader_interlock

### Name String

`VK_EXT_fragment_shader_interlock`

### Extension Type

Device extension

### Registered Extension Number

252

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### SPIR-V Dependencies

- SPV_EXT_fragment_shader_interlock

### Contact

- Piers Daniell [GitHub]pdaniell-nv

### Other Extension Metadata

### Last Modified Date

2019-05-02

### Interactions and External Dependencies

- This extension provides API support for `GL_ARB_fragment_shader_interlock`

### Contributors

- Daniel Koch, NVIDIA
- Graeme Leese, Broadcom
- Jan-Harald Fredriksen, Arm
- Faith Ekstrand, Intel
- Jeff Bolz, NVIDIA

- Ruihao Zhang, Qualcomm
- Slawomir Grajewski, Intel
- Spencer Fricke, Samsung

### Description

This extension adds support for the `FragmentShaderPixelInterlockEXT`, `FragmentShaderSampleInterlockEXT`, and `FragmentShaderShadingRateInterlockEXT` capabilities from the `SPV_EXT_fragment_shader_interlock` extension to Vulkan.

Enabling these capabilities provides a critical section for fragment shaders to avoid overlapping pixels being processed at the same time, and certain guarantees about the ordering of fragment shader invocations of fragments of overlapping pixels.

This extension can be useful for algorithms that need to access per-pixel data structures via shader loads and stores. Algorithms using this extension can access per-pixel data structures in critical sections without other invocations accessing the same per-pixel data. Additionally, the ordering guarantees are useful for cases where the API ordering of fragments is meaningful. For example, applications may be able to execute programmable blending operations in the fragment shader, where the destination buffer is read via image loads and the final value is written via image stores.

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceFragmentShaderInterlockFeaturesEXT

### New Enum Constants

- `VK_EXT_FRAGMENT_SHADER_INTERLOCK_EXTENSION_NAME`
- `VK_EXT_FRAGMENT_SHADER_INTERLOCK_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_FRAGMENT_SHADER_INTERLOCK_FEATURES_EXT`

### New SPIR-V Capabilities

- `FragmentShaderInterlockEXT`
- `FragmentShaderPixelInterlockEXT`
- `FragmentShaderShadingRateInterlockEXT`

### Version History

- Revision 1, 2019-05-24 (Piers Daniell)
  - Internal revisions

## VK_EXT_frame_boundary

### Name String

`VK_EXT_frame_boundary`

### Extension Type

Device extension

### Registered Extension Number

376

### Revision

1

### Ratification Status

Ratified

### Extension and Version Dependencies

None

### Contact

- James Fitzpatrick [GitHub]jamesfitzpatrick

### Extension Proposal

VK_EXT_frame_boundary

### Other Extension Metadata

### Last Modified Date

2023-06-14

### Contributors

- James Fitzpatrick, Imagination Technologies
- Hugues Evrard, Google
- Melih Yasin Yalcin, Google
- Andrew Garrard, Imagination Technologies
- Jan-Harald Fredriksen, Arm
- Vassili Nikolaev, NVIDIA
- Ting Wei, Huawei

### Description

VK_EXT_frame_boundary is a device extension that helps **tools** (such as debuggers) to group queue submissions per frames in non-trivial scenarios, typically when vkQueuePresentKHR is not a relevant frame boundary delimiter.

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceFrameBoundaryFeaturesEXT
- Extending VkSubmitInfo, VkSubmitInfo2, VkPresentInfoKHR, VkBindSparseInfo:
  - VkFrameBoundaryEXT

### New Enums

- VkFrameBoundaryFlagBitsEXT

### New Bitmasks

- VkFrameBoundaryFlagsEXT

### New Enum Constants

- `VK_EXT_FRAME_BOUNDARY_EXTENSION_NAME`
- `VK_EXT_FRAME_BOUNDARY_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_FRAME_BOUNDARY_EXT`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_FRAME_BOUNDARY_FEATURES_EXT`

### Version History

- Revision 0, 2022-01-14 (Hugues Evard)
  - Initial proposal
- Revision 1, 2023-06-14 (James Fitzpatrick)
  - Initial draft

## VK_EXT_full_screen_exclusive

### Name String

`VK_EXT_full_screen_exclusive`

### Extension Type

Device extension

### Registered Extension Number

256

### Revision

4

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1
and
VK_KHR_surface
and
VK_KHR_get_surface_capabilities2
and
VK_KHR_swapchain

### API Interactions

- Interacts with VK_VERSION_1_1
- Interacts with VK_KHR_device_group
- Interacts with VK_KHR_win32_surface

### Contact

- James Jones [GitHub]cubanismo

### Other Extension Metadata

### Last Modified Date

2019-03-12

### IP Status

No known IP claims.

### Interactions and External Dependencies

- Interacts with Vulkan 1.1
- Interacts with `VK_KHR_device_group`
- Interacts with `VK_KHR_win32_surface`

### Contributors

- Hans-Kristian Arntzen, ARM
- Slawomir Grajewski, Intel
- Tobias Hector, AMD
- James Jones, NVIDIA
- Daniel Rakos, AMD
- Jeff Juliano, NVIDIA
- Joshua Schnarr, NVIDIA
- Aaron Hagan, AMD

### Description

This extension allows applications to set the policy for swapchain creation and presentation

mechanisms relating to full-screen access. Implementations may be able to acquire exclusive access to a particular display for an application window that covers the whole screen. This can increase performance on some systems by bypassing composition, however it can also result in disruptive or expensive transitions in the underlying windowing system when a change occurs.

Applications can choose between explicitly disallowing or allowing this behavior, letting the implementation decide, or managing this mode of operation directly using the new vkAcquireFullScreenExclusiveModeEXT and vkReleaseFullScreenExclusiveModeEXT commands.

#### New Commands

- vkAcquireFullScreenExclusiveModeEXT
- vkGetPhysicalDeviceSurfacePresentModes2EXT
- vkReleaseFullScreenExclusiveModeEXT

If VK_KHR_device_group or Vulkan Version 1.1 is supported:

- vkGetDeviceGroupSurfacePresentModes2EXT

#### New Structures

- Extending VkPhysicalDeviceSurfaceInfo2KHR, VkSwapchainCreateInfoKHR:
  - VkSurfaceFullScreenExclusiveInfoEXT
- Extending VkSurfaceCapabilities2KHR:
  - VkSurfaceCapabilitiesFullScreenExclusiveEXT

If VK_KHR_win32_surface is supported:

- Extending VkPhysicalDeviceSurfaceInfo2KHR, VkSwapchainCreateInfoKHR:
  - VkSurfaceFullScreenExclusiveWin32InfoEXT

#### New Enums

- VkFullScreenExclusiveEXT

#### New Enum Constants

- `VK_EXT_FULL_SCREEN_EXCLUSIVE_EXTENSION_NAME`
- `VK_EXT_FULL_SCREEN_EXCLUSIVE_SPEC_VERSION`
- Extending VkResult:
  - `VK_ERROR_FULL_SCREEN_EXCLUSIVE_MODE_LOST_EXT`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_SURFACE_CAPABILITIES_FULL_SCREEN_EXCLUSIVE_EXT`
  - `VK_STRUCTURE_TYPE_SURFACE_FULL_SCREEN_EXCLUSIVE_INFO_EXT`

If VK_KHR_win32_surface is supported:

- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_SURFACE_FULL_SCREEN_EXCLUSIVE_WIN32_INFO_EXT`

**Issues**

1) What should the extension & flag be called?

**RESOLVED**: VK_EXT_full_screen_exclusive.

Other options considered (prior to the app-controlled mode) were:

- VK_EXT_smooth_fullscreen_transition
- VK_EXT_fullscreen_behavior
- VK_EXT_fullscreen_preference
- VK_EXT_fullscreen_hint
- VK_EXT_fast_fullscreen_transition
- VK_EXT_avoid_fullscreen_exclusive

2) Do we need more than a boolean toggle?

**RESOLVED**: Yes.

Using an enum with default/allowed/disallowed/app-controlled enables applications to accept driver default behavior, specifically override it in either direction without implying the driver is ever required to use full-screen exclusive mechanisms, or manage this mode explicitly.

3) Should this be a KHR or EXT extension?

**RESOLVED**: EXT, in order to allow it to be shipped faster.

4) Can the fullscreen hint affect the surface capabilities, and if so, should the hint also be specified as input when querying the surface capabilities?

**RESOLVED**: Yes on both accounts.

While the hint does not guarantee a particular fullscreen mode will be used when the swapchain is created, it can sometimes imply particular modes will NOT be used. If the driver determines that it will opt-out of using a particular mode based on the policy, and knows it can only support certain capabilities if that mode is used, it would be confusing at best to the application to report those capabilities in such cases. Not allowing implementations to report this state to applications could result in situations where applications are unable to determine why swapchain creation fails when they specify certain hint values, which could result in never- terminating surface creation loops.

5) Should full-screen be one word or two?

**RESOLVED**: Two words.

"Fullscreen" is not in my dictionary, and web searches did not turn up definitive proof that it is a colloquially accepted compound word. Documentation for the corresponding Windows API

mechanisms dithers. The text consistently uses a hyphen, but none-the-less, there is a SetFullscreenState method in the DXGI swapchain object. Given this inconclusive external guidance, it is best to adhere to the Vulkan style guidelines and avoid inventing new compound words.

### Version History

- Revision 4, 2019-03-12 (Tobias Hector)
  - Added application-controlled mode, and related functions
  - Tidied up appendix
- Revision 3, 2019-01-03 (James Jones)
  - Renamed to VK_EXT_full_screen_exclusive
  - Made related adjustments to the tri-state enumerant names.
- Revision 2, 2018-11-27 (James Jones)
  - Renamed to VK_KHR_fullscreen_behavior
  - Switched from boolean flag to tri-state enum
- Revision 1, 2018-11-06 (James Jones)
  - Internal revision

## VK_EXT_graphics_pipeline_library

### Name String

`VK_EXT_graphics_pipeline_library`

### Extension Type

Device extension

### Registered Extension Number

321

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1
and
VK_KHR_pipeline_library

**Contact**

- Tobias Hector [GitHub]tobski

**Extension Proposal**

VK_EXT_graphics_pipeline_library

**Other Extension Metadata**

**Last Modified Date**

2021-08-17

**Contributors**

- Tobias Hector, AMD
- Chris Glover, Google
- Jeff Leger, Qualcomm
- Jan-Harald Fredriksen, Arm
- Piers Daniell, NVidia
- Boris Zanin, Mobica
- Krzysztof Niski, NVidia
- Dan Ginsburg, Valve
- Sebastian Aaltonen, Unity
- Arseny Kapoulkine, Roblox
- Calle Lejdfors, Ubisoft
- Tiago Rodrigues, Ubisoft
- Francois Duranleau, Gameloft

**Description**

This extension allows the separate compilation of four distinct parts of graphics pipelines, with the intent of allowing faster pipeline loading for applications reusing the same shaders or state in multiple pipelines. Each part can be independently compiled into a graphics pipeline library, with a final link step required to create an executable pipeline that can be bound to a command buffer.

**New Structures**

- Extending VkGraphicsPipelineCreateInfo:
  - VkGraphicsPipelineLibraryCreateInfoEXT
- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceGraphicsPipelineLibraryFeaturesEXT
- Extending VkPhysicalDeviceProperties2:
  - VkPhysicalDeviceGraphicsPipelineLibraryPropertiesEXT

### New Enums

- VkGraphicsPipelineLibraryFlagBitsEXT
- VkPipelineLayoutCreateFlagBits

### New Bitmasks

- VkGraphicsPipelineLibraryFlagsEXT

### New Enum Constants

- `VK_EXT_GRAPHICS_PIPELINE_LIBRARY_EXTENSION_NAME`
- `VK_EXT_GRAPHICS_PIPELINE_LIBRARY_SPEC_VERSION`
- Extending VkPipelineCreateFlagBits:
  - `VK_PIPELINE_CREATE_LINK_TIME_OPTIMIZATION_BIT_EXT`
  - `VK_PIPELINE_CREATE_RETAIN_LINK_TIME_OPTIMIZATION_INFO_BIT_EXT`
- Extending VkPipelineLayoutCreateFlagBits:
  - `VK_PIPELINE_LAYOUT_CREATE_INDEPENDENT_SETS_BIT_EXT`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_GRAPHICS_PIPELINE_LIBRARY_CREATE_INFO_EXT`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_GRAPHICS_PIPELINE_LIBRARY_FEATURES_EXT`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_GRAPHICS_PIPELINE_LIBRARY_PROPERTIES_EXT`

### Version History

- Revision 1, 2021-08-17 (Tobias Hector)
  - Initial draft.

## VK_EXT_hdr_metadata

### Name String

`VK_EXT_hdr_metadata`

### Extension Type

Device extension

### Registered Extension Number

106

### Revision

3

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_swapchain

### Contact

- Courtney Goeltzenleuchter [GitHub]courtney-g

### Other Extension Metadata

### Last Modified Date

2024-03-26

### IP Status

No known IP claims.

### Contributors

- Courtney Goeltzenleuchter, Google
- Sebastian Wick, Red Hat Inc.
- Tobias Hector, AMD

### Description

This extension defines two new structures and a function to assign SMPTE (the Society of Motion Picture and Television Engineers) 2086 metadata and CTA (Consumer Technology Association) 861.3 metadata to a swapchain.

SMPTE 2086 metadata defines the color volume of the display on which the content was optimized for viewing and includes the color primaries, white point, and luminance range. When such content is reproduced on another display, this metadata can be used by the presentation engine to improve processing of images. For instance, values in the image can first be clamped to the color volume described in the metadata, and then what remains can be remapped to the color volume of the presentation engine.

CTA 861.3 metadata additionally includes the maximum intended luminance for the content and the maximum average light level across frames.

This extension does not define exactly how this metadata is used, however, it simply provides a mechanism to provide it to the presentation engine. Presentation engines may process the image based on the metadata before displaying it, resulting in the image being modified outside of Vulkan. For example, the clamping of colors in the image to the color volume may change those values in the image itself.

The metadata does not override or otherwise influence the color space and color encoding.

### New Commands

- vkSetHdrMetadataEXT

### New Structures

- VkHdrMetadataEXT
- VkXYColorEXT

### New Enum Constants

- `VK_EXT_HDR_METADATA_EXTENSION_NAME`
- `VK_EXT_HDR_METADATA_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_HDR_METADATA_EXT`

### Issues

1) Do we need a query function for the currently specified metadata?

No, Vulkan does not provide queries for state that the application can track on its own.

2) Should we specify default metadata if not specified by the application?

No, the metadata is optional and the absence of the metadata is well-defined.

### Version History

- Revision 1, 2016-12-27 (Courtney Goeltzenleuchter)
  - Initial version
- Revision 2, 2018-12-19 (Courtney Goeltzenleuchter)
  - Correct implicit validity for VkHdrMetadataEXT structure
- Revision 3, 2024-03-26 (Tobias Hector & Sebastian Wick)
  - Clarifications and removal of erroneous "reference monitor" term

## VK_EXT_headless_surface

### Name String

`VK_EXT_headless_surface`

### Extension Type

Instance extension

### Registered Extension Number

257

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_surface

### Contact

- Lisa Wu [GitHub]chengtianww

### Other Extension Metadata

### Last Modified Date

2019-03-21

### IP Status

No known IP claims.

### Contributors

- Ray Smith, Arm

### Description

The `VK_EXT_headless_surface` extension is an instance extension. It provides a mechanism to create VkSurfaceKHR objects independently of any window system or display device. The presentation operation for a swapchain created from a headless surface is by default a no-op, resulting in no externally-visible result.

Because there is no real presentation target, future extensions can layer on top of the headless surface to introduce arbitrary or customizable sets of restrictions or features. These could include features like saving to a file or restrictions to emulate a particular presentation target.

This functionality is expected to be useful for application and driver development because it allows any platform to expose an arbitrary or customizable set of restrictions and features of a presentation engine. This makes it a useful portable test target for applications targeting a wide range of presentation engines where the actual target presentation engines might be scarce, unavailable or otherwise undesirable or inconvenient to use for general Vulkan application development.

### New Commands

- vkCreateHeadlessSurfaceEXT

### New Structures

- VkHeadlessSurfaceCreateInfoEXT

### New Bitmasks

- VkHeadlessSurfaceCreateFlagsEXT

### New Enum Constants

- `VK_EXT_HEADLESS_SURFACE_EXTENSION_NAME`
- `VK_EXT_HEADLESS_SURFACE_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_HEADLESS_SURFACE_CREATE_INFO_EXT`

### Version History

- Revision 1, 2019-03-21 (Ray Smith)
  - Initial draft

## VK_EXT_host_image_copy

### Name String

`VK_EXT_host_image_copy`

### Extension Type

Device extension

### Registered Extension Number

271

### Revision

1

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1
and
VK_KHR_copy_commands2
and
VK_KHR_format_feature_flags2
or
Vulkan Version 1.3

### Contact

- Shahbaz Youssefi [GitHub]syoussefi

### Extension Proposal

VK_EXT_host_image_copy

### Other Extension Metadata

#### Last Modified Date

2023-04-26

#### Contributors

- Shahbaz Youssefi, Google
- Faith Ekstrand, Collabora
- Hans-Kristian Arntzen, Valve
- Piers Daniell, NVIDIA
- Jan-Harald Fredriksen, Arm
- James Fitzpatrick, Imagination
- Daniel Story, Nintendo

#### Description

This extension allows applications to copy data between host memory and images on the host processor, without staging the data through a GPU-accessible buffer. This removes the need to allocate and manage the buffer and its associated memory. On some architectures it may also eliminate an extra copy operation. This extension additionally allows applications to copy data between images on the host.

To support initializing a new image in preparation for a host copy, it is now possible to transition a new image to `VK_IMAGE_LAYOUT_GENERAL` or other host-copyable layouts via vkTransitionImageLayoutEXT. Additionally, it is possible to perform copies that preserve the swizzling layout of the image by using the `VK_HOST_IMAGE_COPY_MEMCPY_EXT` flag. In that case, the memory size needed for copies to or from a buffer can be retrieved by chaining VkSubresourceHostMemcpySizeEXT to `pLayout` in vkGetImageSubresourceLayout2EXT.

#### New Commands

- vkCopyImageToImageEXT
- vkCopyImageToMemoryEXT
- vkCopyMemoryToImageEXT
- vkGetImageSubresourceLayout2EXT
- vkTransitionImageLayoutEXT

#### New Structures

- VkCopyImageToImageInfoEXT
- VkCopyImageToMemoryInfoEXT
- VkCopyMemoryToImageInfoEXT
- VkHostImageLayoutTransitionInfoEXT
- VkImageSubresource2EXT

- VkImageToMemoryCopyEXT
- VkMemoryToImageCopyEXT
- VkSubresourceLayout2EXT
- Extending VkImageFormatProperties2:
  - VkHostImageCopyDevicePerformanceQueryEXT
- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceHostImageCopyFeaturesEXT
- Extending VkPhysicalDeviceProperties2:
  - VkPhysicalDeviceHostImageCopyPropertiesEXT
- Extending VkSubresourceLayout2KHR:
  - VkSubresourceHostMemcpySizeEXT

### New Enums

- VkHostImageCopyFlagBitsEXT

### New Bitmasks

- VkHostImageCopyFlagsEXT

### New Enum Constants

- `VK_EXT_HOST_IMAGE_COPY_EXTENSION_NAME`
- `VK_EXT_HOST_IMAGE_COPY_SPEC_VERSION`
- Extending VkFormatFeatureFlagBits2:
  - `VK_FORMAT_FEATURE_2_HOST_IMAGE_TRANSFER_BIT_EXT`
- Extending VkImageUsageFlagBits:
  - `VK_IMAGE_USAGE_HOST_TRANSFER_BIT_EXT`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_COPY_IMAGE_TO_IMAGE_INFO_EXT`
  - `VK_STRUCTURE_TYPE_COPY_IMAGE_TO_MEMORY_INFO_EXT`
  - `VK_STRUCTURE_TYPE_COPY_MEMORY_TO_IMAGE_INFO_EXT`
  - `VK_STRUCTURE_TYPE_HOST_IMAGE_COPY_DEVICE_PERFORMANCE_QUERY_EXT`
  - `VK_STRUCTURE_TYPE_HOST_IMAGE_LAYOUT_TRANSITION_INFO_EXT`
  - `VK_STRUCTURE_TYPE_IMAGE_TO_MEMORY_COPY_EXT`
  - `VK_STRUCTURE_TYPE_MEMORY_TO_IMAGE_COPY_EXT`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_HOST_IMAGE_COPY_FEATURES_EXT`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_HOST_IMAGE_COPY_PROPERTIES_EXT`
  - `VK_STRUCTURE_TYPE_SUBRESOURCE_HOST_MEMCPY_SIZE_EXT`

### Issues

1) When uploading data to an image, the data is usually loaded from disk. Why not have the application load the data directly into a `VkDeviceMemory` bound to a buffer (instead of host memory), and use vkCmdCopyBufferToImage? The same could be done when downloading data from an image.

**RESOLVED**: This may not always be possible. Complicated Vulkan applications such as game engines often have decoupled subsystems for streaming data and rendering. It may be unreasonable to require the streaming subsystem to coordinate with the rendering subsystem to allocate memory on its behalf, especially as Vulkan may not be the only API supported by the engine. In emulation layers, the image data is necessarily provided by the application in host memory, so an optimization as suggested is not possible. Most importantly, the device memory may not be mappable by an application, but still accessible to the driver.

2) Are `optimalBufferCopyOffsetAlignment` and `optimalBufferCopyRowPitchAlignment` applicable to host memory as well with the functions introduced by this extension? Or should there be new limits?

**RESOLVED**: No alignment requirements for the host memory pointer.

3) Should there be granularity requirements for image offsets and extents?

**RESOLVED**: No granularity requirements, i.e. a granularity of 1 pixel (for non-compressed formats) and 1 texel block (for compressed formats) is assumed.

4) How should the application deal with layout transitions before or after copying to or from images?

**RESOLVED**: An existing issue with linear images is that when emulating other APIs, it is impossible to know when to transition them as they are written to by the host and then used bindlessly. The copy operations in this extension are affected by the same limitation. A new command is thus introduced by this extension to address this problem by allowing the host to perform an image layout transition between a handful of layouts.

### Version History

- Revision 0, 2021-01-20 (Faith Ekstrand)
  - Initial idea and xml
- Revision 1, 2023-04-26 (Shahbaz Youssefi)
  - Initial revision

## VK_EXT_host_query_reset

### Name String

`VK_EXT_host_query_reset`

### Extension Type

Device extension

### Registered Extension Number

262

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### Deprecation State

- *Promoted* to Vulkan 1.2

### Contact

- Bas Nieuwenhuizen [GitHub]BNieuwenhuizen

### Other Extension Metadata

### Last Modified Date

2019-03-06

### IP Status

No known IP claims.

### Contributors

- Bas Nieuwenhuizen, Google
- Faith Ekstrand, Intel
- Jeff Bolz, NVIDIA
- Piers Daniell, NVIDIA

### Description

This extension adds a new function to reset queries from the host.

### Promotion to Vulkan 1.2

All functionality in this extension is included in core Vulkan 1.2, with the EXT suffix omitted. The original type, enum and command names are still available as aliases of the core functionality.

### New Commands

- vkResetQueryPoolEXT

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceHostQueryResetFeaturesEXT

### New Enum Constants

- `VK_EXT_HOST_QUERY_RESET_EXTENSION_NAME`
- `VK_EXT_HOST_QUERY_RESET_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_HOST_QUERY_RESET_FEATURES_EXT`

### Version History

- Revision 1, 2019-03-12 (Bas Nieuwenhuizen)
  - Initial draft

## VK_EXT_image_2d_view_of_3d

### Name String

`VK_EXT_image_2d_view_of_3d`

### Extension Type

Device extension

### Registered Extension Number

394

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_maintenance1
and
VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### Special Use

- OpenGL / ES support

### Contact

- Mike Blumenkrantz [GitHub]zmike

### Other Extension Metadata

#### Last Modified Date

2022-02-22

#### IP Status

No known IP claims.

#### Contributors

- Mike Blumenkrantz, Valve
- Piers Daniell, NVIDIA
- Spencer Fricke, Samsung
- Ricardo Garcia, Igalia
- Graeme Leese, Broadcom
- Ralph Potter, Samsung
- Stu Smith, AMD
- Shahbaz Youssefi, Google
- Alex Walters, Imagination

### Description

This extension allows a single slice of a 3D image to be used as a 2D view in image descriptors, matching both the functionality of glBindImageTexture in OpenGL with the `layer` parameter set to true and 2D view binding provided by the extension EGL_KHR_gl_texture_3D_image. It is primarily intended to support GL emulation.

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceImage2DViewOf3DFeaturesEXT

### New Enum Constants

- `VK_EXT_IMAGE_2D_VIEW_OF_3D_EXTENSION_NAME`
- `VK_EXT_IMAGE_2D_VIEW_OF_3D_SPEC_VERSION`
- Extending VkImageCreateFlagBits:
  - `VK_IMAGE_CREATE_2D_VIEW_COMPATIBLE_BIT_EXT`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_IMAGE_2D_VIEW_OF_3D_FEATURES_EXT`

### Version History

- Revision 1, 2022-03-25 (Mike Blumenkrantz)
  - Internal revisions

## VK_EXT_image_compression_control

### Name String

`VK_EXT_image_compression_control`

### Extension Type

Device extension

### Registered Extension Number

339

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### Contact

- Jan-Harald Fredriksen [GitHub]janharaldfredriksen-arm

### Extension Proposal

VK_EXT_image_compression_control

### Other Extension Metadata

### Last Modified Date

2022-05-02

### IP Status

No known IP claims.

### Contributors

- Jan-Harald Fredriksen, Arm
- Graeme Leese, Broadcom
- Andrew Garrard, Imagination
- Lisa Wu, Arm
- Peter Kohaut, Arm

### Description

This extension enables fixed-rate image compression and adds the ability to control when this kind of compression can be applied. Many implementations support some form of framebuffer

compression. This is typically transparent to applications as lossless compression schemes are used. With fixed-rate compression, the compression is done at a defined bitrate. Such compression algorithms generally produce results that are visually lossless, but the results are typically not bit-exact when compared to a non-compressed result. The implementation may not be able to use the requested compression rate in all cases. This extension adds a query that can be used to determine the compression scheme and rate that was applied to an image.

### New Commands

- vkGetImageSubresourceLayout2EXT

### New Structures

- VkImageSubresource2EXT
- VkSubresourceLayout2EXT
- Extending VkImageCreateInfo, VkSwapchainCreateInfoKHR, VkPhysicalDeviceImageFormatInfo2:
  - VkImageCompressionControlEXT
- Extending VkImageFormatProperties2, VkSurfaceFormat2KHR, VkSubresourceLayout2KHR:
  - VkImageCompressionPropertiesEXT
- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceImageCompressionControlFeaturesEXT

### New Enums

- VkImageCompressionFixedRateFlagBitsEXT
- VkImageCompressionFlagBitsEXT

### New Bitmasks

- VkImageCompressionFixedRateFlagsEXT
- VkImageCompressionFlagsEXT

### New Enum Constants

- `VK_EXT_IMAGE_COMPRESSION_CONTROL_EXTENSION_NAME`
- `VK_EXT_IMAGE_COMPRESSION_CONTROL_SPEC_VERSION`
- Extending VkResult:
  - `VK_ERROR_COMPRESSION_EXHAUSTED_EXT`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_IMAGE_COMPRESSION_CONTROL_EXT`
  - `VK_STRUCTURE_TYPE_IMAGE_COMPRESSION_PROPERTIES_EXT`
  - `VK_STRUCTURE_TYPE_IMAGE_SUBRESOURCE_2_EXT`

- `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_IMAGE_COMPRESSION_CONTROL_FEATURES_EXT`
- `VK_STRUCTURE_TYPE_SUBRESOURCE_LAYOUT_2_EXT`

### Version History

- Revision 1, 2022-05-02 (Jan-Harald Fredriksen)
  - Initial draft

## VK_EXT_image_compression_control_swapchain

### Name String

`VK_EXT_image_compression_control_swapchain`

### Extension Type

Device extension

### Registered Extension Number

438

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_EXT_image_compression_control

### Contact

- Jan-Harald Fredriksen [GitHub]janharaldfredriksen-arm

### Other Extension Metadata

### Last Modified Date

2022-05-02

### IP Status

No known IP claims.

### Contributors

- Jan-Harald Fredriksen, Arm
- Graeme Leese, Broadcom
- Andrew Garrard, Imagination
- Lisa Wu, Arm
- Peter Kohaut, Arm
- Ian Elliott, Google

### Description

This extension enables fixed-rate image compression and adds the ability to control when this kind of compression can be applied to swapchain images.

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceImageCompressionControlSwapchainFeaturesEXT

### New Enum Constants

- `VK_EXT_IMAGE_COMPRESSION_CONTROL_SWAPCHAIN_EXTENSION_NAME`
- `VK_EXT_IMAGE_COMPRESSION_CONTROL_SWAPCHAIN_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_IMAGE_COMPRESSION_CONTROL_SWAPCHAIN_FEATURES_EXT`

### Version History

- Revision 1, 2022-05-02 (Jan-Harald Fredriksen)
  - Initial draft

## VK_EXT_image_drm_format_modifier

### Name String

`VK_EXT_image_drm_format_modifier`

### Extension Type

Device extension

### Registered Extension Number

159

### Revision

2

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_bind_memory2
and
VK_KHR_get_physical_device_properties2
and
VK_KHR_sampler_ycbcr_conversion
or
Vulkan Version 1.1

and
VK_KHR_image_format_list
or
Vulkan Version 1.2

### API Interactions

- Interacts with VK_VERSION_1_3
- Interacts with VK_KHR_format_feature_flags2

### Contact

- Lina Versace [GitHub]versalinyaa

### Other Extension Metadata

### Last Modified Date

2021-09-30

### IP Status

No known IP claims.

### Contributors

- Antoine Labour, Google
- Bas Nieuwenhuizen, Google
- Lina Versace, Google
- James Jones, NVIDIA
- Faith Ekstrand, Intel
- Jőrg Wagner, ARM
- Kristian Høgsberg Kristensen, Google
- Ray Smith, ARM

### Description

This extension provides the ability to use *DRM format modifiers* with images, enabling Vulkan to better integrate with the Linux ecosystem of graphics, video, and display APIs.

Its functionality closely overlaps with `EGL_EXT_image_dma_buf_import_modifiers`[2] and `EGL_MESA_image_dma_buf_export`[3]. Unlike the EGL extensions, this extension does not require the use of a specific handle type (such as a dma_buf) for external memory and provides more explicit control of image creation.

### Introduction to DRM Format Modifiers

A *DRM format modifier* is a 64-bit, vendor-prefixed, semi-opaque unsigned integer. Most *modifiers* represent a concrete, vendor-specific tiling format for images. Some exceptions are `DRM_FORMAT_MOD_LINEAR` (which is not vendor-specific); `DRM_FORMAT_MOD_NONE` (which is an alias of `DRM_FORMAT_MOD_LINEAR` due to historical accident); and `DRM_FORMAT_MOD_INVALID` (which does not

represent a tiling format). The *modifier's* vendor prefix consists of the 8 most significant bits. The canonical list of *modifiers* and vendor prefixes is found in `drm_fourcc.h` in the Linux kernel source. The other dominant source of *modifiers* are vendor kernel trees.

One goal of *modifiers* in the Linux ecosystem is to enumerate for each vendor a reasonably sized set of tiling formats that are appropriate for images shared across processes, APIs, and/or devices, where each participating component may possibly be from different vendors. A non-goal is to enumerate all tiling formats supported by all vendors. Some tiling formats used internally by vendors are inappropriate for sharing; no *modifiers* should be assigned to such tiling formats.

Modifier values typically do not *describe* memory layouts. More precisely, a *modifier*'s lower 56 bits usually have no structure. Instead, modifiers *name* memory layouts; they name a small set of vendor-preferred layouts for image sharing. As a consequence, in each vendor namespace the modifier values are often sequentially allocated starting at 1.

Each *modifier* is usually supported by a single vendor and its name matches the pattern `{VENDOR}_FORMAT_MOD_*` or `DRM_FORMAT_MOD_{VENDOR}_*`. Examples are `I915_FORMAT_MOD_X_TILED` and `DRM_FORMAT_MOD_BROADCOM_VC4_T_TILED`. An exception is `DRM_FORMAT_MOD_LINEAR`, which is supported by most vendors.

Many APIs in Linux use *modifiers* to negotiate and specify the memory layout of shared images. For example, a Wayland compositor and Wayland client may, by relaying *modifiers* over the Wayland protocol `zwp_linux_dmabuf_v1`, negotiate a vendor-specific tiling format for a shared `wl_buffer`. The client may allocate the underlying memory for the `wl_buffer` with GBM, providing the chosen *modifier* to `gbm_bo_create_with_modifiers`. The client may then import the `wl_buffer` into Vulkan for producing image content, providing the resource's dma_buf to VkImportMemoryFdInfoKHR and its *modifier* to VkImageDrmFormatModifierExplicitCreateInfoEXT. The compositor may then import the `wl_buffer` into OpenGL for sampling, providing the resource's dma_buf and *modifier* to `eglCreateImage`. The compositor may also bypass OpenGL and submit the `wl_buffer` directly to the kernel's display API, providing the dma_buf and *modifier* through `drm_mode_fb_cmd2`.

#### Format Translation

*Modifier*-capable APIs often pair *modifiers* with DRM formats, which are defined in `drm_fourcc.h`. However, `VK_EXT_image_drm_format_modifier` uses VkFormat instead of DRM formats. The application must convert between VkFormat and DRM format when it sends or receives a DRM format to or from an external API.

The mapping from VkFormat to DRM format is lossy. Therefore, when receiving a DRM format from an external API, often the application must use information from the external API to accurately map the DRM format to a VkFormat. For example, DRM formats do not distinguish between RGB and sRGB (as of 2018-03-28); external information is required to identify the image's color space.

The mapping between VkFormat and DRM format is also incomplete. For some DRM formats there exist no corresponding Vulkan format, and for some Vulkan formats there exist no corresponding DRM format.

#### Usage Patterns

Three primary usage patterns are intended for this extension:

- **Negotiation.** The application negotiates with *modifier*-aware, external components to determine sets of image creation parameters supported among all components.

  In the Linux ecosystem, the negotiation usually assumes the image is a 2D, single-sampled, non-mipmapped, non-array image; this extension permits that assumption but does not require it. The result of the negotiation usually resembles a set of tuples such as *(drmFormat, drmFormatModifier)*, where each participating component supports all tuples in the set.

  Many details of this negotiation - such as the protocol used during negotiation, the set of image creation parameters expressible in the protocol, and how the protocol chooses which process and which API will create the image - are outside the scope of this specification.

  In this extension, vkGetPhysicalDeviceFormatProperties2 with VkDrmFormatModifierPropertiesListEXT serves a primary role during the negotiation, and vkGetPhysicalDeviceImageFormatProperties2 with VkPhysicalDeviceImageDrmFormatModifierInfoEXT serves a secondary role.

- **Import.** The application imports an image with a *modifier*.

  In this pattern, the application receives from an external source the image's memory and its creation parameters, which are often the result of the negotiation described above. Some image creation parameters are implicitly defined by the external source; for example, `VK_IMAGE_TYPE_2D` is often assumed. Some image creation parameters are usually explicit, such as the image's `format`, `drmFormatModifier`, and `extent`; and each plane's `offset` and `rowPitch`.

  Before creating the image, the application first verifies that the physical device supports the received creation parameters by querying vkGetPhysicalDeviceFormatProperties2 with VkDrmFormatModifierPropertiesListEXT and vkGetPhysicalDeviceImageFormatProperties2 with VkPhysicalDeviceImageDrmFormatModifierInfoEXT. Then the application creates the image by chaining VkImageDrmFormatModifierExplicitCreateInfoEXT and VkExternalMemoryImageCreateInfo onto VkImageCreateInfo.

- **Export.** The application creates an image and allocates its memory. Then the application exports to *modifier*-aware consumers the image's memory handles; its creation parameters; its *modifier*; and the `offset`, `size`, and `rowPitch` of each *memory plane*.

  In this pattern, the Vulkan device is the authority for the image; it is the allocator of the image's memory and the decider of the image's creation parameters. When choosing the image's creation parameters, the application usually chooses a tuple *(format, drmFormatModifier)* from the result of the negotiation described above. The negotiation's result often contains multiple tuples that share the same format but differ in their *modifier*. In this case, the application should defer the choice of the image's *modifier* to the Vulkan implementation by providing all such *modifiers* to VkImageDrmFormatModifierListCreateInfoEXT::`pDrmFormatModifiers`; and the implementation should choose from `pDrmFormatModifiers` the optimal *modifier* in consideration with the other image parameters.

  The application creates the image by chaining VkImageDrmFormatModifierListCreateInfoEXT and VkExternalMemoryImageCreateInfo onto VkImageCreateInfo. The protocol and APIs by which the application will share the image with external consumers will likely determine the value of VkExternalMemoryImageCreateInfo::`handleTypes`. The implementation chooses for the

image an optimal *modifier* from VkImageDrmFormatModifierListCreateInfoEXT::pDrmFormatModifiers. The application then queries the implementation-chosen *modifier* with vkGetImageDrmFormatModifierPropertiesEXT, and queries the memory layout of each plane with vkGetImageSubresourceLayout.

The application then allocates the image's memory with VkMemoryAllocateInfo, adding chained extending structures for external memory; binds it to the image; and exports the memory, for example, with vkGetMemoryFdKHR.

Finally, the application sends the image's creation parameters, its *modifier*, its per-plane memory layout, and the exported memory handle to the external consumers. The details of how the application transmits this information to external consumers is outside the scope of this specification.

**Prior Art**

Extension `EGL_EXT_image_dma_buf_import`[1] introduced the ability to create an `EGLImage` by importing for each plane a dma_buf, offset, and row pitch.

Later, extension `EGL_EXT_image_dma_buf_import_modifiers`[2] introduced the ability to query which combination of formats and *modifiers* the implementation supports and to specify *modifiers* during creation of the `EGLImage`.

Extension `EGL_MESA_image_dma_buf_export`[3] is the inverse of `EGL_EXT_image_dma_buf_import_modifiers`.

The Linux kernel modesetting API (KMS), when configuring the display's framebuffer with `struct drm_mode_fb_cmd2`[4], allows one to specify the framebuffer's *modifier* as well as a per-plane memory handle, offset, and row pitch.

GBM, a graphics buffer manager for Linux, allows creation of a `gbm_bo` (that is, a graphics *buffer object*) by importing data similar to that in `EGL_EXT_image_dma_buf_import_modifiers`[1]; and symmetrically allows exporting the same data from the `gbm_bo`. See the references to *modifier* and *plane* in `gbm.h`[5].

**New Commands**

- vkGetImageDrmFormatModifierPropertiesEXT

**New Structures**

- VkDrmFormatModifierPropertiesEXT
- VkImageDrmFormatModifierPropertiesEXT
- Extending VkFormatProperties2:
  - VkDrmFormatModifierPropertiesListEXT
- Extending VkImageCreateInfo:
  - VkImageDrmFormatModifierExplicitCreateInfoEXT
  - VkImageDrmFormatModifierListCreateInfoEXT

- Extending VkPhysicalDeviceImageFormatInfo2:
  - VkPhysicalDeviceImageDrmFormatModifierInfoEXT

If VK_KHR_format_feature_flags2 or Vulkan Version 1.3 is supported:

- VkDrmFormatModifierProperties2EXT
- Extending VkFormatProperties2:
  - VkDrmFormatModifierPropertiesList2EXT

**New Enum Constants**

- `VK_EXT_IMAGE_DRM_FORMAT_MODIFIER_EXTENSION_NAME`
- `VK_EXT_IMAGE_DRM_FORMAT_MODIFIER_SPEC_VERSION`
- Extending VkImageAspectFlagBits:
  - `VK_IMAGE_ASPECT_MEMORY_PLANE_0_BIT_EXT`
  - `VK_IMAGE_ASPECT_MEMORY_PLANE_1_BIT_EXT`
  - `VK_IMAGE_ASPECT_MEMORY_PLANE_2_BIT_EXT`
  - `VK_IMAGE_ASPECT_MEMORY_PLANE_3_BIT_EXT`
- Extending VkImageTiling:
  - `VK_IMAGE_TILING_DRM_FORMAT_MODIFIER_EXT`
- Extending VkResult:
  - `VK_ERROR_INVALID_DRM_FORMAT_MODIFIER_PLANE_LAYOUT_EXT`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_DRM_FORMAT_MODIFIER_PROPERTIES_LIST_EXT`
  - `VK_STRUCTURE_TYPE_IMAGE_DRM_FORMAT_MODIFIER_EXPLICIT_CREATE_INFO_EXT`
  - `VK_STRUCTURE_TYPE_IMAGE_DRM_FORMAT_MODIFIER_LIST_CREATE_INFO_EXT`
  - `VK_STRUCTURE_TYPE_IMAGE_DRM_FORMAT_MODIFIER_PROPERTIES_EXT`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_IMAGE_DRM_FORMAT_MODIFIER_INFO_EXT`

If VK_KHR_format_feature_flags2 or Vulkan Version 1.3 is supported:

- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_DRM_FORMAT_MODIFIER_PROPERTIES_LIST_2_EXT`

**Issues**

1) Should this extension define a single DRM format modifier per `VkImage`? Or define one per plane?

+

**RESOLVED**: There exists a single DRM format modifier per `VkImage`.

**DISCUSSION**: Prior art, such as `EGL_EXT_image_dma_buf_import_modifiers`[2], `struct drm_mode_fb_cmd2`[4], and `struct gbm_import_fd_modifier_data`[5], allows defining one *modifier* per plane. However, developers of the GBM and kernel APIs concede it was a mistake. Beginning in Linux 4.10, the kernel requires that the application provide the same DRM format *modifier* for each plane. (See Linux commit bae781b259269590109e8a4a8227331362b88212). And GBM provides an entry point, `gbm_bo_get_modifier`, for querying the *modifier* of the image but does not provide one to query the modifier of individual planes.

2) When creating an image with VkImageDrmFormatModifierExplicitCreateInfoEXT, which is typically used when *importing* an image, should the application explicitly provide the size of each plane?

+

**RESOLVED**: No. The application **must** not provide the size. To enforce this, the API requires that VkImageDrmFormatModifierExplicitCreateInfoEXT::`pPlaneLayouts->size` **must** be 0.

**DISCUSSION**: Prior art, such as `EGL_EXT_image_dma_buf_import_modifiers`[2], `struct drm_mode_fb_cmd2`[4], and `struct gbm_import_fd_modifier_data`[5], omits from the API the size of each plane. Instead, the APIs infer each plane's size from the import parameters, which include the image's pixel format and a dma_buf, offset, and row pitch for each plane.

However, Vulkan differs from EGL and GBM with regards to image creation in the following ways:

*Differences in Image Creation*

- **Undedicated allocation by default.** When importing or exporting a set of dma_bufs as an `EGLImage` or `gbm_bo`, common practice mandates that each dma_buf's memory be dedicated (in the sense of `VK_KHR_dedicated_allocation`) to the image (though not necessarily dedicated to a single plane). In particular, neither the GBM documentation nor the EGL extension specifications explicitly state this requirement, but in light of common practice this is likely due to under-specification rather than intentional omission. In contrast, `VK_EXT_image_drm_format_modifier` permits, but does not require, the implementation to require dedicated allocations for images created with `VK_IMAGE_TILING_DRM_FORMAT_MODIFIER_EXT`.
- **Separation of image creation and memory allocation.** When importing a set of dma_bufs as an `EGLImage` or `gbm_bo`, EGL and GBM create the image resource and bind it to memory (the dma_bufs) simultaneously. This allows EGL and GBM to query each dma_buf's size during image creation. In Vulkan, image creation and memory allocation are independent unless a dedicated allocation is used (as in `VK_KHR_dedicated_allocation`). Therefore, without requiring dedicated allocation, Vulkan cannot query the size of each dma_buf (or other external handle) when calculating the image's memory layout. Even if dedication allocation were required, Vulkan cannot calculate the image's memory layout until after the image is bound to its dma_ufs.

The above differences complicate the potential inference of plane size in Vulkan. Consider the following problematic cases:

*Problematic Plane Size Calculations*

- **Padding.** Some plane of the image may require implementation-dependent padding.

- **Metadata.** For some *modifiers*, the image may have a metadata plane which requires a non-trivial calculation to determine its size.
- **Mipmapped, array, and 3D images.** The implementation may support `VK_IMAGE_TILING_DRM_FORMAT_MODIFIER_EXT` for images whose `mipLevels`, `arrayLayers`, or `depth` is greater than 1. For such images with certain *modifiers*, the calculation of each plane's size may be non-trivial.

However, an application-provided plane size solves none of the above problems.

For simplicity, consider an external image with a single memory plane. The implementation is obviously capable calculating the image's size when its tiling is `VK_IMAGE_TILING_OPTIMAL`. Likewise, any reasonable implementation is capable of calculating the image's size when its tiling uses a supported *modifier*.

Suppose that the external image's size is smaller than the implementation-calculated size. If the application provided the external image's size to vkCreateImage, the implementation would observe the mismatched size and recognize its inability to comprehend the external image's layout (unless the implementation used the application-provided size to select a refinement of the tiling layout indicated by the *modifier*, which is strongly discouraged). The implementation would observe the conflict, and reject image creation with `VK_ERROR_INVALID_DRM_FORMAT_MODIFIER_PLANE_LAYOUT_EXT`. On the other hand, if the application did not provide the external image's size to vkCreateImage, then the application would observe after calling vkGetImageMemoryRequirements that the external image's size is less than the size required by the implementation. The application would observe the conflict and refuse to bind the `VkImage` to the external memory. In both cases, the result is explicit failure.

Suppose that the external image's size is larger than the implementation-calculated size. If the application provided the external image's size to vkCreateImage, for reasons similar to above the implementation would observe the mismatched size and recognize its inability to comprehend the image data residing in the extra size. The implementation, however, must assume that image data resides in the entire size provided by the application. The implementation would observe the conflict and reject image creation with `VK_ERROR_INVALID_DRM_FORMAT_MODIFIER_PLANE_LAYOUT_EXT`. On the other hand, if the application did not provide the external image's size to vkCreateImage, then the application would observe after calling vkGetImageMemoryRequirements that the external image's size is larger than the implementation-usable size. The application would observe the conflict and refuse to bind the `VkImage` to the external memory. In both cases, the result is explicit failure.

Therefore, an application-provided size provides no benefit, and this extension should not require it. This decision renders VkSubresourceLayout::`size` an unused field during image creation, and thus introduces a risk that implementations may require applications to submit sideband creation parameters in the unused field. To prevent implementations from relying on sideband data, this extension *requires* the application to set `size` to 0.

**References**

1. `EGL_EXT_image_dma_buf_import`
2. `EGL_EXT_image_dma_buf_import_modifiers`

3. `EGL_MESA_image_dma_buf_export`
4. `struct drm_mode_fb_cmd2`
5. `gbm.h`

#### Version History

- Revision 1, 2018-08-29 (Lina Versace)
  - First stable revision
- Revision 2, 2021-09-30 (Jon Leech)
  - Add interaction with `VK_KHR_format_feature_flags2` to `vk.xml`

## VK_EXT_image_robustness

### Name String

`VK_EXT_image_robustness`

### Extension Type

Device extension

### Registered Extension Number

336

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### Deprecation State

- *Promoted* to Vulkan 1.3

### Contact

- Graeme Leese [GitHub]gnl21

### Other Extension Metadata

### Last Modified Date

2020-04-27

### IP Status

No known IP claims.

### Contributors

- Graeme Leese, Broadcom
- Jan-Harald Fredriksen, ARM
- Jeff Bolz, NVIDIA
- Spencer Fricke, Samsung
- Courtney Goeltzenleuchter, Google
- Slawomir Cygan, Intel

### Description

This extension adds stricter requirements for how out of bounds reads from images are handled. Rather than returning undefined values, most out of bounds reads return R, G, and B values of zero and alpha values of either zero or one. Components not present in the image format may be set to zero or to values based on the format as described in Conversion to RGBA.

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceImageRobustnessFeaturesEXT

### New Enum Constants

- `VK_EXT_IMAGE_ROBUSTNESS_EXTENSION_NAME`
- `VK_EXT_IMAGE_ROBUSTNESS_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_IMAGE_ROBUSTNESS_FEATURES_EXT`

### Promotion to Vulkan 1.3

Functionality in this extension is included in core Vulkan 1.3, with the EXT suffix omitted. The original type, enum and command names are still available as aliases of the core functionality.

### Issues

1. How does this extension differ from VK_EXT_robustness2?

The guarantees provided by this extension are a subset of those provided by the robustImageAccess2 feature of VK_EXT_robustness2. Where this extension allows return values of (0, 0, 0, 0) or (0, 0, 0, 1), robustImageAccess2 requires that a particular value dependent on the image format be returned. This extension provides no guarantees about the values returned for an access to an invalid Lod.

### Examples

None.

### Version History

- Revision 1, 2020-04-27 (Graeme Leese)
- Initial draft

## VK_EXT_image_sliced_view_of_3d

### Name String

`VK_EXT_image_sliced_view_of_3d`

### Extension Type

Device extension

### Registered Extension Number

419

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_maintenance1
and
VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### Special Use

- D3D support

### Contact

- Mike Blumenkrantz [GitHub]zmike

### Extension Proposal

VK_EXT_image_sliced_view_of_3d

### Other Extension Metadata

### Last Modified Date

2023-01-24

### IP Status

No known IP claims.

### Contributors

- Mike Blumenkrantz, Valve
- Hans-Kristian Arntzen, Valve

- Ricardo Garcia, Igalia
- Shahbaz Youssefi, Google
- Piers Daniell, NVIDIA

### Description

This extension allows creating 3D views of 3D images such that the views contain a subset of the slices in the image, using a Z offset and range, for the purpose of using the views as storage image descriptors. This matches functionality in D3D12 and is primarily intended to support D3D12 emulation.

### New Structures

- Extending VkImageViewCreateInfo:
  - VkImageViewSlicedCreateInfoEXT
- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceImageSlicedViewOf3DFeaturesEXT

### New Enum Constants

- `VK_EXT_IMAGE_SLICED_VIEW_OF_3D_EXTENSION_NAME`
- `VK_EXT_IMAGE_SLICED_VIEW_OF_3D_SPEC_VERSION`
- `VK_REMAINING_3D_SLICES_EXT`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_IMAGE_VIEW_SLICED_CREATE_INFO_EXT`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_IMAGE_SLICED_VIEW_OF_3D_FEATURES_EXT`

### Version History

- Revision 1, 2022-10-21 (Mike Blumenkrantz)
  - Initial revision

## VK_EXT_image_view_min_lod

### Name String

`VK_EXT_image_view_min_lod`

### Extension Type

Device extension

### Registered Extension Number

392

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### Contact

- Joshua Ashton [GitHub]Joshua-Ashton

### Other Extension Metadata

### Last Modified Date

2021-11-09

### IP Status

No known IP claims.

### Contributors

- Joshua Ashton, Valve
- Hans-Kristian Arntzen, Valve
- Samuel Iglesias Gonsalvez, Igalia
- Tobias Hector, AMD
- Faith Ekstrand, Intel
- Tom Olson, ARM

### Description

This extension allows applications to clamp the minimum LOD value during Image Level(s) Selection, Texel Gathering and Integer Texel Coordinate Operations with a given VkImageView by VkImageViewMinLodCreateInfoEXT::`minLod`.

This extension may be useful to restrict a VkImageView to only mips which have been uploaded, and the use of fractional `minLod` can be useful for smoothly introducing new mip levels when using linear mipmap filtering.

### New Structures

- Extending VkImageViewCreateInfo:
  - VkImageViewMinLodCreateInfoEXT
- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceImageViewMinLodFeaturesEXT

### New Enum Constants

- `VK_EXT_IMAGE_VIEW_MIN_LOD_EXTENSION_NAME`
- `VK_EXT_IMAGE_VIEW_MIN_LOD_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_IMAGE_VIEW_MIN_LOD_CREATE_INFO_EXT`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_IMAGE_VIEW_MIN_LOD_FEATURES_EXT`

### Version History

- Revision 1, 2021-07-06 (Joshua Ashton)
  - Initial version

## VK_EXT_inline_uniform_block

### Name String

`VK_EXT_inline_uniform_block`

### Extension Type

Device extension

### Registered Extension Number

139

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
and
VK_KHR_maintenance1
or
Vulkan Version 1.1

### API Interactions

- Interacts with VK_EXT_descriptor_indexing

### Deprecation State

- *Promoted* to Vulkan 1.3

### Contact

- Daniel Rakos [GitHub]aqnuep

### Other Extension Metadata

#### Last Modified Date

2018-08-01

#### IP Status

No known IP claims.

#### Contributors

- Daniel Rakos, AMD
- Jeff Bolz, NVIDIA
- Slawomir Grajewski, Intel
- Neil Henning, Codeplay

### Description

This extension introduces the ability to back uniform blocks directly with descriptor sets by storing inline uniform data within descriptor pool storage. Compared to push constants this new construct allows uniform data to be reused across multiple disjoint sets of drawing or dispatching commands and **may** enable uniform data to be accessed with fewer indirections compared to uniforms backed by buffer memory.

### New Structures

- Extending VkDescriptorPoolCreateInfo:
  - VkDescriptorPoolInlineUniformBlockCreateInfoEXT
- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceInlineUniformBlockFeaturesEXT
- Extending VkPhysicalDeviceProperties2:
  - VkPhysicalDeviceInlineUniformBlockPropertiesEXT
- Extending VkWriteDescriptorSet:
  - VkWriteDescriptorSetInlineUniformBlockEXT

### New Enum Constants

- `VK_EXT_INLINE_UNIFORM_BLOCK_EXTENSION_NAME`
- `VK_EXT_INLINE_UNIFORM_BLOCK_SPEC_VERSION`
- Extending VkDescriptorType:
  - `VK_DESCRIPTOR_TYPE_INLINE_UNIFORM_BLOCK_EXT`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_DESCRIPTOR_POOL_INLINE_UNIFORM_BLOCK_CREATE_INFO_EXT`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_INLINE_UNIFORM_BLOCK_FEATURES_EXT`

- `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_INLINE_UNIFORM_BLOCK_PROPERTIES_EXT`
- `VK_STRUCTURE_TYPE_WRITE_DESCRIPTOR_SET_INLINE_UNIFORM_BLOCK_EXT`

**Promotion to Vulkan 1.3**

Functionality in this extension is included in core Vulkan 1.3, with the EXT suffix omitted. The original type, enum and command names are still available as aliases of the core functionality.

Vulkan 1.3 adds additional functionality related to this extension in the form of the `maxInlineUniformTotalSize` limit.

**Issues**

1) Do we need a new storage class for inline uniform blocks vs. uniform blocks?

**RESOLVED**: No. The `Uniform` storage class is used to allow the same syntax used for both uniform buffers and inline uniform blocks.

2) Is the descriptor array index and array size expressed in terms of bytes or dwords for inline uniform block descriptors?

**RESOLVED**: In bytes, but both **must** be a multiple of 4, similar to how push constant ranges are specified. The `descriptorCount` of `VkDescriptorSetLayoutBinding` thus provides the total number of bytes a particular binding with an inline uniform block descriptor type can hold, while the `srcArrayElement`, `dstArrayElement`, and `descriptorCount` members of `VkWriteDescriptorSet`, `VkCopyDescriptorSet`, and `VkDescriptorUpdateTemplateEntry` (where applicable) specify the byte offset and number of bytes to write/copy to the binding's backing store. Additionally, the `stride` member of `VkDescriptorUpdateTemplateEntry` is ignored for inline uniform blocks and a default value of one is used, meaning that the data to update inline uniform block bindings with must be contiguous in memory.

3) What layout rules apply for uniform blocks corresponding to inline constants?

**RESOLVED**: They use the same layout rules as uniform buffers.

4) Do we need to add non-uniform indexing features/properties as introduced by `VK_EXT_descriptor_indexing` for inline uniform blocks?

**RESOLVED**: No, because inline uniform blocks are not allowed to be "arrayed". A single binding with an inline uniform block descriptor type corresponds to a single uniform block instance and the array indices inside that binding refer to individual offsets within the uniform block (see issue #2). However, this extension does introduce new features/properties about the level of support for update-after-bind inline uniform blocks.

5) Is the `descriptorBindingVariableDescriptorCount` feature introduced by `VK_EXT_descriptor_indexing` supported for inline uniform blocks?

**RESOLVED**: Yes, as long as other inline uniform block specific limits are respected.

6) Do the robustness guarantees of `robustBufferAccess` apply to inline uniform block accesses?

**RESOLVED**: No, similarly to push constants, as they are not backed by buffer memory like uniform buffers.

### Version History

- Revision 1, 2018-08-01 (Daniel Rakos)
  - Internal revisions

## VK_EXT_layer_settings

### Name String

`VK_EXT_layer_settings`

### Extension Type

Instance extension

### Registered Extension Number

497

### Revision

2

### Ratification Status

Ratified

### Extension and Version Dependencies

None

### Contact

- Christophe Riccio [GitHub]christophe

### Extension Proposal

VK_EXT_layer_settings

### Other Extension Metadata

### Last Modified Date

2023-09-23

### IP Status

No known IP claims.

### Contributors

- Christophe Riccio, LunarG
- Mark Lobodzinski, LunarG
- Charles Giessen, LunarG
- Spencer Fricke, LunarG

- Juan Ramos, LunarG
- Daniel Rakos, RasterGrid
- Shahbaz Youssefi, Google
- Lina Versace, Google
- Bill Hollings, The Brenwill Workshop
- Jon Leech, Khronos
- Tom Olson, Arm

### Description

This extension provides a mechanism for configuring programmatically through the Vulkan API the behavior of layers.

This extension provides the VkLayerSettingsCreateInfoEXT struct that can be included in the `pNext` chain of the VkInstanceCreateInfo structure passed as the `pCreateInfo` parameter of vkCreateInstance.

The structure contains an array of VkLayerSettingEXT structure values that configure specific features of layers.

| | |
|---|---|
| **NOTE** | The `VK_EXT_layer_settings` extension subsumes all the functionality provided in the `VK_EXT_validation_flags` extension and the `VK_EXT_validation_features` extension. |

### New Structures

- VkLayerSettingEXT
- Extending VkInstanceCreateInfo:
  - VkLayerSettingsCreateInfoEXT

### New Enums

- VkLayerSettingTypeEXT

### New Enum Constants

- `VK_EXT_LAYER_SETTINGS_EXTENSION_NAME`
- `VK_EXT_LAYER_SETTINGS_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_LAYER_SETTINGS_CREATE_INFO_EXT`

### Example

One example usage of `VK_EXT_layer_settings` is as implemented by the Vulkan Profiles layer.

It allows the profiles layer tests used by the profiles layer C.I. to programmatically configure the layer for each test without affecting the C.I. environment, allowing to run multiple tests

concurrently.

```
const char* profile_file_data = JSON_TEST_FILES_PATH "VP_KHR_roadmap_2022.json";
const char* profile_name_data = "VP_KHR_roadmap_2022";
VkBool32 emulate_portability_data = VK_TRUE;
const char* simulate_capabilities[] = {
    "SIMULATE_API_VERSION_BIT",
    "SIMULATE_FEATURES_BIT",
    "SIMULATE_PROPERTIES_BIT",
    "SIMULATE_EXTENSIONS_BIT",
    "SIMULATE_FORMATS_BIT",
    "SIMULATE_QUEUE_FAMILY_PROPERTIES_BIT"
};
const char* debug_reports[] = {
    "DEBUG_REPORT_ERROR_BIT",
    "DEBUG_REPORT_WARNING_BIT",
    "DEBUG_REPORT_NOTIFICATION_BIT",
    "DEBUG_REPORT_DEBUG_BIT"
};

const VkLayerSettingEXT settings[] = {
     {kLayerName, kLayerSettingsProfileFile, VK_LAYER_SETTING_TYPE_STRING_EXT, 1,
&profile_file_data},
     {kLayerName, kLayerSettingsProfileName, VK_LAYER_SETTING_TYPE_STRING_EXT, 1,
&profile_name_data},
     {kLayerName, kLayerSettingsEmulatePortability, VK_LAYER_SETTING_TYPE_BOOL32_EXT,
1, &emulate_portability_data},
     {kLayerName, kLayerSettingsSimulateCapabilities,
VK_LAYER_SETTING_TYPE_STRING_EXT,
        static_cast<uint32_t>(std::size(simulate_capabilities)),
simulate_capabilities},
     {kLayerName, kLayerSettingsDebugReports, VK_LAYER_SETTING_TYPE_STRING_EXT,
        static_cast<uint32_t>(std::size(debug_reports)), debug_reports}
};

const VkLayerSettingsCreateInfoEXT layer_settings_create_info{
    VK_STRUCTURE_TYPE_LAYER_SETTINGS_CREATE_INFO_EXT, nullptr,
    static_cast<uint32_t>(std::size(settings)), settings};

VkInstanceCreateInfo inst_create_info = {};
...
inst_create_info.pNext = &layer_settings_create_info;
vkCreateInstance(&inst_create_info, nullptr, &_instances);
```

**Issues**

- How should application developers figure out the list of available settings?

This extension does not provide a reflection API for layer settings. Layer settings are described in each layer JSON manifest and the documentation of each layer which implements this extension.

### Version History

- Revision 1, 2020-06-17 (Mark Lobodzinski)
  - Initial revision for Validation layer internal usages
- Revision 2, 2023-09-26 (Christophe Riccio)
  - Refactor APIs for any layer usages and public release

## VK_EXT_legacy_dithering

### Name String

`VK_EXT_legacy_dithering`

### Extension Type

Device extension

### Registered Extension Number

466

### Revision

2

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### API Interactions

- Interacts with VK_VERSION_1_3
- Interacts with VK_KHR_dynamic_rendering
- Interacts with VK_KHR_maintenance5

### Special Use

- OpenGL / ES support

### Contact

- Shahbaz Youssefi [GitHub]syoussefi

### Extension Proposal

VK_EXT_legacy_dithering

### Other Extension Metadata

### Last Modified Date

2024-02-22

### Contributors

- Shahbaz Youssefi, Google
- Graeme Leese, Broadcom
- Jan-Harald Fredriksen, Arm

### Description

This extension exposes a hardware feature used by some vendors to implement OpenGL's dithering. The purpose of this extension is to support layering OpenGL over Vulkan, by allowing the layer to take advantage of the same hardware feature and provide equivalent dithering to OpenGL applications.

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceLegacyDitheringFeaturesEXT

### New Enum Constants

- `VK_EXT_LEGACY_DITHERING_EXTENSION_NAME`
- `VK_EXT_LEGACY_DITHERING_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_LEGACY_DITHERING_FEATURES_EXT`
- Extending VkSubpassDescriptionFlagBits:
  - `VK_SUBPASS_DESCRIPTION_ENABLE_LEGACY_DITHERING_BIT_EXT`

If VK_KHR_dynamic_rendering or Vulkan Version 1.3 and VK_KHR_maintenance5 is supported:

- Extending VkPipelineCreateFlagBits2KHR:
  - `VK_PIPELINE_CREATE_2_ENABLE_LEGACY_DITHERING_BIT_EXT`
- Extending VkRenderingFlagBits:
  - `VK_RENDERING_ENABLE_LEGACY_DITHERING_BIT_EXT`

### Version History

- Revision 1, 2022-03-31 (Shahbaz Youssefi)
  - Internal revisions
- Revision 2, 2024-02-22 (Shahbaz Youssefi)
  - Added pipeline create flag to support dynamic rendering

### Issues

1) In OpenGL, the dither state can change dynamically. Should this extension add a pipeline state for dither?

**RESOLVED**: No. Changing dither state is rarely, if ever, done during rendering. Every surveyed Android application either entirely disables dither, explicitly enables it, or uses the default state (which is enabled). Additionally, on some hardware dither can only be specified in a render pass granularity, so a change in dither state would necessarily need to cause a render pass break. This extension considers dynamic changes in OpenGL dither state a theoretical situation, and expects the layer to break the render pass in such a situation without any practical downsides.

## VK_EXT_legacy_vertex_attributes

### Name String

`VK_EXT_legacy_vertex_attributes`

### Extension Type

Device extension

### Registered Extension Number

496

### Revision

1

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_EXT_vertex_input_dynamic_state

### Special Use

- OpenGL / ES support

### Contact

- Mike Blumenkrantz [GitHub]zmike

### Extension Proposal

VK_EXT_legacy_vertex_attributes

### Other Extension Metadata

### Last Modified Date

2024-02-23

### IP Status

No known IP claims.

### Contributors

- Mike Blumenkrantz, Valve
- Piers Daniell, NVIDIA
- Spencer Fricke, LunarG

- Alyssa Rosenzweig, Valve

### Description

This extension adds support for legacy features of (non-64-bit) vertex attributes as found in OpenGL:

- Vertex attributes loaded from arbitrary buffer alignments
- Vertex attributes using arbitrary strides
- Vertex attributes where the component data type of the binding does not match the component numeric type of the shader input

These features are only usable with dynamic vertex input. Unaligned loads of vertex attributes may incur performance penalties, indicated with a property.

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceLegacyVertexAttributesFeaturesEXT
- Extending VkPhysicalDeviceProperties2:
  - VkPhysicalDeviceLegacyVertexAttributesPropertiesEXT

### New Enum Constants

- `VK_EXT_LEGACY_VERTEX_ATTRIBUTES_EXTENSION_NAME`
- `VK_EXT_LEGACY_VERTEX_ATTRIBUTES_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_LEGACY_VERTEX_ATTRIBUTES_FEATURES_EXT`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_LEGACY_VERTEX_ATTRIBUTES_PROPERTIES_EXT`

### Issues

1.) Should implementations convert float/integer values?

**RESOLVED**: No. When fetching an integer data type from float values or float data types from integer values, the resulting shader values are implementation-dependent.

### Version History

- Revision 1, 2024-02-16 (Mike Blumenkrantz)
  - Initial revision

## VK_EXT_map_memory_placed

### Name String

`VK_EXT_map_memory_placed`

**Extension Type**

Device extension

**Registered Extension Number**

273

**Revision**

1

**Ratification Status**

Ratified

**Extension and Version Dependencies**

VK_KHR_map_memory2

**Contact**

- Faith Ekstrand [GitHub]gfxstrand

**Extension Proposal**

VK_EXT_map_memory_placed

**Other Extension Metadata**

**Last Modified Date**

2023-03-21

**IP Status**

No known IP claims.

**Interactions and External Dependencies**

- Depends on apitext:VK_KHR_map_memory2
- Interacts with apitext:VK_EXT_external_memory_host

**Contributors**

- Faith Ekstrand, Collabora
- Tobias Hector, AMD
- James Jones, NVIDIA
- Georg Lehmann, Valve
- Derek Lesho, Codeweavers

**Description**

This extension allows an application to request that vkMapMemory2KHR attempt to place the memory map at a particular virtual address.

### New Structures

- Extending VkMemoryMapInfoKHR:
  - VkMemoryMapPlacedInfoEXT
- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceMapMemoryPlacedFeaturesEXT
- Extending VkPhysicalDeviceProperties2:
  - VkPhysicalDeviceMapMemoryPlacedPropertiesEXT

### New Enum Constants

- `VK_EXT_MAP_MEMORY_PLACED_EXTENSION_NAME`
- `VK_EXT_MAP_MEMORY_PLACED_SPEC_VERSION`
- Extending VkMemoryMapFlagBits:
  - `VK_MEMORY_MAP_PLACED_BIT_EXT`
- Extending VkMemoryUnmapFlagBitsKHR:
  - `VK_MEMORY_UNMAP_RESERVE_BIT_EXT`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_MEMORY_MAP_PLACED_INFO_EXT`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MAP_MEMORY_PLACED_FEATURES_EXT`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MAP_MEMORY_PLACED_PROPERTIES_EXT`

### Version History

- Revision 0, 2024-01-14 (Faith Ekstrand)
  - Internal revisions

## VK_EXT_memory_budget

### Name String

`VK_EXT_memory_budget`

### Extension Type

Device extension

### Registered Extension Number

238

### Revision

1

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### Contact

- Jeff Bolz [GitHub]jeffbolznv

### Other Extension Metadata

### Last Modified Date

2018-10-08

### Contributors

- Jeff Bolz, NVIDIA
- Jeff Juliano, NVIDIA

### Description

While running a Vulkan application, other processes on the machine might also be attempting to use the same device memory, which can pose problems. This extension adds support for querying the amount of memory used and the total memory budget for a memory heap. The values returned by this query are implementation-dependent and can depend on a variety of factors including operating system and system load.

The VkPhysicalDeviceMemoryBudgetPropertiesEXT::`heapBudget` values can be used as a guideline for how much total memory from each heap the **current process** can use at any given time, before allocations may start failing or causing performance degradation. The values may change based on other activity in the system that is outside the scope and control of the Vulkan implementation.

The VkPhysicalDeviceMemoryBudgetPropertiesEXT::`heapUsage` will display the **current process** estimated heap usage.

With this information, the idea is for an application at some interval (once per frame, per few seconds, etc) to query `heapBudget` and `heapUsage`. From here the application can notice if it is over budget and decide how it wants to handle the memory situation (free it, move to host memory, changing mipmap levels, etc). This extension is designed to be used in concert with `VK_EXT_memory_priority` to help with this part of memory management.

### New Structures

- Extending VkPhysicalDeviceMemoryProperties2:
  - VkPhysicalDeviceMemoryBudgetPropertiesEXT

### New Enum Constants

- `VK_EXT_MEMORY_BUDGET_EXTENSION_NAME`
- `VK_EXT_MEMORY_BUDGET_SPEC_VERSION`

- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MEMORY_BUDGET_PROPERTIES_EXT`

#### Version History

- Revision 1, 2018-10-08 (Jeff Bolz)
  - Initial revision

## VK_EXT_memory_priority

#### Name String

`VK_EXT_memory_priority`

#### Extension Type

Device extension

#### Registered Extension Number

239

#### Revision

1

#### Ratification Status

Ratified

#### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

#### Contact

- Jeff Bolz [GitHub]jeffbolznv

#### Other Extension Metadata

#### Last Modified Date

2018-10-08

#### Contributors

- Jeff Bolz, NVIDIA
- Jeff Juliano, NVIDIA

#### Description

This extension adds a `priority` value specified at memory allocation time. On some systems with both device-local and non-device-local memory heaps, the implementation may transparently move memory from one heap to another when a heap becomes full (for example, when the total memory used across all processes exceeds the size of the heap). In such a case, this priority value may be

used to determine which allocations are more likely to remain in device-local memory.

### New Structures

- Extending VkMemoryAllocateInfo:
  - VkMemoryPriorityAllocateInfoEXT
- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceMemoryPriorityFeaturesEXT

### New Enum Constants

- `VK_EXT_MEMORY_PRIORITY_EXTENSION_NAME`
- `VK_EXT_MEMORY_PRIORITY_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_MEMORY_PRIORITY_ALLOCATE_INFO_EXT`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MEMORY_PRIORITY_FEATURES_EXT`

### Version History

- Revision 1, 2018-10-08 (Jeff Bolz)
  - Initial revision

## VK_EXT_mesh_shader

### Name String

`VK_EXT_mesh_shader`

### Extension Type

Device extension

### Registered Extension Number

329

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_spirv_1_4

### API Interactions

- Interacts with VK_NV_device_generated_commands

**SPIR-V Dependencies**

- SPV_EXT_mesh_shader

**Contact**

- Christoph Kubisch [GitHub]pixeljetstream

**Extension Proposal**

VK_EXT_mesh_shader

**Other Extension Metadata**

**Last Modified Date**

2022-01-20

**Interactions and External Dependencies**

- This extension provides API support for `GLSL_EXT_mesh_shader`
- Interacts with Vulkan 1.1
- Interacts with `VK_KHR_multiview`
- Interacts with `VK_KHR_fragment_shading_rate`

**Contributors**

- Christoph Kubisch, NVIDIA
- Pat Brown, NVIDIA
- Jeff Bolz, NVIDIA
- Daniel Koch, NVIDIA
- Piers Daniell, NVIDIA
- Pierre Boudier, NVIDIA
- Patrick Mours, NVIDIA
- David Zhao Akeley, NVIDIA
- Kedarnath Thangudu, NVIDIA
- Timur Kristóf, Valve
- Hans-Kristian Arntzen, Valve
- Philip Rebohle, Valve
- Mike Blumenkrantz, Valve
- Slawomir Grajewski, Intel
- Michal Pietrasiuk, Intel
- Mariusz Merecki, Intel
- Tom Olson, ARM
- Jan-Harald Fredriksen, ARM
- Sandeep Kakarlapudi, ARM
- Ruihao Zhang, QUALCOMM

- Ricardo Garcia, Igalia, S.L.
- Tobias Hector, AMD
- Stu Smith, AMD

### Description

This extension provides a new mechanism allowing applications to generate collections of geometric primitives via programmable mesh shading. It is an alternative to the existing programmable primitive shading pipeline, which relied on generating input primitives by a fixed function assembler as well as fixed function vertex fetch.

This extension also adds support for the following SPIR-V extension in Vulkan:

- `SPV_EXT_mesh_shader`

### New Commands

- vkCmdDrawMeshTasksEXT
- vkCmdDrawMeshTasksIndirectCountEXT
- vkCmdDrawMeshTasksIndirectEXT

### New Structures

- VkDrawMeshTasksIndirectCommandEXT
- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceMeshShaderFeaturesEXT
- Extending VkPhysicalDeviceProperties2:
  - VkPhysicalDeviceMeshShaderPropertiesEXT

### New Enum Constants

- `VK_EXT_MESH_SHADER_EXTENSION_NAME`
- `VK_EXT_MESH_SHADER_SPEC_VERSION`
- Extending VkPipelineStageFlagBits:
  - `VK_PIPELINE_STAGE_MESH_SHADER_BIT_EXT`
  - `VK_PIPELINE_STAGE_TASK_SHADER_BIT_EXT`
- Extending VkQueryPipelineStatisticFlagBits:
  - `VK_QUERY_PIPELINE_STATISTIC_MESH_SHADER_INVOCATIONS_BIT_EXT`
  - `VK_QUERY_PIPELINE_STATISTIC_TASK_SHADER_INVOCATIONS_BIT_EXT`
- Extending VkQueryType:
  - `VK_QUERY_TYPE_MESH_PRIMITIVES_GENERATED_EXT`
- Extending VkShaderStageFlagBits:

    - VK_SHADER_STAGE_MESH_BIT_EXT
    - VK_SHADER_STAGE_TASK_BIT_EXT
- Extending VkStructureType:
    - VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MESH_SHADER_FEATURES_EXT
    - VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MESH_SHADER_PROPERTIES_EXT

If VK_NV_device_generated_commands is supported:

- Extending VkIndirectCommandsTokenTypeNV:
    - VK_INDIRECT_COMMANDS_TOKEN_TYPE_DRAW_MESH_TASKS_NV

**New or Modified Built-In Variables**

- CullPrimitiveEXT
- PrimitivePointIndicesEXT
- PrimitiveLineIndicesEXT
- PrimitiveTriangleIndicesEXT
- (modified)Position
- (modified)PointSize
- (modified)ClipDistance
- (modified)CullDistance
- (modified)PrimitiveId
- (modified)Layer
- (modified)ViewportIndex
- (modified)NumWorkgroups
- (modified)WorkgroupSize
- (modified)WorkgroupId
- (modified)LocalInvocationId
- (modified)GlobalInvocationId
- (modified)LocalInvocationIndex
- (modified)NumSubgroups
- (modified)SubgroupId
- (modified)DrawIndex
- (modified)PrimitiveShadingRateKHR
- (modified)ViewIndex

**New SPIR-V Capability**

- MeshShadingEXT

### Version History

- Revision 1, 2022-03-08 (Christoph Kubisch, Daniel Koch, Patrick Mours)
  - Initial revision

## VK_EXT_metal_objects

### Name String

`VK_EXT_metal_objects`

### Extension Type

Device extension

### Registered Extension Number

312

### Revision

2

### Ratification Status

Not ratified

### Extension and Version Dependencies

None

### Contact

- Bill Hollings [GitHub]billhollings

### Extension Proposal

VK_EXT_metal_objects

### Other Extension Metadata

### Last Modified Date

2024-04-04

### IP Status

No known IP claims.

### Contributors

- Bill Hollings, The Brenwill Workshop Ltd.
- Dzmitry Malyshau, Mozilla Corp.

### Description

In a Vulkan implementation that is layered on top of Metal on Apple device platforms, this extension provides the ability to import and export the underlying Metal objects associated with specific Vulkan objects.

As detailed in the extension proposal document, this extension adds one new Vulkan command, vkExportMetalObjectsEXT, to export underlying Metal objects from Vulkan objects, and supports importing the appropriate existing Metal objects when creating Vulkan objects of types VkDeviceMemory, VkImage, VkSemaphore, and VkEvent,

The intent is that this extension will be advertised and supported only on implementations that are layered on top of Metal on Apple device platforms.

#### New Base Types

- IOSurfaceRef
- MTLBuffer_id
- MTLCommandQueue_id
- MTLDevice_id
- MTLSharedEvent_id
- MTLTexture_id

#### New Commands

- vkExportMetalObjectsEXT

#### New Structures

- VkExportMetalObjectsInfoEXT
- Extending VkExportMetalObjectsInfoEXT:
  - VkExportMetalBufferInfoEXT
  - VkExportMetalCommandQueueInfoEXT
  - VkExportMetalDeviceInfoEXT
  - VkExportMetalIOSurfaceInfoEXT
  - VkExportMetalSharedEventInfoEXT
  - VkExportMetalTextureInfoEXT
- Extending VkImageCreateInfo:
  - VkImportMetalIOSurfaceInfoEXT
  - VkImportMetalTextureInfoEXT
- Extending VkInstanceCreateInfo, VkMemoryAllocateInfo, VkImageCreateInfo, VkImageViewCreateInfo, VkBufferViewCreateInfo, VkSemaphoreCreateInfo, VkEventCreateInfo:
  - VkExportMetalObjectCreateInfoEXT
- Extending VkMemoryAllocateInfo:
  - VkImportMetalBufferInfoEXT
- Extending VkSemaphoreCreateInfo, VkEventCreateInfo:

- VkImportMetalSharedEventInfoEXT

### New Enums

- VkExportMetalObjectTypeFlagBitsEXT

### New Bitmasks

- VkExportMetalObjectTypeFlagsEXT

### New Enum Constants

- `VK_EXT_METAL_OBJECTS_EXTENSION_NAME`
- `VK_EXT_METAL_OBJECTS_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_EXPORT_METAL_BUFFER_INFO_EXT`
  - `VK_STRUCTURE_TYPE_EXPORT_METAL_COMMAND_QUEUE_INFO_EXT`
  - `VK_STRUCTURE_TYPE_EXPORT_METAL_DEVICE_INFO_EXT`
  - `VK_STRUCTURE_TYPE_EXPORT_METAL_IO_SURFACE_INFO_EXT`
  - `VK_STRUCTURE_TYPE_EXPORT_METAL_OBJECTS_INFO_EXT`
  - `VK_STRUCTURE_TYPE_EXPORT_METAL_OBJECT_CREATE_INFO_EXT`
  - `VK_STRUCTURE_TYPE_EXPORT_METAL_SHARED_EVENT_INFO_EXT`
  - `VK_STRUCTURE_TYPE_EXPORT_METAL_TEXTURE_INFO_EXT`
  - `VK_STRUCTURE_TYPE_IMPORT_METAL_BUFFER_INFO_EXT`
  - `VK_STRUCTURE_TYPE_IMPORT_METAL_IO_SURFACE_INFO_EXT`
  - `VK_STRUCTURE_TYPE_IMPORT_METAL_SHARED_EVENT_INFO_EXT`
  - `VK_STRUCTURE_TYPE_IMPORT_METAL_TEXTURE_INFO_EXT`

### Issues

None.

### Version History

- Revision 1, 2022-05-28 (Bill Hollings)
  - Initial draft.
  - Incorporated feedback from review by the Vulkan Working Group. Renamed many structures, moved import/export of MTLBuffer to VkDeviceMemory, added export of MTLSharedEvent, added import of MTLSharedEvent for VkSemaphore and VkEvent, and changed used bit mask fields to individual bit fields to simplify Valid Usage rules.
- Revision 2, 2024-04-04 (Bill Hollings)
  - Add an `__unsafe_unretained` ownership qualifier to all Metal object declarations, to support

Automatic Reference Counting (ARC) on Apple devices.

## VK_EXT_metal_surface

### Name String

`VK_EXT_metal_surface`

### Extension Type

Instance extension

### Registered Extension Number

218

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_surface

### Contact

- Dzmitry Malyshau [GitHub]kvark

### Other Extension Metadata

### Last Modified Date

2018-10-01

### IP Status

No known IP claims.

### Contributors

- Dzmitry Malyshau, Mozilla Corp.

### Description

The `VK_EXT_metal_surface` extension is an instance extension. It provides a mechanism to create a VkSurfaceKHR object (defined by the `VK_KHR_surface` extension) from CAMetalLayer, which is the native rendering surface of Apple's Metal framework.

### New Base Types

- CAMetalLayer

### New Commands

- vkCreateMetalSurfaceEXT

### New Structures

- VkMetalSurfaceCreateInfoEXT

### New Bitmasks

- VkMetalSurfaceCreateFlagsEXT

### New Enum Constants

- `VK_EXT_METAL_SURFACE_EXTENSION_NAME`
- `VK_EXT_METAL_SURFACE_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_METAL_SURFACE_CREATE_INFO_EXT`

### Version History

- Revision 1, 2018-10-01 (Dzmitry Malyshau)
  - Initial version

## VK_EXT_multi_draw

### Name String

`VK_EXT_multi_draw`

### Extension Type

Device extension

### Registered Extension Number

393

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### Contact

- Mike Blumenkrantz [GitHub]zmike

### Other Extension Metadata

**Last Modified Date**

2021-05-19

**Interactions and External Dependencies**

- Interacts with Vulkan 1.1.
- Interacts with `VK_KHR_shader_draw_parameters`.

**IP Status**

No known IP claims.

**Contributors**

- Mike Blumenkrantz, VALVE
- Piers Daniell, NVIDIA
- Faith Ekstrand, INTEL
- Spencer Fricke, SAMSUNG
- Ricardo Garcia, IGALIA
- Jon Leech, KHRONOS
- Stu Smith, AMD

**Description**

Processing multiple draw commands in sequence incurs measurable overhead within drivers due to repeated state checks and updates during dispatch. This extension enables passing the entire sequence of draws directly to the driver in order to avoid any such overhead, using an array of VkMultiDrawInfoEXT or VkMultiDrawIndexedInfoEXT structs with `vkCmdDrawMultiEXT` or `vkCmdDrawMultiIndexedEXT`, respectively. These functions could be used any time multiple draw commands are being recorded without any state changes between them in order to maximize performance.

**New Commands**

- vkCmdDrawMultiEXT
- vkCmdDrawMultiIndexedEXT

**New Structures**

- VkMultiDrawIndexedInfoEXT
- VkMultiDrawInfoEXT
- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceMultiDrawFeaturesEXT
- Extending VkPhysicalDeviceProperties2:
  - VkPhysicalDeviceMultiDrawPropertiesEXT

### New Enum Constants

- `VK_EXT_MULTI_DRAW_EXTENSION_NAME`
- `VK_EXT_MULTI_DRAW_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MULTI_DRAW_FEATURES_EXT`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MULTI_DRAW_PROPERTIES_EXT`

### New or Modified Built-In Variables

- (modified)`DrawIndex`

### Version History

- Revision 1, 2021-01-20 (Mike Blumenkrantz)
  - Initial version

## VK_EXT_multisampled_render_to_single_sampled

### Name String

`VK_EXT_multisampled_render_to_single_sampled`

### Extension Type

Device extension

### Registered Extension Number

377

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_create_renderpass2
and
VK_KHR_depth_stencil_resolve
or
Vulkan Version 1.2

### Contact

- Shahbaz Youssefi [GitHub]syoussefi

### Extension Proposal

VK_EXT_multisampled_render_to_single_sampled

### Other Extension Metadata

#### Last Modified Date

2021-04-16

#### IP Status

No known IP claims.

#### Contributors

- Shahbaz Youssefi, Google
- Jan-Harald Fredriksen, Arm
- Jörg Wagner, Arm
- Matthew Netsch, Qualcomm Technologies, Inc.
- Jarred Davies, Imagination Technologies

#### Description

With careful usage of resolve attachments, multisampled image memory allocated with `VK_MEMORY_PROPERTY_LAZILY_ALLOCATED_BIT`, `loadOp` not equal to `VK_ATTACHMENT_LOAD_OP_LOAD` and `storeOp` not equal to `VK_ATTACHMENT_STORE_OP_STORE`, a Vulkan application is able to efficiently perform multisampled rendering without incurring any additional memory penalty on some implementations.

Under certain circumstances however, the application may not be able to complete its multisampled rendering within a single render pass; for example if it does partial rasterization from frame to frame, blending on an image from a previous frame, or in emulation of GL_EXT_multisampled_render_to_texture. In such cases, the application can use an initial subpass to effectively load single-sampled data from the next subpass's resolve attachment and fill in the multisampled attachment which otherwise uses `loadOp` equal to `VK_ATTACHMENT_LOAD_OP_DONT_CARE`. However, this is not always possible (for example for stencil in the absence of VK_EXT_shader_stencil_export) and has multiple drawbacks.

Some implementations are able to perform said operation efficiently in hardware, effectively loading a multisampled attachment from the contents of a single sampled one. Together with the ability to perform a resolve operation at the end of a subpass, these implementations are able to perform multisampled rendering on single-sampled attachments with no extra memory or bandwidth overhead. This extension exposes this capability by allowing a framebuffer and render pass to include single-sampled attachments while rendering is done with a specified number of samples.

#### New Structures

- Extending VkFormatProperties2:
  - VkSubpassResolvePerformanceQueryEXT
- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceMultisampledRenderToSingleSampledFeaturesEXT

- Extending VkSubpassDescription2, VkRenderingInfo:
  - VkMultisampledRenderToSingleSampledInfoEXT

#### New Enum Constants

- VK_EXT_MULTISAMPLED_RENDER_TO_SINGLE_SAMPLED_EXTENSION_NAME
- VK_EXT_MULTISAMPLED_RENDER_TO_SINGLE_SAMPLED_SPEC_VERSION
- Extending VkImageCreateFlagBits:
  - VK_IMAGE_CREATE_MULTISAMPLED_RENDER_TO_SINGLE_SAMPLED_BIT_EXT
- Extending VkStructureType:
  - VK_STRUCTURE_TYPE_MULTISAMPLED_RENDER_TO_SINGLE_SAMPLED_INFO_EXT
  - VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MULTISAMPLED_RENDER_TO_SINGLE_SAMPLED_FEATURES_EXT
  - VK_STRUCTURE_TYPE_SUBPASS_RESOLVE_PERFORMANCE_QUERY_EXT

#### Issues

1) Could the multisampled attachment be initialized through some form of copy?

**RESOLVED**: No. Some implementations do not support copying between attachments in general, and find expressing this operation through a copy unnatural.

2) Another way to achieve this is by introducing a new loadOp to load the contents of the multisampled image from a single-sampled one. Why is this extension preferred?

**RESOLVED**: Using this extension simplifies the application, as it does not need to manage a secondary lazily-allocated image. Additionally, using this extension leaves less room for error; for example a single mistake in loadOp or storeOp would result in the lazily-allocated image to actually take up memory, and remain so until destruction.

3) There is no guarantee that multisampled data between two subpasses with the same number of samples will be retained as the implementation may be forced to split the render pass implicitly for various reasons. Should this extension require that every subpass that uses multisampled-render-to-single-sampled end in an implicit render pass split (which results in a resolve operation)?

**RESOLVED**: No. Not requiring this allows render passes with multiple multisampled-render-to-single-sampled subpasses to potentially execute more efficiently (though there is no guarantee).

#### Version History

- Revision 1, 2021-04-12 (Shahbaz Youssefi)

## VK_EXT_mutable_descriptor_type

#### Name String

VK_EXT_mutable_descriptor_type

**Extension Type**

Device extension

**Registered Extension Number**

495

**Revision**

1

**Ratification Status**

Not ratified

**Extension and Version Dependencies**

VK_KHR_maintenance3

**Special Use**

- D3D support

**Contact**

- Joshua Ashton [GitHub]Joshua-Ashton
- Hans-Kristian Arntzen [GitHub]HansKristian-Work

**Extension Proposal**

VK_EXT_mutable_descriptor_type

**Other Extension Metadata**

**Last Modified Date**

2022-08-22

**IP Status**

No known IP claims.

**Contributors**

- Joshua Ashton, Valve
- Hans-Kristian Arntzen, Valve

**Description**

This extension allows applications to reduce descriptor memory footprint by allowing a descriptor to be able to mutate to a given list of descriptor types depending on which descriptor types are written into, or copied into a descriptor set.

The main use case this extension intends to address is descriptor indexing with `VK_DESCRIPTOR_BINDING_VARIABLE_DESCRIPTOR_COUNT_BIT` where the descriptor types are completely generic, as this means applications can allocate one large descriptor set, rather than having one large descriptor set per descriptor type, which significantly bloats descriptor memory usage and causes performance issues.

This extension also adds a mechanism to declare that a descriptor pool, and therefore the descriptor sets that are allocated from it, reside only in host memory; as such these descriptors can only be updated/copied, but not bound.

These features together allow much more efficient emulation of the raw D3D12 binding model. This extension is primarily intended to be useful for API layering efforts.

#### New Structures

- VkMutableDescriptorTypeListEXT
- Extending VkDescriptorSetLayoutCreateInfo, VkDescriptorPoolCreateInfo:
  - VkMutableDescriptorTypeCreateInfoEXT
- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceMutableDescriptorTypeFeaturesEXT

#### New Enum Constants

- `VK_EXT_MUTABLE_DESCRIPTOR_TYPE_EXTENSION_NAME`
- `VK_EXT_MUTABLE_DESCRIPTOR_TYPE_SPEC_VERSION`
- Extending VkDescriptorPoolCreateFlagBits:
  - `VK_DESCRIPTOR_POOL_CREATE_HOST_ONLY_BIT_EXT`
- Extending VkDescriptorSetLayoutCreateFlagBits:
  - `VK_DESCRIPTOR_SET_LAYOUT_CREATE_HOST_ONLY_POOL_BIT_EXT`
- Extending VkDescriptorType:
  - `VK_DESCRIPTOR_TYPE_MUTABLE_EXT`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_MUTABLE_DESCRIPTOR_TYPE_CREATE_INFO_EXT`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MUTABLE_DESCRIPTOR_TYPE_FEATURES_EXT`

#### Version History

- Revision 1, 2022-08-22 (Jon Leech)
  - Initial version, promoted from VK_VALVE_mutable_descriptor_type.

## VK_EXT_nested_command_buffer

#### Name String

`VK_EXT_nested_command_buffer`

#### Extension Type

Device extension

### Registered Extension Number

452

### Revision

1

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### Contact

- Piers Daniell [GitHub]pdaniell-nv

### Other Extension Metadata

### Last Modified Date

2023-09-18

### Contributors

- Daniel Story, Nintendo
- Peter Kohaut, NVIDIA
- Shahbaz Youssefi, Google
- Slawomir Grajewski, Intel
- Stu Smith, AMD

### Description

With core Vulkan it is not legal to call vkCmdExecuteCommands when recording a secondary command buffer. This extension relaxes that restriction, allowing secondary command buffers to execute other secondary command buffers.

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceNestedCommandBufferFeaturesEXT
- Extending VkPhysicalDeviceProperties2:
  - VkPhysicalDeviceNestedCommandBufferPropertiesEXT

### New Enum Constants

- `VK_EXT_NESTED_COMMAND_BUFFER_EXTENSION_NAME`
- `VK_EXT_NESTED_COMMAND_BUFFER_SPEC_VERSION`

- Extending VkRenderingFlagBits:
  - `VK_RENDERING_CONTENTS_INLINE_BIT_EXT`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_NESTED_COMMAND_BUFFER_FEATURES_EXT`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_NESTED_COMMAND_BUFFER_PROPERTIES_EXT`
- Extending VkSubpassContents:
  - `VK_SUBPASS_CONTENTS_INLINE_AND_SECONDARY_COMMAND_BUFFERS_EXT`

#### Issues

1) The Command Buffer Levels property for the Vulkan commands comes from the `cmdbufferlevel` attribute in `vk.xml` for the command, and it is currently not possible to modify this attribute based on whether an extension is enabled. For this extension we want the `cmdbufferlevel` attribute for vkCmdExecuteCommands to be `primary,secondary` when this extension is enabled and `primary` otherwise.

**RESOLVED**: The `cmdbufferlevel` attribute for vkCmdExecuteCommands has been changed to `primary,secondary` and a new VUID added to prohibit recording this command in a secondary command buffer unless this extension is enabled.

#### Version History

- Revision 1, 2023-09-18 (Piers Daniell)
  - Internal revisions

## VK_EXT_non_seamless_cube_map

#### Name String

`VK_EXT_non_seamless_cube_map`

#### Extension Type

Device extension

#### Registered Extension Number

423

#### Revision

1

#### Ratification Status

Not ratified

#### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

**Special Uses**

- D3D support
- OpenGL / ES support

**Contact**

- Georg Lehmann [GitHub]DadSchoorse

**Extension Proposal**

VK_EXT_non_seamless_cube_map

**Other Extension Metadata**

**Last Modified Date**

2021-09-04

**IP Status**

No known IP claims.

**Contributors**

- Georg Lehmann

**Description**

This extension provides functionality to disable cube map edge handling on a per sampler level which matches the behavior of other graphics APIs.

This extension may be useful for building translation layers for those APIs or for porting applications that rely on this cube map behavior.

**New Structures**

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceNonSeamlessCubeMapFeaturesEXT

**New Enum Constants**

- `VK_EXT_NON_SEAMLESS_CUBE_MAP_EXTENSION_NAME`
- `VK_EXT_NON_SEAMLESS_CUBE_MAP_SPEC_VERSION`
- Extending VkSamplerCreateFlagBits:
  - `VK_SAMPLER_CREATE_NON_SEAMLESS_CUBE_MAP_BIT_EXT`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_NON_SEAMLESS_CUBE_MAP_FEATURES_EXT`

**Version History**

- Revision 1, 2021-09-04 (Georg Lehmann)
  - First Version

## VK_EXT_opacity_micromap

### Name String

`VK_EXT_opacity_micromap`

### Extension Type

Device extension

### Registered Extension Number

397

### Revision

2

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_acceleration_structure
and
VK_KHR_synchronization2
or
Vulkan Version 1.3

### SPIR-V Dependencies

- SPV_EXT_opacity_micromap

### Contact

- Christoph Kubisch [GitHub]pixeljetstream
- Eric Werness

### Extension Proposal

VK_EXT_opacity_micromap

### Other Extension Metadata

### Last Modified Date

2022-08-24

### Interactions and External Dependencies

- This extension provides API support for `GLSL_EXT_opacity_micromap`

### Contributors

- Christoph Kubisch, NVIDIA
- Eric Werness, NVIDIA
- Josh Barczak, Intel
- Stu Smith, AMD

### Description

When adding transparency to a ray traced scene, an application can choose between further tessellating the geometry or using an any-hit shader to allow the ray through specific parts of the geometry. These options have the downside of either significantly increasing memory consumption or adding runtime overhead to run shader code in the middle of traversal, respectively.

This extension adds the ability to add an *opacity micromap* to geometry when building an acceleration structure. The opacity micromap compactly encodes opacity information which can be read by the implementation to mark parts of triangles as opaque or transparent. The format is externally visible to allow the application to compress its internal geometry and surface representations into the compressed format ahead of time. The compressed format subdivides each triangle into a set of subtriangles, each of which can be assigned either two or four opacity values. These opacity values can control if a ray hitting that subtriangle is treated as an opaque hit, complete miss, or possible hit, depending on the controls described in Ray Opacity Micromap.

This extension provides:

- a VkMicromapEXT structure to store the micromap,
- functions similar to acceleration structure build functions to build the opacity micromap array, and
- a structure to extend VkAccelerationStructureGeometryTrianglesDataKHR to attach a micromap to the geometry of the acceleration structure.

### New Object Types

- VkMicromapEXT

### New Commands

- vkBuildMicromapsEXT
- vkCmdBuildMicromapsEXT
- vkCmdCopyMemoryToMicromapEXT
- vkCmdCopyMicromapEXT
- vkCmdCopyMicromapToMemoryEXT
- vkCmdWriteMicromapsPropertiesEXT
- vkCopyMemoryToMicromapEXT
- vkCopyMicromapEXT
- vkCopyMicromapToMemoryEXT
- vkCreateMicromapEXT
- vkDestroyMicromapEXT
- vkGetDeviceMicromapCompatibilityEXT
- vkGetMicromapBuildSizesEXT
- vkWriteMicromapsPropertiesEXT

### New Structures

- VkCopyMemoryToMicromapInfoEXT
- VkCopyMicromapInfoEXT
- VkCopyMicromapToMemoryInfoEXT
- VkMicromapBuildInfoEXT
- VkMicromapBuildSizesInfoEXT
- VkMicromapCreateInfoEXT
- VkMicromapTriangleEXT
- VkMicromapUsageEXT
- VkMicromapVersionInfoEXT
- Extending VkAccelerationStructureGeometryTrianglesDataKHR:
  - VkAccelerationStructureTrianglesOpacityMicromapEXT
- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceOpacityMicromapFeaturesEXT
- Extending VkPhysicalDeviceProperties2:
  - VkPhysicalDeviceOpacityMicromapPropertiesEXT

### New Enums

- VkBuildMicromapFlagBitsEXT
- VkBuildMicromapModeEXT
- VkCopyMicromapModeEXT
- VkMicromapCreateFlagBitsEXT
- VkMicromapTypeEXT
- VkOpacityMicromapFormatEXT
- VkOpacityMicromapSpecialIndexEXT

### New Bitmasks

- VkBuildMicromapFlagsEXT
- VkMicromapCreateFlagsEXT

### New Enum Constants

- `VK_EXT_OPACITY_MICROMAP_EXTENSION_NAME`
- `VK_EXT_OPACITY_MICROMAP_SPEC_VERSION`
- Extending VkAccessFlagBits2:
  - `VK_ACCESS_2_MICROMAP_READ_BIT_EXT`
  - `VK_ACCESS_2_MICROMAP_WRITE_BIT_EXT`

- Extending VkBufferUsageFlagBits:
  - VK_BUFFER_USAGE_MICROMAP_BUILD_INPUT_READ_ONLY_BIT_EXT
  - VK_BUFFER_USAGE_MICROMAP_STORAGE_BIT_EXT
- Extending VkBuildAccelerationStructureFlagBitsKHR:
  - VK_BUILD_ACCELERATION_STRUCTURE_ALLOW_DISABLE_OPACITY_MICROMAPS_EXT
  - VK_BUILD_ACCELERATION_STRUCTURE_ALLOW_OPACITY_MICROMAP_DATA_UPDATE_EXT
  - VK_BUILD_ACCELERATION_STRUCTURE_ALLOW_OPACITY_MICROMAP_UPDATE_EXT
- Extending VkGeometryInstanceFlagBitsKHR:
  - VK_GEOMETRY_INSTANCE_DISABLE_OPACITY_MICROMAPS_EXT
  - VK_GEOMETRY_INSTANCE_FORCE_OPACITY_MICROMAP_2_STATE_EXT
- Extending VkObjectType:
  - VK_OBJECT_TYPE_MICROMAP_EXT
- Extending VkPipelineCreateFlagBits:
  - VK_PIPELINE_CREATE_RAY_TRACING_OPACITY_MICROMAP_BIT_EXT
- Extending VkPipelineStageFlagBits2:
  - VK_PIPELINE_STAGE_2_MICROMAP_BUILD_BIT_EXT
- Extending VkQueryType:
  - VK_QUERY_TYPE_MICROMAP_COMPACTED_SIZE_EXT
  - VK_QUERY_TYPE_MICROMAP_SERIALIZATION_SIZE_EXT
- Extending VkStructureType:
  - VK_STRUCTURE_TYPE_ACCELERATION_STRUCTURE_TRIANGLES_OPACITY_MICROMAP_EXT
  - VK_STRUCTURE_TYPE_COPY_MEMORY_TO_MICROMAP_INFO_EXT
  - VK_STRUCTURE_TYPE_COPY_MICROMAP_INFO_EXT
  - VK_STRUCTURE_TYPE_COPY_MICROMAP_TO_MEMORY_INFO_EXT
  - VK_STRUCTURE_TYPE_MICROMAP_BUILD_INFO_EXT
  - VK_STRUCTURE_TYPE_MICROMAP_BUILD_SIZES_INFO_EXT
  - VK_STRUCTURE_TYPE_MICROMAP_CREATE_INFO_EXT
  - VK_STRUCTURE_TYPE_MICROMAP_VERSION_INFO_EXT
  - VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_OPACITY_MICROMAP_FEATURES_EXT
  - VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_OPACITY_MICROMAP_PROPERTIES_EXT

### Reference Code

```
uint32_t BarycentricsToSpaceFillingCurveIndex(float u, float v, uint32_t level)
{
    u = clamp(u, 0.0f, 1.0f);
```

```
    v = clamp(v, 0.0f, 1.0f);

    uint32_t iu, iv, iw;

    // Quantize barycentric coordinates
    float fu = u * (1u << level);
    float fv = v * (1u << level);

    iu = (uint32_t)fu;
    iv = (uint32_t)fv;

    float uf = fu - float(iu);
    float vf = fv - float(iv);

    if (iu >= (1u << level)) iu = (1u << level) - 1u;
    if (iv >= (1u << level)) iv = (1u << level) - 1u;

    uint32_t iuv = iu + iv;

    if (iuv >= (1u << level))
        iu -= iuv - (1u << level) + 1u;

    iw = ~(iu + iv);

    if (uf + vf >= 1.0f && iuv < (1u << level) - 1u) --iw;

    uint32_t b0 = ~(iu ^ iw);
    b0 &= ((1u << level) - 1u);
    uint32_t t = (iu ^ iv) & b0;

    uint32_t f = t;
    f ^= f >> 1u;
    f ^= f >> 2u;
    f ^= f >> 4u;
    f ^= f >> 8u;
    uint32_t b1 = ((f ^ iu) & ~b0) | t;

    // Interleave bits
    b0 = (b0 | (b0 << 8u)) & 0x00ff00ffu;
    b0 = (b0 | (b0 << 4u)) & 0x0f0f0f0fu;
    b0 = (b0 | (b0 << 2u)) & 0x33333333u;
    b0 = (b0 | (b0 << 1u)) & 0x55555555u;
    b1 = (b1 | (b1 << 8u)) & 0x00ff00ffu;
    b1 = (b1 | (b1 << 4u)) & 0x0f0f0f0fu;
    b1 = (b1 | (b1 << 2u)) & 0x33333333u;
    b1 = (b1 | (b1 << 1u)) & 0x55555555u;

    return b0 | (b1 << 1u);
}
```

### Issues

(1) Is the build actually similar to an acceleration structure build?

- Resolved: The build should be much lighter-weight than an acceleration structure build, but the infrastructure is similar enough that it makes sense to keep the concepts compatible.

(2) Why does VkMicromapUsageEXT not have type/pNext?

- Resolved: There can be a very large number of these structures, so doubling the size of these can be significant memory consumption. Also, an application may be loading these directly from a file which is more compatible with it being a flat structure. The including structures are extensible and are probably a more suitable place to add extensibility.

(3) Why is there a SPIR-V extension?

- Resolved: There is a ray flag. To be consistent with how the existing ray tracing extensions work that ray flag needs its own extension.

(4) Should there be indirect micromap build?

- Resolved: Not for now. There is more in-depth usage metadata required and it seems less likely that something like a GPU culling system would need to change the counts for a micromap.

(5) Should micromaps have a micromap device address?

- Resolved: There is no need right now (can just use the handle) but that is a bit different from acceleration structures, though the two are not completely parallel in their usage.

(6) Why are the alignment requirements defined as a mix of hardcoded values and caps?

- Resolved: This is most parallel with the definition of `VK_KHR_acceleration_structure` and maintaining commonality makes it easier for applications to share memory.

### Version History

- Revision 2, 2022-06-22 (Eric Werness)
  - EXTify and clean up for discussion
- Revision 1, 2022-01-01 (Eric Werness)
  - Initial revision

## VK_EXT_pageable_device_local_memory

### Name String

`VK_EXT_pageable_device_local_memory`

### Extension Type

Device extension

**Registered Extension Number**

413

**Revision**

1

**Ratification Status**

Not ratified

**Extension and Version Dependencies**

VK_EXT_memory_priority

**Contact**

- Piers Daniell [GitHub]pdaniell-nv

**Other Extension Metadata**

**Last Modified Date**

2021-08-24

**Contributors**

- Hans-Kristian Arntzen, Valve
- Axel Gneiting, id Software
- Billy Khan, id Software
- Daniel Koch, NVIDIA
- Chris Lentini, NVIDIA
- Joshua Schnarr, NVIDIA
- Stu Smith, AMD

**Description**

Vulkan is frequently implemented on multi-user and multi-process operating systems where the device-local memory can be shared by more than one process. On such systems the size of the device-local memory available to the application may not be the full size of the memory heap at all times. In order for these operating systems to support multiple applications the device-local memory is virtualized and paging is used to move memory between device-local and host-local memory heaps, transparent to the application.

The current Vulkan specification does not expose this behavior well and may cause applications to make suboptimal memory choices when allocating memory. For example, in a system with multiple applications running, the application may think that device-local memory is full and revert to making performance-sensitive allocations from host-local memory. In reality the memory heap might not have been full, it just appeared to be at the time memory consumption was queried, and a device-local allocation would have succeeded. A well designed operating system that implements pageable device-local memory will try to have all memory allocations for the foreground application paged into device-local memory, and paged out for other applications as needed when

not in use.

When this extension is exposed by the Vulkan implementation it indicates to the application that the operating system implements pageable device-local memory and the application should adjust its memory allocation strategy accordingly. The extension also exposes a new vkSetDeviceMemoryPriorityEXT function to allow the application to dynamically adjust the priority of existing memory allocations based on its current needs. This will help the operating system page out lower priority memory allocations before higher priority allocations when needed. It will also help the operating system decide which memory allocations to page back into device-local memory first.

To take best advantage of pageable device-local memory the application must create the Vulkan device with the VkPhysicalDevicePageableDeviceLocalMemoryFeaturesEXT ::`pageableDeviceLocalMemory` feature enabled. When enabled the Vulkan implementation will allow device-local memory allocations to be paged in and out by the operating system, and **may** not return VK_ERROR_OUT_OF_DEVICE_MEMORY even if device-local memory appears to be full, but will instead page this, or other allocations, out to make room. The Vulkan implementation will also ensure that host-local memory allocations will never be promoted to device-local memory by the operating system, or consume device-local memory.

#### New Commands

- vkSetDeviceMemoryPriorityEXT

#### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDevicePageableDeviceLocalMemoryFeaturesEXT

#### New Enum Constants

- `VK_EXT_PAGEABLE_DEVICE_LOCAL_MEMORY_EXTENSION_NAME`
- `VK_EXT_PAGEABLE_DEVICE_LOCAL_MEMORY_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PAGEABLE_DEVICE_LOCAL_MEMORY_FEATURES_EXT`

#### Version History

- Revision 1, 2021-08-24 (Piers Daniell)
  - Initial revision

### VK_EXT_pci_bus_info

#### Name String

`VK_EXT_pci_bus_info`

**Extension Type**

Device extension

**Registered Extension Number**

213

**Revision**

2

**Ratification Status**

Ratified

**Extension and Version Dependencies**

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

**Contact**

- Matthaeus G. Chajdas [GitHub]anteru

**Other Extension Metadata**

**Last Modified Date**

2018-12-10

**IP Status**

No known IP claims.

**Contributors**

- Matthaeus G. Chajdas, AMD
- Daniel Rakos, AMD

**Description**

This extension adds a new query to obtain PCI bus information about a physical device.

Not all physical devices have PCI bus information, either due to the device not being connected to the system through a PCI interface or due to platform specific restrictions and policies. Thus this extension is only expected to be supported by physical devices which can provide the information.

As a consequence, applications should always check for the presence of the extension string for each individual physical device for which they intend to issue the new query for and should not have any assumptions about the availability of the extension on any given platform.

**New Structures**

- Extending VkPhysicalDeviceProperties2:
  - VkPhysicalDevicePCIBusInfoPropertiesEXT

### New Enum Constants

- `VK_EXT_PCI_BUS_INFO_EXTENSION_NAME`
- `VK_EXT_PCI_BUS_INFO_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PCI_BUS_INFO_PROPERTIES_EXT`

### Version History

- Revision 2, 2018-12-10 (Daniel Rakos)
  - Changed all members of the new structure to have the uint32_t type
- Revision 1, 2018-10-11 (Daniel Rakos)
  - Initial revision

## VK_EXT_physical_device_drm

### Name String

`VK_EXT_physical_device_drm`

### Extension Type

Device extension

### Registered Extension Number

354

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### Contact

- Simon Ser [GitHub]emersion

### Other Extension Metadata

### Last Modified Date

2021-06-09

### IP Status

No known IP claims.

### Contributors

- Simon Ser

### Description

This extension provides new facilities to query DRM properties for physical devices, enabling users to match Vulkan physical devices with DRM nodes on Linux.

Its functionality closely overlaps with `EGL_EXT_device_drm`[1]. Unlike the EGL extension, this extension does not expose a string containing the name of the device file and instead exposes device minor numbers.

DRM defines multiple device node types. Each physical device may have one primary node and one render node associated. Physical devices may have no primary node (e.g. if the device does not have a display subsystem), may have no render node (e.g. if it is a software rendering engine), or may have neither (e.g. if it is a software rendering engine without a display subsystem).

To query DRM properties for a physical device, chain VkPhysicalDeviceDrmPropertiesEXT to VkPhysicalDeviceProperties2.

### New Structures

- Extending VkPhysicalDeviceProperties2:
  - VkPhysicalDeviceDrmPropertiesEXT

### New Enum Constants

- `VK_EXT_PHYSICAL_DEVICE_DRM_EXTENSION_NAME`
- `VK_EXT_PHYSICAL_DEVICE_DRM_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DRM_PROPERTIES_EXT`

### References

1. `EGL_EXT_device_drm`

### Version History

- Revision 1, 2021-06-09
  - First stable revision

## VK_EXT_pipeline_creation_cache_control

### Name String

`VK_EXT_pipeline_creation_cache_control`

### Extension Type

Device extension

**Registered Extension Number**

298

**Revision**

3

**Ratification Status**

Not ratified

**Extension and Version Dependencies**

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

**Deprecation State**

- *Promoted* to Vulkan 1.3

**Contact**

- Gregory Grebe [GitHub]grgrebe_amd

**Other Extension Metadata**

**Last Modified Date**

2020-03-23

**IP Status**

No known IP claims.

**Contributors**

- Gregory Grebe, AMD
- Tobias Hector, AMD
- Matthaeus Chajdas, AMD
- Mitch Singer, AMD
- Spencer Fricke, Samsung Electronics
- Stuart Smith, Imagination Technologies
- Jeff Bolz, NVIDIA Corporation
- Daniel Koch, NVIDIA Corporation
- Dan Ginsburg, Valve Corporation
- Jeff Leger, QUALCOMM
- Michal Pietrasiuk, Intel
- Jan-Harald Fredriksen, Arm Limited

### Description

This extension adds flags to `Vk*PipelineCreateInfo` and VkPipelineCacheCreateInfo structures with the aim of improving the predictability of pipeline creation cost. The goal is to provide information about potentially expensive hazards within the client driver during pipeline creation to the application before carrying them out rather than after.

### Background

Pipeline creation is a costly operation, and the explicit nature of the Vulkan design means that cost is not hidden from the developer. Applications are also expected to schedule, prioritize, and load balance all calls for pipeline creation. It is strongly advised that applications create pipelines sufficiently ahead of their usage. Failure to do so will result in an unresponsive application, intermittent stuttering, or other poor user experiences. Proper usage of pipeline caches and/or derivative pipelines help mitigate this but is not assured to eliminate disruption in all cases. In the event that an ahead-of-time creation is not possible, considerations should be taken to ensure that the current execution context is suitable for the workload of pipeline creation including possible shader compilation.

Applications making API calls to create a pipeline must be prepared for any of the following to occur:

- OS/kernel calls to be made by the ICD
- Internal memory allocation not tracked by the `pAllocator` passed to `vkCreate*Pipelines`
- Internal thread synchronization or yielding of the current thread's core
- Extremely long (multi-millisecond+), blocking, compilation times
- Arbitrary call stacks depths and stack memory usage

The job or task based game engines that are being developed to take advantage of explicit graphics APIs like Vulkan may behave exceptionally poorly if any of the above scenarios occur. However, most game engines are already built to "stream" in assets dynamically as the user plays the game. By adding control by way of VkPipelineCreateFlags, we can require an ICD to report back a failure in critical execution paths rather than forcing an unexpected wait.

Applications can prevent unexpected compilation by setting `VK_PIPELINE_CREATE_FAIL_ON_PIPELINE_COMPILE_REQUIRED_BIT_EXT` on `Vk*PipelineCreateInfo`::`flags`. When set, an ICD must not attempt pipeline or shader compilation to create the pipeline object. In such a case, if the implementation fails to create a pipeline without compilation, the implementation **must** return the result `VK_PIPELINE_COMPILE_REQUIRED_EXT` and return VK_NULL_HANDLE for the pipeline.

By default `vkCreate*Pipelines` calls must attempt to create all pipelines before returning. Setting `VK_PIPELINE_CREATE_EARLY_RETURN_ON_FAILURE_BIT_EXT` on `Vk*PipelineCreateInfo`::`flags` can be used as an escape hatch for batched pipeline creates.

Hidden locks also add to the unpredictability of the cost of pipeline creation. The most common case of locks inside the `vkCreate*Pipelines` is internal synchronization of the VkPipelineCache object. `VK_PIPELINE_CACHE_CREATE_EXTERNALLY_SYNCHRONIZED_BIT_EXT` can be set when calling

vkCreatePipelineCache to state the cache is externally synchronized.

The hope is that armed with this information application and engine developers can leverage existing asset streaming systems to recover from "just-in-time" pipeline creation stalls.

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDevicePipelineCreationCacheControlFeaturesEXT

### New Enums

- VkPipelineCacheCreateFlagBits

### New Enum Constants

- `VK_EXT_PIPELINE_CREATION_CACHE_CONTROL_EXTENSION_NAME`
- `VK_EXT_PIPELINE_CREATION_CACHE_CONTROL_SPEC_VERSION`
- Extending VkPipelineCacheCreateFlagBits:
  - `VK_PIPELINE_CACHE_CREATE_EXTERNALLY_SYNCHRONIZED_BIT_EXT`
- Extending VkPipelineCreateFlagBits:
  - `VK_PIPELINE_CREATE_EARLY_RETURN_ON_FAILURE_BIT_EXT`
  - `VK_PIPELINE_CREATE_FAIL_ON_PIPELINE_COMPILE_REQUIRED_BIT_EXT`
- Extending VkResult:
  - `VK_ERROR_PIPELINE_COMPILE_REQUIRED_EXT`
  - `VK_PIPELINE_COMPILE_REQUIRED_EXT`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PIPELINE_CREATION_CACHE_CONTROL_FEATURES_EXT`

### Promotion to Vulkan 1.3

Functionality in this extension is included in core Vulkan 1.3, with the EXT suffix omitted. The original type, enum and command names are still available as aliases of the core functionality.

### Version History

- Revision 1, 2019-11-01 (Gregory Grebe)
  - Initial revision
- Revision 2, 2020-02-24 (Gregory Grebe)
  - Initial public revision
- Revision 3, 2020-03-23 (Tobias Hector)
  - Changed `VK_PIPELINE_COMPILE_REQUIRED_EXT` to a success code, adding an alias for the original `VK_ERROR_PIPELINE_COMPILE_REQUIRED_EXT`. Also updated the xml to include these codes as

return values.

## VK_EXT_pipeline_creation_feedback

### Name String

`VK_EXT_pipeline_creation_feedback`

### Extension Type

Device extension

### Registered Extension Number

193

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

None

### Deprecation State

- *Promoted* to Vulkan 1.3

### Special Use

- Developer tools

### Contact

- Jean-Francois Roy [GitHub]jfroy

### Other Extension Metadata

### Last Modified Date

2019-03-12

### IP Status

No known IP claims.

### Contributors

- Jean-Francois Roy, Google
- Hai Nguyen, Google
- Andrew Ellem, Google
- Bob Fraser, Google
- Sujeevan Rajayogam, Google
- Jan-Harald Fredriksen, ARM
- Jeff Leger, Qualcomm Technologies, Inc.

- Jeff Bolz, NVIDIA
- Daniel Koch, NVIDIA
- Neil Henning, AMD

#### Description

This extension adds a mechanism to provide feedback to an application about pipeline creation, with the specific goal of allowing a feedback loop between build systems and in-the-field application executions to ensure effective pipeline caches are shipped to customers.

#### New Structures

- VkPipelineCreationFeedbackEXT
- Extending VkGraphicsPipelineCreateInfo, VkComputePipelineCreateInfo, VkRayTracingPipelineCreateInfoNV, VkRayTracingPipelineCreateInfoKHR, VkExecutionGraphPipelineCreateInfoAMDX:
  - VkPipelineCreationFeedbackCreateInfoEXT

#### New Enums

- VkPipelineCreationFeedbackFlagBitsEXT

#### New Bitmasks

- VkPipelineCreationFeedbackFlagsEXT

#### New Enum Constants

- `VK_EXT_PIPELINE_CREATION_FEEDBACK_EXTENSION_NAME`
- `VK_EXT_PIPELINE_CREATION_FEEDBACK_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PIPELINE_CREATION_FEEDBACK_CREATE_INFO_EXT`

#### Promotion to Vulkan 1.3

Functionality in this extension is included in core Vulkan 1.3, with the EXT suffix omitted. The original type, enum and command names are still available as aliases of the core functionality.

#### Version History

- Revision 1, 2019-03-12 (Jean-Francois Roy)
  - Initial revision

## VK_EXT_pipeline_library_group_handles

#### Name String

`VK_EXT_pipeline_library_group_handles`

**Extension Type**

Device extension

**Registered Extension Number**

499

**Revision**

1

**Ratification Status**

Not ratified

**Extension and Version Dependencies**

VK_KHR_ray_tracing_pipeline
and
VK_KHR_pipeline_library

**Contact**

- Hans-Kristian Arntzen [GitHub]HansKristian-Work

**Extension Proposal**

VK_EXT_pipeline_library_group_handles

**Other Extension Metadata**

**Last Modified Date**

2023-01-25

**IP Status**

No known IP claims.

**Contributors**

- Hans-Kristian Arntzen, Valve
- Stuart Smith, AMD
- Ricardo Garcia, Igalia
- Lionel Landwerlin, Intel
- Eric Werness, NVIDIA
- Daniel Koch, NVIDIA

**Description**

When using pipeline libraries in ray tracing pipelines, a library might get linked into different pipelines in an incremental way. An application can have a strategy where a ray tracing pipeline is comprised of N pipeline libraries and is later augmented by creating a new pipeline with N + 1 libraries. Without this extension, all group handles must be re-queried as the group handle is tied to the pipeline, not the library. This is problematic for applications which aim to decouple construction of record buffers and the linkage of ray tracing pipelines.

To aid in this, this extension enables support for querying group handles directly from pipeline libraries. Group handles obtained from a library **must** remain bitwise identical in any `VkPipeline` that links to the library.

With this feature, the extension also improves compatibility with DXR 1.1 AddToStateObject(), which guarantees that group handles returned remain bitwise identical between parent and child pipelines. In addition, querying group handles from COLLECTION objects is also supported with that API.

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDevicePipelineLibraryGroupHandlesFeaturesEXT

### New Enum Constants

- `VK_EXT_PIPELINE_LIBRARY_GROUP_HANDLES_EXTENSION_NAME`
- `VK_EXT_PIPELINE_LIBRARY_GROUP_HANDLES_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PIPELINE_LIBRARY_GROUP_HANDLES_FEATURES_EXT`

### Version History

- Revision 1, 2023-01-25 (Hans-Kristian Arntzen)
  - Initial draft

## VK_EXT_pipeline_properties

### Name String

`VK_EXT_pipeline_properties`

### Extension Type

Device extension

### Registered Extension Number

373

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### Contact

- Mukund Keshava [GitHub]mkeshavanv

### Other Extension Metadata

### Last Modified Date

2022-04-19

### IP Status

No known IP claims.

### Contributors

- Mukund Keshava, NVIDIA
- Daniel Koch, NVIDIA
- Mark Bellamy, Arm

### Description

Vulkan SC requires offline compilation of pipelines. In order to support this, the pipeline state is represented in a JSON schema that is read by an offline tool for compilation.

One method of developing a Vulkan SC application is to author a Vulkan application and use a layer to record and serialize the pipeline state and shaders for offline compilation. Each pipeline is represented by a separate JSON file, and can be identified with a `pipelineIdentifier`.

Once the pipelines have been compiled by the offline pipeline cache compiler, the Vulkan SC application can then use this `pipelineIdentifier` for identifying the pipeline via Vulkan SC's `VkPipelineIdentifierInfo` structure.

This extension allows the Vulkan application to query the `pipelineIdentifier` associated with each pipeline so that the application can store this with its pipeline metadata and the Vulkan SC application will then use to map the same state to an entry in the Vulkan SC pipeline cache.

It is expected that this extension will initially be implemented in the json generation layer, although we can envision that there might be future uses for it in native Vulkan drivers as well.

### New Commands

- vkGetPipelinePropertiesEXT

### New Structures

- VkPipelineInfoEXT
- VkPipelinePropertiesIdentifierEXT
- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDevicePipelinePropertiesFeaturesEXT

### New Enum Constants

- `VK_EXT_PIPELINE_PROPERTIES_EXTENSION_NAME`
- `VK_EXT_PIPELINE_PROPERTIES_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PIPELINE_PROPERTIES_FEATURES_EXT`
  - `VK_STRUCTURE_TYPE_PIPELINE_INFO_EXT`
  - `VK_STRUCTURE_TYPE_PIPELINE_PROPERTIES_IDENTIFIER_EXT`

### Issues

(1) This extension does not make sense on a strict Vulkan SC implementation. It may however be of potential use in a non-strict Vulkan SC implementation. Should this extension be enabled as part of Vulkan SC as well?

**RESOLVED**: No. This extension will not be enabled for Vulkan SC.

(2) This is intended to be a general pipeline properties query, but is currently only retrieving the pipeline identifier. Should the pipeline identifier query be mandatory for this extension and for all queries using this command?

**RESOLVED**: Use VkBaseOutStructure for the return parameter. Currently this is required to actually be a VkPipelinePropertiesIdentifierEXT structure, but that could be relaxed in the future to allow other structure types or to allow other structures to be chained in along with this one.

(3) Should there be a feature structure? Should it be required?

**RESOLVED**: Add a feature structure, and a feature for querying pipeline identifier, but allow it to be optional so that this extension can be used as the basis for other pipeline property queries without requiring the pipeline identifier to be supported.

### Version History

- Revision 1, 2022-04-19 (Mukund Keshava, Daniel Koch)
  - Initial draft

## VK_EXT_pipeline_protected_access

### Name String

`VK_EXT_pipeline_protected_access`

### Extension Type

Device extension

### Registered Extension Number

467

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### Contact

- Shahbaz Youssefi [GitHub]syoussefi

### Extension Proposal

VK_EXT_pipeline_protected_access

### Other Extension Metadata

### Last Modified Date

2022-07-28

### Contributors

- Shahbaz Youssefi, Google
- Jörg Wagner, Arm
- Ralph Potter, Samsung
- Daniel Koch, NVIDIA

### Description

This extension allows protected memory access to be specified per pipeline as opposed to per device. Through the usage of this extension, any performance penalty paid due to access to protected memory will be limited to the specific pipelines that make such accesses.

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDevicePipelineProtectedAccessFeaturesEXT

### New Enum Constants

- `VK_EXT_PIPELINE_PROTECTED_ACCESS_EXTENSION_NAME`
- `VK_EXT_PIPELINE_PROTECTED_ACCESS_SPEC_VERSION`
- Extending VkPipelineCreateFlagBits:
  - `VK_PIPELINE_CREATE_NO_PROTECTED_ACCESS_BIT_EXT`
  - `VK_PIPELINE_CREATE_PROTECTED_ACCESS_ONLY_BIT_EXT`

- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PIPELINE_PROTECTED_ACCESS_FEATURES_EXT`

### Version History

- Revision 1, 2022-07-28 (Shahbaz Youssefi)
  - Internal revisions

## VK_EXT_pipeline_robustness

### Name String

`VK_EXT_pipeline_robustness`

### Extension Type

Device extension

### Registered Extension Number

69

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### Contact

- Jarred Davies

### Other Extension Metadata

### Last Modified Date

2022-07-12

### Interactions and External Dependencies

- Interacts with `VK_EXT_robustness2`
- Interacts with `VK_EXT_image_robustness`
- Interacts with `VK_KHR_ray_tracing_pipeline`

### Contributors

- Jarred Davies, Imagination Technologies
- Alex Walters, Imagination Technologies

- Piers Daniell, NVIDIA
- Graeme Leese, Broadcom Corporation
- Jeff Leger, Qualcomm Technologies, Inc.
- Faith Ekstrand, Intel
- Lionel Landwerlin, Intel
- Shahbaz Youssefi, Google, Inc.

### Description

This extension allows users to request robustness on a per-pipeline stage basis.

As `robustBufferAccess` and other robustness features may have an adverse effect on performance, this extension is designed to allow users to request robust behavior only where it may be needed.

### New Structures

- Extending VkGraphicsPipelineCreateInfo, VkComputePipelineCreateInfo, VkPipelineShaderStageCreateInfo, VkRayTracingPipelineCreateInfoKHR:
  - VkPipelineRobustnessCreateInfoEXT
- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDevicePipelineRobustnessFeaturesEXT
- Extending VkPhysicalDeviceProperties2:
  - VkPhysicalDevicePipelineRobustnessPropertiesEXT

### New Enums

- VkPipelineRobustnessBufferBehaviorEXT
- VkPipelineRobustnessImageBehaviorEXT

### New Enum Constants

- `VK_EXT_PIPELINE_ROBUSTNESS_EXTENSION_NAME`
- `VK_EXT_PIPELINE_ROBUSTNESS_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PIPELINE_ROBUSTNESS_FEATURES_EXT`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PIPELINE_ROBUSTNESS_PROPERTIES_EXT`
  - `VK_STRUCTURE_TYPE_PIPELINE_ROBUSTNESS_CREATE_INFO_EXT`

### Version History

- Revision 1, 2022-07-12 (Jarred Davies)
  - Initial version

## VK_EXT_post_depth_coverage

### Name String

`VK_EXT_post_depth_coverage`

### Extension Type

Device extension

### Registered Extension Number

156

### Revision

1

### Ratification Status

Ratified

### Extension and Version Dependencies

None

### SPIR-V Dependencies

- SPV_KHR_post_depth_coverage

### Contact

- Daniel Koch [GitHub]dgkoch

### Other Extension Metadata

### Last Modified Date

2017-07-17

### Interactions and External Dependencies

- This extension provides API support for `GL_ARB_post_depth_coverage` and `GL_EXT_post_depth_coverage`

### Contributors

- Jeff Bolz, NVIDIA

### Description

This extension adds support for the following SPIR-V extension in Vulkan:

- `SPV_KHR_post_depth_coverage`

which allows the fragment shader to control whether values in the `SampleMask` built-in input variable reflect the coverage after early depth and stencil tests are applied.

This extension adds a new `PostDepthCoverage` execution mode under the `SampleMaskPostDepthCoverage` capability. When this mode is specified along with `EarlyFragmentTests`, the value of an input variable decorated with the `SampleMask` built-in reflects the coverage after the

early fragment tests are applied. Otherwise, it reflects the coverage before the depth and stencil tests.

When using GLSL source-based shading languages, the `post_depth_coverage` layout qualifier from GL_ARB_post_depth_coverage or GL_EXT_post_depth_coverage maps to the `PostDepthCoverage` execution mode.

#### New Enum Constants

- `VK_EXT_POST_DEPTH_COVERAGE_EXTENSION_NAME`
- `VK_EXT_POST_DEPTH_COVERAGE_SPEC_VERSION`

#### New SPIR-V Capabilities

- `SampleMaskPostDepthCoverage`

#### Version History

- Revision 1, 2017-07-17 (Daniel Koch)
  - Internal revisions

## VK_EXT_primitive_topology_list_restart

#### Name String

`VK_EXT_primitive_topology_list_restart`

#### Extension Type

Device extension

#### Registered Extension Number

357

#### Revision

1

#### Ratification Status

Ratified

#### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

#### Special Use

- OpenGL / ES support

#### Contact

- Shahbaz Youssefi [GitHub]syoussefi

### Other Extension Metadata

#### Last Modified Date

2021-01-11

#### IP Status

No known IP claims.

#### Contributors

- Courtney Goeltzenleuchter, Google
- Shahbaz Youssefi, Google

#### Description

This extension allows list primitives to use the primitive restart index value. This provides a more efficient implementation when layering OpenGL functionality on Vulkan by avoiding emulation which incurs data copies.

#### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDevicePrimitiveTopologyListRestartFeaturesEXT

#### New Enum Constants

- `VK_EXT_PRIMITIVE_TOPOLOGY_LIST_RESTART_EXTENSION_NAME`
- `VK_EXT_PRIMITIVE_TOPOLOGY_LIST_RESTART_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PRIMITIVE_TOPOLOGY_LIST_RESTART_FEATURES_EXT`

#### Version History

- Revision 0, 2020-09-14 (Courtney Goeltzenleuchter)
  - Internal revisions
- Revision 1, 2021-01-11 (Shahbaz Youssefi)
  - Add the `primitiveTopologyPatchListRestart` feature
  - Internal revisions

## VK_EXT_primitives_generated_query

#### Name String

`VK_EXT_primitives_generated_query`

#### Extension Type

Device extension

### Registered Extension Number

383

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_EXT_transform_feedback

### Special Use

- OpenGL / ES support

### Contact

- Shahbaz Youssefi [GitHub]syoussefi

### Extension Proposal

VK_EXT_primitives_generated_query

### Other Extension Metadata

### Last Modified Date

2022-01-24

### Contributors

- Shahbaz Youssefi, Google
- Piers Daniell, NVIDIA
- Faith Ekstrand, Collabora
- Jan-Harald Fredriksen, Arm

### Description

This extension adds support for a new query type to match OpenGL's `GL_PRIMITIVES_GENERATED` to support layering.

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDevicePrimitivesGeneratedQueryFeaturesEXT

### New Enum Constants

- `VK_EXT_PRIMITIVES_GENERATED_QUERY_EXTENSION_NAME`
- `VK_EXT_PRIMITIVES_GENERATED_QUERY_SPEC_VERSION`
- Extending VkQueryType:

- `VK_QUERY_TYPE_PRIMITIVES_GENERATED_EXT`

- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PRIMITIVES_GENERATED_QUERY_FEATURES_EXT`

**Issues**

1) Can the query from `VK_EXT_transform_feedback` be used instead?

**RESOLVED**: No. While the query from VK_EXT_transform_feedback can produce the same results as in this extension, it is only available while transform feedback is active. The OpenGL `GL_PRIMITIVES_GENERATED` query is independent from transform feedback. Emulation through artificial transform feedback is unnecessarily inefficient.

2) Can `VK_QUERY_PIPELINE_STATISTIC_CLIPPING_INVOCATIONS_BIT` be used instead?

**RESOLVED**: It could, but we prefer the extension for simplicity. Vulkan requires that only one query be active at a time. If both the `GL_PRIMITIVES_GENERATED` and the `GL_CLIPPING_INPUT_PRIMITIVES_ARB` queries need to be simultaneously enabled, emulation of both through `VK_QUERY_PIPELINE_STATISTIC_CLIPPING_INVOCATIONS_BIT` is inconvenient.

3) On some hardware, this query cannot be implemented if `VkPipelineRasterizationStateCreateInfo::rasterizerDiscardEnable` is enabled. How will this be handled?

**RESOLVED**: A feature flag is exposed by this extension for this. On said hardware, the GL implementation disables rasterizer-discard and achieves the same effect through other means. It will not be able to do the same in Vulkan due to lack of state information. A feature flag is exposed by this extension so the OpenGL implementation on top of Vulkan would be able to implement a similar workaround.

4) On some hardware, this query cannot be implemented for non-zero query indices. How will this be handled?

**RESOLVED**: A feature flag is exposed by this extension for this. If this feature is not present, the query from `VK_EXT_transform_feedback` can be used to the same effect.

5) How is the interaction of this extension with `transformFeedbackRasterizationStreamSelect` handled?

**RESOLVED**: Disallowed for non-zero streams. In OpenGL, the rasterization stream is always stream zero.

**Version History**

- Revision 1, 2021-06-23 (Shahbaz Youssefi)
  - Internal revisions

## VK_EXT_private_data

### Name String

`VK_EXT_private_data`

### Extension Type

Device extension

### Registered Extension Number

296

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### Deprecation State

- *Promoted* to Vulkan 1.3

### Contact

- Matthew Rusch [GitHub]mattruschnv

### Other Extension Metadata

### Last Modified Date

2020-03-25

### IP Status

No known IP claims.

### Contributors

- Matthew Rusch, NVIDIA
- Nuno Subtil, NVIDIA
- Piers Daniell, NVIDIA
- Jeff Bolz, NVIDIA

### Description

This extension is a device extension which enables attaching arbitrary payloads to Vulkan objects. It introduces the idea of private data slots as a means of storing a 64-bit unsigned integer of application-defined data. Private data slots can be created or destroyed any time an associated device is available. Private data slots can be reserved at device creation time, and limiting use to

the amount reserved will allow the extension to exhibit better performance characteristics.

### New Object Types

- VkPrivateDataSlotEXT

### New Commands

- vkCreatePrivateDataSlotEXT
- vkDestroyPrivateDataSlotEXT
- vkGetPrivateDataEXT
- vkSetPrivateDataEXT

### New Structures

- VkPrivateDataSlotCreateInfoEXT
- Extending VkDeviceCreateInfo:
  - VkDevicePrivateDataCreateInfoEXT
- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDevicePrivateDataFeaturesEXT

### New Bitmasks

- VkPrivateDataSlotCreateFlagsEXT

### New Enum Constants

- `VK_EXT_PRIVATE_DATA_EXTENSION_NAME`
- `VK_EXT_PRIVATE_DATA_SPEC_VERSION`
- Extending VkObjectType:
  - `VK_OBJECT_TYPE_PRIVATE_DATA_SLOT_EXT`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_DEVICE_PRIVATE_DATA_CREATE_INFO_EXT`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PRIVATE_DATA_FEATURES_EXT`
  - `VK_STRUCTURE_TYPE_PRIVATE_DATA_SLOT_CREATE_INFO_EXT`

### Promotion to Vulkan 1.3

Functionality in this extension is included in core Vulkan 1.3, with the EXT suffix omitted. The original type, enum and command names are still available as aliases of the core functionality.

### Examples

- In progress

### Issues

(1) If I have to create a VkPrivateDataSlot to store and retrieve data on an object, how does this extension help me? Will I not need to store the VkPrivateDataSlot mapping with each object, and if I am doing that, I might as well just store the original data!

**RESOLVED:** The VkPrivateDataSlot can be thought of as an opaque index into storage that is reserved in each object. That is, you can use the same VkPrivateDataSlot with each object for a specific piece of information. For example, if a layer wishes to track per-object information, the layer only needs to allocate one VkPrivateDataSlot per device and it can use that private data slot for all of the device's child objects. This allows multiple layers to store private data without conflicting with each other's and/or the application's private data.

(2) What if I need to store more than 64-bits of information per object?

**RESOLVED:** The data that you store per object could be a pointer to another object or structure of your own allocation.

### Version History

- Revision 1, 2020-01-15 (Matthew Rusch)
  - Initial draft

## VK_EXT_provoking_vertex

### Name String

`VK_EXT_provoking_vertex`

### Extension Type

Device extension

### Registered Extension Number

255

### Revision

1

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### Special Use

- OpenGL / ES support

### Contact

- Jesse Hall [GitHub]jessehall

### Other Extension Metadata

### Last Modified Date

2021-02-22

### IP Status

No known IP claims.

### Contributors

- Alexis Hétu, Google
- Bill Licea-Kane, Qualcomm
- Daniel Koch, Nvidia
- Jamie Madill, Google
- Jan-Harald Fredriksen, Arm
- Faith Ekstrand, Intel
- Jeff Bolz, Nvidia
- Jeff Leger, Qualcomm
- Jesse Hall, Google
- Jörg Wagner, Arm
- Matthew Netsch, Qualcomm
- Mike Blumenkrantz, Valve
- Piers Daniell, Nvidia
- Tobias Hector, AMD

### Description

This extension allows changing the provoking vertex convention between Vulkan's default convention (first vertex) and OpenGL's convention (last vertex).

This extension is intended for use by API-translation layers that implement APIs like OpenGL on top of Vulkan, and need to match the source API's provoking vertex convention. Applications using Vulkan directly should use Vulkan's default convention.

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceProvokingVertexFeaturesEXT
- Extending VkPhysicalDeviceProperties2:
  - VkPhysicalDeviceProvokingVertexPropertiesEXT

- Extending VkPipelineRasterizationStateCreateInfo:
  - VkPipelineRasterizationProvokingVertexStateCreateInfoEXT

### New Enums

- VkProvokingVertexModeEXT

### New Enum Constants

- `VK_EXT_PROVOKING_VERTEX_EXTENSION_NAME`
- `VK_EXT_PROVOKING_VERTEX_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PROVOKING_VERTEX_FEATURES_EXT`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PROVOKING_VERTEX_PROPERTIES_EXT`
  - `VK_STRUCTURE_TYPE_PIPELINE_RASTERIZATION_PROVOKING_VERTEX_STATE_CREATE_INFO_EXT`

### Issues

1) At what granularity should this state be set?

**RESOLVED**: At pipeline bind, with an optional per-render pass restriction.

The most natural place to put this state is in the graphics pipeline object. Some implementations require it to be known when creating the pipeline, and pipeline state is convenient for implementing OpenGL 3.2's glProvokingVertex, which can change the state between draw calls. However, some implementations can only change it approximately render pass granularity. To accommodate both, provoking vertex will be pipeline state, but implementations can require that only one mode is used within a render pass instance; the render pass's mode is chosen implicitly when the first pipeline is bound.

2) Does the provoking vertex mode affect the order that vertices are written to transform feedback buffers?

**RESOLVED**: Yes, to enable layered implementations of OpenGL and D3D.

All of OpenGL, OpenGL ES, and Direct3D 11 require that vertices are written to transform feedback buffers such that flat-shaded attributes have the same value when drawing the contents of the transform feedback buffer as they did in the original drawing when the transform feedback buffer was written (assuming the provoking vertex mode has not changed, in APIs that support more than one mode).

### Version History

- Revision 1, (1c) 2021-02-22 (Jesse Hall)
  - Added VkPhysicalDeviceProvokingVertexPropertiesEXT::transformFeedbackPreservesTriangleFanProvokingVertex to accommodate implementations that cannot change the transform

feedback vertex order for triangle fans.

- Revision 1, (1b) 2020-06-14 (Jesse Hall)
  - Added VkPhysicalDeviceProvokingVertexFeaturesEXT::transformFeedbackPreservesProvokingVertex and required that transform feedback write vertices so as to preserve the provoking vertex of each primitive.
- Revision 1, (1a) 2019-10-23 (Jesse Hall)
  - Initial draft, based on a proposal by Alexis Hétu

## VK_EXT_queue_family_foreign

### Name String

`VK_EXT_queue_family_foreign`

### Extension Type

Device extension

### Registered Extension Number

127

### Revision

1

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_external_memory
or
Vulkan Version 1.1

### Contact

- Lina Versace [GitHub]versalinyaa

### Other Extension Metadata

### Last Modified Date

2017-11-01

### IP Status

No known IP claims.

### Contributors

- Lina Versace, Google
- James Jones, NVIDIA
- Faith Ekstrand, Intel

- Jesse Hall, Google
- Daniel Rakos, AMD
- Ray Smith, ARM

#### Description

This extension defines a special queue family, `VK_QUEUE_FAMILY_FOREIGN_EXT`, which can be used to transfer ownership of resources backed by external memory to foreign, external queues. This is similar to `VK_QUEUE_FAMILY_EXTERNAL_KHR`, defined in `VK_KHR_external_memory`. The key differences between the two are:

- The queues represented by `VK_QUEUE_FAMILY_EXTERNAL_KHR` must share the same physical device and the same driver version as the current VkInstance. `VK_QUEUE_FAMILY_FOREIGN_EXT` has no such restrictions. It can represent devices and drivers from other vendors, and can even represent non-Vulkan-capable devices.
- All resources backed by external memory support `VK_QUEUE_FAMILY_EXTERNAL_KHR`. Support for `VK_QUEUE_FAMILY_FOREIGN_EXT` is more restrictive.
- Applications should expect transitions to/from `VK_QUEUE_FAMILY_FOREIGN_EXT` to be more expensive than transitions to/from `VK_QUEUE_FAMILY_EXTERNAL_KHR`.

#### New Enum Constants

- `VK_EXT_QUEUE_FAMILY_FOREIGN_EXTENSION_NAME`
- `VK_EXT_QUEUE_FAMILY_FOREIGN_SPEC_VERSION`
- `VK_QUEUE_FAMILY_FOREIGN_EXT`

#### Version History

- Revision 1, 2017-11-01 (Lina Versace)
  - Squashed internal revisions

## VK_EXT_rasterization_order_attachment_access

#### Name String

`VK_EXT_rasterization_order_attachment_access`

#### Extension Type

Device extension

#### Registered Extension Number

464

#### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### Contact

- Jan-Harald Fredriksen [GitHub]janharaldfredriksen-arm

### Extension Proposal

VK_EXT_rasterization_order_attachment_access

### Other Extension Metadata

### Last Modified Date

2022-07-04

### IP Status

No known IP claims.

### Contributors

- Tobias Hector, AMD
- Jan-Harald Fredriksen, Arm

### Description

This extension extends the mechanism of input attachments to allow access to framebuffer attachments that are used both as input and as color or depth/stencil attachments from one fragment to the next, in rasterization order, without explicit synchronization.

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceRasterizationOrderAttachmentAccessFeaturesEXT

### New Enums

- VkPipelineColorBlendStateCreateFlagBits
- VkPipelineDepthStencilStateCreateFlagBits

### New Enum Constants

- `VK_EXT_RASTERIZATION_ORDER_ATTACHMENT_ACCESS_EXTENSION_NAME`
- `VK_EXT_RASTERIZATION_ORDER_ATTACHMENT_ACCESS_SPEC_VERSION`
- Extending VkPipelineColorBlendStateCreateFlagBits:

    - VK_PIPELINE_COLOR_BLEND_STATE_CREATE_RASTERIZATION_ORDER_ATTACHMENT_ACCESS_BIT_EXT
- Extending VkPipelineDepthStencilStateCreateFlagBits:
    - VK_PIPELINE_DEPTH_STENCIL_STATE_CREATE_RASTERIZATION_ORDER_ATTACHMENT_DEPTH_ACCESS_BIT_EXT
    - VK_PIPELINE_DEPTH_STENCIL_STATE_CREATE_RASTERIZATION_ORDER_ATTACHMENT_STENCIL_ACCESS_BIT_EXT
- Extending VkStructureType:
    - VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_RASTERIZATION_ORDER_ATTACHMENT_ACCESS_FEATURES_EXT
- Extending VkSubpassDescriptionFlagBits:
    - VK_SUBPASS_DESCRIPTION_RASTERIZATION_ORDER_ATTACHMENT_COLOR_ACCESS_BIT_EXT
    - VK_SUBPASS_DESCRIPTION_RASTERIZATION_ORDER_ATTACHMENT_DEPTH_ACCESS_BIT_EXT
    - VK_SUBPASS_DESCRIPTION_RASTERIZATION_ORDER_ATTACHMENT_STENCIL_ACCESS_BIT_EXT

### Examples

None.

### Version History

- Revision 1, 2022-07-04 (Jan-Harald Fredriksen)
    - Initial draft

## VK_EXT_rgba10x6_formats

### Name String

VK_EXT_rgba10x6_formats

### Extension Type

Device extension

### Registered Extension Number

345

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_sampler_ycbcr_conversion
or
Vulkan Version 1.1

### Contact

- Jan-Harald Fredriksen [GitHub]janharaldfredriksen-arm

### Other Extension Metadata

### Last Modified Date

2021-09-29

### IP Status

No known IP claims.

### Contributors

- Jan-Harald Fredriksen, Arm
- Graeme Leese, Broadcom
- Spencer Fricke, Samsung

### Description

This extension enables the `VK_FORMAT_R10X6G10X6B10X6A10X6_UNORM_4PACK16` format to be used without a sampler $Y'C_BC_R$ conversion enabled.

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceRGBA10X6FormatsFeaturesEXT

### New Enum Constants

- `VK_EXT_RGBA10X6_FORMATS_EXTENSION_NAME`
- `VK_EXT_RGBA10X6_FORMATS_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_RGBA10X6_FORMATS_FEATURES_EXT`

### Issues

1) Should we reuse the existing format enumeration or introduce a new one?

**RESOLVED**: We reuse an existing format enumeration, `VK_FORMAT_R10X6G10X6B10X6A10X6_UNORM_4PACK16`, that was previously exclusively used for YCbCr and therefore had a set of limitations related to that usage. The alternative was to introduce a new format token with exactly the same bit representation as the existing token, but without the limitations.

2) Should we only introduce `VK_FORMAT_R10X6G10X6B10X6A10X6_UNORM_4PACK16` or also 1-3 component variations?

**RESOLVED**: Only the 4-component format is introduced because the 1- and 2- component

variations are already not exclusive to YCbCr, and the 3-component variation is not a good match for hardware capabilities.

### Version History

- Revision 1, 2021-09-29 (Jan-Harald Fredriksen)
  - Initial EXT version

## VK_EXT_robustness2

### Name String

`VK_EXT_robustness2`

### Extension Type

Device extension

### Registered Extension Number

287

### Revision

1

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### Contact

- Liam Middlebrook [GitHub]liam-middlebrook

### Other Extension Metadata

### Last Modified Date

2020-01-29

### IP Status

No known IP claims.

### Contributors

- Liam Middlebrook, NVIDIA
- Jeff Bolz, NVIDIA

### Description

This extension adds stricter requirements for how out of bounds reads and writes are handled.

Most accesses **must** be tightly bounds-checked, out of bounds writes **must** be discarded, out of bound reads **must** return zero. Rather than allowing multiple possible (0,0,0,x) vectors, the out of bounds values are treated as zero, and then missing components are inserted based on the format as described in Conversion to RGBA and vertex input attribute extraction.

These additional requirements **may** be expensive on some implementations, and should only be enabled when truly necessary.

This extension also adds support for "null descriptors", where VK_NULL_HANDLE **can** be used instead of a valid handle. Accesses to null descriptors have well-defined behavior, and do not rely on robustness.

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceRobustness2FeaturesEXT
- Extending VkPhysicalDeviceProperties2:
  - VkPhysicalDeviceRobustness2PropertiesEXT

### New Enum Constants

- `VK_EXT_ROBUSTNESS_2_EXTENSION_NAME`
- `VK_EXT_ROBUSTNESS_2_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_ROBUSTNESS_2_FEATURES_EXT`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_ROBUSTNESS_2_PROPERTIES_EXT`

### Issues

1. Why do VkPhysicalDeviceRobustness2PropertiesEXT::`robustUniformBufferAccessSizeAlignment` and VkPhysicalDeviceRobustness2PropertiesEXT::`robustStorageBufferAccessSizeAlignment` exist?

**RESOLVED**: Some implementations cannot efficiently tightly bounds-check all buffer accesses. Rather, the size of the bound range is padded to some power of two multiple, up to 256 bytes for uniform buffers and up to 4 bytes for storage buffers, and that padded size is bounds-checked. This is sufficient to implement D3D-like behavior, because D3D only allows binding whole uniform buffers or ranges that are a multiple of 256 bytes, and D3D raw and structured buffers only support 32-bit accesses.

### Examples

None.

### Version History

- Revision 1, 2019-11-01 (Jeff Bolz, Liam Middlebrook)
  - Initial draft

## VK_EXT_sample_locations

### Name String

`VK_EXT_sample_locations`

### Extension Type

Device extension

### Registered Extension Number

144

### Revision

1

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### Contact

- Daniel Rakos [GitHub]drakos-amd

### Other Extension Metadata

### Last Modified Date

2017-08-02

### Contributors

- Mais Alnasser, AMD
- Matthaeus G. Chajdas, AMD
- Maciej Jesionowski, AMD
- Daniel Rakos, AMD
- Slawomir Grajewski, Intel
- Jeff Bolz, NVIDIA
- Bill Licea-Kane, Qualcomm

### Description

This extension allows an application to modify the locations of samples within a pixel used in rasterization. Additionally, it allows applications to specify different sample locations for each pixel in a group of adjacent pixels, which **can** increase antialiasing quality (particularly if a custom resolve shader is used that takes advantage of these different locations).

It is common for implementations to optimize the storage of depth values by storing values that **can** be used to reconstruct depth at each sample location, rather than storing separate depth values for each sample. For example, the depth values from a single triangle **may** be represented using plane equations. When the depth value for a sample is needed, it is automatically evaluated at the sample location. Modifying the sample locations causes the reconstruction to no longer evaluate the same depth values as when the samples were originally generated, thus the depth aspect of a depth/stencil attachment **must** be cleared before rendering to it using different sample locations.

Some implementations **may** need to evaluate depth image values while performing image layout transitions. To accommodate this, instances of the VkSampleLocationsInfoEXT structure **can** be specified for each situation where an explicit or automatic layout transition has to take place. VkSampleLocationsInfoEXT **can** be chained from VkImageMemoryBarrier structures to provide sample locations for layout transitions performed by vkCmdWaitEvents and vkCmdPipelineBarrier calls, and VkRenderPassSampleLocationsBeginInfoEXT **can** be chained from VkRenderPassBeginInfo to provide sample locations for layout transitions performed implicitly by a render pass instance.

### New Commands

- vkCmdSetSampleLocationsEXT
- vkGetPhysicalDeviceMultisamplePropertiesEXT

### New Structures

- VkAttachmentSampleLocationsEXT
- VkMultisamplePropertiesEXT
- VkSampleLocationEXT
- VkSubpassSampleLocationsEXT
- Extending VkImageMemoryBarrier, VkImageMemoryBarrier2:
  - VkSampleLocationsInfoEXT
- Extending VkPhysicalDeviceProperties2:
  - VkPhysicalDeviceSampleLocationsPropertiesEXT
- Extending VkPipelineMultisampleStateCreateInfo:
  - VkPipelineSampleLocationsStateCreateInfoEXT
- Extending VkRenderPassBeginInfo:
  - VkRenderPassSampleLocationsBeginInfoEXT

### New Enum Constants

- `VK_EXT_SAMPLE_LOCATIONS_EXTENSION_NAME`
- `VK_EXT_SAMPLE_LOCATIONS_SPEC_VERSION`
- Extending VkDynamicState:
  - `VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT`

- Extending VkImageCreateFlagBits:
  - `VK_IMAGE_CREATE_SAMPLE_LOCATIONS_COMPATIBLE_DEPTH_BIT_EXT`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_MULTISAMPLE_PROPERTIES_EXT`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SAMPLE_LOCATIONS_PROPERTIES_EXT`
  - `VK_STRUCTURE_TYPE_PIPELINE_SAMPLE_LOCATIONS_STATE_CREATE_INFO_EXT`
  - `VK_STRUCTURE_TYPE_RENDER_PASS_SAMPLE_LOCATIONS_BEGIN_INFO_EXT`
  - `VK_STRUCTURE_TYPE_SAMPLE_LOCATIONS_INFO_EXT`

#### Version History

- Revision 1, 2017-08-02 (Daniel Rakos)
  - Internal revisions

## VK_EXT_sampler_filter_minmax

#### Name String

`VK_EXT_sampler_filter_minmax`

#### Extension Type

Device extension

#### Registered Extension Number

131

#### Revision

2

#### Ratification Status

Not ratified

#### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

#### Deprecation State

- *Promoted* to Vulkan 1.2

#### Contact

- Jeff Bolz [GitHub]jeffbolznv

#### Other Extension Metadata

#### Last Modified Date

2017-05-19

### IP Status

No known IP claims.

### Contributors

- Jeff Bolz, NVIDIA
- Piers Daniell, NVIDIA

### Description

In unextended Vulkan, minification and magnification filters such as LINEAR allow sampled image lookups to return a filtered texel value produced by computing a weighted average of a collection of texels in the neighborhood of the texture coordinate provided.

This extension provides a new sampler parameter which allows applications to produce a filtered texel value by computing a component-wise minimum (MIN) or maximum (MAX) of the texels that would normally be averaged. The reduction mode is orthogonal to the minification and magnification filter parameters. The filter parameters are used to identify the set of texels used to produce a final filtered value; the reduction mode identifies how these texels are combined.

### Promotion to Vulkan 1.2

All functionality in this extension is included in core Vulkan 1.2, with the EXT suffix omitted. The original type, enum and command names are still available as aliases of the core functionality.

### New Structures

- Extending VkPhysicalDeviceProperties2:
  - VkPhysicalDeviceSamplerFilterMinmaxPropertiesEXT
- Extending VkSamplerCreateInfo:
  - VkSamplerReductionModeCreateInfoEXT

### New Enums

- VkSamplerReductionModeEXT

### New Enum Constants

- `VK_EXT_SAMPLER_FILTER_MINMAX_EXTENSION_NAME`
- `VK_EXT_SAMPLER_FILTER_MINMAX_SPEC_VERSION`
- Extending VkFormatFeatureFlagBits:
  - `VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT_EXT`
- Extending VkSamplerReductionMode:
  - `VK_SAMPLER_REDUCTION_MODE_MAX_EXT`
  - `VK_SAMPLER_REDUCTION_MODE_MIN_EXT`
  - `VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE_EXT`

- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SAMPLER_FILTER_MINMAX_PROPERTIES_EXT`
  - `VK_STRUCTURE_TYPE_SAMPLER_REDUCTION_MODE_CREATE_INFO_EXT`

### Version History

- Revision 2, 2017-05-19 (Piers Daniell)
  - Renamed to EXT
- Revision 1, 2017-03-25 (Jeff Bolz)
  - Internal revisions

## VK_EXT_scalar_block_layout

### Name String

`VK_EXT_scalar_block_layout`

### Extension Type

Device extension

### Registered Extension Number

222

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### Deprecation State

- *Promoted* to Vulkan 1.2

### Contact

- Tobias Hector [GitHub]tobski

### Other Extension Metadata

### Last Modified Date

2018-11-14

### Contributors

- Jeff Bolz

- Jan-Harald Fredriksen
- Graeme Leese
- Faith Ekstrand
- John Kessenich

### Description

This extension enables C-like structure layout for SPIR-V blocks. It modifies the alignment rules for uniform buffers, storage buffers and push constants, allowing non-scalar types to be aligned solely based on the size of their components, without additional requirements.

### Promotion to Vulkan 1.2

Functionality in this extension is included in core Vulkan 1.2, with the EXT suffix omitted. However, if Vulkan 1.2 is supported and this extension is not, the `scalarBlockLayout` capability is optional. The original type, enum and command names are still available as aliases of the core functionality.

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceScalarBlockLayoutFeaturesEXT

### New Enum Constants

- `VK_EXT_SCALAR_BLOCK_LAYOUT_EXTENSION_NAME`
- `VK_EXT_SCALAR_BLOCK_LAYOUT_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SCALAR_BLOCK_LAYOUT_FEATURES_EXT`

### Version History

- Revision 1, 2018-11-14 (Tobias Hector)
  - Initial draft

## VK_EXT_separate_stencil_usage

### Name String

`VK_EXT_separate_stencil_usage`

### Extension Type

Device extension

### Registered Extension Number

247

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

None

### Deprecation State

- *Promoted* to Vulkan 1.2

### Contact

- Daniel Rakos [GitHub]drakos-amd

### Other Extension Metadata

### Last Modified Date

2018-11-08

### IP Status

No known IP claims.

### Contributors

- Daniel Rakos, AMD
- Jordan Logan, AMD

### Description

This extension allows specifying separate usage flags for the stencil aspect of images with a depth-stencil format at image creation time.

### Promotion to Vulkan 1.2

All functionality in this extension is included in core Vulkan 1.2, with the EXT suffix omitted. The original type, enum and command names are still available as aliases of the core functionality.

### New Structures

- Extending VkImageCreateInfo, VkPhysicalDeviceImageFormatInfo2:
  - VkImageStencilUsageCreateInfoEXT

### New Enum Constants

- `VK_EXT_SEPARATE_STENCIL_USAGE_EXTENSION_NAME`
- `VK_EXT_SEPARATE_STENCIL_USAGE_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_IMAGE_STENCIL_USAGE_CREATE_INFO_EXT`

### Version History

- Revision 1, 2018-11-08 (Daniel Rakos)
  - Internal revisions.

## VK_EXT_shader_atomic_float

### Name String

`VK_EXT_shader_atomic_float`

### Extension Type

Device extension

### Registered Extension Number

261

### Revision

1

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### SPIR-V Dependencies

- SPV_EXT_shader_atomic_float_add

### Contact

- Vikram Kushwaha [GitHub]vkushwaha-nv

### Other Extension Metadata

### Last Modified Date

2020-07-15

### IP Status

No known IP claims.

### Interactions and External Dependencies

- This extension provides API support for `GL_EXT_shader_atomic_float`

### Contributors

- Vikram Kushwaha, NVIDIA
- Jeff Bolz, NVIDIA

### Description

This extension allows a shader to contain floating-point atomic operations on buffer, workgroup, and image memory. It also advertises the SPIR-V `AtomicFloat32AddEXT` and `AtomicFloat64AddEXT` capabilities that allows atomic addition on floating-points numbers. The supported operations include `OpAtomicFAddEXT`, `OpAtomicExchange`, `OpAtomicLoad` and `OpAtomicStore`.

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceShaderAtomicFloatFeaturesEXT

### New Enum Constants

- `VK_EXT_SHADER_ATOMIC_FLOAT_EXTENSION_NAME`
- `VK_EXT_SHADER_ATOMIC_FLOAT_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_ATOMIC_FLOAT_FEATURES_EXT`

### New SPIR-V Capabilities

- `AtomicFloat32AddEXT`
- `AtomicFloat64AddEXT`

### Version History

- Revision 1, 2020-07-15 (Vikram Kushwaha)
  - Internal revisions

## VK_EXT_shader_atomic_float2

### Name String

`VK_EXT_shader_atomic_float2`

### Extension Type

Device extension

### Registered Extension Number

274

### Revision

1

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_EXT_shader_atomic_float

### SPIR-V Dependencies

- SPV_EXT_shader_atomic_float16_add
- SPV_EXT_shader_atomic_float_min_max

### Contact

- Faith Ekstrand [GitHub]gfxstrand

### Other Extension Metadata

### Last Modified Date

2020-08-14

### IP Status

No known IP claims.

### Interactions and External Dependencies

- This extension provides API support for `GLSL_EXT_shader_atomic_float2`

### Contributors

- Faith Ekstrand, Intel

### Description

This extension allows a shader to perform 16-bit floating-point atomic operations on buffer and workgroup memory as well as floating-point atomic minimum and maximum operations on buffer, workgroup, and image memory. It advertises the SPIR-V `AtomicFloat16AddEXT` capability which allows atomic add operations on 16-bit floating-point numbers and the SPIR-V `AtomicFloat16MinMaxEXT`, `AtomicFloat32MinMaxEXT` and `AtomicFloat64MinMaxEXT` capabilities which allow atomic minimum and maximum operations on floating-point numbers. The supported operations include `OpAtomicFAddEXT`, `OpAtomicFMinEXT` and `OpAtomicFMaxEXT`.

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceShaderAtomicFloat2FeaturesEXT

### New Enum Constants

- `VK_EXT_SHADER_ATOMIC_FLOAT_2_EXTENSION_NAME`
- `VK_EXT_SHADER_ATOMIC_FLOAT_2_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_ATOMIC_FLOAT_2_FEATURES_EXT`

### Issues

1) Should this extension add support for 16-bit image atomics?

**RESOLVED**: No. While Vulkan supports creating storage images with `VK_FORMAT_R16_SFLOAT` and doing load and store on them, the data in the shader has a 32-bit representation. Vulkan currently has no facility for even basic reading or writing such images using 16-bit float values in the shader. Adding such functionality would be required before 16-bit image atomics would make sense and is outside the scope of this extension.

### New SPIR-V Capabilities

- `AtomicFloat32MinMaxEXT`
- `AtomicFloat32MinMaxEXT`
- `AtomicFloat32MinMaxEXT`
- `AtomicFloat64MinMaxEXT`

### Version History

- Revision 1, 2020-08-14 (Faith Ekstrand)
  - Internal revisions

## VK_EXT_shader_demote_to_helper_invocation

### Name String

`VK_EXT_shader_demote_to_helper_invocation`

### Extension Type

Device extension

### Registered Extension Number

277

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### SPIR-V Dependencies

- SPV_EXT_demote_to_helper_invocation

### Deprecation State

- *Promoted* to Vulkan 1.3

### Contact

- Jeff Bolz [GitHub]jeffbolznv

### Other Extension Metadata

### Last Modified Date

2019-06-01

### IP Status

No known IP claims.

### Interactions and External Dependencies

- This extension provides API support for `GL_EXT_demote_to_helper_invocation`

### Contributors

- Jeff Bolz, NVIDIA

### Description

This extension adds Vulkan support for the `SPV_EXT_demote_to_helper_invocation` SPIR-V extension. That SPIR-V extension provides a new instruction `OpDemoteToHelperInvocationEXT` allowing shaders to “demote” a fragment shader invocation to behave like a helper invocation for its duration. The demoted invocation will have no further side effects and will not output to the framebuffer, but remains active and can participate in computing derivatives and in group operations. This is a better match for the “discard” instruction in HLSL.

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceShaderDemoteToHelperInvocationFeaturesEXT

### New Enum Constants

- `VK_EXT_SHADER_DEMOTE_TO_HELPER_INVOCATION_EXTENSION_NAME`
- `VK_EXT_SHADER_DEMOTE_TO_HELPER_INVOCATION_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_DEMOTE_TO_HELPER_INVOCATION_FEATURES_EXT`

### New SPIR-V Capability

- `DemoteToHelperInvocationEXT`

### Promotion to Vulkan 1.3

Functionality in this extension is included in core Vulkan 1.3, with the EXT suffix omitted. The original type, enum and command names are still available as aliases of the core functionality.

### Version History

- Revision 1, 2019-06-01 (Jeff Bolz)
  - Initial draft

## VK_EXT_shader_image_atomic_int64

### Name String

`VK_EXT_shader_image_atomic_int64`

### Extension Type

Device extension

### Registered Extension Number

235

### Revision

1

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### SPIR-V Dependencies

- SPV_EXT_shader_image_int64

### Contact

- Tobias Hector [GitHub]tobski

### Other Extension Metadata

### Last Modified Date

2020-07-14

### IP Status

No known IP claims.

### Interactions and External Dependencies

- This extension provides API support for `GLSL_EXT_shader_image_int64`

### Contributors

- Matthaeus Chajdas, AMD
- Graham Wihlidal, Epic Games
- Tobias Hector, AMD

- Jeff Bolz, Nvidia
- Faith Ekstrand, Intel

### Description

This extension extends existing 64-bit integer atomic support to enable these operations on images as well.

When working with large 2- or 3-dimensional data sets (e.g. rasterization or screen-space effects), image accesses are generally more efficient than equivalent buffer accesses. This extension allows applications relying on 64-bit integer atomics in this manner to quickly improve performance with only relatively minor code changes.

64-bit integer atomic support is guaranteed for optimally tiled images with the `VK_FORMAT_R64_UINT` and `VK_FORMAT_R64_SINT` formats.

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceShaderImageAtomicInt64FeaturesEXT

### New Enum Constants

- `VK_EXT_SHADER_IMAGE_ATOMIC_INT64_EXTENSION_NAME`
- `VK_EXT_SHADER_IMAGE_ATOMIC_INT64_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_IMAGE_ATOMIC_INT64_FEATURES_EXT`

### Version History

- Revision 1, 2020-07-14 (Tobias Hector)
  - Initial draft

## VK_EXT_shader_module_identifier

### Name String

`VK_EXT_shader_module_identifier`

### Extension Type

Device extension

### Registered Extension Number

463

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1
and
VK_EXT_pipeline_creation_cache_control
or
Vulkan Version 1.3

### Contact

- Hans-Kristian Arntzen [GitHub]HansKristian-Work

### Extension Proposal

VK_EXT_shader_module_identifier

### Other Extension Metadata

### Last Modified Date

2022-05-16

### IP Status

No known IP claims.

### Contributors

- Hans-Kristian Arntzen, Valve
- Ricardo Garcia, Igalia
- Piers Daniell, NVIDIA
- Jan-Harald Fredriksen, Arm
- Tom Olson, Arm
- Faith Ekstrand, Collabora

### Description

Some applications generate SPIR-V code at runtime. When pipeline caches are primed, either explicitly through e.g. VkPipelineCache mechanisms, or implicitly through driver managed caches, having to re-generate SPIR-V modules is redundant. SPIR-V modules could be cached on disk by an application, but the extra disk size requirement might be prohibitive in some use cases.

This extension adds the ability for an application to query a small identifier associated with a VkShaderModule. On subsequent runs of the application, the same identifier **can** be provided in lieu of a VkShaderModule object. A pipeline creation call with such a module **may** succeed if a pipeline could be created without invoking compilation, and information inside the SPIR-V module is not required by the implementation.

VK_PIPELINE_CREATE_FAIL_ON_PIPELINE_COMPILE_REQUIRED_BIT **must** be used if only the identifier is provided, and this use case is intended to work like a non-blocking, speculative compile. Applications **can** fallback as necessary.

The main motivation for identifying the module itself and not the entire pipeline is that pipeline identifiers change when a driver is updated, but module identifiers are expected to be stable for any particular driver implementation. This approach is helpful for shader pre-compilation systems which can prime pipeline caches ahead of time. When on-disk pipeline caches are updated, the same shader identifiers could lead to a pipeline cache hit.

### New Commands

- vkGetShaderModuleCreateInfoIdentifierEXT
- vkGetShaderModuleIdentifierEXT

### New Structures

- VkShaderModuleIdentifierEXT
- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceShaderModuleIdentifierFeaturesEXT
- Extending VkPhysicalDeviceProperties2:
  - VkPhysicalDeviceShaderModuleIdentifierPropertiesEXT
- Extending VkPipelineShaderStageCreateInfo:
  - VkPipelineShaderStageModuleIdentifierCreateInfoEXT

### New Enum Constants

- VK_EXT_SHADER_MODULE_IDENTIFIER_EXTENSION_NAME
- VK_EXT_SHADER_MODULE_IDENTIFIER_SPEC_VERSION
- VK_MAX_SHADER_MODULE_IDENTIFIER_SIZE_EXT
- Extending VkStructureType:
  - VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_MODULE_IDENTIFIER_FEATURES_EXT
  - VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_MODULE_IDENTIFIER_PROPERTIES_EXT
  - VK_STRUCTURE_TYPE_PIPELINE_SHADER_STAGE_MODULE_IDENTIFIER_CREATE_INFO_EXT
  - VK_STRUCTURE_TYPE_SHADER_MODULE_IDENTIFIER_EXT

### Version History

- Revision 1, 2022-03-16 (Hans-Kristian Arntzen)
  - Initial draft

## VK_EXT_shader_object

### Name String

`VK_EXT_shader_object`

### Extension Type

Device extension

### Registered Extension Number

483

### Revision

1

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1
and
VK_KHR_dynamic_rendering
or
Vulkan Version 1.3

### API Interactions

- Interacts with VK_VERSION_1_1
- Interacts with VK_VERSION_1_3
- Interacts with VK_EXT_blend_operation_advanced
- Interacts with VK_EXT_conservative_rasterization
- Interacts with VK_EXT_depth_clip_control
- Interacts with VK_EXT_depth_clip_enable
- Interacts with VK_EXT_fragment_density_map
- Interacts with VK_EXT_line_rasterization
- Interacts with VK_EXT_mesh_shader
- Interacts with VK_EXT_provoking_vertex
- Interacts with VK_EXT_sample_locations
- Interacts with VK_EXT_subgroup_size_control
- Interacts with VK_EXT_transform_feedback
- Interacts with VK_KHR_device_group
- Interacts with VK_KHR_fragment_shading_rate

- Interacts with VK_NV_clip_space_w_scaling
- Interacts with VK_NV_coverage_reduction_mode
- Interacts with VK_NV_fragment_coverage_to_color
- Interacts with VK_NV_framebuffer_mixed_samples
- Interacts with VK_NV_mesh_shader
- Interacts with VK_NV_representative_fragment_test
- Interacts with VK_NV_shading_rate_image
- Interacts with VK_NV_viewport_swizzle

**Contact**

- Daniel Story [GitHub]daniel-story

**Extension Proposal**

VK_EXT_shader_object

### Other Extension Metadata

**Last Modified Date**

2023-03-30

**Interactions and External Dependencies**

- Interacts with `VK_EXT_extended_dynamic_state`
- Interacts with `VK_EXT_extended_dynamic_state2`
- Interacts with `VK_EXT_extended_dynamic_state3`
- Interacts with `VK_EXT_vertex_input_dynamic_state`

**IP Status**

No known IP claims.

**Contributors**

- Piers Daniell, NVIDIA
- Sandy Jamieson, Nintendo
- Žiga Markuš, LunarG
- Tobias Hector, AMD
- Alex Walters, Imagination
- Shahbaz Youssefi, Google
- Ralph Potter, Samsung
- Jan-Harald Fredriksen, ARM
- Connor Abott, Valve
- Arseny Kapoulkine, Roblox
- Patrick Doane, Activision

- Jeff Leger, Qualcomm
- Stu Smith, AMD
- Chris Glover, Google
- Ricardo Garcia, Igalia
- Faith Ekstrand, Collabora
- Timur Kristóf, Valve
- Constantine Shablya, Collabora
- Daniel Koch, NVIDIA
- Alyssa Rosenzweig, Collabora
- Mike Blumenkrantz, Valve
- Samuel Pitoiset, Valve
- Qun Lin, AMD
- Spencer Fricke, LunarG
- Soroush Faghihi Kashani, Imagination

### Description

This extension introduces a new VkShaderEXT object type which represents a single compiled shader stage. Shader objects provide a more flexible alternative to VkPipeline objects, which may be helpful in certain use cases.

### New Object Types

- VkShaderEXT

### New Commands

- vkCmdBindShadersEXT
- vkCmdBindVertexBuffers2EXT
- vkCmdSetAlphaToCoverageEnableEXT
- vkCmdSetAlphaToOneEnableEXT
- vkCmdSetColorBlendEnableEXT
- vkCmdSetColorBlendEquationEXT
- vkCmdSetColorWriteMaskEXT
- vkCmdSetCullModeEXT
- vkCmdSetDepthBiasEnableEXT
- vkCmdSetDepthBoundsTestEnableEXT
- vkCmdSetDepthClampEnableEXT
- vkCmdSetDepthCompareOpEXT

- vkCmdSetDepthTestEnableEXT
- vkCmdSetDepthWriteEnableEXT
- vkCmdSetFrontFaceEXT
- vkCmdSetLogicOpEXT
- vkCmdSetLogicOpEnableEXT
- vkCmdSetPatchControlPointsEXT
- vkCmdSetPolygonModeEXT
- vkCmdSetPrimitiveRestartEnableEXT
- vkCmdSetPrimitiveTopologyEXT
- vkCmdSetRasterizationSamplesEXT
- vkCmdSetRasterizerDiscardEnableEXT
- vkCmdSetSampleMaskEXT
- vkCmdSetScissorWithCountEXT
- vkCmdSetStencilOpEXT
- vkCmdSetStencilTestEnableEXT
- vkCmdSetTessellationDomainOriginEXT
- vkCmdSetVertexInputEXT
- vkCmdSetViewportWithCountEXT
- vkCreateShadersEXT
- vkDestroyShaderEXT
- vkGetShaderBinaryDataEXT

If VK_EXT_blend_operation_advanced is supported:

- vkCmdSetColorBlendAdvancedEXT

If VK_EXT_conservative_rasterization is supported:

- vkCmdSetConservativeRasterizationModeEXT
- vkCmdSetExtraPrimitiveOverestimationSizeEXT

If VK_EXT_depth_clip_control is supported:

- vkCmdSetDepthClipNegativeOneToOneEXT

If VK_EXT_depth_clip_enable is supported:

- vkCmdSetDepthClipEnableEXT

If VK_EXT_line_rasterization is supported:

- vkCmdSetLineRasterizationModeEXT

- vkCmdSetLineStippleEnableEXT

If VK_EXT_provoking_vertex is supported:

- vkCmdSetProvokingVertexModeEXT

If VK_EXT_sample_locations is supported:

- vkCmdSetSampleLocationsEnableEXT

If VK_EXT_transform_feedback is supported:

- vkCmdSetRasterizationStreamEXT

If VK_NV_clip_space_w_scaling is supported:

- vkCmdSetViewportWScalingEnableNV

If VK_NV_coverage_reduction_mode is supported:

- vkCmdSetCoverageReductionModeNV

If VK_NV_fragment_coverage_to_color is supported:

- vkCmdSetCoverageToColorEnableNV
- vkCmdSetCoverageToColorLocationNV

If VK_NV_framebuffer_mixed_samples is supported:

- vkCmdSetCoverageModulationModeNV
- vkCmdSetCoverageModulationTableEnableNV
- vkCmdSetCoverageModulationTableNV

If VK_NV_representative_fragment_test is supported:

- vkCmdSetRepresentativeFragmentTestEnableNV

If VK_NV_shading_rate_image is supported:

- vkCmdSetShadingRateImageEnableNV

If VK_NV_viewport_swizzle is supported:

- vkCmdSetViewportSwizzleNV

**New Structures**

- VkColorBlendAdvancedEXT
- VkColorBlendEquationEXT
- VkShaderCreateInfoEXT

- VkVertexInputAttributeDescription2EXT
- VkVertexInputBindingDescription2EXT
- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceShaderObjectFeaturesEXT
- Extending VkPhysicalDeviceProperties2:
  - VkPhysicalDeviceShaderObjectPropertiesEXT
- Extending VkPipelineShaderStageCreateInfo, VkShaderCreateInfoEXT:
  - VkShaderRequiredSubgroupSizeCreateInfoEXT

#### New Enums

- VkShaderCodeTypeEXT
- VkShaderCreateFlagBitsEXT

#### New Bitmasks

- VkShaderCreateFlagsEXT

#### New Enum Constants

- `VK_EXT_SHADER_OBJECT_EXTENSION_NAME`
- `VK_EXT_SHADER_OBJECT_SPEC_VERSION`
- Extending VkObjectType:
  - `VK_OBJECT_TYPE_SHADER_EXT`
- Extending VkResult:
  - `VK_ERROR_INCOMPATIBLE_SHADER_BINARY_EXT`
  - `VK_INCOMPATIBLE_SHADER_BINARY_EXT`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_OBJECT_FEATURES_EXT`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_OBJECT_PROPERTIES_EXT`
  - `VK_STRUCTURE_TYPE_SHADER_CREATE_INFO_EXT`
  - `VK_STRUCTURE_TYPE_SHADER_REQUIRED_SUBGROUP_SIZE_CREATE_INFO_EXT`
  - `VK_STRUCTURE_TYPE_VERTEX_INPUT_ATTRIBUTE_DESCRIPTION_2_EXT`
  - `VK_STRUCTURE_TYPE_VERTEX_INPUT_BINDING_DESCRIPTION_2_EXT`

If VK_EXT_fragment_density_map is supported:

- Extending VkShaderCreateFlagBitsEXT:
  - `VK_SHADER_CREATE_FRAGMENT_DENSITY_MAP_ATTACHMENT_BIT_EXT`

If VK_EXT_mesh_shader or VK_NV_mesh_shader is supported:

- Extending VkShaderCreateFlagBitsEXT:
  - VK_SHADER_CREATE_NO_TASK_SHADER_BIT_EXT

If VK_EXT_subgroup_size_control or Vulkan Version 1.3 is supported:

- Extending VkShaderCreateFlagBitsEXT:
  - VK_SHADER_CREATE_ALLOW_VARYING_SUBGROUP_SIZE_BIT_EXT
  - VK_SHADER_CREATE_REQUIRE_FULL_SUBGROUPS_BIT_EXT

If VK_KHR_device_group or Vulkan Version 1.1 is supported:

- Extending VkShaderCreateFlagBitsEXT:
  - VK_SHADER_CREATE_DISPATCH_BASE_BIT_EXT

If VK_KHR_fragment_shading_rate is supported:

- Extending VkShaderCreateFlagBitsEXT:
  - VK_SHADER_CREATE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_EXT

**Examples**

**Example 1**

Create linked pair of vertex and fragment shaders.

```
// Logical device created with the shaderObject feature enabled
VkDevice device;

// SPIR-V shader code for a vertex shader, along with its size in bytes
void* pVertexSpirv;
size_t vertexSpirvSize;

// SPIR-V shader code for a fragment shader, along with its size in bytes
void* pFragmentSpirv;
size_t fragmentSpirvSize;

// Descriptor set layout compatible with the shaders
VkDescriptorSetLayout descriptorSetLayout;

VkShaderCreateInfoEXT shaderCreateInfos[2] =
{
    {
        .sType = VK_STRUCTURE_TYPE_SHADER_CREATE_INFO_EXT,
        .pNext = NULL,
        .flags = VK_SHADER_CREATE_LINK_STAGE_BIT_EXT,
        .stage = VK_SHADER_STAGE_VERTEX_BIT,
        .nextStage = VK_SHADER_STAGE_FRAGMENT_BIT,
        .codeType = VK_SHADER_CODE_TYPE_SPIRV_EXT,
        .codeSize = vertexSpirvSize,
```

```
        .pCode = pVertexSpirv,
        .pName = "main",
        .setLayoutCount = 1,
        .pSetLayouts = &descriptorSetLayout;
        .pushConstantRangeCount = 0,
        .pPushConstantRanges = NULL,
        .pSpecializationInfo = NULL
    },
    {
        .sType = VK_STRUCTURE_TYPE_SHADER_CREATE_INFO_EXT,
        .pNext = NULL,
        .flags = VK_SHADER_CREATE_LINK_STAGE_BIT_EXT,
        .stage = VK_SHADER_STAGE_FRAGMENT_BIT,
        .nextStage = 0,
        .codeType = VK_SHADER_CODE_TYPE_SPIRV_EXT,
        .codeSize = fragmentSpirvSize,
        .pCode = pFragmentSpirv,
        .pName = "main",
        .setLayoutCount = 1,
        .pSetLayouts = &descriptorSetLayout;
        .pushConstantRangeCount = 0,
        .pPushConstantRanges = NULL,
        .pSpecializationInfo = NULL
    }
};

VkResult result;
VkShaderEXT shaders[2];

result = vkCreateShadersEXT(device, 2, &shaderCreateInfos, NULL, shaders);
if (result != VK_SUCCESS)
{
    // Handle error
}
```

Later, during command buffer recording, bind the linked shaders and draw.

```
// Command buffer in the recording state
VkCommandBuffer commandBuffer;

// Vertex and fragment shader objects created above
VkShaderEXT shaders[2];

// Assume vertex buffers, descriptor sets, etc. have been bound, and existing
// state setting commands have been called to set all required state

const VkShaderStageFlagBits stages[2] =
{
    VK_SHADER_STAGE_VERTEX_BIT,
    VK_SHADER_STAGE_FRAGMENT_BIT
```

```
};

// Bind linked shaders
vkCmdBindShadersEXT(commandBuffer, 2, stages, shaders);

// Equivalent to the previous line. Linked shaders can be bound one at a time,
// in any order:
// vkCmdBindShadersEXT(commandBuffer, 1, &stages[1], &shaders[1]);
// vkCmdBindShadersEXT(commandBuffer, 1, &stages[0], &shaders[0]);

// The above is sufficient to draw if the device was created with the
// tessellationShader and geometryShader features disabled. Otherwise, since
// those stages should not execute, vkCmdBindShadersEXT() must be called at
// least once with each of their stages in pStages before drawing:

const VkShaderStageFlagBits unusedStages[3] =
{
    VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT,
    VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT,
    VK_SHADER_STAGE_GEOMETRY_BIT
};

// NULL pShaders is equivalent to an array of stageCount VK_NULL_HANDLE values,
// meaning no shaders are bound to those stages, and that any previously bound
// shaders are unbound
vkCmdBindShadersEXT(commandBuffer, 3, unusedStages, NULL);

// Graphics shader objects may only be used to draw inside dynamic render pass
// instances begun with vkCmdBeginRendering(), assume one has already been begun

// Draw a triangle
vkCmdDraw(commandBuffer, 3, 1, 0, 0);
```

**Example 2**

Create unlinked vertex, geometry, and fragment shaders.

```
// Logical device created with the shaderObject feature enabled
VkDevice device;

// SPIR-V shader code for vertex shaders, along with their sizes in bytes
void* pVertexSpirv[2];
size_t vertexSpirvSize[2];

// SPIR-V shader code for a geometry shader, along with its size in bytes
void pGeometrySpirv;
size_t geometrySpirvSize;

// SPIR-V shader code for fragment shaders, along with their sizes in bytes
void* pFragmentSpirv[2];
```

```
size_t fragmentSpirvSize[2];

// Descriptor set layout compatible with the shaders
VkDescriptorSetLayout descriptorSetLayout;

VkShaderCreateInfoEXT shaderCreateInfos[5] =
{
    // Stage order does not matter
    {
        .sType = VK_STRUCTURE_TYPE_SHADER_CREATE_INFO_EXT,
        .pNext = NULL,
        .flags = 0,
        .stage = VK_SHADER_STAGE_GEOMETRY_BIT,
        .nextStage = VK_SHADER_STAGE_FRAGMENT_BIT,
        .codeType = VK_SHADER_CODE_TYPE_SPIRV_EXT,
        .codeSize = pGeometrySpirv,
        .pCode = geometrySpirvSize,
        .pName = "main",
        .setLayoutCount = 1,
        .pSetLayouts = &descriptorSetLayout;
        .pushConstantRangeCount = 0,
        .pPushConstantRanges = NULL,
        .pSpecializationInfo = NULL
    },
    {
        .sType = VK_STRUCTURE_TYPE_SHADER_CREATE_INFO_EXT,
        .pNext = NULL,
        .flags = 0,
        .stage = VK_SHADER_STAGE_VERTEX_BIT,
        .nextStage = VK_SHADER_STAGE_GEOMETRY_BIT,
        .codeType = VK_SHADER_CODE_TYPE_SPIRV_EXT,
        .codeSize = vertexSpirvSize[0],
        .pCode = pVertexSpirv[0],
        .pName = "main",
        .setLayoutCount = 1,
        .pSetLayouts = &descriptorSetLayout;
        .pushConstantRangeCount = 0,
        .pPushConstantRanges = NULL,
        .pSpecializationInfo = NULL
    },
    {
        .sType = VK_STRUCTURE_TYPE_SHADER_CREATE_INFO_EXT,
        .pNext = NULL,
        .flags = 0,
        .stage = VK_SHADER_STAGE_FRAGMENT_BIT,
        .nextStage = 0,
        .codeType = VK_SHADER_CODE_TYPE_SPIRV_EXT,
        .codeSize = fragmentSpirvSize[0],
        .pCode = pFragmentSpirv[0],
        .pName = "main",
        .setLayoutCount = 1,
```

```
        .pSetLayouts = &descriptorSetLayout;
        .pushConstantRangeCount = 0,
        .pPushConstantRanges = NULL,
        .pSpecializationInfo = NULL
    },
    {
        .sType = VK_STRUCTURE_TYPE_SHADER_CREATE_INFO_EXT,
        .pNext = NULL,
        .flags = 0,
        .stage = VK_SHADER_STAGE_FRAGMENT_BIT,
        .nextStage = 0,
        .codeType = VK_SHADER_CODE_TYPE_SPIRV_EXT,
        .codeSize = fragmentSpirvSize[1],
        .pCode = pFragmentSpirv[1],
        .pName = "main",
        .setLayoutCount = 1,
        .pSetLayouts = &descriptorSetLayout;
        .pushConstantRangeCount = 0,
        .pPushConstantRanges = NULL,
        .pSpecializationInfo = NULL
    },
    {
        .sType = VK_STRUCTURE_TYPE_SHADER_CREATE_INFO_EXT,
        .pNext = NULL,
        .flags = 0,
        .stage = VK_SHADER_STAGE_VERTEX_BIT,
        // Suppose we want this vertex shader to be able to be followed by
        // either a geometry shader or fragment shader:
        .nextStage = VK_SHADER_STAGE_GEOMETRY_BIT | VK_SHADER_STAGE_FRAGMENT_BIT,
        .codeType = VK_SHADER_CODE_TYPE_SPIRV_EXT,
        .codeSize = vertexSpirvSize[1],
        .pCode = pVertexSpirv[1],
        .pName = "main",
        .setLayoutCount = 1,
        .pSetLayouts = &descriptorSetLayout;
        .pushConstantRangeCount = 0,
        .pPushConstantRanges = NULL,
        .pSpecializationInfo = NULL
    }
};

VkResult result;
VkShaderEXT shaders[5];

result = vkCreateShadersEXT(device, 5, &shaderCreateInfos, NULL, shaders);
if (result != VK_SUCCESS)
{
    // Handle error
}
```

Later, during command buffer recording, bind the linked shaders in different combinations and draw.

```
// Command buffer in the recording state
VkCommandBuffer commandBuffer;

// Vertex, geometry, and fragment shader objects created above
VkShaderEXT shaders[5];

// Assume vertex buffers, descriptor sets, etc. have been bound, and existing
// state setting commands have been called to set all required state

const VkShaderStageFlagBits stages[3] =
{
    // Any order is allowed
    VK_SHADER_STAGE_FRAGMENT_BIT,
    VK_SHADER_STAGE_VERTEX_BIT,
    VK_SHADER_STAGE_GEOMETRY_BIT,
};

VkShaderEXT bindShaders[3] =
{
    shaders[2], // FS
    shaders[1], // VS
    shaders[0]  // GS
};

// Bind unlinked shaders
vkCmdBindShadersEXT(commandBuffer, 3, stages, bindShaders);

// Assume the tessellationShader feature is disabled, so vkCmdBindShadersEXT()
// need not have been called with either tessellation stage

// Graphics shader objects may only be used to draw inside dynamic render pass
// instances begun with vkCmdBeginRendering(), assume one has already been begun

// Draw a triangle
vkCmdDraw(commandBuffer, 3, 1, 0, 0);

// Bind a different unlinked fragment shader
const VkShaderStageFlagBits fragmentStage = VK_SHADER_STAGE_FRAGMENT_BIT;
vkCmdBindShadersEXT(commandBuffer, 1, &fragmentStage, &shaders[3]);

// Draw another triangle
vkCmdDraw(commandBuffer, 3, 1, 0, 0);

// Bind a different unlinked vertex shader
const VkShaderStageFlagBits vertexStage = VK_SHADER_STAGE_VERTEX_BIT;
vkCmdBindShadersEXT(commandBuffer, 1, &vertexStage, &shaders[4]);
```

```
// Draw another triangle
vkCmdDraw(commandBuffer, 3, 1, 0, 0);
```

### Version History

- Revision 1, 2023-03-30 (Daniel Story)
  - Initial draft

## VK_EXT_shader_replicated_composites

### Name String

`VK_EXT_shader_replicated_composites`

### Extension Type

Device extension

### Registered Extension Number

565

### Revision

1

### Ratification Status

Ratified

### Extension and Version Dependencies

None

### SPIR-V Dependencies

- SPV_EXT_replicated_composites

### Contact

- Kevin Petit [GitHub]kpet

### Extension Proposal

VK_EXT_shader_replicated_composites

### Last Modified Date

2024-02-08

### IP Status

No known IP claims.

### Contributors

- Kévin Petit, Arm Ltd.
- Jeff Bolz, NVIDIA
- Piers Daniell, NVIDIA

This extension adds support for creating composites from a single value in SPIR-V modules, as defined by SPV_EXT_replicated_composites.

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceShaderReplicatedCompositesFeaturesEXT

### New Enum Constants

- `VK_EXT_SHADER_REPLICATED_COMPOSITES_EXTENSION_NAME`
- `VK_EXT_SHADER_REPLICATED_COMPOSITES_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_REPLICATED_COMPOSITES_FEATURES_EXT`

### New SPIR-V Capabilities

- ReplicatedCompositesEXT

### Version History

- Revision 1, 2024-02-08 (Kévin Petit)
  - Initial revision

## VK_EXT_shader_stencil_export

### Name String

`VK_EXT_shader_stencil_export`

### Extension Type

Device extension

### Registered Extension Number

141

### Revision

1

### Ratification Status

Ratified

### Extension and Version Dependencies

None

### SPIR-V Dependencies

- SPV_EXT_shader_stencil_export

### Contact

- Dominik Witczak [GitHub]dominikwitczakamd

### Other Extension Metadata

### Last Modified Date

2017-07-19

### IP Status

No known IP claims.

### Interactions and External Dependencies

- This extension provides API support for `GL_ARB_shader_stencil_export`

### Contributors

- Dominik Witczak, AMD
- Daniel Rakos, AMD
- Rex Xu, AMD

### Description

This extension adds support for the SPIR-V extension `SPV_EXT_shader_stencil_export`, providing a mechanism whereby a shader may generate the stencil reference value per invocation. When stencil testing is enabled, this allows the test to be performed against the value generated in the shader.

### New Enum Constants

- `VK_EXT_SHADER_STENCIL_EXPORT_EXTENSION_NAME`
- `VK_EXT_SHADER_STENCIL_EXPORT_SPEC_VERSION`

### Version History

- Revision 1, 2017-07-19 (Dominik Witczak)
  - Initial draft

## VK_EXT_shader_subgroup_ballot

### Name String

`VK_EXT_shader_subgroup_ballot`

### Extension Type

Device extension

### Registered Extension Number

65

**Revision**

1

**Ratification Status**

Not ratified

**Extension and Version Dependencies**

None

**SPIR-V Dependencies**

- SPV_KHR_shader_ballot

**Deprecation State**

- *Deprecated* by Vulkan 1.2

**Contact**

- Daniel Koch [GitHub]dgkoch

**Other Extension Metadata**

**Last Modified Date**

2016-11-28

**IP Status**

No known IP claims.

**Interactions and External Dependencies**

- This extension provides API support for `GL_ARB_shader_ballot`

**Contributors**

- Jeff Bolz, NVIDIA
- Neil Henning, Codeplay
- Daniel Koch, NVIDIA Corporation

**Description**

This extension adds support for the following SPIR-V extension in Vulkan:

- `SPV_KHR_shader_ballot`

This extension provides the ability for a group of invocations, which execute in parallel, to do limited forms of cross-invocation communication via a group broadcast of an invocation value, or broadcast of a bit array representing a predicate value from each invocation in the group.

This extension provides access to a number of additional built-in shader variables in Vulkan:

- `SubgroupEqMaskKHR`, containing the subgroup mask of the current subgroup invocation,
- `SubgroupGeMaskKHR`, containing the subgroup mask of the invocations greater than or equal to the current invocation,

- SubgroupGtMaskKHR, containing the subgroup mask of the invocations greater than the current invocation,
- SubgroupLeMaskKHR, containing the subgroup mask of the invocations less than or equal to the current invocation,
- SubgroupLtMaskKHR, containing the subgroup mask of the invocations less than the current invocation,
- SubgroupLocalInvocationId, containing the index of an invocation within a subgroup, and
- SubgroupSize, containing the maximum number of invocations in a subgroup.

Additionally, this extension provides access to the new SPIR-V instructions:

- OpSubgroupBallotKHR,
- OpSubgroupFirstInvocationKHR, and
- OpSubgroupReadInvocationKHR,

When using GLSL source-based shader languages, the following variables and shader functions from GL_ARB_shader_ballot can map to these SPIR-V built-in decorations and instructions:

- `in uint64_t gl_SubGroupEqMaskARB;` → SubgroupEqMaskKHR,
- `in uint64_t gl_SubGroupGeMaskARB;` → SubgroupGeMaskKHR,
- `in uint64_t gl_SubGroupGtMaskARB;` → SubgroupGtMaskKHR,
- `in uint64_t gl_SubGroupLeMaskARB;` → SubgroupLeMaskKHR,
- `in uint64_t gl_SubGroupLtMaskARB;` → SubgroupLtMaskKHR,
- `in uint gl_SubGroupInvocationARB;` → SubgroupLocalInvocationId,
- `uniform uint gl_SubGroupSizeARB;` → SubgroupSize,
- ballotARB() → OpSubgroupBallotKHR,
- readFirstInvocationARB() → OpSubgroupFirstInvocationKHR, and
- readInvocationARB() → OpSubgroupReadInvocationKHR.

#### Deprecated by Vulkan 1.2

Most of the functionality in this extension is superseded by the core Vulkan 1.1 subgroup operations. However, Vulkan 1.1 required the OpGroupNonUniformBroadcast "Id" to be constant. This restriction was removed in Vulkan 1.2 with the addition of the subgroupBroadcastDynamicId feature.

#### New Enum Constants

- VK_EXT_SHADER_SUBGROUP_BALLOT_EXTENSION_NAME
- VK_EXT_SHADER_SUBGROUP_BALLOT_SPEC_VERSION

#### New Built-In Variables

- SubgroupEqMaskKHR

- `SubgroupGeMaskKHR`
- `SubgroupGtMaskKHR`
- `SubgroupLeMaskKHR`
- `SubgroupLtMaskKHR`
- `SubgroupLocalInvocationId`
- `SubgroupSize`

#### New SPIR-V Capabilities

- `SubgroupBallotKHR`

#### Version History

- Revision 1, 2016-11-28 (Daniel Koch)
  - Initial draft

## VK_EXT_shader_tile_image

#### Name String

`VK_EXT_shader_tile_image`

#### Extension Type

Device extension

#### Registered Extension Number

396

#### Revision

1

#### Ratification Status

Ratified

#### Extension and Version Dependencies

Vulkan Version 1.3

#### SPIR-V Dependencies

- SPV_EXT_shader_tile_image

#### Contact

- Jan-Harald Fredriksen [GitHub]janharaldfredriksen-arm

#### Extension Proposal

VK_EXT_shader_tile_image

### Other Extension Metadata

#### Last Modified Date

2023-03-23

#### IP Status

No known IP claims.

#### Interactions and External Dependencies

- This extension provides API support for `GL_EXT_shader_tile_image`

#### Contributors

- Sandeep Kakarlapudi, Arm
- Jan-Harald Fredriksen, Arm
- James Fitzpatrick, Imagination
- Andrew Garrard, Imagination
- Jeff Leger, Qualcomm
- Huilong Wang, Huawei
- Graeme Leese, Broadcom
- Hans-Kristian Arntzen, Valve
- Tobias Hector, AMD
- Jeff Bolz, NVIDIA
- Shahbaz Youssefi, Google

#### Description

This extension allows fragment shader invocations to read color, depth and stencil values at their pixel location in rasterization order. The functionality is only available when using dynamic render passes introduced by VK_KHR_dynamic_rendering. Example use cases are programmable blending and deferred shading.

See fragment shader tile image reads for more information.

#### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceShaderTileImageFeaturesEXT
- Extending VkPhysicalDeviceProperties2:
  - VkPhysicalDeviceShaderTileImagePropertiesEXT

#### New Enum Constants

- `VK_EXT_SHADER_TILE_IMAGE_EXTENSION_NAME`
- `VK_EXT_SHADER_TILE_IMAGE_SPEC_VERSION`

- Extending VkStructureType:
  - VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_TILE_IMAGE_FEATURES_EXT
  - VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_TILE_IMAGE_PROPERTIES_EXT

### Issues

None.

### Examples

Color read example.

```
layout( location = 0 /* aliased to color attachment 0 */ ) tileImageEXT highp
attachmentEXT color0;
layout( location = 1 /* aliased to color attachment 1 */ ) tileImageEXT highp
attachmentEXT color1;

layout( location = 0 ) out vec4 fragColor;

void main()
{
    vec4 value = colorAttachmentReadEXT(color0) + colorAttachmentReadEXT(color1);
    fragColor = value;
}
```

Depth & Stencil read example.

```
void main()
{
    // read sample 0: works for non-MSAA or MSAA targets
    highp float last_depth = depthAttachmentReadEXT();
    lowp uint last_stencil = stencilAttachmentReadEXT();

    //..
}
```

### Version History

- Revision 1, 2023-03-23 (Sandeep Kakarlapudi)
  - Initial version

## VK_EXT_shader_viewport_index_layer

### Name String

VK_EXT_shader_viewport_index_layer

### Extension Type

Device extension

### Registered Extension Number

163

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

None

### SPIR-V Dependencies

- SPV_EXT_shader_viewport_index_layer

### Deprecation State

- *Promoted* to Vulkan 1.2

### Contact

- Daniel Koch [GitHub]dgkoch

### Other Extension Metadata

### Last Modified Date

2017-08-08

### Interactions and External Dependencies

- This extension provides API support for `GL_ARB_shader_viewport_layer_array`, `GL_AMD_vertex_shader_layer`, `GL_AMD_vertex_shader_viewport_index`, and `GL_NV_viewport_array2`
- This extension requires the `multiViewport` feature.
- This extension interacts with the `tessellationShader` feature.

### Contributors

- Piers Daniell, NVIDIA
- Jeff Bolz, NVIDIA
- Jan-Harald Fredriksen, ARM
- Daniel Rakos, AMD
- Slawomir Grajeswki, Intel

### Description

This extension adds support for the `ShaderViewportIndexLayerEXT` capability from the `SPV_EXT_shader_viewport_index_layer` extension in Vulkan.

This extension allows variables decorated with the `Layer` and `ViewportIndex` built-ins to be exported from vertex or tessellation shaders, using the `ShaderViewportIndexLayerEXT` capability.

When using GLSL source-based shading languages, the `gl_ViewportIndex` and `gl_Layer` built-in variables map to the SPIR-V `ViewportIndex` and `Layer` built-in decorations, respectively. Behavior of these variables is extended as described in the `GL_ARB_shader_viewport_layer_array` (or the precursor `GL_AMD_vertex_shader_layer`, `GL_AMD_vertex_shader_viewport_index`, and `GL_NV_viewport_array2` extensions).

| | |
|---|---|
| **NOTE** | The `ShaderViewportIndexLayerEXT` capability is equivalent to the `ShaderViewportIndexLayerNV` capability added by `VK_NV_viewport_array2`. |

#### Promotion to Vulkan 1.2

All functionality in this extension is included in core Vulkan 1.2.

The single `ShaderViewportIndexLayerEXT` capability from the `SPV_EXT_shader_viewport_index_layer` extension is replaced by the `ShaderViewportIndex` and `ShaderLayer` capabilities from SPIR-V 1.5 which are enabled by the `shaderOutputViewportIndex` and `shaderOutputLayer` features, respectively. Additionally, if Vulkan 1.2 is supported but this extension is not, these capabilities are optional.

Enabling both features is equivalent to enabling the `VK_EXT_shader_viewport_index_layer` extension.

#### New Enum Constants

- `VK_EXT_SHADER_VIEWPORT_INDEX_LAYER_EXTENSION_NAME`
- `VK_EXT_SHADER_VIEWPORT_INDEX_LAYER_SPEC_VERSION`

#### New or Modified Built-In Variables

- (modified) `Layer`
- (modified) `ViewportIndex`

#### New SPIR-V Capabilities

- `ShaderViewportIndexLayerEXT`

#### Version History

- Revision 1, 2017-08-08 (Daniel Koch)
  - Internal drafts

## VK_EXT_subgroup_size_control

#### Name String

`VK_EXT_subgroup_size_control`

#### Extension Type

Device extension

**Registered Extension Number**

226

**Revision**

2

**Ratification Status**

Not ratified

**Extension and Version Dependencies**

Vulkan Version 1.1

**Deprecation State**

- *Promoted* to Vulkan 1.3

**Contact**

- Neil Henning [GitHub]sheredom

**Other Extension Metadata**

**Last Modified Date**

2019-03-05

**Contributors**

- Jeff Bolz, NVIDIA
- Faith Ekstrand, Intel
- Sławek Grajewski, Intel
- Jesse Hall, Google
- Neil Henning, AMD
- Daniel Koch, NVIDIA
- Jeff Leger, Qualcomm
- Graeme Leese, Broadcom
- Allan MacKinnon, Google
- Mariusz Merecki, Intel
- Graham Wihlidal, Electronic Arts

**Description**

This extension enables an implementation to control the subgroup size by allowing a varying subgroup size and also specifying a required subgroup size.

It extends the subgroup support in Vulkan 1.1 to allow an implementation to expose a varying subgroup size. Previously Vulkan exposed a single subgroup size per physical device, with the expectation that implementations will behave as if all subgroups have the same size. Some implementations **may** dispatch shaders with a varying subgroup size for different subgroups. As a

result they could implicitly split a large subgroup into smaller subgroups or represent a small subgroup as a larger subgroup, some of whose invocations were inactive on launch.

To aid developers in understanding the performance characteristics of their programs, this extension exposes a minimum and maximum subgroup size that a physical device supports and a pipeline create flag to enable that pipeline to vary its subgroup size. If enabled, any `SubgroupSize` decorated variables in the SPIR-V shader modules provided to pipeline creation **may** vary between the minimum and maximum subgroup sizes.

An implementation is also optionally allowed to support specifying a required subgroup size for a given pipeline stage. Implementations advertise which stages support a required subgroup size, and any pipeline of a supported stage can be passed a VkPipelineShaderStageRequiredSubgroupSizeCreateInfoEXT structure to set the subgroup size for that shader stage of the pipeline. For compute shaders, this requires the developer to query the `maxComputeWorkgroupSubgroups` and ensure that:

$$s = WorkGroupSize.x \times WorkGroupSize.y \times WorkgroupSize.z \leq SubgroupSize \times maxComputeWorkgroupSubgroups$$

Developers can also specify a new pipeline shader stage create flag that requires the implementation to have fully populated subgroups within local workgroups. This requires the workgroup size in the X dimension to be a multiple of the subgroup size.

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceSubgroupSizeControlFeaturesEXT
- Extending VkPhysicalDeviceProperties2:
  - VkPhysicalDeviceSubgroupSizeControlPropertiesEXT
- Extending VkPipelineShaderStageCreateInfo, VkShaderCreateInfoEXT:
  - VkPipelineShaderStageRequiredSubgroupSizeCreateInfoEXT

### New Enum Constants

- `VK_EXT_SUBGROUP_SIZE_CONTROL_EXTENSION_NAME`
- `VK_EXT_SUBGROUP_SIZE_CONTROL_SPEC_VERSION`
- Extending VkPipelineShaderStageCreateFlagBits:
  - `VK_PIPELINE_SHADER_STAGE_CREATE_ALLOW_VARYING_SUBGROUP_SIZE_BIT_EXT`
  - `VK_PIPELINE_SHADER_STAGE_CREATE_REQUIRE_FULL_SUBGROUPS_BIT_EXT`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SUBGROUP_SIZE_CONTROL_FEATURES_EXT`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SUBGROUP_SIZE_CONTROL_PROPERTIES_EXT`
  - `VK_STRUCTURE_TYPE_PIPELINE_SHADER_STAGE_REQUIRED_SUBGROUP_SIZE_CREATE_INFO_EXT`

### Promotion to Vulkan 1.3

Functionality in this extension is included in core Vulkan 1.3, with the EXT suffix omitted. The original type, enum and command names are still available as aliases of the core functionality.

### Version History

- Revision 1, 2019-03-05 (Neil Henning)
  - Initial draft
- Revision 2, 2019-07-26 (Faith Ekstrand)
  - Add the missing VkPhysicalDeviceSubgroupSizeControlFeaturesEXT for querying subgroup size control features.

## VK_EXT_subpass_merge_feedback

### Name String

`VK_EXT_subpass_merge_feedback`

### Extension Type

Device extension

### Registered Extension Number

459

### Revision

2

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### Contact

- Ting Wei [GitHub]catweiting

### Extension Proposal

VK_EXT_subpass_merge_feedback

### Other Extension Metadata

### Last Modified Date

2022-05-24

### IP Status

No known IP claims.

### Contributors

- Jan-Harald Fredriksen, Arm
- Jorg Wagner, Arm
- Ting Wei, Arm

### Description

This extension adds a mechanism to provide feedback to an application about whether the subpasses specified on render pass creation are merged by the implementation. Additionally, it provides a control to enable or disable subpass merging in the render pass.

### New Structures

- VkRenderPassCreationFeedbackInfoEXT
- VkRenderPassSubpassFeedbackInfoEXT
- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceSubpassMergeFeedbackFeaturesEXT
- Extending VkRenderPassCreateInfo2:
  - VkRenderPassCreationFeedbackCreateInfoEXT
- Extending VkRenderPassCreateInfo2, VkSubpassDescription2:
  - VkRenderPassCreationControlEXT
- Extending VkSubpassDescription2:
  - VkRenderPassSubpassFeedbackCreateInfoEXT

### New Enums

- VkSubpassMergeStatusEXT

### New Enum Constants

- `VK_EXT_SUBPASS_MERGE_FEEDBACK_EXTENSION_NAME`
- `VK_EXT_SUBPASS_MERGE_FEEDBACK_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SUBPASS_MERGE_FEEDBACK_FEATURES_EXT`
  - `VK_STRUCTURE_TYPE_RENDER_PASS_CREATION_CONTROL_EXT`
  - `VK_STRUCTURE_TYPE_RENDER_PASS_CREATION_FEEDBACK_CREATE_INFO_EXT`
  - `VK_STRUCTURE_TYPE_RENDER_PASS_SUBPASS_FEEDBACK_CREATE_INFO_EXT`

### Version History

- Revision 1, 2022-03-10
  - Initial draft.

- Revision 2, 2022-05-24
  - Fix structextends and constness issues.

## VK_EXT_surface_maintenance1

### Name String

`VK_EXT_surface_maintenance1`

### Extension Type

Instance extension

### Registered Extension Number

275

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_surface
and
VK_KHR_get_surface_capabilities2

### Contact

- Shahbaz Youssefi [GitHub]syoussefi

### Extension Proposal

VK_EXT_surface_maintenance1

### Other Extension Metadata

### Last Modified Date

2022-11-09

### Contributors

- Jeff Juliano, NVIDIA
- Lionel Landwerlin, Intel
- Shahbaz Youssefi, Google
- Chris Forbes, Google
- Ian Elliott, Google
- Hans-Kristian Arntzen, Valve
- Daniel Stone, Collabora

### Description

`VK_EXT_surface_maintenance1` adds a collection of window system integration features that were intentionally left out or overlooked in the original `VK_KHR_surface` extension.

The new features are as follows:

- Allow querying number of min/max images from a surface for a particular presentation mode.
- Allow querying a surface's scaled presentation capabilities.
- Allow querying a surface for the set of presentation modes which can be easily switched between without requiring swapchain recreation.

### New Structures

- Extending VkPhysicalDeviceSurfaceInfo2KHR:
  - VkSurfacePresentModeEXT
- Extending VkSurfaceCapabilities2KHR:
  - VkSurfacePresentModeCompatibilityEXT
  - VkSurfacePresentScalingCapabilitiesEXT

### New Enums

- VkPresentGravityFlagBitsEXT
- VkPresentScalingFlagBitsEXT

### New Bitmasks

- VkPresentGravityFlagsEXT
- VkPresentScalingFlagsEXT

### New Enum Constants

- `VK_EXT_SURFACE_MAINTENANCE_1_EXTENSION_NAME`
- `VK_EXT_SURFACE_MAINTENANCE_1_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_SURFACE_PRESENT_MODE_COMPATIBILITY_EXT`
  - `VK_STRUCTURE_TYPE_SURFACE_PRESENT_MODE_EXT`
  - `VK_STRUCTURE_TYPE_SURFACE_PRESENT_SCALING_CAPABILITIES_EXT`

### Version History

- Revision 0, 2019-02-27 (Lionel Landwerlin)
  - Internal revisions
- Revision 1, 2022-11-09 (Shahbaz Youssefi)

- Add functionality and complete spec

## VK_EXT_swapchain_colorspace

### Name String

`VK_EXT_swapchain_colorspace`

### Extension Type

Instance extension

### Registered Extension Number

105

### Revision

5

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_surface

### Contact

- Courtney Goeltzenleuchter [GitHub]courtney-g

### Other Extension Metadata

### Last Modified Date

2019-04-26

### IP Status

No known IP claims.

### Contributors

- Courtney Goeltzenleuchter, Google

### Description

This extension expands VkColorSpaceKHR to add support for most standard color spaces beyond `VK_COLOR_SPACE_SRGB_NONLINEAR_KHR`. This extension also adds support for `VK_COLOR_SPACE_PASS_THROUGH_EXT` which allows applications to use color spaces not explicitly enumerated in VkColorSpaceKHR.

### New Enum Constants

- `VK_EXT_SWAPCHAIN_COLOR_SPACE_EXTENSION_NAME`
- `VK_EXT_SWAPCHAIN_COLOR_SPACE_SPEC_VERSION`
- Extending VkColorSpaceKHR:

- `VK_COLOR_SPACE_ADOBERGB_LINEAR_EXT`
- `VK_COLOR_SPACE_ADOBERGB_NONLINEAR_EXT`
- `VK_COLOR_SPACE_BT2020_LINEAR_EXT`
- `VK_COLOR_SPACE_BT709_LINEAR_EXT`
- `VK_COLOR_SPACE_BT709_NONLINEAR_EXT`
- `VK_COLOR_SPACE_DCI_P3_LINEAR_EXT`
- `VK_COLOR_SPACE_DCI_P3_NONLINEAR_EXT`
- `VK_COLOR_SPACE_DISPLAY_P3_LINEAR_EXT`
- `VK_COLOR_SPACE_DISPLAY_P3_NONLINEAR_EXT`
- `VK_COLOR_SPACE_DOLBYVISION_EXT`
- `VK_COLOR_SPACE_EXTENDED_SRGB_LINEAR_EXT`
- `VK_COLOR_SPACE_EXTENDED_SRGB_NONLINEAR_EXT`
- `VK_COLOR_SPACE_HDR10_HLG_EXT`
- `VK_COLOR_SPACE_HDR10_ST2084_EXT`
- `VK_COLOR_SPACE_PASS_THROUGH_EXT`

**Issues**

1) Does the spec need to specify which kinds of image formats support the color spaces?

**RESOLVED**: Pixel format is independent of color space (though some color spaces really want / need floating-point color components to be useful). Therefore, do not plan on documenting what formats support which color spaces. An application **can** call vkGetPhysicalDeviceSurfaceFormatsKHR to query what a particular implementation supports.

2) How does application determine if HW supports appropriate transfer function for a color space?

**RESOLVED**: Extension indicates that implementation **must** not do the OETF encoding if it is not sRGB. That responsibility falls to the application shaders. Any other native OETF / EOTF functions supported by an implementation can be described by separate extension.

**Version History**

- Revision 1, 2016-12-27 (Courtney Goeltzenleuchter)
  - Initial version
- Revision 2, 2017-01-19 (Courtney Goeltzenleuchter)
  - Add pass through and multiple options for BT2020.
  - Clean up some issues with equations not displaying properly.
- Revision 3, 2017-06-23 (Courtney Goeltzenleuchter)
  - Add extended sRGB non-linear enum.
- Revision 4, 2019-04-26 (Graeme Leese)

    - Clarify color space transfer function usage.
    - Refer to normative definitions in the Data Format Specification.
    - Clarify DCI-P3 and Display P3 usage.
- Revision 5, 2024-03-16 (Zehui Lin)
    - Fix interchanged concepts of EOTF and OETF.
    - Clarify that the presentation engine can accept the color spaces.

## VK_EXT_swapchain_maintenance1

### Name String

`VK_EXT_swapchain_maintenance1`

### Extension Type

Device extension

### Registered Extension Number

276

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_swapchain
and
VK_EXT_surface_maintenance1
and
  VK_KHR_get_physical_device_properties2
  or
  Vulkan Version 1.1

### Contact

- Shahbaz Youssefi [GitHub]syoussefi

### Extension Proposal

VK_EXT_swapchain_maintenance1

### Other Extension Metadata

### Last Modified Date

2022-10-28

### Contributors

- Jeff Juliano, NVIDIA

- Shahbaz Youssefi, Google
- Chris Forbes, Google
- Ian Elliott, Google
- Yiwei Zhang, Google
- Charlie Lao, Google
- Lina Versace, Google
- Ralph Potter, Samsung
- Igor Nazarov, Samsung
- Hyunchang Kim, Samsung
- Suenghwan Lee, Samsung
- Munseong Kang, Samsung
- Joonyong Park, Samsung
- Hans-Kristian Arntzen, Valve
- Lisa Wu, Arm
- Daniel Stone, Collabora
- Pan Gao, Huawei

### Description

`VK_EXT_swapchain_maintenance1` adds a collection of window system integration features that were intentionally left out or overlooked in the original `VK_KHR_swapchain` extension.

The new features are as follows:

- Specify a fence that will be signaled when the resources associated with a present operation **can** be safely destroyed.
- Allow changing the present mode a swapchain is using at per-present granularity.
- Allow applications to define the behavior when presenting a swapchain image to a surface with different dimensions than the image. Using this feature **may** allow implementations to avoid returning `VK_ERROR_OUT_OF_DATE_KHR` in this situation.
- Allow applications to defer swapchain memory allocation for improved startup time and memory footprint.
- Allow applications to release previously acquired images without presenting them.

### New Commands

- vkReleaseSwapchainImagesEXT

### New Structures

- VkReleaseSwapchainImagesInfoEXT

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceSwapchainMaintenance1FeaturesEXT
- Extending VkPresentInfoKHR:
  - VkSwapchainPresentFenceInfoEXT
  - VkSwapchainPresentModeInfoEXT
- Extending VkSwapchainCreateInfoKHR:
  - VkSwapchainPresentModesCreateInfoEXT
  - VkSwapchainPresentScalingCreateInfoEXT

#### New Enum Constants

- `VK_EXT_SWAPCHAIN_MAINTENANCE_1_EXTENSION_NAME`
- `VK_EXT_SWAPCHAIN_MAINTENANCE_1_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SWAPCHAIN_MAINTENANCE_1_FEATURES_EXT`
  - `VK_STRUCTURE_TYPE_RELEASE_SWAPCHAIN_IMAGES_INFO_EXT`
  - `VK_STRUCTURE_TYPE_SWAPCHAIN_PRESENT_FENCE_INFO_EXT`
  - `VK_STRUCTURE_TYPE_SWAPCHAIN_PRESENT_MODES_CREATE_INFO_EXT`
  - `VK_STRUCTURE_TYPE_SWAPCHAIN_PRESENT_MODE_INFO_EXT`
  - `VK_STRUCTURE_TYPE_SWAPCHAIN_PRESENT_SCALING_CREATE_INFO_EXT`
- Extending VkSwapchainCreateFlagBitsKHR:
  - `VK_SWAPCHAIN_CREATE_DEFERRED_MEMORY_ALLOCATION_BIT_EXT`

#### Version History

- Revision 0, 2019-05-28
  - Initial revisions
- Revision 1, 2022-08-21 (Shahbaz Youssefi)
  - Add functionality and complete spec

## VK_EXT_texel_buffer_alignment

#### Name String

`VK_EXT_texel_buffer_alignment`

#### Extension Type

Device extension

#### Registered Extension Number

282

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### Deprecation State

- *Promoted* to Vulkan 1.3

### Contact

- Jeff Bolz [GitHub]jeffbolznv

### Other Extension Metadata

### Last Modified Date

2019-06-06

### IP Status

No known IP claims.

### Contributors

- Jeff Bolz, NVIDIA

### Description

This extension adds more expressive alignment requirements for uniform and storage texel buffers. Some implementations have single texel alignment requirements that cannot be expressed via VkPhysicalDeviceLimits::`minTexelBufferOffsetAlignment`.

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceTexelBufferAlignmentFeaturesEXT
- Extending VkPhysicalDeviceProperties2:
  - VkPhysicalDeviceTexelBufferAlignmentPropertiesEXT

### New Enum Constants

- `VK_EXT_TEXEL_BUFFER_ALIGNMENT_EXTENSION_NAME`
- `VK_EXT_TEXEL_BUFFER_ALIGNMENT_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_TEXEL_BUFFER_ALIGNMENT_FEATURES_EXT`

- `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_TEXEL_BUFFER_ALIGNMENT_PROPERTIES_EXT`

#### Promotion to Vulkan 1.3

Functionality in this extension is included in core Vulkan 1.3, with the EXT suffix omitted. However, only the properties structure is promoted. The feature structure is not promoted and `texelBufferAlignment` is enabled if using a Vulkan 1.3 instance. The original type name is still available as an alias of the core functionality.

#### Version History

- Revision 1, 2019-06-06 (Jeff Bolz)
  - Initial draft

## VK_EXT_texture_compression_astc_hdr

#### Name String

`VK_EXT_texture_compression_astc_hdr`

#### Extension Type

Device extension

#### Registered Extension Number

67

#### Revision

1

#### Ratification Status

Not ratified

#### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

#### Deprecation State

- *Promoted* to Vulkan 1.3

#### Contact

- Jan-Harald Fredriksen [GitHub]janharaldfredriksen-arm

#### Other Extension Metadata

#### Last Modified Date

2019-05-28

#### IP Status

No known issues.

### Contributors

- Jan-Harald Fredriksen, Arm

### Description

This extension adds support for textures compressed using the Adaptive Scalable Texture Compression (ASTC) High Dynamic Range (HDR) profile.

When this extension is enabled, the HDR profile is supported for all ASTC formats listed in ASTC Compressed Image Formats.

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceTextureCompressionASTCHDRFeaturesEXT

### New Enum Constants

- `VK_EXT_TEXTURE_COMPRESSION_ASTC_HDR_EXTENSION_NAME`
- `VK_EXT_TEXTURE_COMPRESSION_ASTC_HDR_SPEC_VERSION`
- Extending VkFormat:
  - `VK_FORMAT_ASTC_10x10_SFLOAT_BLOCK_EXT`
  - `VK_FORMAT_ASTC_10x5_SFLOAT_BLOCK_EXT`
  - `VK_FORMAT_ASTC_10x6_SFLOAT_BLOCK_EXT`
  - `VK_FORMAT_ASTC_10x8_SFLOAT_BLOCK_EXT`
  - `VK_FORMAT_ASTC_12x10_SFLOAT_BLOCK_EXT`
  - `VK_FORMAT_ASTC_12x12_SFLOAT_BLOCK_EXT`
  - `VK_FORMAT_ASTC_4x4_SFLOAT_BLOCK_EXT`
  - `VK_FORMAT_ASTC_5x4_SFLOAT_BLOCK_EXT`
  - `VK_FORMAT_ASTC_5x5_SFLOAT_BLOCK_EXT`
  - `VK_FORMAT_ASTC_6x5_SFLOAT_BLOCK_EXT`
  - `VK_FORMAT_ASTC_6x6_SFLOAT_BLOCK_EXT`
  - `VK_FORMAT_ASTC_8x5_SFLOAT_BLOCK_EXT`
  - `VK_FORMAT_ASTC_8x6_SFLOAT_BLOCK_EXT`
  - `VK_FORMAT_ASTC_8x8_SFLOAT_BLOCK_EXT`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_TEXTURE_COMPRESSION_ASTC_HDR_FEATURES_EXT`

### Promotion to Vulkan 1.3

This extension has been partially promoted. Functionality in this extension is included in core Vulkan 1.3, with the EXT suffix omitted. However, the feature is made optional in Vulkan 1.3. The

original type, enum and command names are still available as aliases of the core functionality.

#### Issues

1) Should we add a feature or limit for this functionality?

Yes. It is consistent with the ASTC LDR support to add a feature like textureCompressionASTC_HDR.

The feature is strictly speaking redundant as long as this is just an extension; it would be sufficient to just enable the extension. But adding the feature is more forward-looking if wanted to make this an optional core feature in the future.

2) Should we introduce new format enums for HDR?

Yes. Vulkan 1.0 describes the ASTC format enums as UNORM, e.g. `VK_FORMAT_ASTC_4x4_UNORM_BLOCK`, so it is confusing to make these contain HDR data. Note that the OpenGL (ES) extensions did not make this distinction because a single ASTC HDR texture may contain both unorm and float blocks. Implementations **may** not be able to distinguish between LDR and HDR ASTC textures internally and just treat them as the same format, i.e. if this extension is supported then sampling from a `VK_FORMAT_ASTC_4x4_UNORM_BLOCK` image format **may** return HDR results. Applications **can** get predictable results by using the appropriate image format.

#### Version History

- Revision 1, 2019-05-28 (Jan-Harald Fredriksen)
  - Initial version

## VK_EXT_tooling_info

#### Name String

`VK_EXT_tooling_info`

#### Extension Type

Device extension

#### Registered Extension Number

246

#### Revision

1

#### Ratification Status

Not ratified

#### Extension and Version Dependencies

None

#### API Interactions

- Interacts with VK_EXT_debug_marker

- Interacts with VK_EXT_debug_report
- Interacts with VK_EXT_debug_utils

**Deprecation State**

- *Promoted* to Vulkan 1.3

**Contact**

- Tobias Hector [GitHub]tobski

**Other Extension Metadata**

**Last Modified Date**

2018-11-05

**Contributors**

- Rolando Caloca
- Matthaeus Chajdas
- Baldur Karlsson
- Daniel Rakos

**Description**

When an error occurs during application development, a common question is "What tools are actually running right now?" This extension adds the ability to query that information directly from the Vulkan implementation.

Outdated versions of one tool might not play nicely with another, or perhaps a tool is not actually running when it should have been. Trying to figure that out can cause headaches as it is necessary to consult each known tool to figure out what is going on — in some cases the tool might not even be known.

Typically, the expectation is that developers will simply print out this information for visual inspection when an issue occurs, however a small amount of semantic information about what the tool is doing is provided to help identify it programmatically. For example, if the advertised limits or features of an implementation are unexpected, is there a tool active which modifies these limits? Or if an application is providing debug markers, but the implementation is not actually doing anything with that information, this can quickly point that out.

**New Commands**

- vkGetPhysicalDeviceToolPropertiesEXT

**New Structures**

- VkPhysicalDeviceToolPropertiesEXT

**New Enums**

- VkToolPurposeFlagBitsEXT

### New Bitmasks

- VkToolPurposeFlagsEXT

### New Enum Constants

- VK_EXT_TOOLING_INFO_EXTENSION_NAME
- VK_EXT_TOOLING_INFO_SPEC_VERSION
- Extending VkStructureType:
  - VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_TOOL_PROPERTIES_EXT

If VK_EXT_debug_marker is supported:

- Extending VkToolPurposeFlagBits:
  - VK_TOOL_PURPOSE_DEBUG_MARKERS_BIT_EXT

If VK_EXT_debug_report is supported:

- Extending VkToolPurposeFlagBits:
  - VK_TOOL_PURPOSE_DEBUG_REPORTING_BIT_EXT

If VK_EXT_debug_utils is supported:

- Extending VkToolPurposeFlagBits:
  - VK_TOOL_PURPOSE_DEBUG_MARKERS_BIT_EXT
  - VK_TOOL_PURPOSE_DEBUG_REPORTING_BIT_EXT

### Promotion to Vulkan 1.3

Functionality in this extension is included in core Vulkan 1.3, with the EXT suffix omitted. The original type, enum and command names are still available as aliases of the core functionality.

### Examples

*Printing Tool Information*

```
uint32_t num_tools;
VkPhysicalDeviceToolPropertiesEXT *pToolProperties;
vkGetPhysicalDeviceToolPropertiesEXT(physicalDevice, &num_tools, NULL);

pToolProperties =
(VkPhysicalDeviceToolPropertiesEXT*)malloc(sizeof(VkPhysicalDeviceToolPropertiesEXT) *
num_tools);

vkGetPhysicalDeviceToolPropertiesEXT(physicalDevice, &num_tools, pToolProperties);

for (int i = 0; i < num_tools; ++i) {
    printf("%s:\n", pToolProperties[i].name);
    printf("Version:\n");
```

```
    printf("%s:\n", pToolProperties[i].version);
    printf("Description:\n");
    printf("\t%s\n", pToolProperties[i].description);
    printf("Purposes:\n");
    printf("\t%s\n", VkToolPurposeFlagBitsEXT_to_string(pToolProperties[i].purposes));
    if (strnlen_s(pToolProperties[i].layer,VK_MAX_EXTENSION_NAME_SIZE) > 0) {
        printf("Corresponding Layer:\n");
        printf("\t%s\n", pToolProperties[i].layer);
    }
}
```

#### Issues

1) Why is this information separate from the layer mechanism?

Some tooling may be built into a driver, or be part of the Vulkan loader etc. Tying this information directly to layers would have been awkward at best.

#### Version History

- Revision 1, 2018-11-05 (Tobias Hector)
  - Initial draft

## VK_EXT_transform_feedback

#### Name String

`VK_EXT_transform_feedback`

#### Extension Type

Device extension

#### Registered Extension Number

29

#### Revision

1

#### Ratification Status

Ratified

#### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

#### Special Uses

- OpenGL / ES support
- D3D support

- Developer tools

**Contact**

- Piers Daniell [GitHub]pdaniell-nv

**Other Extension Metadata**

**Last Modified Date**

2018-10-09

**Contributors**

- Baldur Karlsson, Valve
- Boris Zanin, Mobica
- Daniel Rakos, AMD
- Donald Scorgie, Imagination
- Henri Verbeet, CodeWeavers
- Jan-Harald Fredriksen, Arm
- Faith Ekstrand, Intel
- Jeff Bolz, NVIDIA
- Jesse Barker, Unity
- Jesse Hall, Google
- Pierre-Loup Griffais, Valve
- Philip Rebohle, DXVK
- Ruihao Zhang, Qualcomm
- Samuel Pitoiset, Valve
- Slawomir Grajewski, Intel
- Stu Smith, Imagination Technologies

**Description**

This extension adds transform feedback to the Vulkan API by exposing the SPIR-V `TransformFeedback` and `GeometryStreams` capabilities to capture vertex, tessellation or geometry shader outputs to one or more buffers. It adds API functionality to bind transform feedback buffers to capture the primitives emitted by the graphics pipeline from SPIR-V outputs decorated for transform feedback. The transform feedback capture can be paused and resumed by way of storing and retrieving a byte counter. The captured data can be drawn again where the vertex count is derived from the byte counter without CPU intervention. If the implementation is capable, a vertex stream other than zero can be rasterized.

All these features are designed to match the full capabilities of OpenGL core transform feedback functionality and beyond. Many of the features are optional to allow base OpenGL ES GPUs to also implement this extension.

The primary purpose of the functionality exposed by this extension is to support translation layers from other 3D APIs. This functionality is not considered forward looking, and is not expected to be promoted to a KHR extension or to core Vulkan. Unless this is needed for translation, it is recommended that developers use alternative techniques of using the GPU to process and capture vertex data.

### New Commands

- vkCmdBeginQueryIndexedEXT
- vkCmdBeginTransformFeedbackEXT
- vkCmdBindTransformFeedbackBuffersEXT
- vkCmdDrawIndirectByteCountEXT
- vkCmdEndQueryIndexedEXT
- vkCmdEndTransformFeedbackEXT

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceTransformFeedbackFeaturesEXT
- Extending VkPhysicalDeviceProperties2:
  - VkPhysicalDeviceTransformFeedbackPropertiesEXT
- Extending VkPipelineRasterizationStateCreateInfo:
  - VkPipelineRasterizationStateStreamCreateInfoEXT

### New Bitmasks

- VkPipelineRasterizationStateStreamCreateFlagsEXT

### New Enum Constants

- `VK_EXT_TRANSFORM_FEEDBACK_EXTENSION_NAME`
- `VK_EXT_TRANSFORM_FEEDBACK_SPEC_VERSION`
- Extending VkAccessFlagBits:
  - `VK_ACCESS_TRANSFORM_FEEDBACK_COUNTER_READ_BIT_EXT`
  - `VK_ACCESS_TRANSFORM_FEEDBACK_COUNTER_WRITE_BIT_EXT`
  - `VK_ACCESS_TRANSFORM_FEEDBACK_WRITE_BIT_EXT`
- Extending VkBufferUsageFlagBits:
  - `VK_BUFFER_USAGE_TRANSFORM_FEEDBACK_BUFFER_BIT_EXT`
  - `VK_BUFFER_USAGE_TRANSFORM_FEEDBACK_COUNTER_BUFFER_BIT_EXT`
- Extending VkPipelineStageFlagBits:
  - `VK_PIPELINE_STAGE_TRANSFORM_FEEDBACK_BIT_EXT`

- Extending VkQueryType:
  - VK_QUERY_TYPE_TRANSFORM_FEEDBACK_STREAM_EXT
- Extending VkStructureType:
  - VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_TRANSFORM_FEEDBACK_FEATURES_EXT
  - VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_TRANSFORM_FEEDBACK_PROPERTIES_EXT
  - VK_STRUCTURE_TYPE_PIPELINE_RASTERIZATION_STATE_STREAM_CREATE_INFO_EXT

#### Issues

1) Should we include pause/resume functionality?

**RESOLVED**: Yes, this is needed to ease layering other APIs which have this functionality. To pause use vkCmdEndTransformFeedbackEXT and provide valid buffer handles in the pCounterBuffers array and offsets in the pCounterBufferOffsets array for the implementation to save the resume points. Then to resume use vkCmdBeginTransformFeedbackEXT with the previous pCounterBuffers and pCounterBufferOffsets values. Between the pause and resume there needs to be a memory barrier for the counter buffers with a source access of VK_ACCESS_TRANSFORM_FEEDBACK_COUNTER_WRITE_BIT_EXT at pipeline stage VK_PIPELINE_STAGE_TRANSFORM_FEEDBACK_BIT_EXT to a destination access of VK_ACCESS_TRANSFORM_FEEDBACK_COUNTER_READ_BIT_EXT at pipeline stage VK_PIPELINE_STAGE_TRANSFORM_FEEDBACK_BIT_EXT.

2) How does this interact with multiview?

**RESOLVED**: Transform feedback cannot be made active in a render pass with multiview enabled.

3) How should queries be done?

**RESOLVED**: There is a new query type VK_QUERY_TYPE_TRANSFORM_FEEDBACK_STREAM_EXT. A query pool created with this type will capture 2 integers - numPrimitivesWritten and numPrimitivesNeeded - for the specified vertex stream output from the last pre-rasterization shader stage. The vertex stream output queried is zero by default, but can be specified with the new vkCmdBeginQueryIndexedEXT and vkCmdEndQueryIndexedEXT commands.

#### Version History

- Revision 1, 2018-10-09 (Piers Daniell)
  - Internal revisions

## VK_EXT_validation_cache

#### Name String

VK_EXT_validation_cache

#### Extension Type

Device extension

### Registered Extension Number

161

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

None

### Contact

- Cort Stratton [GitHub]cdwfs

### Other Extension Metadata

### Last Modified Date

2017-08-29

### IP Status

No known IP claims.

### Contributors

- Cort Stratton, Google
- Chris Forbes, Google

### Description

This extension provides a mechanism for caching the results of potentially expensive internal validation operations across multiple runs of a Vulkan application. At the core is the VkValidationCacheEXT object type, which is managed similarly to the existing VkPipelineCache.

The new struct VkShaderModuleValidationCacheCreateInfoEXT can be included in the `pNext` chain at vkCreateShaderModule time. It contains a VkValidationCacheEXT to use when validating the VkShaderModule.

### New Object Types

- VkValidationCacheEXT

### New Commands

- vkCreateValidationCacheEXT
- vkDestroyValidationCacheEXT
- vkGetValidationCacheDataEXT
- vkMergeValidationCachesEXT

### New Structures

- VkValidationCacheCreateInfoEXT
- Extending VkShaderModuleCreateInfo, VkPipelineShaderStageCreateInfo:
  - VkShaderModuleValidationCacheCreateInfoEXT

### New Enums

- VkValidationCacheHeaderVersionEXT

### New Bitmasks

- VkValidationCacheCreateFlagsEXT

### New Enum Constants

- `VK_EXT_VALIDATION_CACHE_EXTENSION_NAME`
- `VK_EXT_VALIDATION_CACHE_SPEC_VERSION`
- Extending VkObjectType:
  - `VK_OBJECT_TYPE_VALIDATION_CACHE_EXT`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_SHADER_MODULE_VALIDATION_CACHE_CREATE_INFO_EXT`
  - `VK_STRUCTURE_TYPE_VALIDATION_CACHE_CREATE_INFO_EXT`

### Version History

- Revision 1, 2017-08-29 (Cort Stratton)
  - Initial draft

## VK_EXT_vertex_input_dynamic_state

### Name String

`VK_EXT_vertex_input_dynamic_state`

### Extension Type

Device extension

### Registered Extension Number

353

### Revision

2

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### Contact

- Piers Daniell [GitHub]pdaniell-nv

### Other Extension Metadata

### Last Modified Date

2020-08-21

### IP Status

No known IP claims.

### Contributors

- Jeff Bolz, NVIDIA
- Spencer Fricke, Samsung
- Stu Smith, AMD

### Description

One of the states that contributes to the combinatorial explosion of pipeline state objects that need to be created, is the vertex input binding and attribute descriptions. By allowing them to be dynamic applications may reduce the number of pipeline objects they need to create.

This extension adds dynamic state support for what is normally static state in VkPipelineVertexInputStateCreateInfo.

### New Commands

- vkCmdSetVertexInputEXT

### New Structures

- VkVertexInputAttributeDescription2EXT
- VkVertexInputBindingDescription2EXT
- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceVertexInputDynamicStateFeaturesEXT

### New Enum Constants

- `VK_EXT_VERTEX_INPUT_DYNAMIC_STATE_EXTENSION_NAME`
- `VK_EXT_VERTEX_INPUT_DYNAMIC_STATE_SPEC_VERSION`
- Extending VkDynamicState:

    - VK_DYNAMIC_STATE_VERTEX_INPUT_EXT
- Extending VkStructureType:
    - VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_VERTEX_INPUT_DYNAMIC_STATE_FEATURES_EXT
    - VK_STRUCTURE_TYPE_VERTEX_INPUT_ATTRIBUTE_DESCRIPTION_2_EXT
    - VK_STRUCTURE_TYPE_VERTEX_INPUT_BINDING_DESCRIPTION_2_EXT

#### Version History

- Revision 2, 2020-11-05 (Piers Daniell)
    - Make VkVertexInputBindingDescription2EXT extensible
    - Add new VkVertexInputAttributeDescription2EXT struct for the pVertexAttributeDescriptions parameter to vkCmdSetVertexInputEXT so it is also extensible
- Revision 1, 2020-08-21 (Piers Daniell)
    - Internal revisions

## VK_EXT_ycbcr_2plane_444_formats

#### Name String

VK_EXT_ycbcr_2plane_444_formats

#### Extension Type

Device extension

#### Registered Extension Number

331

#### Revision

1

#### Ratification Status

Not ratified

#### Extension and Version Dependencies

VK_KHR_sampler_ycbcr_conversion
or
Vulkan Version 1.1

#### Deprecation State

- *Promoted* to Vulkan 1.3

#### Contact

- Tony Zlatinski [GitHub]tzlatinski

#### Other Extension Metadata

### Last Modified Date

2020-07-28

### IP Status

No known IP claims.

### Contributors

- Piers Daniell, NVIDIA
- Ping Liu, Intel

### Description

This extension adds some $Y'C_BC_R$ formats that are in common use for video encode and decode, but were not part of the `VK_KHR_sampler_ycbcr_conversion` extension.

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceYcbcr2Plane444FormatsFeaturesEXT

### New Enum Constants

- `VK_EXT_YCBCR_2PLANE_444_FORMATS_EXTENSION_NAME`
- `VK_EXT_YCBCR_2PLANE_444_FORMATS_SPEC_VERSION`
- Extending VkFormat:
  - `VK_FORMAT_G10X6_B10X6R10X6_2PLANE_444_UNORM_3PACK16_EXT`
  - `VK_FORMAT_G12X4_B12X4R12X4_2PLANE_444_UNORM_3PACK16_EXT`
  - `VK_FORMAT_G16_B16R16_2PLANE_444_UNORM_EXT`
  - `VK_FORMAT_G8_B8R8_2PLANE_444_UNORM_EXT`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_YCBCR_2_PLANE_444_FORMATS_FEATURES_EXT`

### Promotion to Vulkan 1.3

This extension has been partially promoted. The format enumerants introduced by the extension are included in core Vulkan 1.3, with the EXT suffix omitted. However, runtime support for these formats is optional in core Vulkan 1.3, while if this extension is supported, runtime support is mandatory. The feature structure is not promoted. The original enum names are still available as aliases of the core functionality.

### Version History

- Revision 1, 2020-03-08 (Piers Daniell)
  - Initial draft

## VK_EXT_ycbcr_image_arrays

### Name String

`VK_EXT_ycbcr_image_arrays`

### Extension Type

Device extension

### Registered Extension Number

253

### Revision

1

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_sampler_ycbcr_conversion
or
Vulkan Version 1.1

### Contact

- Piers Daniell [GitHub]pdaniell-nv

### Other Extension Metadata

### Last Modified Date

2019-01-15

### Contributors

- Piers Daniell, NVIDIA

### Description

This extension allows images of a format that requires $Y'C_BC_R$ conversion to be created with multiple array layers, which is otherwise restricted.

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceYcbcrImageArraysFeaturesEXT

### New Enum Constants

- `VK_EXT_YCBCR_IMAGE_ARRAYS_EXTENSION_NAME`
- `VK_EXT_YCBCR_IMAGE_ARRAYS_SPEC_VERSION`
- Extending VkStructureType:

- `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_YCBCR_IMAGE_ARRAYS_FEATURES_EXT`

### Version History

- Revision 1, 2019-01-15 (Piers Daniell)
  - Initial revision

## VK_AMD_anti_lag

### Name String

`VK_AMD_anti_lag`

### Extension Type

Device extension

### Registered Extension Number

477

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

None

### Contact

- Stu Smith

### Extension Proposal

VK_AMD_anti_lag

### Other Extension Metadata

### Last Modified Date

2024-06-06

### IP Status

No known IP claims.

### Contributors

- Tobias Hector, AMD
- Stuart Smith, AMD
- Arkadiusz Sarwa, AMD

### Description

This extension automatically paces the CPU to make sure it does not get too far ahead of the GPU, reducing the latency between inputs received and updates on the screen. Additionally, Anti-Lag+ offers applications the ability to inform the driver when input processing begins, in order to align the timing of display updates, enabling even lower latency between receiving input and displaying on the screen.

### New Commands

- vkAntiLagUpdateAMD

### New Structures

- VkAntiLagDataAMD
- VkAntiLagPresentationInfoAMD
- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceAntiLagFeaturesAMD

### New Enums

- VkAntiLagModeAMD
- VkAntiLagStageAMD

### New Enum Constants

- `VK_AMD_ANTI_LAG_EXTENSION_NAME`
- `VK_AMD_ANTI_LAG_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_ANTI_LAG_DATA_AMD`
  - `VK_STRUCTURE_TYPE_ANTI_LAG_PRESENTATION_INFO_AMD`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_ANTI_LAG_FEATURES_AMD`

### Version History

- Revision 1, 2024-06-06 (Arkadiusz Sarw)
  - Initial version

## VK_AMD_buffer_marker

### Name String

`VK_AMD_buffer_marker`

### Extension Type

Device extension

### Registered Extension Number

180

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

None

### Special Use

- Developer tools

### Contact

- Daniel Rakos [GitHub]drakos-amd

### Other Extension Metadata

### Last Modified Date

2018-01-26

### IP Status

No known IP claims.

### Contributors

- Matthaeus G. Chajdas, AMD
- Jaakko Konttinen, AMD
- Daniel Rakos, AMD

### Description

This extension adds a new operation to execute pipelined writes of small marker values into a `VkBuffer` object.

The primary purpose of these markers is to facilitate the development of debugging tools for tracking which pipelined command contributed to device loss.

### New Commands

- vkCmdWriteBufferMarkerAMD

### New Enum Constants

- `VK_AMD_BUFFER_MARKER_EXTENSION_NAME`
- `VK_AMD_BUFFER_MARKER_SPEC_VERSION`

### Examples

None.

### Version History

- Revision 1, 2018-01-26 (Jaakko Konttinen)
  - Initial revision

## VK_AMD_device_coherent_memory

### Name String

`VK_AMD_device_coherent_memory`

### Extension Type

Device extension

### Registered Extension Number

230

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### Contact

- Tobias Hector [GitHub]tobski

### Other Extension Metadata

### Last Modified Date

2019-02-04

### Contributors

- Ping Fu, AMD
- Timothy Lottes, AMD
- Tobias Hector, AMD

### Description

This extension adds the device coherent and device uncached memory types. Any device accesses to device coherent memory are automatically made visible to any other device access. Device

uncached memory indicates to applications that caches are disabled for a particular memory type, which guarantees device coherence.

Device coherent and uncached memory are expected to have lower performance for general access than non-device coherent memory, but can be useful in certain scenarios; particularly so for debugging.

#### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceCoherentMemoryFeaturesAMD

#### New Enum Constants

- `VK_AMD_DEVICE_COHERENT_MEMORY_EXTENSION_NAME`
- `VK_AMD_DEVICE_COHERENT_MEMORY_SPEC_VERSION`
- Extending VkMemoryPropertyFlagBits:
  - `VK_MEMORY_PROPERTY_DEVICE_COHERENT_BIT_AMD`
  - `VK_MEMORY_PROPERTY_DEVICE_UNCACHED_BIT_AMD`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_COHERENT_MEMORY_FEATURES_AMD`

#### Version History

- Revision 1, 2019-02-04 (Tobias Hector)
  - Initial revision

### VK_AMD_display_native_hdr

#### Name String

`VK_AMD_display_native_hdr`

#### Extension Type

Device extension

#### Registered Extension Number

214

#### Revision

1

#### Ratification Status

Not ratified

#### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2

or
Vulkan Version 1.1
and
VK_KHR_get_surface_capabilities2
and
VK_KHR_swapchain

### Contact

- Matthaeus G. Chajdas [GitHub]anteru

### Other Extension Metadata

### Last Modified Date

2018-12-18

### IP Status

No known IP claims.

### Contributors

- Matthaeus G. Chajdas, AMD
- Aaron Hagan, AMD
- Aric Cyr, AMD
- Timothy Lottes, AMD
- Derrick Owens, AMD
- Daniel Rakos, AMD

### Description

This extension introduces the following display native HDR features to Vulkan:

- A new VkColorSpaceKHR enum for setting the native display color space. For example, this color space would be set by the swapchain to use the native color space in Freesync2 displays.
- Local dimming control

### New Commands

- vkSetLocalDimmingAMD

### New Structures

- Extending VkSurfaceCapabilities2KHR:
  - VkDisplayNativeHdrSurfaceCapabilitiesAMD
- Extending VkSwapchainCreateInfoKHR:
  - VkSwapchainDisplayNativeHdrCreateInfoAMD

### New Enum Constants

- `VK_AMD_DISPLAY_NATIVE_HDR_EXTENSION_NAME`
- `VK_AMD_DISPLAY_NATIVE_HDR_SPEC_VERSION`
- Extending VkColorSpaceKHR:
  - `VK_COLOR_SPACE_DISPLAY_NATIVE_AMD`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_DISPLAY_NATIVE_HDR_SURFACE_CAPABILITIES_AMD`
  - `VK_STRUCTURE_TYPE_SWAPCHAIN_DISPLAY_NATIVE_HDR_CREATE_INFO_AMD`

### Issues

None.

### Examples

None.

### Version History

- Revision 1, 2018-12-18 (Daniel Rakos)
  - Initial revision

## VK_AMD_gcn_shader

### Name String

`VK_AMD_gcn_shader`

### Extension Type

Device extension

### Registered Extension Number

26

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

None

### SPIR-V Dependencies

- SPV_AMD_gcn_shader

### Contact

- Dominik Witczak [GitHub]dominikwitczakamd

### Other Extension Metadata

### Last Modified Date

2016-05-30

### IP Status

No known IP claims.

### Interactions and External Dependencies

- This extension provides API support for `GL_AMD_gcn_shader`

### Contributors

- Dominik Witczak, AMD
- Daniel Rakos, AMD
- Rex Xu, AMD
- Graham Sellers, AMD

### Description

This extension adds support for the following SPIR-V extension in Vulkan:

- `SPV_AMD_gcn_shader`

### New Enum Constants

- `VK_AMD_GCN_SHADER_EXTENSION_NAME`
- `VK_AMD_GCN_SHADER_SPEC_VERSION`

### Version History

- Revision 1, 2016-05-30 (Dominik Witczak)
  - Initial draft

## VK_AMD_memory_overallocation_behavior

### Name String

`VK_AMD_memory_overallocation_behavior`

### Extension Type

Device extension

### Registered Extension Number

190

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

None

### Contact

- Martin Dinkov [GitHub]mdinkov

### Other Extension Metadata

### Last Modified Date

2018-09-19

### IP Status

No known IP claims.

### Contributors

- Martin Dinkov, AMD
- Matthaeus Chajdas, AMD
- Daniel Rakos, AMD
- Jon Campbell, AMD

### Description

This extension allows controlling whether explicit overallocation beyond the device memory heap sizes (reported by VkPhysicalDeviceMemoryProperties) is allowed or not. Overallocation may lead to performance loss and is not supported for all platforms.

### New Structures

- Extending VkDeviceCreateInfo:
  - VkDeviceMemoryOverallocationCreateInfoAMD

### New Enums

- VkMemoryOverallocationBehaviorAMD

### New Enum Constants

- `VK_AMD_MEMORY_OVERALLOCATION_BEHAVIOR_EXTENSION_NAME`
- `VK_AMD_MEMORY_OVERALLOCATION_BEHAVIOR_SPEC_VERSION`
- Extending VkStructureType:

- `VK_STRUCTURE_TYPE_DEVICE_MEMORY_OVERALLOCATION_CREATE_INFO_AMD`

### Version History

- Revision 1, 2018-09-19 (Martin Dinkov)
  - Initial draft.

## VK_AMD_mixed_attachment_samples

### Name String

`VK_AMD_mixed_attachment_samples`

### Extension Type

Device extension

### Registered Extension Number

137

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

None

### Contact

- Matthaeus G. Chajdas [GitHub]anteru

### Other Extension Metadata

### Last Modified Date

2017-07-24

### Contributors

- Mais Alnasser, AMD
- Matthaeus G. Chajdas, AMD
- Maciej Jesionowski, AMD
- Daniel Rakos, AMD

### Description

This extension enables applications to use multisampled rendering with a depth/stencil sample count that is larger than the color sample count. Having a depth/stencil sample count larger than the color sample count allows maintaining geometry and coverage information at a higher sample rate than color information. All samples are depth/stencil tested, but only the first color sample

count number of samples get a corresponding color output.

### New Enum Constants

- `VK_AMD_MIXED_ATTACHMENT_SAMPLES_EXTENSION_NAME`
- `VK_AMD_MIXED_ATTACHMENT_SAMPLES_SPEC_VERSION`

### Issues

None.

### Version History

- Revision 1, 2017-07-24 (Daniel Rakos)
  - Internal revisions

## VK_AMD_pipeline_compiler_control

### Name String

`VK_AMD_pipeline_compiler_control`

### Extension Type

Device extension

### Registered Extension Number

184

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

None

### Contact

- Matthaeus G. Chajdas [GitHub]anteru

### Other Extension Metadata

### Last Modified Date

2019-07-26

### IP Status

No known IP claims.

### Contributors

- Matthaeus G. Chajdas, AMD
- Daniel Rakos, AMD
- Maciej Jesionowski, AMD
- Tobias Hector, AMD

### Description

This extension introduces VkPipelineCompilerControlCreateInfoAMD structure that can be chained to a pipeline's creation information to specify additional flags that affect pipeline compilation.

### New Structures

- Extending VkGraphicsPipelineCreateInfo, VkComputePipelineCreateInfo, VkExecutionGraphPipelineCreateInfoAMDX:
  - VkPipelineCompilerControlCreateInfoAMD

### New Enums

- VkPipelineCompilerControlFlagBitsAMD

### New Bitmasks

- VkPipelineCompilerControlFlagsAMD

### New Enum Constants

- `VK_AMD_PIPELINE_COMPILER_CONTROL_EXTENSION_NAME`
- `VK_AMD_PIPELINE_COMPILER_CONTROL_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PIPELINE_COMPILER_CONTROL_CREATE_INFO_AMD`

### Issues

None.

### Examples

None.

### Version History

- Revision 1, 2019-07-26 (Tobias Hector)
  - Initial revision.

## VK_AMD_rasterization_order

### Name String

`VK_AMD_rasterization_order`

### Extension Type

Device extension

### Registered Extension Number

19

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

None

### Contact

- Daniel Rakos [GitHub]drakos-amd

### Other Extension Metadata

### Last Modified Date

2016-04-25

### IP Status

No known IP claims.

### Contributors

- Matthaeus G. Chajdas, AMD
- Jaakko Konttinen, AMD
- Daniel Rakos, AMD
- Graham Sellers, AMD
- Dominik Witczak, AMD

### Description

This extension introduces the possibility for the application to control the order of primitive rasterization. In unextended Vulkan, the following stages are guaranteed to execute in *API order*:

- depth bounds test
- stencil test, stencil op, and stencil write
- depth test and depth write

- occlusion queries
- blending, logic op, and color write

This extension enables applications to opt into a relaxed, implementation defined primitive rasterization order that may allow better parallel processing of primitives and thus enabling higher primitive throughput. It is applicable in cases where the primitive rasterization order is known to not affect the output of the rendering or any differences caused by a different rasterization order are not a concern from the point of view of the application's purpose.

A few examples of cases when using the relaxed primitive rasterization order would not have an effect on the final rendering:

- If the primitives rendered are known to not overlap in framebuffer space.
- If depth testing is used with a comparison operator of `VK_COMPARE_OP_LESS`, `VK_COMPARE_OP_LESS_OR_EQUAL`, `VK_COMPARE_OP_GREATER`, or `VK_COMPARE_OP_GREATER_OR_EQUAL`, and the primitives rendered are known to not overlap in clip space.
- If depth testing is not used and blending is enabled for all attachments with a commutative blend operator.

**New Structures**

- Extending VkPipelineRasterizationStateCreateInfo:
  - VkPipelineRasterizationStateRasterizationOrderAMD

**New Enums**

- VkRasterizationOrderAMD

**New Enum Constants**

- `VK_AMD_RASTERIZATION_ORDER_EXTENSION_NAME`
- `VK_AMD_RASTERIZATION_ORDER_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PIPELINE_RASTERIZATION_STATE_RASTERIZATION_ORDER_AMD`

**Issues**

1) How is this extension useful to application developers?

**RESOLVED**: Allows them to increase primitive throughput for cases when strict API order rasterization is not important due to the nature of the content, the configuration used, or the requirements towards the output of the rendering.

2) How does this extension interact with content optimizations aiming to reduce overdraw by appropriately ordering the input primitives?

**RESOLVED**: While the relaxed rasterization order might somewhat limit the effectiveness of such content optimizations, most of the benefits of it are expected to be retained even when the relaxed

rasterization order is used, so applications **should** still apply these optimizations even if they intend to use the extension.

3) Are there any guarantees about the primitive rasterization order when using the new relaxed mode?

**RESOLVED**: No. In this case the rasterization order is completely implementation-dependent, but in practice it is expected to partially still follow the order of incoming primitives.

4) Does the new relaxed rasterization order have any adverse effect on repeatability and other invariance rules of the API?

**RESOLVED**: Yes, in the sense that it extends the list of exceptions when the repeatability requirement does not apply.

### Examples

None

### Issues

None

### Version History

- Revision 1, 2016-04-25 (Daniel Rakos)
  - Initial draft.

## VK_AMD_shader_ballot

### Name String

`VK_AMD_shader_ballot`

### Extension Type

Device extension

### Registered Extension Number

38

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

None

### SPIR-V Dependencies

- SPV_AMD_shader_ballot

### Contact

- Dominik Witczak [GitHub]dominikwitczakamd

### Other Extension Metadata

### Last Modified Date

2016-09-19

### IP Status

No known IP claims.

### Interactions and External Dependencies

- This extension provides API support for `GL_AMD_shader_ballot`

### Contributors

- Qun Lin, AMD
- Graham Sellers, AMD
- Daniel Rakos, AMD
- Rex Xu, AMD
- Dominik Witczak, AMD
- Matthäus G. Chajdas, AMD

### Description

This extension adds support for the following SPIR-V extension in Vulkan:

- `SPV_AMD_shader_ballot`

### New Enum Constants

- `VK_AMD_SHADER_BALLOT_EXTENSION_NAME`
- `VK_AMD_SHADER_BALLOT_SPEC_VERSION`

### Version History

- Revision 1, 2016-09-19 (Dominik Witczak)
  - Initial draft

## VK_AMD_shader_core_properties

### Name String

`VK_AMD_shader_core_properties`

### Extension Type

Device extension

### Registered Extension Number

186

### Revision

2

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### Contact

- Martin Dinkov [GitHub]mdinkov

### Other Extension Metadata

### Last Modified Date

2019-06-25

### IP Status

No known IP claims.

### Contributors

- Martin Dinkov, AMD
- Matthaeus G. Chajdas, AMD

### Description

This extension exposes shader core properties for a target physical device through the `VK_KHR_get_physical_device_properties2` extension. Please refer to the example below for proper usage.

### New Structures

- Extending VkPhysicalDeviceProperties2:
  - VkPhysicalDeviceShaderCorePropertiesAMD

### New Enum Constants

- `VK_AMD_SHADER_CORE_PROPERTIES_EXTENSION_NAME`
- `VK_AMD_SHADER_CORE_PROPERTIES_SPEC_VERSION`

- Extending VkStructureType:
  - VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_CORE_PROPERTIES_AMD

### Examples

This example retrieves the shader core properties for a physical device.

```
extern VkInstance       instance;

PFN_vkGetPhysicalDeviceProperties2 pfnVkGetPhysicalDeviceProperties2 =
    reinterpret_cast<PFN_vkGetPhysicalDeviceProperties2>
    (vkGetInstanceProcAddr(instance, "vkGetPhysicalDeviceProperties2") );

VkPhysicalDeviceProperties2             general_props;
VkPhysicalDeviceShaderCorePropertiesAMD shader_core_properties;

shader_core_properties.pNext = nullptr;
shader_core_properties.sType =
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_CORE_PROPERTIES_AMD;

general_props.pNext = &shader_core_properties;
general_props.sType = VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PROPERTIES_2;

// After this call, shader_core_properties has been populated
pfnVkGetPhysicalDeviceProperties2(device, &general_props);

printf("Number of shader engines: %d\n",
    m_shader_core_properties.shader_engine_count =
    shader_core_properties.shaderEngineCount;
printf("Number of shader arrays: %d\n",
    m_shader_core_properties.shader_arrays_per_engine_count =
    shader_core_properties.shaderArraysPerEngineCount;
printf("Number of CUs per shader array: %d\n",
    m_shader_core_properties.compute_units_per_shader_array =
    shader_core_properties.computeUnitsPerShaderArray;
printf("Number of SIMDs per compute unit: %d\n",
    m_shader_core_properties.simd_per_compute_unit =
    shader_core_properties.simdPerComputeUnit;
printf("Number of wavefront slots in each SIMD: %d\n",
    m_shader_core_properties.wavefronts_per_simd =
    shader_core_properties.wavefrontsPerSimd;
printf("Number of threads per wavefront: %d\n",
    m_shader_core_properties.wavefront_size =
    shader_core_properties.wavefrontSize;
printf("Number of physical SGPRs per SIMD: %d\n",
    m_shader_core_properties.sgprs_per_simd =
    shader_core_properties.sgprsPerSimd;
printf("Minimum number of SGPRs that can be allocated by a wave: %d\n",
    m_shader_core_properties.min_sgpr_allocation =
    shader_core_properties.minSgprAllocation;
```

```
printf("Number of available SGPRs: %d\n",
    m_shader_core_properties.max_sgpr_allocation =
    shader_core_properties.maxSgprAllocation;
printf("SGPRs are allocated in groups of this size: %d\n",
    m_shader_core_properties.sgpr_allocation_granularity =
    shader_core_properties.sgprAllocationGranularity;
printf("Number of physical VGPRs per SIMD: %d\n",
    m_shader_core_properties.vgprs_per_simd =
    shader_core_properties.vgprsPerSimd;
printf("Minimum number of VGPRs that can be allocated by a wave: %d\n",
    m_shader_core_properties.min_vgpr_allocation =
    shader_core_properties.minVgprAllocation;
printf("Number of available VGPRs: %d\n",
    m_shader_core_properties.max_vgpr_allocation =
    shader_core_properties.maxVgprAllocation;
printf("VGPRs are allocated in groups of this size: %d\n",
    m_shader_core_properties.vgpr_allocation_granularity =
    shader_core_properties.vgprAllocationGranularity;
```

### Version History

- Revision 2, 2019-06-25 (Matthaeus G. Chajdas)
  - Clarified the meaning of a few fields.
- Revision 1, 2018-02-15 (Martin Dinkov)
  - Initial draft.

## VK_AMD_shader_core_properties2

### Name String

`VK_AMD_shader_core_properties2`

### Extension Type

Device extension

### Registered Extension Number

228

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_AMD_shader_core_properties

### Contact

- Matthaeus G. Chajdas [GitHub]anteru

### Other Extension Metadata

### Last Modified Date

2019-07-26

### IP Status

No known IP claims.

### Contributors

- Matthaeus G. Chajdas, AMD
- Tobias Hector, AMD

### Description

This extension exposes additional shader core properties for a target physical device through the `VK_KHR_get_physical_device_properties2` extension.

### New Structures

- Extending VkPhysicalDeviceProperties2:
  - VkPhysicalDeviceShaderCoreProperties2AMD

### New Enums

- VkShaderCorePropertiesFlagBitsAMD

### New Bitmasks

- VkShaderCorePropertiesFlagsAMD

### New Enum Constants

- `VK_AMD_SHADER_CORE_PROPERTIES_2_EXTENSION_NAME`
- `VK_AMD_SHADER_CORE_PROPERTIES_2_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_CORE_PROPERTIES_2_AMD`

### Examples

None.

### Version History

- Revision 1, 2019-07-26 (Matthaeus G. Chajdas)
  - Initial draft.

## VK_AMD_shader_early_and_late_fragment_tests

### Name String

`VK_AMD_shader_early_and_late_fragment_tests`

### Extension Type

Device extension

### Registered Extension Number

322

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### SPIR-V Dependencies

- SPV_AMD_shader_early_and_late_fragment_tests

### Contact

- Tobias Hector [GitHub]tobski

### Extension Proposal

VK_AMD_shader_early_and_late_fragment_tests

### Other Extension Metadata

### Last Modified Date

2021-09-14

### Interactions and External Dependencies

- This extension interacts with `VK_EXT_shader_stencil_export`

### Contributors

- Tobias Hector, AMD

### Description

This extension adds support for the `SPV_AMD_shader_early_and_late_fragment_tests` extension, allowing shaders to explicitly opt in to allowing both early *and* late fragment tests with the `EarlyAndLateFragmentTestsAMD` execution mode.

If `VK_EXT_shader_stencil_export` is supported, additional execution modes allowing early depth tests similar to `DepthUnchanged`, `DepthLess`, and `DepthGreater` are provided.

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceShaderEarlyAndLateFragmentTestsFeaturesAMD

### New Enum Constants

- `VK_AMD_SHADER_EARLY_AND_LATE_FRAGMENT_TESTS_EXTENSION_NAME`
- `VK_AMD_SHADER_EARLY_AND_LATE_FRAGMENT_TESTS_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_EARLY_AND_LATE_FRAGMENT_TESTS_FEATURES_AMD`

### Version History

- Revision 1, 2021-09-14 (Tobias Hector)
  - Initial draft

## VK_AMD_shader_explicit_vertex_parameter

### Name String

`VK_AMD_shader_explicit_vertex_parameter`

### Extension Type

Device extension

### Registered Extension Number

22

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

None

### SPIR-V Dependencies

- SPV_AMD_shader_explicit_vertex_parameter

### Contact

- Qun Lin [GitHub]linqun

### Other Extension Metadata

### Last Modified Date

2016-05-10

### IP Status

No known IP claims.

### Interactions and External Dependencies

- This extension provides API support for `GL_AMD_shader_explicit_vertex_parameter`

### Contributors

- Matthaeus G. Chajdas, AMD
- Qun Lin, AMD
- Daniel Rakos, AMD
- Graham Sellers, AMD
- Rex Xu, AMD

### Description

This extension adds support for the following SPIR-V extension in Vulkan:

- `SPV_AMD_shader_explicit_vertex_parameter`

### New Enum Constants

- `VK_AMD_SHADER_EXPLICIT_VERTEX_PARAMETER_EXTENSION_NAME`
- `VK_AMD_SHADER_EXPLICIT_VERTEX_PARAMETER_SPEC_VERSION`

### Version History

- Revision 1, 2016-05-10 (Daniel Rakos)
  - Initial draft

## VK_AMD_shader_fragment_mask

### Name String

`VK_AMD_shader_fragment_mask`

### Extension Type

Device extension

### Registered Extension Number

138

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

None

### SPIR-V Dependencies

- SPV_AMD_shader_fragment_mask

### Contact

- Aaron Hagan [GitHub]AaronHaganAMD

### Other Extension Metadata

### Last Modified Date

2017-08-16

### IP Status

No known IP claims.

### Interactions and External Dependencies

- This extension provides API support for `GL_AMD_shader_fragment_mask`

### Contributors

- Aaron Hagan, AMD
- Daniel Rakos, AMD
- Timothy Lottes, AMD

### Description

This extension provides efficient read access to the fragment mask in compressed multisampled color surfaces. The fragment mask is a lookup table that associates color samples with color fragment values.

From a shader, the fragment mask can be fetched with a call to `fragmentMaskFetchAMD`, which returns a single `uint` where each subsequent four bits specify the color fragment index corresponding to the color sample, starting from the least significant bit. For example, when eight color samples are used, the color fragment index for color sample 0 will be in bits 0-3 of the fragment mask, for color sample 7 the index will be in bits 28-31.

The color fragment for a particular color sample may then be fetched with the corresponding fragment mask value using the `fragmentFetchAMD` shader function.

### New Enum Constants

- `VK_AMD_SHADER_FRAGMENT_MASK_EXTENSION_NAME`
- `VK_AMD_SHADER_FRAGMENT_MASK_SPEC_VERSION`

### New SPIR-V Capabilities

- `FragmentMaskAMD`

### Examples

This example shows a shader that queries the fragment mask from a multisampled compressed surface and uses it to query fragment values.

```
#version 450 core

#extension GL_AMD_shader_fragment_mask: enable

layout(binding = 0) uniform sampler2DMS       s2DMS;
layout(binding = 1) uniform isampler2DMSArray is2DMSArray;

layout(binding = 2, input_attachment_index = 0) uniform usubpassInputMS usubpassMS;

layout(location = 0) out vec4 fragColor;

void main()
{
    vec4 fragOne = vec4(0.0);

    uint fragMask = fragmentMaskFetchAMD(s2DMS, ivec2(2, 3));
    uint fragIndex = (fragMask & 0xF0) >> 4;
    fragOne += fragmentFetchAMD(s2DMS, ivec2(2, 3), 1);

    fragMask = fragmentMaskFetchAMD(is2DMSArray, ivec3(2, 3, 1));
    fragIndex = (fragMask & 0xF0) >> 4;
    fragOne += fragmentFetchAMD(is2DMSArray, ivec3(2, 3, 1), fragIndex);

    fragMask = fragmentMaskFetchAMD(usubpassMS);
    fragIndex = (fragMask & 0xF0) >> 4;
    fragOne += fragmentFetchAMD(usubpassMS, fragIndex);

    fragColor = fragOne;
}
```

### Version History

- Revision 1, 2017-08-16 (Aaron Hagan)
  - Initial draft

## VK_AMD_shader_image_load_store_lod

### Name String

`VK_AMD_shader_image_load_store_lod`

### Extension Type

Device extension

### Registered Extension Number

47

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

None

### SPIR-V Dependencies

- SPV_AMD_shader_image_load_store_lod

### Contact

- Dominik Witczak [GitHub]dominikwitczakamd

### Other Extension Metadata

### Last Modified Date

2017-08-21

### Interactions and External Dependencies

- This extension provides API support for `GL_AMD_shader_image_load_store_lod`

### IP Status

No known IP claims.

### Contributors

- Dominik Witczak, AMD
- Qun Lin, AMD
- Rex Xu, AMD

### Description

This extension adds support for the following SPIR-V extension in Vulkan:

- `SPV_AMD_shader_image_load_store_lod`

### New Enum Constants

- `VK_AMD_SHADER_IMAGE_LOAD_STORE_LOD_EXTENSION_NAME`
- `VK_AMD_SHADER_IMAGE_LOAD_STORE_LOD_SPEC_VERSION`

### Version History

- Revision 1, 2017-08-21 (Dominik Witczak)

- Initial draft

## VK_AMD_shader_info

### Name String

`VK_AMD_shader_info`

### Extension Type

Device extension

### Registered Extension Number

43

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

None

### Special Use

- Developer tools

### Contact

- Jaakko Konttinen [GitHub]jaakkoamd

### Other Extension Metadata

### Last Modified Date

2017-10-09

### IP Status

No known IP claims.

### Contributors

- Jaakko Konttinen, AMD

### Description

This extension adds a way to query certain information about a compiled shader which is part of a pipeline. This information may include shader disassembly, shader binary and various statistics about a shader's resource usage.

While this extension provides a mechanism for extracting this information, the details regarding the contents or format of this information are not specified by this extension and may be provided by the vendor externally.

Furthermore, all information types are optionally supported, and users should not assume every implementation supports querying every type of information.

### New Commands

- vkGetShaderInfoAMD

### New Structures

- VkShaderResourceUsageAMD
- VkShaderStatisticsInfoAMD

### New Enums

- VkShaderInfoTypeAMD

### New Enum Constants

- VK_AMD_SHADER_INFO_EXTENSION_NAME
- VK_AMD_SHADER_INFO_SPEC_VERSION

### Examples

This example extracts the register usage of a fragment shader within a particular graphics pipeline:

```
extern VkDevice device;
extern VkPipeline gfxPipeline;

PFN_vkGetShaderInfoAMD pfnGetShaderInfoAMD = (PFN_vkGetShaderInfoAMD
)vkGetDeviceProcAddr(
    device, "vkGetShaderInfoAMD");

VkShaderStatisticsInfoAMD statistics = {};

size_t dataSize = sizeof(statistics);

if (pfnGetShaderInfoAMD(device,
    gfxPipeline,
    VK_SHADER_STAGE_FRAGMENT_BIT,
    VK_SHADER_INFO_TYPE_STATISTICS_AMD,
    &dataSize,
    &statistics) == VK_SUCCESS)
{
    printf("VGPR usage: %d\n", statistics.resourceUsage.numUsedVgprs);
    printf("SGPR usage: %d\n", statistics.resourceUsage.numUsedSgprs);
}
```

The following example continues the previous example by subsequently attempting to query and print shader disassembly about the fragment shader:

```
// Query disassembly size (if available)
if (pfnGetShaderInfoAMD(device,
    gfxPipeline,
    VK_SHADER_STAGE_FRAGMENT_BIT,
    VK_SHADER_INFO_TYPE_DISASSEMBLY_AMD,
    &dataSize,
    nullptr) == VK_SUCCESS)
{
    printf("Fragment shader disassembly:\n");

    void* disassembly = malloc(dataSize);

    // Query disassembly and print
    if (pfnGetShaderInfoAMD(device,
        gfxPipeline,
        VK_SHADER_STAGE_FRAGMENT_BIT,
        VK_SHADER_INFO_TYPE_DISASSEMBLY_AMD,
        &dataSize,
        disassembly) == VK_SUCCESS)
    {
        printf((char*)disassembly);
    }

    free(disassembly);
}
```

### Version History

- Revision 1, 2017-10-09 (Jaakko Konttinen)
  - Initial revision

## VK_AMD_shader_trinary_minmax

### Name String

`VK_AMD_shader_trinary_minmax`

### Extension Type

Device extension

### Registered Extension Number

21

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

None

### SPIR-V Dependencies

- SPV_AMD_shader_trinary_minmax

### Contact

- Qun Lin [GitHub]linqun

### Other Extension Metadata

### Last Modified Date

2016-05-10

### IP Status

No known IP claims.

### Interactions and External Dependencies

- This extension provides API support for `GL_AMD_shader_trinary_minmax`

### Contributors

- Matthaeus G. Chajdas, AMD
- Qun Lin, AMD
- Daniel Rakos, AMD
- Graham Sellers, AMD
- Rex Xu, AMD

### Description

This extension adds support for the following SPIR-V extension in Vulkan:

- `SPV_AMD_shader_trinary_minmax`

### New Enum Constants

- `VK_AMD_SHADER_TRINARY_MINMAX_EXTENSION_NAME`
- `VK_AMD_SHADER_TRINARY_MINMAX_SPEC_VERSION`

### Version History

- Revision 1, 2016-05-10 (Daniel Rakos)
  - Initial draft

## VK_AMD_texture_gather_bias_lod

### Name String

`VK_AMD_texture_gather_bias_lod`

### Extension Type

Device extension

### Registered Extension Number

42

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### SPIR-V Dependencies

- SPV_AMD_texture_gather_bias_lod

### Contact

- Rex Xu [GitHub]amdrexu

### Other Extension Metadata

### Last Modified Date

2017-03-21

### IP Status

No known IP claims.

### Interactions and External Dependencies

- This extension provides API support for `GL_AMD_texture_gather_bias_lod`

### Contributors

- Dominik Witczak, AMD
- Daniel Rakos, AMD
- Graham Sellers, AMD
- Matthaeus G. Chajdas, AMD
- Qun Lin, AMD
- Rex Xu, AMD
- Timothy Lottes, AMD

### Description

This extension adds two related features.

Firstly, support for the following SPIR-V extension in Vulkan is added:

- `SPV_AMD_texture_gather_bias_lod`

Secondly, the extension allows the application to query which formats can be used together with the new function prototypes introduced by the SPIR-V extension.

### New Structures

- Extending VkImageFormatProperties2:
  - VkTextureLODGatherFormatPropertiesAMD

### New Enum Constants

- `VK_AMD_TEXTURE_GATHER_BIAS_LOD_EXTENSION_NAME`
- `VK_AMD_TEXTURE_GATHER_BIAS_LOD_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_TEXTURE_LOD_GATHER_FORMAT_PROPERTIES_AMD`

### New SPIR-V Capabilities

- `ImageGatherBiasLodAMD`

### Examples

```
struct VkTextureLODGatherFormatPropertiesAMD
{
    VkStructureType sType;
    const void*     pNext;
    VkBool32        supportsTextureGatherLODBiasAMD;
};

//
------------------------------------------------------------------------------------
--
// How to detect if an image format can be used with the new function prototypes.
VkPhysicalDeviceImageFormatInfo2   formatInfo;
VkImageFormatProperties2           formatProps;
VkTextureLODGatherFormatPropertiesAMD textureLODGatherSupport;

textureLODGatherSupport.sType =
VK_STRUCTURE_TYPE_TEXTURE_LOD_GATHER_FORMAT_PROPERTIES_AMD;
textureLODGatherSupport.pNext = nullptr;

formatInfo.sType  = VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_IMAGE_FORMAT_INFO_2;
formatInfo.pNext  = nullptr;
formatInfo.format = ...;
formatInfo.type   = ...;
formatInfo.tiling = ...;
```

```
formatInfo.usage  = ...;
formatInfo.flags  = ...;

formatProps.sType = VK_STRUCTURE_TYPE_IMAGE_FORMAT_PROPERTIES_2;
formatProps.pNext = &textureLODGatherSupport;

vkGetPhysicalDeviceImageFormatProperties2(physical_device, &formatInfo, &formatProps);

if (textureLODGatherSupport.supportsTextureGatherLODBiasAMD == VK_TRUE)
{
    // physical device supports SPV_AMD_texture_gather_bias_lod for the specified
    // format configuration.
}
else
{
    // physical device does not support SPV_AMD_texture_gather_bias_lod for the
    // specified format configuration.
}
```

### Version History

- Revision 1, 2017-03-21 (Dominik Witczak)
  - Initial draft

## VK_ANDROID_external_format_resolve

### Name String

`VK_ANDROID_external_format_resolve`

### Extension Type

Device extension

### Registered Extension Number

469

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_ANDROID_external_memory_android_hardware_buffer

### API Interactions

- Interacts with VK_VERSION_1_3
- Interacts with VK_KHR_dynamic_rendering

**Special Use**

- OpenGL / ES support

**Contact**

- Chris Forbes [GitHub]chrisforbes

**Extension Proposal**

VK_ANDROID_external_format_resolve

**Other Extension Metadata**

**Last Modified Date**

2023-05-03

**IP Status**

No known IP claims.

**Contributors**

- Tobias Hector, AMD
- Chris Forbes, Google
- Jan-Harald Fredriksen, Arm
- Shahbaz Youssefi, Google
- Matthew Netsch, Qualcomm
- Tony Zlatsinki, Nvidia
- Daniel Koch, Nvidia
- Jeff Leger, Qualcomm
- Alex Walters, Imagination
- Andrew Garrard, Imagination
- Ralph Potter, Samsung
- Ian Elliott, Google

**Description**

This extension enables rendering to Android Hardware Buffers with external formats which cannot be directly represented as renderable in Vulkan, including $Y'C_BC_R$ formats.

**New Structures**

- Extending VkAndroidHardwareBufferPropertiesANDROID:
  - VkAndroidHardwareBufferFormatResolvePropertiesANDROID
- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceExternalFormatResolveFeaturesANDROID
- Extending VkPhysicalDeviceProperties2:

- VkPhysicalDeviceExternalFormatResolvePropertiesANDROID

#### New Enum Constants

- `VK_ANDROID_EXTERNAL_FORMAT_RESOLVE_EXTENSION_NAME`
- `VK_ANDROID_EXTERNAL_FORMAT_RESOLVE_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_ANDROID_HARDWARE_BUFFER_FORMAT_RESOLVE_PROPERTIES_ANDROID`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTERNAL_FORMAT_RESOLVE_FEATURES_ANDROID`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTERNAL_FORMAT_RESOLVE_PROPERTIES_ANDROID`

If VK_KHR_dynamic_rendering or Vulkan Version 1.3 is supported:

- Extending VkResolveModeFlagBits:
  - `VK_RESOLVE_MODE_EXTERNAL_FORMAT_DOWNSAMPLE_ANDROID`

#### Version History

- Revision 1, 2023-05-34 (Tobias Hector)
  - Initial version

## VK_ANDROID_external_memory_android_hardware_buffer

#### Name String

`VK_ANDROID_external_memory_android_hardware_buffer`

#### Extension Type

Device extension

#### Registered Extension Number

130

#### Revision

5

#### Ratification Status

Not ratified

#### Extension and Version Dependencies

VK_KHR_sampler_ycbcr_conversion
and
VK_KHR_external_memory
and
VK_KHR_dedicated_allocation
or
Vulkan Version 1.1

and
VK_EXT_queue_family_foreign

### API Interactions

- Interacts with VK_VERSION_1_3
- Interacts with VK_KHR_format_feature_flags2

### Contact

- Jesse Hall [GitHub]critsec

### Other Extension Metadata

### Last Modified Date

2021-09-30

### IP Status

No known IP claims.

### Contributors

- Ray Smith, ARM
- Lina Versace, Google
- Jesse Hall, Google
- Tobias Hector, Imagination
- James Jones, NVIDIA
- Tony Zlatinski, NVIDIA
- Matthew Netsch, Qualcomm
- Andrew Garrard, Samsung

### Description

This extension enables an application to import Android AHardwareBuffer objects created outside of the Vulkan device into Vulkan memory objects, where they **can** be bound to images and buffers. It also allows exporting an AHardwareBuffer from a Vulkan memory object for symmetry with other operating systems. But since not all AHardwareBuffer usages and formats have Vulkan equivalents, exporting from Vulkan provides strictly less functionality than creating the AHardwareBuffer externally and importing it.

Some AHardwareBuffer images have implementation-defined *external formats* that **may** not correspond to Vulkan formats. Sampler $Y'C_BC_R$ conversion **can** be used to sample from these images and convert them to a known color space.

### New Base Types

- AHardwareBuffer

### New Commands

- vkGetAndroidHardwareBufferPropertiesANDROID
- vkGetMemoryAndroidHardwareBufferANDROID

### New Structures

- VkAndroidHardwareBufferPropertiesANDROID
- VkMemoryGetAndroidHardwareBufferInfoANDROID
- Extending VkAndroidHardwareBufferPropertiesANDROID:
  - VkAndroidHardwareBufferFormatPropertiesANDROID
- Extending VkImageCreateInfo, VkSamplerYcbcrConversionCreateInfo, VkAttachmentDescription2, VkGraphicsPipelineCreateInfo, VkCommandBufferInheritanceInfo:
  - VkExternalFormatANDROID
- Extending VkImageFormatProperties2:
  - VkAndroidHardwareBufferUsageANDROID
- Extending VkMemoryAllocateInfo:
  - VkImportAndroidHardwareBufferInfoANDROID

If VK_KHR_format_feature_flags2 or Vulkan Version 1.3 is supported:

- Extending VkAndroidHardwareBufferPropertiesANDROID:
  - VkAndroidHardwareBufferFormatProperties2ANDROID

### New Enum Constants

- `VK_ANDROID_EXTERNAL_MEMORY_ANDROID_HARDWARE_BUFFER_EXTENSION_NAME`
- `VK_ANDROID_EXTERNAL_MEMORY_ANDROID_HARDWARE_BUFFER_SPEC_VERSION`
- Extending VkExternalMemoryHandleTypeFlagBits:
  - `VK_EXTERNAL_MEMORY_HANDLE_TYPE_ANDROID_HARDWARE_BUFFER_BIT_ANDROID`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_ANDROID_HARDWARE_BUFFER_FORMAT_PROPERTIES_ANDROID`
  - `VK_STRUCTURE_TYPE_ANDROID_HARDWARE_BUFFER_PROPERTIES_ANDROID`
  - `VK_STRUCTURE_TYPE_ANDROID_HARDWARE_BUFFER_USAGE_ANDROID`
  - `VK_STRUCTURE_TYPE_EXTERNAL_FORMAT_ANDROID`
  - `VK_STRUCTURE_TYPE_IMPORT_ANDROID_HARDWARE_BUFFER_INFO_ANDROID`
  - `VK_STRUCTURE_TYPE_MEMORY_GET_ANDROID_HARDWARE_BUFFER_INFO_ANDROID`

If VK_KHR_format_feature_flags2 or Vulkan Version 1.3 is supported:

- Extending VkStructureType:

- `VK_STRUCTURE_TYPE_ANDROID_HARDWARE_BUFFER_FORMAT_PROPERTIES_2_ANDROID`

**Issues**

1) Other external memory objects are represented as weakly-typed handles (e.g. Win32 `HANDLE` or POSIX file descriptor), and require a handle type parameter along with handles. AHardwareBuffer is strongly typed, so naming the handle type is redundant. Does symmetry justify adding handle type parameters/fields anyway?

**RESOLVED**: No. The handle type is already provided in places that treat external memory objects generically. In the places we would add it, the application code that would have to provide the handle type value is already dealing with AHardwareBuffer-specific commands/structures; the extra symmetry would not be enough to make that code generic.

2) The internal layout and therefore size of a AHardwareBuffer image may depend on native usage flags that do not have corresponding Vulkan counterparts. Do we provide this information to vkCreateImage somehow, or allow the allocation size reported by vkGetImageMemoryRequirements to be approximate?

**RESOLVED**: Allow the allocation size to be unspecified when allocating the memory. It has to work this way for exported image memory anyway, since AHardwareBuffer allocation happens in vkAllocateMemory, and internally is performed by a separate HAL, not the Vulkan implementation itself. There is a similar issue with vkGetImageSubresourceLayout: the layout is determined by the allocator HAL, so it is not known until the image is bound to memory.

3) Should the result of sampling an external-format image with the suggested $Y'C_BC_R$ conversion parameters yield the same results as using a `samplerExternalOES` in OpenGL ES?

**RESOLVED**: This would be desirable, so that apps converting from OpenGL ES to Vulkan could get the same output given the same input. But since sampling and conversion from $Y'C_BC_R$ images is so loosely defined in OpenGL ES, multiple implementations do it in a way that does not conform to Vulkan's requirements. Modifying the OpenGL ES implementation would be difficult, and would change the output of existing unmodified applications. Changing the output only for applications that are being modified gives developers the chance to notice and mitigate any problems. Implementations are encouraged to minimize differences as much as possible without causing compatibility problems for existing OpenGL ES applications or violating Vulkan requirements.

4) Should an AHardwareBuffer with `AHARDWAREBUFFER_USAGE_CPU_*` usage be mappable in Vulkan? Should it be possible to export an `AHardwareBuffers` with such usage?

**RESOLVED**: Optional, and mapping in Vulkan is not the same as `AHardwareBuffer_lock`. The semantics of these are different: mapping in memory is persistent, just gives a raw view of the memory contents, and does not involve ownership. `AHardwareBuffer_lock` gives the host exclusive access to the buffer, is temporary, and allows for reformatting copy-in/copy-out. Implementations are not required to support host-visible memory types for imported Android hardware buffers or resources backed by them. If a host-visible memory type is supported and used, the memory can be mapped in Vulkan, but doing so follows Vulkan semantics: it is just a raw view of the data and does not imply ownership (this means implementations must not internally call `AHardwareBuffer_lock` to implement vkMapMemory, or assume the application has done so). Implementations are not required to support linear-tiled images backed by Android hardware buffers, even if the

AHardwareBuffer has CPU usage. There is no reliable way to allocate memory in Vulkan that can be exported to a AHardwareBuffer with CPU usage.

5) Android may add new AHardwareBuffer formats and usage flags over time. Can reference to them be added to this extension, or do they need a new extension?

**RESOLVED**: This extension can document the interaction between the new AHB formats/usages and existing Vulkan features. No new Vulkan features or implementation requirements can be added. The extension version number will be incremented when this additional documentation is added, but the version number does not indicate that an implementation supports Vulkan memory or resources that map to the new AHardwareBuffer features: support for that must be queried with vkGetPhysicalDeviceImageFormatProperties2 or is implied by successfully allocating a AHardwareBuffer outside of Vulkan that uses the new feature and has a GPU usage flag.

In essence, these are new features added to a new Android API level, rather than new Vulkan features. The extension will only document how existing Vulkan features map to that new Android feature.

### Version History

- Revision 5, 2022-02-04 (Chris Forbes)
  - Describe mapping of flags for storage image support
- Revision 4, 2021-09-30 (Jon Leech)
  - Add interaction with `VK_KHR_format_feature_flags2` to `vk.xml`
- Revision 3, 2019-08-27 (Jon Leech)
  - Update revision history to correspond to XML version number
- Revision 2, 2018-04-09 (Petr Kraus)
  - Markup fixes and remove incorrect Draft status
- Revision 1, 2018-03-04 (Jesse Hall)
  - Initial version

## VK_ARM_render_pass_striped

### Name String

`VK_ARM_render_pass_striped`

### Extension Type

Device extension

### Registered Extension Number

425

### Revision

1

**Ratification Status**

Not ratified

**Extension and Version Dependencies**

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1
and
VK_KHR_synchronization2
or
Vulkan Version 1.3

**Contact**

- Jan-Harald Fredriksen [GitHub]janharaldfredriksen-arm

**Extension Proposal**

VK_ARM_render_pass_striped

**Other Extension Metadata**

**Last Modified Date**

2023-11-21

**IP Status**

No known IP claims.

**Contributors**

- Jan-Harald Fredriksen, Arm
- Lisa Wu, Arm
- Torbjorn Nilsson, Arm
- Ying-Chieh Chen, Mediatek
- Jim Chiu, Mediatek

**Description**

This extension adds the ability to split a render pass instance into stripes, and to get a notification when rendering has completed for each stripe.

**New Structures**

- VkRenderPassStripeInfoARM
- Extending VkCommandBufferSubmitInfo:
  - VkRenderPassStripeSubmitInfoARM
- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceRenderPassStripedFeaturesARM

- Extending VkPhysicalDeviceProperties2:
  - VkPhysicalDeviceRenderPassStripedPropertiesARM
- Extending VkRenderingInfo, VkRenderPassBeginInfo:
  - VkRenderPassStripeBeginInfoARM

### New Enum Constants

- `VK_ARM_RENDER_PASS_STRIPED_EXTENSION_NAME`
- `VK_ARM_RENDER_PASS_STRIPED_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_RENDER_PASS_STRIPED_FEATURES_ARM`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_RENDER_PASS_STRIPED_PROPERTIES_ARM`
  - `VK_STRUCTURE_TYPE_RENDER_PASS_STRIPE_BEGIN_INFO_ARM`
  - `VK_STRUCTURE_TYPE_RENDER_PASS_STRIPE_INFO_ARM`
  - `VK_STRUCTURE_TYPE_RENDER_PASS_STRIPE_SUBMIT_INFO_ARM`

### Examples

None.

### Version History

- Revision 1, 2023-11-21
  - Initial revision

## VK_ARM_scheduling_controls

### Name String

`VK_ARM_scheduling_controls`

### Extension Type

Device extension

### Registered Extension Number

418

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_ARM_shader_core_builtins

### Contact

- Kevin Petit [GitHub]kpet

### Other Extension Metadata

### Last Modified Date

2023-08-23

### Interactions and External Dependencies

None

### IP Status

No known IP claims.

### Contributors

- Kévin Petit, Arm Ltd.
- Jan-Harald Fredriksen, Arm Ltd.
- Mikel Garai, Arm Ltd.

### Description

This extension exposes a collection of controls to modify the scheduling behavior of Arm Mali devices.

### New Structures

- Extending VkDeviceQueueCreateInfo, VkDeviceCreateInfo:
  - VkDeviceQueueShaderCoreControlCreateInfoARM
- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceSchedulingControlsFeaturesARM
- Extending VkPhysicalDeviceProperties2:
  - VkPhysicalDeviceSchedulingControlsPropertiesARM

### New Enums

- VkPhysicalDeviceSchedulingControlsFlagBitsARM

### New Bitmasks

- VkPhysicalDeviceSchedulingControlsFlagsARM

### New Enum Constants

- `VK_ARM_SCHEDULING_CONTROLS_EXTENSION_NAME`
- `VK_ARM_SCHEDULING_CONTROLS_SPEC_VERSION`
- Extending VkStructureType:

- `VK_STRUCTURE_TYPE_DEVICE_QUEUE_SHADER_CORE_CONTROL_CREATE_INFO_ARM`
- `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SCHEDULING_CONTROLS_FEATURES_ARM`
- `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SCHEDULING_CONTROLS_PROPERTIES_ARM`

### New SPIR-V Capabilities

None.

### Issues

None.

### Version History

- Revision 1, 2023-08-23 (Kévin Petit)
  - Initial revision

## VK_ARM_shader_core_builtins

### Name String

`VK_ARM_shader_core_builtins`

### Extension Type

Device extension

### Registered Extension Number

498

### Revision

2

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### SPIR-V Dependencies

- SPV_ARM_core_builtins

### Contact

- Kevin Petit [GitHub]kpet

### Other Extension Metadata

**Last Modified Date**

2022-10-05

**Interactions and External Dependencies**

- This extension provides API support for `GL_ARM_shader_core_builtins`

**Contributors**

- Kevin Petit, Arm Ltd.
- Jan-Harald Fredriksen, Arm Ltd.

**Description**

This extension provides the ability to determine device-specific properties on Arm GPUs. It exposes properties for the number of shader cores, the maximum number of warps that can run on a shader core, and shader builtins to enable invocations to identify which core and warp a shader invocation is executing on.

This extension enables support for the SPIR-V `CoreBuiltinsARM` capability.

These properties and built-ins can be used for debugging or performance optimization purposes. A typical optimization example would be to use `CoreIDARM` to select a per-shader-core instance of a data structure in algorithms that use atomics so as to reduce contention.

**New Structures**

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceShaderCoreBuiltinsFeaturesARM
- Extending VkPhysicalDeviceProperties2:
  - VkPhysicalDeviceShaderCoreBuiltinsPropertiesARM

**New Enum Constants**

- `VK_ARM_SHADER_CORE_BUILTINS_EXTENSION_NAME`
- `VK_ARM_SHADER_CORE_BUILTINS_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_CORE_BUILTINS_FEATURES_ARM`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_CORE_BUILTINS_PROPERTIES_ARM`

**New or Modified Built-In Variables**

- `CoreCountARM`
- `CoreMaxIDARM`
- `CoreIDARM`
- `WarpsMaxIDARM`
- `WarpIDARM`

### New SPIR-V Capabilities

- `CoreBuiltinsARM`

### Issues

None.

### Version History

- Revision 1, 2022-10-05 (Kevin Petit)
  - Initial revision
- Revision 2, 2022-10-26 (Kevin Petit)
  - Add `shaderCoreMask` property

## VK_ARM_shader_core_properties

### Name String

`VK_ARM_shader_core_properties`

### Extension Type

Device extension

### Registered Extension Number

416

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

Vulkan Version 1.1

### Contact

- Jan-Harald Fredriksen [GitHub]janharaldfredriksen-arm

### Other Extension Metadata

### Last Modified Date

2023-02-07

### IP Status

No known IP claims.

### Contributors

- Jan-Harald Fredriksen, Arm Ltd.

### Description

This extension provides the ability to determine device-specific performance properties of Arm GPUs.

It exposes properties for the number of texel, pixel, and fused multiply-add operations per clock per shader core. This can be used in combination with the `VK_ARM_shader_core_builtins` extension that provides the ability to query the number of shader cores on the physical device.

### New Structures

- Extending VkPhysicalDeviceProperties2:
  - VkPhysicalDeviceShaderCorePropertiesARM

### New Enum Constants

- `VK_ARM_SHADER_CORE_PROPERTIES_EXTENSION_NAME`
- `VK_ARM_SHADER_CORE_PROPERTIES_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_CORE_PROPERTIES_ARM`

### Version History

- Revision 1, 2023-02-07 (Jan-Harald Fredriksen)
  - Initial draft.

## VK_FUCHSIA_buffer_collection

### Name String

`VK_FUCHSIA_buffer_collection`

### Extension Type

Device extension

### Registered Extension Number

367

### Revision

2

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_FUCHSIA_external_memory
and
VK_KHR_sampler_ycbcr_conversion
or

Vulkan Version 1.1

### API Interactions

- Interacts with VK_EXT_debug_report

### Contact

- John Rosasco [GitHub]rosasco

### Other Extension Metadata

### Last Modified Date

2021-09-23

### IP Status

No known IP claims.

### Contributors

- Craig Stout, Google
- John Bauman, Google
- John Rosasco, Google

### Description

A buffer collection is a collection of one or more buffers which were allocated together as a group and which all have the same properties. These properties describe the buffers' internal representation such as its dimensions and memory layout. This ensures that all of the buffers can be used interchangeably by tasks that require swapping among multiple buffers, such as double-buffered graphics rendering.

By sharing such a collection of buffers between components, communication about buffer lifecycle can be made much simpler and more efficient. For example, when a content producer finishes writing to a buffer, it can message the consumer of the buffer with the buffer index, rather than passing a handle to the shared memory.

On Fuchsia, the Sysmem service uses buffer collections as a core construct in its design. VK_FUCHSIA_buffer_collection is the Vulkan extension that allows Vulkan applications to interoperate with the Sysmem service on Fuchsia.

### New Object Types

- VkBufferCollectionFUCHSIA

### New Commands

- vkCreateBufferCollectionFUCHSIA
- vkDestroyBufferCollectionFUCHSIA
- vkGetBufferCollectionPropertiesFUCHSIA
- vkSetBufferCollectionBufferConstraintsFUCHSIA

- vkSetBufferCollectionImageConstraintsFUCHSIA

### New Structures

- VkBufferCollectionConstraintsInfoFUCHSIA
- VkBufferCollectionCreateInfoFUCHSIA
- VkBufferCollectionPropertiesFUCHSIA
- VkBufferConstraintsInfoFUCHSIA
- VkImageConstraintsInfoFUCHSIA
- VkImageFormatConstraintsInfoFUCHSIA
- VkSysmemColorSpaceFUCHSIA
- Extending VkBufferCreateInfo:
  - VkBufferCollectionBufferCreateInfoFUCHSIA
- Extending VkImageCreateInfo:
  - VkBufferCollectionImageCreateInfoFUCHSIA
- Extending VkMemoryAllocateInfo:
  - VkImportMemoryBufferCollectionFUCHSIA

### New Enums

- VkImageConstraintsInfoFlagBitsFUCHSIA

### New Bitmasks

- VkImageConstraintsInfoFlagsFUCHSIA
- VkImageFormatConstraintsFlagsFUCHSIA

### New Enum Constants

- `VK_FUCHSIA_BUFFER_COLLECTION_EXTENSION_NAME`
- `VK_FUCHSIA_BUFFER_COLLECTION_SPEC_VERSION`
- Extending VkObjectType:
  - `VK_OBJECT_TYPE_BUFFER_COLLECTION_FUCHSIA`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_BUFFER_COLLECTION_BUFFER_CREATE_INFO_FUCHSIA`
  - `VK_STRUCTURE_TYPE_BUFFER_COLLECTION_CONSTRAINTS_INFO_FUCHSIA`
  - `VK_STRUCTURE_TYPE_BUFFER_COLLECTION_CREATE_INFO_FUCHSIA`
  - `VK_STRUCTURE_TYPE_BUFFER_COLLECTION_IMAGE_CREATE_INFO_FUCHSIA`
  - `VK_STRUCTURE_TYPE_BUFFER_COLLECTION_PROPERTIES_FUCHSIA`
  - `VK_STRUCTURE_TYPE_BUFFER_CONSTRAINTS_INFO_FUCHSIA`

- `VK_STRUCTURE_TYPE_IMAGE_CONSTRAINTS_INFO_FUCHSIA`
- `VK_STRUCTURE_TYPE_IMAGE_FORMAT_CONSTRAINTS_INFO_FUCHSIA`
- `VK_STRUCTURE_TYPE_IMPORT_MEMORY_BUFFER_COLLECTION_FUCHSIA`
- `VK_STRUCTURE_TYPE_SYSMEM_COLOR_SPACE_FUCHSIA`

If VK_EXT_debug_report is supported:

- Extending VkDebugReportObjectTypeEXT:
  - `VK_DEBUG_REPORT_OBJECT_TYPE_BUFFER_COLLECTION_FUCHSIA_EXT`

#### Issues

1) When configuring a VkImageConstraintsInfoFUCHSIA structure for constraint setting, should a NULL `pFormatConstraints` parameter be allowed ?

**RESOLVED**: No. Specifying a NULL `pFormatConstraints` results in logical complexity of interpreting the relationship between the VkImageCreateInfo::`usage` settings of the elements of the `pImageCreateInfos` array and the implied or desired VkFormatFeatureFlags.

The explicit requirement for `pFormatConstraints` to be non-NULL simplifies the implied logic of the implementation and expectations for the Vulkan application.

#### Version History

- Revision 2, 2021-09-23 (John Rosasco)
  - Review passes
- Revision 1, 2021-03-09 (John Rosasco)
  - Initial revision

## VK_FUCHSIA_external_memory

#### Name String

`VK_FUCHSIA_external_memory`

#### Extension Type

Device extension

#### Registered Extension Number

365

#### Revision

1

#### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_external_memory_capabilities
and
VK_KHR_external_memory
or
Vulkan Version 1.1

### Contact

- John Rosasco [GitHub]rosasco

### Other Extension Metadata

### Last Modified Date

2021-03-01

### IP Status

No known IP claims.

### Contributors

- Craig Stout, Google
- John Bauman, Google
- John Rosasco, Google

### Description

Vulkan apps may wish to export or import device memory handles to or from other logical devices, instances or APIs.

This memory sharing can eliminate copies of memory buffers when different subsystems need to interoperate on them. Sharing memory buffers may also facilitate a better distribution of processing workload for more complex memory manipulation pipelines.

### New Commands

- vkGetMemoryZirconHandleFUCHSIA
- vkGetMemoryZirconHandlePropertiesFUCHSIA

### New Structures

- VkMemoryGetZirconHandleInfoFUCHSIA
- VkMemoryZirconHandlePropertiesFUCHSIA
- Extending VkMemoryAllocateInfo:
  - VkImportMemoryZirconHandleInfoFUCHSIA

### New Enum Constants

- `VK_FUCHSIA_EXTERNAL_MEMORY_EXTENSION_NAME`

- `VK_FUCHSIA_EXTERNAL_MEMORY_SPEC_VERSION`
- Extending VkExternalMemoryHandleTypeFlagBits:
  - `VK_EXTERNAL_MEMORY_HANDLE_TYPE_ZIRCON_VMO_BIT_FUCHSIA`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_IMPORT_MEMORY_ZIRCON_HANDLE_INFO_FUCHSIA`
  - `VK_STRUCTURE_TYPE_MEMORY_GET_ZIRCON_HANDLE_INFO_FUCHSIA`
  - `VK_STRUCTURE_TYPE_MEMORY_ZIRCON_HANDLE_PROPERTIES_FUCHSIA`

#### Issues

See `VK_KHR_external_memory` issues list for further information.

#### Version History

- Revision 1, 2021-03-01 (John Rosasco)
  - Initial draft

## VK_FUCHSIA_external_semaphore

#### Name String

`VK_FUCHSIA_external_semaphore`

#### Extension Type

Device extension

#### Registered Extension Number

366

#### Revision

1

#### Ratification Status

Not ratified

#### Extension and Version Dependencies

VK_KHR_external_semaphore_capabilities
and
VK_KHR_external_semaphore

#### Contact

- John Rosasco [GitHub]rosasco

#### Other Extension Metadata

#### Last Modified Date

2021-03-08

### IP Status

No known IP claims.

### Contributors

- Craig Stout, Google
- John Bauman, Google
- John Rosasco, Google

### Description

An application using external memory may wish to synchronize access to that memory using semaphores. This extension enables an application to export semaphore payload to and import semaphore payload from Zircon event handles.

### New Commands

- vkGetSemaphoreZirconHandleFUCHSIA
- vkImportSemaphoreZirconHandleFUCHSIA

### New Structures

- VkImportSemaphoreZirconHandleInfoFUCHSIA
- VkSemaphoreGetZirconHandleInfoFUCHSIA

### New Enum Constants

- `VK_FUCHSIA_EXTERNAL_SEMAPHORE_EXTENSION_NAME`
- `VK_FUCHSIA_EXTERNAL_SEMAPHORE_SPEC_VERSION`
- Extending VkExternalSemaphoreHandleTypeFlagBits:
  - `VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_ZIRCON_EVENT_BIT_FUCHSIA`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_IMPORT_SEMAPHORE_ZIRCON_HANDLE_INFO_FUCHSIA`
  - `VK_STRUCTURE_TYPE_SEMAPHORE_GET_ZIRCON_HANDLE_INFO_FUCHSIA`

### Issues

1) Does the application need to close the Zircon event handle returned by vkGetSemaphoreZirconHandleFUCHSIA?

**RESOLVED**: Yes, unless it is passed back in to a driver instance to import the semaphore. A successful get call transfers ownership of the Zircon event handle to the application, and a successful import transfers it back to the driver. Destroying the original semaphore object will not close the Zircon event handle nor remove its reference to the underlying semaphore resource associated with it.

### Version History

- Revision 1, 2021-03-08 (John Rosasco)
  - Initial revision

## VK_FUCHSIA_imagepipe_surface

### Name String

`VK_FUCHSIA_imagepipe_surface`

### Extension Type

Instance extension

### Registered Extension Number

215

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_surface

### Contact

- Craig Stout [GitHub]cdotstout

### Other Extension Metadata

### Last Modified Date

2018-07-27

### IP Status

No known IP claims.

### Contributors

- Craig Stout, Google
- Ian Elliott, Google
- Jesse Hall, Google

### Description

The `VK_FUCHSIA_imagepipe_surface` extension is an instance extension. It provides a mechanism to create a VkSurfaceKHR object (defined by the `VK_KHR_surface` extension) that refers to a Fuchsia `imagePipeHandle`.

### New Commands

- vkCreateImagePipeSurfaceFUCHSIA

### New Structures

- VkImagePipeSurfaceCreateInfoFUCHSIA

### New Bitmasks

- VkImagePipeSurfaceCreateFlagsFUCHSIA

### New Enum Constants

- `VK_FUCHSIA_IMAGEPIPE_SURFACE_EXTENSION_NAME`
- `VK_FUCHSIA_IMAGEPIPE_SURFACE_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_IMAGEPIPE_SURFACE_CREATE_INFO_FUCHSIA`

### Version History

- Revision 1, 2018-07-27 (Craig Stout)
  - Initial draft.

## VK_GGP_frame_token

### Name String

`VK_GGP_frame_token`

### Extension Type

Device extension

### Registered Extension Number

192

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_swapchain
and
VK_GGP_stream_descriptor_surface

### Contact

- Jean-Francois Roy [GitHub]jfroy

### Other Extension Metadata

#### Last Modified Date

2019-01-28

#### IP Status

No known IP claims.

#### Contributors

- Jean-Francois Roy, Google
- Richard O'Grady, Google

#### Description

This extension allows an application that uses the `VK_KHR_swapchain` extension in combination with a Google Games Platform surface provided by the `VK_GGP_stream_descriptor_surface` extension to associate a Google Games Platform frame token with a present operation.

#### New Structures

- Extending VkPresentInfoKHR:
  - VkPresentFrameTokenGGP

#### New Enum Constants

- `VK_GGP_FRAME_TOKEN_EXTENSION_NAME`
- `VK_GGP_FRAME_TOKEN_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PRESENT_FRAME_TOKEN_GGP`

#### Version History

- Revision 1, 2018-11-26 (Jean-Francois Roy)
  - Initial revision.

## VK_GGP_stream_descriptor_surface

#### Name String

`VK_GGP_stream_descriptor_surface`

#### Extension Type

Instance extension

#### Registered Extension Number

50

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_surface

### Contact

- Jean-Francois Roy [GitHub]jfroy

### Other Extension Metadata

### Last Modified Date

2019-01-28

### IP Status

No known IP claims.

### Contributors

- Jean-Francois Roy, Google
- Brad Grantham, Google
- Connor Smith, Google
- Cort Stratton, Google
- Hai Nguyen, Google
- Ian Elliott, Google
- Jesse Hall, Google
- Jim Ray, Google
- Katherine Wu, Google
- Kaye Mason, Google
- Kuangye Guo, Google
- Mark Segal, Google
- Nicholas Vining, Google
- Paul Lalonde, Google
- Richard O'Grady, Google

### Description

The `VK_GGP_stream_descriptor_surface` extension is an instance extension. It provides a mechanism to create a VkSurfaceKHR object (defined by the `VK_KHR_surface` extension) that refers to a Google Games Platform `GgpStreamDescriptor`.

### New Commands

- vkCreateStreamDescriptorSurfaceGGP

### New Structures

- VkStreamDescriptorSurfaceCreateInfoGGP

### New Bitmasks

- VkStreamDescriptorSurfaceCreateFlagsGGP

### New Enum Constants

- `VK_GGP_STREAM_DESCRIPTOR_SURFACE_EXTENSION_NAME`
- `VK_GGP_STREAM_DESCRIPTOR_SURFACE_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_STREAM_DESCRIPTOR_SURFACE_CREATE_INFO_GGP`

### Version History

- Revision 1, 2018-11-26 (Jean-Francois Roy)
  - Initial revision.

## VK_GOOGLE_decorate_string

### Name String

`VK_GOOGLE_decorate_string`

### Extension Type

Device extension

### Registered Extension Number

225

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

None

### SPIR-V Dependencies

- SPV_GOOGLE_decorate_string

### Contact

- Hai Nguyen [GitHub]chaoticbob

### Other Extension Metadata

### Last Modified Date

2018-07-09

### IP Status

No known IP claims.

### Contributors

- Hai Nguyen, Google
- Neil Henning, AMD

### Description

The `VK_GOOGLE_decorate_string` extension allows use of the `SPV_GOOGLE_decorate_string` extension in SPIR-V shader modules.

### New Enum Constants

- `VK_GOOGLE_DECORATE_STRING_EXTENSION_NAME`
- `VK_GOOGLE_DECORATE_STRING_SPEC_VERSION`

### Version History

- Revision 1, 2018-07-09 (Neil Henning)
  - Initial draft

## VK_GOOGLE_display_timing

### Name String

`VK_GOOGLE_display_timing`

### Extension Type

Device extension

### Registered Extension Number

93

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_swapchain

### Contact

- Ian Elliott [GitHub]ianelliottus

### Other Extension Metadata

### Last Modified Date

2017-02-14

### IP Status

No known IP claims.

### Contributors

- Ian Elliott, Google
- Jesse Hall, Google

### Description

This device extension allows an application that uses the `VK_KHR_swapchain` extension to obtain information about the presentation engine's display, to obtain timing information about each present, and to schedule a present to happen no earlier than a desired time. An application can use this to minimize various visual anomalies (e.g. stuttering).

Traditional game and real-time animation applications need to correctly position their geometry for when the presentable image will be presented to the user. To accomplish this, applications need various timing information about the presentation engine's display. They need to know when presentable images were actually presented, and when they could have been presented. Applications also need to tell the presentation engine to display an image no sooner than a given time. This allows the application to avoid stuttering, so the animation looks smooth to the user.

This extension treats variable-refresh-rate (VRR) displays as if they are fixed-refresh-rate (FRR) displays.

### New Commands

- vkGetPastPresentationTimingGOOGLE
- vkGetRefreshCycleDurationGOOGLE

### New Structures

- VkPastPresentationTimingGOOGLE
- VkPresentTimeGOOGLE
- VkRefreshCycleDurationGOOGLE
- Extending VkPresentInfoKHR:
  - VkPresentTimesInfoGOOGLE

### New Enum Constants

- `VK_GOOGLE_DISPLAY_TIMING_EXTENSION_NAME`
- `VK_GOOGLE_DISPLAY_TIMING_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PRESENT_TIMES_INFO_GOOGLE`

### Examples

| | |
|---|---|
| **NOTE** | The example code for the this extension (like the `VK_KHR_surface` and `VK_GOOGLE_display_timing` extensions) is contained in the cube demo that is shipped with the official Khronos SDK, and is being kept up-to-date in that location (see: https://github.com/KhronosGroup/Vulkan-Tools/blob/main/cube/cube.c ). |

### Version History

- Revision 1, 2017-02-14 (Ian Elliott)
  - Internal revisions

## VK_GOOGLE_hlsl_functionality1

### Name String

`VK_GOOGLE_hlsl_functionality1`

### Extension Type

Device extension

### Registered Extension Number

224

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

None

### SPIR-V Dependencies

- SPV_GOOGLE_hlsl_functionality1

### Contact

- Hai Nguyen [GitHub]chaoticbob

### Other Extension Metadata

#### Last Modified Date

2018-07-09

#### IP Status

No known IP claims.

#### Contributors

- Hai Nguyen, Google
- Neil Henning, AMD

#### Description

The `VK_GOOGLE_hlsl_functionality1` extension allows use of the `SPV_GOOGLE_hlsl_functionality1` extension in SPIR-V shader modules.

#### New Enum Constants

- `VK_GOOGLE_HLSL_FUNCTIONALITY1_EXTENSION_NAME`
- `VK_GOOGLE_HLSL_FUNCTIONALITY1_SPEC_VERSION`
- `VK_GOOGLE_HLSL_FUNCTIONALITY_1_EXTENSION_NAME`
- `VK_GOOGLE_HLSL_FUNCTIONALITY_1_SPEC_VERSION`

#### Version History

- Revision 1, 2018-07-09 (Neil Henning)
  - Initial draft

## VK_GOOGLE_surfaceless_query

#### Name String

`VK_GOOGLE_surfaceless_query`

#### Extension Type

Instance extension

#### Registered Extension Number

434

#### Revision

2

#### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_surface

### Special Use

- OpenGL / ES support

### Contact

- Shahbaz Youssefi [GitHub]syoussefi

### Extension Proposal

VK_GOOGLE_surfaceless_query

### Other Extension Metadata

### Last Modified Date

2022-08-03

### IP Status

No known IP claims.

### Contributors

- Ian Elliott, Google
- Shahbaz Youssefi, Google
- James Jones, NVIDIA

### Description

This extension allows the vkGetPhysicalDeviceSurfaceFormatsKHR and vkGetPhysicalDeviceSurfacePresentModesKHR functions to accept VK_NULL_HANDLE as their `surface` parameter, allowing potential surface formats, color spaces and present modes to be queried without providing a surface. Identically, vkGetPhysicalDeviceSurfaceFormats2KHR, vkGetPhysicalDeviceSurfacePresentModes2EXT, and vkGetPhysicalDeviceSurfaceCapabilities2KHR would accept VK_NULL_HANDLE in VkPhysicalDeviceSurfaceInfo2KHR::`surface`. **This can only be supported on platforms where the results of these queries are surface-agnostic and a single presentation engine is the implicit target of all present operations**.

### New Enum Constants

- `VK_GOOGLE_SURFACELESS_QUERY_EXTENSION_NAME`
- `VK_GOOGLE_SURFACELESS_QUERY_SPEC_VERSION`

### Version History

- Revision 1, 2021-12-14 (Shahbaz Youssefi)
  - Internal revisions
- Revision 2, 2022-08-03 (Shahbaz Youssefi)
  - Precisions to which parts of the query responses are defined when surfaceless

## VK_GOOGLE_user_type

### Name String

`VK_GOOGLE_user_type`

### Extension Type

Device extension

### Registered Extension Number

290

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

None

### SPIR-V Dependencies

- SPV_GOOGLE_user_type

### Contact

- Kaye Mason [GitHub]chaleur

### Other Extension Metadata

### Last Modified Date

2019-07-09

### IP Status

No known IP claims.

### Contributors

- Kaye Mason, Google
- Hai Nguyen, Google

### Description

The `VK_GOOGLE_user_type` extension allows use of the `SPV_GOOGLE_user_type` extension in SPIR-V shader modules.

### New Enum Constants

- `VK_GOOGLE_USER_TYPE_EXTENSION_NAME`
- `VK_GOOGLE_USER_TYPE_SPEC_VERSION`

### Version History

- Revision 1, 2019-09-07 (Kaye Mason)
  - Initial draft

## VK_HUAWEI_cluster_culling_shader

### Name String

`VK_HUAWEI_cluster_culling_shader`

### Extension Type

Device extension

### Registered Extension Number

405

### Revision

3

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### SPIR-V Dependencies

- SPV_HUAWEI_cluster_culling_shader

### Contact

- Yuchang Wang [GitHub]richard_Wang2

### Extension Proposal

VK_HUAWEI_cluster_culling_shader

### Other Extension Metadata

### Last Modified Date

2023-08-16

### Interactions and External Dependencies

- This extension provides API support for `GL_HUAWEI_cluster_culling_shader`.

### Contributors

- Yuchang Wang, Huawei
- Juntao Li, Huawei
- Pan Gao, Huawei

- Jie Cao, Huawei
- Yunjin Zhang, Huawei
- Shujie Zhou, Huawei
- Chaojun Wang, Huawei
- Jiajun Hu, Huawei
- Cong Zhang, Huawei

**Description**

Cluster Culling Shaders (CCS) are similar to the existing compute shaders. Their main purpose is to provide an execution environment in order to perform coarse-level geometry culling and LOD selection more efficiently on the GPU.

The traditional 2-pass GPU culling solution using a compute shader sometimes needs a pipeline barrier between compute and graphics pipeline to optimize performance. An additional compaction process may also be required. This extension addresses these shortcomings, allowing compute shaders to directly emit visible clusters to the following graphics pipeline.

A set of new built-in output variables are used to express a visible cluster, including per-cluster shading rate. In addition, a new built-in function is used to emit these variables from CCS to the IA stage. The IA stage can use these variables to fetches vertices of a visible cluster and drive vertex shaders to shading these vertices.

Note that CCS do not work with geometry or tessellation shaders, but both IA and vertex shaders are preserved. Vertex shaders are still used for vertex position shading, instead of directly outputting transformed vertices from the compute shader. This makes CCS more suitable for mobile GPUs.

**New Commands**

- vkCmdDrawClusterHUAWEI
- vkCmdDrawClusterIndirectHUAWEI

**New Structures**

- Extending VkPhysicalDeviceClusterCullingShaderFeaturesHUAWEI:
  - VkPhysicalDeviceClusterCullingShaderVrsFeaturesHUAWEI
- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceClusterCullingShaderFeaturesHUAWEI
- Extending VkPhysicalDeviceProperties2:
  - VkPhysicalDeviceClusterCullingShaderPropertiesHUAWEI

**New Enum Constants**

- `VK_HUAWEI_CLUSTER_CULLING_SHADER_EXTENSION_NAME`

- VK_HUAWEI_CLUSTER_CULLING_SHADER_SPEC_VERSION
- Extending VkPipelineStageFlagBits2:
  - VK_PIPELINE_STAGE_2_CLUSTER_CULLING_SHADER_BIT_HUAWEI
- Extending VkQueryPipelineStatisticFlagBits:
  - VK_QUERY_PIPELINE_STATISTIC_CLUSTER_CULLING_SHADER_INVOCATIONS_BIT_HUAWEI
- Extending VkShaderStageFlagBits:
  - VK_SHADER_STAGE_CLUSTER_CULLING_BIT_HUAWEI
- Extending VkStructureType:
  - VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_CLUSTER_CULLING_SHADER_FEATURES_HUAWEI
  - VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_CLUSTER_CULLING_SHADER_PROPERTIES_HUAWEI
  - VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_CLUSTER_CULLING_SHADER_VRS_FEATURES_HUAWEI

### New Built-In Variables

- IndexCountHUAWEI
- VertexCountHUAWEI
- InstanceCountHUAWEI
- FirstIndexHUAWEI
- FirstVertexHUAWEI
- VertexOffsetHUAWEI
- FirstInstanceHUAWEI
- ClusterIDHUAWEI
- ClusterShadingRateHUAWEI

### New SPIR-V Capability

- ClusterCullingShadingHUAWEI

### Sample Code

Example of cluster culling in a GLSL shader

```
#extension GL_HUAWEI_cluster_culling_shader: enable

#define GPU_WARP_SIZE                    32
#define GPU_GROUP_SIZE                   GPU_WARP_SIZE

#define GPU_CLUSTER_PER_INVOCATION       1
#define GPU_CLUSTER_PER_WORKGROUP        (GPU_GROUP_SIZE * GPU_CLUSTER_PER_INVOCATION)

// Number of threads per workgroup
// - 1D only
```

```
// - warpsize = 32
layout(local_size_x=GPU_GROUP_SIZE, local_size_y=1, local_size_z=1) in;

#define GPU_DRAW_BUFFER_BINDING             0
#define GPU_INSTANCE_DESCRIPTOR_BINDING     1

struct BoundingSphere
{
  vec3 center;
  float radius;
};

struct InstanceData
{
  mat4 mvp_matrix;                      // mvp matrix.
  vec4 frustum_planes[6];               // six frustum planes
  mat4 model_matrix_transpose_inverse;  // inverse transpose of model matrix.
  vec3 view_origin;                     // view original
};

struct InstanceDescriptor
{
  uint begin;
  uint end;
  uint cluster_count;
  uint debug;
  BoundingSphere sphere;
  InstanceData instance_data;
};

struct DrawElementsCommand{
  uint indexcount;
  uint instanceCount;
  uint firstIndex;
  int  vertexoffset;
  uint firstInstance;
  uint cluster_id;
};

// indexed mode
out gl_PerClusterHUAWEI{
  uint gl_IndexCountHUAWEI;
  uint gl_InstanceCountHUAWEI;
  uint gl_FirstIndexHUAWEI;
  int  gl_VertexOffsetHUAWEI;
  uint gl_FirstInstanceHUAWEI;
  uint gl_ClusterIDHUAWEI;
  uint gl_ClusterShadingRateHUAWEI;
};

layout(binding = GPU_DRAW_BUFFER_BINDING, std430) buffer draw_indirect_ssbo
```

```
{
        DrawElementsCommand draw_commands[];
};

layout(binding = GPU_INSTANCE_DESCRIPTOR_BINDING, std430) buffer
instance_descriptor_ssbo
{
        InstanceDescriptor instance_descriptors[];
};


float Distance(uint instance_id)
{
    vec3 v = normalize(instance_descriptor[instance_id].sphere.center -
                       instance_descriptor[instance_id].instance_data.view_origin);
    float dist = sqrt(dot(v,v));

    return dist;
}

bool isSphereOutsideFrustum( vec3 sphere_center, float sphere_radius )
{
  bool isInside = false;

  for(int i = 0; i < 6; i++)
  {
      isInside = isInside ||
      (dot(instance_descriptors[instance_id].instance_data.frustum_planes[i].xyz,
      sphere_center) + instance_descriptors[instance_id].instance_data.frustum_planes
[i].w <
      sphere_radius);
  }
  return isInside;
}


void main()
{
    // get instance description
    instance_id = gl_GlobalInvocationID.x;
    InstanceDescriptor inst_desc = instance_descriptors[instance_id];

    //instance based culling
    bool render = !isSphereOutsideFrustum(inst_desc.sphere.center, inst_desc.sphere
.radius);

    if (render)
    {
        // calculate distance
        float distance = Distance(instance_id);
```

```
        // update shading rate built-in variable
        if(distance > 0.7)
            gl_ClusterShadingRateHUAWEI =
                gl_ShadingRateFlag4VerticalPixelsEXT |
gl_ShadingRateFlag4HorizontalPixelsEXT;
        else if(distance > 0.3)
            gl_ClusterShadingRateHUAWEI =
                gl_ShadingRateFlag2VerticalPixelsEXT |
gl_ShadingRateFlag2HorizontalPixelsEXT;
        else
            gl_ClusterShadingRateHUAWEI = 0;

        // this is a visible cluster, update built-in output variable.
        // in case of indexed mode:
        gl_IndexCountHUAWEI     = draw_commands[cluster_id].indexcount;
        gl_InstanceCountHUAWEI  = draw_commands[cluster_id].instanceCount;
        gl_FirstIndexHUAWEI     = draw_commands[cluster_id].firstIndex;
        gl_VertexOffsetHUAWEI   = draw_commands[cluster_id].vertexoffset;
        gl_FirstInstanceHUAWEI  = draw_commands[cluster_id].firstInstance;
        gl_ClusterIDHUAWEI      = draw_commands[cluster_id].cluster_id;

        // emit built-in output variables as a drawing command to subsequent
        // rendering pipeline.
        dispatchClusterHUAWEI();
    }
}
```

Example of graphics pipeline creation with cluster culling shader

```
// create a cluster culling shader stage info structure.
VkPipelineShaderStageCreateInfo ccsStageInfo{};
ccsStageInfo.sType = VK_STRUCTURE_TYPE_PIPELINE_SHADER_STAGE_CREATE_INFO;
ccsStageInfo.stage = VK_SHADER_STAGE_CLUSTER_CULLING_BIT_HUAWEI;
ccsStageInfo.module = clustercullingshaderModule;
ccsStageInfo.pName =  "main";

// pipeline shader stage creation
VkPipelineShaderStageCreateInfo shaderStages[] = { ccsStageInfo,
vertexShaderStageInfo, fragmentShaderStageInfo };

// create graphics pipeline
VkGraphicsPipelineCreateInfo pipelineInfo{};
pipelineInfo.sType = VK_STRUCTURE_TYPE_GRAPHICS_PIPELINE_CREATE_INFO;
pipelineInfo.stageCount = 3;
pipelineInfo.pStage = shaderStages;
pipelineInfo.pVertexInputState = &vertexInputInfo;
// ...
VkPipeline graphicsPipeline;
VkCreateGraphicsPipelines(device, VK_NULL_HANDLE, 1, &pipelineInfo, nullptr,
```

```
&graphicsPipeline);
```

Example of launching the execution of cluster culling shader

```
vkCmdBindPipeline(commandBuffer, VK_PIPELINE_BIND_POINT_GRAPHICS, graphicsPipeline);
vkCmdDrawClusterHUAWEI(commandBuffer, groupCountX, 1, 1);
vkCmdEndRenderPass(commandBuffer);
```

#### Version History

- Revision 1, 2022-11-18 (YuChang Wang)
  - Internal revisions
- Revision 2, 2023-04-02 (Jon Leech)
  - Grammar edits.
- Revision 3, 2023-08-21 (YuChang Wang)
  - Add per-cluster shading rate.

## VK_HUAWEI_invocation_mask

#### Name String

`VK_HUAWEI_invocation_mask`

#### Extension Type

Device extension

#### Registered Extension Number

371

#### Revision

1

#### Ratification Status

Not ratified

#### Extension and Version Dependencies

VK_KHR_ray_tracing_pipeline
and
VK_KHR_synchronization2
or
Vulkan Version 1.3

#### Contact

- Pan Gao [GitHub]PanGao-h

### Extension Proposal

VK_HUAWEI_invocation_mask

### Other Extension Metadata

### Last Modified Date

2021-05-27

### Interactions and External Dependencies

- This extension requires `VK_KHR_ray_tracing_pipeline`, which allow to bind an invocation mask image before the ray tracing command
- This extension requires `VK_KHR_synchronization2`, which allows new pipeline stage for the invocation mask image

### Contributors

- Yunpeng Zhu
- Juntao Li, Huawei
- Liang Chen, Huawei
- Shaozhuang Shi, Huawei
- Hailong Chu, Huawei

### Description

The rays to trace may be sparse in some use cases. For example, the scene only have a few regions to reflect. Providing an invocation mask image to the ray tracing commands could potentially give the hardware the hint to do certain optimization without invoking an additional pass to compact the ray buffer.

### New Commands

- vkCmdBindInvocationMaskHUAWEI

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceInvocationMaskFeaturesHUAWEI

### New Enum Constants

- `VK_HUAWEI_INVOCATION_MASK_EXTENSION_NAME`
- `VK_HUAWEI_INVOCATION_MASK_SPEC_VERSION`
- Extending VkAccessFlagBits2:
  - `VK_ACCESS_2_INVOCATION_MASK_READ_BIT_HUAWEI`
- Extending VkImageUsageFlagBits:
  - `VK_IMAGE_USAGE_INVOCATION_MASK_BIT_HUAWEI`

- Extending VkPipelineStageFlagBits2:
  - VK_PIPELINE_STAGE_2_INVOCATION_MASK_BIT_HUAWEI
- Extending VkStructureType:
  - VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_INVOCATION_MASK_FEATURES_HUAWEI

### Examples

RT mask is updated before each traceRay.

Step 1. Generate InvocationMask.

```
//the rt mask image bind as color attachment in the fragment shader
Layout(location = 2) out vec4 outRTmask
vec4 mask = vec4(x,x,x,x);
outRTmask = mask;
```

Step 2. traceRay with InvocationMask

```
vkCmdBindPipeline(
    commandBuffers[imageIndex],
    VK_PIPELINE_BIND_POINT_RAY_TRACING_KHR, m_rtPipeline);
    vkCmdBindDescriptorSets(commandBuffers[imageIndex],
    VK_PIPELINE_BIND_POINT_RAY_TRACING_NV,
    m_rtPipelineLayout, 0, 1, &m_rtDescriptorSet,
    0, nullptr);

vkCmdBindInvocationMaskHUAWEI(
    commandBuffers[imageIndex],
    InvocationMaskimageView,
    InvocationMaskimageLayout);
    vkCmdTraceRaysKHR(commandBuffers[imageIndex],
    pRaygenShaderBindingTable,
    pMissShaderBindingTable,
    swapChainExtent.width,
    swapChainExtent.height, 1);
```

### Version History

- Revision 1, 2021-05-27 (Yunpeng Zhu)
  - Initial draft.

## VK_HUAWEI_subpass_shading

### Name String

VK_HUAWEI_subpass_shading

**Extension Type**

Device extension

**Registered Extension Number**

370

**Revision**

3

**Ratification Status**

Not ratified

**Extension and Version Dependencies**

VK_KHR_create_renderpass2
or
Vulkan Version 1.2
and
VK_KHR_synchronization2
or
Vulkan Version 1.3

**SPIR-V Dependencies**

- SPV_HUAWEI_subpass_shading

**Contact**

- Pan Gao [GitHub]PanGao-h

**Other Extension Metadata**

**Last Modified Date**

2021-06-01

**Interactions and External Dependencies**

- This extension provides API support for `GL_HUAWEI_subpass_shading`.

**Contributors**

- Hueilong Wang
- Juntao Li, Huawei
- Renmiao Lu, Huawei
- Pan Gao, Huawei

**Description**

This extension allows applications to execute a subpass shading pipeline in a subpass of a render pass in order to save memory bandwidth for algorithms like tile-based deferred rendering and forward plus. A subpass shading pipeline is a pipeline with the compute pipeline ability, allowed to read values from input attachments, and only allowed to be dispatched inside a stand-alone subpass. Its work dimension is defined by the render pass's render area size. Its workgroup size

(width, height) shall be a power-of-two number in width or height, with minimum value from 8, and maximum value shall be decided from the render pass attachments and sample counts but depends on implementation.

The `GlobalInvocationId.xy` of a subpass shading pipeline is equal to the `FragCoord.xy` of a graphic pipeline in the same render pass subtracted the `offset` of the VkRenderPassBeginInfo::`renderArea`. `GlobalInvocationId.z` is mapped to the Layer if `VK_EXT_shader_viewport_index_layer` is supported. The `GlobalInvocationId.xy` is equal to the index of the local workgroup multiplied by the size of the local workgroup plus the `LocalInvocationId` and the `offset` of the VkRenderPassBeginInfo::`renderArea`.

This extension allows a subpass's pipeline bind point to be `VK_PIPELINE_BIND_POINT_SUBPASS_SHADING_HUAWEI`.

### New Commands

- vkCmdSubpassShadingHUAWEI
- vkGetDeviceSubpassShadingMaxWorkgroupSizeHUAWEI

### New Structures

- Extending VkComputePipelineCreateInfo:
  - VkSubpassShadingPipelineCreateInfoHUAWEI
- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceSubpassShadingFeaturesHUAWEI
- Extending VkPhysicalDeviceProperties2:
  - VkPhysicalDeviceSubpassShadingPropertiesHUAWEI

### New Enum Constants

- `VK_HUAWEI_SUBPASS_SHADING_EXTENSION_NAME`
- `VK_HUAWEI_SUBPASS_SHADING_SPEC_VERSION`
- Extending VkPipelineBindPoint:
  - `VK_PIPELINE_BIND_POINT_SUBPASS_SHADING_HUAWEI`
- Extending VkPipelineStageFlagBits2:
  - `VK_PIPELINE_STAGE_2_SUBPASS_SHADER_BIT_HUAWEI`
  - `VK_PIPELINE_STAGE_2_SUBPASS_SHADING_BIT_HUAWEI`
- Extending VkShaderStageFlagBits:
  - `VK_SHADER_STAGE_SUBPASS_SHADING_BIT_HUAWEI`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SUBPASS_SHADING_FEATURES_HUAWEI`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SUBPASS_SHADING_PROPERTIES_HUAWEI`

- VK_STRUCTURE_TYPE_SUBPASS_SHADING_PIPELINE_CREATE_INFO_HUAWEI

**Sample Code**

Example of subpass shading in a GLSL shader

```
#extension GL_HUAWEI_subpass_shading: enable
#extension GL_KHR_shader_subgroup_arithmetic: enable

layout(constant_id = 0) const uint tileWidth = 8;
layout(constant_id = 1) const uint tileHeight = 8;
layout(local_size_x_id = 0, local_size_y_id = 1, local_size_z = 1) in;
layout(set=0, binding=0, input_attachment_index=0) uniform subpassInput depth;

void main()
{
  float d = subpassLoad(depth).x;
  float minD = subgroupMin(d);
  float maxD = subgroupMax(d);
}
```

Example of subpass shading dispatching in a subpass

```
vkCmdNextSubpass(commandBuffer, VK_SUBPASS_CONTENTS_INLINE);
vkCmdBindPipeline(commandBuffer, VK_PIPELINE_BIND_POINT_SUBPASS_SHADING_HUAWEI,
subpassShadingPipeline);
vkCmdBindDescriptorSets(commandBuffer, VK_PIPELINE_BIND_POINT_SUBPASS_SHADING_HUAWEI,
subpassShadingPipelineLayout,
  firstSet, descriptorSetCount, pDescriptorSets, dynamicOffsetCount, pDynamicOffsets);
vkCmdSubpassShadingHUAWEI(commandBuffer)
vkCmdEndRenderPass(commandBuffer);
```

Example of subpass shading render pass creation

```
VkAttachmentDescription2 attachments[] = {
  {
    VK_STRUCTURE_TYPE_ATTACHMENT_DESCRIPTION_2, NULL,
    0, VK_FORMAT_R8G8B8A8_UNORM, VK_SAMPLE_COUNT_1_BIT,
    VK_ATTACHMENT_LOAD_OP_CLEAR, VK_ATTACHMENT_STORE_OP_DONT_CARE,
    VK_ATTACHMENT_LOAD_OP_DONT_CARE, VK_ATTACHMENT_LOAD_OP_DONT_CARE,
    VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL, VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL
  },
  {
    VK_STRUCTURE_TYPE_ATTACHMENT_DESCRIPTION_2, NULL,
    0, VK_FORMAT_R8G8B8A8_UNORM, VK_SAMPLE_COUNT_1_BIT,
    VK_ATTACHMENT_LOAD_OP_CLEAR, VK_ATTACHMENT_STORE_OP_DONT_CARE,
    VK_ATTACHMENT_LOAD_OP_DONT_CARE, VK_ATTACHMENT_LOAD_OP_DONT_CARE,
    VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL, VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL
```

```
  },
  {
    VK_STRUCTURE_TYPE_ATTACHMENT_DESCRIPTION_2, NULL,
    0, VK_FORMAT_R8G8B8A8_UNORM, VK_SAMPLE_COUNT_1_BIT,
    VK_ATTACHMENT_LOAD_OP_CLEAR, VK_ATTACHMENT_STORE_OP_DONT_CARE,
    VK_ATTACHMENT_LOAD_OP_DONT_CARE, VK_ATTACHMENT_LOAD_OP_DONT_CARE,
    VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL, VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL
  },
  {
    VK_STRUCTURE_TYPE_ATTACHMENT_DESCRIPTION_2, NULL,
    0, VK_FORMAT_D24_UNORM_S8_UINT, VK_SAMPLE_COUNT_1_BIT,
    VK_ATTACHMENT_LOAD_OP_CLEAR, VK_ATTACHMENT_STORE_OP_DONT_CARE,
    VK_ATTACHMENT_LOAD_OP_CLEAR, VK_ATTACHMENT_LOAD_OP_DONT_CARE,
    VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL,
VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL
  },
  {
    VK_STRUCTURE_TYPE_ATTACHMENT_DESCRIPTION_2, NULL,
    0, VK_FORMAT_R8G8B8A8_UNORM, VK_SAMPLE_COUNT_1_BIT,
    VK_ATTACHMENT_LOAD_OP_CLEAR, VK_ATTACHMENT_STORE_OP_STORE,
    VK_ATTACHMENT_LOAD_OP_DONT_CARE, VK_ATTACHMENT_LOAD_OP_DONT_CARE,
    VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL, VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL
  }
};

VkAttachmentReference2 gBufferAttachmentReferences[] = {
  { VK_STRUCTURE_TYPE_ATTACHMENT_REFERENCE_2, NULL, 0,
VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL, VK_IMAGE_ASPECT_COLOR_BIT },
  { VK_STRUCTURE_TYPE_ATTACHMENT_REFERENCE_2, NULL, 1,
VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL, VK_IMAGE_ASPECT_COLOR_BIT },
  { VK_STRUCTURE_TYPE_ATTACHMENT_REFERENCE_2, NULL, 2,
VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL, VK_IMAGE_ASPECT_COLOR_BIT }
};
VkAttachmentReference2 gBufferDepthStencilAttachmentReferences =
  { VK_STRUCTURE_TYPE_ATTACHMENT_REFERENCE_2, NULL, 3,
VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL, VK_IMAGE_ASPECT_DEPTH_BIT
|VK_IMAGE_ASPECT_STENCIL_BIT };
VkAttachmentReference2 depthInputAttachmentReferences[] = {
  { VK_STRUCTURE_TYPE_ATTACHMENT_REFERENCE_2, NULL, 3,
VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL, VK_IMAGE_ASPECT_DEPTH_BIT
|VK_IMAGE_ASPECT_STENCIL_BIT };
};
VkAttachmentReference2 preserveAttachmentReferences[] = {
  { VK_STRUCTURE_TYPE_ATTACHMENT_REFERENCE_2, NULL, 0,
VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL, VK_IMAGE_ASPECT_COLOR_BIT },
  { VK_STRUCTURE_TYPE_ATTACHMENT_REFERENCE_2, NULL, 1,
VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL, VK_IMAGE_ASPECT_COLOR_BIT },
  { VK_STRUCTURE_TYPE_ATTACHMENT_REFERENCE_2, NULL, 2,
VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL, VK_IMAGE_ASPECT_COLOR_BIT },
  { VK_STRUCTURE_TYPE_ATTACHMENT_REFERENCE_2, NULL, 3,
VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL, VK_IMAGE_ASPECT_DEPTH_BIT
```

```
|VK_IMAGE_ASPECT_STENCIL_BIT }
}; // G buffer including depth/stencil
VkAttachmentReference2 colorAttachmentReferences[] = {
  { VK_STRUCTURE_TYPE_ATTACHMENT_REFERENCE_2, NULL, 4,
VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL, VK_IMAGE_ASPECT_COLOR_BIT }
};
VkAttachmentReference2 resolveAttachmentReference =
  { VK_STRUCTURE_TYPE_ATTACHMENT_REFERENCE_2, NULL, 4,
VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL, VK_IMAGE_ASPECT_COLOR_BIT };

VkSubpassDescription2 subpasses[] = {
  {
    VK_STRUCTURE_TYPE_SUBPASS_DESCRIPTION_2, NULL, 0, VK_PIPELINE_BIND_POINT_GRAPHICS,
0,
    0, NULL, // input
    sizeof(gBufferAttachmentReferences)/sizeof(gBufferAttachmentReferences[0]),
gBufferAttachmentReferences, // color
    NULL, &gBufferDepthStencilAttachmentReferences, // resolve & DS
    0, NULL
  },
  {
    VK_STRUCTURE_TYPE_SUBPASS_DESCRIPTION_2, NULL, 0,
VK_PIPELINE_BIND_POINT_SUBPASS_SHADING_HUAWEI , 0,
    sizeof(depthInputAttachmentReferences)/sizeof(depthInputAttachmentReferences[0]),
depthInputAttachmentReferences, // input
    0, NULL, // color
    NULL, NULL, // resolve & DS
    sizeof(preserveAttachmentReferences)/sizeof(preserveAttachmentReferences[0]),
preserveAttachmentReferences,
  },
  {
    VK_STRUCTURE_TYPE_SUBPASS_DESCRIPTION_2, NULL, 0, VK_PIPELINE_BIND_POINT_GRAPHICS,
0,
    sizeof(gBufferAttachmentReferences)/sizeof(gBufferAttachmentReferences[0]),
gBufferAttachmentReferences, // input
    sizeof(colorAttachmentReferences)/sizeof(colorAttachmentReferences[0]),
colorAttachmentReferences, // color
    &resolveAttachmentReference, &gBufferDepthStencilAttachmentReferences, // resolve
& DS
    0, NULL
  },
};

VkMemoryBarrier2KHR fragmentToSubpassShading = {
  VK_STRUCTURE_TYPE_MEMORY_BARRIER_2_KHR, NULL,
  VK_PIPELINE_STAGE_2_FRAGMENT_SHADER_BIT_KHR, VK_ACCESS_COLOR_ATTACHMENT_WRITE_BIT
|VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_WRITE_BIT,
  VK_PIPELINE_STAGE_2_SUBPASS_SHADER_BIT_HUAWEI, VK_ACCESS_INPUT_ATTACHMENT_READ_BIT
};

VkMemoryBarrier2KHR subpassShadingToFragment = {
```

```
  VK_STRUCTURE_TYPE_MEMORY_BARRIER_2_KHR, NULL,
  VK_PIPELINE_STAGE_2_SUBPASS_SHADER_BIT_HUAWEI, VK_ACCESS_SHADER_WRITE_BIT,
  VK_PIPELINE_STAGE_2_FRAGMENT_SHADER_BIT_KHR, VK_ACCESS_SHADER_READ_BIT
};

VkSubpassDependency2 dependencies[] = {
  {
    VK_STRUCTURE_TYPE_SUBPASS_DEPENDENCY_2, &fragmentToSubpassShading,
    0, 1,
    0, 0, 0, 0,
    0, 0
  },
  {
    VK_STRUCTURE_TYPE_SUBPASS_DEPENDENCY_2, &subpassShadingToFragment,
    1, 2,
    0, 0, 0, 0,
    0, 0
  },
};

VkRenderPassCreateInfo2 renderPassCreateInfo = {
  VK_STRUCTURE_TYPE_RENDER_PASS_CREATE_INFO_2, NULL, 0,
    sizeof(attachments)/sizeof(attachments[0]), attachments,
    sizeof(subpasses)/sizeof(subpasses[0]), subpasses,
    sizeof(dependencies)/sizeof(dependencies[0]), dependencies,
    0, NULL
};
VKRenderPass renderPass;
vkCreateRenderPass2(device, &renderPassCreateInfo, NULL, &renderPass);
```

Example of subpass shading pipeline creation

```
VkExtent2D maxWorkgroupSize;

VkSpecializationMapEntry subpassShadingConstantMapEntries[] = {
  { 0, 0 * sizeof(uint32_t), sizeof(uint32_t) },
  { 1, 1 * sizeof(uint32_t), sizeof(uint32_t) }
};

VkSpecializationInfo subpassShadingConstants = {
  2, subpassShadingConstantMapEntries,
  sizeof(VkExtent2D), &maxWorkgroupSize
};

VkSubpassShadingPipelineCreateInfoHUAWEI subpassShadingPipelineCreateInfo {
  VK_STRUCTURE_TYPE_SUBPASSS_SHADING_PIPELINE_CREATE_INFO_HUAWEI, NULL,
  renderPass, 1
};

VkPipelineShaderStageCreateInfo subpassShadingPipelineStageCreateInfo {
```

```
  VK_STRUCTURE_TYPE_PIPELINE_SHADER_STAGE_CREATE_INFO, NULL,
  0, VK_SHADER_STAGE_SUBPASS_SHADING_BIT_HUAWEI,
  shaderModule, "main",
  &subpassShadingConstants
};

VkComputePipelineCreateInfo subpassShadingComputePipelineCreateInfo = {
  VK_STRUCTURE_TYPE_COMPUTE_PIPELINE_CREATE_INFO, &subpassShadingPipelineCreateInfo,
  0, &subpassShadingPipelineStageCreateInfo,
  pipelineLayout, basePipelineHandle, basePipelineIndex
};

VKPipeline pipeline;

vkGetDeviceSubpassShadingMaxWorkgroupSizeHUAWEI(device, renderPass, &
maxWorkgroupSize);
vkCreateComputePipelines(device, pipelineCache, 1,
&subpassShadingComputePipelineCreateInfo, NULL, &pipeline);
```

### Version History

- Revision 3, 2023-06-19 (Pan Gao)
  - Rename `VK_PIPELINE_STAGE_2_SUBPASS_SHADING_BIT_HUAWEI` to `VK_PIPELINE_STAGE_2_SUBPASS_SHADER_BIT_HUAWEI` to better aligned with naming of other pipeline stages
- Revision 2, 2021-06-28 (Hueilong Wang)
  - Change vkGetSubpassShadingMaxWorkgroupSizeHUAWEI to vkGetDeviceSubpassShadingMaxWorkgroupSizeHUAWEI to resolve issue `pub1564`
- Revision 1, 2020-12-15 (Hueilong Wang)
  - Initial draft.

## VK_IMG_filter_cubic

### Name String

`VK_IMG_filter_cubic`

### Extension Type

Device extension

### Registered Extension Number

16

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

None

### Contact

- Tobias Hector [GitHub]tobski

### Other Extension Metadata

### Last Modified Date

2016-02-23

### Contributors

- Tobias Hector, Imagination Technologies

### Description

`VK_IMG_filter_cubic` adds an additional, high quality cubic filtering mode to Vulkan, using a Catmull-Rom bicubic filter. Performing this kind of filtering can be done in a shader by using 16 samples and a number of instructions, but this can be inefficient. The cubic filter mode exposes an optimized high quality texture sampling using fixed texture sampling functionality.

### New Enum Constants

- `VK_IMG_FILTER_CUBIC_EXTENSION_NAME`
- `VK_IMG_FILTER_CUBIC_SPEC_VERSION`
- Extending VkFilter:
  - `VK_FILTER_CUBIC_IMG`
- Extending VkFormatFeatureFlagBits:
  - `VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_CUBIC_BIT_IMG`

### Example

Creating a sampler with the new filter for both magnification and minification

```
VkSamplerCreateInfo createInfo =
{
    .sType = VK_STRUCTURE_TYPE_SAMPLER_CREATE_INFO,
    // Other members set to application-desired values
};

createInfo.magFilter = VK_FILTER_CUBIC_IMG;
createInfo.minFilter = VK_FILTER_CUBIC_IMG;

VkSampler sampler;
VkResult result = vkCreateSampler(
    device,
    &createInfo,
```

```
        &sampler);
```

### Version History

- Revision 1, 2016-02-23 (Tobias Hector)
  - Initial version

## VK_IMG_relaxed_line_rasterization

### Name String

`VK_IMG_relaxed_line_rasterization`

### Extension Type

Device extension

### Registered Extension Number

111

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### Special Use

- OpenGL / ES support

### Contact

- James Fitzpatrick [GitHub]jamesfitzpatrick

### Other Extension Metadata

### Last Modified Date

2023-10-22

### IP Status

No known IP claims.

### Contributors

- James Fitzpatrick, Imagination
- Andrew Garrard, Imagination
- Alex Walters, Imagination

### Description

OpenGL specifies that implementations should rasterize lines using the diamond exit rule (a slightly modified version of Bresenham's algorithm). To implement OpenGL some implementations have a device-level compatibility mode to rasterize lines according to the OpenGL specification.

This extension allows OpenGL emulation layers to enable the OpenGL compatible line rasterization mode of such implementations.

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceRelaxedLineRasterizationFeaturesIMG

### New Enum Constants

- `VK_IMG_RELAXED_LINE_RASTERIZATION_EXTENSION_NAME`
- `VK_IMG_RELAXED_LINE_RASTERIZATION_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_RELAXED_LINE_RASTERIZATION_FEATURES_IMG`

### Issues

None.

### Version History

- Revision 1, 2023-10-22 (James Fitzpatrick)
  - Initial version

## VK_INTEL_performance_query

### Name String

`VK_INTEL_performance_query`

### Extension Type

Device extension

### Registered Extension Number

211

### Revision

2

### Ratification Status

Not ratified

### Extension and Version Dependencies

None

### Special Use

- Developer tools

### Contact

- Lionel Landwerlin [GitHub]llandwerlin

### Other Extension Metadata

### Last Modified Date

2018-05-16

### IP Status

No known IP claims.

### Contributors

- Lionel Landwerlin, Intel
- Piotr Maciejewski, Intel

### Description

This extension allows an application to capture performance data to be interpreted by an external application or library.

Such a library is available at : https://github.com/intel/metrics-discovery

Performance analysis tools such as Graphics Performance Analyzers make use of this extension and the metrics-discovery library to present the data in a human readable way.

### New Object Types

- VkPerformanceConfigurationINTEL

### New Commands

- vkAcquirePerformanceConfigurationINTEL
- vkCmdSetPerformanceMarkerINTEL
- vkCmdSetPerformanceOverrideINTEL
- vkCmdSetPerformanceStreamMarkerINTEL
- vkGetPerformanceParameterINTEL
- vkInitializePerformanceApiINTEL
- vkQueueSetPerformanceConfigurationINTEL
- vkReleasePerformanceConfigurationINTEL
- vkUninitializePerformanceApiINTEL

### New Structures

- VkInitializePerformanceApiInfoINTEL
- VkPerformanceConfigurationAcquireInfoINTEL
- VkPerformanceMarkerInfoINTEL
- VkPerformanceOverrideInfoINTEL
- VkPerformanceStreamMarkerInfoINTEL
- VkPerformanceValueINTEL
- Extending VkQueryPoolCreateInfo:
  - VkQueryPoolCreateInfoINTEL
  - VkQueryPoolPerformanceQueryCreateInfoINTEL

### New Unions

- VkPerformanceValueDataINTEL

### New Enums

- VkPerformanceConfigurationTypeINTEL
- VkPerformanceOverrideTypeINTEL
- VkPerformanceParameterTypeINTEL
- VkPerformanceValueTypeINTEL
- VkQueryPoolSamplingModeINTEL

### New Enum Constants

- `VK_INTEL_PERFORMANCE_QUERY_EXTENSION_NAME`
- `VK_INTEL_PERFORMANCE_QUERY_SPEC_VERSION`
- Extending VkObjectType:
  - `VK_OBJECT_TYPE_PERFORMANCE_CONFIGURATION_INTEL`
- Extending VkQueryType:
  - `VK_QUERY_TYPE_PERFORMANCE_QUERY_INTEL`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_INITIALIZE_PERFORMANCE_API_INFO_INTEL`
  - `VK_STRUCTURE_TYPE_PERFORMANCE_CONFIGURATION_ACQUIRE_INFO_INTEL`
  - `VK_STRUCTURE_TYPE_PERFORMANCE_MARKER_INFO_INTEL`
  - `VK_STRUCTURE_TYPE_PERFORMANCE_OVERRIDE_INFO_INTEL`
  - `VK_STRUCTURE_TYPE_PERFORMANCE_STREAM_MARKER_INFO_INTEL`
  - `VK_STRUCTURE_TYPE_QUERY_POOL_CREATE_INFO_INTEL`
  - `VK_STRUCTURE_TYPE_QUERY_POOL_PERFORMANCE_QUERY_CREATE_INFO_INTEL`

### Example Code

```
// A previously created device
VkDevice device;

// A queue derived from the device
VkQueue queue;

VkInitializePerformanceApiInfoINTEL performanceApiInfoIntel = {
  VK_STRUCTURE_TYPE_INITIALIZE_PERFORMANCE_API_INFO_INTEL,
  NULL,
  NULL
};

vkInitializePerformanceApiINTEL(
  device,
  &performanceApiInfoIntel);

VkQueryPoolPerformanceQueryCreateInfoINTEL queryPoolIntel = {
  VK_STRUCTURE_TYPE_QUERY_POOL_CREATE_INFO_INTEL,
  NULL,
  VK_QUERY_POOL_SAMPLING_MODE_MANUAL_INTEL,
};

VkQueryPoolCreateInfo queryPoolCreateInfo = {
  VK_STRUCTURE_TYPE_QUERY_POOL_CREATE_INFO,
  &queryPoolIntel,
  0,
  VK_QUERY_TYPE_PERFORMANCE_QUERY_INTEL,
  1,
  0
};

VkQueryPool queryPool;

VkResult result = vkCreateQueryPool(
  device,
  &queryPoolCreateInfo,
  NULL,
  &queryPool);

assert(VK_SUCCESS == result);

// A command buffer we want to record counters on
VkCommandBuffer commandBuffer;

VkCommandBufferBeginInfo commandBufferBeginInfo = {
  VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO,
  NULL,
  VK_COMMAND_BUFFER_USAGE_ONE_TIME_SUBMIT_BIT,
  NULL
```

```
};

result = vkBeginCommandBuffer(commandBuffer, &commandBufferBeginInfo);

assert(VK_SUCCESS == result);

vkCmdResetQueryPool(
  commandBuffer,
  queryPool,
  0,
  1);

vkCmdBeginQuery(
  commandBuffer,
  queryPool,
  0,
  0);

// Perform the commands you want to get performance information on
// ...

// Perform a barrier to ensure all previous commands were complete before
// ending the query
vkCmdPipelineBarrier(commandBuffer,
  VK_PIPELINE_STAGE_BOTTOM_OF_PIPE_BIT,
  VK_PIPELINE_STAGE_BOTTOM_OF_PIPE_BIT,
  0,
  0,
  NULL,
  0,
  NULL,
  0,
  NULL);

vkCmdEndQuery(
  commandBuffer,
  queryPool,
  0);

result = vkEndCommandBuffer(commandBuffer);

assert(VK_SUCCESS == result);

VkPerformanceConfigurationAcquireInfoINTEL performanceConfigurationAcquireInfo = {
  VK_STRUCTURE_TYPE_PERFORMANCE_CONFIGURATION_ACQUIRE_INFO_INTEL,
  NULL,
  VK_PERFORMANCE_CONFIGURATION_TYPE_COMMAND_QUEUE_METRICS_DISCOVERY_ACTIVATED_INTEL
};

VkPerformanceConfigurationINTEL performanceConfigurationIntel;
```

```
result = vkAcquirePerformanceConfigurationINTEL(
  device,
  &performanceConfigurationAcquireInfo,
  &performanceConfigurationIntel);

vkQueueSetPerformanceConfigurationINTEL(queue, performanceConfigurationIntel);

assert(VK_SUCCESS == result);

// Submit the command buffer and wait for its completion
// ...

result = vkReleasePerformanceConfigurationINTEL(
  device,
  performanceConfigurationIntel);

assert(VK_SUCCESS == result);

// Get the report size from metrics-discovery's QueryReportSize

result = vkGetQueryPoolResults(
  device,
  queryPool,
  0, 1, QueryReportSize,
  data, QueryReportSize, 0);

assert(VK_SUCCESS == result);

// The data can then be passed back to metrics-discovery from which
// human readable values can be queried.
```

### Version History

- Revision 2, 2020-03-06 (Lionel Landwerlin)
  - Rename VkQueryPoolCreateInfoINTEL in VkQueryPoolPerformanceQueryCreateInfoINTEL
- Revision 1, 2018-05-16 (Lionel Landwerlin)
  - Initial revision

## VK_INTEL_shader_integer_functions2

### Name String

`VK_INTEL_shader_integer_functions2`

### Extension Type

Device extension

### Registered Extension Number

210

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### SPIR-V Dependencies

- SPV_INTEL_shader_integer_functions2

### Contact

- Ian Romanick [GitHub]ianromanick

### Other Extension Metadata

### Last Modified Date

2019-04-30

### IP Status

No known IP claims.

### Interactions and External Dependencies

- This extension provides API support for `GL_INTEL_shader_integer_functions2`.

### Contributors

- Ian Romanick, Intel
- Ben Ashbaugh, Intel

### Description

This extension adds support for several new integer instructions in SPIR-V for use in graphics shaders. Many of these instructions have pre-existing counterparts in the Kernel environment.

The added integer functions are defined by the `SPV_INTEL_shader_integer_functions2` SPIR-V extension and can be used with the `GL_INTEL_shader_integer_functions2` GLSL extension.

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceShaderIntegerFunctions2FeaturesINTEL

### New Enum Constants

- `VK_INTEL_SHADER_INTEGER_FUNCTIONS_2_EXTENSION_NAME`
- `VK_INTEL_SHADER_INTEGER_FUNCTIONS_2_SPEC_VERSION`

- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_INTEGER_FUNCTIONS_2_FEATURES_INTEL`

#### New SPIR-V Capabilities

- `IntegerFunctions2INTEL`

#### Version History

- Revision 1, 2019-04-30 (Ian Romanick)
  - Initial draft

## VK_LUNARG_direct_driver_loading

#### Name String

`VK_LUNARG_direct_driver_loading`

#### Extension Type

Instance extension

#### Registered Extension Number

460

#### Revision

1

#### Ratification Status

Not ratified

#### Extension and Version Dependencies

None

#### Contact

- Charles Giessen [GitHub]charles-lunarg

#### Extension Proposal

VK_LUNARG_direct_driver_loading

#### Other Extension Metadata

#### Last Modified Date

2022-11-29

#### Contributors

- Charles Giessen, LunarG

#### Description

This extension provides a mechanism for applications to add drivers to the implementation. This

allows drivers to be included with an application without requiring installation and is capable of being used in any execution environment, such as a process running with elevated privileges.

#### New Structures

- VkDirectDriverLoadingInfoLUNARG
- Extending VkInstanceCreateInfo:
  - VkDirectDriverLoadingListLUNARG

#### New Function Pointers

- PFN_vkGetInstanceProcAddrLUNARG

#### New Enums

- VkDirectDriverLoadingModeLUNARG

#### New Bitmasks

- VkDirectDriverLoadingFlagsLUNARG

#### New Enum Constants

- `VK_LUNARG_DIRECT_DRIVER_LOADING_EXTENSION_NAME`
- `VK_LUNARG_DIRECT_DRIVER_LOADING_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_DIRECT_DRIVER_LOADING_INFO_LUNARG`
  - `VK_STRUCTURE_TYPE_DIRECT_DRIVER_LOADING_LIST_LUNARG`

#### Version History

- Revision 1, 2022-11-29 (Charles Giessen)
  - Initial version

## VK_MESA_image_alignment_control

#### Name String

`VK_MESA_image_alignment_control`

#### Extension Type

Device extension

#### Registered Extension Number

576

#### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### Special Use

- D3D support

### Contact

- Hans-Kristian Arntzen [GitHub]HansKristian-Work

### Other Extension Metadata

### Last Modified Date

2024-05-03

### IP Status

No known IP claims.

### Contributors

- Hans-Kristian Arntzen, Valve

### Description

This extension allows applications to request a narrower alignment for images than an implementation would otherwise require. Some implementations internally support multiple image layouts in `VK_IMAGE_TILING_OPTIMAL`, each with different alignment requirements and performance trade-offs. In some API layering use cases such as D3D12, it is beneficial to be able to control the alignment, since certain alignments for placed resources are guaranteed to be supported, and emulating that expectation requires unnecessary padding of allocations.

VkImageAlignmentControlCreateInfoMESA **can** be chained to VkImageCreateInfo, requesting that the alignment is no more than the provided alignment. If the requested alignment is not supported for a given VkImageCreateInfo, a larger alignment **may** be returned.

While something similar could be achieved with `VK_EXT_image_drm_format_modifier` in theory, this is not the intended way to use that extension. Format modifiers are generally used for externally shareable images, and would not be platform portable. It is also a cumbersome API to use just to lower the alignment.

### New Structures

- Extending VkImageCreateInfo:
  - VkImageAlignmentControlCreateInfoMESA
- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:

    - VkPhysicalDeviceImageAlignmentControlFeaturesMESA
- Extending VkPhysicalDeviceProperties2:
    - VkPhysicalDeviceImageAlignmentControlPropertiesMESA

#### New Enum Constants

- `VK_MESA_IMAGE_ALIGNMENT_CONTROL_EXTENSION_NAME`
- `VK_MESA_IMAGE_ALIGNMENT_CONTROL_SPEC_VERSION`
- Extending VkStructureType:
    - `VK_STRUCTURE_TYPE_IMAGE_ALIGNMENT_CONTROL_CREATE_INFO_MESA`
    - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_IMAGE_ALIGNMENT_CONTROL_FEATURES_MESA`
    - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_IMAGE_ALIGNMENT_CONTROL_PROPERTIES_MESA`

#### Version History

- Revision 1, 2024-04-05 (Hans-Kristian Arntzen)
    - Initial specification

## VK_MSFT_layered_driver

#### Name String

`VK_MSFT_layered_driver`

#### Extension Type

Device extension

#### Registered Extension Number

531

#### Revision

1

#### Ratification Status

Not ratified

#### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

#### Contact

- Jesse Natalie [GitHub]jenatali

#### Extension Proposal

VK_MSFT_layered_driver

### Other Extension Metadata

#### Last Modified Date

2023-06-21

#### IP Status

No known IP claims.

#### Contributors

- Jesse Natalie, Microsoft

#### Description

This extension adds new physical device properties to allow applications and the Vulkan ICD loader to understand when a physical device is implemented as a layered driver on top of another underlying API.

#### New Structures

- Extending VkPhysicalDeviceProperties2:
  - VkPhysicalDeviceLayeredDriverPropertiesMSFT

#### New Enums

- VkLayeredDriverUnderlyingApiMSFT

#### New Enum Constants

- `VK_MSFT_LAYERED_DRIVER_EXTENSION_NAME`
- `VK_MSFT_LAYERED_DRIVER_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_LAYERED_DRIVER_PROPERTIES_MSFT`

#### Examples

None.

#### Version History

- Revision 1, 2023-06-21 (Jesse Natalie)
  - Initial revision

## VK_NN_vi_surface

#### Name String

`VK_NN_vi_surface`

**Extension Type**

Instance extension

**Registered Extension Number**

63

**Revision**

1

**Ratification Status**

Not ratified

**Extension and Version Dependencies**

VK_KHR_surface

**Contact**

- Mathias Heyer [GitLab]mheyer

**Other Extension Metadata**

**Last Modified Date**

2016-12-02

**IP Status**

No known IP claims.

**Contributors**

- Mathias Heyer, NVIDIA
- Michael Chock, NVIDIA
- Yasuhiro Yoshioka, Nintendo
- Daniel Koch, NVIDIA

**Description**

The `VK_NN_vi_surface` extension is an instance extension. It provides a mechanism to create a VkSurfaceKHR object (defined by the `VK_KHR_surface` extension) associated with an `nn::vi::Layer`.

**New Commands**

- vkCreateViSurfaceNN

**New Structures**

- VkViSurfaceCreateInfoNN

**New Bitmasks**

- VkViSurfaceCreateFlagsNN

### New Enum Constants

- `VK_NN_VI_SURFACE_EXTENSION_NAME`
- `VK_NN_VI_SURFACE_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_VI_SURFACE_CREATE_INFO_NN`

### Issues

1) Does VI need a way to query for compatibility between a particular physical device (and queue family?) and a specific VI display?

**RESOLVED**: No. It is currently always assumed that the device and display will always be compatible.

2) VkViSurfaceCreateInfoNN::`pWindow` is intended to store an `nn`::`vi`::`NativeWindowHandle`, but its declared type is a bare `void*` to store the window handle. Why the discrepancy?

**RESOLVED**: It is for C compatibility. The definition for the VI native window handle type is defined inside the `nn`::`vi` C++ namespace. This prevents its use in C source files. `nn`::`vi`::`NativeWindowHandle` is always defined to be `void*`, so this extension uses `void*` to match.

### Version History

- Revision 1, 2016-12-2 (Michael Chock)
  - Initial draft.

## VK_NV_acquire_winrt_display

### Name String

`VK_NV_acquire_winrt_display`

### Extension Type

Device extension

### Registered Extension Number

346

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_EXT_direct_mode_display

### Contact

- Jeff Juliano [GitHub]jjuliano

### Other Extension Metadata

### Last Modified Date

2020-09-29

### IP Status

No known IP claims.

### Contributors

- Jeff Juliano, NVIDIA

### Description

This extension allows an application to take exclusive control of a display on Windows 10 provided that the display is not already controlled by a compositor. Examples of compositors include the Windows desktop compositor, other applications using this Vulkan extension, and applications that "Acquire" a "DisplayTarget" using a "WinRT" command such as "winrt::Windows::Devices::Display::Core::DisplayManager.TryAcquireTarget()".

When control is acquired the application has exclusive access to the display until control is released or the application terminates. An application's attempt to acquire is denied if a different application has already acquired the display.

### New Commands

- vkAcquireWinrtDisplayNV
- vkGetWinrtDisplayNV

### New Enum Constants

- `VK_NV_ACQUIRE_WINRT_DISPLAY_EXTENSION_NAME`
- `VK_NV_ACQUIRE_WINRT_DISPLAY_SPEC_VERSION`

### Issues

1) What should the platform substring be for this extension:

**RESOLVED**: The platform substring is "Winrt".

The substring "Winrt" matches the fact that the OS API exposing the acquire and release functionality is called "WinRT".

The substring "Win32" is wrong because the related "WinRT" API is explicitly **not** a "Win32" API. "WinRT" is a competing API family to the "Win32" API family.

The substring "Windows" is suboptimal because there could be more than one relevant API on the

Windows platform. There is preference to use the more-specific substring “Winrt”.

2) Should vkAcquireWinrtDisplayNV take a winRT DisplayTarget, or a Vulkan display handle as input?

**RESOLVED**: A Vulkan display handle. This matches the design of vkAcquireXlibDisplayEXT.

3) Should the acquire command be platform-independent named “vkAcquireDisplayNV”, or platform-specific named “vkAcquireWinrtDisplayNV”?

**RESOLVED**: Add a platform-specific command.

The inputs to the Acquire command are all Vulkan types. None are WinRT types. This opens the possibility of the winrt extension defining a platform-independent acquire command.

The X11 acquire command does need to accept a platform-specific parameter. This could be handled by adding to a platform-independent acquire command a params struct to which platform-dependent types can be chained by `pNext` pointer.

The prevailing opinion is that it would be odd to create a second platform-independent function that is used on the Windows 10 platform, but that is not used for the X11 platform. Since a Windows 10 platform-specific command is needed anyway for converting between vkDisplayKHR and platform-native handles, opinion was to create a platform-specific acquire function.

4) Should the vkGetWinrtDisplayNV parameter identifying a display be named “deviceRelativeId” or “adapterRelativeId”?

**RESOLVED**: The WinRT name is “AdapterRelativeId”. The name “adapter” is the Windows analog to a Vulkan “physical device”. Vulkan already has precedent to use the name `deviceLUID` for the concept that Windows APIs call “AdapterLuid”. Keeping form with this precedent, the name “deviceRelativeId” is chosen.

5) Does vkAcquireWinrtDisplayNV cause the Windows desktop compositor to release a display?

**RESOLVED**: No. vkAcquireWinrtDisplayNV does not itself cause the Windows desktop compositor to release a display. This action must be performed outside of Vulkan.

Beginning with Windows 10 version 2004 it is possible to cause the Windows desktop compositor to release a display by using the “Advanced display settings” sub-page of the “Display settings” control panel. See https://docs.microsoft.com/en-us/windows-hardware/drivers/display/specialized-monitors

6) Where can one find additional information about custom compositors for Windows 10?

**RESOLVED**: Relevant references are as follows.

According to Microsoft’s documentation on "building a custom compositor", the ability to write a custom compositor is not a replacement for a fullscreen desktop window. The feature is for writing compositor apps that drive specialized hardware.

Only certain editions of Windows 10 support custom compositors, "documented here". The product

type can be queried from Windows 10. See https://docs.microsoft.com/en-us/windows/win32/api/sysinfoapi/nf-sysinfoapi-getproductinfo

#### Version History

- Revision 1, 2020-09-29 (Jeff Juliano)
  - Initial draft

## VK_NV_clip_space_w_scaling

#### Name String

`VK_NV_clip_space_w_scaling`

#### Extension Type

Device extension

#### Registered Extension Number

88

#### Revision

1

#### Ratification Status

Not ratified

#### Extension and Version Dependencies

None

#### Contact

- Eric Werness [GitHub]ewerness-nv

#### Other Extension Metadata

#### Last Modified Date

2017-02-15

#### Contributors

- Eric Werness, NVIDIA
- Kedarnath Thangudu, NVIDIA

#### Description

Virtual Reality (VR) applications often involve a post-processing step to apply a “barrel” distortion to the rendered image to correct the “pincushion” distortion introduced by the optics in a VR device. The barrel distorted image has lower resolution along the edges compared to the center. Since the original image is rendered at high resolution, which is uniform across the complete image, a lot of pixels towards the edges do not make it to the final post-processed image.

This extension provides a mechanism to render VR scenes at a non-uniform resolution, in particular a resolution that falls linearly from the center towards the edges. This is achieved by scaling the w coordinate of the vertices in the clip space before perspective divide. The clip space w coordinate of the vertices **can** be offset as of a function of x and y coordinates as follows:

$w' = w + Ax + By$

In the intended use case for viewport position scaling, an application should use a set of four viewports, one for each of the four quadrants of a Cartesian coordinate system. Each viewport is set to the dimension of the image, but is scissored to the quadrant it represents. The application should specify A and B coefficients of the w-scaling equation above, that have the same value, but different signs, for each of the viewports. The signs of A and B should match the signs of x and y for the quadrant that they represent such that the value of w' will always be greater than or equal to the original w value for the entire image. Since the offset to w, (Ax + By), is always positive, and increases with the absolute values of x and y, the effective resolution will fall off linearly from the center of the image to its edges.

### New Commands

- vkCmdSetViewportWScalingNV

### New Structures

- VkViewportWScalingNV
- Extending VkPipelineViewportStateCreateInfo:
  - VkPipelineViewportWScalingStateCreateInfoNV

### New Enum Constants

- `VK_NV_CLIP_SPACE_W_SCALING_EXTENSION_NAME`
- `VK_NV_CLIP_SPACE_W_SCALING_SPEC_VERSION`
- Extending VkDynamicState:
  - `VK_DYNAMIC_STATE_VIEWPORT_W_SCALING_NV`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PIPELINE_VIEWPORT_W_SCALING_STATE_CREATE_INFO_NV`

### Issues

1) Is the pipeline struct name too long?

**RESOLVED**: It fits with the naming convention.

2) Separate W scaling section or fold into coordinate transformations?

**RESOLVED**: Leaving it as its own section for now.

### Examples

```
VkViewport viewports[4];
VkRect2D scissors[4];
VkViewportWScalingNV scalings[4];

for (int i = 0; i < 4; i++) {
    int x = (i & 2) ? 0 : currentWindowWidth / 2;
    int y = (i & 1) ? 0 : currentWindowHeight / 2;

    viewports[i].x = 0;
    viewports[i].y = 0;
    viewports[i].width = currentWindowWidth;
    viewports[i].height = currentWindowHeight;
    viewports[i].minDepth = 0.0f;
    viewports[i].maxDepth = 1.0f;

    scissors[i].offset.x = x;
    scissors[i].offset.y = y;
    scissors[i].extent.width = currentWindowWidth/2;
    scissors[i].extent.height = currentWindowHeight/2;

    const float factor = 0.15;
    scalings[i].xcoeff = ((i & 2) ? -1.0 : 1.0) * factor;
    scalings[i].ycoeff = ((i & 1) ? -1.0 : 1.0) * factor;
}

VkPipelineViewportWScalingStateCreateInfoNV vpWScalingStateInfo = {
VK_STRUCTURE_TYPE_PIPELINE_VIEWPORT_W_SCALING_STATE_CREATE_INFO_NV };

vpWScalingStateInfo.viewportWScalingEnable = VK_TRUE;
vpWScalingStateInfo.viewportCount = 4;
vpWScalingStateInfo.pViewportWScalings = &scalings[0];

VkPipelineViewportStateCreateInfo vpStateInfo = {
VK_STRUCTURE_TYPE_PIPELINE_VIEWPORT_STATE_CREATE_INFO };
vpStateInfo.viewportCount = 4;
vpStateInfo.pViewports = &viewports[0];
vpStateInfo.scissorCount = 4;
vpStateInfo.pScissors = &scissors[0];
vpStateInfo.pNext = &vpWScalingStateInfo;
```

Example shader to read from a w-scaled texture:

```
// Vertex Shader
// Draw a triangle that covers the whole screen
const vec4 positions[3] = vec4[3](vec4(-1, -1, 0, 1),
                                  vec4( 3, -1, 0, 1),
                                  vec4(-1,  3, 0, 1));
```

```
out vec2 uv;
void main()
{
    vec4 pos = positions[ gl_VertexID ];
    gl_Position = pos;
    uv = pos.xy;
}

// Fragment Shader
uniform sampler2D tex;
uniform float xcoeff;
uniform float ycoeff;
out vec4 Color;
in vec2 uv;

void main()
{
    // Handle uv as if upper right quadrant
    vec2 uvabs = abs(uv);

    // unscale: transform w-scaled image into an unscaled image
    //   scale: transform unscaled image int a w-scaled image
    float unscale = 1.0 / (1 + xcoeff * uvabs.x + xcoeff * uvabs.y);
    //float scale = 1.0 / (1 - xcoeff * uvabs.x - xcoeff * uvabs.y);

    vec2 P = vec2(unscale * uvabs.x, unscale * uvabs.y);

    // Go back to the right quadrant
    P *= sign(uv);

    Color = texture(tex, P * 0.5 + 0.5);
}
```

### Version History

- Revision 1, 2017-02-15 (Eric Werness)
  - Internal revisions

## VK_NV_command_buffer_inheritance

### Name String

`VK_NV_command_buffer_inheritance`

### Extension Type

Device extension

### Registered Extension Number

560

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

None

### Contact

- Lujin Wang [GitHub]lujinwangnv

### Other Extension Metadata

### Last Modified Date

2024-02-15

### IP Status

No known IP claims.

### Contributors

- Piers Daniell, NVIDIA
- Daniel Story, Nintendo

### Description

This extension allows applications to take advantage of the graphics and compute state that remains valid in the queue between executions of submitted command buffers. This works across both primary and secondary command buffers.

The state inherited includes the previously bound pipeline state, previously bound shader objects, previously bound vertex and index buffers, previously bound descriptor sets and push constants, and all previously set dynamic state.

This extension relaxes the requirement that all that state needs to be bound and set after begin command buffer and before the next draw or dispatch.

By not having to set state that has been inherited applications can save both CPU and GPU cycles by not having to set state redundantly, and also have improved flexibility when reusing secondary command buffers.

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceCommandBufferInheritanceFeaturesNV

### New Enum Constants

- `VK_NV_COMMAND_BUFFER_INHERITANCE_EXTENSION_NAME`

- VK_NV_COMMAND_BUFFER_INHERITANCE_SPEC_VERSION
- Extending VkStructureType:
  - VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_COMMAND_BUFFER_INHERITANCE_FEATURES_NV

#### Issues

1) How can the validation layer know when state is valid at draw or dispatch time if it is inherited at execution time?

**RESOLVED**: Validation of invalid state at draw and dispatch time cannot be done while recording those commands. Instead the validation layer will need to keep track of any unset state when draw and dispatch commands are recorded, but not report an error at that time. It should also keep track of what state is valid at the end of each recorded command buffer. When secondary command buffer execution is recorded the validation layer can update its unset state tracking for that command buffer, and also for draw and dispatch commands recorded after execution of the secondary as they will inherit state from the executed secondary. This can be done recursively so every recorded primary command buffer has a final tally of any unset state used at draw and dispatch time. Finally when the primary is submitted to the queue the validation layer will know the previous primaries submitted to the queue and will know if there is any unset state used and can report the error then.

#### Version History

- Revision 1, 2024-02-15 (Lujin Wang)
  - Internal revisions

## VK_NV_cooperative_matrix

#### Name String

VK_NV_cooperative_matrix

#### Extension Type

Device extension

#### Registered Extension Number

250

#### Revision

1

#### Ratification Status

Not ratified

#### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### SPIR-V Dependencies

- SPV_NV_cooperative_matrix

### Contact

- Jeff Bolz [GitHub]jeffbolznv

### Other Extension Metadata

### Last Modified Date

2019-02-05

### Interactions and External Dependencies

- This extension provides API support for `GL_NV_cooperative_matrix`

### Contributors

- Jeff Bolz, NVIDIA
- Markus Tavenrath, NVIDIA
- Daniel Koch, NVIDIA

### Description

This extension adds support for using cooperative matrix types in SPIR-V. Cooperative matrix types are medium-sized matrices that are primarily supported in compute shaders, where the storage for the matrix is spread across all invocations in some scope (usually a subgroup) and those invocations cooperate to efficiently perform matrix multiplies.

Cooperative matrix types are defined by the `SPV_NV_cooperative_matrix` SPIR-V extension and can be used with the `GL_NV_cooperative_matrix` GLSL extension.

This extension includes support for enumerating the matrix types and dimensions that are supported by the implementation.

### New Commands

- vkGetPhysicalDeviceCooperativeMatrixPropertiesNV

### New Structures

- VkCooperativeMatrixPropertiesNV
- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceCooperativeMatrixFeaturesNV
- Extending VkPhysicalDeviceProperties2:
  - VkPhysicalDeviceCooperativeMatrixPropertiesNV

### New Enums

- VkComponentTypeNV
- VkScopeNV

### New Enum Constants

- `VK_NV_COOPERATIVE_MATRIX_EXTENSION_NAME`
- `VK_NV_COOPERATIVE_MATRIX_SPEC_VERSION`
- Extending VkComponentTypeKHR:
  - `VK_COMPONENT_TYPE_FLOAT16_NV`
  - `VK_COMPONENT_TYPE_FLOAT32_NV`
  - `VK_COMPONENT_TYPE_FLOAT64_NV`
  - `VK_COMPONENT_TYPE_SINT16_NV`
  - `VK_COMPONENT_TYPE_SINT32_NV`
  - `VK_COMPONENT_TYPE_SINT64_NV`
  - `VK_COMPONENT_TYPE_SINT8_NV`
  - `VK_COMPONENT_TYPE_UINT16_NV`
  - `VK_COMPONENT_TYPE_UINT32_NV`
  - `VK_COMPONENT_TYPE_UINT64_NV`
  - `VK_COMPONENT_TYPE_UINT8_NV`
- Extending VkScopeKHR:
  - `VK_SCOPE_DEVICE_NV`
  - `VK_SCOPE_QUEUE_FAMILY_NV`
  - `VK_SCOPE_SUBGROUP_NV`
  - `VK_SCOPE_WORKGROUP_NV`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_COOPERATIVE_MATRIX_PROPERTIES_NV`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_COOPERATIVE_MATRIX_FEATURES_NV`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_COOPERATIVE_MATRIX_PROPERTIES_NV`

### New SPIR-V Capabilities

- `CooperativeMatrixNV`

### Issues

(1) What matrix properties will be supported in practice?

**RESOLVED**: In NVIDIA's initial implementation, we will support:

- AType = BType = fp16 CType = DType = fp16 MxNxK = 16x8x16 scope = Subgroup
- AType = BType = fp16 CType = DType = fp16 MxNxK = 16x8x8 scope = Subgroup
- AType = BType = fp16 CType = DType = fp32 MxNxK = 16x8x16 scope = Subgroup
- AType = BType = fp16 CType = DType = fp32 MxNxK = 16x8x8 scope = Subgroup

### Version History

- Revision 1, 2019-02-05 (Jeff Bolz)
  - Internal revisions

## VK_NV_copy_memory_indirect

### Name String

`VK_NV_copy_memory_indirect`

### Extension Type

Device extension

### Registered Extension Number

427

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1
and
VK_KHR_buffer_device_address
or
Vulkan Version 1.2

### Contact

- Vikram Kushwaha [GitHub]vkushwaha-nv

### Other Extension Metadata

### Last Modified Date

2022-10-14

### Contributors

- Vikram Kushwaha, NVIDIA
- Jeff Bolz, NVIDIA
- Christoph Kubisch, NVIDIA
- Daniel Koch, NVIDIA

### Description

This extension adds support for performing copies between memory and image regions using indirect parameters that are read by the device from a buffer during execution. This functionality **may** be useful for performing copies where the copy parameters are not known during the command buffer creation time.

### New Commands

- vkCmdCopyMemoryIndirectNV
- vkCmdCopyMemoryToImageIndirectNV

### New Structures

- VkCopyMemoryIndirectCommandNV
- VkCopyMemoryToImageIndirectCommandNV
- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceCopyMemoryIndirectFeaturesNV
- Extending VkPhysicalDeviceProperties2:
  - VkPhysicalDeviceCopyMemoryIndirectPropertiesNV

### New Enum Constants

- `VK_NV_COPY_MEMORY_INDIRECT_EXTENSION_NAME`
- `VK_NV_COPY_MEMORY_INDIRECT_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_COPY_MEMORY_INDIRECT_FEATURES_NV`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_COPY_MEMORY_INDIRECT_PROPERTIES_NV`

### Version History

- Revision 1, 2022-10-14 (Vikram Kushwaha)
  - Initial draft

## VK_NV_corner_sampled_image

### Name String

`VK_NV_corner_sampled_image`

### Extension Type

Device extension

### Registered Extension Number

51

### Revision

2

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### Contact

- Daniel Koch [GitHub]dgkoch

### Other Extension Metadata

### Last Modified Date

2018-08-13

### Contributors

- Jeff Bolz, NVIDIA
- Pat Brown, NVIDIA
- Chris Lentini, NVIDIA

### Description

This extension adds support for a new image organization, which this extension refers to as “corner-sampled” images. A corner-sampled image differs from a conventional image in the following ways:

- Texels are centered on integer coordinates. See Unnormalized Texel Coordinate Operations
- Normalized coordinates are scaled using coord × (dim - 1) rather than coord × dim, where dim is the size of one dimension of the image. See normalized texel coordinate transform.
- Partial derivatives are scaled using coord × (dim - 1) rather than coord × dim. See Scale Factor Operation.
- Calculation of the next higher LOD size goes according to ⌊dim / 2⌋ rather than ⌈dim / 2⌉. See Image Mip Level Sizing.
- The minimum level size is 2x2 for 2D images and 2x2x2 for 3D images. See Image Mip Level Sizing.

This image organization is designed to facilitate a system like Ptex with separate textures for each face of a subdivision or polygon mesh. Placing sample locations at pixel corners allows applications to maintain continuity between adjacent patches by duplicating values along shared edges. Additionally, using the modified mipmapping logic along with texture dimensions of the form $2^n$+1 allows continuity across shared edges even if the adjacent patches use different LOD values.

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceCornerSampledImageFeaturesNV

### New Enum Constants

- `VK_NV_CORNER_SAMPLED_IMAGE_EXTENSION_NAME`
- `VK_NV_CORNER_SAMPLED_IMAGE_SPEC_VERSION`
- Extending VkImageCreateFlagBits:
  - `VK_IMAGE_CREATE_CORNER_SAMPLED_BIT_NV`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_CORNER_SAMPLED_IMAGE_FEATURES_NV`

### Issues

1. What should this extension be named?

   **DISCUSSION**: While naming this extension, we chose the most distinctive aspect of the image organization and referred to such images as "corner-sampled images". As a result, we decided to name the extension NV_corner_sampled_image.

2. Do we need a format feature flag so formats can advertise if they support corner-sampling?

   **DISCUSSION**: Currently NVIDIA supports this for all 2D and 3D formats, but not for cube maps or depth-stencil formats. A format feature might be useful if other vendors would only support this on some formats.

3. Do integer texel coordinates have a different range for corner-sampled images?

   **RESOLVED**: No, these are unchanged.

4. Do unnormalized sampler coordinates work with corner-sampled images? Are there any functional differences?

   **RESOLVED**: Yes. Unnormalized coordinates are treated as already scaled for corner-sample usage.

5. Should we have a diagram in the "Image Operations" chapter demonstrating different texel sampling locations?

   **UNRESOLVED**: Probably, but later.

### Version History

- Revision 1, 2018-08-14 (Daniel Koch)
  - Internal revisions
- Revision 2, 2018-08-14 (Daniel Koch)

- ???

## VK_NV_coverage_reduction_mode

### Name String

`VK_NV_coverage_reduction_mode`

### Extension Type

Device extension

### Registered Extension Number

251

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_NV_framebuffer_mixed_samples
and
  VK_KHR_get_physical_device_properties2
  or
  Vulkan Version 1.1

### Contact

- Kedarnath Thangudu [GitHub]kthangudu

### Other Extension Metadata

### Last Modified Date

2019-01-29

### Contributors

- Kedarnath Thangudu, NVIDIA
- Jeff Bolz, NVIDIA

### Description

When using a framebuffer with mixed samples, a per-fragment coverage reduction operation is performed which generates color sample coverage from the pixel coverage. This extension defines the following modes to control how this reduction is performed.

- Merge: When there are more samples in the pixel coverage than color samples, there is an implementation-dependent association of each pixel coverage sample to a color sample. In the merge mode, the color sample coverage is computed such that only if any associated sample in the pixel coverage is covered, the color sample is covered. This is the default mode.

- Truncate: When there are more raster samples (N) than color samples(M), there is one to one association of the first M raster samples to the M color samples; other raster samples are ignored.

When the number of raster samples is equal to the color samples, there is a one to one mapping between them in either of the above modes.

The new command vkGetPhysicalDeviceSupportedFramebufferMixedSamplesCombinationsNV can be used to query the various raster, color, depth/stencil sample count and reduction mode combinations that are supported by the implementation. This extension would allow an implementation to support the behavior of both `VK_NV_framebuffer_mixed_samples` and `VK_AMD_mixed_attachment_samples` extensions simultaneously.

### New Commands

- vkGetPhysicalDeviceSupportedFramebufferMixedSamplesCombinationsNV

### New Structures

- VkFramebufferMixedSamplesCombinationNV
- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceCoverageReductionModeFeaturesNV
- Extending VkPipelineMultisampleStateCreateInfo:
  - VkPipelineCoverageReductionStateCreateInfoNV

### New Enums

- VkCoverageReductionModeNV

### New Bitmasks

- VkPipelineCoverageReductionStateCreateFlagsNV

### New Enum Constants

- `VK_NV_COVERAGE_REDUCTION_MODE_EXTENSION_NAME`
- `VK_NV_COVERAGE_REDUCTION_MODE_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_FRAMEBUFFER_MIXED_SAMPLES_COMBINATION_NV`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_COVERAGE_REDUCTION_MODE_FEATURES_NV`
  - `VK_STRUCTURE_TYPE_PIPELINE_COVERAGE_REDUCTION_STATE_CREATE_INFO_NV`

### Version History

- Revision 1, 2019-01-29 (Kedarnath Thangudu)
  - Internal revisions

## VK_NV_dedicated_allocation_image_aliasing

### Name String

`VK_NV_dedicated_allocation_image_aliasing`

### Extension Type

Device extension

### Registered Extension Number

241

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_dedicated_allocation
and
VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### Contact

- Nuno Subtil [GitHub]nsubtil

### Other Extension Metadata

### Last Modified Date

2019-01-04

### Contributors

- Nuno Subtil, NVIDIA
- Jeff Bolz, NVIDIA
- Eric Werness, NVIDIA
- Axel Gneiting, id Software

### Description

This extension allows applications to alias images on dedicated allocations, subject to specific restrictions: the extent and the number of layers in the image being aliased must be smaller than or equal to those of the original image for which the allocation was created, and every other image parameter must match.

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceDedicatedAllocationImageAliasingFeaturesNV

### New Enum Constants

- `VK_NV_DEDICATED_ALLOCATION_IMAGE_ALIASING_EXTENSION_NAME`
- `VK_NV_DEDICATED_ALLOCATION_IMAGE_ALIASING_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DEDICATED_ALLOCATION_IMAGE_ALIASING_FEATURES_NV`

### Version History

- Revision 1, 2019-01-04 (Nuno Subtil)
  - Internal revisions

## VK_NV_descriptor_pool_overallocation

### Name String

`VK_NV_descriptor_pool_overallocation`

### Extension Type

Device extension

### Registered Extension Number

547

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

Vulkan Version 1.1

### Contact

- Piers Daniell [GitHub]pdaniell-nv

### Other Extension Metadata

### Last Modified Date

2023-08-30

### Contributors

- Jeff Bolz, NVIDIA

### Description

There are scenarios where the application does not know ahead of time how many descriptor sets it may need to allocate from a descriptor pool, or how many descriptors of any of the descriptor types it may need to allocate from the descriptor pool.

This extension gives applications the ability to request the implementation allow more sets or descriptors to be allocated than initially specified at descriptor pool creation time, subject to available resources.

The `VK_DESCRIPTOR_POOL_CREATE_ALLOW_OVERALLOCATION_SETS_BIT_NV` flag lets the application allocate more than VkDescriptorPoolCreateInfo::`maxSets` descriptor sets, and the `VK_DESCRIPTOR_POOL_CREATE_ALLOW_OVERALLOCATION_POOLS_BIT_NV` lets the application allocate more descriptors than initially specified by VkDescriptorPoolSize::`descriptorCount` for any descriptor types.

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceDescriptorPoolOverallocationFeaturesNV

### New Enum Constants

- `VK_NV_DESCRIPTOR_POOL_OVERALLOCATION_EXTENSION_NAME`
- `VK_NV_DESCRIPTOR_POOL_OVERALLOCATION_SPEC_VERSION`
- Extending VkDescriptorPoolCreateFlagBits:
  - `VK_DESCRIPTOR_POOL_CREATE_ALLOW_OVERALLOCATION_POOLS_BIT_NV`
  - `VK_DESCRIPTOR_POOL_CREATE_ALLOW_OVERALLOCATION_SETS_BIT_NV`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DESCRIPTOR_POOL_OVERALLOCATION_FEATURES_NV`

### Version History

- Revision 1, 2023-08-30 (Piers Daniell)
  - Internal revisions

## VK_NV_device_diagnostic_checkpoints

### Name String

`VK_NV_device_diagnostic_checkpoints`

### Extension Type

Device extension

### Registered Extension Number

207

### Revision

2

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### Contact

- Nuno Subtil [GitHub]nsubtil

### Other Extension Metadata

### Last Modified Date

2018-07-16

### Contributors

- Oleg Kuznetsov, NVIDIA
- Alex Dunn, NVIDIA
- Jeff Bolz, NVIDIA
- Eric Werness, NVIDIA
- Daniel Koch, NVIDIA

### Description

This extension allows applications to insert markers in the command stream and associate them with custom data.

If a device lost error occurs, the application **may** then query the implementation for the last markers to cross specific implementation-defined pipeline stages, in order to narrow down which commands were executing at the time and might have caused the failure.

### New Commands

- vkCmdSetCheckpointNV
- vkGetQueueCheckpointDataNV

### New Structures

- VkCheckpointDataNV
- Extending VkQueueFamilyProperties2:
  - VkQueueFamilyCheckpointPropertiesNV

### New Enum Constants

- `VK_NV_DEVICE_DIAGNOSTIC_CHECKPOINTS_EXTENSION_NAME`
- `VK_NV_DEVICE_DIAGNOSTIC_CHECKPOINTS_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_CHECKPOINT_DATA_NV`
  - `VK_STRUCTURE_TYPE_QUEUE_FAMILY_CHECKPOINT_PROPERTIES_NV`

### Version History

- Revision 1, 2018-07-16 (Nuno Subtil)
  - Internal revisions
- Revision 2, 2018-07-16 (Nuno Subtil)
  - ???

## VK_NV_device_diagnostics_config

### Name String

`VK_NV_device_diagnostics_config`

### Extension Type

Device extension

### Registered Extension Number

301

### Revision

2

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### Contact

- Kedarnath Thangudu [GitHub]kthangudu

### Other Extension Metadata

### Last Modified Date

2022-04-06

### Contributors

- Kedarnath Thangudu, NVIDIA
- Thomas Klein, NVIDIA

### Description

Applications using Nvidia Nsight™ Aftermath SDK for Vulkan to integrate device crash dumps into their error reporting mechanisms, **may** use this extension to configure options related to device crash dump creation.

Version 2 of this extension adds `VK_DEVICE_DIAGNOSTICS_CONFIG_ENABLE_SHADER_ERROR_REPORTING_BIT_NV` which when set enables enhanced reporting of shader execution errors.

### New Structures

- Extending VkDeviceCreateInfo:
  - VkDeviceDiagnosticsConfigCreateInfoNV
- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceDiagnosticsConfigFeaturesNV

### New Enums

- VkDeviceDiagnosticsConfigFlagBitsNV

### New Bitmasks

- VkDeviceDiagnosticsConfigFlagsNV

### New Enum Constants

- `VK_NV_DEVICE_DIAGNOSTICS_CONFIG_EXTENSION_NAME`
- `VK_NV_DEVICE_DIAGNOSTICS_CONFIG_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_DEVICE_DIAGNOSTICS_CONFIG_CREATE_INFO_NV`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DIAGNOSTICS_CONFIG_FEATURES_NV`

### Version History

- Revision 1, 2019-11-21 (Kedarnath Thangudu)
  - Internal revisions
- Revision 2, 2022-04-06 (Kedarnath Thangudu)
  - Added a config bit `VK_DEVICE_DIAGNOSTICS_CONFIG_ENABLE_SHADER_ERROR_REPORTING_BIT_NV`

## VK_NV_device_generated_commands

### Name String

`VK_NV_device_generated_commands`

### Extension Type

Device extension

### Registered Extension Number

278

### Revision

3

### Ratification Status

Not ratified

### Extension and Version Dependencies

Vulkan Version 1.1
and
VK_KHR_buffer_device_address
or
Vulkan Version 1.2

### Contact

- Christoph Kubisch [GitHub]pixeljetstream

### Other Extension Metadata

### Last Modified Date

2020-02-20

### Interactions and External Dependencies

- This extension requires Vulkan 1.1
- This extension requires `VK_EXT_buffer_device_address` or `VK_KHR_buffer_device_address` or Vulkan 1.2 for the ability to bind vertex and index buffers on the device.
- This extension interacts with `VK_NV_mesh_shader`. If the latter extension is not supported, remove the command token to initiate mesh tasks drawing in this extension.

### Contributors

- Christoph Kubisch, NVIDIA
- Pierre Boudier, NVIDIA
- Jeff Bolz, NVIDIA
- Eric Werness, NVIDIA
- Yuriy O'Donnell, Epic Games
- Baldur Karlsson, Valve

- Mathias Schott, NVIDIA
- Tyson Smith, NVIDIA
- Ingo Esser, NVIDIA

**Description**

This extension allows the device to generate a number of critical graphics commands for command buffers.

When rendering a large number of objects, the device can be leveraged to implement a number of critical functions, like updating matrices, or implementing occlusion culling, frustum culling, front to back sorting, etc. Implementing those on the device does not require any special extension, since an application is free to define its own data structures, and just process them using shaders.

However, if the application desires to quickly kick off the rendering of the final stream of objects, then unextended Vulkan forces the application to read back the processed stream and issue graphics command from the host. For very large scenes, the synchronization overhead and cost to generate the command buffer can become the bottleneck. This extension allows an application to generate a device side stream of state changes and commands, and convert it efficiently into a command buffer without having to read it back to the host.

Furthermore, it allows incremental changes to such command buffers by manipulating only partial sections of a command stream — for example pipeline bindings. Unextended Vulkan requires re-creation of entire command buffers in such a scenario, or updates synchronized on the host.

The intended usage for this extension is for the application to:

- create `VkBuffer` objects and retrieve physical addresses from them via vkGetBufferDeviceAddressEXT
- create a graphics pipeline using `VkGraphicsPipelineShaderGroupsCreateInfoNV` for the ability to change shaders on the device.
- create a VkIndirectCommandsLayoutNV, which lists the VkIndirectCommandsTokenTypeNV it wants to dynamically execute as an atomic command sequence. This step likely involves some internal device code compilation, since the intent is for the GPU to generate the command buffer in the pipeline.
- fill the input stream buffers with the data for each of the inputs it needs. Each input is an array that will be filled with token-dependent data.
- set up a preprocess `VkBuffer` that uses memory according to the information retrieved via vkGetGeneratedCommandsMemoryRequirementsNV.
- optionally preprocess the generated content using vkCmdPreprocessGeneratedCommandsNV, for example on an asynchronous compute queue, or for the purpose of re-using the data in multiple executions.
- call vkCmdExecuteGeneratedCommandsNV to create and execute the actual device commands for all sequences based on the inputs provided.

For each draw in a sequence, the following can be specified:

- a different shader group
- a number of vertex buffer bindings
- a different index buffer, with an optional dynamic offset and index type
- a number of different push constants
- a flag that encodes the primitive winding

While the GPU can be faster than a CPU to generate the commands, it will not happen asynchronously to the device, therefore the primary use case is generating “less” total work (occlusion culling, classification to use specialized shaders, etc.).

#### New Object Types

- VkIndirectCommandsLayoutNV

#### New Commands

- vkCmdBindPipelineShaderGroupNV
- vkCmdExecuteGeneratedCommandsNV
- vkCmdPreprocessGeneratedCommandsNV
- vkCreateIndirectCommandsLayoutNV
- vkDestroyIndirectCommandsLayoutNV
- vkGetGeneratedCommandsMemoryRequirementsNV

#### New Structures

- VkBindIndexBufferIndirectCommandNV
- VkBindShaderGroupIndirectCommandNV
- VkBindVertexBufferIndirectCommandNV
- VkGeneratedCommandsInfoNV
- VkGeneratedCommandsMemoryRequirementsInfoNV
- VkGraphicsShaderGroupCreateInfoNV
- VkIndirectCommandsLayoutCreateInfoNV
- VkIndirectCommandsLayoutTokenNV
- VkIndirectCommandsStreamNV
- VkSetStateFlagsIndirectCommandNV
- Extending VkGraphicsPipelineCreateInfo:
  - VkGraphicsPipelineShaderGroupsCreateInfoNV
- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceDeviceGeneratedCommandsFeaturesNV
- Extending VkPhysicalDeviceProperties2:

- VkPhysicalDeviceDeviceGeneratedCommandsPropertiesNV

### New Enums

- VkIndirectCommandsLayoutUsageFlagBitsNV
- VkIndirectCommandsTokenTypeNV
- VkIndirectStateFlagBitsNV

### New Bitmasks

- VkIndirectCommandsLayoutUsageFlagsNV
- VkIndirectStateFlagsNV

### New Enum Constants

- `VK_NV_DEVICE_GENERATED_COMMANDS_EXTENSION_NAME`
- `VK_NV_DEVICE_GENERATED_COMMANDS_SPEC_VERSION`
- Extending VkAccessFlagBits:
  - `VK_ACCESS_COMMAND_PREPROCESS_READ_BIT_NV`
  - `VK_ACCESS_COMMAND_PREPROCESS_WRITE_BIT_NV`
- Extending VkObjectType:
  - `VK_OBJECT_TYPE_INDIRECT_COMMANDS_LAYOUT_NV`
- Extending VkPipelineCreateFlagBits:
  - `VK_PIPELINE_CREATE_INDIRECT_BINDABLE_BIT_NV`
- Extending VkPipelineStageFlagBits:
  - `VK_PIPELINE_STAGE_COMMAND_PREPROCESS_BIT_NV`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_GENERATED_COMMANDS_INFO_NV`
  - `VK_STRUCTURE_TYPE_GENERATED_COMMANDS_MEMORY_REQUIREMENTS_INFO_NV`
  - `VK_STRUCTURE_TYPE_GRAPHICS_PIPELINE_SHADER_GROUPS_CREATE_INFO_NV`
  - `VK_STRUCTURE_TYPE_GRAPHICS_SHADER_GROUP_CREATE_INFO_NV`
  - `VK_STRUCTURE_TYPE_INDIRECT_COMMANDS_LAYOUT_CREATE_INFO_NV`
  - `VK_STRUCTURE_TYPE_INDIRECT_COMMANDS_LAYOUT_TOKEN_NV`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DEVICE_GENERATED_COMMANDS_FEATURES_NV`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DEVICE_GENERATED_COMMANDS_PROPERTIES_NV`

### Issues

1) How to name this extension ?

`VK_NV_device_generated_commands`

As usual, one of the hardest issues ;)

Alternatives: `VK_gpu_commands`, `VK_execute_commands`, `VK_device_commands`, `VK_device_execute_commands`, `VK_device_execute`, `VK_device_created_commands`, `VK_device_recorded_commands`, `VK_device_generated_commands` `VK_indirect_generated_commands`

2) Should we use a serial stateful token stream or stateless sequence descriptions?

Similarly to VkPipeline, fixed layouts have the most likelihood to be cross-vendor adoptable. They also benefit from being processable in parallel. This is a different design choice compared to the serial command stream generated through `GL_NV_command_list`.

3) How to name a sequence description?

`VkIndirectCommandsLayout` as in the NVX extension predecessor.

Alternative: `VkGeneratedCommandsLayout`

4) Do we want to provide `indirectCommands` inputs with layout or at `indirectCommands` time?

Separate layout from data as Vulkan does. Provide full flexibility for `indirectCommands`.

5) Should the input be provided as SoA or AoS?

Both ways are desirable. AoS can provide portability to other APIs and easier to setup, while SoA allows to update individual inputs in a cache-efficient manner, when others remain static.

6) How do we make developers aware of the memory requirements of implementation-dependent data used for the generated commands?

Make the API explicit and introduce a `preprocess` VkBuffer. Developers have to allocate it using vkGetGeneratedCommandsMemoryRequirementsNV.

In the NVX version the requirements were hidden implicitly as part of the command buffer reservation process, however as the memory requirements can be substantial, we want to give developers the ability to budget the memory themselves. By lowering the `maxSequencesCount` the memory consumption can be reduced. Furthermore reuse of the memory is possible, for example for doing explicit preprocessing and execution in a ping-pong fashion.

The actual buffer size is implementation-dependent and may be zero, i.e. not always required.

When making use of Graphics Shader Groups, the programs should behave similar with regards to vertex inputs, clipping and culling outputs of the geometry stage, as well as sample shading behavior in fragment shaders, to reduce the amount of the worst-case memory approximation.

7) Should we allow additional per-sequence dynamic state changes?

Yes

Introduced a lightweight indirect state flag VkIndirectStateFlagBitsNV. So far only switching front face winding state is exposed. Especially in CAD/DCC mirrored transforms that require such changes are common, and similar flexibility is given in the ray tracing instance description.

The flag could be extended further, for example to switch between primitive-lists or -strips, or make other state modifications.

Furthermore, as new tokens can be added easily, future extension could add the ability to change any VkDynamicState.

8) How do we allow re-using already “generated” `indirectCommands`?

Expose a `preprocessBuffer` to reuse implementation-dependencyFlags data. Set the `isPreprocessed` to true in vkCmdExecuteGeneratedCommandsNV.

9) Under which conditions is vkCmdExecuteGeneratedCommandsNV legal?

It behaves like a regular draw call command.

10) Is vkCmdPreprocessGeneratedCommandsNV copying the input data or referencing it?

There are multiple implementations possible:

- one could have some emulation code that parses the inputs, and generates an output command buffer, therefore copying the inputs.
- one could just reference the inputs, and have the processing done in pipe at execution time.

If the data is mandated to be copied, then it puts a penalty on implementation that could process the inputs directly in pipe. If the data is “referenced”, then it allows both types of implementation.

The inputs are “referenced”, and **must** not be modified after the call to vkCmdExecuteGeneratedCommandsNV has completed.

11) Which buffer usage flags are required for the buffers referenced by `VkGeneratedCommandsInfoNV` ?

Reuse existing `VK_BUFFER_USAGE_INDIRECT_BUFFER_BIT`

- VkGeneratedCommandsInfoNV::`preprocessBuffer`
- VkGeneratedCommandsInfoNV::`sequencesCountBuffer`
- VkGeneratedCommandsInfoNV::`sequencesIndexBuffer`
- VkIndirectCommandsStreamNV::`buffer`

12) In which pipeline stage does the device generated command expansion happen?

vkCmdPreprocessGeneratedCommandsNV is treated as if it occurs in a separate logical pipeline from either graphics or compute, and that pipeline only includes `VK_PIPELINE_STAGE_TOP_OF_PIPE_BIT`, a new stage `VK_PIPELINE_STAGE_COMMAND_PREPROCESS_BIT_NV`, and `VK_PIPELINE_STAGE_BOTTOM_OF_PIPE_BIT`. This new stage has two corresponding new access types, `VK_ACCESS_COMMAND_PREPROCESS_READ_BIT_NV` and `VK_ACCESS_COMMAND_PREPROCESS_WRITE_BIT_NV`, used to synchronize reading the buffer inputs and writing the preprocess memory output.

The generated output written in the preprocess buffer memory by vkCmdExecuteGeneratedCommandsNV is considered to be consumed by the `VK_PIPELINE_STAGE_DRAW_INDIRECT_BIT` pipeline stage.

Thus, to synchronize from writing the input buffers to preprocessing via vkCmdPreprocessGeneratedCommandsNV, use:

- `dstStageMask` = `VK_PIPELINE_STAGE_COMMAND_PREPROCESS_BIT_NV`
- `dstAccessMask` = `VK_ACCESS_COMMAND_PREPROCESS_READ_BIT_NV`

To synchronize from vkCmdPreprocessGeneratedCommandsNV to executing the generated commands by vkCmdExecuteGeneratedCommandsNV, use:

- `srcStageMask` = `VK_PIPELINE_STAGE_COMMAND_PREPROCESS_BIT_NV`
- `srcAccessMask` = `VK_ACCESS_COMMAND_PREPROCESS_WRITE_BIT_NV`
- `dstStageMask` = `VK_PIPELINE_STAGE_DRAW_INDIRECT_BIT`
- `dstAccessMask` = `VK_ACCESS_INDIRECT_COMMAND_READ_BIT`

When vkCmdExecuteGeneratedCommandsNV is used with a `isPreprocessed` of `VK_FALSE`, the generated commands are implicitly preprocessed, therefore one only needs to synchronize the inputs via:

- `dstStageMask` = `VK_PIPELINE_STAGE_DRAW_INDIRECT_BIT`
- `dstAccessMask` = `VK_ACCESS_INDIRECT_COMMAND_READ_BIT`

13) What if most token data is “static”, but we frequently want to render a subsection?

Added “sequencesIndexBuffer”. This allows to easier sort and filter what should actually be executed.

14) What are the changes compared to the previous NVX extension?

- Compute dispatch support was removed (was never implemented in drivers). There are different approaches how dispatching from the device should work, hence we defer this to a future extension.
- The `ObjectTableNVX` was replaced by using physical buffer addresses and introducing Shader Groups for the graphics pipeline.
- Less state changes are possible overall, but the important operations are still there (reduces complexity of implementation).
- The API was redesigned so all inputs must be passed at both preprocessing and execution time (this was implicit in NVX, now it is explicit)
- The reservation of intermediate command space is now mandatory and explicit through a preprocess buffer.
- The VkIndirectStateFlagBitsNV were introduced

15) When porting from other APIs, their indirect buffers may use different enums, for example for index buffer types. How to solve this?

Added “pIndexTypeValues” to map custom `uint32_t` values to corresponding `VkIndexType`.

16) Do we need more shader group state overrides?

The NVX version allowed all PSO states to be different, however as the goal is not to replace all state setup, but focus on highly-frequent state changes for drawing lots of objects, we reduced the amount of state overrides. Especially VkPipelineLayout as well as VkRenderPass configuration should be left static, the rest is still open for discussion.

The current focus is just to allow VertexInput changes as well as shaders, while all shader groups use the same shader stages.

Too much flexibility will increase the test coverage requirement as well. However, further extensions could allow more dynamic state as well.

17) Do we need more detailed physical device feature queries/enables?

An EXT version would need detailed implementor feedback to come up with a good set of features. Please contact us if you are interested, we are happy to make more features optional, or add further restrictions to reduce the minimum feature set of an EXT.

18) Is there an interaction with VK_KHR_pipeline_library planned?

Yes, a future version of this extension will detail the interaction, once VK_KHR_pipeline_library is no longer provisional.

#### Example Code

Open-Source samples illustrating the usage of the extension can be found at the following location (may not yet exist at time of writing):

https://github.com/nvpro-samples/vk_device_generated_cmds

#### Version History

- Revision 1, 2020-02-20 (Christoph Kubisch)
  - Initial version
- Revision 2, 2020-03-09 (Christoph Kubisch)
  - Remove VK_EXT_debug_report interactions
- Revision 3, 2020-03-09 (Christoph Kubisch)
  - Fix naming VkPhysicalDeviceGenerated to VkPhysicalDeviceDeviceGenerated

## VK_NV_device_generated_commands_compute

#### Name String

`VK_NV_device_generated_commands_compute`

#### Extension Type

Device extension

#### Registered Extension Number

429

### Revision

2

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_NV_device_generated_commands

### Contact

- Vikram Kushwaha [GitHub]vkushwaha-nv

### Other Extension Metadata

### Last Modified Date

2023-07-21

### Contributors

- Vikram Kushwaha, NVIDIA
- Jeff Bolz, NVIDIA
- Christoph Kubisch, NVIDIA
- Piers Daniell, NVIDIA
- Daniel Koch, NVIDIA
- Hans-Kristian Arntzen, Valve
- Mike Blumenkrantz, VALVE

### Description

This extension allows the device to generate commands for binding compute pipelines, setting push constants and launching compute dispatches.

### New Commands

- vkCmdUpdatePipelineIndirectBufferNV
- vkGetPipelineIndirectDeviceAddressNV
- vkGetPipelineIndirectMemoryRequirementsNV

### New Structures

- VkBindPipelineIndirectCommandNV
- VkPipelineIndirectDeviceAddressInfoNV
- Extending VkComputePipelineCreateInfo:
  - VkComputePipelineIndirectBufferInfoNV
- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:

- VkPhysicalDeviceDeviceGeneratedCommandsComputeFeaturesNV

#### New Enum Constants

- `VK_NV_DEVICE_GENERATED_COMMANDS_COMPUTE_EXTENSION_NAME`
- `VK_NV_DEVICE_GENERATED_COMMANDS_COMPUTE_SPEC_VERSION`
- Extending VkDescriptorSetLayoutCreateFlagBits:
  - `VK_DESCRIPTOR_SET_LAYOUT_CREATE_INDIRECT_BINDABLE_BIT_NV`
- Extending VkIndirectCommandsTokenTypeNV:
  - `VK_INDIRECT_COMMANDS_TOKEN_TYPE_DISPATCH_NV`
  - `VK_INDIRECT_COMMANDS_TOKEN_TYPE_PIPELINE_NV`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_COMPUTE_PIPELINE_INDIRECT_BUFFER_INFO_NV`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DEVICE_GENERATED_COMMANDS_COMPUTE_FEATURES_NV`
  - `VK_STRUCTURE_TYPE_PIPELINE_INDIRECT_DEVICE_ADDRESS_INFO_NV`

#### Version History

- Revision 2, 2023-07-21 (Vikram Kushwaha)
  - Rename vkCmdUpdatePipelineIndirectBuffer to vkCmdUpdatePipelineIndirectBufferNV
- Revision 1, 2023-06-09 (Vikram Kushwaha)
  - First Revision

## VK_NV_extended_sparse_address_space

#### Name String

`VK_NV_extended_sparse_address_space`

#### Extension Type

Device extension

#### Registered Extension Number

493

#### Revision

1

#### Ratification Status

Not ratified

#### Extension and Version Dependencies

None

### Contact

- Russell Chou [GitHub]russellcnv

### Other Extension Metadata

### Last Modified Date

2023-10-03

### Contributors

- Russell Chou, NVIDIA
- Christoph Kubisch, NVIDIA
- Eric Werness, NVIDIA
- Jeff Bolz, NVIDIA

### Description

Implementations may be able to support an extended address space for sparse memory resources, but only for a certain set of usages.

This extension adds a query for the extended limit, and the supported usages that are allowed for that limit. This limit is an increase to VkPhysicalDeviceLimits::`sparseAddressSpaceSize` when the VkImage or VkBuffer uses only usages that are supported.

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceExtendedSparseAddressSpaceFeaturesNV
- Extending VkPhysicalDeviceProperties2:
  - VkPhysicalDeviceExtendedSparseAddressSpacePropertiesNV

### New Enum Constants

- `VK_NV_EXTENDED_SPARSE_ADDRESS_SPACE_EXTENSION_NAME`
- `VK_NV_EXTENDED_SPARSE_ADDRESS_SPACE_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTENDED_SPARSE_ADDRESS_SPACE_FEATURES_NV`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTENDED_SPARSE_ADDRESS_SPACE_PROPERTIES_NV`

### Version History

- Revision 1, 2023-10-03 (Russell Chou)
  - Initial draft

## VK_NV_external_memory_rdma

### Name String

`VK_NV_external_memory_rdma`

### Extension Type

Device extension

### Registered Extension Number

372

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_external_memory
or
Vulkan Version 1.1

### Contact

- Carsten Rohde [GitHub]crohde

### Other Extension Metadata

### Last Modified Date

2021-04-19

### IP Status

No known IP claims.

### Contributors

- Carsten Rohde, NVIDIA

### Description

This extension adds support for allocating memory which can be used for remote direct memory access (RDMA) from other devices.

### New Base Types

- VkRemoteAddressNV

### New Commands

- vkGetMemoryRemoteAddressNV

### New Structures

- VkMemoryGetRemoteAddressInfoNV
- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceExternalMemoryRDMAFeaturesNV

### New Enum Constants

- VK_NV_EXTERNAL_MEMORY_RDMA_EXTENSION_NAME
- VK_NV_EXTERNAL_MEMORY_RDMA_SPEC_VERSION
- Extending VkExternalMemoryHandleTypeFlagBits:
  - VK_EXTERNAL_MEMORY_HANDLE_TYPE_RDMA_ADDRESS_BIT_NV
- Extending VkMemoryPropertyFlagBits:
  - VK_MEMORY_PROPERTY_RDMA_CAPABLE_BIT_NV
- Extending VkStructureType:
  - VK_STRUCTURE_TYPE_MEMORY_GET_REMOTE_ADDRESS_INFO_NV
  - VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTERNAL_MEMORY_RDMA_FEATURES_NV

### Issues

### Examples

```
VkPhysicalDeviceMemoryBudgetPropertiesEXT memoryBudgetProperties = {
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MEMORY_BUDGET_PROPERTIES_EXT };
VkPhysicalDeviceMemoryProperties2 memoryProperties2 = {
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MEMORY_PROPERTIES_2, &memoryBudgetProperties };
vkGetPhysicalDeviceMemoryProperties2(physicalDevice, &memoryProperties2);
uint32_t heapIndex = (uint32_t)-1;
for (uint32_t memoryType = 0; memoryType < memoryProperties2.memoryProperties
.memoryTypeCount; memoryType++) {
    if (memoryProperties2.memoryProperties.memoryTypes[memoryType].propertyFlags &
VK_MEMORY_PROPERTY_RDMA_CAPABLE_BIT_NV) {
        heapIndex = memoryProperties2.memoryProperties.memoryTypes[memoryType
].heapIndex;
        break;
    }
}
if ((heapIndex == (uint32_t)-1) ||
    (memoryBudgetProperties.heapBudget[heapIndex] < size)) {
    return;
}

VkPhysicalDeviceExternalBufferInfo externalBufferInfo = {
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTERNAL_BUFFER_INFO };
externalBufferInfo.usage = VK_BUFFER_USAGE_TRANSFER_SRC_BIT |
VK_BUFFER_USAGE_TRANSFER_DST_BIT;
```

```
externalBufferInfo.handleType = VK_EXTERNAL_MEMORY_HANDLE_TYPE_RDMA_ADDRESS_BIT_NV;

VkExternalBufferProperties externalBufferProperties = {
VK_STRUCTURE_TYPE_EXTERNAL_BUFFER_PROPERTIES };
vkGetPhysicalDeviceExternalBufferProperties(physicalDevice, &externalBufferInfo,
&externalBufferProperties);

if (!(externalBufferProperties.externalMemoryProperties.externalMemoryFeatures &
VK_EXTERNAL_MEMORY_FEATURE_EXPORTABLE_BIT)) {
    return;
}

VkExternalMemoryBufferCreateInfo externalMemoryBufferCreateInfo = {
VK_STRUCTURE_TYPE_EXTERNAL_MEMORY_BUFFER_CREATE_INFO };
externalMemoryBufferCreateInfo.handleTypes =
VK_EXTERNAL_MEMORY_HANDLE_TYPE_RDMA_ADDRESS_BIT_NV;

VkBufferCreateInfo bufferCreateInfo = { VK_STRUCTURE_TYPE_BUFFER_CREATE_INFO,
&externalMemoryBufferCreateInfo };
bufferCreateInfo.size = size;
bufferCreateInfo.usage = VK_BUFFER_USAGE_TRANSFER_SRC_BIT |
VK_BUFFER_USAGE_TRANSFER_DST_BIT;

VkMemoryRequirements mem_reqs;
vkCreateBuffer(device, &bufferCreateInfo, NULL, &buffer);
vkGetBufferMemoryRequirements(device, buffer, &mem_reqs);

VkExportMemoryAllocateInfo exportMemoryAllocateInfo = {
VK_STRUCTURE_TYPE_EXPORT_MEMORY_ALLOCATE_INFO };
exportMemoryAllocateInfo.handleTypes =
VK_EXTERNAL_MEMORY_HANDLE_TYPE_RDMA_ADDRESS_BIT_NV;

// Find memory type index
uint32_t i = 0;
for (; i < VK_MAX_MEMORY_TYPES; i++) {
    if ((mem_reqs.memoryTypeBits & (1 << i)) &&
        (memoryProperties.memoryTypes[i].propertyFlags &
VK_MEMORY_PROPERTY_RDMA_CAPABLE_BIT_NV)) {
        break;
    }
}

VkMemoryAllocateInfo memAllocInfo = { VK_STRUCTURE_TYPE_MEMORY_ALLOCATE_INFO,
&exportMemoryAllocateInfo };
memAllocInfo.allocationSize = mem_reqs.size;
memAllocInfo.memoryTypeIndex = i;

vkAllocateMemory(device, &memAllocInfo, NULL, &mem);
vkBindBufferMemory(device, buffer, mem, 0);

VkMemoryGetRemoteAddressInfoNV getMemoryRemoteAddressInfo = {
```

```
VK_STRUCTURE_TYPE_MEMORY_GET_REMOTE_ADDRESS_INFO_NV };
getMemoryRemoteAddressInfo.memory = mem;
getMemoryRemoteAddressInfo.handleType =
VK_EXTERNAL_MEMORY_HANDLE_TYPE_RDMA_ADDRESS_BIT_NV;

VkRemoteAddressNV rdmaAddress;
vkGetMemoryRemoteAddressNV(device, &getMemoryRemoteAddressInfo, &rdmaAddress);
// address returned in 'rdmaAddress' can be used by external devices to initiate RDMA
transfers
```

#### Version History

- Revision 1, 2020-12-15 (Carsten Rohde)
  - Internal revisions

## VK_NV_fill_rectangle

#### Name String

`VK_NV_fill_rectangle`

#### Extension Type

Device extension

#### Registered Extension Number

154

#### Revision

1

#### Ratification Status

Not ratified

#### Extension and Version Dependencies

None

#### Contact

- Jeff Bolz [GitHub]jeffbolznv

#### Other Extension Metadata

#### Last Modified Date

2017-05-22

#### Contributors

- Jeff Bolz, NVIDIA

### Description

This extension adds a new VkPolygonMode `enum` where a triangle is rasterized by computing and filling its axis-aligned screen-space bounding box, disregarding the actual triangle edges. This can be useful for drawing a rectangle without being split into two triangles with an internal edge. It is also useful to minimize the number of primitives that need to be drawn, particularly for a user interface.

### New Enum Constants

- `VK_NV_FILL_RECTANGLE_EXTENSION_NAME`
- `VK_NV_FILL_RECTANGLE_SPEC_VERSION`
- Extending VkPolygonMode:
  - `VK_POLYGON_MODE_FILL_RECTANGLE_NV`

### Version History

- Revision 1, 2017-05-22 (Jeff Bolz)
  - Internal revisions

## VK_NV_fragment_coverage_to_color

### Name String

`VK_NV_fragment_coverage_to_color`

### Extension Type

Device extension

### Registered Extension Number

150

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

None

### Contact

- Jeff Bolz [GitHub]jeffbolznv

### Other Extension Metadata

### Last Modified Date

2017-05-21

### Contributors

- Jeff Bolz, NVIDIA

### Description

This extension allows the fragment coverage value, represented as an integer bitmask, to be substituted for a color output being written to a single-component color attachment with integer components (e.g. `VK_FORMAT_R8_UINT`). The functionality provided by this extension is different from simply writing the `SampleMask` fragment shader output, in that the coverage value written to the framebuffer is taken after stencil test and depth test, as well as after fragment operations such as alpha-to-coverage.

This functionality may be useful for deferred rendering algorithms, where the second pass needs to know which samples belong to which original fragments.

### New Structures

- Extending VkPipelineMultisampleStateCreateInfo:
  - VkPipelineCoverageToColorStateCreateInfoNV

### New Bitmasks

- VkPipelineCoverageToColorStateCreateFlagsNV

### New Enum Constants

- `VK_NV_FRAGMENT_COVERAGE_TO_COLOR_EXTENSION_NAME`
- `VK_NV_FRAGMENT_COVERAGE_TO_COLOR_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PIPELINE_COVERAGE_TO_COLOR_STATE_CREATE_INFO_NV`

### Version History

- Revision 1, 2017-05-21 (Jeff Bolz)
  - Internal revisions

## VK_NV_fragment_shading_rate_enums

### Name String

`VK_NV_fragment_shading_rate_enums`

### Extension Type

Device extension

### Registered Extension Number

327

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_fragment_shading_rate

### Contact

- Pat Brown [GitHub]nvpbrown

### Other Extension Metadata

### Last Modified Date

2020-09-02

### Contributors

- Pat Brown, NVIDIA
- Jeff Bolz, NVIDIA

### Description

This extension builds on the fragment shading rate functionality provided by the VK_KHR_fragment_shading_rate extension, adding support for “supersample” fragment shading rates that trigger multiple fragment shader invocations per pixel as well as a “no invocations” shading rate that discards any portions of a primitive that would use that shading rate.

### New Commands

- vkCmdSetFragmentShadingRateEnumNV

### New Structures

- Extending VkGraphicsPipelineCreateInfo:
  - VkPipelineFragmentShadingRateEnumStateCreateInfoNV
- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceFragmentShadingRateEnumsFeaturesNV
- Extending VkPhysicalDeviceProperties2:
  - VkPhysicalDeviceFragmentShadingRateEnumsPropertiesNV

### New Enums

- VkFragmentShadingRateNV
- VkFragmentShadingRateTypeNV

### New Enum Constants

- `VK_NV_FRAGMENT_SHADING_RATE_ENUMS_EXTENSION_NAME`
- `VK_NV_FRAGMENT_SHADING_RATE_ENUMS_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_FRAGMENT_SHADING_RATE_ENUMS_FEATURES_NV`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_FRAGMENT_SHADING_RATE_ENUMS_PROPERTIES_NV`
  - `VK_STRUCTURE_TYPE_PIPELINE_FRAGMENT_SHADING_RATE_ENUM_STATE_CREATE_INFO_NV`

### Issues

1. Why was this extension created? How should it be named?

   **RESOLVED**: The primary goal of this extension was to expose support for supersample and "no invocations" shading rates, which are supported by the VK_NV_shading_rate_image extension but not by VK_KHR_fragment_shading_rate. Because VK_KHR_fragment_shading_rate specifies the primitive shading rate using a fragment size in pixels, it lacks a good way to specify supersample rates. To deal with this, we defined enums covering shading rates supported by the KHR extension as well as the new shading rates and added structures and APIs accepting shading rate enums instead of fragment sizes.

   Since this extension adds two different types of shading rates, both expressed using enums, we chose the extension name VK_NV_fragment_shading_rate_enums.

2. Is this a standalone extension?

   **RESOLVED**: No, this extension requires VK_KHR_fragment_shading_rate. In order to use the features of this extension, applications must enable the relevant features of KHR extension.

3. How are the shading rate enums used, and how were the enum values assigned?

   **RESOLVED**: The shading rates supported by the enums in this extension are accepted as pipeline, primitive, and attachment shading rates and behave identically. For the shading rates also supported by the KHR extension, the values assigned to the corresponding enums are identical to the values already used for the primitive and attachment shading rates in the KHR extension. For those enums, bits 0 and 1 specify the base two logarithm of the fragment height and bits 2 and 3 specify the base two logarithm of the fragment width. For the new shading rates added by this extension, we chose to use 11 through 14 (10 plus the base two logarithm of the invocation count) for the supersample rates and 15 for the "no invocations" rate. None of those values are supported as primitive or attachment shading rates by the KHR extension.

4. Between this extension, VK_KHR_fragment_shading_rate, and VK_NV_shading_rate_image, there are three different ways to specify shading rate state in a pipeline. How should we handle this?

   **RESOLVED**: We do not allow the concurrent use of VK_NV_shading_rate_image and VK_KHR_fragment_shading_rate; it is an error to enable shading rate features from both extensions. But we do allow applications to enable this extension together with VK_KHR_fragment_shading_rate together. While we expect that applications will never attach

pipeline CreateInfo structures for both this extension and the KHR extension concurrently, Vulkan does not have any precedent forbidding such behavior and instead typically treats a pipeline created without an extension-specific CreateInfo structure as equivalent to one containing default values specified by the extension. Rather than adding such a rule considering the presence or absence of our new CreateInfo structure, we instead included a `shadingRateType` member to VkPipelineFragmentShadingRateEnumStateCreateInfoNV that selects between using state specified by that structure and state specified by VkPipelineFragmentShadingRateStateCreateInfoKHR.

#### Version History

- Revision 1, 2020-09-02 (pbrown)
  - Internal revisions

### VK_NV_framebuffer_mixed_samples

#### Name String

`VK_NV_framebuffer_mixed_samples`

#### Extension Type

Device extension

#### Registered Extension Number

153

#### Revision

1

#### Ratification Status

Not ratified

#### Extension and Version Dependencies

None

#### Contact

- Jeff Bolz [GitHub]jeffbolznv

#### Other Extension Metadata

#### Last Modified Date

2017-06-04

#### Contributors

- Jeff Bolz, NVIDIA

#### Description

This extension allows multisample rendering with a raster and depth/stencil sample count that is

larger than the color sample count. Rasterization and the results of the depth and stencil tests together determine the portion of a pixel that is “covered”. It can be useful to evaluate coverage at a higher frequency than color samples are stored. This coverage is then “reduced” to a collection of covered color samples, each having an opacity value corresponding to the fraction of the color sample covered. The opacity can optionally be blended into individual color samples.

Rendering with fewer color samples than depth/stencil samples greatly reduces the amount of memory and bandwidth consumed by the color buffer. However, converting the coverage values into opacity introduces artifacts where triangles share edges and **may** not be suitable for normal triangle mesh rendering.

One expected use case for this functionality is Stencil-then-Cover path rendering (similar to the OpenGL GL_NV_path_rendering extension). The stencil step determines the coverage (in the stencil buffer) for an entire path at the higher sample frequency, and then the cover step draws the path into the lower frequency color buffer using the coverage information to antialias path edges. With this two-step process, internal edges are fully covered when antialiasing is applied and there is no corruption on these edges.

The key features of this extension are:

- It allows render pass and framebuffer objects to be created where the number of samples in the depth/stencil attachment in a subpass is a multiple of the number of samples in the color attachments in the subpass.
- A coverage reduction step is added to Fragment Operations which converts a set of covered raster/depth/stencil samples to a set of color samples that perform blending and color writes. The coverage reduction step also includes an optional coverage modulation step, multiplying color values by a fractional opacity corresponding to the number of associated raster/depth/stencil samples covered.

#### New Structures

- Extending VkPipelineMultisampleStateCreateInfo:
  - VkPipelineCoverageModulationStateCreateInfoNV

#### New Enums

- VkCoverageModulationModeNV

#### New Bitmasks

- VkPipelineCoverageModulationStateCreateFlagsNV

#### New Enum Constants

- `VK_NV_FRAMEBUFFER_MIXED_SAMPLES_EXTENSION_NAME`
- `VK_NV_FRAMEBUFFER_MIXED_SAMPLES_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PIPELINE_COVERAGE_MODULATION_STATE_CREATE_INFO_NV`

### Version History

- Revision 1, 2017-06-04 (Jeff Bolz)
  - Internal revisions

## VK_NV_geometry_shader_passthrough

### Name String

`VK_NV_geometry_shader_passthrough`

### Extension Type

Device extension

### Registered Extension Number

96

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

None

### SPIR-V Dependencies

- SPV_NV_geometry_shader_passthrough

### Contact

- Daniel Koch [GitHub]dgkoch

### Other Extension Metadata

### Last Modified Date

2017-02-15

### Interactions and External Dependencies

- This extension provides API support for `GL_NV_geometry_shader_passthrough`
- This extension requires the `geometryShader` feature.

### Contributors

- Piers Daniell, NVIDIA
- Jeff Bolz, NVIDIA

### Description

This extension adds support for the following SPIR-V extension in Vulkan:

- `SPV_NV_geometry_shader_passthrough`

Geometry shaders provide the ability for applications to process each primitive sent through the graphics pipeline using a programmable shader. However, one common use case treats them largely as a “passthrough”. In this use case, the bulk of the geometry shader code simply copies inputs from each vertex of the input primitive to corresponding outputs in the vertices of the output primitive. Such shaders might also compute values for additional built-in or user-defined per-primitive attributes (e.g., `Layer`) to be assigned to all the vertices of the output primitive.

This extension provides access to the `PassthroughNV` decoration under the `GeometryShaderPassthroughNV` capability. Adding this to a geometry shader input variable specifies that the values of this input are copied to the corresponding vertex of the output primitive.

When using GLSL source-based shading languages, the `passthrough` layout qualifier from `GL_NV_geometry_shader_passthrough` maps to the `PassthroughNV` decoration. To use the `passthrough` layout, in GLSL the `GL_NV_geometry_shader_passthrough` extension must be enabled. Behavior is described in the `GL_NV_geometry_shader_passthrough` extension specification.

### New Enum Constants

- `VK_NV_GEOMETRY_SHADER_PASSTHROUGH_EXTENSION_NAME`
- `VK_NV_GEOMETRY_SHADER_PASSTHROUGH_SPEC_VERSION`

### New Variable Decoration

- `PassthroughNV` in Geometry Shader Passthrough

### New SPIR-V Capabilities

- `GeometryShaderPassthroughNV`

### Issues

1) Should we require or allow a passthrough geometry shader to specify the output layout qualifiers for the output primitive type and maximum vertex count in the SPIR-V?

**RESOLVED**: Yes they should be required in the SPIR-V. Per GL_NV_geometry_shader_passthrough they are not permitted in the GLSL source shader, but SPIR-V is lower-level. It is straightforward for the GLSL compiler to infer them from the input primitive type and to explicitly emit them in the SPIR-V according to the following table.

| Input Layout | Implied Output Layout |
|---|---|
| points | `layout(points, max_vertices=1)` |
| lines | `layout(line_strip, max_vertices=2)` |
| triangles | `layout(triangle_strip, max_vertices=3)` |

2) How does interface matching work with passthrough geometry shaders?

**RESOLVED**: This is described in Passthrough Interface Matching. In GL when using passthrough geometry shaders in separable mode, all inputs must also be explicitly assigned location layout

qualifiers. In Vulkan all SPIR-V shader inputs (except built-ins) must also have location decorations specified. Redeclarations of built-in variables that add the passthrough layout qualifier are exempted from the rule requiring location assignment because built-in variables do not have locations and are matched by `BuiltIn` decoration.

**Sample Code**

Consider the following simple geometry shader in unextended GLSL:

```
layout(triangles) in;
layout(triangle_strip) out;
layout(max_vertices=3) out;

in Inputs {
    vec2 texcoord;
    vec4 baseColor;
} v_in[];
out Outputs {
    vec2 texcoord;
    vec4 baseColor;
};

void main()
{
    int layer = compute_layer();
    for (int i = 0; i < 3; i++) {
        gl_Position = gl_in[i].gl_Position;
        texcoord = v_in[i].texcoord;
        baseColor = v_in[i].baseColor;
        gl_Layer = layer;
        EmitVertex();
    }
}
```

In this shader, the inputs `gl_Position`, `Inputs.texcoord`, and `Inputs.baseColor` are simply copied from the input vertex to the corresponding output vertex. The only “interesting” work done by the geometry shader is computing and emitting a `gl_Layer` value for the primitive.

The following geometry shader, using this extension, is equivalent:

```
#extension GL_NV_geometry_shader_passthrough : require

layout(triangles) in;
// No output primitive layout qualifiers required.

// Redeclare gl_PerVertex to pass through "gl_Position".
layout(passthrough) in gl_PerVertex {
    vec4 gl_Position;
} gl_in[];
```

```
// Declare "Inputs" with "passthrough" to automatically copy members.
layout(passthrough) in Inputs {
    vec2 texcoord;
    vec4 baseColor;
} v_in[];

// No output block declaration required.

void main()
{
    // The shader simply computes and writes gl_Layer.  We do not
    // loop over three vertices or call EmitVertex().
    gl_Layer = compute_layer();
}
```

#### Version History

- Revision 1, 2017-02-15 (Daniel Koch)
  - Internal revisions

## VK_NV_inherited_viewport_scissor

#### Name String

`VK_NV_inherited_viewport_scissor`

#### Extension Type

Device extension

#### Registered Extension Number

279

#### Revision

1

#### Ratification Status

Not ratified

#### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

#### Contact

- David Zhao Akeley [GitHub]akeley98

### Other Extension Metadata

#### Last Modified Date

2021-02-04

#### Contributors

- David Zhao Akeley, NVIDIA
- Jeff Bolz, NVIDIA
- Piers Daniell, NVIDIA
- Christoph Kubisch, NVIDIA

### Description

This extension adds the ability for a secondary command buffer to inherit the dynamic viewport and scissor state from a primary command buffer, or a previous secondary command buffer executed within the same vkCmdExecuteCommands call. It addresses a frequent scenario in applications that deal with window resizing and want to improve utilization of reusable secondary command buffers. The functionality is provided through VkCommandBufferInheritanceViewportScissorInfoNV. Viewport inheritance is effectively limited to the 2D rectangle; secondary command buffers must re-specify the inherited depth range values.

### New Structures

- Extending VkCommandBufferInheritanceInfo:
  - VkCommandBufferInheritanceViewportScissorInfoNV
- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceInheritedViewportScissorFeaturesNV

### New Enum Constants

- `VK_NV_INHERITED_VIEWPORT_SCISSOR_EXTENSION_NAME`
- `VK_NV_INHERITED_VIEWPORT_SCISSOR_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_COMMAND_BUFFER_INHERITANCE_VIEWPORT_SCISSOR_INFO_NV`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_INHERITED_VIEWPORT_SCISSOR_FEATURES_NV`

### Issues

(1) Why are viewport depth values configured in the VkCommandBufferInheritanceViewportScissorInfoNV struct, rather than by a `vkCmd⋯` function?

**DISCUSSION**:

We considered both adding a new `vkCmdSetViewportDepthNV` function, and modifying vkCmdSetViewport to ignore the `x`, `y`, `width`, and `height` values when called with a secondary command buffer that activates this extension.

The primary design considerations for this extension are debuggability and easy integration into existing applications. The main issue with adding a new `vkCmdSetViewportDepthNV` function is reducing ease-of-integration. A new function pointer will have to be loaded, but more importantly, a new function would require changes to be supported in graphics debuggers; this would delay widespread adoption of the extension.

The proposal to modify vkCmdSetViewport would avoid these issues. However, we expect that the intent of applications using this extension is to have the viewport values used for drawing exactly match the inherited values; thus, it would be better for debuggability if no function for modifying the viewport depth alone is provided. By specifying viewport depth values when starting secondary command buffer recording, and requiring the specified depth values to match the inherited depth values, we allow for validation layers that flag depth changes as errors.

This design also better matches the hardware model. In fact, there is no need to re-execute a depth-setting command. The graphics device retains the viewport depth state; it is the CPU-side state of VkCommandBuffer that must be re-initialized.

(2) Why are viewport depth values specified as a partial VkViewport struct, rather than a leaner depth-only struct?

**DISCUSSION**:

We considered adding a new `VkViewportDepthNV` struct containing only `minDepth` and `maxDepth`. However, as application developers would need to maintain both a `VK_NV_inherited_viewport_scissor` code path and a fallback code path (at least in the short term), we ultimately chose to continue using the existing VkViewport structure. Doing so would allow application developers to reuse the same VkViewport array for both code paths, rather than constructing separate `VkViewportDepthNV` and VkViewport arrays for each code path.

### Version History

- Revision 1, 2020-02-04 (David Zhao Akeley)
  - Internal revisions

## VK_NV_linear_color_attachment

### Name String

`VK_NV_linear_color_attachment`

### Extension Type

Device extension

### Registered Extension Number

431

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### API Interactions

- Interacts with VK_VERSION_1_3
- Interacts with VK_KHR_format_feature_flags2

### Contact

- sourav parmar [GitHub]souravpNV

### Other Extension Metadata

### Last Modified Date

2021-12-02

### Interactions and External Dependencies

- This extension requires `VK_KHR_format_feature_flags2`

### Contributors

- Pat Brown, NVIDIA
- Piers Daniell, NVIDIA
- Sourav Parmar, NVIDIA

### Description

This extension expands support for using `VK_IMAGE_TILING_LINEAR` images as color attachments when all the color attachments in the render pass instance have `VK_IMAGE_TILING_LINEAR` tiling. This extension adds a new flag bit `VK_FORMAT_FEATURE_2_LINEAR_COLOR_ATTACHMENT_BIT_NV` that extends the existing VkFormatFeatureFlagBits2KHR bits. This flag **can** be set for renderable color formats in the VkFormatProperties3KHR::`linearTilingFeatures` format properties structure member. Formats with the `VK_FORMAT_FEATURE_2_LINEAR_COLOR_ATTACHMENT_BIT_NV` flag **may** be used as color attachments as long as all the color attachments in the render pass instance have `VK_IMAGE_TILING_LINEAR` tiling, and the formats their images views are created with have VkFormatProperties3KHR::`linearTilingFeatures` which include `VK_FORMAT_FEATURE_2_LINEAR_COLOR_ATTACHMENT_BIT_NV`. This extension supports both dynamic rendering and traditional render passes.

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceLinearColorAttachmentFeaturesNV

### New Enum Constants

- `VK_NV_LINEAR_COLOR_ATTACHMENT_EXTENSION_NAME`
- `VK_NV_LINEAR_COLOR_ATTACHMENT_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_LINEAR_COLOR_ATTACHMENT_FEATURES_NV`

If VK_KHR_format_feature_flags2 or Vulkan Version 1.3 is supported:

- Extending VkFormatFeatureFlagBits2:
  - `VK_FORMAT_FEATURE_2_LINEAR_COLOR_ATTACHMENT_BIT_NV`

### Version History

- Revision 1, 2021-11-29 (sourav parmar)
  - Initial draft

## VK_NV_low_latency

### Name String

`VK_NV_low_latency`

### Extension Type

Device extension

### Registered Extension Number

311

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

None

### Contact

- Charles Hansen [GitHub]cshansen

### Other Extension Metadata

### Last Modified Date

2023-02-10

### Contributors

- Charles Hansen, NVIDIA

### Description

This extension adds the VkQueryLowLatencySupportNV structure, a structure used to query support for NVIDIA Reflex.

### New Structures

- Extending VkSemaphoreCreateInfo:
  - VkQueryLowLatencySupportNV

### New Enum Constants

- `VK_NV_LOW_LATENCY_EXTENSION_NAME`
- `VK_NV_LOW_LATENCY_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_QUERY_LOW_LATENCY_SUPPORT_NV`

### Issues

1) Why does `VkQueryLowLatencySupportNV` have output parameters in an input chain?

**RESOLVED**: We are stuck with this for legacy reasons - we are aware this is bad behavior and this should not be used as a precedent for future extensions.

### Version History

- Revision 1, 2023-02-10 (Charles Hansen)
  - Internal revisions

## VK_NV_low_latency2

### Name String

`VK_NV_low_latency2`

### Extension Type

Device extension

### Registered Extension Number

506

### Revision

2

### Ratification Status

Not ratified

### Extension and Version Dependencies

Vulkan Version 1.2

or
VK_KHR_timeline_semaphore

### Contact

- Charles Hansen [GitHub]cshansen

### Other Extension Metadata

### Last Modified Date

2023-09-25

### Contributors

- Charles Hansen, NVIDIA
- Liam Middlebrook, NVIDIA
- Lionel Duc, NVIDIA
- James Jones, NVIDIA
- Eric Sullivan, NVIDIA

### New Commands

- vkGetLatencyTimingsNV
- vkLatencySleepNV
- vkQueueNotifyOutOfBandNV
- vkSetLatencyMarkerNV
- vkSetLatencySleepModeNV

### New Structures

- VkGetLatencyMarkerInfoNV
- VkLatencySleepInfoNV
- VkLatencySleepModeInfoNV
- VkLatencyTimingsFrameReportNV
- VkOutOfBandQueueTypeInfoNV
- VkSetLatencyMarkerInfoNV
- Extending VkSubmitInfo, VkSubmitInfo2:
  - VkLatencySubmissionPresentIdNV
- Extending VkSurfaceCapabilities2KHR:
  - VkLatencySurfaceCapabilitiesNV
- Extending VkSwapchainCreateInfoKHR:
  - VkSwapchainLatencyCreateInfoNV

### New Enums

- VkLatencyMarkerNV
- VkOutOfBandQueueTypeNV

### New Enum Constants

- `VK_NV_LOW_LATENCY_2_EXTENSION_NAME`
- `VK_NV_LOW_LATENCY_2_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_GET_LATENCY_MARKER_INFO_NV`
  - `VK_STRUCTURE_TYPE_LATENCY_SLEEP_INFO_NV`
  - `VK_STRUCTURE_TYPE_LATENCY_SLEEP_MODE_INFO_NV`
  - `VK_STRUCTURE_TYPE_LATENCY_SUBMISSION_PRESENT_ID_NV`
  - `VK_STRUCTURE_TYPE_LATENCY_SURFACE_CAPABILITIES_NV`
  - `VK_STRUCTURE_TYPE_LATENCY_TIMINGS_FRAME_REPORT_NV`
  - `VK_STRUCTURE_TYPE_OUT_OF_BAND_QUEUE_TYPE_INFO_NV`
  - `VK_STRUCTURE_TYPE_SET_LATENCY_MARKER_INFO_NV`
  - `VK_STRUCTURE_TYPE_SWAPCHAIN_LATENCY_CREATE_INFO_NV`

### Description

This extension gives applications timing suggestions on when to start the recording of new frames to reduce the latency between input sampling and frame presentation. Applications can accomplish this through the extension by calling vkSetLatencySleepModeNV to allow the driver to pace a given swapchain, then calling vkLatencySleepNV before input sampling to delay the start of the CPU side work. Additional methods and structures are provided to give insight into the latency pipeline of an application through the latency markers. `VK_NV_low_latency` provides legacy support for applications that make use of the NVIDIA Reflex SDK whereas new implementations should use the `VK_NV_low_latency2` extension.

### Issues

1) How does Low Latency 2 work with applications that utilize device groups?

Low Latency 2 does not support device groups.

### Version History

- Revision 2, 2023-11-15 (Charles Hansen)
  - Update vkGetLatencyTimingsNV. This is a breaking API change which brings behavior in line with other array querying commands. More background can be found in https://github.com/KhronosGroup/Vulkan-Docs/issues/2269
- Revision 1, 2023-09-25 (Charles Hansen)

- Internal revisions

## VK_NV_memory_decompression

### Name String

`VK_NV_memory_decompression`

### Extension Type

Device extension

### Registered Extension Number

428

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1
and
VK_KHR_buffer_device_address
or
Vulkan Version 1.2

### Contact

- Vikram Kushwaha [GitHub]vkushwaha-nv

### Other Extension Metadata

### Last Modified Date

2022-01-31

### Contributors

- Vikram Kushwaha, NVIDIA
- Jeff Bolz, NVIDIA
- Christoph Kubisch, NVIDIA
- Piers Daniell, NVIDIA

### Description

This extension adds support for performing memory to memory decompression.

### New Commands

- vkCmdDecompressMemoryIndirectCountNV
- vkCmdDecompressMemoryNV

### New Structures

- VkDecompressMemoryRegionNV
- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceMemoryDecompressionFeaturesNV
- Extending VkPhysicalDeviceProperties2:
  - VkPhysicalDeviceMemoryDecompressionPropertiesNV

### New Enums

- VkMemoryDecompressionMethodFlagBitsNV

### New Bitmasks

- VkMemoryDecompressionMethodFlagsNV

### New Enum Constants

- `VK_NV_MEMORY_DECOMPRESSION_EXTENSION_NAME`
- `VK_NV_MEMORY_DECOMPRESSION_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MEMORY_DECOMPRESSION_FEATURES_NV`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MEMORY_DECOMPRESSION_PROPERTIES_NV`

### Version History

- Revision 1, 2022-01-31 (Vikram Kushwaha)
  - Initial draft

## VK_NV_mesh_shader

### Name String

`VK_NV_mesh_shader`

### Extension Type

Device extension

### Registered Extension Number

203

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### SPIR-V Dependencies

- SPV_NV_mesh_shader

### Contact

- Christoph Kubisch [GitHub]pixeljetstream

### Other Extension Metadata

### Last Modified Date

2018-07-19

### Interactions and External Dependencies

- This extension provides API support for `GLSL_NV_mesh_shader`

### Contributors

- Pat Brown, NVIDIA
- Jeff Bolz, NVIDIA
- Daniel Koch, NVIDIA
- Piers Daniell, NVIDIA
- Pierre Boudier, NVIDIA

### Description

This extension provides a new mechanism allowing applications to generate collections of geometric primitives via programmable mesh shading. It is an alternative to the existing programmable primitive shading pipeline, which relied on generating input primitives by a fixed function assembler as well as fixed function vertex fetch.

There are new programmable shader types — the task and mesh shader — to generate these collections to be processed by fixed-function primitive assembly and rasterization logic. When task and mesh shaders are dispatched, they replace the core pre-rasterization stages, including vertex array attribute fetching, vertex shader processing, tessellation, and geometry shader processing.

This extension also adds support for the following SPIR-V extension in Vulkan:

- `SPV_NV_mesh_shader`

### New Commands

- vkCmdDrawMeshTasksIndirectCountNV
- vkCmdDrawMeshTasksIndirectNV
- vkCmdDrawMeshTasksNV

### New Structures

- VkDrawMeshTasksIndirectCommandNV
- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceMeshShaderFeaturesNV
- Extending VkPhysicalDeviceProperties2:
  - VkPhysicalDeviceMeshShaderPropertiesNV

### New Enum Constants

- `VK_NV_MESH_SHADER_EXTENSION_NAME`
- `VK_NV_MESH_SHADER_SPEC_VERSION`
- Extending VkPipelineStageFlagBits:
  - `VK_PIPELINE_STAGE_MESH_SHADER_BIT_NV`
  - `VK_PIPELINE_STAGE_TASK_SHADER_BIT_NV`
- Extending VkShaderStageFlagBits:
  - `VK_SHADER_STAGE_MESH_BIT_NV`
  - `VK_SHADER_STAGE_TASK_BIT_NV`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MESH_SHADER_FEATURES_NV`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MESH_SHADER_PROPERTIES_NV`

### New or Modified Built-In Variables

- TaskCountNV
- PrimitiveCountNV
- PrimitiveIndicesNV
- ClipDistancePerViewNV
- CullDistancePerViewNV
- LayerPerViewNV
- MeshViewCountNV
- MeshViewIndicesNV
- (modified)`Position`
- (modified)`PointSize`

- (modified)`ClipDistance`
- (modified)`CullDistance`
- (modified)`PrimitiveId`
- (modified)`Layer`
- (modified)`ViewportIndex`
- (modified)`WorkgroupSize`
- (modified)`WorkgroupId`
- (modified)`LocalInvocationId`
- (modified)`GlobalInvocationId`
- (modified)`LocalInvocationIndex`
- (modified)`DrawIndex`
- (modified)`ViewportMaskNV`
- (modified)`PositionPerViewNV`
- (modified)`ViewportMaskPerViewNV`

**New SPIR-V Capability**

- `MeshShadingNV`

**Issues**

1. How to name this extension?

   **RESOLVED**: VK_NV_mesh_shader

   Other options considered:

   - VK_NV_mesh_shading
   - VK_NV_programmable_mesh_shading
   - VK_NV_primitive_group_shading
   - VK_NV_grouped_drawing

2. Do we need a new VkPrimitiveTopology?

   **RESOLVED**: No. We skip the InputAssembler stage.

3. Should we allow Instancing?

   **RESOLVED**: No. There is no fixed function input, other than the IDs. However, allow offsetting with a “first” value.

4. Should we use existing vkCmdDraw or introduce new functions?

   **RESOLVED**: Introduce new functions.

New functions make it easier to separate from “programmable primitive shading” chapter, less “dual use” language about existing functions having alternative behavior. The text around the existing “draws” is heavily based around emitting vertices.

5. If new functions, how to name?

   **RESOLVED**: CmdDrawMeshTasks*

   Other options considered:

   - CmdDrawMeshed
   - CmdDrawTasked
   - CmdDrawGrouped

6. Should VK_SHADER_STAGE_ALL_GRAPHICS be updated to include the new stages?

   **RESOLVED**: No. If an application were to be recompiled with headers that include additional shader stage bits in VK_SHADER_STAGE_ALL_GRAPHICS, then the previously valid application would no longer be valid on implementations that do not support mesh or task shaders. This means the change would not be backwards compatible. It is too bad VkShaderStageFlagBits does not have a dedicated “all supported graphics stages” bit like VK_PIPELINE_STAGE_ALL_GRAPHICS_BIT, which would have avoided this problem.

### Version History

- Revision 1, 2018-07-19 (Christoph Kubisch, Daniel Koch)
  - Internal revisions

## VK_NV_optical_flow

### Name String

`VK_NV_optical_flow`

### Extension Type

Device extension

### Registered Extension Number

465

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or

Vulkan Version 1.1
and
VK_KHR_format_feature_flags2
and
VK_KHR_synchronization2
or
Vulkan Version 1.3

### Contact

- Carsten Rohde [GitHub]crohde

### Other Extension Metadata

### Last Modified Date

2022-09-26

### Contributors

- Carsten Rohde, NVIDIA
- Vipul Parashar, NVIDIA
- Jeff Bolz, NVIDIA
- Eric Werness, NVIDIA

### Description

Optical flow are fundamental algorithms in computer vision (CV) area. This extension allows applications to estimate 2D displacement of pixels between two frames.

| | |
|---|---|
| **NOTE** | This extension is designed to be used with upcoming NVIDIA Optical Flow SDK Version 5 which will be available on NVIDIA Developer webpage. |

### New Object Types

- VkOpticalFlowSessionNV

### New Commands

- vkBindOpticalFlowSessionImageNV
- vkCmdOpticalFlowExecuteNV
- vkCreateOpticalFlowSessionNV
- vkDestroyOpticalFlowSessionNV
- vkGetPhysicalDeviceOpticalFlowImageFormatsNV

### New Structures

- VkOpticalFlowExecuteInfoNV
- VkOpticalFlowImageFormatPropertiesNV

- VkOpticalFlowSessionCreateInfoNV
- Extending VkOpticalFlowSessionCreateInfoNV:
  - VkOpticalFlowSessionCreatePrivateDataInfoNV
- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceOpticalFlowFeaturesNV
- Extending VkPhysicalDeviceImageFormatInfo2, VkImageCreateInfo:
  - VkOpticalFlowImageFormatInfoNV
- Extending VkPhysicalDeviceProperties2:
  - VkPhysicalDeviceOpticalFlowPropertiesNV

### New Enums

- VkOpticalFlowExecuteFlagBitsNV
- VkOpticalFlowGridSizeFlagBitsNV
- VkOpticalFlowPerformanceLevelNV
- VkOpticalFlowSessionBindingPointNV
- VkOpticalFlowSessionCreateFlagBitsNV
- VkOpticalFlowUsageFlagBitsNV

### New Bitmasks

- VkOpticalFlowExecuteFlagsNV
- VkOpticalFlowGridSizeFlagsNV
- VkOpticalFlowSessionCreateFlagsNV
- VkOpticalFlowUsageFlagsNV

### New Enum Constants

- `VK_NV_OPTICAL_FLOW_EXTENSION_NAME`
- `VK_NV_OPTICAL_FLOW_SPEC_VERSION`
- Extending VkAccessFlagBits2:
  - `VK_ACCESS_2_OPTICAL_FLOW_READ_BIT_NV`
  - `VK_ACCESS_2_OPTICAL_FLOW_WRITE_BIT_NV`
- Extending VkFormat:
  - `VK_FORMAT_R16G16_S10_5_NV`
  - `VK_FORMAT_R16G16_SFIXED5_NV`
- Extending VkFormatFeatureFlagBits2:
  - `VK_FORMAT_FEATURE_2_OPTICAL_FLOW_COST_BIT_NV`
  - `VK_FORMAT_FEATURE_2_OPTICAL_FLOW_IMAGE_BIT_NV`

    - VK_FORMAT_FEATURE_2_OPTICAL_FLOW_VECTOR_BIT_NV
- Extending VkObjectType:
    - VK_OBJECT_TYPE_OPTICAL_FLOW_SESSION_NV
- Extending VkPipelineStageFlagBits2:
    - VK_PIPELINE_STAGE_2_OPTICAL_FLOW_BIT_NV
- Extending VkQueueFlagBits:
    - VK_QUEUE_OPTICAL_FLOW_BIT_NV
- Extending VkStructureType:
    - VK_STRUCTURE_TYPE_OPTICAL_FLOW_EXECUTE_INFO_NV
    - VK_STRUCTURE_TYPE_OPTICAL_FLOW_IMAGE_FORMAT_INFO_NV
    - VK_STRUCTURE_TYPE_OPTICAL_FLOW_IMAGE_FORMAT_PROPERTIES_NV
    - VK_STRUCTURE_TYPE_OPTICAL_FLOW_SESSION_CREATE_INFO_NV
    - VK_STRUCTURE_TYPE_OPTICAL_FLOW_SESSION_CREATE_PRIVATE_DATA_INFO_NV
    - VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_OPTICAL_FLOW_FEATURES_NV
    - VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_OPTICAL_FLOW_PROPERTIES_NV

### Examples

```
// Example querying available input formats
VkOpticalFlowImageFormatInfoNV ofFormatInfo = {
VK_STRUCTURE_TYPE_OPTICAL_FLOW_IMAGE_FORMAT_INFO_NV };
ofFormatInfo.usage = VK_OPTICAL_FLOW_USAGE_INPUT_BIT_NV;

uint32_t count = 0;
vkGetPhysicalDeviceOpticalFlowImageFormatsNV(physicalDevice, &ofFormatInfo, &count,
NULL);
VkOpticalFlowImageFormatPropertiesNV* fmt = new VkOpticalFlowImageFormatPropertiesNV
[count];
memset(fmt, 0, count  * sizeof(VkOpticalFlowImageFormatPropertiesNV));
for (uint32_t i = 0; i < count; i++) {
    fmt[i].sType = VK_STRUCTURE_TYPE_OPTICAL_FLOW_IMAGE_FORMAT_PROPERTIES_NV;
}
vkGetPhysicalDeviceOpticalFlowImageFormatsNV(physicalDevice, &ofFormatInfo, &count,
fmt);

// Pick one of the available formats
VkFormat inputFormat = fmt[0].format;

// Check feature support for optimal tiling
VkFormatProperties3 formatProperties3 = { VK_STRUCTURE_TYPE_FORMAT_PROPERTIES_3 };
VkFormatProperties2 formatProperties2 = { VK_STRUCTURE_TYPE_FORMAT_PROPERTIES_2,
&formatProperties3 };
vkGetPhysicalDeviceFormatProperties2(physicalDevice, inputFormat, &formatProperties2);
if (!(formatProperties3.optimalTilingFeatures &
```

```
VK_FORMAT_FEATURE_2_OPTICAL_FLOW_IMAGE_BIT_NV)) {
    return false;
}

// Check support for image creation parameters
VkPhysicalDeviceImageFormatInfo2 imageFormatInfo2 = {
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_IMAGE_FORMAT_INFO_2, &ofFormatInfo };
imageFormatInfo2.format = inputFormat;
imageFormatInfo2.type = VK_IMAGE_TYPE_2D;
imageFormatInfo2.tiling = VK_IMAGE_TILING_OPTIMAL;
imageFormatInfo2.usage = VK_IMAGE_USAGE_SAMPLED_BIT | VK_IMAGE_USAGE_TRANSFER_DST_BIT;

VkImageFormatProperties2 imageFormatProperties2 = {
VK_STRUCTURE_TYPE_IMAGE_FORMAT_PROPERTIES_2 };
if (vkGetPhysicalDeviceImageFormatProperties2(physicalDevice, &imageFormatInfo2,
&imageFormatProperties2) != VK_SUCCESS) {
    return false;
}

VkImageCreateInfo imageCreateInfo = { VK_STRUCTURE_TYPE_IMAGE_CREATE_INFO,
&ofFormatInfo };
imageCreateInfo.imageType = VK_IMAGE_TYPE_2D;
imageCreateInfo.format = inputFormat;
imageCreateInfo.extent = { width, height, (uint32_t)1};
imageCreateInfo.mipLevels = 1;
imageCreateInfo.arrayLayers = 1;
imageCreateInfo.samples = VK_SAMPLE_COUNT_1_BIT;
imageCreateInfo.usage = VK_IMAGE_USAGE_SAMPLED_BIT | VK_IMAGE_USAGE_TRANSFER_DST_BIT;;
imageCreateInfo.tiling = VK_IMAGE_TILING_OPTIMAL;

vkCreateImage(device, &imageCreateInfo, NULL, &input);
"allocate memory, bind image, create view"

"do the same for reference and output"

// Create optical flow session
VkOpticalFlowSessionCreateInfoNV sessionCreateInfo = {
VK_STRUCTURE_TYPE_OPTICAL_FLOW_SESSION_CREATE_INFO_NV };
sessionCreateInfo.width = width;
sessionCreateInfo.height = height;
sessionCreateInfo.imageFormat = inputFormat;
sessionCreateInfo.flowVectorFormat = outputFormat;
sessionCreateInfo.outputGridSize = VK_OPTICAL_FLOW_GRID_SIZE_4X4_BIT_NV;
sessionCreateInfo.performanceLevel = VK_OPTICAL_FLOW_PERFORMANCE_LEVEL_SLOW_NV;
VkOpticalFlowSessionNV session;
vkCreateOpticalFlowSessionNV(device, &sessionCreateInfo, NULL, &session);

"allocate command buffer"

"transfer images to VK_PIPELINE_STAGE_2_OPTICAL_FLOW_BIT_NV"
"transfer input images to VK_ACCESS_2_OPTICAL_FLOW_READ_BIT_NV and output image to
```

```
VK_ACCESS_2_OPTICAL_FLOW_WRITE_BIT_NV"

vkBindOpticalFlowSessionImageNV(device, session,
VK_OPTICAL_FLOW_SESSION_BINDING_POINT_INPUT_NV, inputView, VK_IMAGE_LAYOUT_GENERAL);
vkBindOpticalFlowSessionImageNV(device, session,
VK_OPTICAL_FLOW_SESSION_BINDING_POINT_REFERENCE_NV, refView, VK_IMAGE_LAYOUT_GENERAL);
vkBindOpticalFlowSessionImageNV(device, session,
VK_OPTICAL_FLOW_SESSION_BINDING_POINT_FLOW_VECTOR_NV, outputView,
VK_IMAGE_LAYOUT_GENERAL);

VkOpticalFlowExecuteInfoNV opticalFlowExecuteInfo = {
VK_STRUCTURE_TYPE_OPTICAL_FLOW_EXECUTE_INFO_NV };
vkCmdOpticalFlowExecuteNV(cmd, session, &opticalFlowExecuteInfo);

"submit command buffer"
```

### Version History

- Revision 1, 2022-09-26 (Carsten Rohde)
  - Internal revisions

## VK_NV_per_stage_descriptor_set

### Name String

`VK_NV_per_stage_descriptor_set`

### Extension Type

Device extension

### Registered Extension Number

517

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_maintenance6

### Contact

- Piers Daniell [GitHub]pdaniell-nv

### Other Extension Metadata

### Last Modified Date

2023-10-16

### IP Status

No known IP claims.

### Contributors

- Daniel Story, Nintendo

### Description

This extension introduces a new descriptor set layout creation flag that allows bindings in a descriptor set to be scoped to each shader stage. This means that shaders bound at the same time **may** use completely different descriptor set layouts without any restrictions on compatibility, and that the descriptor limits that would otherwise apply to the union of all stages together instead apply to each stage individually. It also means that descriptors shared by multiple stages **must** be bound to each stage or set of stages that use a unique descriptor set layout using their specific per stage descriptor set layout(s).

This extension also allows each of the new descriptor binding functions from VK_KHR_maintenance6 to have their VkPipelineLayout member be optionally set to VK_NULL_HANDLE, in which case the pipeline layout information is taken from a VkPipelineLayoutCreateInfo structure in the `pNext` chain. This enables descriptors to be directly bound using descriptor set layouts without applications needing to create and manage VkPipelineLayout objects at command recording time.

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDevicePerStageDescriptorSetFeaturesNV

### New Enum Constants

- `VK_NV_PER_STAGE_DESCRIPTOR_SET_EXTENSION_NAME`
- `VK_NV_PER_STAGE_DESCRIPTOR_SET_SPEC_VERSION`
- Extending VkDescriptorSetLayoutCreateFlagBits:
  - `VK_DESCRIPTOR_SET_LAYOUT_CREATE_PER_STAGE_BIT_NV`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PER_STAGE_DESCRIPTOR_SET_FEATURES_NV`

### Issues

None

### Version History

- Revision 1, 2023-10-16 (Piers Daniell)
  - Initial draft

## VK_NV_present_barrier

### Name String

`VK_NV_present_barrier`

### Extension Type

Device extension

### Registered Extension Number

293

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1
and
VK_KHR_surface
and
VK_KHR_get_surface_capabilities2
and
VK_KHR_swapchain

### Contact

- Liya Li [GitHub]liyli

### Other Extension Metadata

### Last Modified Date

2022-05-16

### Contributors

- Liya Li, Nvidia
- Martin Schwarzer, Nvidia
- Andy Wolf, Nvidia
- Ian Williams, Nvidia
- Ben Morris, Nvidia
- James Jones, Nvidia
- Jeff Juliano, Nvidia

### Description

This extension adds support for synchronizing corresponding presentation requests across multiple swapchains using the *present barrier*.

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDevicePresentBarrierFeaturesNV
- Extending VkSurfaceCapabilities2KHR:
  - VkSurfaceCapabilitiesPresentBarrierNV
- Extending VkSwapchainCreateInfoKHR:
  - VkSwapchainPresentBarrierCreateInfoNV

### New Enum Constants

- `VK_NV_PRESENT_BARRIER_EXTENSION_NAME`
- `VK_NV_PRESENT_BARRIER_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PRESENT_BARRIER_FEATURES_NV`
  - `VK_STRUCTURE_TYPE_SURFACE_CAPABILITIES_PRESENT_BARRIER_NV`
  - `VK_STRUCTURE_TYPE_SWAPCHAIN_PRESENT_BARRIER_CREATE_INFO_NV`

### Issues

1) Is there a query interface to check if a swapchain is using the present barrier?

**RESOLVED**. There is no such query interface. When creating a swapchain, an application can specify to use the *present barrier*, and if the swapchain is created successfully, this swapchain will be using the present barrier.

2) Do we need an extra interface to set up the present barrier across distributed systems?

**RESOLVED**. If the required hardware is presented in the system, and all settings for the physical synchronization with other systems are set up, an implementation manages the configuration automatically when creating a swapchain, without any extra calls from the application.

### Version History

- Revision 1, 2022-07-20
  - Initial version

## VK_NV_raw_access_chains

### Name String

`VK_NV_raw_access_chains`

### Extension Type

Device extension

### Registered Extension Number

556

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

None

### SPIR-V Dependencies

- SPV_NV_raw_access_chains

### Contact

- Rodrigo Locatti [GitHub]rlocatti

### Other Extension Metadata

### Last Modified Date

2023-12-04

### Interactions and External Dependencies

- This extension requires `SPV_NV_raw_access_chains`

### Contributors

- Hans-Kristian Arntzen, Valve
- Rodrigo Locatti, NVIDIA

### Description

This extension allows the use of the `SPV_NV_raw_access_chains` extension in SPIR-V shader modules. This enables SPIR-V producers to efficiently implement interfaces similar to Direct3D structured buffers and byte address buffers, allowing shaders compiled from an HLSL source to generate more efficient code.

### New SPIR-V Capabilities

- RawAccessChainsNV

### Version History

- Revision 1, 2023-12-04 (Rodrigo Locatti)
  - Initial revision

## VK_NV_ray_tracing_invocation_reorder

### Name String

`VK_NV_ray_tracing_invocation_reorder`

### Extension Type

Device extension

### Registered Extension Number

491

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_ray_tracing_pipeline

### SPIR-V Dependencies

- SPV_NV_shader_invocation_reorder

### Contact

- Eric Werness [GitHub]ewerness-nv

### Other Extension Metadata

### Last Modified Date

2022-11-02

### Interactions and External Dependencies

- This extension provides API support for `GL_NV_shader_invocation_reorder`

### Contributors

- Eric Werness, NVIDIA
- Ashwin Lele, NVIDIA

### Description

The ray tracing pipeline API provides some ability to reorder for locality, but it is useful to have more control over how the reordering happens and what information is included in the reordering. The shader API provides a hit object to contain result information from the hit which can be used as part of the explicit sorting plus options that contain an integer for hint bits to use to add more locality.

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:

- VkPhysicalDeviceRayTracingInvocationReorderFeaturesNV

- Extending VkPhysicalDeviceProperties2:
  - VkPhysicalDeviceRayTracingInvocationReorderPropertiesNV

### New Enums

- VkRayTracingInvocationReorderModeNV

### New Enum Constants

- `VK_NV_RAY_TRACING_INVOCATION_REORDER_EXTENSION_NAME`
- `VK_NV_RAY_TRACING_INVOCATION_REORDER_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_RAY_TRACING_INVOCATION_REORDER_FEATURES_NV`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_RAY_TRACING_INVOCATION_REORDER_PROPERTIES_NV`

### HLSL Mapping

HLSL does not provide this functionality natively yet.

However, it is possible to use this functionality via SPIR-V Intrinsics.

The codes for shader invocation reorder are obtained from this page:

```
#define ShaderInvocationReorderNV 5383
#define HitObjectAttributeNV 5385

#define OpHitObjectRecordHitMotionNV 5249
#define OpHitObjectRecordHitWithIndexMotionNV 5250
#define OpHitObjectRecordMissMotionNV 5251
#define OpHitObjectGetWorldToObjectNV 5252
#define OpHitObjectGetObjectToWorldNV 5253
#define OpHitObjectGetObjectRayDirectionNV 5254
#define OpHitObjectGetObjectRayOriginNV 5255
#define OpHitObjectTraceRayMotionNV 5256
#define OpHitObjectGetShaderRecordBufferHandleNV 5257
#define OpHitObjectGetShaderBindingTableRecordIndexNV 5258
#define OpHitObjectRecordEmptyNV 5259
#define OpHitObjectTraceRayNV 5260
#define OpHitObjectRecordHitNV 5261
#define OpHitObjectRecordHitWithIndexNV 5262
#define OpHitObjectRecordMissNV 5263
#define OpHitObjectExecuteShaderNV 5264
#define OpHitObjectGetCurrentTimeNV 5265
#define OpHitObjectGetAttributesNV 5266
#define OpHitObjectGetHitKindNV 5267
#define OpHitObjectGetPrimitiveIndexNV 5268
#define OpHitObjectGetGeometryIndexNV 5269
```

```
#define OpHitObjectGetInstanceIdNV 5270
#define OpHitObjectGetInstanceCustomIndexNV 5271
#define OpHitObjectGetWorldRayDirectionNV 5272
#define OpHitObjectGetWorldRayOriginNV 5273
#define OpHitObjectGetRayTMaxNV 5274
#define OpHitObjectGetRayTMinNV 5275
#define OpHitObjectIsEmptyNV 5276
#define OpHitObjectIsHitNV 5277
#define OpHitObjectIsMissNV 5278
#define OpReorderThreadWithHitObjectNV 5279
#define OpReorderThreadWithHintNV 5280
#define OpTypeHitObjectNV 5281
```

The capability and extension need to be added:

```
[[vk::ext_capability(ShaderInvocationReorderNV)]]
[[vk::ext_extension("SPV_NV_shader_invocation_reorder")]]
```

The creation of the `HitObject` type can be done like this:

```
[[vk::ext_type_def(HitObjectAttributeNV, OpTypeHitObjectNV)]]
void createHitObjectNV();
#define HitObjectNV vk::ext_type<HitObjectAttributeNV>
```

The payload:

- must be global
- needs the `RayPayloadKHR` attribute as an extra storage class

```
struct [raypayload] HitPayload
{
  float hitT : write(closesthit, miss) : read(caller);
  int instanceIndex : write(closesthit) : read(caller);
  float3 pos : write(closesthit) : read(caller);
  float3 nrm : write(closesthit) : read(caller);
};

#define RayPayloadKHR 5338
[[vk::ext_storage_class(RayPayloadKHR)]] static HitPayload payload;
```

Here is the declaration of a few invocation reordering functions:

```
[[vk::ext_instruction(OpHitObjectRecordEmptyNV)]]
void hitObjectRecordEmptyNV([[vk::ext_reference]] HitObjectNV hitObject);

[[vk::ext_instruction(OpHitObjectTraceRayNV)]]
```

```
void hitObjectTraceRayNV(
    [[vk::ext_reference]] HitObjectNV hitObject,
    RaytracingAccelerationStructure as,
    uint RayFlags,
    uint CullMask,
    uint SBTOffset,
    uint SBTStride,
    uint MissIndex,
    float3 RayOrigin,
    float RayTmin,
    float3 RayDirection,
    float RayTMax,
    [[vk::ext_reference]] [[vk::ext_storage_class(RayPayloadKHR)]] HitPayload payload
  );

[[vk::ext_instruction(OpReorderThreadWithHintNV)]]
void reorderThreadWithHintNV(int Hint, int Bits);

[[vk::ext_instruction(OpReorderThreadWithHitObjectNV)]]
void reorderThreadWithHitObjectNV([[vk::ext_reference]] HitObjectNV hitObject);

[[vk::ext_instruction(OpHitObjectExecuteShaderNV)]]
void hitObjectExecuteShaderNV([[vk::ext_reference]] HitObjectNV hitObject, [[vk
::ext_reference]] [[vk::ext_storage_class(RayPayloadKHR)]] HitPayload payload);

[[vk::ext_instruction(OpHitObjectIsHitNV)]]
bool hitObjectIsHitNV([[vk::ext_reference]] HitObjectNV hitObject);
```

Using the function in the code, can be done like this

```
  if (USE_SER == 1)
  {
    createHitObjectNV();
    HitObjectNV hObj; //  hitObjectNV hObj;
    hitObjectRecordEmptyNV(hObj); //Initialize to an empty hit object
    hitObjectTraceRayNV(hObj, topLevelAS, rayFlags, 0xFF, 0, 0, 0, r.Origin, 0.0, r
.Direction, INFINITE, payload);
    reorderThreadWithHitObjectNV(hObj);
    hitObjectExecuteShaderNV(hObj, payload);
  }
```

Note:

- createHitObjectNV() needs to be call at least once. This can be also done in the main entry of the shader.
- Function with a payload parameter, needs to have the payload struct defined before. There are no templated declaration of the function.

### Version History

- Revision 1, 2020-09-12 (Eric Werness, Ashwin Lele)
  - Initial external release

## VK_NV_ray_tracing_motion_blur

### Name String

`VK_NV_ray_tracing_motion_blur`

### Extension Type

Device extension

### Registered Extension Number

328

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_ray_tracing_pipeline

### SPIR-V Dependencies

- SPV_NV_ray_tracing_motion_blur

### Contact

- Eric Werness

### Other Extension Metadata

### Last Modified Date

2021-06-16

### Interactions and External Dependencies

- This extension provides API support for `GL_NV_ray_tracing_motion_blur`

### Contributors

- Eric Werness, NVIDIA
- Ashwin Lele, NVIDIA

### Description

Ray tracing support in the API provides an efficient mechanism to intersect rays against static geometry, but rendering algorithms often want to support motion, which is more efficiently supported with motion-specific algorithms. This extension adds a set of mechanisms to support fast tracing of moving geometry:

- A ray pipeline trace call which takes a time parameter
- Flags to enable motion support in an acceleration structure
- Support for time-varying vertex positions in a geometry
- Motion instances to move existing instances over time

The motion represented here is parameterized across a normalized timestep between 0.0 and 1.0. A motion trace using `OpTraceRayMotionNV` provides a time within that normalized range to be used when intersecting that ray with geometry. The geometry can be provided with motion by a combination of adding a second vertex position for time of 1.0 using `VkAccelerationStructureGeometryMotionTrianglesDataNV` and providing multiple transforms in the instance using `VkAccelerationStructureMotionInstanceNV`.

#### New Structures

- VkAccelerationStructureMatrixMotionInstanceNV
- VkAccelerationStructureMotionInstanceNV
- VkAccelerationStructureSRTMotionInstanceNV
- VkSRTDataNV
- Extending VkAccelerationStructureCreateInfoKHR:
  - VkAccelerationStructureMotionInfoNV
- Extending VkAccelerationStructureGeometryTrianglesDataKHR:
  - VkAccelerationStructureGeometryMotionTrianglesDataNV
- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceRayTracingMotionBlurFeaturesNV

#### New Unions

- VkAccelerationStructureMotionInstanceDataNV

#### New Enums

- VkAccelerationStructureMotionInstanceTypeNV

#### New Bitmasks

- VkAccelerationStructureMotionInfoFlagsNV
- VkAccelerationStructureMotionInstanceFlagsNV

#### New Enum Constants

- `VK_NV_RAY_TRACING_MOTION_BLUR_EXTENSION_NAME`
- `VK_NV_RAY_TRACING_MOTION_BLUR_SPEC_VERSION`
- Extending VkAccelerationStructureCreateFlagBitsKHR:

    - `VK_ACCELERATION_STRUCTURE_CREATE_MOTION_BIT_NV`
- Extending VkBuildAccelerationStructureFlagBitsKHR:
    - `VK_BUILD_ACCELERATION_STRUCTURE_MOTION_BIT_NV`
- Extending VkPipelineCreateFlagBits:
    - `VK_PIPELINE_CREATE_RAY_TRACING_ALLOW_MOTION_BIT_NV`
- Extending VkStructureType:
    - `VK_STRUCTURE_TYPE_ACCELERATION_STRUCTURE_GEOMETRY_MOTION_TRIANGLES_DATA_NV`
    - `VK_STRUCTURE_TYPE_ACCELERATION_STRUCTURE_MOTION_INFO_NV`
    - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_RAY_TRACING_MOTION_BLUR_FEATURES_NV`

#### Issues

(1) What size is VkAccelerationStructureMotionInstanceNV?

- Added a note on the structure size and made the stride explicit in the language.

(2) Allow arrayOfPointers for motion TLAS?

- Yes, with a packed encoding to minimize the amount of data sent for metadata.

#### Version History

- Revision 1, 2020-06-16 (Eric Werness, Ashwin Lele)
    - Initial external release

## VK_NV_ray_tracing_validation

#### Name String

`VK_NV_ray_tracing_validation`

#### Extension Type

Device extension

#### Registered Extension Number

569

#### Revision

1

#### Ratification Status

Not ratified

#### Extension and Version Dependencies

None

### Contact

- Vikram Kushwaha [GitHub]vkushwaha-nv

### Extension Proposal

VK_NV_ray_tracing_validation

### Other Extension Metadata

### Last Modified Date

2024-03-04

### Contributors

- Vikram Kushwaha, NVIDIA
- Eric Werness, NVIDIA
- Piers Daniell, NVIDIA

### Description

This extension adds support for performing ray tracing validation at an implementation level.

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceRayTracingValidationFeaturesNV

### New Enum Constants

- `VK_NV_RAY_TRACING_VALIDATION_EXTENSION_NAME`
- `VK_NV_RAY_TRACING_VALIDATION_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_RAY_TRACING_VALIDATION_FEATURES_NV`

### Version History

- Revision 1, 2024-03-04 (Vikram Kushwaha)
  - Initial draft

## VK_NV_representative_fragment_test

### Name String

`VK_NV_representative_fragment_test`

### Extension Type

Device extension

### Registered Extension Number

167

**Revision**

2

**Ratification Status**

Not ratified

**Extension and Version Dependencies**

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

**Contact**

- Kedarnath Thangudu [GitHub]kthangudu

**Other Extension Metadata**

**Last Modified Date**

2018-09-13

**Contributors**

- Kedarnath Thangudu, NVIDIA
- Christoph Kubisch, NVIDIA
- Pierre Boudier, NVIDIA
- Pat Brown, NVIDIA
- Jeff Bolz, NVIDIA
- Eric Werness, NVIDIA

**Description**

This extension provides a new representative fragment test that allows implementations to reduce the amount of rasterization and fragment processing work performed for each point, line, or triangle primitive. For any primitive that produces one or more fragments that pass all other early fragment tests, the implementation is permitted to choose one or more “representative” fragments for processing and discard all other fragments. For draw calls rendering multiple points, lines, or triangles arranged in lists, strips, or fans, the representative fragment test is performed independently for each of those primitives.

This extension is useful for applications that use an early render pass to determine the full set of primitives that would be visible in the final scene. In this render pass, such applications would set up a fragment shader that enables early fragment tests and writes to an image or shader storage buffer to record the ID of the primitive that generated the fragment. Without this extension, the shader would record the ID separately for each visible fragment of each primitive. With this extension, fewer stores will be performed, particularly for large primitives.

The representative fragment test has no effect if early fragment tests are not enabled via the fragment shader. The set of fragments discarded by the representative fragment test is implementation-dependent and may vary from frame to frame. In some cases, the representative

fragment test may not discard any fragments for a given primitive.

### New Structures

- Extending VkGraphicsPipelineCreateInfo:
  - VkPipelineRepresentativeFragmentTestStateCreateInfoNV
- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceRepresentativeFragmentTestFeaturesNV

### New Enum Constants

- `VK_NV_REPRESENTATIVE_FRAGMENT_TEST_EXTENSION_NAME`
- `VK_NV_REPRESENTATIVE_FRAGMENT_TEST_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_REPRESENTATIVE_FRAGMENT_TEST_FEATURES_NV`
  - `VK_STRUCTURE_TYPE_PIPELINE_REPRESENTATIVE_FRAGMENT_TEST_STATE_CREATE_INFO_NV`

### Issues

(1) Is the representative fragment test guaranteed to have any effect?

**RESOLVED**: No. As specified, we only guarantee that each primitive with at least one fragment that passes prior tests will have one fragment passing the representative fragment tests. We do not guarantee that any particular fragment will fail the test.

In the initial implementation of this extension, the representative fragment test is treated as an optimization that may be completely disabled for some pipeline states. This feature was designed for a use case where the fragment shader records information on individual primitives using shader storage buffers or storage images, with no writes to color or depth buffers.

(2) Will the set of fragments that pass the representative fragment test be repeatable if you draw the same scene over and over again?

**RESOLVED**: No. The set of fragments that pass the representative fragment test is implementation-dependent and may vary due to the timing of operations performed by the GPU.

(3) What happens if you enable the representative fragment test with writes to color and/or depth render targets enabled?

**RESOLVED**: If writes to the color or depth buffer are enabled, they will be performed for any fragments that survive the relevant tests. Any fragments that fail the representative fragment test will not update color buffers. For the use cases intended for this feature, we do not expect color or depth writes to be enabled.

(4) How do derivatives and automatic texture LOD computations work with the representative fragment test enabled?

**RESOLVED**: If a fragment shader uses derivative functions or texture lookups using automatic LOD

computation, derivatives will be computed identically whether or not the representative fragment test is enabled. For the use cases intended for this feature, we do not expect the use of derivatives in the fragment shader.

### Version History

- Revision 2, 2018-09-13 (pbrown)
  - Add issues.
- Revision 1, 2018-08-22 (Kedarnath Thangudu)
  - Internal Revisions

## VK_NV_sample_mask_override_coverage

### Name String

`VK_NV_sample_mask_override_coverage`

### Extension Type

Device extension

### Registered Extension Number

95

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

None

### SPIR-V Dependencies

- SPV_NV_sample_mask_override_coverage

### Contact

- Piers Daniell [GitHub]pdaniell-nv

### Other Extension Metadata

### Last Modified Date

2016-12-08

### IP Status

No known IP claims.

### Interactions and External Dependencies

- This extension provides API support for `GL_NV_sample_mask_override_coverage`

### Contributors

- Daniel Koch, NVIDIA
- Jeff Bolz, NVIDIA

### Description

This extension adds support for the following SPIR-V extension in Vulkan:

- `SPV_NV_sample_mask_override_coverage`

The extension provides access to the `OverrideCoverageNV` decoration under the `SampleMaskOverrideCoverageNV` capability. Adding this decoration to a variable with the `SampleMask` builtin decoration allows the shader to modify the coverage mask and affect which samples are used to process the fragment.

When using GLSL source-based shader languages, the `override_coverage` layout qualifier from `GL_NV_sample_mask_override_coverage` maps to the `OverrideCoverageNV` decoration. To use the `override_coverage` layout qualifier in GLSL the `GL_NV_sample_mask_override_coverage` extension must be enabled. Behavior is described in the `GL_NV_sample_mask_override_coverage` extension spec.

### New Enum Constants

- `VK_NV_SAMPLE_MASK_OVERRIDE_COVERAGE_EXTENSION_NAME`
- `VK_NV_SAMPLE_MASK_OVERRIDE_COVERAGE_SPEC_VERSION`

### New Variable Decoration

- OverrideCoverageNV in SampleMask

### New SPIR-V Capabilities

- `SampleMaskOverrideCoverageNV`

### Version History

- Revision 1, 2016-12-08 (Piers Daniell)
  - Internal revisions

## VK_NV_scissor_exclusive

### Name String

`VK_NV_scissor_exclusive`

### Extension Type

Device extension

### Registered Extension Number

206

### Revision

2

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### Contact

- Pat Brown [GitHub]nvpbrown

### Other Extension Metadata

### Last Modified Date

2023-01-18

### IP Status

No known IP claims.

### Interactions and External Dependencies

None

### Contributors

- Pat Brown, NVIDIA
- Jeff Bolz, NVIDIA
- Piers Daniell, NVIDIA
- Daniel Koch, NVIDIA

### Description

This extension adds support for an exclusive scissor test to Vulkan. The exclusive scissor test behaves like the scissor test, except that the exclusive scissor test fails for pixels inside the corresponding rectangle and passes for pixels outside the rectangle. If the same rectangle is used for both the scissor and exclusive scissor tests, the exclusive scissor test will pass if and only if the scissor test fails.

Version 2 of this extension introduces `VK_DYNAMIC_STATE_EXCLUSIVE_SCISSOR_ENABLE_NV` and vkCmdSetExclusiveScissorEnableNV. Applications that use this dynamic state must ensure the implementation advertises at least `specVersion 2` of this extension.

### New Commands

- vkCmdSetExclusiveScissorEnableNV
- vkCmdSetExclusiveScissorNV

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceExclusiveScissorFeaturesNV
- Extending VkPipelineViewportStateCreateInfo:
  - VkPipelineViewportExclusiveScissorStateCreateInfoNV

### New Enum Constants

- `VK_NV_SCISSOR_EXCLUSIVE_EXTENSION_NAME`
- `VK_NV_SCISSOR_EXCLUSIVE_SPEC_VERSION`
- Extending VkDynamicState:
  - `VK_DYNAMIC_STATE_EXCLUSIVE_SCISSOR_ENABLE_NV`
  - `VK_DYNAMIC_STATE_EXCLUSIVE_SCISSOR_NV`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXCLUSIVE_SCISSOR_FEATURES_NV`
  - `VK_STRUCTURE_TYPE_PIPELINE_VIEWPORT_EXCLUSIVE_SCISSOR_STATE_CREATE_INFO_NV`

### Issues

1) For the scissor test, the viewport state must be created with a matching number of scissor and viewport rectangles. Should we have the same requirement for exclusive scissors?

**RESOLVED**: For exclusive scissors, we relax this requirement and allow an exclusive scissor rectangle count that is either zero or equal to the number of viewport rectangles. If you pass in an exclusive scissor count of zero, the exclusive scissor test is treated as disabled.

### Version History

- Revision 2, 2023-01-18 (Piers Daniell)
  - Add dynamic state for explicit exclusive scissor enables
- Revision 1, 2018-07-31 (Pat Brown)
  - Internal revisions

## VK_NV_shader_atomic_float16_vector

### Name String

`VK_NV_shader_atomic_float16_vector`

### Extension Type

Device extension

### Registered Extension Number

564

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

None

### SPIR-V Dependencies

- SPV_NV_shader_atomic_fp16_vector

### Contact

- Jeff Bolz [GitHub]jeffbolznv

### Other Extension Metadata

### Last Modified Date

2024-02-03

### IP Status

No known IP claims.

### Interactions and External Dependencies

- This extension provides API support for `GL_NV_shader_atomic_fp16_vector`

### Contributors

- Jeff Bolz, NVIDIA

### Description

This extension allows a shader to perform atomic add, min, max, and exchange operations on 2- and 4-component vectors of float16. Buffer, workgroup, and image storage classes are all supported.

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceShaderAtomicFloat16VectorFeaturesNV

### New Enum Constants

- `VK_NV_SHADER_ATOMIC_FLOAT16_VECTOR_EXTENSION_NAME`
- `VK_NV_SHADER_ATOMIC_FLOAT16_VECTOR_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_ATOMIC_FLOAT16_VECTOR_FEATURES_NV`

### Issues

None.

### New SPIR-V Capabilities

- `AtomicFloat16VectorNV`

### Version History

- Revision 1, 2024-02-03 (Jeff Bolz)
  - Internal revisions

## VK_NV_shader_image_footprint

### Name String

`VK_NV_shader_image_footprint`

### Extension Type

Device extension

### Registered Extension Number

205

### Revision

2

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### SPIR-V Dependencies

- SPV_NV_shader_image_footprint

### Contact

- Pat Brown [GitHub]nvpbrown

### Other Extension Metadata

### Last Modified Date

2018-09-13

### IP Status

No known IP claims.

**Interactions and External Dependencies**

- This extension provides API support for `GL_NV_shader_texture_footprint`

**Contributors**

- Pat Brown, NVIDIA
- Chris Lentini, NVIDIA
- Daniel Koch, NVIDIA
- Jeff Bolz, NVIDIA

**Description**

This extension adds Vulkan support for the `SPV_NV_shader_image_footprint` SPIR-V extension. That SPIR-V extension provides a new instruction `OpImageSampleFootprintNV` allowing shaders to determine the set of texels that would be accessed by an equivalent filtered texture lookup.

Instead of returning a filtered texture value, the instruction returns a structure that can be interpreted by shader code to determine the footprint of a filtered texture lookup. This structure includes integer values that identify a small neighborhood of texels in the image being accessed and a bitfield that indicates which texels in that neighborhood would be used. The structure also includes a bitfield where each bit identifies whether any texel in a small aligned block of texels would be fetched by the texture lookup. The size of each block is specified by an access *granularity* provided by the shader. The minimum granularity supported by this extension is 2x2 (for 2D textures) and 2x2x2 (for 3D textures); the maximum granularity is 256x256 (for 2D textures) or 64x32x32 (for 3D textures). Each footprint query returns the footprint from a single texture level. When using minification filters that combine accesses from multiple mipmap levels, shaders must perform separate queries for the two levels accessed (“fine” and “coarse”). The footprint query also returns a flag indicating if the texture lookup would access texels from only one mipmap level or from two neighboring levels.

This extension should be useful for multi-pass rendering operations that do an initial expensive rendering pass to produce a first image that is then used as a texture for a second pass. If the second pass ends up accessing only portions of the first image (e.g., due to visibility), the work spent rendering the non-accessed portion of the first image was wasted. With this feature, an application can limit this waste using an initial pass over the geometry in the second image that performs a footprint query for each visible pixel to determine the set of pixels that it needs from the first image. This pass would accumulate an aggregate footprint of all visible pixels into a separate “footprint image” using shader atomics. Then, when rendering the first image, the application can kill all shading work for pixels not in this aggregate footprint.

This extension has a number of limitations. The `OpImageSampleFootprintNV` instruction only supports for two- and three-dimensional textures. Footprint evaluation only supports the CLAMP_TO_EDGE wrap mode; results are undefined for all other wrap modes. Only a limited set of granularity values and that set does not support separate coverage information for each texel in the original image.

When using SPIR-V generated from the OpenGL Shading Language, the new instruction will be generated from code using the new `textureFootprint*NV` built-in functions from the `GL_NV_shader_texture_footprint` shading language extension.

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceShaderImageFootprintFeaturesNV

### New Enum Constants

- `VK_NV_SHADER_IMAGE_FOOTPRINT_EXTENSION_NAME`
- `VK_NV_SHADER_IMAGE_FOOTPRINT_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_IMAGE_FOOTPRINT_FEATURES_NV`

### New SPIR-V Capability

- `ImageFootprintNV`

### Issues

(1) The footprint returned by the SPIR-V instruction is a structure that includes an anchor, an offset, and a mask that represents a 8x8 or 4x4x4 neighborhood of texel groups. But the bits of the mask are not stored in simple pitch order. Why is the footprint built this way?

**RESOLVED**: We expect that applications using this feature will want to use a fixed granularity and accumulate coverage information from the returned footprints into an aggregate "footprint image" that tracks the portions of an image that would be needed by regular texture filtering. If an application is using a two-dimensional image with 4x4 pixel granularity, we expect that the footprint image will use 64-bit texels where each bit in an 8x8 array of bits corresponds to coverage for a 4x4 block in the original image. Texel (0,0) in the footprint image would correspond to texels (0,0) through (31,31) in the original image.

In the usual case, the footprint for a single access will fully contained in a 32x32 aligned region of the original texture, which corresponds to a single 64-bit texel in the footprint image. In that case, the implementation will return an anchor coordinate pointing at the single footprint image texel, an offset vector of (0,0), and a mask whose bits are aligned with the bits in the footprint texel. For this case, the shader can simply atomically OR the mask bits into the contents of the footprint texel to accumulate footprint coverage.

In the worst case, the footprint for a single access spans multiple 32x32 aligned regions and may require updates to four separate footprint image texels. In this case, the implementation will return an anchor coordinate pointing at the lower right footprint image texel and an offset will identify how many "columns" and "rows" of the returned 8x8 mask correspond to footprint texels to the left and above the anchor texel. If the anchor is (2,3), the 64 bits of the returned mask are arranged spatially as follows, where each 4x4 block is assigned a bit number that matches its bit number in the footprint image texels:

```
    +-------------------------+-------------------------+
    | -- -- -- -- -- -- -- -- | -- -- -- -- -- -- -- -- |
    | -- -- -- -- -- -- -- -- | -- -- -- -- -- -- -- -- |
```

```
    | -- -- -- -- -- -- -- -- | -- -- -- -- -- -- -- -- |
    | -- -- -- -- -- -- -- -- | -- -- -- -- -- -- -- -- |
    | -- -- -- -- -- -- -- -- | -- -- -- -- -- -- -- -- |
    | -- -- -- -- -- -- 46 47 | 40 41 42 43 44 45 -- -- |
    | -- -- -- -- -- -- 54 55 | 48 49 50 51 52 53 -- -- |
    | -- -- -- -- -- -- 62 63 | 56 57 58 59 60 61 -- -- |
    +-------------------------+-------------------------+
    | -- -- -- -- -- -- 06 07 | 00 01 02 03 04 05 -- -- |
    | -- -- -- -- -- -- 14 15 | 08 09 10 11 12 13 -- -- |
    | -- -- -- -- -- -- 22 23 | 16 17 18 19 20 21 -- -- |
    | -- -- -- -- -- -- 30 31 | 24 25 26 27 28 29 -- -- |
    | -- -- -- -- -- -- 38 39 | 32 33 34 35 36 37 -- -- |
    | -- -- -- -- -- -- -- -- | -- -- -- -- -- -- -- -- |
    | -- -- -- -- -- -- -- -- | -- -- -- -- -- -- -- -- |
    | -- -- -- -- -- -- -- -- | -- -- -- -- -- -- -- -- |
    +-------------------------+-------------------------+
```

To accumulate coverage for each of the four footprint image texels, a shader can AND the returned mask with simple masks derived from the x and y offset values and then atomically OR the updated mask bits into the contents of the corresponding footprint texel.

```
    uint64_t returnedMask = (uint64_t(footprint.mask.x) | (uint64_t(footprint.mask.y)
<< 32));
    uint64_t rightMask    = ((0xFF >> footprint.offset.x) * 0x0101010101010101UL);
    uint64_t bottomMask   = 0xFFFFFFFFFFFFFFFFUL >> (8 * footprint.offset.y);
    uint64_t bottomRight  = returnedMask & bottomMask & rightMask;
    uint64_t bottomLeft   = returnedMask & bottomMask & (~rightMask);
    uint64_t topRight     = returnedMask & (~bottomMask) & rightMask;
    uint64_t topLeft      = returnedMask & (~bottomMask) & (~rightMask);
```

(2) What should an application do to ensure maximum performance when accumulating footprints into an aggregate footprint image?

**RESOLVED**: We expect that the most common usage of this feature will be to accumulate aggregate footprint coverage, as described in the previous issue. Even if you ignore the anisotropic filtering case where the implementation may return a granularity larger than that requested by the caller, each shader invocation will need to use atomic functions to update up to four footprint image texels for each LOD accessed. Having each active shader invocation perform multiple atomic operations can be expensive, particularly when neighboring invocations will want to update the same footprint image texels.

Techniques can be used to reduce the number of atomic operations performed when accumulating coverage include:

- Have logic that detects returned footprints where all components of the returned offset vector are zero. In that case, the mask returned by the footprint function is guaranteed to be aligned with the footprint image texels and affects only a single footprint image texel.
- Have fragment shaders communicate using built-in functions from the

`VK_NV_shader_subgroup_partitioned` extension or other shader subgroup extensions. If you have multiple invocations in a subgroup that need to update the same texel (x,y) in the footprint image, compute an aggregate footprint mask across all invocations in the subgroup updating that texel and have a single invocation perform an atomic operation using that aggregate mask.

- When the returned footprint spans multiple texels in the footprint image, each invocation need to perform four atomic operations. In the previous issue, we had an example that computed separate masks for “topLeft”, “topRight”, “bottomLeft”, and “bottomRight”. When the invocations in a subgroup have good locality, it might be the case the “top left” for some invocations might refer to footprint image texel (10,10), while neighbors might have their “top left” texels at (11,10), (10,11), and (11,11). If you compute separate masks for even/odd x and y values instead of left/right or top/bottom, the “odd/odd” mask for all invocations in the subgroup hold coverage for footprint image texel (11,11), which can be updated by a single atomic operation for the entire subgroup.

### Examples

TBD

### Version History

- Revision 2, 2018-09-13 (Pat Brown)
  - Add issue (2) with performance tips.
- Revision 1, 2018-08-12 (Pat Brown)
  - Initial draft

## VK_NV_shader_sm_builtins

### Name String

`VK_NV_shader_sm_builtins`

### Extension Type

Device extension

### Registered Extension Number

155

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

Vulkan Version 1.1

### SPIR-V Dependencies

- SPV_NV_shader_sm_builtins

### Contact

- Daniel Koch [GitHub]dgkoch

### Other Extension Metadata

### Last Modified Date

2019-05-28

### Interactions and External Dependencies

- This extension provides API support for `GL_NV_shader_sm_builtins`

### Contributors

- Jeff Bolz, NVIDIA
- Eric Werness, NVIDIA

### Description

This extension provides the ability to determine device-specific properties on NVIDIA GPUs. It provides the number of streaming multiprocessors (SMs), the maximum number of warps (subgroups) that can run on an SM, and shader builtins to enable invocations to identify which SM and warp a shader invocation is executing on.

This extension enables support for the SPIR-V `ShaderSMBuiltinsNV` capability.

These properties and built-ins **should** typically only be used for debugging purposes.

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceShaderSMBuiltinsFeaturesNV
- Extending VkPhysicalDeviceProperties2:
  - VkPhysicalDeviceShaderSMBuiltinsPropertiesNV

### New Enum Constants

- `VK_NV_SHADER_SM_BUILTINS_EXTENSION_NAME`
- `VK_NV_SHADER_SM_BUILTINS_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_SM_BUILTINS_FEATURES_NV`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_SM_BUILTINS_PROPERTIES_NV`

### New or Modified Built-In Variables

- `WarpsPerSMNV`
- `SMCountNV`
- `WarpIDNV`

- `SMIDNV`

### New SPIR-V Capabilities

- `ShaderSMBuiltinsNV`

### Issues

1. What should we call this extension?

   **RESOLVED**: `NV_shader_sm_builtins`. Other options considered included:

   - `NV_shader_smid` - but SMID is really easy to typo/confuse as SIMD.
   - `NV_shader_sm_info` - but **Info** is typically reserved for input structures

### Version History

- Revision 1, 2019-05-28 (Daniel Koch)
  - Internal revisions

## VK_NV_shader_subgroup_partitioned

### Name String

`VK_NV_shader_subgroup_partitioned`

### Extension Type

Device extension

### Registered Extension Number

199

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

Vulkan Version 1.1

### SPIR-V Dependencies

- SPV_NV_shader_subgroup_partitioned

### Contact

- Jeff Bolz [GitHub]jeffbolznv

### Other Extension Metadata

### Last Modified Date

2018-03-17

### Interactions and External Dependencies

- This extension provides API support for `GL_NV_shader_subgroup_partitioned`

### Contributors

- Jeff Bolz, NVIDIA

### Description

This extension enables support for a new class of group operations on subgroups via the `GL_NV_shader_subgroup_partitioned` GLSL extension and `SPV_NV_shader_subgroup_partitioned` SPIR-V extension. Support for these new operations is advertised via the `VK_SUBGROUP_FEATURE_PARTITIONED_BIT_NV` bit.

This extension requires Vulkan 1.1, for general subgroup support.

### New Enum Constants

- `VK_NV_SHADER_SUBGROUP_PARTITIONED_EXTENSION_NAME`
- `VK_NV_SHADER_SUBGROUP_PARTITIONED_SPEC_VERSION`
- Extending VkSubgroupFeatureFlagBits:
  - `VK_SUBGROUP_FEATURE_PARTITIONED_BIT_NV`

### Version History

- Revision 1, 2018-03-17 (Jeff Bolz)
  - Internal revisions

## VK_NV_shading_rate_image

### Name String

`VK_NV_shading_rate_image`

### Extension Type

Device extension

### Registered Extension Number

165

### Revision

3

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### SPIR-V Dependencies

- SPV_NV_shading_rate

### Contact

- Pat Brown [GitHub]nvpbrown

### Other Extension Metadata

### Last Modified Date

2019-07-18

### Interactions and External Dependencies

- This extension provides API support for `GL_NV_shading_rate_image`

### Contributors

- Pat Brown, NVIDIA
- Carsten Rohde, NVIDIA
- Jeff Bolz, NVIDIA
- Daniel Koch, NVIDIA
- Mathias Schott, NVIDIA
- Matthew Netsch, Qualcomm Technologies, Inc.

### Description

This extension allows applications to use a variable shading rate when processing fragments of rasterized primitives. By default, Vulkan will spawn one fragment shader for each pixel covered by a primitive. In this extension, applications can bind a *shading rate image* that can be used to vary the number of fragment shader invocations across the framebuffer. Some portions of the screen may be configured to spawn up to 16 fragment shaders for each pixel, while other portions may use a single fragment shader invocation for a 4x4 block of pixels. This can be useful for use cases like eye tracking, where the portion of the framebuffer that the user is looking at directly can be processed at high frequency, while distant corners of the image can be processed at lower frequency. Each texel in the shading rate image represents a fixed-size rectangle in the framebuffer, covering 16x16 pixels in the initial implementation of this extension. When rasterizing a primitive covering one of these rectangles, the Vulkan implementation reads a texel in the bound shading rate image and looks up the fetched value in a palette to determine a base shading rate.

In addition to the API support controlling rasterization, this extension also adds Vulkan support for the `SPV_NV_shading_rate` extension to SPIR-V. That extension provides two fragment shader variable decorations that allow fragment shaders to determine the shading rate used for processing the fragment:

- `FragmentSizeNV`, which indicates the width and height of the set of pixels processed by the fragment shader.
- `InvocationsPerPixel`, which indicates the maximum number of fragment shader invocations that could be spawned for the pixel(s) covered by the fragment.

When using SPIR-V in conjunction with the OpenGL Shading Language (GLSL), the fragment shader capabilities are provided by the `GL_NV_shading_rate_image` language extension and correspond to the built-in variables `gl_FragmentSizeNV` and `gl_InvocationsPerPixelNV`, respectively.

#### New Commands

- vkCmdBindShadingRateImageNV
- vkCmdSetCoarseSampleOrderNV
- vkCmdSetViewportShadingRatePaletteNV

#### New Structures

- VkCoarseSampleLocationNV
- VkCoarseSampleOrderCustomNV
- VkShadingRatePaletteNV
- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceShadingRateImageFeaturesNV
- Extending VkPhysicalDeviceProperties2:
  - VkPhysicalDeviceShadingRateImagePropertiesNV
- Extending VkPipelineViewportStateCreateInfo:
  - VkPipelineViewportCoarseSampleOrderStateCreateInfoNV
  - VkPipelineViewportShadingRateImageStateCreateInfoNV

#### New Enums

- VkCoarseSampleOrderTypeNV
- VkShadingRatePaletteEntryNV

#### New Enum Constants

- `VK_NV_SHADING_RATE_IMAGE_EXTENSION_NAME`
- `VK_NV_SHADING_RATE_IMAGE_SPEC_VERSION`
- Extending VkAccessFlagBits:
  - `VK_ACCESS_SHADING_RATE_IMAGE_READ_BIT_NV`
- Extending VkDynamicState:
  - `VK_DYNAMIC_STATE_VIEWPORT_COARSE_SAMPLE_ORDER_NV`
  - `VK_DYNAMIC_STATE_VIEWPORT_SHADING_RATE_PALETTE_NV`

- Extending VkImageLayout:
  - VK_IMAGE_LAYOUT_SHADING_RATE_OPTIMAL_NV
- Extending VkImageUsageFlagBits:
  - VK_IMAGE_USAGE_SHADING_RATE_IMAGE_BIT_NV
- Extending VkPipelineStageFlagBits:
  - VK_PIPELINE_STAGE_SHADING_RATE_IMAGE_BIT_NV
- Extending VkStructureType:
  - VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADING_RATE_IMAGE_FEATURES_NV
  - VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADING_RATE_IMAGE_PROPERTIES_NV
  - VK_STRUCTURE_TYPE_PIPELINE_VIEWPORT_COARSE_SAMPLE_ORDER_STATE_CREATE_INFO_NV
  - VK_STRUCTURE_TYPE_PIPELINE_VIEWPORT_SHADING_RATE_IMAGE_STATE_CREATE_INFO_NV

**Issues**

(1) When using shading rates specifying "coarse" fragments covering multiple pixels, we will generate a combined coverage mask that combines the coverage masks of all pixels covered by the fragment. By default, these masks are combined in an implementation-dependent order. Should we provide a mechanism allowing applications to query or specify an exact order?

**RESOLVED**: Yes, this feature is useful for cases where most of the fragment shader can be evaluated once for an entire coarse fragment, but where some per-pixel computations are also required. For example, a per-pixel alpha test may want to kill all the samples for some pixels in a coarse fragment. This sort of test can be implemented using an output sample mask, but such a shader would need to know which bit in the mask corresponds to each sample in the coarse fragment. We are including a mechanism to allow applications to specify the orders of coverage samples for each shading rate and sample count, either as static pipeline state or dynamically via a command buffer. This portion of the extension has its own feature bit.

We will not be providing a query to determine the implementation-dependent default ordering. The thinking here is that if an application cares enough about the coarse fragment sample ordering to perform such a query, it could instead just set its own order, also using custom per-pixel sample locations if required.

(2) For the pipeline stage VK_PIPELINE_STAGE_SHADING_RATE_IMAGE_BIT_NV, should we specify a precise location in the pipeline the shading rate image is accessed (after geometry shading, but before the early fragment tests) or leave it under-specified in case there are other implementations that access the image in a different pipeline location?

**RESOLVED** We are specifying the pipeline stage to be between the final pre-rasterization shader stage (VK_PIPELINE_STAGE_GEOMETRY_SHADER_BIT) and before the first stage used for fragment processing (VK_PIPELINE_STAGE_EARLY_FRAGMENT_TESTS_BIT), which seems to be the natural place to access the shading rate image.

(3) How do centroid-sampled variables work with fragments larger than one pixel?

**RESOLVED** For single-pixel fragments, fragment shader inputs decorated with Centroid are

sampled at an implementation-dependent location in the intersection of the area of the primitive being rasterized and the area of the pixel that corresponds to the fragment. With multi-pixel fragments, we follow a similar pattern, using the intersection of the primitive and the **set** of pixels corresponding to the fragment.

One important thing to keep in mind when using such “coarse” shading rates is that fragment attributes are sampled at the center of the fragment by default, regardless of the set of pixels/samples covered by the fragment. For fragments with a size of 4x4 pixels, this center location will be more than two pixels (1.5 * sqrt(2)) away from the center of the pixels at the corners of the fragment. When rendering a primitive that covers only a small part of a coarse fragment, sampling a color outside the primitive can produce overly bright or dark color values if the color values have a large gradient. To deal with this, an application can use centroid sampling on attributes where “extrapolation” artifacts can lead to overly bright or dark pixels. Note that this same problem also exists for multisampling with single-pixel fragments, but is less severe because it only affects certain samples of a pixel and such bright/dark samples may be averaged with other samples that do not have a similar problem.

#### Version History

- Revision 3, 2019-07-18 (Mathias Schott)
  - Fully list extension interfaces in this appendix.
- Revision 2, 2018-09-13 (Pat Brown)
  - Miscellaneous edits preparing the specification for publication.
- Revision 1, 2018-08-08 (Pat Brown)
  - Internal revisions

## VK_NV_viewport_array2

#### Name String

`VK_NV_viewport_array2`

#### Extension Type

Device extension

#### Registered Extension Number

97

#### Revision

1

#### Ratification Status

Not ratified

#### Extension and Version Dependencies

None

### SPIR-V Dependencies

- SPV_NV_viewport_array2

### Contact

- Daniel Koch [GitHub]dgkoch

### Other Extension Metadata

### Last Modified Date

2017-02-15

### Interactions and External Dependencies

- This extension provides API support for `GL_NV_viewport_array2`
- This extension requires the `geometryShader` and `multiViewport` features.
- This extension interacts with the `tessellationShader` feature.

### Contributors

- Piers Daniell, NVIDIA
- Jeff Bolz, NVIDIA

### Description

This extension adds support for the following SPIR-V extension in Vulkan:

- `SPV_NV_viewport_array2`

which allows a single primitive to be broadcast to multiple viewports and/or multiple layers. A new shader built-in output `ViewportMaskNV` is provided, which allows a single primitive to be output to multiple viewports simultaneously. Also, a new SPIR-V decoration is added to control whether the effective viewport index is added into the variable decorated with the `Layer` built-in decoration. These capabilities allow a single primitive to be output to multiple layers simultaneously.

This extension allows variables decorated with the `Layer` and `ViewportIndex` built-ins to be exported from vertex or tessellation shaders, using the `ShaderViewportIndexLayerNV` capability.

This extension adds a new `ViewportMaskNV` built-in decoration that is available for output variables in vertex, tessellation evaluation, and geometry shaders, and a new `ViewportRelativeNV` decoration that can be added on variables decorated with `Layer` when using the `ShaderViewportMaskNV` capability.

When using GLSL source-based shading languages, the `gl_ViewportMask`[] built-in output variable and `viewport_relative` layout qualifier from `GL_NV_viewport_array2` map to the `ViewportMaskNV` and `ViewportRelativeNV` decorations, respectively. Behavior is described in the `GL_NV_viewport_array2` extension specification.

| | |
|---|---|
| **NOTE** | The `ShaderViewportIndexLayerNV` capability is equivalent to the `ShaderViewportIndexLayerEXT` capability added by `VK_EXT_shader_viewport_index_layer`. |

### New Enum Constants

- `VK_NV_VIEWPORT_ARRAY2_EXTENSION_NAME`
- `VK_NV_VIEWPORT_ARRAY2_SPEC_VERSION`
- `VK_NV_VIEWPORT_ARRAY_2_EXTENSION_NAME`
- `VK_NV_VIEWPORT_ARRAY_2_SPEC_VERSION`

### New or Modified Built-In Variables

- (modified) `Layer`
- (modified) `ViewportIndex`
- `ViewportMaskNV`

### New Variable Decoration

- `ViewportRelativeNV` in `Layer`

### New SPIR-V Capabilities

- `ShaderViewportIndexLayerNV`
- `ShaderViewportMaskNV`

### Version History

- Revision 1, 2017-02-15 (Daniel Koch)
  - Internal revisions

## VK_NV_viewport_swizzle

### Name String

`VK_NV_viewport_swizzle`

### Extension Type

Device extension

### Registered Extension Number

99

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

None

### Contact

- Piers Daniell [GitHub]pdaniell-nv

### Other Extension Metadata

### Last Modified Date

2016-12-22

### Interactions and External Dependencies

- This extension requires `multiViewport` and `geometryShader` features to be useful.

### Contributors

- Daniel Koch, NVIDIA
- Jeff Bolz, NVIDIA

### Description

This extension provides a new per-viewport swizzle that can modify the position of primitives sent to each viewport. New viewport swizzle state is added for each viewport, and a new position vector is computed for each vertex by selecting from and optionally negating any of the four components of the original position vector.

This new viewport swizzle is useful for a number of algorithms, including single-pass cube map rendering (broadcasting a primitive to multiple faces and reorienting the vertex position for each face) and voxel rasterization. The per-viewport component remapping and negation provided by the swizzle allows application code to re-orient three-dimensional geometry with a view along any of the **X**, **Y**, or **Z** axes. If a perspective projection and depth buffering is required, 1/W buffering should be used, as described in the single-pass cube map rendering example in the “Issues” section below.

### New Structures

- VkViewportSwizzleNV
- Extending VkPipelineViewportStateCreateInfo:
  - VkPipelineViewportSwizzleStateCreateInfoNV

### New Enums

- VkViewportCoordinateSwizzleNV

### New Bitmasks

- VkPipelineViewportSwizzleStateCreateFlagsNV

### New Enum Constants

- `VK_NV_VIEWPORT_SWIZZLE_EXTENSION_NAME`
- `VK_NV_VIEWPORT_SWIZZLE_SPEC_VERSION`

- Extending VkStructureType:
  - VK_STRUCTURE_TYPE_PIPELINE_VIEWPORT_SWIZZLE_STATE_CREATE_INFO_NV

**Issues**

1) Where does viewport swizzling occur in the pipeline?

**RESOLVED**: Despite being associated with the viewport, viewport swizzling must happen prior to the viewport transform. In particular, it needs to be performed before clipping and perspective division.

The viewport mask expansion (VK_NV_viewport_array2) and the viewport swizzle could potentially be performed before or after transform feedback, but feeding back several viewports worth of primitives with different swizzles does not seem particularly useful. This specification applies the viewport mask and swizzle after transform feedback, and makes primitive queries only count each primitive once.

2) Any interesting examples of how this extension, VK_NV_viewport_array2, and VK_NV_geometry_shader_passthrough can be used together in practice?

**RESOLVED**: One interesting use case for this extension is for single-pass rendering to a cube map. In this example, the application would attach a cube map texture to a layered FBO where the six cube faces are treated as layers. Vertices are sent through the vertex shader without applying a projection matrix, where the gl_Position output is (x,y,z,1) and the center of the cube map is at (0,0,0). With unextended Vulkan, one could have a conventional instanced geometry shader that looks something like the following:

```
layout(invocations = 6) in;     // separate invocation per face
layout(triangles) in;
layout(triangle_strip) out;
layout(max_vertices = 3) out;

in Inputs {
vec2 texcoord;
vec3 normal;
vec4 baseColor;
} v[];

    out Outputs {
    vec2 texcoord;
    vec3 normal;
    vec4 baseColor;
    };

    void main()
    {
    int face = gl_InvocationID;  // which face am I?

    // Project gl_Position for each vertex onto the cube map face.
    vec4 positions[3];
```

```
    for (int i = 0; i < 3; i++) {
        positions[i] = rotate(gl_in[i].gl_Position, face);
    }

    // If the primitive does not project onto this face, we are done.
    if (shouldCull(positions)) {
        return;
    }

    // Otherwise, emit a copy of the input primitive to the
    // appropriate face (using gl_Layer).
    for (int i = 0; i < 3; i++) {
        gl_Layer = face;
        gl_Position = positions[i];
        texcoord = v[i].texcoord;
        normal = v[i].normal;
        baseColor = v[i].baseColor;
        EmitVertex();
    }
}
```

With passthrough geometry shaders, this can be done using a much simpler shader:

```
layout(triangles) in;
layout(passthrough) in Inputs {
    vec2 texcoord;
    vec3 normal;
    vec4 baseColor;
}
layout(passthrough) in gl_PerVertex {
    vec4 gl_Position;
} gl_in[];
layout(viewport_relative) out int gl_Layer;

void main()
{
    // Figure out which faces the primitive projects onto and
    // generate a corresponding viewport mask.
    uint mask = 0;
    for (int i = 0; i < 6; i++) {
        if (!shouldCull(face)) {
        mask |= 1U << i;
        }
    }
    gl_ViewportMask = mask;
    gl_Layer = 0;
}
```

The application code is set up so that each of the six cube faces has a separate viewport (numbered

0 to 5). Each face also has a separate swizzle, programmed via the VkPipelineViewportSwizzleStateCreateInfoNV pipeline state. The viewport swizzle feature performs the coordinate transformation handled by the `rotate`() function in the original shader. The `viewport_relative` layout qualifier says that the viewport number (0 to 5) is added to the base `gl_Layer` value of 0 to determine which layer (cube face) the primitive should be sent to.

Note that the use of the passed through input `normal` in this example suggests that the fragment shader in this example would perform an operation like per-fragment lighting. The viewport swizzle would transform the position to be face-relative, but `normal` would remain in the original coordinate system. It seems likely that the fragment shader in either version of the example would want to perform lighting in the original coordinate system. It would likely do this by reconstructing the position of the fragment in the original coordinate system using `gl_FragCoord`, a constant or uniform holding the size of the cube face, and the input `gl_ViewportIndex` (or `gl_Layer`), which identifies the cube face. Since the value of `normal` is in the original coordinate system, it would not need to be modified as part of this coordinate transformation.

Note that while the `rotate`() operation in the regular geometry shader above could include an arbitrary post-rotation projection matrix, the viewport swizzle does not support arbitrary math. To get proper projection, 1/W buffering should be used. To do this:

1. Program the viewport swizzles to move the pre-projection W eye coordinate (typically 1.0) into the Z coordinate of the swizzle output and the eye coordinate component used for depth into the W coordinate. For example, the viewport corresponding to the +Z face might use a swizzle of (+X, -Y, +W, +Z). The Z normalized device coordinate computed after swizzling would then be $z'/w' = 1/Z_{eye}$.
2. On NVIDIA implementations supporting floating-point depth buffers with values outside [0,1], prevent unwanted near plane clipping by enabling `depthClampEnable`. Ensure that the depth clamp does not mess up depth testing by programming the depth range to very large values, such as `minDepthBounds`=-z, `maxDepthBounds`=+z, where $z = 2^{127}$. It should be possible to use IEEE infinity encodings also (`0xFF800000` for `-INF`, `0x7F800000` for `+INF`). Even when near/far clipping is disabled, primitives extending behind the eye will still be clipped because one or more vertices will have a negative W coordinate and fail X/Y clipping tests.

   On other implementations, scale X, Y, and Z eye coordinates so that vertices on the near plane have a post-swizzle W coordinate of 1.0. For example, if the near plane is at $Z_{eye} = 1/256$, scale X, Y, and Z by 256.
3. Adjust depth testing to reflect the fact that 1/W values are large near the eye and small away from the eye. Clear the depth buffer to zero (infinitely far away) and use a depth test of `VK_COMPARE_OP_GREATER` instead of `VK_COMPARE_OP_LESS`.

**Version History**

- Revision 1, 2016-12-22 (Piers Daniell)
  - Internal revisions

## VK_NVX_binary_import

### Name String

`VK_NVX_binary_import`

### Extension Type

Device extension

### Registered Extension Number

30

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

None

### API Interactions

- Interacts with VK_EXT_debug_report

### Contact

- Eric Werness [GitHub]ewerness-nv
- Liam Middlebrook [GitHub]liam-middlebrook

### Other Extension Metadata

### Last Modified Date

2021-04-09

### Contributors

- Eric Werness, NVIDIA
- Liam Middlebrook, NVIDIA

### Description

This extension allows applications to import CuBIN binaries and execute them.

| | |
|---|---|
| **NOTE** | There is currently no specification language written for this extension. The links to APIs defined by the extension are to stubs that only include generated content such as API declarations and implicit valid usage statements. |

### New Object Types

- VkCuFunctionNVX
- VkCuModuleNVX

### New Commands

- vkCmdCuLaunchKernelNVX
- vkCreateCuFunctionNVX
- vkCreateCuModuleNVX
- vkDestroyCuFunctionNVX
- vkDestroyCuModuleNVX

### New Structures

- VkCuFunctionCreateInfoNVX
- VkCuLaunchInfoNVX
- VkCuModuleCreateInfoNVX

### New Enum Constants

- `VK_NVX_BINARY_IMPORT_EXTENSION_NAME`
- `VK_NVX_BINARY_IMPORT_SPEC_VERSION`
- Extending VkObjectType:
  - `VK_OBJECT_TYPE_CU_FUNCTION_NVX`
  - `VK_OBJECT_TYPE_CU_MODULE_NVX`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_CU_FUNCTION_CREATE_INFO_NVX`
  - `VK_STRUCTURE_TYPE_CU_LAUNCH_INFO_NVX`
  - `VK_STRUCTURE_TYPE_CU_MODULE_CREATE_INFO_NVX`

If VK_EXT_debug_report is supported:

- Extending VkDebugReportObjectTypeEXT:
  - `VK_DEBUG_REPORT_OBJECT_TYPE_CU_FUNCTION_NVX_EXT`
  - `VK_DEBUG_REPORT_OBJECT_TYPE_CU_MODULE_NVX_EXT`

### Stub API References

There is currently no specification language written for this type. This section acts only as placeholder and to avoid dead links in the specification and reference pages.

```
// Provided by VK_NVX_binary_import
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkCuFunctionNVX)
```

There is currently no specification language written for this type. This section acts only as placeholder and to avoid dead links in the specification and reference pages.

```
// Provided by VK_NVX_binary_import
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkCuModuleNVX)
```

There is currently no specification language written for this command. This section acts only as placeholder and to avoid dead links in the specification and reference pages.

```
// Provided by VK_NVX_binary_import
VkResult vkCreateCuFunctionNVX(
    VkDevice                                    device,
    const VkCuFunctionCreateInfoNVX*            pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkCuFunctionNVX*                            pFunction);
```

## Valid Usage (Implicit)

- VUID-vkCreateCuFunctionNVX-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkCreateCuFunctionNVX-pCreateInfo-parameter
  `pCreateInfo` **must** be a valid pointer to a valid VkCuFunctionCreateInfoNVX structure
- VUID-vkCreateCuFunctionNVX-pAllocator-parameter
  If `pAllocator` is not `NULL`, `pAllocator` **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkCreateCuFunctionNVX-pFunction-parameter
  `pFunction` **must** be a valid pointer to a VkCuFunctionNVX handle

## Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_INITIALIZATION_FAILED`

There is currently no specification language written for this type. This section acts only as placeholder and to avoid dead links in the specification and reference pages.

```
// Provided by VK_NVX_binary_import
typedef struct VkCuFunctionCreateInfoNVX {
    VkStructureType    sType;
    const void*        pNext;
    VkCuModuleNVX      module;
    const char*        pName;
} VkCuFunctionCreateInfoNVX;
```

### Valid Usage (Implicit)

- VUID-VkCuFunctionCreateInfoNVX-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_CU_FUNCTION_CREATE_INFO_NVX`
- VUID-VkCuFunctionCreateInfoNVX-pNext-pNext
  `pNext` **must** be `NULL`
- VUID-VkCuFunctionCreateInfoNVX-module-parameter
  `module` **must** be a valid VkCuModuleNVX handle
- VUID-VkCuFunctionCreateInfoNVX-pName-parameter
  `pName` **must** be a null-terminated UTF-8 string

There is currently no specification language written for this command. This section acts only as placeholder and to avoid dead links in the specification and reference pages.

```
// Provided by VK_NVX_binary_import
void vkDestroyCuFunctionNVX(
    VkDevice                                    device,
    VkCuFunctionNVX                             function,
    const VkAllocationCallbacks*                pAllocator);
```

### Valid Usage (Implicit)

- VUID-vkDestroyCuFunctionNVX-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkDestroyCuFunctionNVX-function-parameter
  `function` **must** be a valid VkCuFunctionNVX handle
- VUID-vkDestroyCuFunctionNVX-pAllocator-parameter
  If `pAllocator` is not `NULL`, `pAllocator` **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkDestroyCuFunctionNVX-function-parent
  `function` **must** have been created, allocated, or retrieved from `device`

There is currently no specification language written for this command. This section acts only as placeholder and to avoid dead links in the specification and reference pages.

```
// Provided by VK_NVX_binary_import
VkResult vkCreateCuModuleNVX(
    VkDevice                                    device,
    const VkCuModuleCreateInfoNVX*              pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkCuModuleNVX*                              pModule);
```

## Valid Usage (Implicit)

- VUID-vkCreateCuModuleNVX-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkCreateCuModuleNVX-pCreateInfo-parameter
  `pCreateInfo` **must** be a valid pointer to a valid VkCuModuleCreateInfoNVX structure
- VUID-vkCreateCuModuleNVX-pAllocator-parameter
  If `pAllocator` is not `NULL`, `pAllocator` **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkCreateCuModuleNVX-pModule-parameter
  `pModule` **must** be a valid pointer to a VkCuModuleNVX handle

## Return Codes

**Success**

- `VK_SUCCESS`

**Failure**

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_INITIALIZATION_FAILED`

There is currently no specification language written for this type. This section acts only as placeholder and to avoid dead links in the specification and reference pages.

```
// Provided by VK_NVX_binary_import
typedef struct VkCuModuleCreateInfoNVX {
    VkStructureType    sType;
    const void*        pNext;
    size_t             dataSize;
    const void*        pData;
} VkCuModuleCreateInfoNVX;
```

## Valid Usage (Implicit)

- VUID-VkCuModuleCreateInfoNVX-sType-sType

sType **must** be VK_STRUCTURE_TYPE_CU_MODULE_CREATE_INFO_NVX

- VUID-VkCuModuleCreateInfoNVX-pNext-pNext
  pNext **must** be NULL
- VUID-VkCuModuleCreateInfoNVX-pData-parameter
  If dataSize is not 0, pData **must** be a valid pointer to an array of dataSize bytes

There is currently no specification language written for this command. This section acts only as placeholder and to avoid dead links in the specification and reference pages.

```
// Provided by VK_NVX_binary_import
void vkDestroyCuModuleNVX(
    VkDevice                                    device,
    VkCuModuleNVX                               module,
    const VkAllocationCallbacks*                pAllocator);
```

## Valid Usage (Implicit)

- VUID-vkDestroyCuModuleNVX-device-parameter
  device **must** be a valid VkDevice handle
- VUID-vkDestroyCuModuleNVX-module-parameter
  module **must** be a valid VkCuModuleNVX handle
- VUID-vkDestroyCuModuleNVX-pAllocator-parameter
  If pAllocator is not NULL, pAllocator **must** be a valid pointer to a valid VkAllocationCallbacks structure
- VUID-vkDestroyCuModuleNVX-module-parent
  module **must** have been created, allocated, or retrieved from device

There is currently no specification language written for this command. This section acts only as placeholder and to avoid dead links in the specification and reference pages.

```
// Provided by VK_NVX_binary_import
void vkCmdCuLaunchKernelNVX(
    VkCommandBuffer                             commandBuffer,
    const VkCuLaunchInfoNVX*                    pLaunchInfo);
```

## Valid Usage (Implicit)

- VUID-vkCmdCuLaunchKernelNVX-commandBuffer-parameter
  commandBuffer **must** be a valid VkCommandBuffer handle
- VUID-vkCmdCuLaunchKernelNVX-pLaunchInfo-parameter
  pLaunchInfo **must** be a valid pointer to a valid VkCuLaunchInfoNVX structure

- VUID-vkCmdCuLaunchKernelNVX-commandBuffer-recording
  `commandBuffer` **must** be in the recording state
- VUID-vkCmdCuLaunchKernelNVX-commandBuffer-cmdpool
  The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics, or compute operations
- VUID-vkCmdCuLaunchKernelNVX-videocoding
  This command **must** only be called outside of a video coding scope

## Host Synchronization

- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

## Command Properties

| Command Buffer Levels | Render Pass Scope | Video Coding Scope | Supported Queue Types | Command Type |
|---|---|---|---|---|
| Primary<br>Secondary | Both | Outside | Graphics<br>Compute | Action |

There is currently no specification language written for this type. This section acts only as placeholder and to avoid dead links in the specification and reference pages.

```
// Provided by VK_NVX_binary_import
typedef struct VkCuLaunchInfoNVX {
    VkStructureType        sType;
    const void*            pNext;
    VkCuFunctionNVX        function;
    uint32_t               gridDimX;
    uint32_t               gridDimY;
    uint32_t               gridDimZ;
    uint32_t               blockDimX;
    uint32_t               blockDimY;
    uint32_t               blockDimZ;
    uint32_t               sharedMemBytes;
    size_t                 paramCount;
    const void* const *    pParams;
    size_t                 extraCount;
    const void* const *    pExtras;
} VkCuLaunchInfoNVX;
```

### Valid Usage (Implicit)

- VUID-VkCuLaunchInfoNVX-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_CU_LAUNCH_INFO_NVX`
- VUID-VkCuLaunchInfoNVX-pNext-pNext
  `pNext` **must** be `NULL`
- VUID-VkCuLaunchInfoNVX-function-parameter
  `function` **must** be a valid VkCuFunctionNVX handle
- VUID-VkCuLaunchInfoNVX-pParams-parameter
  If `paramCount` is not `0`, `pParams` **must** be a valid pointer to an array of `paramCount` bytes
- VUID-VkCuLaunchInfoNVX-pExtras-parameter
  If `extraCount` is not `0`, `pExtras` **must** be a valid pointer to an array of `extraCount` bytes

#### Version History

- Revision 1, 2021-04-09 (Eric Werness)
  - Internal revisions

## VK_NVX_image_view_handle

#### Name String

`VK_NVX_image_view_handle`

#### Extension Type

Device extension

#### Registered Extension Number

31

#### Revision

2

#### Ratification Status

Not ratified

#### Extension and Version Dependencies

None

#### Contact

- Eric Werness [GitHub]ewerness-nv

#### Other Extension Metadata

#### Last Modified Date

2020-04-03

#### Contributors

- Eric Werness, NVIDIA
- Jeff Bolz, NVIDIA
- Daniel Koch, NVIDIA

#### Description

This extension allows applications to query an opaque handle from an image view for use as a sampled image or storage image. This provides no direct functionality itself.

#### New Commands

- vkGetImageViewAddressNVX
- vkGetImageViewHandleNVX

#### New Structures

- VkImageViewAddressPropertiesNVX
- VkImageViewHandleInfoNVX

#### New Enum Constants

- `VK_NVX_IMAGE_VIEW_HANDLE_EXTENSION_NAME`
- `VK_NVX_IMAGE_VIEW_HANDLE_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_IMAGE_VIEW_ADDRESS_PROPERTIES_NVX`
  - `VK_STRUCTURE_TYPE_IMAGE_VIEW_HANDLE_INFO_NVX`

#### Version History

- Revision 2, 2020-04-03 (Piers Daniell)
  - Add vkGetImageViewAddressNVX
- Revision 1, 2018-12-07 (Eric Werness)
  - Internal revisions

### VK_NVX_multiview_per_view_attributes

#### Name String

`VK_NVX_multiview_per_view_attributes`

#### Extension Type

Device extension

#### Registered Extension Number

98

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_multiview
or
Vulkan Version 1.1

### SPIR-V Dependencies

- SPV_NVX_multiview_per_view_attributes

### Contact

- Jeff Bolz [GitHub]jeffbolznv

### Other Extension Metadata

### Last Modified Date

2017-01-13

### IP Status

No known IP claims.

### Interactions and External Dependencies

- This extension provides API support for `GL_NVX_multiview_per_view_attributes`
- This extension interacts with `VK_NV_viewport_array2`.

### Contributors

- Jeff Bolz, NVIDIA
- Daniel Koch, NVIDIA

### Description

This extension adds a new way to write shaders to be used with multiview subpasses, where the attributes for all views are written out by a single invocation of the pre-rasterization shader stages. Related SPIR-V and GLSL extensions `SPV_NVX_multiview_per_view_attributes` and `GL_NVX_multiview_per_view_attributes` introduce per-view position and viewport mask attributes arrays, and this extension defines how those per-view attribute arrays are interpreted by Vulkan. Pipelines using per-view attributes **may** only execute the pre-rasterization shader stages once for all views rather than once per-view, which reduces redundant shading work.

A subpass creation flag controls whether the subpass uses this extension. A subpass **must** either exclusively use this extension or not use it at all.

Some Vulkan implementations only support the position attribute varying between views in the X component. A subpass can declare via a second creation flag whether all pipelines compiled for this subpass will obey this restriction.

Shaders that use the new per-view outputs (e.g. `gl_PositionPerViewNV`) **must** also write the non-per-view output (`gl_Position`), and the values written **must** be such that `gl_Position = gl_PositionPerViewNV[gl_ViewIndex]` for all views in the subpass. Implementations are free to either use the per-view outputs or the non-per-view outputs, whichever would be more efficient.

If `VK_NV_viewport_array2` is not also supported and enabled, the per-view viewport mask **must** not be used.

### New Structures

- Extending VkPhysicalDeviceProperties2:
  - VkPhysicalDeviceMultiviewPerViewAttributesPropertiesNVX

### New Enum Constants

- `VK_NVX_MULTIVIEW_PER_VIEW_ATTRIBUTES_EXTENSION_NAME`
- `VK_NVX_MULTIVIEW_PER_VIEW_ATTRIBUTES_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MULTIVIEW_PER_VIEW_ATTRIBUTES_PROPERTIES_NVX`
- Extending VkSubpassDescriptionFlagBits:
  - `VK_SUBPASS_DESCRIPTION_PER_VIEW_ATTRIBUTES_BIT_NVX`
  - `VK_SUBPASS_DESCRIPTION_PER_VIEW_POSITION_X_ONLY_BIT_NVX`

### New Built-In Variables

- `PositionPerViewNV`
- `ViewportMaskPerViewNV`

### New SPIR-V Capabilities

- `PerViewAttributesNV`

### Examples

```
#version 450 core

#extension GL_KHX_multiview : enable
#extension GL_NVX_multiview_per_view_attributes : enable

layout(location = 0) in vec4 position;
layout(set = 0, binding = 0) uniform Block { mat4 mvpPerView[2]; } buf;

void main()
{
    // Output both per-view positions and gl_Position as a function
    // of gl_ViewIndex
    gl_PositionPerViewNV[0] = buf.mvpPerView[0] * position;
```

```
    gl_PositionPerViewNV[1] = buf.mvpPerView[1] * position;
    gl_Position = buf.mvpPerView[gl_ViewIndex] * position;
}
```

#### Version History

- Revision 1, 2017-01-13 (Jeff Bolz)
  - Internal revisions

## VK_QCOM_filter_cubic_clamp

#### Name String

`VK_QCOM_filter_cubic_clamp`

#### Extension Type

Device extension

#### Registered Extension Number

522

#### Revision

1

#### Ratification Status

Not ratified

#### Extension and Version Dependencies

VK_EXT_filter_cubic
and
  Vulkan Version 1.2
  or
  VK_EXT_sampler_filter_minmax

#### Contact

- Matthew Netsch [GitHub]mnetsch

#### Other Extension Metadata

#### Last Modified Date

2023-08-02

#### Contributors

- Jeff Leger, Qualcomm Technologies, Inc.

#### Description

This extension extends cubic filtering by adding the ability to enable an anti-ringing clamp. Cubic filtering samples from a 4x4 region of texels and computes a cubic weighted average of the region.

In some cases, the resulting value is outside the range of any of the texels in the 4x4 region. This is sometimes referred to as “filter overshoot” or “filter ringing” and can occur when there is a sharp discontinuity in the 4x4 region being filtered. For some use cases this “ringing” can produces unacceptable artifacts.

The solution to the ringing problem is to clamp the post-cubic-filtered value to be within the max and min of texel values in the 4x4 region. While such “range clamping” can be performed in shader code, the additional texture fetches and clamping ALU operations can be costly.

Certain Adreno GPUs are able to perform the range clamp in the texture unit during cubic filtering at significant performance/power savings versus a shader-based clamping approach. This extension exposes such hardware functionality.

This extension extends VkSamplerReductionMode, adding `VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE_RANGECLAMP_QCOM` which enables the range clamp operation.

#### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceCubicClampFeaturesQCOM

#### New Enum Constants

- `VK_QCOM_FILTER_CUBIC_CLAMP_EXTENSION_NAME`
- `VK_QCOM_FILTER_CUBIC_CLAMP_SPEC_VERSION`
- Extending VkSamplerReductionMode:
  - `VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE_RANGECLAMP_QCOM`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_CUBIC_CLAMP_FEATURES_QCOM`

#### Version History

- Revision 1, 2023-08-02 (jleger)
  - Initial version

## VK_QCOM_filter_cubic_weights

#### Name String

`VK_QCOM_filter_cubic_weights`

#### Extension Type

Device extension

#### Registered Extension Number

520

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_EXT_filter_cubic

### Contact

- Matthew Netsch [GitHub]mnetsch

### Other Extension Metadata

### Last Modified Date

2023-06-23

### Contributors

- Jeff Leger, Qualcomm Technologies, Inc.
- Jonathan Wicks, Qualcomm Technologies, Inc.

### Description

This extension extends cubic filtering by adding the ability to select a set of weights. Without this extension, the weights used in cubic filtering are limited to those corresponding to a Catmull-Rom spline. This extension adds support for 3 additional spline weights.

This extension adds a new structure that **can** be added to the `pNext` chain of VkSamplerCreateInfo that **can** be used to specify which set of cubic weights are used in cubic filtering. A similar structure can be added to the `pNext` chain of VkBlitImageInfo2 to specify cubic weights used in a blit operation.

With this extension weights corresponding to the following additional splines can be selected for cubic filtered sampling and blits:

- Zero Tangent Cardinal
- B-Spline
- Mitchell-Netravali

### New Structures

- Extending VkBlitImageInfo2:
  - VkBlitImageCubicWeightsInfoQCOM
- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceCubicWeightsFeaturesQCOM
- Extending VkSamplerCreateInfo:

- VkSamplerCubicWeightsCreateInfoQCOM

### New Enums

- VkCubicFilterWeightsQCOM

### New Enum Constants

- `VK_QCOM_FILTER_CUBIC_WEIGHTS_EXTENSION_NAME`
- `VK_QCOM_FILTER_CUBIC_WEIGHTS_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_BLIT_IMAGE_CUBIC_WEIGHTS_INFO_QCOM`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_CUBIC_WEIGHTS_FEATURES_QCOM`
  - `VK_STRUCTURE_TYPE_SAMPLER_CUBIC_WEIGHTS_CREATE_INFO_QCOM`

### Version History

- Revision 1, 2023-06-23 (jleger)
  - Initial version

## VK_QCOM_fragment_density_map_offset

### Name String

`VK_QCOM_fragment_density_map_offset`

### Extension Type

Device extension

### Registered Extension Number

426

### Revision

2

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1
and
VK_EXT_fragment_density_map

### Contact

- Matthew Netsch [GitHub]mnetsch

### Other Extension Metadata

#### Last Modified Date

2024-06-17

#### Contributors

- Matthew Netsch, Qualcomm Technologies, Inc.
- Jonathan Wicks, Qualcomm Technologies, Inc.
- Jonathan Tinkham, Qualcomm Technologies, Inc.
- Jeff Leger, Qualcomm Technologies, Inc.
- Manan Katwala, Qualcomm Technologies, Inc.

#### Description

This extension allows an application to specify offsets to a fragment density map attachment, changing the framebuffer location where density values are applied to without having to regenerate the fragment density map.

#### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceFragmentDensityMapOffsetFeaturesQCOM
- Extending VkPhysicalDeviceProperties2:
  - VkPhysicalDeviceFragmentDensityMapOffsetPropertiesQCOM
- Extending VkSubpassEndInfo:
  - VkSubpassFragmentDensityMapOffsetEndInfoQCOM

#### New Enum Constants

- `VK_QCOM_FRAGMENT_DENSITY_MAP_OFFSET_EXTENSION_NAME`
- `VK_QCOM_FRAGMENT_DENSITY_MAP_OFFSET_SPEC_VERSION`
- Extending VkImageCreateFlagBits:
  - `VK_IMAGE_CREATE_FRAGMENT_DENSITY_MAP_OFFSET_BIT_QCOM`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_FRAGMENT_DENSITY_MAP_OFFSET_FEATURES_QCOM`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_FRAGMENT_DENSITY_MAP_OFFSET_PROPERTIES_QCOM`
  - `VK_STRUCTURE_TYPE_SUBPASS_FRAGMENT_DENSITY_MAP_OFFSET_END_INFO_QCOM`

#### Version History

- Revision 2, 2024-06-17 (Matthew Netsch)
  - Fix typo in spec regarding fragmentDensityMapOffset feature

- Revision 1, 2021-09-03 (Matthew Netsch)
  - Initial version

## VK_QCOM_image_processing

### Name String

`VK_QCOM_image_processing`

### Extension Type

Device extension

### Registered Extension Number

441

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_format_feature_flags2
or
Vulkan Version 1.3

### API Interactions

- Interacts with VK_VERSION_1_3
- Interacts with VK_KHR_format_feature_flags2

### SPIR-V Dependencies

- SPV_QCOM_image_processing

### Contact

- Matthew Netsch [GitHub]mnetsch

### Extension Proposal

VK_QCOM_image_processing

### Other Extension Metadata

### Last Modified Date

2022-07-08

### Interactions and External Dependencies

- This extension provides API support for `GL_QCOM_image_processing`

### Contributors

- Jeff Leger, Qualcomm Technologies, Inc.
- Ruihao Zhang, Qualcomm Technologies, Inc.

### Description

GPUs are commonly used to process images for various applications from 3D graphics to UI and from composition to compute applications. Simple scaling and filtering can be done with bilinear filtering, which comes for free during texture sampling. However, as screen sizes get larger and more use cases rely on GPU such as camera and video post-processing needs, there is increasing demand for GPU to support higher order filtering and other advanced image processing.

This extension introduces a new set of SPIR-V built-in functions for image processing. It exposes the following new imaging operations

- The `OpImageSampleWeightedQCOM` instruction takes 3 operands: *sampled image*, *weight image*, and texture coordinates. The instruction computes a weighted average of an MxN region of texels in the *sampled image*, using a set of MxN weights in the *weight image*.
- The `OpImageBoxFilterQCOM` instruction takes 3 operands: *sampled image*, *box size*, and texture coordinates. Note that *box size* specifies a floating-point width and height in texels. The instruction computes a weighted average of all texels in the *sampled image* that are covered (either partially or fully) by a box with the specified size and centered at the specified texture coordinates.
- The `OpImageBlockMatchSADQCOM` and `OpImageBlockMatchSSDQCOM` instructions each takes 5 operands: *target image*, *target coordinates*, *reference image*, *reference coordinates*, and *block size*. Each instruction computes an error metric, that describes whether a block of texels in the *target image* matches a corresponding block of texels in the *reference image*. The error metric is computed per-component. `OpImageBlockMatchSADQCOM` computes "Sum Of Absolute Difference" and `OpImageBlockMatchSSDQCOM` computes "Sum of Squared Difference".

Each of the image processing instructions operate only on 2D images. The instructions do not-support sampling of mipmap, multi-plane, multi-layer, multi-sampled, or depth/stencil images. The instructions can be used in any shader stage.

Implementations of this extension should support these operations natively at the HW instruction level, offering potential performance gains as well as ease of development.

### New Structures

- Extending VkImageViewCreateInfo:
  - VkImageViewSampleWeightCreateInfoQCOM
- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceImageProcessingFeaturesQCOM
- Extending VkPhysicalDeviceProperties2:
  - VkPhysicalDeviceImageProcessingPropertiesQCOM

### New Enum Constants

- `VK_QCOM_IMAGE_PROCESSING_EXTENSION_NAME`
- `VK_QCOM_IMAGE_PROCESSING_SPEC_VERSION`

- Extending VkDescriptorType:
  - VK_DESCRIPTOR_TYPE_BLOCK_MATCH_IMAGE_QCOM
  - VK_DESCRIPTOR_TYPE_SAMPLE_WEIGHT_IMAGE_QCOM
- Extending VkImageUsageFlagBits:
  - VK_IMAGE_USAGE_SAMPLE_BLOCK_MATCH_BIT_QCOM
  - VK_IMAGE_USAGE_SAMPLE_WEIGHT_BIT_QCOM
- Extending VkSamplerCreateFlagBits:
  - VK_SAMPLER_CREATE_IMAGE_PROCESSING_BIT_QCOM
- Extending VkStructureType:
  - VK_STRUCTURE_TYPE_IMAGE_VIEW_SAMPLE_WEIGHT_CREATE_INFO_QCOM
  - VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_IMAGE_PROCESSING_FEATURES_QCOM
  - VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_IMAGE_PROCESSING_PROPERTIES_QCOM

If VK_KHR_format_feature_flags2 or Vulkan Version 1.3 is supported:

- Extending VkFormatFeatureFlagBits2:
  - VK_FORMAT_FEATURE_2_BLOCK_MATCHING_BIT_QCOM
  - VK_FORMAT_FEATURE_2_BOX_FILTER_SAMPLED_BIT_QCOM
  - VK_FORMAT_FEATURE_2_WEIGHT_IMAGE_BIT_QCOM
  - VK_FORMAT_FEATURE_2_WEIGHT_SAMPLED_IMAGE_BIT_QCOM

#### Version History

- Revision 1, 2022-07-08 (Jeff Leger)

## VK_QCOM_image_processing2

#### Name String

VK_QCOM_image_processing2

#### Extension Type

Device extension

#### Registered Extension Number

519

#### Revision

1

#### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_QCOM_image_processing

### SPIR-V Dependencies

- SPV_QCOM_image_processing2

### Contact

- Matthew Netsch [GitHub]mnetsch

### Other Extension Metadata

### Last Modified Date

2023-03-10

### Interactions and External Dependencies

- This extension provides API support for `GL_QCOM_image_processing2`

### Contributors

- Jeff Leger, Qualcomm Technologies, Inc.

### Description

This extension enables support for the SPIR-V `TextureBlockMatch2QCOM` capability. It builds on the functionality of QCOM_image_processing with the addition of 4 new image processing operations.

- The `opImageBlockMatchWindowSADQCOM` SPIR-V instruction builds upon the functionality of `opImageBlockMatchSADQCOM` by repeatedly performing block match operations across a 2D window. The "2D windowExtent" and "compareMode" are specified by VkSamplerBlockMatchWindowCreateInfoQCOM in the sampler used to create the *target image*. Like `OpImageBlockMatchSADQCOM`, `opImageBlockMatchWindowSADQCOM` computes an error metric, that describes whether a block of texels in the *target image* matches a corresponding block of texels in the *reference image*. Unlike `OpImageBlockMatchSADQCOM`, this instruction computes an error metric at each (X,Y) location within the 2D window and returns either the minimum or maximum error. The instruction only supports single-component formats. Refer to the pseudocode below for details.
- The `opImageBlockMatchWindowSSDQCOM` follows the same pattern, computing the SSD error metric at each location within the 2D window.
- The `opImageBlockMatchGatherSADQCOM` builds upon `OpImageBlockMatchSADQCOM`. This instruction computes an error metric, that describes whether a block of texels in the *target image* matches a corresponding block of texels in the *reference image*. The instruction computes the SAD error metric at 4 texel offsets and returns the error metric for each offset in the X,Y,Z,and W components. The instruction only supports single-component texture formats. Refer to the pseudocode below for details.
- The `opImageBlockMatchGatherSSDQCOM` follows the same pattern, computing the SSD error metric for 4 offsets.

Each of the above 4 image processing instructions are limited to single-component formats.

Below is the pseudocode for GLSL built-in function `textureWindowBlockMatchSADQCOM`. The pseudocode for `textureWindowBlockMatchSSD` is identical other than replacing all instances of `"SAD"` with `"SSD"`.

```
vec4 textureBlockMatchWindowSAD( sampler2D target,
                                 uvec2 targetCoord,
                                 samler2D reference,
                                 uvec2 refCoord,
                                 uvec2 blocksize) {
    // compareMode (MIN or MAX) comes from the vkSampler associated with `target`
    // uvec2 window  comes from the vkSampler associated with `target`
    minSAD = INF;
    maxSAD = -INF;
    uvec2 minCoord;
    uvec2 maxCoord;

    for (uint x=0, x < window.width; x++) {
        for (uint y=0; y < window.height; y++) {
            float SAD = textureBlockMatchSAD(target,
                                             targetCoord + uvec2(x, y),
                                             reference,
                                             refCoord,
                                             blocksize).x;
            // Note: the below comparison operator will produce undefined results
            // if SAD is a denorm value.
            if (SAD < minSAD) {
                minSAD = SAD;
                minCoord = uvec2(x,y);
            }
            if (SAD > maxSAD) {
                maxSAD = SAD;
                maxCoord = uvec2(x,y);
            }
        }
    }
    if (compareMode=MIN) {
        return vec4(minSAD, minCoord.x, minCoord.y, 0.0);
    } else {
        return vec4(maxSAD, maxCoord.x, maxCoord.y, 0.0);
    }
}
```

Below is the pseudocode for `textureBlockMatchGatherSADQCOM`. The pseudocode for `textureBlockMatchGatherSSD` follows an identical pattern.

```
vec4 textureBlockMatchGatherSAD( sampler2D target,
                                 uvec2 targetCoord,
                                 samler2D reference,
                                 uvec2 refCoord,
```

```
                                uvec2 blocksize) {
    vec4 out;
    for (uint x=0, x<4; x++) {
            float SAD = textureBlockMatchSAD(target,
                                             targetCoord + uvec2(x, 0),
                                             reference,
                                             refCoord,
                                             blocksize).x;
            out[x] = SAD;
    }
    return out;
}
```

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceImageProcessing2FeaturesQCOM
- Extending VkPhysicalDeviceProperties2:
  - VkPhysicalDeviceImageProcessing2PropertiesQCOM
- Extending VkSamplerCreateInfo:
  - VkSamplerBlockMatchWindowCreateInfoQCOM

### New Enums

- VkBlockMatchWindowCompareModeQCOM

### New Enum Constants

- `VK_QCOM_IMAGE_PROCESSING_2_EXTENSION_NAME`
- `VK_QCOM_IMAGE_PROCESSING_2_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_IMAGE_PROCESSING_2_FEATURES_QCOM`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_IMAGE_PROCESSING_2_PROPERTIES_QCOM`
  - `VK_STRUCTURE_TYPE_SAMPLER_BLOCK_MATCH_WINDOW_CREATE_INFO_QCOM`

### Issues

1) What is the precision of the min/max comparison checks?

**RESOLVED**: Intermediate computations for the new operations are performed at 16-bit floating-point precision. If the value of `"float SAD"` in the above code sample is a 16-bit denorm value, then behavior of the MIN/MAX comparison is undefined.

### Version History

- Revision 1, 2023-03-10 (Jeff Leger)

## VK_QCOM_multiview_per_view_render_areas

### Name String

`VK_QCOM_multiview_per_view_render_areas`

### Extension Type

Device extension

### Registered Extension Number

511

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

None

### Contact

- Matthew Netsch [GitHub]mnetsch

### Other Extension Metadata

### Last Modified Date

2023-01-10

### IP Status

No known IP claims.

### Interactions and External Dependencies

- This extension interacts with `VK_KHR_dynamic_rendering`
- This extension interacts with `VK_QCOM_render_pass_transform`

### Contributors

- Jeff Leger, Qualcomm
- Jonathan Tinkham, Qualcomm
- Jonathan Wicks, Qualcomm

### Description

Certain use cases (e.g., side-by-side VR rendering) use multiview and render to distinct regions of the framebuffer for each view. On some implementations, there may be a performance benefit for providing per-view render areas to the implementation. Such per-view render areas can be used by the implementation to reduce the pixels that are affected by attachment load, store, and multisample resolve operations.

The extension enables a multiview render pass instance to define per-view render areas. For each view of a multiview render pass instance, only those pixels in the per-view render area are affected by load, store and resolve operations.

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceMultiviewPerViewRenderAreasFeaturesQCOM
- Extending VkRenderPassBeginInfo, VkRenderingInfo:
  - VkMultiviewPerViewRenderAreasRenderPassBeginInfoQCOM

### New Enum Constants

- `VK_QCOM_MULTIVIEW_PER_VIEW_RENDER_AREAS_EXTENSION_NAME`
- `VK_QCOM_MULTIVIEW_PER_VIEW_RENDER_AREAS_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_MULTIVIEW_PER_VIEW_RENDER_AREAS_RENDER_PASS_BEGIN_INFO_QCOM`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MULTIVIEW_PER_VIEW_RENDER_AREAS_FEATURES_QCOM`

### Issues

1) Do the per-view `renderAreas` interact with vkGetRenderAreaGranularity ?

**RESOLVED**: There is no change. The granularity returned by vkGetRenderAreaGranularity also applies to the per-view `renderAreas`.

2) How does this extension interact with `VK_QCOM_render_pass_transform`?

**RESOLVED**: When `VK_QCOM_render_pass_transform` is enabled, the application provides render area in non-rotated coordinates which is rotated by the implementation to the rotated coordinate system. When this extension is used in combination with `VK_QCOM_render_pass_transform`, then the `renderArea` provided in VkRenderingInfo::`renderArea`, VkRenderPassBeginInfo::`renderArea`, or VkCommandBufferInheritanceRenderPassTransformInfoQCOM::`renderArea` is rotated by the implementation. The per-view render areas are not rotated.

3) How does this extension interact with `VK_QCOM_multiview_per_view_viewports`

**RESOLVED**: There is no direct interaction. The per-view viewports and the per-view renderAreas are orthogonal features.

4) When a per-view `renderArea` is specified, must multiview rendering for each view of a multiview render pass be contained within the per-view `renderArea`?

**RESOLVED**: Yes, and the `VK_QCOM_multiview_per_view_viewports` may help here since it provides per-view scissors.

5) When per-view render areas are specified, what purpose if any do VkRenderPassBeginInfo::`renderArea` and VkRenderingInfo::`renderArea` serve?

**RESOLVED**: The per-view `renderArea` effectively overrides the per-renderpass `renderArea`. The per-view `renderArea` defines the regions of the attachments that are effected by load, store, and multisample resolve operations. A valid implementation could ignore the per-renderpass `renderArea`. However, as an aid to the implementation, the application must set the per-renderpass `renderArea` to an area that is at least as large as the union of all the per-view render areas. Pixels that are within the per-renderpass `renderArea` but not within any per-view render area must not be affected by load, store, or multisample resolve operations.

### Version History

- Revision 1, 2023-01-10 (Jeff Leger)

## VK_QCOM_multiview_per_view_viewports

### Name String

`VK_QCOM_multiview_per_view_viewports`

### Extension Type

Device extension

### Registered Extension Number

489

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### Contact

- Matthew Netsch [GitHub]mnetsch

### Other Extension Metadata

### Last Modified Date

2022-11-22

### IP Status

No known IP claims.

### Interactions and External Dependencies

- This extension interacts with `VK_KHR_dynamic_rendering`
- This extension interacts with `VK_EXT_extended_dynamic_state`

### Contributors

- Jeff Leger, Qualcomm
- Jonathan Tinkham, Qualcomm
- Jonathan Wicks, Qualcomm

### Description

Certain use cases for multiview have a need for specifying a separate viewport and scissor for each view, without using shader-based viewport indexing as introduced with `VK_EXT_shader_viewport_index_layer`.

This extension adds a new way to control ViewportIndex with multiview. When the `multiviewPerViewViewports` feature is enabled and if the last pre-rasterization shader entry point's interface does not use the `ViewportIndex` built-in decoration, then each view of a multiview render pass instance will use a viewport and scissor index equal to the `ViewIndex`.

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceMultiviewPerViewViewportsFeaturesQCOM

### New Enum Constants

- `VK_QCOM_MULTIVIEW_PER_VIEW_VIEWPORTS_EXTENSION_NAME`
- `VK_QCOM_MULTIVIEW_PER_VIEW_VIEWPORTS_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MULTIVIEW_PER_VIEW_VIEWPORTS_FEATURES_QCOM`

### Issues

1) Is it possible to enable/disable the `multiviewPerViewViewports` feature for individual render pass instances?

**RESOLVED**: No, when the multiviewPerViewViewports feature is enabled during vkCreateDevice, then all created render pass instances (including dynamic render passes from `VK_KHR_dynamic_rendering`) and all created VkPipelines will have the feature enabled. This approach was chosen because it simplifies application code and there is no known use case to enable/disable the feature for individual render passes or pipelines.

2) When this extension is used, is the value of `ViewportIndex` implicitly written by the last pre-rasterization shader stage and can the value of `ViewportIndex` be read in the fragment shader?

**RESOLVED**: No, use of the extension does not add an implicit write to `ViewportIndex` in any shader stage, and additionally, the value of `ViewportIndex` in the fragment shader is undefined.

### Version History

- Revision 1, 2022-11-22 (Jeff Leger)

## VK_QCOM_render_pass_shader_resolve

### Name String

`VK_QCOM_render_pass_shader_resolve`

### Extension Type

Device extension

### Registered Extension Number

172

### Revision

4

### Ratification Status

Not ratified

### Extension and Version Dependencies

None

### Contact

- Matthew Netsch [GitHub]mnetsch

### Other Extension Metadata

### Last Modified Date

2019-11-07

### IP Status

No known IP claims.

### Interactions and External Dependencies

None.

### Contributors

- Srihari Babu Alla, Qualcomm
- Bill Licea-Kane, Qualcomm
- Jeff Leger, Qualcomm

### Description

This extension allows a shader resolve to replace fixed-function resolve.

Fixed-function resolve is limited in function to simple filters of multisample buffers to a single sample buffer.

Fixed-function resolve is more performance efficient and/or power efficient than shader resolve for such simple filters.

Shader resolve allows a shader writer to create complex, non-linear filtering of a multisample buffer in the last subpass of a subpass dependency chain.

This extension also provides a bit which **can** be used to enlarge a sample region dependency to a fragment region dependency, so that a framebuffer-region dependency **can** replace a framebuffer-global dependency in some cases.

### New Enum Constants

- `VK_QCOM_RENDER_PASS_SHADER_RESOLVE_EXTENSION_NAME`
- `VK_QCOM_RENDER_PASS_SHADER_RESOLVE_SPEC_VERSION`
- Extending VkSubpassDescriptionFlagBits:
  - `VK_SUBPASS_DESCRIPTION_FRAGMENT_REGION_BIT_QCOM`
  - `VK_SUBPASS_DESCRIPTION_SHADER_RESOLVE_BIT_QCOM`

### Issues

1) Should this extension be named render_pass_shader_resolve?

**RESOLVED** Yes.

This is part of suite of small extensions to render pass.

Following the style guide, instead of following VK_KHR_create_renderpass2.

2) Should the VK_SAMPLE_COUNT_1_BIT be required for each pColorAttachment and the DepthStencilAttachent?

**RESOLVED** No.

While this may not be a common use case, and while most fixed-function resolve hardware has this limitation, there is little reason to require a shader resolve to resolve to a single sample buffer.

3) Should a shader resolve subpass be the last subpass in a render pass?

**RESOLVED** Yes.

To be more specific, it should be the last subpass in a subpass dependency chain.

4) Do we need the `VK_SUBPASS_DESCRIPTION_FRAGMENT_REGION_BIT_QCOM` bit?

**RESOLVED** Yes.

This applies when an input attachment's sample count is equal to `rasterizationSamples`. Further, if `sampleShading` is enabled (explicitly or implicitly) then `minSampleShading` **must** equal 0.0.

However, this bit may be set on any subpass, it is not restricted to a shader resolve subpass.

### Version History

- Revision 1, 2019-06-28 (wwlk)
  - Initial draft
- Revision 2, 2019-11-06 (wwlk)
  - General clean-up/spec updates
  - Added issues
- Revision 3, 2019-11-07 (wwlk)
  - Typos
  - Additional issues
  - Clarified that a shader resolve subpass is the last subpass in a subpass dependency chain
- Revision 4, 2020-01-06 (wwlk)
  - Change resolution of Issue 1 (*render_pass*, not *renderpass*)

## VK_QCOM_render_pass_store_ops

### Name String

`VK_QCOM_render_pass_store_ops`

### Extension Type

Device extension

### Registered Extension Number

302

### Revision

2

### Ratification Status

Not ratified

### Extension and Version Dependencies

None

### Contact

- Matthew Netsch [GitHub]mnetsch

### Other Extension Metadata

### Last Modified Date

2020-03-25

### Contributors

- Bill Licea-Kane, Qualcomm Technologies, Inc.

### Description

Render pass attachments **can** be read-only for the duration of a render pass.

Examples include input attachments and depth attachments where depth tests are enabled but depth writes are not enabled.

In such cases, there **can** be no contents generated for an attachment within the render area.

This extension adds a new VkAttachmentStoreOp `VK_ATTACHMENT_STORE_OP_NONE_QCOM` specifying that the contents within the render area **may** not be written to memory, but that the prior contents of the attachment in memory are preserved. However, if any contents were generated within the render area during rendering, the contents of the attachment will be undefined inside the render area.

| | |
|---|---|
| **NOTE** | The VkAttachmentStoreOp `VK_ATTACHMENT_STORE_OP_STORE` **may** force an implementation to assume that the attachment was written and force an implementation to flush data to memory or to a higher level cache. The VkAttachmentStoreOp `VK_ATTACHMENT_STORE_OP_NONE_QCOM` **may** allow an implementation to assume that the attachment was not written and allow an implementation to avoid such a flush. |

### New Enum Constants

- `VK_QCOM_RENDER_PASS_STORE_OPS_EXTENSION_NAME`
- `VK_QCOM_RENDER_PASS_STORE_OPS_SPEC_VERSION`
- Extending VkAttachmentStoreOp:
  - `VK_ATTACHMENT_STORE_OP_NONE_QCOM`

### Version History

- Revision 1, 2019-12-20 (wwlk)
  - Initial version
- Revision 2, 2020-03-25 (wwlk)
  - Minor renaming

## VK_QCOM_render_pass_transform

### Name String

`VK_QCOM_render_pass_transform`

### Extension Type

Device extension

### Registered Extension Number

283

**Revision**

4

**Ratification Status**

Not ratified

**Extension and Version Dependencies**

None

**Contact**

- Matthew Netsch [GitHub]mnetsch

**Other Extension Metadata**

**Last Modified Date**

2023-12-13

**Interactions and External Dependencies**

- This extension interacts with `VK_KHR_swapchain`
- This extension interacts with `VK_KHR_surface`
- This extension interacts with `VK_EXT_fragment_density_map`
- This extension interacts with `VK_KHR_fragment_shading_rate`
- This extension interacts with `VK_QCOM_tile_properties`

**Contributors**

- Jeff Leger, Qualcomm Technologies, Inc.
- Brandon Light, Qualcomm Technologies, Inc.
- Matthew Netsch, Qualcomm Technologies, Inc.
- Arpit Agarwal, Qualcomm Technologies, Inc.

**Description**

This extension provides a mechanism for applications to enable driver support for render pass transform.

Mobile devices can be rotated and mobile applications need to render properly when a device is held in a landscape or portrait orientation. When the current orientation differs from the device's native orientation, a rotation is required so that the "up" direction of the rendered scene matches the current orientation.

If the Display Processing Unit (DPU) does not natively support rotation, the Vulkan presentation engine can handle this rotation in a separate composition pass. Alternatively, the application can render frames "pre-rotated" to avoid this extra pass. The latter is preferred to reduce power consumption and achieve the best performance because it avoids tasking the GPU with extra work to perform the copy/rotate operation.

Unlike OpenGL ES, the burden of pre-rotation in Vulkan falls on the application. To implement pre-

rotation, applications render into swapchain images matching the device native aspect ratio of the display and “pre-rotate” the rendering content to match the device’s current orientation. The burden is more than adjusting the Model View Projection (MVP) matrix in the vertex shader to account for rotation and aspect ratio. The coordinate systems of scissors, viewports, derivatives and several shader built-ins may need to be adapted to produce the correct result.

It is difficult for some game engines to manage this burden; many chose to simply accept the performance/power overhead of performing rotation in the presentation engine.

This extension allows applications to achieve the performance benefits of pre-rotated rendering by moving much of the above-mentioned burden to the graphics driver. The following is unchanged with this extension:

- Applications create a swapchain matching the native orientation of the display. Applications must also set the VkSwapchainCreateInfoKHR::`preTransform` equal to the `currentTransform` as returned by vkGetPhysicalDeviceSurfaceCapabilitiesKHR.

The following is changed with this extension:

- At vkCmdBeginRenderPass, the application provides extension struct VkRenderPassTransformBeginInfoQCOM specifying the render pass transform parameters.
- At vkBeginCommandBuffer for secondary command buffers, the application provides extension struct VkCommandBufferInheritanceRenderPassTransformInfoQCOM specifying the render pass transform parameters.
- The `renderArea`, viewports, scissors, and `fragmentSize` are all provided in the current (non-rotated) coordinate system. The implementation will transform those into the native (rotated) coordinate system.
- The implementation is responsible for transforming shader built-ins (`FragCoord`, `PointCoord`, `SamplePosition`, `PrimitiveShadingRateKHR`, interpolateAt(), dFdx, dFdy, fWidth) into the rotated coordinate system.
- The implementation is responsible for transforming `position` to the rotated coordinate system.
- If this extension is used with `VK_QCOM_tile_properties`, then vkGetFramebufferTilePropertiesQCOM and vkGetDynamicRenderingTilePropertiesQCOM return tile properties in the rotated coordinate space.

### New Structures

- Extending VkCommandBufferInheritanceInfo:
  - VkCommandBufferInheritanceRenderPassTransformInfoQCOM
- Extending VkRenderPassBeginInfo:
  - VkRenderPassTransformBeginInfoQCOM

### New Enum Constants

- `VK_QCOM_RENDER_PASS_TRANSFORM_EXTENSION_NAME`
- `VK_QCOM_RENDER_PASS_TRANSFORM_SPEC_VERSION`

- Extending VkRenderPassCreateFlagBits:
  - `VK_RENDER_PASS_CREATE_TRANSFORM_BIT_QCOM`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_COMMAND_BUFFER_INHERITANCE_RENDER_PASS_TRANSFORM_INFO_QCOM`
  - `VK_STRUCTURE_TYPE_RENDER_PASS_TRANSFORM_BEGIN_INFO_QCOM`

**Issues**

1) Some early Adreno drivers (October 2019 through March 2020) advertised support for this extension but expected VK_STRUCTURE_TYPE values different from those in the vukan headers. To cover all Adreno devices on the market, applications need to detect the driver version and use the appropriate VK_STRUCTURE_TYPE values from the table below.

The driver version reported in VkPhysicalDeviceProperties.driverVersion is a `uint32_t` type. You can decode the `uint32_t` value into a major.minor.patch version as shown below:

```
uint32_t  major = ((driverVersion) >> 22);
uint32_t  minor = ((driverVersion) >> 12) & 0x3ff);
uint32_t  patch = ((driverVersion) & 0xfff);
```

If the Adreno major.minor.patch version is greater than or equal to 512.469.0, then simply use the VK_STRUCTURE_TYPE values as defined in vulkan_core.h. If the version is less than or equal to 512.468.0, then use the alternate values for the two VK_STRUCTURE_TYPEs in the table below.

*Table 107. `Adreno` Driver Requirements*

| | **Adreno Driver Version** | |
|---|---|---|
| | 512.468.0 and earlier | 512.469.0 and later |
| VK_STRUCTURE_TYPE_RENDER_PASS_TRANSFORM_BEGIN_INFO_QCOM | 1000282000 | 1000282001 |
| VK_STRUCTURE_TYPE_COMMAND_BUFFER_INHERITANCE_RENDER_PASS_TRANSFORM_INFO_QCOM | 1000282001 | 1000282000 |

2) Should the extension support only rotations (e.g. 90, 180, 270-degrees), or also mirror transforms (e.g. vertical flips)? Mobile use cases only require rotation. Other display systems such as projectors might require a flipped transform.

**RESOLVED**: In this version of the extension, the functionality is restricted to 90, 180, and 270-degree rotations to address mobile use cases.

3) How does this extension interact with VK_EXT_fragment_density_map?

**RESOLVED** Some implementations may not be able to support a render pass that enables both render pass transform and fragment density maps. For simplicity, this extension disallows enabling both features within a single render pass.

4) What should this extension be named?

We considered names such as “rotated_rendering”, “pre_rotation” and others. Since the functionality is limited to a render pass, it seemed the name should include “render_pass”. While the current extension is limited to rotations, it could be extended to other transforms (like mirror) in the future.

**RESOLVED** The name “render_pass_transform” seems like the most accurate description of the introduced functionality.

5) How does this extension interact with VK_KHR_fragment_shading_rate?

**RESOLVED**: For the same reasons as issue 3, this extension disallows enabling both `pFragmentShadingRateAttachment` and render pass transform within a single render pass.

However, pipeline shading rate and primitive shading rate are supported, and their respective `fragmentSize` and `PrimitiveShadingRateKHR` are provided in the current (non-rotated) coordinate system. The implementation is responsible for transforming them to the rotated coordinate system.

The set of supported shading rates **may** be different per transform. Supported rates queried from vkGetPhysicalDeviceFragmentShadingRatesKHR are in the native (rotated) coordinate system. This means that the application **must** swap the x/y of the reported rates to get the set of rates supported for 90 and 270 degree rotation.

### Version History

- Revision 1, 2020-02-05 (Jeff Leger)
- Revision 2, 2021-03-09 (Matthew Netsch)
  - Adds interactions with VK_KHR_fragment_shading_rate
- Revision 3, 2022-07-11 (Arpit Agarwal)
  - Adds interactions with VK_QCOM_tile_properties
- Revision 4, 2023-12-13 (Matthew Netsch)
  - Relax dependencies on VK_KHR_surface and VK_KHR_swapchain

## VK_QCOM_rotated_copy_commands

### Name String

`VK_QCOM_rotated_copy_commands`

### Extension Type

Device extension

### Registered Extension Number

334

### Revision

2

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_copy_commands2
or
Vulkan Version 1.3

### Contact

- Matthew Netsch [GitHub]mnetsch

### Other Extension Metadata

### Last Modified Date

2023-12-13

### Interactions and External Dependencies

- This extension interacts with `VK_KHR_swapchain`
- This extension interacts with `VK_KHR_surface`

### Contributors

- Jeff Leger, Qualcomm Technologies, Inc.
- Matthew Netsch, Qualcomm Technologies, Inc.

### Description

This extension extends adds an optional rotation transform to copy commands vkCmdBlitImage2KHR, vkCmdCopyImageToBuffer2KHR and vkCmdCopyBufferToImage2KHR. When copying between two resources, where one resource contains rotated content and the other does not, a rotated copy may be desired. This extension may be used in combination with VK_QCOM_render_pass_transform which adds rotated render passes.

This extension adds an extension structure to the following commands: vkCmdBlitImage2KHR, vkCmdCopyImageToBuffer2KHR and vkCmdCopyBufferToImage2KHR

### Issues

1) What is an appropriate name for the added extension structure? The style guide says “Structures which extend other structures through the `pNext` chain should reflect the name of the base structure they extend.”, but in this case a single extension structure is used to extend three base structures (vkCmdBlitImage2KHR, vkCmdCopyImageToBuffer2KHR and vkCmdCopyBufferToImage2KHR). Creating three identical structures with unique names seemed

undesirable.

**RESOLVED**: Deviate from the style guide for extension structure naming.

2) Should this extension add a rotation capability to vkCmdCopyImage2KHR?

**RESOLVED**: No. Use of rotated vkCmdBlitImage2KHR can fully address this use case.

3) Should this extension add a rotation capability to vkCmdResolveImage2KHR?

**RESOLVED** No. Use of vkCmdResolveImage2KHR is very slow and extremely bandwidth intensive on Qualcomm's GPU architecture and use of pResolveAttachments in vkRenderPass is the strongly preferred approach. Therefore, we choose not to introduce a rotation capability to vkCmdResolveImage2KHR.

### New Structures

- Extending VkBufferImageCopy2, VkImageBlit2:
  - VkCopyCommandTransformInfoQCOM

### New Enum Constants

- `VK_QCOM_ROTATED_COPY_COMMANDS_EXTENSION_NAME`
- `VK_QCOM_ROTATED_COPY_COMMANDS_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_COPY_COMMAND_TRANSFORM_INFO_QCOM`

### Version History

- Revision 1, 2020-09-19 (Jeff Leger)
- Revision 2, 2023-12-13 (Matthew Netsch)
  - Relax dependency on VK_KHR_swapchain

## VK_QCOM_tile_properties

### Name String

`VK_QCOM_tile_properties`

### Extension Type

Device extension

### Registered Extension Number

485

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### API Interactions

- Interacts with VK_VERSION_1_3
- Interacts with VK_KHR_dynamic_rendering

### Contact

- Matthew Netsch [GitHub]mnetsch

### Extension Proposal

VK_QCOM_tile_properties

### Other Extension Metadata

### Last Modified Date

2022-07-11

### Interactions and External Dependencies

- This extension interacts with `VK_EXT_subpass_merge_feedback`

### Contributors

- Jonathan Wicks, Qualcomm Technologies, Inc.
- Jonathan Tinkham, Qualcomm Technologies, Inc.
- Arpit Agarwal, Qualcomm Technologies, Inc.
- Jeff Leger, Qualcomm Technologies, Inc.

### Description

This extension allows an application to query the tile properties. This extension supports both renderpasses and dynamic rendering.

### New Commands

- vkGetDynamicRenderingTilePropertiesQCOM
- vkGetFramebufferTilePropertiesQCOM

### New Structures

- VkTilePropertiesQCOM
- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceTilePropertiesFeaturesQCOM

If VK_KHR_dynamic_rendering or Vulkan Version 1.3 is supported:

- VkRenderingInfoKHR

### New Enum Constants

- `VK_QCOM_TILE_PROPERTIES_EXTENSION_NAME`
- `VK_QCOM_TILE_PROPERTIES_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_TILE_PROPERTIES_FEATURES_QCOM`
  - `VK_STRUCTURE_TYPE_TILE_PROPERTIES_QCOM`

### Version History

- Revision 1, 2022-07-11 (Arpit Agarwal)
  - Initial version

## VK_QCOM_ycbcr_degamma

### Name String

`VK_QCOM_ycbcr_degamma`

### Extension Type

Device extension

### Registered Extension Number

521

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

None

### Contact

- Matthew Netsch [GitHub]mnetsch

### Other Extension Metadata

### Last Modified Date

2023-07-31

### IP Status

No known IP claims.

### Interactions and External Dependencies

None

### Contributors

- Jeff Leger, Qualcomm
- Jonathan Wicks, Qualcomm

### Description

This extension allows implementations to expose support for “sRGB EOTF” also known as “sRGB degamma”, used in combination with images using 8-bit $Y'C_BC_R$ formats. In addition, the degamma can be selectively applied to the Y (luma) or CrCb (chroma).

`VK_KHR_sampler_ycbcr_conversion` adds support for $Y'C_BC_R$ conversion, but allows texture sampling in a non-linear space which can cause artifacts. This extension allows implementations to expose sRGB degamma for $Y'C_BC_R$ formats, which is performed during texture filtering, allowing texture filtering to operate in a linear space.

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceYcbcrDegammaFeaturesQCOM
- Extending VkSamplerYcbcrConversionCreateInfo:
  - VkSamplerYcbcrConversionYcbcrDegammaCreateInfoQCOM

### New Enum Constants

- `VK_QCOM_YCBCR_DEGAMMA_EXTENSION_NAME`
- `VK_QCOM_YCBCR_DEGAMMA_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_YCBCR_DEGAMMA_FEATURES_QCOM`
  - `VK_STRUCTURE_TYPE_SAMPLER_YCBCR_CONVERSION_YCBCR_DEGAMMA_CREATE_INFO_QCOM`

### Issues

1) Which $Y'C_BC_R$ formats support the degamma feature?

**RESOLVED**: For implementations that support the extension, each format that contains 8-bit R, G, and B components and supports either `VK_FORMAT_FEATURE_MIDPOINT_CHROMA_SAMPLES_BIT` or `VK_FORMAT_FEATURE_COSITED_CHROMA_SAMPLES_BIT` must support degamma.

Since non-compressed Vulkan sRGB formats are already limited to 8-bit components, and since Adreno supports degamma for all 8bit $Y'C_BC_R$ formats, this extension does not introduce a new VK_FORMAT_FEATURE* bit for the degamma feature.

2) On which $Y'C_BC_R$ components is the degamma applied?

**RESOLVED**: While degamma is expected to be applied to only the Y (luma) component, the extension provides the ability to selectively enable degamma for both the Y (luma) and/or CbCr (chroma) components.

3) Should degamma be enabled for the sampler object or for the image view object?

**RESOLVED**: Both. This extension extends VkSamplerYcbcrConversionCreateInfo and the specification already requires that both sampler and view objects must be created with an *identical* VkSamplerYcbcrConversionCreateInfo in their pNext chains.

4) Why apply the "sRGB" transfer function directly to $Y'C_BC_R$ data when it would be more correct to use the "ITU transfer function", and do so only after the values have been converted into non-linear R'G'B'?

**RESOLVED**: $Y'C_BC_R$ is frequently stored according to standards (e.g. BT.601 and BT.709) that specify that the conversion between linear and non-linear should use the ITU Transfer function. The ITU transfer function is mathematically different from the sRGB transfer function and while sRGB and ITU define similar curves, the difference is significant. Performing the "sRGB degamma" prior to range expansion can introduce artifacts if the content uses `VK_SAMPLER_YCBCR_RANGE_ITU_NARROW` encoding. Nevertheless, using sRGB can make sense for certain use cases where camera YCbCr images are known to be encoded with sRGB (or a pure gamma 2.2) transfer function and are known to use full-range encoding.

For those use cases, this extension leverages the GPU ability to enable sRGB degamma at little cost, and can improve quality because texture filtering is able to occur in linear space.

### Version History

- Revision 1, 2023-07-31 (Jeff Leger)

## VK_QNX_external_memory_screen_buffer

### Name String

`VK_QNX_external_memory_screen_buffer`

### Extension Type

Device extension

### Registered Extension Number

530

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_sampler_ycbcr_conversion

and
VK_KHR_external_memory
and
VK_KHR_dedicated_allocation
or
Vulkan Version 1.1
and
VK_EXT_queue_family_foreign

### Contact

- Mike Gorchak [GitHub]mgorchak-blackberry
- Aaron Ruby [GitHub]aruby-blackberry

### Other Extension Metadata

### Last Modified Date

2023-05-17

### IP Status

No known IP claims.

### Contributors

- Mike Gorchak, QNX / Blackberry Limited
- Aaron Ruby, QNX / Blackberry Limited

### Description

This extension enables an application to import QNX Screen `_screen_buffer` objects created outside of the Vulkan device into Vulkan memory objects, where they can be bound to images and buffers.

Some `_screen_buffer` images have implementation-defined *external formats* that **may** not correspond to Vulkan formats. Sampler $Y'C_BC_R$ conversion **can** be used to sample from these images and convert them to a known color space.

`_screen_buffer` is strongly typed, so naming the handle type is redundant. The internal layout and therefore size of a `_screen_buffer` image may depend on native usage flags that do not have corresponding Vulkan counterparts.

### New Commands

- vkGetScreenBufferPropertiesQNX

### New Structures

- VkScreenBufferPropertiesQNX
- Extending VkImageCreateInfo, VkSamplerYcbcrConversionCreateInfo:
  - VkExternalFormatQNX

- Extending VkMemoryAllocateInfo:
  - VkImportScreenBufferInfoQNX
- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceExternalMemoryScreenBufferFeaturesQNX
- Extending VkScreenBufferPropertiesQNX:
  - VkScreenBufferFormatPropertiesQNX

### New Enum Constants

- `VK_QNX_EXTERNAL_MEMORY_SCREEN_BUFFER_EXTENSION_NAME`
- `VK_QNX_EXTERNAL_MEMORY_SCREEN_BUFFER_SPEC_VERSION`
- Extending VkExternalMemoryHandleTypeFlagBits:
  - `VK_EXTERNAL_MEMORY_HANDLE_TYPE_SCREEN_BUFFER_BIT_QNX`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_EXTERNAL_FORMAT_QNX`
  - `VK_STRUCTURE_TYPE_IMPORT_SCREEN_BUFFER_INFO_QNX`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTERNAL_MEMORY_SCREEN_BUFFER_FEATURES_QNX`
  - `VK_STRUCTURE_TYPE_SCREEN_BUFFER_FORMAT_PROPERTIES_QNX`
  - `VK_STRUCTURE_TYPE_SCREEN_BUFFER_PROPERTIES_QNX`

### Issues

### Version History

- Revision 1, 2023-05-17 (Mike Gorchak)
  - Initial version

## VK_QNX_screen_surface

### Name String

`VK_QNX_screen_surface`

### Extension Type

Instance extension

### Registered Extension Number

379

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_surface

### Contact

- Mike Gorchak [GitHub]mgorchak-blackberry

### Other Extension Metadata

### Last Modified Date

2021-01-11

### IP Status

No known IP claims.

### Contributors

- Mike Gorchak, BlackBerry Limited

### Description

The `VK_QNX_screen_surface` extension is an instance extension. It provides a mechanism to create a VkSurfaceKHR object (defined by the `VK_KHR_surface` extension) that refers to a QNX Screen `window`, as well as a query to determine support for rendering to a QNX Screen compositor.

### New Commands

- vkCreateScreenSurfaceQNX
- vkGetPhysicalDeviceScreenPresentationSupportQNX

### New Structures

- VkScreenSurfaceCreateInfoQNX

### New Bitmasks

- VkScreenSurfaceCreateFlagsQNX

### New Enum Constants

- `VK_QNX_SCREEN_SURFACE_EXTENSION_NAME`
- `VK_QNX_SCREEN_SURFACE_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_SCREEN_SURFACE_CREATE_INFO_QNX`

### Version History

- Revision 1, 2021-01-11 (Mike Gorchak)
  - Initial draft.

## VK_SEC_amigo_profiling

### Name String

`VK_SEC_amigo_profiling`

### Extension Type

Device extension

### Registered Extension Number

486

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### Contact

- Ralph Potter [GitLab]r_potter

### Other Extension Metadata

### Last Modified Date

2022-07-29

### IP Status

No known IP claims.

### Contributors

- Ralph Potter, Samsung
- Sangrak Oh, Samsung
- Jinku Kang, Samsung

### Description

This extension is intended to communicate information from layered API implementations such as ANGLE to internal proprietary system schedulers. It has no behavioral implications beyond enabling more intelligent behavior from the system scheduler.

Application developers should avoid using this extension. It is documented solely for the benefit of tools and layer developers, who may need to manipulate `pNext` chains that include these structures.

| | |
|---|---|
| **NOTE** | There is currently no specification language written for this extension. The links to |

APIs defined by the extension are to stubs that only include generated content such as API declarations and implicit valid usage statements.

**NOTE** This extension is only intended for use in specific embedded environments with known implementation details, and is therefore undocumented.

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceAmigoProfilingFeaturesSEC
- Extending VkSubmitInfo:
  - VkAmigoProfilingSubmitInfoSEC

### New Enum Constants

- `VK_SEC_AMIGO_PROFILING_EXTENSION_NAME`
- `VK_SEC_AMIGO_PROFILING_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_AMIGO_PROFILING_SUBMIT_INFO_SEC`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_AMIGO_PROFILING_FEATURES_SEC`

### Stub API References

There is currently no specification language written for this type. This section acts only as placeholder and to avoid dead links in the specification and reference pages.

```
// Provided by VK_SEC_amigo_profiling
typedef struct VkPhysicalDeviceAmigoProfilingFeaturesSEC {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           amigoProfiling;
} VkPhysicalDeviceAmigoProfilingFeaturesSEC;
```

#### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceAmigoProfilingFeaturesSEC-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_AMIGO_PROFILING_FEATURES_SEC`

There is currently no specification language written for this type. This section acts only as placeholder and to avoid dead links in the specification and reference pages.

```
// Provided by VK_SEC_amigo_profiling
typedef struct VkAmigoProfilingSubmitInfoSEC {
    VkStructureType    sType;
    const void*        pNext;
    uint64_t           firstDrawTimestamp;
    uint64_t           swapBufferTimestamp;
} VkAmigoProfilingSubmitInfoSEC;
```

### Valid Usage (Implicit)

- VUID-VkAmigoProfilingSubmitInfoSEC-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_AMIGO_PROFILING_SUBMIT_INFO_SEC`

#### Version History

- Revision 1, 2022-07-29 (Ralph Potter)
  - Initial specification

## VK_VALVE_descriptor_set_host_mapping

#### Name String

`VK_VALVE_descriptor_set_host_mapping`

#### Extension Type

Device extension

#### Registered Extension Number

421

#### Revision

1

#### Ratification Status

Not ratified

#### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

#### Special Use

- D3D support

#### Contact

- Hans-Kristian Arntzen [GitHub]HansKristian-Work

### Other Extension Metadata

#### Last Modified Date

2022-02-22

#### IP Status

No known IP claims.

#### Contributors

- Hans-Kristian Arntzen, Valve

#### Description

This extension allows applications to directly query a host pointer for a VkDescriptorSet which **can** be used to copy descriptors between descriptor sets without the use of an API command. Memory offsets and sizes for descriptors **can** be queried from a VkDescriptorSetLayout as well.

| | |
|---|---|
| **NOTE** | There is currently no specification language written for this extension. The links to APIs defined by the extension are to stubs that only include generated content such as API declarations and implicit valid usage statements. |

| | |
|---|---|
| **NOTE** | This extension is only intended for use in specific embedded environments with known implementation details, and is therefore undocumented. |

#### New Commands

- vkGetDescriptorSetHostMappingVALVE
- vkGetDescriptorSetLayoutHostMappingInfoVALVE

#### New Structures

- VkDescriptorSetBindingReferenceVALVE
- VkDescriptorSetLayoutHostMappingInfoVALVE
- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceDescriptorSetHostMappingFeaturesVALVE

#### New Enum Constants

- `VK_VALVE_DESCRIPTOR_SET_HOST_MAPPING_EXTENSION_NAME`
- `VK_VALVE_DESCRIPTOR_SET_HOST_MAPPING_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_DESCRIPTOR_SET_BINDING_REFERENCE_VALVE`
  - `VK_STRUCTURE_TYPE_DESCRIPTOR_SET_LAYOUT_HOST_MAPPING_INFO_VALVE`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DESCRIPTOR_SET_HOST_MAPPING_FEATURES_VALVE`

#### Stub API References

There is currently no specification language written for this command. This section acts only as placeholder and to avoid dead links in the specification and reference pages.

```
// Provided by VK_VALVE_descriptor_set_host_mapping
void vkGetDescriptorSetLayoutHostMappingInfoVALVE(
    VkDevice                                    device,
    const VkDescriptorSetBindingReferenceVALVE* pBindingReference,
    VkDescriptorSetLayoutHostMappingInfoVALVE*  pHostMapping);
```

### Valid Usage (Implicit)

- VUID-vkGetDescriptorSetLayoutHostMappingInfoVALVE-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkGetDescriptorSetLayoutHostMappingInfoVALVE-pBindingReference-parameter
  `pBindingReference` **must** be a valid pointer to a valid VkDescriptorSetBindingReferenceVALVE structure
- VUID-vkGetDescriptorSetLayoutHostMappingInfoVALVE-pHostMapping-parameter
  `pHostMapping` **must** be a valid pointer to a VkDescriptorSetLayoutHostMappingInfoVALVE structure

There is currently no specification language written for this command. This section acts only as placeholder and to avoid dead links in the specification and reference pages.

```
// Provided by VK_VALVE_descriptor_set_host_mapping
void vkGetDescriptorSetHostMappingVALVE(
    VkDevice                                    device,
    VkDescriptorSet                             descriptorSet,
    void**                                      ppData);
```

### Valid Usage (Implicit)

- VUID-vkGetDescriptorSetHostMappingVALVE-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkGetDescriptorSetHostMappingVALVE-descriptorSet-parameter
  `descriptorSet` **must** be a valid VkDescriptorSet handle
- VUID-vkGetDescriptorSetHostMappingVALVE-ppData-parameter
  `ppData` **must** be a valid pointer to a pointer value
- VUID-vkGetDescriptorSetHostMappingVALVE-descriptorSet-parent
  `descriptorSet` **must** have been created, allocated, or retrieved from `device`

There is currently no specification language written for this type. This section acts only as

placeholder and to avoid dead links in the specification and reference pages.

```
// Provided by VK_VALVE_descriptor_set_host_mapping
typedef struct VkPhysicalDeviceDescriptorSetHostMappingFeaturesVALVE {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           descriptorSetHostMapping;
} VkPhysicalDeviceDescriptorSetHostMappingFeaturesVALVE;
```

### Valid Usage (Implicit)

- VUID-VkPhysicalDeviceDescriptorSetHostMappingFeaturesVALVE-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DESCRIPTOR_SET_HOST_MAPPING_FEATURES_VALVE`

There is currently no specification language written for this type. This section acts only as placeholder and to avoid dead links in the specification and reference pages.

```
// Provided by VK_VALVE_descriptor_set_host_mapping
typedef struct VkDescriptorSetBindingReferenceVALVE {
    VkStructureType          sType;
    const void*              pNext;
    VkDescriptorSetLayout    descriptorSetLayout;
    uint32_t                 binding;
} VkDescriptorSetBindingReferenceVALVE;
```

### Valid Usage (Implicit)

- VUID-VkDescriptorSetBindingReferenceVALVE-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_DESCRIPTOR_SET_BINDING_REFERENCE_VALVE`
- VUID-VkDescriptorSetBindingReferenceVALVE-pNext-pNext
  `pNext` **must** be `NULL`
- VUID-VkDescriptorSetBindingReferenceVALVE-descriptorSetLayout-parameter
  `descriptorSetLayout` **must** be a valid VkDescriptorSetLayout handle

There is currently no specification language written for this type. This section acts only as placeholder and to avoid dead links in the specification and reference pages.

```
// Provided by VK_VALVE_descriptor_set_host_mapping
typedef struct VkDescriptorSetLayoutHostMappingInfoVALVE {
    VkStructureType    sType;
    void*              pNext;
    size_t             descriptorOffset;
    uint32_t           descriptorSize;
} VkDescriptorSetLayoutHostMappingInfoVALVE;
```

### Valid Usage (Implicit)

- VUID-VkDescriptorSetLayoutHostMappingInfoVALVE-sType-sType
  `sType` **must** be `VK_STRUCTURE_TYPE_DESCRIPTOR_SET_LAYOUT_HOST_MAPPING_INFO_VALVE`
- VUID-VkDescriptorSetLayoutHostMappingInfoVALVE-pNext-pNext
  `pNext` **must** be `NULL`

**Version History**

- Revision 1, 2022-02-22 (Hans-Kristian Arntzen)
  - Initial specification

# List of Provisional Extensions

- VK_KHR_portability_subset
- VK_AMDX_shader_enqueue
- VK_NV_cuda_kernel_launch
- VK_NV_displacement_micromap

## VK_KHR_portability_subset

### Name String

`VK_KHR_portability_subset`

### Extension Type

Device extension

### Registered Extension Number

164

### Revision

1

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

- **This is a *provisional* extension and must be used with caution. See the description of provisional header files for enablement and stability details.**

### Contact

- Bill Hollings [GitHub]billhollings

### Other Extension Metadata

### Last Modified Date

2020-07-21

### IP Status

No known IP claims.

### Contributors

- Bill Hollings, The Brenwill Workshop Ltd.
- Daniel Koch, NVIDIA
- Dzmitry Malyshau, Mozilla
- Chip Davis, CodeWeavers
- Dan Ginsburg, Valve
- Mike Weiblen, LunarG
- Neil Trevett, NVIDIA
- Alexey Knyazev, Independent

### Description

The `VK_KHR_portability_subset` extension allows a non-conformant Vulkan implementation to be built on top of another non-Vulkan graphics API, and identifies differences between that implementation and a fully-conformant native Vulkan implementation.

This extension provides Vulkan implementations with the ability to mark otherwise-required capabilities as unsupported, or to establish additional properties and limits that the application should adhere to in order to guarantee portable behavior and operation across platforms, including platforms where Vulkan is not natively supported.

The goal of this specification is to document, and make queryable, capabilities which are required to be supported by a fully-conformant Vulkan 1.0 implementation, but may be optional for an implementation of the Vulkan 1.0 Portability Subset.

The intent is that this extension will be advertised only on implementations of the Vulkan 1.0 Portability Subset, and not on conformant implementations of Vulkan 1.0. Fully-conformant Vulkan implementations provide all the required capabilities, and so will not provide this extension. Therefore, the existence of this extension can be used to determine that an implementation is likely not fully conformant with the Vulkan spec.

If this extension is supported by the Vulkan implementation, the application must enable this extension.

This extension defines several new structures that can be chained to the existing structures used by certain standard Vulkan calls, in order to query for non-conformant portable behavior.

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDevicePortabilitySubsetFeaturesKHR
- Extending VkPhysicalDeviceProperties2:
  - VkPhysicalDevicePortabilitySubsetPropertiesKHR

### New Enum Constants

- `VK_KHR_PORTABILITY_SUBSET_EXTENSION_NAME`
- `VK_KHR_PORTABILITY_SUBSET_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PORTABILITY_SUBSET_FEATURES_KHR`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PORTABILITY_SUBSET_PROPERTIES_KHR`

### Issues

None.

### Version History

- Revision 1, 2020-07-21 (Bill Hollings)
  - Initial draft.

## VK_AMDX_shader_enqueue

### Name String

`VK_AMDX_shader_enqueue`

### Extension Type

Device extension

### Registered Extension Number

135

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1
and
VK_KHR_synchronization2
or
Vulkan Version 1.3
and
VK_KHR_pipeline_library
and
VK_KHR_spirv_1_4

- **This is a *provisional* extension and must be used with caution. See the description of provisional header files for enablement and stability details.**

### API Interactions

- Interacts with VK_KHR_maintenance5

### SPIR-V Dependencies

- SPV_AMDX_shader_enqueue

### Contact

- Tobias Hector [GitHub]tobski

### Extension Proposal

VK_AMDX_shader_enqueue

### Other Extension Metadata

#### Last Modified Date

2021-07-22

#### Provisional

**This extension is *provisional* and should not be used in production applications. The functionality may change in ways that break backwards compatibility between revisions, and before final release.**

#### Contributors

- Tobias Hector, AMD
- Matthaeus Chajdas, AMD
- Maciej Jesionowski, AMD
- Robert Martin, AMD
- Qun Lin, AMD
- Rex Xu, AMD
- Dominik Witczak, AMD
- Karthik Srinivasan, AMD
- Nicolai Haehnle, AMD
- Stuart Smith, AMD

#### Description

This extension adds the ability for developers to enqueue compute shader workgroups from other compute shaders.

#### New Commands

- vkCmdDispatchGraphAMDX
- vkCmdDispatchGraphIndirectAMDX
- vkCmdDispatchGraphIndirectCountAMDX
- vkCmdInitializeGraphScratchMemoryAMDX
- vkCreateExecutionGraphPipelinesAMDX
- vkGetExecutionGraphPipelineNodeIndexAMDX
- vkGetExecutionGraphPipelineScratchSizeAMDX

#### New Structures

- VkDispatchGraphCountInfoAMDX
- VkDispatchGraphInfoAMDX
- VkExecutionGraphPipelineCreateInfoAMDX

- VkExecutionGraphPipelineScratchSizeAMDX
- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceShaderEnqueueFeaturesAMDX
- Extending VkPhysicalDeviceProperties2:
  - VkPhysicalDeviceShaderEnqueuePropertiesAMDX
- Extending VkPipelineShaderStageCreateInfo:
  - VkPipelineShaderStageNodeCreateInfoAMDX

#### New Unions

- VkDeviceOrHostAddressConstAMDX

#### New Enum Constants

- `VK_AMDX_SHADER_ENQUEUE_EXTENSION_NAME`
- `VK_AMDX_SHADER_ENQUEUE_SPEC_VERSION`
- `VK_SHADER_INDEX_UNUSED_AMDX`
- Extending VkBufferUsageFlagBits:
  - `VK_BUFFER_USAGE_EXECUTION_GRAPH_SCRATCH_BIT_AMDX`
- Extending VkPipelineBindPoint:
  - `VK_PIPELINE_BIND_POINT_EXECUTION_GRAPH_AMDX`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_EXECUTION_GRAPH_PIPELINE_CREATE_INFO_AMDX`
  - `VK_STRUCTURE_TYPE_EXECUTION_GRAPH_PIPELINE_SCRATCH_SIZE_AMDX`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_ENQUEUE_FEATURES_AMDX`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_ENQUEUE_PROPERTIES_AMDX`
  - `VK_STRUCTURE_TYPE_PIPELINE_SHADER_STAGE_NODE_CREATE_INFO_AMDX`

If VK_KHR_maintenance5 is supported:

- Extending VkBufferUsageFlagBits2KHR:
  - `VK_BUFFER_USAGE_2_EXECUTION_GRAPH_SCRATCH_BIT_AMDX`

#### Version History

- Revision 1, 2021-07-22 (Tobias Hector)
  - Initial revision

## VK_NV_cuda_kernel_launch

**Name String**

`VK_NV_cuda_kernel_launch`

**Extension Type**

Device extension

**Registered Extension Number**

308

**Revision**

2

**Ratification Status**

Not ratified

**Extension and Version Dependencies**

None

- **This is a *provisional* extension and must be used with caution. See the description of provisional header files for enablement and stability details.**

**API Interactions**

- Interacts with VK_EXT_debug_report

**Contact**

- Tristan Lorach [GitHub]tlorach

**Other Extension Metadata**

**Last Modified Date**

2020-09-30

**Contributors**

- Eric Werness, NVIDIA

**Description**

Interoperability between APIs can sometimes create additional overhead depending on the platform used. This extension targets deployment of existing CUDA kernels via Vulkan, with a way to directly upload PTX kernels and dispatch the kernels from Vulkan's command buffer without the need to use interoperability between the Vulkan and CUDA contexts. However, we do encourage actual development using the native CUDA runtime for the purpose of debugging and profiling.

The application will first have to create a CUDA module using vkCreateCudaModuleNV then create the CUDA function entry point with vkCreateCudaFunctionNV.

Then in order to dispatch this function, the application will create a command buffer where it will launch the kernel with vkCmdCudaLaunchKernelNV.

When done, the application will then destroy the function handle, as well as the CUDA module

handle with vkDestroyCudaFunctionNV and vkDestroyCudaModuleNV.

To reduce the impact of compilation time, this extension offers the capability to return a binary cache from the PTX that was provided. For this, a first query for the required cache size is made with vkGetCudaModuleCacheNV with a `NULL` pointer to a buffer and with a valid pointer receiving the size; then another call of the same function with a valid pointer to a buffer to retrieve the data. The resulting cache could then be used later for further runs of this application by sending this cache instead of the PTX code (using the same vkCreateCudaModuleNV), thus significantly speeding up the initialization of the CUDA module.

As with VkPipelineCache, the binary cache depends on the hardware architecture. The application must assume the cache might fail, and need to handle falling back to the original PTX code as necessary. Most often, the cache will succeed if the same GPU driver and architecture is used between the cache generation from PTX and the use of this cache. In the event of a new driver version, or if using a different GPU architecture, the cache is likely to become invalid.

#### New Object Types

- VkCudaFunctionNV
- VkCudaModuleNV

#### New Commands

- vkCmdCudaLaunchKernelNV
- vkCreateCudaFunctionNV
- vkCreateCudaModuleNV
- vkDestroyCudaFunctionNV
- vkDestroyCudaModuleNV
- vkGetCudaModuleCacheNV

#### New Structures

- VkCudaFunctionCreateInfoNV
- VkCudaLaunchInfoNV
- VkCudaModuleCreateInfoNV
- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceCudaKernelLaunchFeaturesNV
- Extending VkPhysicalDeviceProperties2:
  - VkPhysicalDeviceCudaKernelLaunchPropertiesNV

#### New Enum Constants

- `VK_NV_CUDA_KERNEL_LAUNCH_EXTENSION_NAME`
- `VK_NV_CUDA_KERNEL_LAUNCH_SPEC_VERSION`

- Extending VkObjectType:
  - VK_OBJECT_TYPE_CUDA_FUNCTION_NV
  - VK_OBJECT_TYPE_CUDA_MODULE_NV
- Extending VkStructureType:
  - VK_STRUCTURE_TYPE_CUDA_FUNCTION_CREATE_INFO_NV
  - VK_STRUCTURE_TYPE_CUDA_LAUNCH_INFO_NV
  - VK_STRUCTURE_TYPE_CUDA_MODULE_CREATE_INFO_NV
  - VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_CUDA_KERNEL_LAUNCH_FEATURES_NV
  - VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_CUDA_KERNEL_LAUNCH_PROPERTIES_NV

If VK_EXT_debug_report is supported:

- Extending VkDebugReportObjectTypeEXT:
  - VK_DEBUG_REPORT_OBJECT_TYPE_CUDA_FUNCTION_NV_EXT
  - VK_DEBUG_REPORT_OBJECT_TYPE_CUDA_MODULE_NV_EXT

#### Issues

None.

#### Version History

- Revision 1, 2020-03-01 (Tristan Lorach)
- Revision 2, 2020-09-30 (Tristan Lorach)

## VK_NV_displacement_micromap

#### Name String

VK_NV_displacement_micromap

#### Extension Type

Device extension

#### Registered Extension Number

398

#### Revision

2

#### Ratification Status

Not ratified

#### Extension and Version Dependencies

VK_EXT_opacity_micromap

- **This is a *provisional* extension and must be used with caution. See the description of provisional header files for enablement and stability details.**

**Contact**

- Christoph Kubisch [GitHub]pixeljetstream
- Eric Werness [GitHub]ewerness-nv

### Other Extension Metadata

**Last Modified Date**

2023-03-17

**Interactions and External Dependencies**

TBD

**Contributors**

- Christoph Kubisch, NVIDIA
- Eric Werness, NVIDIA

### Description

Ray tracing can very efficiently render from geometry which has very fine detail, but when using only a basic triangle representation, memory consumption can be an issue. This extension adds the ability to add a *displacement map* to add more detail to triangles in an acceleration structure with an efficient in-memory format. The format is externally visible to allow the application to compress its internal geometry representations into the compressed format ahead of time. This format adds displacements along a defined vector to subtriangle vertices which are subdivided from the main triangles.

This extension provides:

- a new VkMicromapTypeEXT format for the displacement micromap,
- a structure to extend VkAccelerationStructureGeometryTrianglesDataKHR to attach a displacement micromap to the geometry of the acceleration structure,
- enums extending VkBuildAccelerationStructureFlagBitsKHR to allow for updates.

### New Structures

- Extending VkAccelerationStructureGeometryTrianglesDataKHR:
  - VkAccelerationStructureTrianglesDisplacementMicromapNV
- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceDisplacementMicromapFeaturesNV
- Extending VkPhysicalDeviceProperties2:
  - VkPhysicalDeviceDisplacementMicromapPropertiesNV

### New Enums

- VkDisplacementMicromapFormatNV

### New Enum Constants

- `VK_NV_DISPLACEMENT_MICROMAP_EXTENSION_NAME`
- `VK_NV_DISPLACEMENT_MICROMAP_SPEC_VERSION`
- Extending VkBuildAccelerationStructureFlagBitsKHR:
  - `VK_BUILD_ACCELERATION_STRUCTURE_ALLOW_DISPLACEMENT_MICROMAP_UPDATE_NV`
- Extending VkMicromapTypeEXT:
  - `VK_MICROMAP_TYPE_DISPLACEMENT_MICROMAP_NV`
- Extending VkPipelineCreateFlagBits:
  - `VK_PIPELINE_CREATE_RAY_TRACING_DISPLACEMENT_MICROMAP_BIT_NV`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_ACCELERATION_STRUCTURE_TRIANGLES_DISPLACEMENT_MICROMAP_NV`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DISPLACEMENT_MICROMAP_FEATURES_NV`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DISPLACEMENT_MICROMAP_PROPERTIES_NV`

### Issues

(1) What is the status of this extension?

- Provisional and expected to change. The broad structure and encoding format are stable, but there will likely be changes to the structures, enumerant values, and shader interface.

### Version History

- Revision 1, 2023-03-17 (Eric Werness)
  - Initial public revision
- Revision 2, 2023-07-07 (Eric Werness)
  - Add shader support for decode intrinsics

# List of Deprecated Extensions

- VK_KHR_16bit_storage (promoted to core)
- VK_KHR_bind_memory2 (promoted to core)
- VK_KHR_dedicated_allocation (promoted to core)
- VK_KHR_descriptor_update_template (promoted to core)
- VK_KHR_device_group (promoted to core)
- VK_KHR_device_group_creation (promoted to core)

- VK_KHR_external_fence (promoted to core)
- VK_KHR_external_fence_capabilities (promoted to core)
- VK_KHR_external_memory (promoted to core)
- VK_KHR_external_memory_capabilities (promoted to core)
- VK_KHR_external_semaphore (promoted to core)
- VK_KHR_external_semaphore_capabilities (promoted to core)
- VK_KHR_get_memory_requirements2 (promoted to core)
- VK_KHR_get_physical_device_properties2 (promoted to core)
- VK_KHR_maintenance1 (promoted to core)
- VK_KHR_maintenance2 (promoted to core)
- VK_KHR_maintenance3 (promoted to core)
- VK_KHR_multiview (promoted to core)
- VK_KHR_relaxed_block_layout (promoted to core)
- VK_KHR_sampler_ycbcr_conversion (promoted to core)
- VK_KHR_shader_draw_parameters (promoted to core)
- VK_KHR_storage_buffer_storage_class (promoted to core)
- VK_KHR_variable_pointers (promoted to core)
- VK_EXT_buffer_device_address
- VK_EXT_calibrated_timestamps
- VK_EXT_debug_marker
- VK_EXT_debug_report
- VK_EXT_global_priority
- VK_EXT_global_priority_query
- VK_EXT_index_type_uint8
- VK_EXT_line_rasterization
- VK_EXT_load_store_op_none
- VK_EXT_shader_subgroup_vote
- VK_EXT_validation_features
- VK_EXT_validation_flags
- VK_EXT_vertex_attribute_divisor
- VK_AMD_draw_indirect_count
- VK_AMD_gpu_shader_half_float
- VK_AMD_gpu_shader_int16
- VK_AMD_negative_viewport_height
- VK_ARM_rasterization_order_attachment_access

- VK_IMG_format_pvrtc
- VK_MVK_ios_surface
- VK_MVK_macos_surface
- VK_NV_compute_shader_derivatives
- VK_NV_dedicated_allocation
- VK_NV_external_memory
- VK_NV_external_memory_capabilities
- VK_NV_external_memory_win32
- VK_NV_fragment_shader_barycentric
- VK_NV_glsl_shader
- VK_NV_ray_tracing
- VK_NV_win32_keyed_mutex
- VK_VALVE_mutable_descriptor_type

## VK_KHR_16bit_storage

### Name String

`VK_KHR_16bit_storage`

### Extension Type

Device extension

### Registered Extension Number

84

### Revision

1

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
and
VK_KHR_storage_buffer_storage_class
or
Vulkan Version 1.1

### SPIR-V Dependencies

- SPV_KHR_16bit_storage

### Deprecation State

- *Promoted* to Vulkan 1.1

### Contact

- Jan-Harald Fredriksen [GitHub]janharaldfredriksen-arm

### Other Extension Metadata

### Last Modified Date

2017-09-05

### IP Status

No known IP claims.

### Interactions and External Dependencies

- This extension provides API support for `GL_EXT_shader_16bit_storage`

### Contributors

- Alexander Galazin, ARM
- Jan-Harald Fredriksen, ARM
- Joerg Wagner, ARM

- Neil Henning, Codeplay
- Jeff Bolz, Nvidia
- Daniel Koch, Nvidia
- David Neto, Google
- John Kessenich, Google

### Description

The `VK_KHR_16bit_storage` extension allows use of 16-bit types in shader input and output interfaces, and push constant blocks. This extension introduces several new optional features which map to SPIR-V capabilities and allow access to 16-bit data in `Block`-decorated objects in the `Uniform` and the `StorageBuffer` storage classes, and objects in the `PushConstant` storage class. This extension allows 16-bit variables to be declared and used as user-defined shader inputs and outputs but does not change location assignment and component assignment rules.

### Promotion to Vulkan 1.1

All functionality in this extension is included in core Vulkan 1.1, with the KHR suffix omitted. However, if Vulkan 1.1 is supported and this extension is not, the `storageBuffer16BitAccess` capability is optional. The original type, enum and command names are still available as aliases of the core functionality.

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDevice16BitStorageFeaturesKHR

### New Enum Constants

- `VK_KHR_16BIT_STORAGE_EXTENSION_NAME`
- `VK_KHR_16BIT_STORAGE_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_16BIT_STORAGE_FEATURES_KHR`

### New SPIR-V Capabilities

- `StorageBuffer16BitAccess`
- `UniformAndStorageBuffer16BitAccess`
- `StoragePushConstant16`
- `StorageInputOutput16`

### Version History

- Revision 1, 2017-03-23 (Alexander Galazin)
  - Initial draft

## VK_KHR_bind_memory2

### Name String

`VK_KHR_bind_memory2`

### Extension Type

Device extension

### Registered Extension Number

158

### Revision

1

### Ratification Status

Ratified

### Extension and Version Dependencies

None

### Deprecation State

- *Promoted* to Vulkan 1.1

### Contact

- Tobias Hector [GitHub]tobski

### Other Extension Metadata

### Last Modified Date

2017-09-05

### IP Status

No known IP claims.

### Contributors

- Jeff Bolz, NVIDIA
- Tobias Hector, Imagination Technologies

### Description

This extension provides versions of vkBindBufferMemory and vkBindImageMemory that allow multiple bindings to be performed at once, and are extensible.

This extension also introduces `VK_IMAGE_CREATE_ALIAS_BIT_KHR`, which allows “identical” images that alias the same memory to interpret the contents consistently, even across image layout changes.

### Promotion to Vulkan 1.1

All functionality in this extension is included in core Vulkan 1.1, with the KHR suffix omitted. The

original type, enum and command names are still available as aliases of the core functionality.

### New Commands

- vkBindBufferMemory2KHR
- vkBindImageMemory2KHR

### New Structures

- VkBindBufferMemoryInfoKHR
- VkBindImageMemoryInfoKHR

### New Enum Constants

- `VK_KHR_BIND_MEMORY_2_EXTENSION_NAME`
- `VK_KHR_BIND_MEMORY_2_SPEC_VERSION`
- Extending VkImageCreateFlagBits:
  - `VK_IMAGE_CREATE_ALIAS_BIT_KHR`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_BIND_BUFFER_MEMORY_INFO_KHR`
  - `VK_STRUCTURE_TYPE_BIND_IMAGE_MEMORY_INFO_KHR`

### Version History

- Revision 1, 2017-05-19 (Tobias Hector)
  - Pulled bind memory functions into their own extension

## VK_KHR_dedicated_allocation

### Name String

`VK_KHR_dedicated_allocation`

### Extension Type

Device extension

### Registered Extension Number

128

### Revision

3

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_get_memory_requirements2

or
Vulkan Version 1.1

### Deprecation State

- *Promoted* to Vulkan 1.1

### Contact

- James Jones [GitHub]cubanismo

### Other Extension Metadata

### Last Modified Date

2017-09-05

### IP Status

No known IP claims.

### Contributors

- Jeff Bolz, NVIDIA
- Faith Ekstrand, Intel

### Description

This extension enables resources to be bound to a dedicated allocation, rather than suballocated. For any particular resource, applications **can** query whether a dedicated allocation is recommended, in which case using a dedicated allocation **may** improve the performance of access to that resource. Normal device memory allocations must support multiple resources per allocation, memory aliasing and sparse binding, which could interfere with some optimizations. Applications should query the implementation for when a dedicated allocation **may** be beneficial by adding a `VkMemoryDedicatedRequirementsKHR` structure to the `pNext` chain of the `VkMemoryRequirements2` structure passed as the `pMemoryRequirements` parameter of a call to `vkGetBufferMemoryRequirements2` or `vkGetImageMemoryRequirements2`. Certain external handle types and external images or buffers **may** also depend on dedicated allocations on implementations that associate image or buffer metadata with OS-level memory objects.

This extension adds a two small structures to memory requirements querying and memory allocation: a new structure that flags whether an image/buffer should have a dedicated allocation, and a structure indicating the image or buffer that an allocation will be bound to.

### Promotion to Vulkan 1.1

All functionality in this extension is included in core Vulkan 1.1, with the KHR suffix omitted. The original type, enum and command names are still available as aliases of the core functionality.

### New Structures

- Extending VkMemoryAllocateInfo:
  - VkMemoryDedicatedAllocateInfoKHR

- Extending VkMemoryRequirements2:
  - VkMemoryDedicatedRequirementsKHR

### New Enum Constants

- VK_KHR_DEDICATED_ALLOCATION_EXTENSION_NAME
- VK_KHR_DEDICATED_ALLOCATION_SPEC_VERSION
- Extending VkStructureType:
  - VK_STRUCTURE_TYPE_MEMORY_DEDICATED_ALLOCATE_INFO_KHR
  - VK_STRUCTURE_TYPE_MEMORY_DEDICATED_REQUIREMENTS_KHR

### Examples

```
// Create an image with a dedicated allocation based on the
// implementation's preference

VkImageCreateInfo imageCreateInfo =
{
    // Image creation parameters
};

VkImage image;
VkResult result = vkCreateImage(
    device,
    &imageCreateInfo,
    NULL,               // pAllocator
    &image);

VkMemoryDedicatedRequirementsKHR dedicatedRequirements =
{
    .sType = VK_STRUCTURE_TYPE_MEMORY_DEDICATED_REQUIREMENTS_KHR,
    .pNext = NULL,
};

VkMemoryRequirements2 memoryRequirements =
{
    .sType = VK_STRUCTURE_TYPE_MEMORY_REQUIREMENTS_2,
    .pNext = &dedicatedRequirements,
};

const VkImageMemoryRequirementsInfo2 imageRequirementsInfo =
{
    .sType = VK_STRUCTURE_TYPE_IMAGE_MEMORY_REQUIREMENTS_INFO_2,
    .pNext = NULL,
    .image = image
};

vkGetImageMemoryRequirements2(
```

```
        device,
        &imageRequirementsInfo,
        &memoryRequirements);

    if (dedicatedRequirements.prefersDedicatedAllocation) {
        // Allocate memory with VkMemoryDedicatedAllocateInfoKHR::image
        // pointing to the image we are allocating the memory for

        VkMemoryDedicatedAllocateInfoKHR dedicatedInfo =
        {
            .sType = VK_STRUCTURE_TYPE_MEMORY_DEDICATED_ALLOCATE_INFO_KHR,
            .pNext = NULL,
            .image = image,
            .buffer = VK_NULL_HANDLE,
        };

        VkMemoryAllocateInfo memoryAllocateInfo =
        {
            .sType = VK_STRUCTURE_TYPE_MEMORY_ALLOCATE_INFO,
            .pNext = &dedicatedInfo,
            .allocationSize = memoryRequirements.size,
            .memoryTypeIndex = FindMemoryTypeIndex(memoryRequirements.memoryTypeBits),
        };

        VkDeviceMemory memory;
        vkAllocateMemory(
            device,
            &memoryAllocateInfo,
            NULL,               // pAllocator
            &memory);

        // Bind the image to the memory

        vkBindImageMemory(
            device,
            image,
            memory,
            0);
    } else {
        // Take the normal memory sub-allocation path
    }
```

### Version History

- Revision 1, 2017-02-27 (James Jones)
  - Copy content from VK_NV_dedicated_allocation
  - Add some references to external object interactions to the overview.
- Revision 2, 2017-03-27 (Faith Ekstrand)

    - Rework the extension to be query-based
- Revision 3, 2017-07-31 (Faith Ekstrand)
    - Clarify that memory objects allocated with VkMemoryDedicatedAllocateInfoKHR can only have the specified resource bound and no others.

## VK_KHR_descriptor_update_template

### Name String

`VK_KHR_descriptor_update_template`

### Extension Type

Device extension

### Registered Extension Number

86

### Revision

1

### Ratification Status

Ratified

### Extension and Version Dependencies

None

### API Interactions

- Interacts with VK_EXT_debug_report
- Interacts with VK_KHR_push_descriptor

### Deprecation State

- *Promoted* to Vulkan 1.1

### Contact

- Markus Tavenrath [GitHub]mtavenrath

### Other Extension Metadata

### Last Modified Date

2017-09-05

### IP Status

No known IP claims.

### Interactions and External Dependencies

- Interacts with `VK_KHR_push_descriptor`

### Contributors

- Jeff Bolz, NVIDIA

- Michael Worcester, Imagination Technologies

### Description

Applications may wish to update a fixed set of descriptors in a large number of descriptor sets very frequently, i.e. during initialization phase or if it is required to rebuild descriptor sets for each frame. For those cases it is also not unlikely that all information required to update a single descriptor set is stored in a single struct. This extension provides a way to update a fixed set of descriptors in a single VkDescriptorSet with a pointer to an application-defined data structure describing the new descriptors.

### Promotion to Vulkan 1.1

vkCmdPushDescriptorSetWithTemplateKHR is included as an interaction with `VK_KHR_push_descriptor`. If Vulkan 1.1 and `VK_KHR_push_descriptor` are supported, this is included by `VK_KHR_push_descriptor`.

The base functionality in this extension is included in core Vulkan 1.1, with the KHR suffix omitted. The original type, enum and command names are still available as aliases of the core functionality.

### New Object Types

- VkDescriptorUpdateTemplateKHR

### New Commands

- vkCreateDescriptorUpdateTemplateKHR
- vkDestroyDescriptorUpdateTemplateKHR
- vkUpdateDescriptorSetWithTemplateKHR

If VK_KHR_push_descriptor is supported:

- vkCmdPushDescriptorSetWithTemplateKHR

### New Structures

- VkDescriptorUpdateTemplateCreateInfoKHR
- VkDescriptorUpdateTemplateEntryKHR

### New Enums

- VkDescriptorUpdateTemplateTypeKHR

### New Bitmasks

- VkDescriptorUpdateTemplateCreateFlagsKHR

### New Enum Constants

- `VK_KHR_DESCRIPTOR_UPDATE_TEMPLATE_EXTENSION_NAME`

- `VK_KHR_DESCRIPTOR_UPDATE_TEMPLATE_SPEC_VERSION`
- Extending VkDescriptorUpdateTemplateType:
  - `VK_DESCRIPTOR_UPDATE_TEMPLATE_TYPE_DESCRIPTOR_SET_KHR`
- Extending VkObjectType:
  - `VK_OBJECT_TYPE_DESCRIPTOR_UPDATE_TEMPLATE_KHR`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_DESCRIPTOR_UPDATE_TEMPLATE_CREATE_INFO_KHR`

If VK_EXT_debug_report is supported:

- Extending VkDebugReportObjectTypeEXT:
  - `VK_DEBUG_REPORT_OBJECT_TYPE_DESCRIPTOR_UPDATE_TEMPLATE_KHR_EXT`

If VK_KHR_push_descriptor is supported:

- Extending VkDescriptorUpdateTemplateType:
  - `VK_DESCRIPTOR_UPDATE_TEMPLATE_TYPE_PUSH_DESCRIPTORS_KHR`

#### Version History

- Revision 1, 2016-01-11 (Markus Tavenrath)
  - Initial draft

## VK_KHR_device_group

#### Name String

`VK_KHR_device_group`

#### Extension Type

Device extension

#### Registered Extension Number

61

#### Revision

4

#### Ratification Status

Ratified

#### Extension and Version Dependencies

VK_KHR_device_group_creation

#### API Interactions

- Interacts with VK_KHR_bind_memory2

- Interacts with VK_KHR_surface
- Interacts with VK_KHR_swapchain

**SPIR-V Dependencies**

- SPV_KHR_device_group

**Deprecation State**

- *Promoted* to Vulkan 1.1

**Contact**

- Jeff Bolz [GitHub]jeffbolznv

### Other Extension Metadata

**Last Modified Date**

2017-10-10

**IP Status**

No known IP claims.

**Contributors**

- Jeff Bolz, NVIDIA
- Tobias Hector, Imagination Technologies

### Description

This extension provides functionality to use a logical device that consists of multiple physical devices, as created with the `VK_KHR_device_group_creation` extension. A device group can allocate memory across the subdevices, bind memory from one subdevice to a resource on another subdevice, record command buffers where some work executes on an arbitrary subset of the subdevices, and potentially present a swapchain image from one or more subdevices.

### Promotion to Vulkan 1.1

The following enums, types and commands are included as interactions with `VK_KHR_swapchain`:

- `VK_STRUCTURE_TYPE_DEVICE_GROUP_PRESENT_CAPABILITIES_KHR`
- `VK_STRUCTURE_TYPE_IMAGE_SWAPCHAIN_CREATE_INFO_KHR`
- `VK_STRUCTURE_TYPE_BIND_IMAGE_MEMORY_SWAPCHAIN_INFO_KHR`
- `VK_STRUCTURE_TYPE_ACQUIRE_NEXT_IMAGE_INFO_KHR`
- `VK_STRUCTURE_TYPE_DEVICE_GROUP_PRESENT_INFO_KHR`
- `VK_STRUCTURE_TYPE_DEVICE_GROUP_SWAPCHAIN_CREATE_INFO_KHR`
- `VK_SWAPCHAIN_CREATE_SPLIT_INSTANCE_BIND_REGIONS_BIT_KHR`
- VkDeviceGroupPresentModeFlagBitsKHR
- VkDeviceGroupPresentCapabilitiesKHR
- VkImageSwapchainCreateInfoKHR

- VkBindImageMemorySwapchainInfoKHR
- VkAcquireNextImageInfoKHR
- VkDeviceGroupPresentInfoKHR
- VkDeviceGroupSwapchainCreateInfoKHR
- vkGetDeviceGroupPresentCapabilitiesKHR
- vkGetDeviceGroupSurfacePresentModesKHR
- vkGetPhysicalDevicePresentRectanglesKHR
- vkAcquireNextImage2KHR

If Vulkan 1.1 and `VK_KHR_swapchain` are supported, these are included by `VK_KHR_swapchain`.

The base functionality in this extension is included in core Vulkan 1.1, with the KHR suffix omitted. The original type, enum and command names are still available as aliases of the core functionality.

#### New Commands

- vkCmdDispatchBaseKHR
- vkCmdSetDeviceMaskKHR
- vkGetDeviceGroupPeerMemoryFeaturesKHR

If VK_KHR_surface is supported:

- vkGetDeviceGroupPresentCapabilitiesKHR
- vkGetDeviceGroupSurfacePresentModesKHR
- vkGetPhysicalDevicePresentRectanglesKHR

If VK_KHR_swapchain is supported:

- vkAcquireNextImage2KHR

#### New Structures

- Extending VkBindSparseInfo:
  - VkDeviceGroupBindSparseInfoKHR
- Extending VkCommandBufferBeginInfo:
  - VkDeviceGroupCommandBufferBeginInfoKHR
- Extending VkMemoryAllocateInfo:
  - VkMemoryAllocateFlagsInfoKHR
- Extending VkRenderPassBeginInfo, VkRenderingInfo:
  - VkDeviceGroupRenderPassBeginInfoKHR
- Extending VkSubmitInfo:
  - VkDeviceGroupSubmitInfoKHR

If VK_KHR_bind_memory2 is supported:

- Extending VkBindBufferMemoryInfo:
  - VkBindBufferMemoryDeviceGroupInfoKHR
- Extending VkBindImageMemoryInfo:
  - VkBindImageMemoryDeviceGroupInfoKHR

If VK_KHR_surface is supported:

- VkDeviceGroupPresentCapabilitiesKHR

If VK_KHR_swapchain is supported:

- VkAcquireNextImageInfoKHR
- Extending VkBindImageMemoryInfo:
  - VkBindImageMemorySwapchainInfoKHR
- Extending VkImageCreateInfo:
  - VkImageSwapchainCreateInfoKHR
- Extending VkPresentInfoKHR:
  - VkDeviceGroupPresentInfoKHR
- Extending VkSwapchainCreateInfoKHR:
  - VkDeviceGroupSwapchainCreateInfoKHR

**New Enums**

- VkMemoryAllocateFlagBitsKHR
- VkPeerMemoryFeatureFlagBitsKHR

If VK_KHR_surface is supported:

- VkDeviceGroupPresentModeFlagBitsKHR

**New Bitmasks**

- VkMemoryAllocateFlagsKHR
- VkPeerMemoryFeatureFlagsKHR

If VK_KHR_surface is supported:

- VkDeviceGroupPresentModeFlagsKHR

**New Enum Constants**

- `VK_KHR_DEVICE_GROUP_EXTENSION_NAME`
- `VK_KHR_DEVICE_GROUP_SPEC_VERSION`

- Extending VkDependencyFlagBits:
  - VK_DEPENDENCY_DEVICE_GROUP_BIT_KHR
- Extending VkMemoryAllocateFlagBits:
  - VK_MEMORY_ALLOCATE_DEVICE_MASK_BIT_KHR
- Extending VkPeerMemoryFeatureFlagBits:
  - VK_PEER_MEMORY_FEATURE_COPY_DST_BIT_KHR
  - VK_PEER_MEMORY_FEATURE_COPY_SRC_BIT_KHR
  - VK_PEER_MEMORY_FEATURE_GENERIC_DST_BIT_KHR
  - VK_PEER_MEMORY_FEATURE_GENERIC_SRC_BIT_KHR
- Extending VkPipelineCreateFlagBits:
  - VK_PIPELINE_CREATE_DISPATCH_BASE_KHR
  - VK_PIPELINE_CREATE_VIEW_INDEX_FROM_DEVICE_INDEX_BIT_KHR
- Extending VkStructureType:
  - VK_STRUCTURE_TYPE_DEVICE_GROUP_BIND_SPARSE_INFO_KHR
  - VK_STRUCTURE_TYPE_DEVICE_GROUP_COMMAND_BUFFER_BEGIN_INFO_KHR
  - VK_STRUCTURE_TYPE_DEVICE_GROUP_RENDER_PASS_BEGIN_INFO_KHR
  - VK_STRUCTURE_TYPE_DEVICE_GROUP_SUBMIT_INFO_KHR
  - VK_STRUCTURE_TYPE_MEMORY_ALLOCATE_FLAGS_INFO_KHR

If VK_KHR_bind_memory2 is supported:

- Extending VkImageCreateFlagBits:
  - VK_IMAGE_CREATE_SPLIT_INSTANCE_BIND_REGIONS_BIT_KHR
- Extending VkStructureType:
  - VK_STRUCTURE_TYPE_BIND_BUFFER_MEMORY_DEVICE_GROUP_INFO_KHR
  - VK_STRUCTURE_TYPE_BIND_IMAGE_MEMORY_DEVICE_GROUP_INFO_KHR

If VK_KHR_surface is supported:

- Extending VkStructureType:
  - VK_STRUCTURE_TYPE_DEVICE_GROUP_PRESENT_CAPABILITIES_KHR

If VK_KHR_swapchain is supported:

- Extending VkStructureType:
  - VK_STRUCTURE_TYPE_ACQUIRE_NEXT_IMAGE_INFO_KHR
  - VK_STRUCTURE_TYPE_BIND_IMAGE_MEMORY_SWAPCHAIN_INFO_KHR
  - VK_STRUCTURE_TYPE_DEVICE_GROUP_PRESENT_INFO_KHR
  - VK_STRUCTURE_TYPE_DEVICE_GROUP_SWAPCHAIN_CREATE_INFO_KHR

    - `VK_STRUCTURE_TYPE_IMAGE_SWAPCHAIN_CREATE_INFO_KHR`
- Extending VkSwapchainCreateFlagBitsKHR:
    - `VK_SWAPCHAIN_CREATE_SPLIT_INSTANCE_BIND_REGIONS_BIT_KHR`

#### New Built-in Variables

- `DeviceIndex`

#### New SPIR-V Capabilities

- `DeviceGroup`

#### Version History

- Revision 1, 2016-10-19 (Jeff Bolz)
    - Internal revisions
- Revision 2, 2017-05-19 (Tobias Hector)
    - Removed extended memory bind functions to VK_KHR_bind_memory2, added dependency on that extension, and device-group-specific structs for those functions.
- Revision 3, 2017-10-06 (Ian Elliott)
    - Corrected Vulkan 1.1 interactions with the WSI extensions. All Vulkan 1.1 WSI interactions are with the VK_KHR_swapchain extension.
- Revision 4, 2017-10-10 (Jeff Bolz)
    - Rename “SFR” bits and structure members to use the phrase “split instance bind regions”.

## VK_KHR_device_group_creation

#### Name String

`VK_KHR_device_group_creation`

#### Extension Type

Instance extension

#### Registered Extension Number

71

#### Revision

1

#### Ratification Status

Ratified

#### Extension and Version Dependencies

None

### Deprecation State

- *Promoted* to Vulkan 1.1

### Contact

- Jeff Bolz [GitHub]jeffbolznv

### Other Extension Metadata

### Last Modified Date

2016-10-19

### IP Status

No known IP claims.

### Contributors

- Jeff Bolz, NVIDIA

### Description

This extension provides instance-level commands to enumerate groups of physical devices, and to create a logical device from a subset of one of those groups. Such a logical device can then be used with new features in the `VK_KHR_device_group` extension.

### Promotion to Vulkan 1.1

All functionality in this extension is included in core Vulkan 1.1, with the KHR suffix omitted. The original type, enum and command names are still available as aliases of the core functionality.

### New Commands

- vkEnumeratePhysicalDeviceGroupsKHR

### New Structures

- VkPhysicalDeviceGroupPropertiesKHR
- Extending VkDeviceCreateInfo:
  - VkDeviceGroupDeviceCreateInfoKHR

### New Enum Constants

- `VK_KHR_DEVICE_GROUP_CREATION_EXTENSION_NAME`
- `VK_KHR_DEVICE_GROUP_CREATION_SPEC_VERSION`
- `VK_MAX_DEVICE_GROUP_SIZE_KHR`
- Extending VkMemoryHeapFlagBits:
  - `VK_MEMORY_HEAP_MULTI_INSTANCE_BIT_KHR`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_DEVICE_GROUP_DEVICE_CREATE_INFO_KHR`

- VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_GROUP_PROPERTIES_KHR

### Examples

```
    VkDeviceCreateInfo devCreateInfo = { VK_STRUCTURE_TYPE_DEVICE_CREATE_INFO };
    // (not shown) fill out devCreateInfo as usual.
    uint32_t deviceGroupCount = 0;
    VkPhysicalDeviceGroupPropertiesKHR *props = NULL;

    // Query the number of device groups
    vkEnumeratePhysicalDeviceGroupsKHR(g_vkInstance, &deviceGroupCount, NULL);

    // Allocate and initialize structures to query the device groups
    props = (VkPhysicalDeviceGroupPropertiesKHR *)malloc(deviceGroupCount*sizeof
(VkPhysicalDeviceGroupPropertiesKHR));
    for (i = 0; i < deviceGroupCount; ++i) {
        props[i].sType = VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_GROUP_PROPERTIES_KHR;
        props[i].pNext = NULL;
    }
    vkEnumeratePhysicalDeviceGroupsKHR(g_vkInstance, &deviceGroupCount, props);

    // If the first device group has more than one physical device. create
    // a logical device using all of the physical devices.
    VkDeviceGroupDeviceCreateInfoKHR deviceGroupInfo = {
VK_STRUCTURE_TYPE_DEVICE_GROUP_DEVICE_CREATE_INFO_KHR };
    if (props[0].physicalDeviceCount > 1) {
        deviceGroupInfo.physicalDeviceCount = props[0].physicalDeviceCount;
        deviceGroupInfo.pPhysicalDevices = props[0].physicalDevices;
        devCreateInfo.pNext = &deviceGroupInfo;
    }

    vkCreateDevice(props[0].physicalDevices[0], &devCreateInfo, NULL, &g_vkDevice);
    free(props);
```

### Version History

- Revision 1, 2016-10-19 (Jeff Bolz)
  - Internal revisions

## VK_KHR_external_fence

### Name String

VK_KHR_external_fence

### Extension Type

Device extension

**Registered Extension Number**

114

**Revision**

1

**Ratification Status**

Ratified

**Extension and Version Dependencies**

VK_KHR_external_fence_capabilities

**Deprecation State**

- *Promoted* to Vulkan 1.1

**Contact**

- Jesse Hall [GitHub]critsec

**Other Extension Metadata**

**Last Modified Date**

2017-05-08

**IP Status**

No known IP claims.

**Contributors**

- Jesse Hall, Google
- James Jones, NVIDIA
- Jeff Juliano, NVIDIA
- Cass Everitt, Oculus
- Contributors to `VK_KHR_external_semaphore`

**Description**

An application using external memory may wish to synchronize access to that memory using fences. This extension enables an application to create fences from which non-Vulkan handles that reference the underlying synchronization primitive can be exported.

**Promotion to Vulkan 1.1**

All functionality in this extension is included in core Vulkan 1.1, with the KHR suffix omitted. The original type, enum and command names are still available as aliases of the core functionality.

**New Structures**

- Extending VkFenceCreateInfo:

- VkExportFenceCreateInfoKHR

#### New Enums

- VkFenceImportFlagBitsKHR

#### New Bitmasks

- VkFenceImportFlagsKHR

#### New Enum Constants

- `VK_KHR_EXTERNAL_FENCE_EXTENSION_NAME`
- `VK_KHR_EXTERNAL_FENCE_SPEC_VERSION`
- Extending VkFenceImportFlagBits:
  - `VK_FENCE_IMPORT_TEMPORARY_BIT_KHR`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_EXPORT_FENCE_CREATE_INFO_KHR`

#### Issues

This extension borrows concepts, semantics, and language from `VK_KHR_external_semaphore`. That extension's issues apply equally to this extension.

#### Version History

- Revision 1, 2017-05-08 (Jesse Hall)
  - Initial revision

## VK_KHR_external_fence_capabilities

#### Name String

`VK_KHR_external_fence_capabilities`

#### Extension Type

Instance extension

#### Registered Extension Number

113

#### Revision

1

#### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### Deprecation State

- *Promoted* to Vulkan 1.1

### Contact

- Jesse Hall [GitHub]critsec

### Other Extension Metadata

### Last Modified Date

2017-05-08

### IP Status

No known IP claims.

### Contributors

- Jesse Hall, Google
- James Jones, NVIDIA
- Jeff Juliano, NVIDIA
- Cass Everitt, Oculus
- Contributors to `VK_KHR_external_semaphore_capabilities`

### Description

An application may wish to reference device fences in multiple Vulkan logical devices or instances, in multiple processes, and/or in multiple APIs. This extension provides a set of capability queries and handle definitions that allow an application to determine what types of “external” fence handles an implementation supports for a given set of use cases.

### Promotion to Vulkan 1.1

All functionality in this extension is included in core Vulkan 1.1, with the KHR suffix omitted. The original type, enum and command names are still available as aliases of the core functionality.

### New Commands

- vkGetPhysicalDeviceExternalFencePropertiesKHR

### New Structures

- VkExternalFencePropertiesKHR
- VkPhysicalDeviceExternalFenceInfoKHR
- Extending VkPhysicalDeviceProperties2:

- VkPhysicalDeviceIDPropertiesKHR

### New Enums

- VkExternalFenceFeatureFlagBitsKHR
- VkExternalFenceHandleTypeFlagBitsKHR

### New Bitmasks

- VkExternalFenceFeatureFlagsKHR
- VkExternalFenceHandleTypeFlagsKHR

### New Enum Constants

- `VK_KHR_EXTERNAL_FENCE_CAPABILITIES_EXTENSION_NAME`
- `VK_KHR_EXTERNAL_FENCE_CAPABILITIES_SPEC_VERSION`
- `VK_LUID_SIZE_KHR`
- Extending VkExternalFenceFeatureFlagBits:
  - `VK_EXTERNAL_FENCE_FEATURE_EXPORTABLE_BIT_KHR`
  - `VK_EXTERNAL_FENCE_FEATURE_IMPORTABLE_BIT_KHR`
- Extending VkExternalFenceHandleTypeFlagBits:
  - `VK_EXTERNAL_FENCE_HANDLE_TYPE_OPAQUE_FD_BIT_KHR`
  - `VK_EXTERNAL_FENCE_HANDLE_TYPE_OPAQUE_WIN32_BIT_KHR`
  - `VK_EXTERNAL_FENCE_HANDLE_TYPE_OPAQUE_WIN32_KMT_BIT_KHR`
  - `VK_EXTERNAL_FENCE_HANDLE_TYPE_SYNC_FD_BIT_KHR`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_EXTERNAL_FENCE_PROPERTIES_KHR`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTERNAL_FENCE_INFO_KHR`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_ID_PROPERTIES_KHR`

### Version History

- Revision 1, 2017-05-08 (Jesse Hall)
  - Initial version

## VK_KHR_external_memory

### Name String

`VK_KHR_external_memory`

### Extension Type

Device extension

**Registered Extension Number**

73

**Revision**

1

**Ratification Status**

Ratified

**Extension and Version Dependencies**

VK_KHR_external_memory_capabilities
or
Vulkan Version 1.1

**Deprecation State**

- *Promoted* to Vulkan 1.1

**Contact**

- James Jones [GitHub]cubanismo

**Other Extension Metadata**

**Last Modified Date**

2016-10-20

**IP Status**

No known IP claims.

**Interactions and External Dependencies**

- Interacts with `VK_KHR_dedicated_allocation`.
- Interacts with `VK_NV_dedicated_allocation`.

**Contributors**

- Faith Ekstrand, Intel
- Ian Elliott, Google
- Jesse Hall, Google
- Tobias Hector, Imagination Technologies
- James Jones, NVIDIA
- Jeff Juliano, NVIDIA
- Matthew Netsch, Qualcomm Technologies, Inc.
- Daniel Rakos, AMD
- Carsten Rohde, NVIDIA
- Ray Smith, ARM
- Lina Versace, Google

### Description

An application may wish to reference device memory in multiple Vulkan logical devices or instances, in multiple processes, and/or in multiple APIs. This extension enables an application to export non-Vulkan handles from Vulkan memory objects such that the underlying resources can be referenced outside the scope of the Vulkan logical device that created them.

### Promotion to Vulkan 1.1

All functionality in this extension is included in core Vulkan 1.1, with the KHR suffix omitted. The original type, enum and command names are still available as aliases of the core functionality.

### New Structures

- Extending VkBufferCreateInfo:
  - VkExternalMemoryBufferCreateInfoKHR
- Extending VkImageCreateInfo:
  - VkExternalMemoryImageCreateInfoKHR
- Extending VkMemoryAllocateInfo:
  - VkExportMemoryAllocateInfoKHR

### New Enum Constants

- `VK_KHR_EXTERNAL_MEMORY_EXTENSION_NAME`
- `VK_KHR_EXTERNAL_MEMORY_SPEC_VERSION`
- `VK_QUEUE_FAMILY_EXTERNAL_KHR`
- Extending VkResult:
  - `VK_ERROR_INVALID_EXTERNAL_HANDLE_KHR`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_EXPORT_MEMORY_ALLOCATE_INFO_KHR`
  - `VK_STRUCTURE_TYPE_EXTERNAL_MEMORY_BUFFER_CREATE_INFO_KHR`
  - `VK_STRUCTURE_TYPE_EXTERNAL_MEMORY_IMAGE_CREATE_INFO_KHR`

### Issues

1) How do applications correlate two physical devices across process or Vulkan instance boundaries?

**RESOLVED**: New device ID fields have been introduced by `VK_KHR_external_memory_capabilities`. These fields, combined with the existing VkPhysicalDeviceProperties::`driverVersion` field can be used to identify compatible devices across processes, drivers, and APIs. VkPhysicalDeviceProperties::`pipelineCacheUUID` is not sufficient for this purpose because despite its description in the specification, it need only identify a unique pipeline cache format in practice. Multiple devices may be able to use the same pipeline cache data, and hence it would be desirable for all of them to have the same pipeline cache UUID. However, only the same concrete physical

device can be used when sharing memory, so an actual unique device ID was introduced. Further, the pipeline cache UUID was specific to Vulkan, but correlation with other, non-extensible APIs is required to enable interoperation with those APIs.

2) If memory objects are shared between processes and APIs, is this considered aliasing according to the rules outlined in the Memory Aliasing section?

**RESOLVED**: Yes. Applications must take care to obey all restrictions imposed on aliased resources when using memory across multiple Vulkan instances or other APIs.

3) Are new image layouts or metadata required to specify image layouts and layout transitions compatible with non-Vulkan APIs, or with other instances of the same Vulkan driver?

**RESOLVED**: Separate instances of the same Vulkan driver running on the same GPU should have identical internal layout semantics, so applications have the tools they need to ensure views of images are consistent between the two instances. Other APIs will fall into two categories: Those that are Vulkan- compatible, and those that are Vulkan-incompatible. Vulkan-incompatible APIs will require the image to be in the GENERAL layout whenever they are accessing them.

Note this does not attempt to address cross-device transitions, nor transitions to engines on the same device which are not visible within the Vulkan API. Both of these are beyond the scope of this extension.

4) Is a new barrier flag or operation of some type needed to prepare external memory for handoff to another Vulkan instance or API and/or receive it from another instance or API?

**RESOLVED**: Yes. Some implementations need to perform additional cache management when transitioning memory between address spaces and other APIs, instances, or processes which may operate in a separate address space. Options for defining this transition include:

- A new structure that can be added to the `pNext` list in VkMemoryBarrier, VkBufferMemoryBarrier, and VkImageMemoryBarrier.
- A new bit in VkAccessFlags that can be set to indicate an “external” access.
- A new bit in VkDependencyFlags
- A new special queue family that represents an “external” queue.

A new structure has the advantage that the type of external transition can be described in as much detail as necessary. However, there is not currently a known need for anything beyond differentiating between external and internal accesses, so this is likely an over-engineered solution. The access flag bit has the advantage that it can be applied at buffer, image, or global granularity, and semantically it maps pretty well to the operation being described. Additionally, the API already includes `VK_ACCESS_MEMORY_READ_BIT` and `VK_ACCESS_MEMORY_WRITE_BIT` which appear to be intended for this purpose. However, there is no obvious pipeline stage that would correspond to an external access, and therefore no clear way to use `VK_ACCESS_MEMORY_READ_BIT` or `VK_ACCESS_MEMORY_WRITE_BIT`. VkDependencyFlags and VkPipelineStageFlags operate at command granularity rather than image or buffer granularity, which would make an entire pipeline barrier an internal→external or external→internal barrier. This may not be a problem in practice, but seems like the wrong scope. Another downside of VkDependencyFlags is that it lacks inherent directionality: there are no `src` and `dst` variants of it in the barrier or dependency description semantics, so two bits might need to

be added to describe both internal→external and external→internal transitions. Transitioning a resource to a special queue family corresponds well with the operation of transitioning to a separate Vulkan instance, in that both operations ideally include scheduling a barrier on both sides of the transition: Both the releasing and the acquiring queue or process. Using a special queue family requires adding an additional reserved queue family index. Re-using `VK_QUEUE_FAMILY_IGNORED` would have left it unclear how to transition a concurrent usage resource from one process to another, since the semantics would have likely been equivalent to the currently-ignored transition of `VK_QUEUE_FAMILY_IGNORED` → `VK_QUEUE_FAMILY_IGNORED`. Fortunately, creating a new reserved queue family index is not invasive.

Based on the above analysis, the approach of transitioning to a special "external" queue family was chosen.

5) Do internal driver memory arrangements and/or other internal driver image properties need to be exported and imported when sharing images across processes or APIs.

**RESOLVED**: Some vendors claim this is necessary on their implementations, but it was determined that the security risks of allowing opaque metadata to be passed from applications to the driver were too high. Therefore, implementations which require metadata will need to associate it with the objects represented by the external handles, and rely on the dedicated allocation mechanism to associate the exported and imported memory objects with a single image or buffer.

6) Most prior interoperation and cross-process sharing APIs have been based on image-level sharing. Should Vulkan sharing be based on memory-object sharing or image sharing?

**RESOLVED**: These extensions have assumed memory-level sharing is the correct granularity. Vulkan is a lower-level API than most prior APIs, and as such attempts to closely align with to the underlying primitives of the hardware and system-level drivers it abstracts. In general, the resource that holds the backing store for both images and buffers of various types is memory. Images and buffers are merely metadata containing brief descriptions of the layout of bits within that memory.

Because memory object-based sharing is aligned with the overall Vulkan API design, it enables the full range of Vulkan capabilities with external objects. External memory can be used as backing for sparse images, for example, whereas such usage would be awkward at best with a sharing mechanism based on higher-level primitives such as images. Further, aligning the mechanism with the API in this way provides some hope of trivial compatibility with future API enhancements. If new objects backed by memory objects are added to the API, they too can be used across processes with minimal additions to the base external memory APIs.

Earlier APIs implemented interop at a higher level, and this necessitated entirely separate sharing APIs for images and buffers. To co-exist and interoperate with those APIs, the Vulkan external sharing mechanism must accommodate their model. However, if it can be agreed that memory-based sharing is the more desirable and forward-looking design, legacy interoperation constraints can be considered another reason to favor memory-based sharing: while native and legacy driver primitives that may be used to implement sharing may not be as low-level as the API here suggests, raw memory is still the least common denominator among the types. Image-based sharing can be cleanly derived from a set of base memory- object sharing APIs with minimal effort, whereas image-based sharing does not generalize well to buffer or raw-memory sharing. Therefore,

following the general Vulkan design principle of minimalism, it is better to expose interopability with image-based native and external primitives via the memory sharing API, and place sufficient limits on their usage to ensure they can be used only as backing for equivalent Vulkan images. This provides a consistent API for applications regardless of which platform or external API they are targeting, which makes development of multi-API and multi-platform applications simpler.

7) Should Vulkan define a common external handle type and provide Vulkan functions to facilitate cross-process sharing of such handles rather than relying on native handles to define the external objects?

**RESOLVED**: No. Cross-process sharing of resources is best left to native platforms. There are myriad security and extensibility issues with such a mechanism, and attempting to re-solve all those issues within Vulkan does not align with Vulkan's purpose as a graphics API. If desired, such a mechanism could be built as a layer or helper library on top of the opaque native handle defined in this family of extensions.

8) Must implementations provide additional guarantees about state implicitly included in memory objects for those memory objects that may be exported?

**RESOLVED**: Implementations must ensure that sharing memory objects does not transfer any information between the exporting and importing instances and APIs other than that required to share the data contained in the memory objects explicitly shared. As specific examples, data from previously freed memory objects that used the same underlying physical memory, and data from memory objects using adjacent physical memory must not be visible to applications importing an exported memory object.

9) Must implementations validate external handles the application provides as inputs to memory import operations?

**RESOLVED**: Implementations must return an error to the application if the provided memory handle cannot be used to complete the requested import operation. However, implementations need not validate handles are of the exact type specified by the application.

### Version History

- Revision 1, 2016-10-20 (James Jones)
  - Initial version

## VK_KHR_external_memory_capabilities

### Name String

`VK_KHR_external_memory_capabilities`

### Extension Type

Instance extension

### Registered Extension Number

72

### Revision

1

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### Deprecation State

- *Promoted* to Vulkan 1.1

### Contact

- James Jones [GitHub]cubanismo

### Other Extension Metadata

### Last Modified Date

2016-10-17

### IP Status

No known IP claims.

### Interactions and External Dependencies

- Interacts with `VK_KHR_dedicated_allocation`.
- Interacts with `VK_NV_dedicated_allocation`.

### Contributors

- Ian Elliott, Google
- Jesse Hall, Google
- James Jones, NVIDIA

### Description

An application may wish to reference device memory in multiple Vulkan logical devices or instances, in multiple processes, and/or in multiple APIs. This extension provides a set of capability queries and handle definitions that allow an application to determine what types of “external” memory handles an implementation supports for a given set of use cases.

### Promotion to Vulkan 1.1

All functionality in this extension is included in core Vulkan 1.1, with the KHR suffix omitted. The original type, enum and command names are still available as aliases of the core functionality.

### New Commands

- vkGetPhysicalDeviceExternalBufferPropertiesKHR

### New Structures

- VkExternalBufferPropertiesKHR
- VkExternalMemoryPropertiesKHR
- VkPhysicalDeviceExternalBufferInfoKHR
- Extending VkImageFormatProperties2:
  - VkExternalImageFormatPropertiesKHR
- Extending VkPhysicalDeviceImageFormatInfo2:
  - VkPhysicalDeviceExternalImageFormatInfoKHR
- Extending VkPhysicalDeviceProperties2:
  - VkPhysicalDeviceIDPropertiesKHR

### New Enums

- VkExternalMemoryFeatureFlagBitsKHR
- VkExternalMemoryHandleTypeFlagBitsKHR

### New Bitmasks

- VkExternalMemoryFeatureFlagsKHR
- VkExternalMemoryHandleTypeFlagsKHR

### New Enum Constants

- `VK_KHR_EXTERNAL_MEMORY_CAPABILITIES_EXTENSION_NAME`
- `VK_KHR_EXTERNAL_MEMORY_CAPABILITIES_SPEC_VERSION`
- `VK_LUID_SIZE_KHR`
- Extending VkExternalMemoryFeatureFlagBits:
  - `VK_EXTERNAL_MEMORY_FEATURE_DEDICATED_ONLY_BIT_KHR`
  - `VK_EXTERNAL_MEMORY_FEATURE_EXPORTABLE_BIT_KHR`
  - `VK_EXTERNAL_MEMORY_FEATURE_IMPORTABLE_BIT_KHR`
- Extending VkExternalMemoryHandleTypeFlagBits:
  - `VK_EXTERNAL_MEMORY_HANDLE_TYPE_D3D11_TEXTURE_BIT_KHR`
  - `VK_EXTERNAL_MEMORY_HANDLE_TYPE_D3D11_TEXTURE_KMT_BIT_KHR`
  - `VK_EXTERNAL_MEMORY_HANDLE_TYPE_D3D12_HEAP_BIT_KHR`
  - `VK_EXTERNAL_MEMORY_HANDLE_TYPE_D3D12_RESOURCE_BIT_KHR`
  - `VK_EXTERNAL_MEMORY_HANDLE_TYPE_OPAQUE_FD_BIT_KHR`

    - VK_EXTERNAL_MEMORY_HANDLE_TYPE_OPAQUE_WIN32_BIT_KHR
    - VK_EXTERNAL_MEMORY_HANDLE_TYPE_OPAQUE_WIN32_KMT_BIT_KHR
- Extending VkStructureType:
    - VK_STRUCTURE_TYPE_EXTERNAL_BUFFER_PROPERTIES_KHR
    - VK_STRUCTURE_TYPE_EXTERNAL_IMAGE_FORMAT_PROPERTIES_KHR
    - VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTERNAL_BUFFER_INFO_KHR
    - VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTERNAL_IMAGE_FORMAT_INFO_KHR
    - VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_ID_PROPERTIES_KHR

**Issues**

1) Why do so many external memory capabilities need to be queried on a per-memory-handle-type basis?

**PROPOSED RESOLUTION**: This is because some handle types are based on OS-native objects that have far more limited capabilities than the very generic Vulkan memory objects. Not all memory handle types can name memory objects that support 3D images, for example. Some handle types cannot even support the deferred image and memory binding behavior of Vulkan and require specifying the image when allocating or importing the memory object.

2) Do the VkExternalImageFormatPropertiesKHR and VkExternalBufferPropertiesKHR structs need to include a list of memory type bits that support the given handle type?

**PROPOSED RESOLUTION**: No. The memory types that do not support the handle types will simply be filtered out of the results returned by vkGetImageMemoryRequirements and vkGetBufferMemoryRequirements when a set of handle types was specified at image or buffer creation time.

3) Should the non-opaque handle types be moved to their own extension?

**PROPOSED RESOLUTION**: Perhaps. However, defining the handle type bits does very little and does not require any platform-specific types on its own, and it is easier to maintain the bitfield values in a single extension for now. Presumably more handle types could be added by separate extensions though, and it would be midly weird to have some platform-specific ones defined in the core spec and some in extensions

4) Do we need a D3D11_TILEPOOL type?

**PROPOSED RESOLUTION**: No. This is technically possible, but the synchronization is awkward. D3D11 surfaces must be synchronized using shared mutexes, and these synchronization primitives are shared by the entire memory object, so D3D11 shared allocations divided among multiple buffer and image bindings may be difficult to synchronize.

5) Should the Windows 7-compatible handle types be named “KMT” handles or “GLOBAL_SHARE” handles?

**PROPOSED RESOLUTION**: KMT, simply because it is more concise.

6) How do applications identify compatible devices and drivers across instance, process, and API boundaries when sharing memory?

**PROPOSED RESOLUTION**: New device properties are exposed that allow applications to correctly correlate devices and drivers. A device and driver UUID that must both match to ensure sharing compatibility between two Vulkan instances, or a Vulkan instance and an extensible external API are added. To allow correlating with Direct3D devices, a device LUID is added that corresponds to a DXGI adapter LUID. A driver ID is not needed for Direct3D because mismatched driver component versions are not currently supported on the Windows OS. Should support for such configurations be introduced at the OS level, further Vulkan extensions would be needed to correlate userspace component builds.

### Version History

- Revision 1, 2016-10-17 (James Jones)
  - Initial version

## VK_KHR_external_semaphore

### Name String

`VK_KHR_external_semaphore`

### Extension Type

Device extension

### Registered Extension Number

78

### Revision

1

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_external_semaphore_capabilities

### Deprecation State

- *Promoted* to Vulkan 1.1

### Contact

- James Jones [GitHub]cubanismo

### Other Extension Metadata

### Last Modified Date

2016-10-21

### IP Status

No known IP claims.

### Contributors

- Faith Ekstrand, Intel
- Jesse Hall, Google
- Tobias Hector, Imagination Technologies
- James Jones, NVIDIA
- Jeff Juliano, NVIDIA
- Matthew Netsch, Qualcomm Technologies, Inc.
- Ray Smith, ARM
- Lina Versace, Google

### Description

An application using external memory may wish to synchronize access to that memory using semaphores. This extension enables an application to create semaphores from which non-Vulkan handles that reference the underlying synchronization primitive can be exported.

### Promotion to Vulkan 1.1

All functionality in this extension is included in core Vulkan 1.1, with the KHR suffix omitted. The original type, enum and command names are still available as aliases of the core functionality.

### New Structures

- Extending VkSemaphoreCreateInfo:
  - VkExportSemaphoreCreateInfoKHR

### New Enums

- VkSemaphoreImportFlagBitsKHR

### New Bitmasks

- VkSemaphoreImportFlagsKHR

### New Enum Constants

- `VK_KHR_EXTERNAL_SEMAPHORE_EXTENSION_NAME`
- `VK_KHR_EXTERNAL_SEMAPHORE_SPEC_VERSION`
- Extending VkSemaphoreImportFlagBits:
  - `VK_SEMAPHORE_IMPORT_TEMPORARY_BIT_KHR`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_EXPORT_SEMAPHORE_CREATE_INFO_KHR`

### Issues

1) Should there be restrictions on what side effects can occur when waiting on imported semaphores that are in an invalid state?

**RESOLVED**: Yes. Normally, validating such state would be the responsibility of the application, and the implementation would be free to enter an undefined state if valid usage rules were violated. However, this could cause security concerns when using imported semaphores, as it would require the importing application to trust the exporting application to ensure the state is valid. Requiring this level of trust is undesirable for many potential use cases.

2) Must implementations validate external handles the application provides as input to semaphore state import operations?

**RESOLVED**: Implementations must return an error to the application if the provided semaphore state handle cannot be used to complete the requested import operation. However, implementations need not validate handles are of the exact type specified by the application.

### Version History

- Revision 1, 2016-10-21 (James Jones)
  - Initial revision

## VK_KHR_external_semaphore_capabilities

### Name String

`VK_KHR_external_semaphore_capabilities`

### Extension Type

Instance extension

### Registered Extension Number

77

### Revision

1

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### Deprecation State

- *Promoted* to Vulkan 1.1

### Contact

- James Jones [GitHub]cubanismo

### Other Extension Metadata

### Last Modified Date

2016-10-20

### IP Status

No known IP claims.

### Contributors

- Jesse Hall, Google
- James Jones, NVIDIA
- Jeff Juliano, NVIDIA

### Description

An application may wish to reference device semaphores in multiple Vulkan logical devices or instances, in multiple processes, and/or in multiple APIs. This extension provides a set of capability queries and handle definitions that allow an application to determine what types of “external” semaphore handles an implementation supports for a given set of use cases.

### Promotion to Vulkan 1.1

All functionality in this extension is included in core Vulkan 1.1, with the KHR suffix omitted. The original type, enum and command names are still available as aliases of the core functionality.

### New Commands

- vkGetPhysicalDeviceExternalSemaphorePropertiesKHR

### New Structures

- VkExternalSemaphorePropertiesKHR
- VkPhysicalDeviceExternalSemaphoreInfoKHR
- Extending VkPhysicalDeviceProperties2:
  - VkPhysicalDeviceIDPropertiesKHR

### New Enums

- VkExternalSemaphoreFeatureFlagBitsKHR
- VkExternalSemaphoreHandleTypeFlagBitsKHR

### New Bitmasks

- VkExternalSemaphoreFeatureFlagsKHR

- VkExternalSemaphoreHandleTypeFlagsKHR

### New Enum Constants

- VK_KHR_EXTERNAL_SEMAPHORE_CAPABILITIES_EXTENSION_NAME
- VK_KHR_EXTERNAL_SEMAPHORE_CAPABILITIES_SPEC_VERSION
- VK_LUID_SIZE_KHR
- Extending VkExternalSemaphoreFeatureFlagBits:
  - VK_EXTERNAL_SEMAPHORE_FEATURE_EXPORTABLE_BIT_KHR
  - VK_EXTERNAL_SEMAPHORE_FEATURE_IMPORTABLE_BIT_KHR
- Extending VkExternalSemaphoreHandleTypeFlagBits:
  - VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_D3D12_FENCE_BIT_KHR
  - VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_OPAQUE_FD_BIT_KHR
  - VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_OPAQUE_WIN32_BIT_KHR
  - VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_OPAQUE_WIN32_KMT_BIT_KHR
  - VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_SYNC_FD_BIT_KHR
- Extending VkStructureType:
  - VK_STRUCTURE_TYPE_EXTERNAL_SEMAPHORE_PROPERTIES_KHR
  - VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTERNAL_SEMAPHORE_INFO_KHR
  - VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_ID_PROPERTIES_KHR

### Version History

- Revision 1, 2016-10-20 (James Jones)
  - Initial revision

## VK_KHR_get_memory_requirements2

### Name String

VK_KHR_get_memory_requirements2

### Extension Type

Device extension

### Registered Extension Number

147

### Revision

1

### Ratification Status

Ratified

### Extension and Version Dependencies

None

### Deprecation State

- *Promoted* to Vulkan 1.1

### Contact

- Faith Ekstrand [GitHub]gfxstrand

### Other Extension Metadata

### Last Modified Date

2017-09-05

### IP Status

No known IP claims.

### Contributors

- Faith Ekstrand, Intel
- Jeff Bolz, NVIDIA
- Jesse Hall, Google

### Description

This extension provides new queries for memory requirements of images and buffers that can be easily extended by other extensions, without introducing any additional commands. The Vulkan 1.0 VkMemoryRequirements and VkSparseImageMemoryRequirements structures do not include `sType` and `pNext` members. This extension wraps them in new structures with these members, so an application can query a chain of memory requirements structures by constructing the chain and letting the implementation fill them in. A new command is added for each `vkGet*MemoryRequrements` command in core Vulkan 1.0.

### Promotion to Vulkan 1.1

All functionality in this extension is included in core Vulkan 1.1, with the KHR suffix omitted. The original type, enum and command names are still available as aliases of the core functionality.

### New Commands

- vkGetBufferMemoryRequirements2KHR
- vkGetImageMemoryRequirements2KHR
- vkGetImageSparseMemoryRequirements2KHR

### New Structures

- VkBufferMemoryRequirementsInfo2KHR
- VkImageMemoryRequirementsInfo2KHR

- VkImageSparseMemoryRequirementsInfo2KHR
- VkMemoryRequirements2KHR
- VkSparseImageMemoryRequirements2KHR

#### New Enum Constants

- `VK_KHR_GET_MEMORY_REQUIREMENTS_2_EXTENSION_NAME`
- `VK_KHR_GET_MEMORY_REQUIREMENTS_2_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_BUFFER_MEMORY_REQUIREMENTS_INFO_2_KHR`
  - `VK_STRUCTURE_TYPE_IMAGE_MEMORY_REQUIREMENTS_INFO_2_KHR`
  - `VK_STRUCTURE_TYPE_IMAGE_SPARSE_MEMORY_REQUIREMENTS_INFO_2_KHR`
  - `VK_STRUCTURE_TYPE_MEMORY_REQUIREMENTS_2_KHR`
  - `VK_STRUCTURE_TYPE_SPARSE_IMAGE_MEMORY_REQUIREMENTS_2_KHR`

#### Version History

- Revision 1, 2017-03-23 (Faith Ekstrand)
  - Internal revisions

## VK_KHR_get_physical_device_properties2

#### Name String

`VK_KHR_get_physical_device_properties2`

#### Extension Type

Instance extension

#### Registered Extension Number

60

#### Revision

2

#### Ratification Status

Ratified

#### Extension and Version Dependencies

None

#### Deprecation State

- *Promoted* to Vulkan 1.1

#### Contact

- Jeff Bolz [GitHub]jeffbolznv

### Other Extension Metadata

#### Last Modified Date

2017-09-05

#### IP Status

No known IP claims.

#### Contributors

- Jeff Bolz, NVIDIA
- Ian Elliott, Google

#### Description

This extension provides new queries for device features, device properties, and format properties that can be easily extended by other extensions, without introducing any further queries. The Vulkan 1.0 feature/limit/formatproperty structures do not include `sType`/`pNext` members. This extension wraps them in new structures with `sType`/`pNext` members, so an application can query a chain of feature/limit/formatproperty structures by constructing the chain and letting the implementation fill them in. A new command is added for each `vkGetPhysicalDevice*` command in core Vulkan 1.0. The new feature structure (and a `pNext` chain of extending structures) can also be passed in to device creation to enable features.

This extension also allows applications to use the physical-device components of device extensions before vkCreateDevice is called.

#### Promotion to Vulkan 1.1

All functionality in this extension is included in core Vulkan 1.1, with the KHR suffix omitted. The original type, enum and command names are still available as aliases of the core functionality.

#### New Commands

- vkGetPhysicalDeviceFeatures2KHR
- vkGetPhysicalDeviceFormatProperties2KHR
- vkGetPhysicalDeviceImageFormatProperties2KHR
- vkGetPhysicalDeviceMemoryProperties2KHR
- vkGetPhysicalDeviceProperties2KHR
- vkGetPhysicalDeviceQueueFamilyProperties2KHR
- vkGetPhysicalDeviceSparseImageFormatProperties2KHR

#### New Structures

- VkFormatProperties2KHR
- VkImageFormatProperties2KHR
- VkPhysicalDeviceImageFormatInfo2KHR

- VkPhysicalDeviceMemoryProperties2KHR
- VkPhysicalDeviceProperties2KHR
- VkPhysicalDeviceSparseImageFormatInfo2KHR
- VkQueueFamilyProperties2KHR
- VkSparseImageFormatProperties2KHR
- Extending VkDeviceCreateInfo:
  - VkPhysicalDeviceFeatures2KHR

### New Enum Constants

- VK_KHR_GET_PHYSICAL_DEVICE_PROPERTIES_2_EXTENSION_NAME
- VK_KHR_GET_PHYSICAL_DEVICE_PROPERTIES_2_SPEC_VERSION
- Extending VkStructureType:
  - VK_STRUCTURE_TYPE_FORMAT_PROPERTIES_2_KHR
  - VK_STRUCTURE_TYPE_IMAGE_FORMAT_PROPERTIES_2_KHR
  - VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_FEATURES_2_KHR
  - VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_IMAGE_FORMAT_INFO_2_KHR
  - VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MEMORY_PROPERTIES_2_KHR
  - VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PROPERTIES_2_KHR
  - VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SPARSE_IMAGE_FORMAT_INFO_2_KHR
  - VK_STRUCTURE_TYPE_QUEUE_FAMILY_PROPERTIES_2_KHR
  - VK_STRUCTURE_TYPE_SPARSE_IMAGE_FORMAT_PROPERTIES_2_KHR

### Examples

```
// Get features with a hypothetical future extension.
VkHypotheticalExtensionFeaturesKHR hypotheticalFeatures =
{
    .sType = VK_STRUCTURE_TYPE_HYPOTHETICAL_FEATURES_KHR,
    .pNext = NULL,
};

VkPhysicalDeviceFeatures2KHR features =
{
    .sType = VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_FEATURES_2_KHR,
    .pNext = &hypotheticalFeatures,
};

// After this call, features and hypotheticalFeatures have been filled out.
vkGetPhysicalDeviceFeatures2KHR(physicalDevice, &features);

// Properties/limits can be chained and queried similarly.
```

```
// Enable some features:
VkHypotheticalExtensionFeaturesKHR enabledHypotheticalFeatures =
{
    .sType = VK_STRUCTURE_TYPE_HYPOTHETICAL_FEATURES_KHR,
    .pNext = NULL,
};

VkPhysicalDeviceFeatures2KHR enabledFeatures =
{
    .sType = VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_FEATURES_2_KHR,
    .pNext = &enabledHypotheticalFeatures,
};

enabledFeatures.features.xyz = VK_TRUE;
enabledHypotheticalFeatures.abc = VK_TRUE;

VkDeviceCreateInfo deviceCreateInfo =
{
    .sType = VK_STRUCTURE_TYPE_DEVICE_CREATE_INFO,
    .pNext = &enabledFeatures,
    ...
    .pEnabledFeatures = NULL,
};

VkDevice device;
vkCreateDevice(physicalDevice, &deviceCreateInfo, NULL, &device);
```

### Version History

- Revision 1, 2016-09-12 (Jeff Bolz)
  - Internal revisions
- Revision 2, 2016-11-02 (Ian Elliott)
  - Added ability for applications to use the physical-device components of device extensions before vkCreateDevice is called.

## VK_KHR_maintenance1

### Name String

`VK_KHR_maintenance1`

### Extension Type

Device extension

### Registered Extension Number

70

**Revision**

2

**Ratification Status**

Ratified

**Extension and Version Dependencies**

None

**Deprecation State**

- *Promoted* to Vulkan 1.1

**Contact**

- Piers Daniell [GitHub]pdaniell-nv

**Other Extension Metadata**

**Last Modified Date**

2018-03-13

**Contributors**

- Dan Ginsburg, Valve
- Daniel Koch, NVIDIA
- Daniel Rakos, AMD
- Jan-Harald Fredriksen, ARM
- Faith Ekstrand, Intel
- Jeff Bolz, NVIDIA
- Jesse Hall, Google
- John Kessenich, Google
- Michael Worcester, Imagination Technologies
- Neil Henning, Codeplay Software Ltd.
- Piers Daniell, NVIDIA
- Slawomir Grajewski, Intel
- Tobias Hector, Imagination Technologies
- Tom Olson, ARM

**Description**

`VK_KHR_maintenance1` adds a collection of minor features that were intentionally left out or overlooked from the original Vulkan 1.0 release.

The new features are as follows:

- Allow 2D and 2D array image views to be created from 3D images, which can then be used as

color framebuffer attachments. This allows applications to render to slices of a 3D image.

- Support vkCmdCopyImage between 2D array layers and 3D slices. This extension allows copying from layers of a 2D array image to slices of a 3D image and vice versa.
- Allow negative height to be specified in the VkViewport::`height` field to perform y-inversion of the clip-space to framebuffer-space transform. This allows apps to avoid having to use `gl_Position.y = -gl_Position.y` in shaders also targeting other APIs.
- Allow implementations to express support for doing just transfers and clears of image formats that they otherwise support no other format features for. This is done by adding new format feature flags `VK_FORMAT_FEATURE_TRANSFER_SRC_BIT_KHR` and `VK_FORMAT_FEATURE_TRANSFER_DST_BIT_KHR`.
- Support vkCmdFillBuffer on transfer-only queues. Previously vkCmdFillBuffer was defined to only work on command buffers allocated from command pools which support graphics or compute queues. It is now allowed on queues that just support transfer operations.
- Fix the inconsistency of how error conditions are returned between the vkCreateGraphicsPipelines and vkCreateComputePipelines functions and the vkAllocateDescriptorSets and vkAllocateCommandBuffers functions.
- Add new `VK_ERROR_OUT_OF_POOL_MEMORY_KHR` error so implementations can give a more precise reason for vkAllocateDescriptorSets failures.
- Add a new command vkTrimCommandPoolKHR which gives the implementation an opportunity to release any unused command pool memory back to the system.

#### Promotion to Vulkan 1.1

All functionality in this extension is included in core Vulkan 1.1, with the KHR suffix omitted. The original type, enum and command names are still available as aliases of the core functionality.

#### New Commands

- vkTrimCommandPoolKHR

#### New Bitmasks

- VkCommandPoolTrimFlagsKHR

#### New Enum Constants

- `VK_KHR_MAINTENANCE1_EXTENSION_NAME`
- `VK_KHR_MAINTENANCE1_SPEC_VERSION`
- `VK_KHR_MAINTENANCE_1_EXTENSION_NAME`
- `VK_KHR_MAINTENANCE_1_SPEC_VERSION`
- Extending VkFormatFeatureFlagBits:
  - `VK_FORMAT_FEATURE_TRANSFER_DST_BIT_KHR`
  - `VK_FORMAT_FEATURE_TRANSFER_SRC_BIT_KHR`
- Extending VkImageCreateFlagBits:

    - `VK_IMAGE_CREATE_2D_ARRAY_COMPATIBLE_BIT_KHR`
- Extending VkResult:
    - `VK_ERROR_OUT_OF_POOL_MEMORY_KHR`

#### Issues

1. Are viewports with zero height allowed?

   **RESOLVED**: Yes, although they have low utility.

#### Version History

- Revision 1, 2016-10-26 (Piers Daniell)
    - Internal revisions
- Revision 2, 2018-03-13 (Jon Leech)
    - Add issue for zero-height viewports

## VK_KHR_maintenance2

#### Name String

`VK_KHR_maintenance2`

#### Extension Type

Device extension

#### Registered Extension Number

118

#### Revision

1

#### Ratification Status

Ratified

#### Extension and Version Dependencies

None

#### Deprecation State

- *Promoted* to Vulkan 1.1

#### Contact

- Michael Worcester [GitHub]michaelworcester

#### Other Extension Metadata

#### Last Modified Date

2017-09-05

### Contributors

- Michael Worcester, Imagination Technologies
- Stuart Smith, Imagination Technologies
- Jeff Bolz, NVIDIA
- Daniel Koch, NVIDIA
- Jan-Harald Fredriksen, ARM
- Daniel Rakos, AMD
- Neil Henning, Codeplay
- Piers Daniell, NVIDIA

### Description

`VK_KHR_maintenance2` adds a collection of minor features that were intentionally left out or overlooked from the original Vulkan 1.0 release.

The new features are as follows:

- Allow the application to specify which aspect of an input attachment might be read for a given subpass.
- Allow implementations to express the clipping behavior of points.
- Allow creating images with usage flags that may not be supported for the base image's format, but are supported for image views of the image that have a different but compatible format.
- Allow creating uncompressed image views of compressed images.
- Allow the application to select between an upper-left and lower-left origin for the tessellation domain space.
- Adds two new image layouts for depth stencil images to allow either the depth or stencil aspect to be read-only while the other aspect is writable.

### Input Attachment Specification

Input attachment specification allows an application to specify which aspect of a multi-aspect image (e.g. a depth/stencil format) will be accessed via a `subpassLoad` operation.

On some implementations there **may** be a performance penalty if the implementation does not know (at vkCreateRenderPass time) which aspect(s) of multi-aspect images **can** be accessed as input attachments.

### Promotion to Vulkan 1.1

All functionality in this extension is included in core Vulkan 1.1, with the KHR suffix omitted. The original type, enum and command names are still available as aliases of the core functionality.

### New Structures

- VkInputAttachmentAspectReferenceKHR

- Extending VkImageViewCreateInfo:
  - VkImageViewUsageCreateInfoKHR
- Extending VkPhysicalDeviceProperties2:
  - VkPhysicalDevicePointClippingPropertiesKHR
- Extending VkPipelineTessellationStateCreateInfo:
  - VkPipelineTessellationDomainOriginStateCreateInfoKHR
- Extending VkRenderPassCreateInfo:
  - VkRenderPassInputAttachmentAspectCreateInfoKHR

### New Enums

- VkPointClippingBehaviorKHR
- VkTessellationDomainOriginKHR

### New Enum Constants

- `VK_KHR_MAINTENANCE2_EXTENSION_NAME`
- `VK_KHR_MAINTENANCE2_SPEC_VERSION`
- `VK_KHR_MAINTENANCE_2_EXTENSION_NAME`
- `VK_KHR_MAINTENANCE_2_SPEC_VERSION`
- Extending VkImageCreateFlagBits:
  - `VK_IMAGE_CREATE_BLOCK_TEXEL_VIEW_COMPATIBLE_BIT_KHR`
  - `VK_IMAGE_CREATE_EXTENDED_USAGE_BIT_KHR`
- Extending VkImageLayout:
  - `VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_STENCIL_READ_ONLY_OPTIMAL_KHR`
  - `VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_STENCIL_ATTACHMENT_OPTIMAL_KHR`
- Extending VkPointClippingBehavior:
  - `VK_POINT_CLIPPING_BEHAVIOR_ALL_CLIP_PLANES_KHR`
  - `VK_POINT_CLIPPING_BEHAVIOR_USER_CLIP_PLANES_ONLY_KHR`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_IMAGE_VIEW_USAGE_CREATE_INFO_KHR`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_POINT_CLIPPING_PROPERTIES_KHR`
  - `VK_STRUCTURE_TYPE_PIPELINE_TESSELLATION_DOMAIN_ORIGIN_STATE_CREATE_INFO_KHR`
  - `VK_STRUCTURE_TYPE_RENDER_PASS_INPUT_ATTACHMENT_ASPECT_CREATE_INFO_KHR`
- Extending VkTessellationDomainOrigin:
  - `VK_TESSELLATION_DOMAIN_ORIGIN_LOWER_LEFT_KHR`
  - `VK_TESSELLATION_DOMAIN_ORIGIN_UPPER_LEFT_KHR`

#### Input Attachment Specification Example

Consider the case where a render pass has two subpasses and two attachments.

Attachment 0 has the format `VK_FORMAT_D24_UNORM_S8_UINT`, attachment 1 has some color format.

Subpass 0 writes to attachment 0, subpass 1 reads only the depth information from attachment 0 (using inputAttachmentRead) and writes to attachment 1.

```
    VkInputAttachmentAspectReferenceKHR references[] = {
        {
            .subpass = 1,
            .inputAttachmentIndex = 0,
            .aspectMask = VK_IMAGE_ASPECT_DEPTH_BIT
        }
    };

    VkRenderPassInputAttachmentAspectCreateInfoKHR specifyAspects = {
        .sType =
VK_STRUCTURE_TYPE_RENDER_PASS_INPUT_ATTACHMENT_ASPECT_CREATE_INFO_KHR,
        .pNext = NULL,
        .aspectReferenceCount = 1,
        .pAspectReferences = references
    };


    VkRenderPassCreateInfo createInfo = {
        ...
        .pNext = &specifyAspects,
        ...
    };

    vkCreateRenderPass(...);
```

#### Issues

1) What is the default tessellation domain origin?

**RESOLVED**: Vulkan 1.0 originally inadvertently documented a lower-left origin, but the conformance tests and all implementations implemented an upper-left origin. This extension adds a control to select between lower-left (for compatibility with OpenGL) and upper-left, and we retroactively fix unextended Vulkan to have a default of an upper-left origin.

#### Version History

- Revision 1, 2017-04-28

## VK_KHR_maintenance3

**Name String**

`VK_KHR_maintenance3`

**Extension Type**

Device extension

**Registered Extension Number**

169

**Revision**

1

**Ratification Status**

Ratified

**Extension and Version Dependencies**

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

**Deprecation State**

- *Promoted* to Vulkan 1.1

**Contact**

- Jeff Bolz [GitHub]jeffbolznv

**Other Extension Metadata**

**Last Modified Date**

2017-09-05

**Contributors**

- Jeff Bolz, NVIDIA

**Description**

`VK_KHR_maintenance3` adds a collection of minor features that were intentionally left out or overlooked from the original Vulkan 1.0 release.

The new features are as follows:

- A limit on the maximum number of descriptors that are supported in a single descriptor set layout. Some implementations have a limit on the total size of descriptors in a set, which cannot be expressed in terms of the limits in Vulkan 1.0.
- A limit on the maximum size of a single memory allocation. Some platforms have kernel interfaces that limit the maximum size of an allocation.

### Promotion to Vulkan 1.1

All functionality in this extension is included in core Vulkan 1.1, with the KHR suffix omitted. The original type, enum and command names are still available as aliases of the core functionality.

### New Commands

- vkGetDescriptorSetLayoutSupportKHR

### New Structures

- VkDescriptorSetLayoutSupportKHR
- Extending VkPhysicalDeviceProperties2:
  - VkPhysicalDeviceMaintenance3PropertiesKHR

### New Enum Constants

- `VK_KHR_MAINTENANCE3_EXTENSION_NAME`
- `VK_KHR_MAINTENANCE3_SPEC_VERSION`
- `VK_KHR_MAINTENANCE_3_EXTENSION_NAME`
- `VK_KHR_MAINTENANCE_3_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_DESCRIPTOR_SET_LAYOUT_SUPPORT_KHR`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MAINTENANCE_3_PROPERTIES_KHR`

### Version History

- Revision 1, 2017-08-22

## VK_KHR_multiview

### Name String

`VK_KHR_multiview`

### Extension Type

Device extension

### Registered Extension Number

54

### Revision

1

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### SPIR-V Dependencies

- SPV_KHR_multiview

### Deprecation State

- *Promoted* to Vulkan 1.1

### Contact

- Jeff Bolz [GitHub]jeffbolznv

### Other Extension Metadata

### Last Modified Date

2016-10-28

### IP Status

No known IP claims.

### Interactions and External Dependencies

- This extension provides API support for `GL_EXT_multiview`

### Contributors

- Jeff Bolz, NVIDIA

### Description

This extension has the same goal as the OpenGL ES `GL_OVR_multiview` extension. Multiview is a rendering technique originally designed for VR where it is more efficient to record a single set of commands to be executed with slightly different behavior for each “view”.

It includes a concise way to declare a render pass with multiple views, and gives implementations freedom to render the views in the most efficient way possible. This is done with a multiview configuration specified during render pass creation with the VkRenderPassMultiviewCreateInfo passed into VkRenderPassCreateInfo::`pNext`.

This extension enables the use of the `SPV_KHR_multiview` shader extension, which adds a new `ViewIndex` built-in type that allows shaders to control what to do for each view. If using GLSL there is also the `GL_EXT_multiview` extension that introduces a `highp int gl_ViewIndex;` built-in variable for vertex, tessellation, geometry, and fragment shaders.

### Promotion to Vulkan 1.1

All functionality in this extension is included in core Vulkan 1.1, with the KHR suffix omitted. The original type, enum and command names are still available as aliases of the core functionality.

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceMultiviewFeaturesKHR
- Extending VkPhysicalDeviceProperties2:
  - VkPhysicalDeviceMultiviewPropertiesKHR
- Extending VkRenderPassCreateInfo:
  - VkRenderPassMultiviewCreateInfoKHR

### New Enum Constants

- `VK_KHR_MULTIVIEW_EXTENSION_NAME`
- `VK_KHR_MULTIVIEW_SPEC_VERSION`
- Extending VkDependencyFlagBits:
  - `VK_DEPENDENCY_VIEW_LOCAL_BIT_KHR`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MULTIVIEW_FEATURES_KHR`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MULTIVIEW_PROPERTIES_KHR`
  - `VK_STRUCTURE_TYPE_RENDER_PASS_MULTIVIEW_CREATE_INFO_KHR`

### New Built-In Variables

- `ViewIndex`

### New SPIR-V Capabilities

- `MultiView`

### Version History

- Revision 1, 2016-10-28 (Jeff Bolz)
  - Internal revisions

## VK_KHR_relaxed_block_layout

### Name String

`VK_KHR_relaxed_block_layout`

### Extension Type

Device extension

### Registered Extension Number

145

### Revision

1

### Ratification Status

Ratified

### Extension and Version Dependencies

None

### Deprecation State

- *Promoted* to Vulkan 1.1

### Contact

- John Kessenich [GitHub]johnkslang

### Other Extension Metadata

### Last Modified Date

2017-03-26

### IP Status

No known IP claims.

### Contributors

- John Kessenich, Google

### Description

The `VK_KHR_relaxed_block_layout` extension allows implementations to indicate they can support more variation in block `Offset` decorations. For example, placing a vector of three floats at an offset of 16×N + 4.

See Offset and Stride Assignment for details.

### Promotion to Vulkan 1.1

All functionality in this extension is included in core Vulkan 1.1, with the KHR suffix omitted. The original type, enum and command names are still available as aliases of the core functionality.

### New Enum Constants

- `VK_KHR_RELAXED_BLOCK_LAYOUT_EXTENSION_NAME`
- `VK_KHR_RELAXED_BLOCK_LAYOUT_SPEC_VERSION`

### Version History

- Revision 1, 2017-03-26 (JohnK)

## VK_KHR_sampler_ycbcr_conversion

### Name String

`VK_KHR_sampler_ycbcr_conversion`

### Extension Type

Device extension

### Registered Extension Number

157

### Revision

14

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_maintenance1
and
VK_KHR_bind_memory2
and
VK_KHR_get_memory_requirements2
and
VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### API Interactions

- Interacts with VK_EXT_debug_report

### Deprecation State

- *Promoted* to Vulkan 1.1

### Contact

- Andrew Garrard [GitHub]fluppeteer

### Other Extension Metadata

### Last Modified Date

2017-08-11

### IP Status

No known IP claims.

### Contributors

- Andrew Garrard, Samsung Electronics
- Tobias Hector, Imagination Technologies

- James Jones, NVIDIA
- Daniel Koch, NVIDIA
- Daniel Rakos, AMD
- Romain Guy, Google
- Jesse Hall, Google
- Tom Cooksey, ARM Ltd
- Jeff Leger, Qualcomm Technologies, Inc
- Jan-Harald Fredriksen, ARM Ltd
- Jan Outters, Samsung Electronics
- Alon Or-bach, Samsung Electronics
- Michael Worcester, Imagination Technologies
- Jeff Bolz, NVIDIA
- Tony Zlatinski, NVIDIA
- Matthew Netsch, Qualcomm Technologies, Inc

### Description

The use of $Y'C_BC_R$ sampler conversion is an area in 3D graphics not used by most Vulkan developers. It is mainly used for processing inputs from video decoders and cameras. The use of the extension assumes basic knowledge of $Y'C_BC_R$ concepts.

This extension provides the ability to perform specified color space conversions during texture sampling operations for the $Y'C_BC_R$ color space natively. It also adds a selection of multi-planar formats, image aspect plane, and the ability to bind memory to the planes of an image collectively or separately.

### Promotion to Vulkan 1.1

All functionality in this extension is included in core Vulkan 1.1, with the KHR suffix omitted. However, if Vulkan 1.1 is supported and this extension is not, the `samplerYcbcrConversion` capability is optional. The original type, enum and command names are still available as aliases of the core functionality.

### New Object Types

- VkSamplerYcbcrConversionKHR

### New Commands

- vkCreateSamplerYcbcrConversionKHR
- vkDestroySamplerYcbcrConversionKHR

### New Structures

- VkSamplerYcbcrConversionCreateInfoKHR
- Extending VkBindImageMemoryInfo:
  - VkBindImagePlaneMemoryInfoKHR
- Extending VkImageFormatProperties2:
  - VkSamplerYcbcrConversionImageFormatPropertiesKHR
- Extending VkImageMemoryRequirementsInfo2:
  - VkImagePlaneMemoryRequirementsInfoKHR
- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceSamplerYcbcrConversionFeaturesKHR
- Extending VkSamplerCreateInfo, VkImageViewCreateInfo:
  - VkSamplerYcbcrConversionInfoKHR

### New Enums

- VkChromaLocationKHR
- VkSamplerYcbcrModelConversionKHR
- VkSamplerYcbcrRangeKHR

### New Enum Constants

- `VK_KHR_SAMPLER_YCBCR_CONVERSION_EXTENSION_NAME`
- `VK_KHR_SAMPLER_YCBCR_CONVERSION_SPEC_VERSION`
- Extending VkChromaLocation:
  - `VK_CHROMA_LOCATION_COSITED_EVEN_KHR`
  - `VK_CHROMA_LOCATION_MIDPOINT_KHR`
- Extending VkFormat:
  - `VK_FORMAT_B10X6G10X6R10X6G10X6_422_UNORM_4PACK16_KHR`
  - `VK_FORMAT_B12X4G12X4R12X4G12X4_422_UNORM_4PACK16_KHR`
  - `VK_FORMAT_B16G16R16G16_422_UNORM_KHR`
  - `VK_FORMAT_B8G8R8G8_422_UNORM_KHR`
  - `VK_FORMAT_G10X6B10X6G10X6R10X6_422_UNORM_4PACK16_KHR`
  - `VK_FORMAT_G10X6_B10X6R10X6_2PLANE_420_UNORM_3PACK16_KHR`
  - `VK_FORMAT_G10X6_B10X6R10X6_2PLANE_422_UNORM_3PACK16_KHR`
  - `VK_FORMAT_G10X6_B10X6_R10X6_3PLANE_420_UNORM_3PACK16_KHR`
  - `VK_FORMAT_G10X6_B10X6_R10X6_3PLANE_422_UNORM_3PACK16_KHR`
  - `VK_FORMAT_G10X6_B10X6_R10X6_3PLANE_444_UNORM_3PACK16_KHR`

    - VK_FORMAT_G12X4B12X4G12X4R12X4_422_UNORM_4PACK16_KHR
    - VK_FORMAT_G12X4_B12X4R12X4_2PLANE_420_UNORM_3PACK16_KHR
    - VK_FORMAT_G12X4_B12X4R12X4_2PLANE_422_UNORM_3PACK16_KHR
    - VK_FORMAT_G12X4_B12X4_R12X4_3PLANE_420_UNORM_3PACK16_KHR
    - VK_FORMAT_G12X4_B12X4_R12X4_3PLANE_422_UNORM_3PACK16_KHR
    - VK_FORMAT_G12X4_B12X4_R12X4_3PLANE_444_UNORM_3PACK16_KHR
    - VK_FORMAT_G16B16G16R16_422_UNORM_KHR
    - VK_FORMAT_G16_B16R16_2PLANE_420_UNORM_KHR
    - VK_FORMAT_G16_B16R16_2PLANE_422_UNORM_KHR
    - VK_FORMAT_G16_B16_R16_3PLANE_420_UNORM_KHR
    - VK_FORMAT_G16_B16_R16_3PLANE_422_UNORM_KHR
    - VK_FORMAT_G16_B16_R16_3PLANE_444_UNORM_KHR
    - VK_FORMAT_G8B8G8R8_422_UNORM_KHR
    - VK_FORMAT_G8_B8R8_2PLANE_420_UNORM_KHR
    - VK_FORMAT_G8_B8R8_2PLANE_422_UNORM_KHR
    - VK_FORMAT_G8_B8_R8_3PLANE_420_UNORM_KHR
    - VK_FORMAT_G8_B8_R8_3PLANE_422_UNORM_KHR
    - VK_FORMAT_G8_B8_R8_3PLANE_444_UNORM_KHR
    - VK_FORMAT_R10X6G10X6B10X6A10X6_UNORM_4PACK16_KHR
    - VK_FORMAT_R10X6G10X6_UNORM_2PACK16_KHR
    - VK_FORMAT_R10X6_UNORM_PACK16_KHR
    - VK_FORMAT_R12X4G12X4B12X4A12X4_UNORM_4PACK16_KHR
    - VK_FORMAT_R12X4G12X4_UNORM_2PACK16_KHR
    - VK_FORMAT_R12X4_UNORM_PACK16_KHR
- Extending VkFormatFeatureFlagBits:
    - VK_FORMAT_FEATURE_COSITED_CHROMA_SAMPLES_BIT_KHR
    - VK_FORMAT_FEATURE_DISJOINT_BIT_KHR
    - VK_FORMAT_FEATURE_MIDPOINT_CHROMA_SAMPLES_BIT_KHR
    - VK_FORMAT_FEATURE_SAMPLED_IMAGE_YCBCR_CONVERSION_CHROMA_RECONSTRUCTION_EXPLICIT_BIT_KHR
    - VK_FORMAT_FEATURE_SAMPLED_IMAGE_YCBCR_CONVERSION_CHROMA_RECONSTRUCTION_EXPLICIT_FORCEABLE_BIT_KHR
    - VK_FORMAT_FEATURE_SAMPLED_IMAGE_YCBCR_CONVERSION_LINEAR_FILTER_BIT_KHR
    - VK_FORMAT_FEATURE_SAMPLED_IMAGE_YCBCR_CONVERSION_SEPARATE_RECONSTRUCTION_FILTER_BIT_KHR
- Extending VkImageAspectFlagBits:
    - VK_IMAGE_ASPECT_PLANE_0_BIT_KHR

    - VK_IMAGE_ASPECT_PLANE_1_BIT_KHR
    - VK_IMAGE_ASPECT_PLANE_2_BIT_KHR
- Extending VkImageCreateFlagBits:
    - VK_IMAGE_CREATE_DISJOINT_BIT_KHR
- Extending VkObjectType:
    - VK_OBJECT_TYPE_SAMPLER_YCBCR_CONVERSION_KHR
- Extending VkSamplerYcbcrModelConversion:
    - VK_SAMPLER_YCBCR_MODEL_CONVERSION_RGB_IDENTITY_KHR
    - VK_SAMPLER_YCBCR_MODEL_CONVERSION_YCBCR_2020_KHR
    - VK_SAMPLER_YCBCR_MODEL_CONVERSION_YCBCR_601_KHR
    - VK_SAMPLER_YCBCR_MODEL_CONVERSION_YCBCR_709_KHR
    - VK_SAMPLER_YCBCR_MODEL_CONVERSION_YCBCR_IDENTITY_KHR
- Extending VkSamplerYcbcrRange:
    - VK_SAMPLER_YCBCR_RANGE_ITU_FULL_KHR
    - VK_SAMPLER_YCBCR_RANGE_ITU_NARROW_KHR
- Extending VkStructureType:
    - VK_STRUCTURE_TYPE_BIND_IMAGE_PLANE_MEMORY_INFO_KHR
    - VK_STRUCTURE_TYPE_IMAGE_PLANE_MEMORY_REQUIREMENTS_INFO_KHR
    - VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SAMPLER_YCBCR_CONVERSION_FEATURES_KHR
    - VK_STRUCTURE_TYPE_SAMPLER_YCBCR_CONVERSION_CREATE_INFO_KHR
    - VK_STRUCTURE_TYPE_SAMPLER_YCBCR_CONVERSION_IMAGE_FORMAT_PROPERTIES_KHR
    - VK_STRUCTURE_TYPE_SAMPLER_YCBCR_CONVERSION_INFO_KHR

If VK_EXT_debug_report is supported:

- Extending VkDebugReportObjectTypeEXT:
    - VK_DEBUG_REPORT_OBJECT_TYPE_SAMPLER_YCBCR_CONVERSION_EXT
    - VK_DEBUG_REPORT_OBJECT_TYPE_SAMPLER_YCBCR_CONVERSION_KHR_EXT

**Version History**

- Revision 1, 2017-01-24 (Andrew Garrard)
    - Initial draft
- Revision 2, 2017-01-25 (Andrew Garrard)
    - After initial feedback
- Revision 3, 2017-01-27 (Andrew Garrard)
    - Higher bit depth formats, renaming, swizzle

- Revision 4, 2017-02-22 (Andrew Garrard)
  - Added query function, formats as RGB, clarifications
- Revision 5, 2017-04-?? (Andrew Garrard)
  - Simplified query and removed output conversions
- Revision 6, 2017-04-24 (Andrew Garrard)
  - Tidying, incorporated new image query, restored transfer functions
- Revision 7, 2017-04-25 (Andrew Garrard)
  - Added cosited option/midpoint requirement for formats, “bypassConversion”
- Revision 8, 2017-04-25 (Andrew Garrard)
  - Simplified further
- Revision 9, 2017-04-27 (Andrew Garrard)
  - Disjoint no more
- Revision 10, 2017-04-28 (Andrew Garrard)
  - Restored disjoint
- Revision 11, 2017-04-29 (Andrew Garrard)
  - Now Ycbcr conversion, and KHR
- Revision 12, 2017-06-06 (Andrew Garrard)
  - Added conversion to image view creation
- Revision 13, 2017-07-13 (Andrew Garrard)
  - Allowed cosited-only chroma samples for formats
- Revision 14, 2017-08-11 (Andrew Garrard)
  - Reflected quantization changes in BT.2100-1

## VK_KHR_shader_draw_parameters

### Name String

`VK_KHR_shader_draw_parameters`

### Extension Type

Device extension

### Registered Extension Number

64

### Revision

1

### Ratification Status

Ratified

### Extension and Version Dependencies

None

### SPIR-V Dependencies

- SPV_KHR_shader_draw_parameters

### Deprecation State

- *Promoted* to Vulkan 1.1

### Contact

- Daniel Koch [GitHub]dgkoch

### Other Extension Metadata

### Last Modified Date

2017-09-05

### IP Status

No known IP claims.

### Interactions and External Dependencies

- This extension provides API support for `GL_ARB_shader_draw_parameters`

### Contributors

- Daniel Koch, NVIDIA Corporation
- Jeff Bolz, NVIDIA
- Daniel Rakos, AMD
- Jan-Harald Fredriksen, ARM
- John Kessenich, Google
- Stuart Smith, IMG

### Description

This extension adds support for the following SPIR-V extension in Vulkan:

- `SPV_KHR_shader_draw_parameters`

The extension provides access to three additional built-in shader variables in Vulkan:

- `BaseInstance`, containing the `firstInstance` parameter passed to drawing commands,
- `BaseVertex`, containing the `firstVertex` or `vertexOffset` parameter passed to drawing commands, and
- `DrawIndex`, containing the index of the draw call currently being processed from an indirect drawing call.

When using GLSL source-based shader languages, the following variables from `GL_ARB_shader_draw_parameters` can map to these SPIR-V built-in decorations:

- `in int gl_BaseInstanceARB;` → `BaseInstance`,
- `in int gl_BaseVertexARB;` → `BaseVertex`, and
- `in int gl_DrawIDARB;` → `DrawIndex`.

#### Promotion to Vulkan 1.1

All functionality in this extension is included in core Vulkan 1.1. However, the `shaderDrawParameters` feature bit was added to distinguish whether it is actually available or not.

#### New Enum Constants

- `VK_KHR_SHADER_DRAW_PARAMETERS_EXTENSION_NAME`
- `VK_KHR_SHADER_DRAW_PARAMETERS_SPEC_VERSION`

#### New Built-In Variables

- `BaseInstance`
- `BaseVertex`
- `DrawIndex`

#### New SPIR-V Capabilities

- `DrawParameters`

#### Issues

1) Is this the same functionality as `GL_ARB_shader_draw_parameters`?

**RESOLVED**: It is actually a superset, as it also adds in support for arrayed drawing commands.

In GL for `GL_ARB_shader_draw_parameters`, `gl_BaseVertexARB` holds the integer value passed to the parameter to the command that resulted in the current shader invocation. In the case where the command has no `baseVertex` parameter, the value of `gl_BaseVertexARB` is zero. This means that `gl_BaseVertexARB` = `baseVertex` (for `glDrawElements` commands with `baseVertex`) or 0. In particular there are no `glDrawArrays` commands that take a `baseVertex` parameter.

Now in Vulkan, we have `BaseVertex` = `vertexOffset` (for indexed drawing commands) or `firstVertex` (for arrayed drawing commands), and so Vulkan's version is really a superset of GL functionality.

#### Version History

- Revision 1, 2016-10-05 (Daniel Koch)
  - Internal revisions

### VK_KHR_storage_buffer_storage_class

#### Name String

`VK_KHR_storage_buffer_storage_class`

**Extension Type**

Device extension

**Registered Extension Number**

132

**Revision**

1

**Ratification Status**

Ratified

**Extension and Version Dependencies**

None

**SPIR-V Dependencies**

- SPV_KHR_storage_buffer_storage_class

**Deprecation State**

- *Promoted* to Vulkan 1.1

**Contact**

- Alexander Galazin [GitHub]alegal-arm

**Other Extension Metadata**

**Last Modified Date**

2017-09-05

**IP Status**

No known IP claims.

**Contributors**

- Alexander Galazin, ARM
- David Neto, Google

**Description**

This extension adds support for the following SPIR-V extension in Vulkan:

- `SPV_KHR_storage_buffer_storage_class`

This extension provides a new SPIR-V `StorageBuffer` storage class. A `Block`-decorated object in this class is equivalent to a `BufferBlock`-decorated object in the `Uniform` storage class.

**Promotion to Vulkan 1.1**

All functionality in this extension is included in core Vulkan 1.1.

### New Enum Constants

- `VK_KHR_STORAGE_BUFFER_STORAGE_CLASS_EXTENSION_NAME`
- `VK_KHR_STORAGE_BUFFER_STORAGE_CLASS_SPEC_VERSION`

### Version History

- Revision 1, 2017-03-23 (Alexander Galazin)
  - Initial draft

## VK_KHR_variable_pointers

### Name String

`VK_KHR_variable_pointers`

### Extension Type

Device extension

### Registered Extension Number

121

### Revision

1

### Ratification Status

Ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
and
VK_KHR_storage_buffer_storage_class
or
Vulkan Version 1.1

### SPIR-V Dependencies

- SPV_KHR_variable_pointers

### Deprecation State

- *Promoted* to Vulkan 1.1

### Contact

- Jesse Hall [GitHub]critsec

### Other Extension Metadata

### Last Modified Date

2017-09-05

### IP Status

No known IP claims.

### Contributors

- John Kessenich, Google
- Neil Henning, Codeplay
- David Neto, Google
- Daniel Koch, Nvidia
- Graeme Leese, Broadcom
- Weifeng Zhang, Qualcomm
- Stephen Clarke, Imagination Technologies
- Faith Ekstrand, Intel
- Jesse Hall, Google

### Description

The `VK_KHR_variable_pointers` extension allows implementations to indicate their level of support for the `SPV_KHR_variable_pointers` SPIR-V extension. The SPIR-V extension allows shader modules to use invocation-private pointers into uniform and/or storage buffers, where the pointer values can be dynamic and non-uniform.

The `SPV_KHR_variable_pointers` extension introduces two capabilities. The first, `VariablePointersStorageBuffer`, **must** be supported by all implementations of this extension. The second, `VariablePointers`, is optional.

### Promotion to Vulkan 1.1

All functionality in this extension is included in core Vulkan 1.1, with the KHR suffix omitted, however support for the `variablePointersStorageBuffer` feature is made optional. The original type, enum and command names are still available as aliases of the core functionality.

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceVariablePointerFeaturesKHR
  - VkPhysicalDeviceVariablePointersFeaturesKHR

### New Enum Constants

- `VK_KHR_VARIABLE_POINTERS_EXTENSION_NAME`
- `VK_KHR_VARIABLE_POINTERS_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_VARIABLE_POINTERS_FEATURES_KHR`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_VARIABLE_POINTER_FEATURES_KHR`

### New SPIR-V Capabilities

- `VariablePointers`
- `VariablePointersStorageBuffer`

### Issues

1) Do we need an optional property for the SPIR-V `VariablePointersStorageBuffer` capability or should it be mandatory when this extension is advertised?

**RESOLVED**: Add it as a distinct feature, but make support mandatory. Adding it as a feature makes the extension easier to include in a future core API version. In the extension, the feature is mandatory, so that presence of the extension guarantees some functionality. When included in a core API version, the feature would be optional.

2) Can support for these capabilities vary between shader stages?

**RESOLVED**: No, if the capability is supported in any stage it must be supported in all stages.

3) Should the capabilities be features or limits?

**RESOLVED**: Features, primarily for consistency with other similar extensions.

### Version History

- Revision 1, 2017-03-14 (Jesse Hall and John Kessenich)
  - Internal revisions

## VK_EXT_buffer_device_address

### Name String

`VK_EXT_buffer_device_address`

### Extension Type

Device extension

### Registered Extension Number

245

### Revision

2

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### SPIR-V Dependencies

- SPV_EXT_physical_storage_buffer

### Deprecation State

- *Deprecated* by VK_KHR_buffer_device_address extension
  - Which in turn was *promoted* to Vulkan 1.2

### Contact

- Jeff Bolz [GitHub]jeffbolznv

### Other Extension Metadata

### Last Modified Date

2019-01-06

### IP Status

No known IP claims.

### Interactions and External Dependencies

- This extension provides API support for `GLSL_EXT_buffer_reference` and `GLSL_EXT_buffer_reference_uvec2`

### Contributors

- Jeff Bolz, NVIDIA
- Neil Henning, AMD
- Tobias Hector, AMD
- Faith Ekstrand, Intel
- Baldur Karlsson, Valve

### Description

This extension allows the application to query a 64-bit buffer device address value for a buffer, which can be used to access the buffer memory via the `PhysicalStorageBufferEXT` storage class in the `GL_EXT_buffer_reference` GLSL extension and `SPV_EXT_physical_storage_buffer` SPIR-V extension.

It also allows buffer device addresses to be provided by a trace replay tool, so that it matches the address used when the trace was captured.

### New Commands

- vkGetBufferDeviceAddressEXT

### New Structures

- VkBufferDeviceAddressInfoEXT
- Extending VkBufferCreateInfo:
  - VkBufferDeviceAddressCreateInfoEXT

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceBufferAddressFeaturesEXT
  - VkPhysicalDeviceBufferDeviceAddressFeaturesEXT

### New Enum Constants

- `VK_EXT_BUFFER_DEVICE_ADDRESS_EXTENSION_NAME`
- `VK_EXT_BUFFER_DEVICE_ADDRESS_SPEC_VERSION`
- Extending VkBufferCreateFlagBits:
  - `VK_BUFFER_CREATE_DEVICE_ADDRESS_CAPTURE_REPLAY_BIT_EXT`
- Extending VkBufferUsageFlagBits:
  - `VK_BUFFER_USAGE_SHADER_DEVICE_ADDRESS_BIT_EXT`
- Extending VkResult:
  - `VK_ERROR_INVALID_DEVICE_ADDRESS_EXT`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_BUFFER_DEVICE_ADDRESS_CREATE_INFO_EXT`
  - `VK_STRUCTURE_TYPE_BUFFER_DEVICE_ADDRESS_INFO_EXT`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_BUFFER_ADDRESS_FEATURES_EXT`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_BUFFER_DEVICE_ADDRESS_FEATURES_EXT`

### New SPIR-V Capabilities

- `PhysicalStorageBufferAddressesEXT`

### Issues

1) Where is VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_BUFFER_ADDRESS_FEATURES_EXT and VkPhysicalDeviceBufferAddressFeaturesEXT?

**RESOLVED**: They were renamed as `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_BUFFER_DEVICE_ADDRESS_FEATURES_EXT` and VkPhysicalDeviceBufferDeviceAddressFeaturesEXT accordingly for consistency. Even though, the old names can still be found in the generated header files for compatibility.

### Version History

- Revision 1, 2018-11-01 (Jeff Bolz)
  - Internal revisions
- Revision 2, 2019-01-06 (Jon Leech)
  - Minor updates to appendix for publication

## VK_EXT_calibrated_timestamps

### Name String

`VK_EXT_calibrated_timestamps`

### Extension Type

Device extension

### Registered Extension Number

185

### Revision

2

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### Deprecation State

- *Promoted* to VK_KHR_calibrated_timestamps extension

### Contact

- Daniel Rakos [GitHub]drakos-amd

### Extension Proposal

VK_EXT_calibrated_timestamps

### Other Extension Metadata

### Last Modified Date

2018-10-04

### IP Status

No known IP claims.

### Contributors

- Matthaeus G. Chajdas, AMD
- Alan Harrison, AMD
- Derrick Owens, AMD
- Daniel Rakos, AMD
- Faith Ekstrand, Intel
- Keith Packard, Valve

### Description

This extension provides an interface to query calibrated timestamps obtained quasi simultaneously from two time domains.

### Promotion to `VK_KHR_calibrated_timestamps`

All functionality in this extension is included in `VK_KHR_calibrated_timestamps`, with the suffix changed to KHR. The original enum names are still available as aliases of the KHR functionality.

### New Commands

- vkGetCalibratedTimestampsEXT
- vkGetPhysicalDeviceCalibrateableTimeDomainsEXT

### New Structures

- VkCalibratedTimestampInfoEXT

### New Enums

- VkTimeDomainEXT

### New Enum Constants

- `VK_EXT_CALIBRATED_TIMESTAMPS_EXTENSION_NAME`
- `VK_EXT_CALIBRATED_TIMESTAMPS_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_CALIBRATED_TIMESTAMP_INFO_EXT`
- Extending VkTimeDomainKHR:
  - `VK_TIME_DOMAIN_CLOCK_MONOTONIC_EXT`
  - `VK_TIME_DOMAIN_CLOCK_MONOTONIC_RAW_EXT`
  - `VK_TIME_DOMAIN_DEVICE_EXT`
  - `VK_TIME_DOMAIN_QUERY_PERFORMANCE_COUNTER_EXT`

### Version History

- Revision 2, 2021-03-16 (Lionel Landwerlin)
  - Specify requirement on device timestamps
- Revision 1, 2018-10-04 (Daniel Rakos)
  - Internal revisions.

## VK_EXT_debug_marker

**Name String**

`VK_EXT_debug_marker`

**Extension Type**

Device extension

**Registered Extension Number**

23

**Revision**

4

**Ratification Status**

Not ratified

**Extension and Version Dependencies**

VK_EXT_debug_report

**Deprecation State**

- *Promoted* to VK_EXT_debug_utils extension

**Special Use**

- Debugging tools

**Contact**

- Baldur Karlsson [GitHub]baldurk

**Other Extension Metadata**

**Last Modified Date**

2017-01-31

**IP Status**

No known IP claims.

**Contributors**

- Baldur Karlsson
- Dan Ginsburg, Valve
- Jon Ashburn, LunarG
- Kyle Spagnoli, NVIDIA

**Description**

The `VK_EXT_debug_marker` extension is a device extension. It introduces concepts of object naming and tagging, for better tracking of Vulkan objects, as well as additional commands for recording annotations of named sections of a workload to aid organization and offline analysis in external tools.

### New Commands

- vkCmdDebugMarkerBeginEXT
- vkCmdDebugMarkerEndEXT
- vkCmdDebugMarkerInsertEXT
- vkDebugMarkerSetObjectNameEXT
- vkDebugMarkerSetObjectTagEXT

### New Structures

- VkDebugMarkerMarkerInfoEXT
- VkDebugMarkerObjectNameInfoEXT
- VkDebugMarkerObjectTagInfoEXT

### New Enums

- VkDebugReportObjectTypeEXT

### New Enum Constants

- `VK_EXT_DEBUG_MARKER_EXTENSION_NAME`
- `VK_EXT_DEBUG_MARKER_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_DEBUG_MARKER_MARKER_INFO_EXT`
  - `VK_STRUCTURE_TYPE_DEBUG_MARKER_OBJECT_NAME_INFO_EXT`
  - `VK_STRUCTURE_TYPE_DEBUG_MARKER_OBJECT_TAG_INFO_EXT`

### Examples

### Example 1

Associate a name with an image, for easier debugging in external tools or with validation layers that can print a friendly name when referring to objects in error messages.

```
    extern VkDevice device;
    extern VkImage image;

    // Must call extension functions through a function pointer:
    PFN_vkDebugMarkerSetObjectNameEXT pfnDebugMarkerSetObjectNameEXT =
(PFN_vkDebugMarkerSetObjectNameEXT)vkGetDeviceProcAddr(device,
"vkDebugMarkerSetObjectNameEXT");

    // Set a name on the image
    const VkDebugMarkerObjectNameInfoEXT imageNameInfo =
    {
        .sType = VK_STRUCTURE_TYPE_DEBUG_MARKER_OBJECT_NAME_INFO_EXT,
```

```
        .pNext = NULL,
        .objectType = VK_DEBUG_REPORT_OBJECT_TYPE_IMAGE_EXT,
        .object = (uint64_t)image,
        .pObjectName = "Brick Diffuse Texture",
    };

    pfnDebugMarkerSetObjectNameEXT(device, &imageNameInfo);

    // A subsequent error might print:
    //   Image 'Brick Diffuse Texture' (0xc0dec0dedeadbeef) is used in a
    //   command buffer with no memory bound to it.
```

**Example 2**

Annotating regions of a workload with naming information so that offline analysis tools can display a more usable visualization of the commands submitted.

```
    extern VkDevice device;
    extern VkCommandBuffer commandBuffer;

    // Must call extension functions through a function pointer:
    PFN_vkCmdDebugMarkerBeginEXT pfnCmdDebugMarkerBeginEXT =
(PFN_vkCmdDebugMarkerBeginEXT)vkGetDeviceProcAddr(device, "vkCmdDebugMarkerBeginEXT");
    PFN_vkCmdDebugMarkerEndEXT pfnCmdDebugMarkerEndEXT = (PFN_vkCmdDebugMarkerEndEXT
)vkGetDeviceProcAddr(device, "vkCmdDebugMarkerEndEXT");
    PFN_vkCmdDebugMarkerInsertEXT pfnCmdDebugMarkerInsertEXT =
(PFN_vkCmdDebugMarkerInsertEXT)vkGetDeviceProcAddr(device,
"vkCmdDebugMarkerInsertEXT");

    // Describe the area being rendered
    const VkDebugMarkerMarkerInfoEXT houseMarker =
    {
        .sType = VK_STRUCTURE_TYPE_DEBUG_MARKER_MARKER_INFO_EXT,
        .pNext = NULL,
        .pMarkerName = "Brick House",
        .color = { 1.0f, 0.0f, 0.0f, 1.0f },
    };

    // Start an annotated group of calls under the 'Brick House' name
    pfnCmdDebugMarkerBeginEXT(commandBuffer, &houseMarker);
    {
        // A mutable structure for each part being rendered
        VkDebugMarkerMarkerInfoEXT housePartMarker =
        {
            .sType = VK_STRUCTURE_TYPE_DEBUG_MARKER_MARKER_INFO_EXT,
            .pNext = NULL,
            .pMarkerName = NULL,
            .color = { 0.0f, 0.0f, 0.0f, 0.0f },
        };
```

```
        // Set the name and insert the marker
        housePartMarker.pMarkerName = "Walls";
        pfnCmdDebugMarkerInsertEXT(commandBuffer, &housePartMarker);

        // Insert the drawcall for the walls
        vkCmdDrawIndexed(commandBuffer, 1000, 1, 0, 0, 0);

        // Insert a recursive region for two sets of windows
        housePartMarker.pMarkerName = "Windows";
        pfnCmdDebugMarkerBeginEXT(commandBuffer, &housePartMarker);
        {
            vkCmdDrawIndexed(commandBuffer, 75, 6, 1000, 0, 0);
            vkCmdDrawIndexed(commandBuffer, 100, 2, 1450, 0, 0);
        }
        pfnCmdDebugMarkerEndEXT(commandBuffer);

        housePartMarker.pMarkerName = "Front Door";
        pfnCmdDebugMarkerInsertEXT(commandBuffer, &housePartMarker);

        vkCmdDrawIndexed(commandBuffer, 350, 1, 1650, 0, 0);

        housePartMarker.pMarkerName = "Roof";
        pfnCmdDebugMarkerInsertEXT(commandBuffer, &housePartMarker);

        vkCmdDrawIndexed(commandBuffer, 500, 1, 2000, 0, 0);
    }
    // End the house annotation started above
    pfnCmdDebugMarkerEndEXT(commandBuffer);
```

**Issues**

1) Should the tag or name for an object be specified using the `pNext` parameter in the object's `Vk*CreateInfo` structure?

**RESOLVED**: No. While this fits with other Vulkan patterns and would allow more type safety and future proofing against future objects, it has notable downsides. In particular passing the name at `Vk*CreateInfo` time does not allow renaming, prevents late binding of naming information, and does not allow naming of implicitly created objects such as queues and swapchain images.

2) Should the command annotation functions vkCmdDebugMarkerBeginEXT and vkCmdDebugMarkerEndEXT support the ability to specify a color?

**RESOLVED**: Yes. The functions have been expanded to take an optional color which can be used at will by implementations consuming the command buffer annotations in their visualization.

3) Should the functions added in this extension accept an extensible structure as their parameter for a more flexible API, as opposed to direct function parameters? If so, which functions?

**RESOLVED**: Yes. All functions have been modified to take a structure type with extensible `pNext` pointer, to allow future extensions to add additional annotation information in the same

commands.

### Version History

- Revision 1, 2016-02-24 (Baldur Karlsson)
  - Initial draft, based on LunarG marker spec
- Revision 2, 2016-02-26 (Baldur Karlsson)
  - Renamed Dbg to DebugMarker in function names
  - Allow markers in secondary command buffers under certain circumstances
  - Minor language tweaks and edits
- Revision 3, 2016-04-23 (Baldur Karlsson)
  - Reorganize spec layout to closer match desired organization
  - Added optional color to markers (both regions and inserted labels)
  - Changed functions to take extensible structs instead of direct function parameters
- Revision 4, 2017-01-31 (Baldur Karlsson)
  - Added explicit dependency on VK_EXT_debug_report
  - Moved definition of VkDebugReportObjectTypeEXT to debug report chapter.
  - Fixed typo in dates in revision history

## VK_EXT_debug_report

### Name String

`VK_EXT_debug_report`

### Extension Type

Instance extension

### Registered Extension Number

12

### Revision

10

### Ratification Status

Not ratified

### Extension and Version Dependencies

None

### API Interactions

- Interacts with VK_VERSION_1_1

### Deprecation State

- *Deprecated* by VK_EXT_debug_utils extension

**Special Use**

- Debugging tools

**Contact**

- Courtney Goeltzenleuchter [GitHub]courtney-g

**Other Extension Metadata**

**Last Modified Date**

2020-12-14

**IP Status**

No known IP claims.

**Contributors**

- Courtney Goeltzenleuchter, LunarG
- Dan Ginsburg, Valve
- Jon Ashburn, LunarG
- Mark Lobodzinski, LunarG

**Description**

Due to the nature of the Vulkan interface, there is very little error information available to the developer and application. By enabling optional validation layers and using the `VK_EXT_debug_report` extension, developers **can** obtain much more detailed feedback on the application's use of Vulkan. This extension defines a way for layers and the implementation to call back to the application for events of interest to the application.

**New Object Types**

- VkDebugReportCallbackEXT

**New Commands**

- vkCreateDebugReportCallbackEXT
- vkDebugReportMessageEXT
- vkDestroyDebugReportCallbackEXT

**New Structures**

- Extending VkInstanceCreateInfo:
  - VkDebugReportCallbackCreateInfoEXT

**New Function Pointers**

- PFN_vkDebugReportCallbackEXT

### New Enums

- VkDebugReportFlagBitsEXT
- VkDebugReportObjectTypeEXT

### New Bitmasks

- VkDebugReportFlagsEXT

### New Enum Constants

- `VK_EXT_DEBUG_REPORT_EXTENSION_NAME`
- `VK_EXT_DEBUG_REPORT_SPEC_VERSION`
- Extending VkObjectType:
  - `VK_OBJECT_TYPE_DEBUG_REPORT_CALLBACK_EXT`
- Extending VkResult:
  - `VK_ERROR_VALIDATION_FAILED_EXT`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_DEBUG_REPORT_CALLBACK_CREATE_INFO_EXT`
  - `VK_STRUCTURE_TYPE_DEBUG_REPORT_CREATE_INFO_EXT`

If Vulkan Version 1.1 is supported:

- Extending VkDebugReportObjectTypeEXT:
  - `VK_DEBUG_REPORT_OBJECT_TYPE_DESCRIPTOR_UPDATE_TEMPLATE_EXT`
  - `VK_DEBUG_REPORT_OBJECT_TYPE_SAMPLER_YCBCR_CONVERSION_EXT`

### Examples

`VK_EXT_debug_report` allows an application to register multiple callbacks with the validation layers. Some callbacks may log the information to a file, others may cause a debug break point or other application-defined behavior. An application **can** register callbacks even when no validation layers are enabled, but they will only be called for loader and, if implemented, driver events.

To capture events that occur while creating or destroying an instance an application **can** link a VkDebugReportCallbackCreateInfoEXT structure to the `pNext` element of the VkInstanceCreateInfo structure given to vkCreateInstance.

Example uses: Create three callback objects. One will log errors and warnings to the debug console using Windows `OutputDebugString`. The second will cause the debugger to break at that callback when an error happens and the third will log warnings to stdout.

```
VkResult res;
VkDebugReportCallbackEXT cb1, cb2, cb3;

VkDebugReportCallbackCreateInfoEXT callback1 = {
```

```
        .sType = VK_STRUCTURE_TYPE_DEBUG_REPORT_CALLBACK_CREATE_INFO_EXT,
        .pNext = NULL,
        .flags = VK_DEBUG_REPORT_ERROR_BIT_EXT |
                 VK_DEBUG_REPORT_WARNING_BIT_EXT,
        .pfnCallback = myOutputDebugString,
        .pUserData = NULL
    };
    res = vkCreateDebugReportCallbackEXT(instance, &callback1, &cb1);
    if (res != VK_SUCCESS)
       /* Do error handling for VK_ERROR_OUT_OF_MEMORY */

    callback.flags = VK_DEBUG_REPORT_ERROR_BIT_EXT;
    callback.pfnCallback = myDebugBreak;
    callback.pUserData = NULL;
    res = vkCreateDebugReportCallbackEXT(instance, &callback, &cb2);
    if (res != VK_SUCCESS)
       /* Do error handling for VK_ERROR_OUT_OF_MEMORY */

    VkDebugReportCallbackCreateInfoEXT callback3 = {
        .sType = VK_STRUCTURE_TYPE_DEBUG_REPORT_CALLBACK_CREATE_INFO_EXT,
        .pNext = NULL,
        .flags = VK_DEBUG_REPORT_WARNING_BIT_EXT,
        .pfnCallback = mystdOutLogger,
        .pUserData = NULL
    };
    res = vkCreateDebugReportCallbackEXT(instance, &callback3, &cb3);
    if (res != VK_SUCCESS)
       /* Do error handling for VK_ERROR_OUT_OF_MEMORY */

    ...

    /* remove callbacks when cleaning up */
    vkDestroyDebugReportCallbackEXT(instance, cb1);
    vkDestroyDebugReportCallbackEXT(instance, cb2);
    vkDestroyDebugReportCallbackEXT(instance, cb3);
```

**NOTE** In the initial release of the `VK_EXT_debug_report` extension, the token `VK_STRUCTURE_TYPE_DEBUG_REPORT_CREATE_INFO_EXT` was used. Starting in version 2 of the extension branch, `VK_STRUCTURE_TYPE_DEBUG_REPORT_CALLBACK_CREATE_INFO_EXT` is used instead for consistency with Vulkan naming rules. The older enum is still available for backwards compatibility.

**NOTE** In the initial release of the `VK_EXT_debug_report` extension, the token `VK_DEBUG_REPORT_OBJECT_TYPE_DEBUG_REPORT_EXT` was used. Starting in version 8 of the extension branch, `VK_DEBUG_REPORT_OBJECT_TYPE_DEBUG_REPORT_CALLBACK_EXT_EXT` is used instead for consistency with Vulkan naming rules. The older enum is still available for backwards compatibility.

**Issues**

1) What is the hierarchy / seriousness of the message flags? E.g. `ERROR` > `WARN` > `PERF_WARN` ...

**RESOLVED**: There is no specific hierarchy. Each bit is independent and should be checked via bitwise AND. For example:

```
    if (localFlags & VK_DEBUG_REPORT_ERROR_BIT_EXT) {
        process error message
    }
    if (localFlags & VK_DEBUG_REPORT_DEBUG_BIT_EXT) {
        process debug message
    }
```

The validation layers do use them in a hierarchical way (`ERROR` > `WARN` > `PERF`, `WARN` > `DEBUG` > `INFO`) and they (at least at the time of this writing) only set one bit at a time. But it is not a requirement of this extension.

It is possible that a layer may intercept and change, or augment the flags with extension values the application's debug report handler may not be familiar with, so it is important to treat each flag independently.

2) Should there be a VU requiring VkDebugReportCallbackCreateInfoEXT::`flags` to be non-zero?

**RESOLVED**: It may not be very useful, but we do not need VU statement requiring the `VkDebugReportCallbackCreateInfoEXT`::`msgFlags` at create-time to be non-zero. One can imagine that apps may prefer it as it allows them to set the mask as desired - including nothing - at runtime without having to check.

3) What is the difference between `VK_DEBUG_REPORT_DEBUG_BIT_EXT` and `VK_DEBUG_REPORT_INFORMATION_BIT_EXT`?

**RESOLVED**: `VK_DEBUG_REPORT_DEBUG_BIT_EXT` specifies information that could be useful debugging the Vulkan implementation itself.

4) How do you compare handles returned by the debug_report callback to the application's handles?

**RESOLVED**: Due to the different nature of dispatchable and nondispatchable handles there is no generic way (that we know of) that works for common compilers with 32bit, 64bit, C and C++. We recommend applications use the same cast that the validation layers use:

+

```
reinterpret_cast<uint64_t &>(dispatchableHandle)
(uint64_t)(nondispatchableHandle)
```

+ This does require that the application treat dispatchable and nondispatchable handles differently.

### Version History

- Revision 1, 2015-05-20 (Courtney Goetzenleuchter)
  - Initial draft, based on LunarG KHR spec, other KHR specs
- Revision 2, 2016-02-16 (Courtney Goetzenleuchter)
  - Update usage, documentation
- Revision 3, 2016-06-14 (Courtney Goetzenleuchter)
  - Update VK_EXT_DEBUG_REPORT_SPEC_VERSION to indicate added support for vkCreateInstance and vkDestroyInstance
- Revision 4, 2016-12-08 (Mark Lobodzinski)
  - Added Display_KHR, DisplayModeKHR extension objects
  - Added ObjectTable_NVX, IndirectCommandsLayout_NVX extension objects
  - Bumped spec revision
  - Retroactively added version history
- Revision 5, 2017-01-31 (Baldur Karlsson)
  - Moved definition of VkDebugReportObjectTypeEXT from debug marker chapter
- Revision 6, 2017-01-31 (Baldur Karlsson)
  - Added VK_DEBUG_REPORT_OBJECT_TYPE_DESCRIPTOR_UPDATE_TEMPLATE_KHR_EXT
- Revision 7, 2017-04-20 (Courtney Goeltzenleuchter)
  - Clarify wording and address questions from developers.
- Revision 8, 2017-04-21 (Courtney Goeltzenleuchter)
  - Remove unused enum VkDebugReportErrorEXT
- Revision 9, 2017-09-12 (Tobias Hector)
  - Added interactions with Vulkan 1.1
- Revision 10, 2020-12-14 (Courtney Goetzenleuchter)
  - Add issue 4 discussing matching handles returned by the extension, based on suggestion in public issue 368.

## VK_EXT_global_priority

### Name String

`VK_EXT_global_priority`

### Extension Type

Device extension

### Registered Extension Number

175

**Revision**

2

**Ratification Status**

Not ratified

**Extension and Version Dependencies**

None

**Deprecation State**

- *Promoted* to VK_KHR_global_priority extension

**Contact**

- Andres Rodriguez [GitHub]lostgoat

**Other Extension Metadata**

**Last Modified Date**

2017-10-06

**IP Status**

No known IP claims.

**Contributors**

- Andres Rodriguez, Valve
- Pierre-Loup Griffais, Valve
- Dan Ginsburg, Valve
- Mitch Singer, AMD

**Description**

In Vulkan, users can specify device-scope queue priorities. In some cases it may be useful to extend this concept to a system-wide scope. This extension provides a mechanism for callers to set their system-wide priority. The default queue priority is `VK_QUEUE_GLOBAL_PRIORITY_MEDIUM_EXT`.

The driver implementation will attempt to skew hardware resource allocation in favor of the higher-priority task. Therefore, higher-priority work may retain similar latency and throughput characteristics even if the system is congested with lower priority work.

The global priority level of a queue shall take precedence over the per-process queue priority (`VkDeviceQueueCreateInfo`::`pQueuePriorities`).

Abuse of this feature may result in starving the rest of the system from hardware resources. Therefore, the driver implementation may deny requests to acquire a priority above the default priority (`VK_QUEUE_GLOBAL_PRIORITY_MEDIUM_EXT`) if the caller does not have sufficient privileges. In this scenario `VK_ERROR_NOT_PERMITTED_EXT` is returned.

The driver implementation may fail the queue allocation request if resources required to complete

the operation have been exhausted (either by the same process or a different process). In this scenario `VK_ERROR_INITIALIZATION_FAILED` is returned.

### New Structures

- Extending VkDeviceQueueCreateInfo:
  - VkDeviceQueueGlobalPriorityCreateInfoEXT

### New Enums

- VkQueueGlobalPriorityEXT

### New Enum Constants

- `VK_EXT_GLOBAL_PRIORITY_EXTENSION_NAME`
- `VK_EXT_GLOBAL_PRIORITY_SPEC_VERSION`
- Extending VkResult:
  - `VK_ERROR_NOT_PERMITTED_EXT`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_DEVICE_QUEUE_GLOBAL_PRIORITY_CREATE_INFO_EXT`

### Version History

- Revision 2, 2017-11-03 (Andres Rodriguez)
  - Fixed VkQueueGlobalPriorityEXT missing _EXT suffix
- Revision 1, 2017-10-06 (Andres Rodriguez)
  - First version.

## VK_EXT_global_priority_query

### Name String

`VK_EXT_global_priority_query`

### Extension Type

Device extension

### Registered Extension Number

389

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_EXT_global_priority
and
VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### Deprecation State

- *Promoted* to VK_KHR_global_priority extension

### Contact

- Yiwei Zhang [GitHub]zhangyiwei

### Other Extension Metadata

### Last Modified Date

2021-03-29

### IP Status

No known IP claims.

### Contributors

- Yiwei Zhang, Google

### Description

This device extension allows applications to query the global queue priorities supported by a queue family. It allows implementations to report which global priority levels are treated differently by the implementation, instead of silently mapping multiple requested global priority levels to the same internal priority, or using device creation failure to signal that a requested priority is not supported. It is intended primarily for use by system integration along with certain platform-specific priority enforcement rules.

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceGlobalPriorityQueryFeaturesEXT
- Extending VkQueueFamilyProperties2:
  - VkQueueFamilyGlobalPriorityPropertiesEXT

### New Enum Constants

- `VK_EXT_GLOBAL_PRIORITY_QUERY_EXTENSION_NAME`
- `VK_EXT_GLOBAL_PRIORITY_QUERY_SPEC_VERSION`
- `VK_MAX_GLOBAL_PRIORITY_SIZE_EXT`
- Extending VkStructureType:

- VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_GLOBAL_PRIORITY_QUERY_FEATURES_EXT
- VK_STRUCTURE_TYPE_QUEUE_FAMILY_GLOBAL_PRIORITY_PROPERTIES_EXT

**Issues**

1) Can we additionally query whether a caller is permitted to acquire a specific global queue priority in this extension?

**RESOLVED**: No. Whether a caller has enough privilege goes with the OS, and the Vulkan driver cannot really guarantee that the privilege will not change in between this query and the actual queue creation call.

2) If more than 1 queue using global priority is requested, is there a good way to know which queue is failing the device creation?

**RESOLVED**: No. There is not a good way at this moment, and it is also not quite actionable for the applications to know that because the information may not be accurate. Queue creation can fail because of runtime constraints like insufficient privilege or lack of resource, and the failure is not necessarily tied to that particular queue configuration requested.

**Version History**

- Revision 1, 2021-03-29 (Yiwei Zhang)

## VK_EXT_index_type_uint8

**Name String**

VK_EXT_index_type_uint8

**Extension Type**

Device extension

**Registered Extension Number**

266

**Revision**

1

**Ratification Status**

Not ratified

**Extension and Version Dependencies**

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

**Deprecation State**

- *Promoted* to VK_KHR_index_type_uint8 extension

### Contact

- Piers Daniell [GitHub]pdaniell-nv

### Other Extension Metadata

### Last Modified Date

2019-05-02

### IP Status

No known IP claims.

### Contributors

- Jeff Bolz, NVIDIA

### Description

This extension allows `uint8_t` indices to be used with vkCmdBindIndexBuffer.

### Promotion to `VK_KHR_index_type_uint8`

All functionality in this extension is included in `VK_KHR_index_type_uint8`, with the suffix changed to KHR. The original enum names are still available as aliases of the KHR functionality.

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceIndexTypeUint8FeaturesEXT

### New Enum Constants

- `VK_EXT_INDEX_TYPE_UINT8_EXTENSION_NAME`
- `VK_EXT_INDEX_TYPE_UINT8_SPEC_VERSION`
- Extending VkIndexType:
  - `VK_INDEX_TYPE_UINT8_EXT`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_INDEX_TYPE_UINT8_FEATURES_EXT`

### Version History

- Revision 1, 2019-05-02 (Piers Daniell)
  - Internal revisions

## VK_EXT_line_rasterization

### Name String

`VK_EXT_line_rasterization`

**Extension Type**

Device extension

**Registered Extension Number**

260

**Revision**

1

**Ratification Status**

Not ratified

**Extension and Version Dependencies**

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

**Deprecation State**

- *Promoted* to VK_KHR_line_rasterization extension

**Special Use**

- CAD support

**Contact**

- Jeff Bolz [GitHub]jeffbolznv

**Other Extension Metadata**

**Last Modified Date**

2019-05-09

**IP Status**

No known IP claims.

**Contributors**

- Jeff Bolz, NVIDIA
- Allen Jensen, NVIDIA
- Faith Ekstrand, Intel

**Description**

This extension adds some line rasterization features that are commonly used in CAD applications and supported in other APIs like OpenGL. Bresenham-style line rasterization is supported, smooth rectangular lines (coverage to alpha) are supported, and stippled lines are supported for all three line rasterization modes.

### Promotion to `VK_KHR_line_rasterization`

All functionality in this extension is included in `VK_KHR_line_rasterization`, with the suffix changed to KHR. The original enum names are still available as aliases of the KHR functionality.

### New Commands

- vkCmdSetLineStippleEXT

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceLineRasterizationFeaturesEXT
- Extending VkPhysicalDeviceProperties2:
  - VkPhysicalDeviceLineRasterizationPropertiesEXT
- Extending VkPipelineRasterizationStateCreateInfo:
  - VkPipelineRasterizationLineStateCreateInfoEXT

### New Enums

- VkLineRasterizationModeEXT

### New Enum Constants

- `VK_EXT_LINE_RASTERIZATION_EXTENSION_NAME`
- `VK_EXT_LINE_RASTERIZATION_SPEC_VERSION`
- Extending VkDynamicState:
  - `VK_DYNAMIC_STATE_LINE_STIPPLE_EXT`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_LINE_RASTERIZATION_FEATURES_EXT`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_LINE_RASTERIZATION_PROPERTIES_EXT`
  - `VK_STRUCTURE_TYPE_PIPELINE_RASTERIZATION_LINE_STATE_CREATE_INFO_EXT`

### Issues

1) Do we need to support Bresenham-style and smooth lines with more than one rasterization sample? i.e. the equivalent of glDisable(GL_MULTISAMPLE) in OpenGL when the framebuffer has more than one sample?

**RESOLVED**: Yes. For simplicity, Bresenham line rasterization carries forward a few restrictions from OpenGL, such as not supporting per-sample shading, alpha to coverage, or alpha to one.

### Version History

- Revision 1, 2019-05-09 (Jeff Bolz)

- Initial draft

## VK_EXT_load_store_op_none

### Name String

`VK_EXT_load_store_op_none`

### Extension Type

Device extension

### Registered Extension Number

401

### Revision

1

### Ratification Status

Ratified

### Extension and Version Dependencies

None

### Deprecation State

- *Promoted* to VK_KHR_load_store_op_none extension

### Contact

- Shahbaz Youssefi [GitHub]syoussefi

### Other Extension Metadata

### Last Modified Date

2021-06-06

### Contributors

- Shahbaz Youssefi, Google
- Bill Licea-Kane, Qualcomm Technologies, Inc.
- Tobias Hector, AMD

### Description

This extension incorporates `VK_ATTACHMENT_STORE_OP_NONE_EXT` from `VK_QCOM_render_pass_store_ops`, enabling applications to avoid unnecessary synchronization when an attachment is not written during a render pass.

Additionally, `VK_ATTACHMENT_LOAD_OP_NONE_EXT` is introduced to avoid unnecessary synchronization when an attachment is not used during a render pass at all. In combination with `VK_ATTACHMENT_STORE_OP_NONE_EXT`, this is useful as an alternative to preserve attachments in applications that cannot decide if an attachment will be used in a render pass until after the

necessary pipelines have been created.

### Promotion to VK_KHR_load_store_op_none

All functionality in this extension is included in VK_KHR_load_store_op_none, with the suffix changed to KHR. The original enum names are still available as aliases of the KHR functionality.

### New Enum Constants

- VK_EXT_LOAD_STORE_OP_NONE_EXTENSION_NAME
- VK_EXT_LOAD_STORE_OP_NONE_SPEC_VERSION
- Extending VkAttachmentLoadOp:
  - VK_ATTACHMENT_LOAD_OP_NONE_EXT
- Extending VkAttachmentStoreOp:
  - VK_ATTACHMENT_STORE_OP_NONE_EXT

| | |
|---|---|
| **NOTE** | While VK_ATTACHMENT_STORE_OP_NONE is part of Vulkan 1.3, this extension was not promoted to core either in whole or in part. This functionality was promoted from VK_KHR_dynamic_rendering. |

### Version History

- Revision 1, 2021-06-06 (Shahbaz Youssefi)
  - Initial revision, based on VK_QCOM_render_pass_store_ops.
  - Added VK_ATTACHMENT_LOAD_OP_NONE_EXT.

## VK_EXT_shader_subgroup_vote

### Name String

VK_EXT_shader_subgroup_vote

### Extension Type

Device extension

### Registered Extension Number

66

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

None

**SPIR-V Dependencies**

- SPV_KHR_subgroup_vote

**Deprecation State**

- *Deprecated* by Vulkan 1.1

**Contact**

- Daniel Koch [GitHub]dgkoch

### Other Extension Metadata

**Last Modified Date**

2016-11-28

**IP Status**

No known IP claims.

**Interactions and External Dependencies**

- This extension provides API support for `GL_ARB_shader_group_vote`

**Contributors**

- Neil Henning, Codeplay
- Daniel Koch, NVIDIA Corporation

### Description

This extension adds support for the following SPIR-V extension in Vulkan:

- `SPV_KHR_subgroup_vote`

This extension provides new SPIR-V instructions:

- `OpSubgroupAllKHR`,
- `OpSubgroupAnyKHR`, and
- `OpSubgroupAllEqualKHR`.

to compute the composite of a set of boolean conditions across a group of shader invocations that are running concurrently (a *subgroup*). These composite results may be used to execute shaders more efficiently on a VkPhysicalDevice.

When using GLSL source-based shader languages, the following shader functions from GL_ARB_shader_group_vote can map to these SPIR-V instructions:

- `anyInvocationARB`() → `OpSubgroupAnyKHR`,
- `allInvocationsARB`() → `OpSubgroupAllKHR`, and
- `allInvocationsEqualARB`() → `OpSubgroupAllEqualKHR`.

The subgroup across which the boolean conditions are evaluated is implementation-dependent, and this extension provides no guarantee over how individual shader invocations are assigned to

subgroups. In particular, a subgroup has no necessary relationship with the compute shader *local workgroup* — any pair of shader invocations in a compute local workgroup may execute in different subgroups as used by these instructions.

Compute shaders operate on an explicitly specified group of threads (a local workgroup), but many implementations will also group non-compute shader invocations and execute them concurrently. When executing code like

```
if (condition) {
  result = do_fast_path();
} else {
  result = do_general_path();
}
```

where `condition` diverges between invocations, an implementation might first execute `do_fast_path`() for the invocations where `condition` is true and leave the other invocations dormant. Once `do_fast_path`() returns, it might call `do_general_path`() for invocations where `condition` is `false` and leave the other invocations dormant. In this case, the shader executes **both** the fast and the general path and might be better off just using the general path for all invocations.

This extension provides the ability to avoid divergent execution by evaluating a condition across an entire subgroup using code like:

```
if (allInvocationsARB(condition)) {
  result = do_fast_path();
} else {
  result = do_general_path();
}
```

The built-in function `allInvocationsARB`() will return the same value for all invocations in the group, so the group will either execute `do_fast_path`() or `do_general_path`(), but never both. For example, shader code might want to evaluate a complex function iteratively by starting with an approximation of the result and then refining the approximation. Some input values may require a small number of iterations to generate an accurate result (`do_fast_path`) while others require a larger number (`do_general_path`). In another example, shader code might want to evaluate a complex function (`do_general_path`) that can be greatly simplified when assuming a specific value for one of its inputs (`do_fast_path`).

#### Deprecated by Vulkan 1.1

All functionality in this extension is superseded by the core Vulkan 1.1 subgroup operations.

#### New Enum Constants

- `VK_EXT_SHADER_SUBGROUP_VOTE_EXTENSION_NAME`
- `VK_EXT_SHADER_SUBGROUP_VOTE_SPEC_VERSION`

### New SPIR-V Capabilities

- `SubgroupVoteKHR`

### Version History

- Revision 1, 2016-11-28 (Daniel Koch)
  - Initial draft

## VK_EXT_validation_features

### Name String

`VK_EXT_validation_features`

### Extension Type

Instance extension

### Registered Extension Number

248

### Revision

6

### Ratification Status

Not ratified

### Extension and Version Dependencies

None

### Deprecation State

- *Deprecated* by VK_EXT_layer_settings extension

### Special Use

- Debugging tools

### Contact

- Karl Schultz [GitHub]karl-lunarg

### Other Extension Metadata

### Last Modified Date

2018-11-14

### IP Status

No known IP claims.

### Contributors

- Karl Schultz, LunarG
- Dave Houlton, LunarG

- Mark Lobodzinski, LunarG
- Camden Stocker, LunarG
- Tony Barbour, LunarG
- John Zulauf, LunarG

### Description

This extension provides the VkValidationFeaturesEXT struct that can be included in the `pNext` chain of the VkInstanceCreateInfo structure passed as the `pCreateInfo` parameter of vkCreateInstance. The structure contains an array of VkValidationFeatureEnableEXT enum values that enable specific validation features that are disabled by default. The structure also contains an array of VkValidationFeatureDisableEXT enum values that disable specific validation layer features that are enabled by default.

### Deprecation by `VK_EXT_layer_settings`

Functionality in this extension is subsumed into the `VK_EXT_layer_settings` extension.

### New Structures

- Extending VkInstanceCreateInfo, VkShaderModuleCreateInfo, VkShaderCreateInfoEXT:
  - VkValidationFeaturesEXT

### New Enums

- VkValidationFeatureDisableEXT
- VkValidationFeatureEnableEXT

### New Enum Constants

- `VK_EXT_VALIDATION_FEATURES_EXTENSION_NAME`
- `VK_EXT_VALIDATION_FEATURES_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_VALIDATION_FEATURES_EXT`

### Version History

- Revision 1, 2018-11-14 (Karl Schultz)
  - Initial revision
- Revision 2, 2019-08-06 (Mark Lobodzinski)
  - Add Best Practices enable
- Revision 3, 2020-03-04 (Tony Barbour)
  - Add Debug Printf enable
- Revision 4, 2020-07-29 (John Zulauf)

    - Add Synchronization Validation enable
- Revision 5, 2021-05-18 (Tony Barbour)
    - Add Shader Validation Cache disable
- Revision 6, 2023-09-25 (Christophe Riccio)
    - Marked as deprecated by VK_EXT_layer_settings

## VK_EXT_validation_flags

### Name String

`VK_EXT_validation_flags`

### Extension Type

Instance extension

### Registered Extension Number

62

### Revision

3

### Ratification Status

Not ratified

### Extension and Version Dependencies

None

### Deprecation State

- *Deprecated* by VK_EXT_layer_settings extension

### Special Use

- Debugging tools

### Contact

- Tobin Ehlis [GitHub]tobine

### Other Extension Metadata

### Last Modified Date

2019-08-19

### IP Status

No known IP claims.

### Contributors

- Tobin Ehlis, Google
- Courtney Goeltzenleuchter, Google

### Description

This extension provides the VkValidationFlagsEXT struct that can be included in the `pNext` chain of the VkInstanceCreateInfo structure passed as the `pCreateInfo` parameter of vkCreateInstance. The structure contains an array of VkValidationCheckEXT values that will be disabled by the validation layers.

### Deprecation by `VK_EXT_layer_settings`

Functionality in this extension is subsumed into the `VK_EXT_layer_settings` extension.

### New Structures

- Extending VkInstanceCreateInfo:
  - VkValidationFlagsEXT

### New Enums

- VkValidationCheckEXT

### New Enum Constants

- `VK_EXT_VALIDATION_FLAGS_EXTENSION_NAME`
- `VK_EXT_VALIDATION_FLAGS_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_VALIDATION_FLAGS_EXT`

### Version History

- Revision 3, 2023-09-25 (Christophe Riccio)
  - Marked as deprecated by VK_EXT_layer_settings
- Revision 2, 2019-08-19 (Mark Lobodzinski)
  - Marked as deprecated by VK_EXT_validation_features
- Revision 1, 2016-08-26 (Courtney Goeltzenleuchter)
  - Initial draft

## VK_EXT_vertex_attribute_divisor

### Name String

`VK_EXT_vertex_attribute_divisor`

### Extension Type

Device extension

### Registered Extension Number

191

### Revision

3

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### Deprecation State

- *Promoted* to VK_KHR_vertex_attribute_divisor extension

### Contact

- Vikram Kushwaha [GitHub]vkushwaha

### Other Extension Metadata

### Last Modified Date

2018-08-03

### IP Status

No known IP claims.

### Contributors

- Vikram Kushwaha, NVIDIA
- Faith Ekstrand, Intel

### Description

This extension allows instance-rate vertex attributes to be repeated for certain number of instances instead of advancing for every instance when instanced rendering is enabled.

### New Structures

- VkVertexInputBindingDivisorDescriptionEXT
- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceVertexAttributeDivisorFeaturesEXT
- Extending VkPhysicalDeviceProperties2:
  - VkPhysicalDeviceVertexAttributeDivisorPropertiesEXT
- Extending VkPipelineVertexInputStateCreateInfo:
  - VkPipelineVertexInputDivisorStateCreateInfoEXT

### New Enum Constants

- `VK_EXT_VERTEX_ATTRIBUTE_DIVISOR_EXTENSION_NAME`
- `VK_EXT_VERTEX_ATTRIBUTE_DIVISOR_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_VERTEX_ATTRIBUTE_DIVISOR_FEATURES_EXT`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_VERTEX_ATTRIBUTE_DIVISOR_PROPERTIES_EXT`
  - `VK_STRUCTURE_TYPE_PIPELINE_VERTEX_INPUT_DIVISOR_STATE_CREATE_INFO_EXT`

### Issues

1) What is the effect of a non-zero value for `firstInstance`?

**RESOLVED**: The Vulkan API should follow the OpenGL convention and offset attribute fetching by `firstInstance` while computing vertex attribute offsets.

2) Should zero be an allowed divisor?

**RESOLVED**: Yes. A zero divisor means the vertex attribute is repeated for all instances.

### Examples

To create a vertex binding such that the first binding uses instanced rendering and the same attribute is used for every 4 draw instances, an application could use the following set of structures:

```
    const VkVertexInputBindingDivisorDescriptionEXT divisorDesc =
    {
        .binding = 0,
        .divisor = 4
    };

    const VkPipelineVertexInputDivisorStateCreateInfoEXT divisorInfo =
    {
        .sType =
VK_STRUCTURE_TYPE_PIPELINE_VERTEX_INPUT_DIVISOR_STATE_CREATE_INFO_EXT,
        .pNext = NULL,
        .vertexBindingDivisorCount = 1,
        .pVertexBindingDivisors = &divisorDesc
    }

    const VkVertexInputBindingDescription binding =
    {
        .binding = 0,
        .stride = sizeof(Vertex),
        .inputRate = VK_VERTEX_INPUT_RATE_INSTANCE
    };

    const VkPipelineVertexInputStateCreateInfo viInfo =
    {
```

```
        .sType = VK_STRUCTURE_TYPE_PIPELINE_VERTEX_INPUT_CREATE_INFO,
        .pNext = &divisorInfo,
        ...
    };
    //...
```

#### Version History

- Revision 1, 2017-12-04 (Vikram Kushwaha)
  - First Version
- Revision 2, 2018-07-16 (Faith Ekstrand)
  - Adjust the interaction between `divisor` and `firstInstance` to match the OpenGL convention.
  - Disallow divisors of zero.
- Revision 3, 2018-08-03 (Vikram Kushwaha)
  - Allow a zero divisor.
  - Add a physical device features structure to query/enable this feature.

## VK_AMD_draw_indirect_count

#### Name String

`VK_AMD_draw_indirect_count`

#### Extension Type

Device extension

#### Registered Extension Number

34

#### Revision

2

#### Ratification Status

Not ratified

#### Extension and Version Dependencies

None

#### Deprecation State

- *Promoted* to VK_KHR_draw_indirect_count extension
  - Which in turn was *promoted* to Vulkan 1.2

#### Contact

- Daniel Rakos [GitHub]drakos-amd

### Other Extension Metadata

#### Last Modified Date

2016-08-23

#### IP Status

No known IP claims.

#### Contributors

- Matthaeus G. Chajdas, AMD
- Derrick Owens, AMD
- Graham Sellers, AMD
- Daniel Rakos, AMD
- Dominik Witczak, AMD

### Description

This extension allows an application to source the number of draws for indirect drawing commands from a buffer. This enables applications to generate an arbitrary number of drawing commands and execute them without host intervention.

### Promotion to `VK_KHR_draw_indirect_count`

All functionality in this extension is included in `VK_KHR_draw_indirect_count`, with the suffix changed to KHR. The original type, enum and command names are still available as aliases of the core functionality.

### New Commands

- vkCmdDrawIndexedIndirectCountAMD
- vkCmdDrawIndirectCountAMD

### New Enum Constants

- `VK_AMD_DRAW_INDIRECT_COUNT_EXTENSION_NAME`
- `VK_AMD_DRAW_INDIRECT_COUNT_SPEC_VERSION`

### Version History

- Revision 2, 2016-08-23 (Dominik Witczak)
  - Minor fixes
- Revision 1, 2016-07-21 (Matthaeus Chajdas)
  - Initial draft

## VK_AMD_gpu_shader_half_float

### Name String

`VK_AMD_gpu_shader_half_float`

### Extension Type

Device extension

### Registered Extension Number

37

### Revision

2

### Ratification Status

Not ratified

### Extension and Version Dependencies

None

### SPIR-V Dependencies

- SPV_AMD_gpu_shader_half_float

### Deprecation State

- *Deprecated* by VK_KHR_shader_float16_int8 extension
  - Which in turn was *promoted* to Vulkan 1.2

### Contact

- Dominik Witczak [GitHub]dominikwitczakamd

### Other Extension Metadata

### Last Modified Date

2019-04-11

### IP Status

No known IP claims.

### Interactions and External Dependencies

- This extension provides API support for `GL_AMD_gpu_shader_half_float`

### Contributors

- Daniel Rakos, AMD
- Dominik Witczak, AMD
- Donglin Wei, AMD
- Graham Sellers, AMD
- Qun Lin, AMD

- Rex Xu, AMD

#### Description

This extension adds support for using half float variables in shaders.

#### Deprecation by `VK_KHR_shader_float16_int8`

Functionality in this extension was included in `VK_KHR_shader_float16_int8` extension, when VkPhysicalDeviceShaderFloat16Int8FeaturesKHR::`shaderFloat16` is enabled.

#### New Enum Constants

- `VK_AMD_GPU_SHADER_HALF_FLOAT_EXTENSION_NAME`
- `VK_AMD_GPU_SHADER_HALF_FLOAT_SPEC_VERSION`

#### Version History

- Revision 2, 2019-04-11 (Tobias Hector)
  - Marked as deprecated
- Revision 1, 2016-09-21 (Dominik Witczak)
  - Initial draft

### VK_AMD_gpu_shader_int16

#### Name String

`VK_AMD_gpu_shader_int16`

#### Extension Type

Device extension

#### Registered Extension Number

133

#### Revision

2

#### Ratification Status

Not ratified

#### Extension and Version Dependencies

None

#### SPIR-V Dependencies

- SPV_AMD_gpu_shader_int16

#### Deprecation State

- *Deprecated* by VK_KHR_shader_float16_int8 extension

    - Which in turn was *promoted* to Vulkan 1.2

**Contact**

- Qun Lin [GitHub]linqun

### Other Extension Metadata

**Last Modified Date**

2019-04-11

**IP Status**

No known IP claims.

**Interactions and External Dependencies**

- This extension provides API support for `GL_AMD_gpu_shader_int16`

**Contributors**

- Daniel Rakos, AMD
- Dominik Witczak, AMD
- Matthaeus G. Chajdas, AMD
- Rex Xu, AMD
- Timothy Lottes, AMD
- Zhi Cai, AMD

### Description

This extension adds support for using 16-bit integer variables in shaders.

### Deprecation by `VK_KHR_shader_float16_int8`

Functionality in this extension was included in `VK_KHR_shader_float16_int8` extension, when VkPhysicalDeviceFeatures::`shaderInt16` and VkPhysicalDeviceShaderFloat16Int8FeaturesKHR::`shaderFloat16` are enabled.

### New Enum Constants

- `VK_AMD_GPU_SHADER_INT16_EXTENSION_NAME`
- `VK_AMD_GPU_SHADER_INT16_SPEC_VERSION`

### Version History

- Revision 2, 2019-04-11 (Tobias Hector)
    - Marked as deprecated
- Revision 1, 2017-06-18 (Dominik Witczak)
    - First version

## VK_AMD_negative_viewport_height

### Name String

`VK_AMD_negative_viewport_height`

### Extension Type

Device extension

### Registered Extension Number

36

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

None

### Deprecation State

- *Obsoleted* by VK_KHR_maintenance1 extension
  - Which in turn was *promoted* to Vulkan 1.1

### Contact

- Matthaeus G. Chajdas [GitHub]anteru

### Other Extension Metadata

### Last Modified Date

2016-09-02

### IP Status

No known IP claims.

### Contributors

- Matthaeus G. Chajdas, AMD
- Graham Sellers, AMD
- Baldur Karlsson

### Description

This extension allows an application to specify a negative viewport height. The result is that the viewport transformation will flip along the y-axis.

- Allow negative height to be specified in the VkViewport::`height` field to perform y-inversion of the clip-space to framebuffer-space transform. This allows apps to avoid having to use `gl_Position.y = -gl_Position.y` in shaders also targeting other APIs.

#### Obsoletion by VK_KHR_maintenance1 and Vulkan 1.1

Functionality in this extension is included in VK_KHR_maintenance1 and subsequently Vulkan 1.1. Due to some slight behavioral differences, this extension **must** not be enabled alongside VK_KHR_maintenance1, or in an instance created with version 1.1 or later requested in VkApplicationInfo::apiVersion.

#### New Enum Constants

- VK_AMD_NEGATIVE_VIEWPORT_HEIGHT_EXTENSION_NAME
- VK_AMD_NEGATIVE_VIEWPORT_HEIGHT_SPEC_VERSION

#### Version History

- Revision 1, 2016-09-02 (Matthaeus Chajdas)
  - Initial draft

### VK_ARM_rasterization_order_attachment_access

#### Name String

VK_ARM_rasterization_order_attachment_access

#### Extension Type

Device extension

#### Registered Extension Number

343

#### Revision

1

#### Ratification Status

Not ratified

#### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

#### Deprecation State

- *Promoted* to VK_EXT_rasterization_order_attachment_access extension

#### Contact

- Jan-Harald Fredriksen [GitHub]janharaldfredriksen-arm

#### Other Extension Metadata

#### Last Modified Date

2021-11-12

### IP Status

No known IP claims.

### Contributors

- Tobias Hector, AMD
- Jan-Harald Fredriksen, Arm

### Description

Render passes, and specifically subpass dependencies, enable much of the same functionality as the framebuffer fetch and pixel local storage extensions did for OpenGL ES. But certain techniques such as programmable blending are awkward or impractical to implement with these alone, in part because a self-dependency is required every time a fragment will read a value at a given sample coordinate.

This extension extends the mechanism of input attachments to allow access to framebuffer attachments when used as both input and color, or depth/stencil, attachments from one fragment to the next, in rasterization order, without explicit synchronization.

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceRasterizationOrderAttachmentAccessFeaturesARM

### New Enum Constants

- `VK_ARM_RASTERIZATION_ORDER_ATTACHMENT_ACCESS_EXTENSION_NAME`
- `VK_ARM_RASTERIZATION_ORDER_ATTACHMENT_ACCESS_SPEC_VERSION`
- Extending VkPipelineColorBlendStateCreateFlagBits:
  - `VK_PIPELINE_COLOR_BLEND_STATE_CREATE_RASTERIZATION_ORDER_ATTACHMENT_ACCESS_BIT_ARM`
- Extending VkPipelineDepthStencilStateCreateFlagBits:
  - `VK_PIPELINE_DEPTH_STENCIL_STATE_CREATE_RASTERIZATION_ORDER_ATTACHMENT_DEPTH_ACCESS_BIT_ARM`
  - `VK_PIPELINE_DEPTH_STENCIL_STATE_CREATE_RASTERIZATION_ORDER_ATTACHMENT_STENCIL_ACCESS_BIT_ARM`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_RASTERIZATION_ORDER_ATTACHMENT_ACCESS_FEATURES_ARM`
- Extending VkSubpassDescriptionFlagBits:
  - `VK_SUBPASS_DESCRIPTION_RASTERIZATION_ORDER_ATTACHMENT_COLOR_ACCESS_BIT_ARM`
  - `VK_SUBPASS_DESCRIPTION_RASTERIZATION_ORDER_ATTACHMENT_DEPTH_ACCESS_BIT_ARM`
  - `VK_SUBPASS_DESCRIPTION_RASTERIZATION_ORDER_ATTACHMENT_STENCIL_ACCESS_BIT_ARM`

### Issues

1) Is there any interaction with the `VK_KHR_dynamic_rendering` extension?

No. This extension only affects reads from input attachments. Render pass instances begun with vkCmdBeginRenderingKHR do not have input attachments and a different mechanism will be needed to provide similar functionality in this case.

### Examples

None.

### Version History

- Revision 1, 2021-11-12 (Jan-Harald Fredriksen)
  - Initial draft

## VK_IMG_format_pvrtc

### Name String

`VK_IMG_format_pvrtc`

### Extension Type

Device extension

### Registered Extension Number

55

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

None

### Deprecation State

- *Deprecated* without replacement

### Contact

- Stuart Smith

### Other Extension Metadata

### Last Modified Date

2019-09-02

### IP Status

Imagination Technologies Proprietary

### Contributors

- Stuart Smith, Imagination Technologies

### Description

`VK_IMG_format_pvrtc` provides additional texture compression functionality specific to Imagination Technologies PowerVR Texture compression format (called PVRTC).

| | |
|---|---|
| **NOTE** | As also noted in the Khronos Data Format Specification, PVRTC1 images must have dimensions that are a power of two. |

### Deprecation

Both PVRTC1 and PVRTC2 are slower than standard image formats on PowerVR GPUs, and support will be removed from future hardware.

### New Enum Constants

- `VK_IMG_FORMAT_PVRTC_EXTENSION_NAME`
- `VK_IMG_FORMAT_PVRTC_SPEC_VERSION`
- Extending VkFormat:
  - `VK_FORMAT_PVRTC1_2BPP_SRGB_BLOCK_IMG`
  - `VK_FORMAT_PVRTC1_2BPP_UNORM_BLOCK_IMG`
  - `VK_FORMAT_PVRTC1_4BPP_SRGB_BLOCK_IMG`
  - `VK_FORMAT_PVRTC1_4BPP_UNORM_BLOCK_IMG`
  - `VK_FORMAT_PVRTC2_2BPP_SRGB_BLOCK_IMG`
  - `VK_FORMAT_PVRTC2_2BPP_UNORM_BLOCK_IMG`
  - `VK_FORMAT_PVRTC2_4BPP_SRGB_BLOCK_IMG`
  - `VK_FORMAT_PVRTC2_4BPP_UNORM_BLOCK_IMG`

### Version History

- Revision 1, 2019-09-02 (Stuart Smith)
  - Initial version

## VK_MVK_ios_surface

### Name String

`VK_MVK_ios_surface`

### Extension Type

Instance extension

### Registered Extension Number

123

### Revision

3

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_surface

### Deprecation State

- *Deprecated* by VK_EXT_metal_surface extension

### Contact

- Bill Hollings [GitHub]billhollings

### Other Extension Metadata

### Last Modified Date

2020-07-31

### IP Status

No known IP claims.

### Contributors

- Bill Hollings, The Brenwill Workshop Ltd.

### Description

The `VK_MVK_ios_surface` extension is an instance extension. It provides a mechanism to create a VkSurfaceKHR object (defined by the `VK_KHR_surface` extension) based on a `UIView`, the native surface type of iOS, which is underpinned by a CAMetalLayer, to support rendering to the surface using Apple's Metal framework.

### Deprecation by `VK_EXT_metal_surface`

The `VK_MVK_ios_surface` extension is considered deprecated and has been superseded by the `VK_EXT_metal_surface` extension.

### New Commands

- vkCreateIOSSurfaceMVK

### New Structures

- VkIOSSurfaceCreateInfoMVK

### New Bitmasks

- VkIOSSurfaceCreateFlagsMVK

### New Enum Constants

- `VK_MVK_IOS_SURFACE_EXTENSION_NAME`
- `VK_MVK_IOS_SURFACE_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_IOS_SURFACE_CREATE_INFO_MVK`

### Version History

- Revision 1, 2017-02-15 (Bill Hollings)
  - Initial draft.
- Revision 2, 2017-02-24 (Bill Hollings)
  - Minor syntax fix to emphasize firm requirement for `UIView` to be backed by a `CAMetalLayer`.
- Revision 3, 2020-07-31 (Bill Hollings)
  - Update documentation on requirements for `UIView`.
  - Mark as deprecated by `VK_EXT_metal_surface`.

## VK_MVK_macos_surface

### Name String

`VK_MVK_macos_surface`

### Extension Type

Instance extension

### Registered Extension Number

124

### Revision

3

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_surface

### Deprecation State

- *Deprecated* by VK_EXT_metal_surface extension

### Contact

- Bill Hollings [GitHub]billhollings

### Other Extension Metadata

### Last Modified Date

2020-07-31

### IP Status

No known IP claims.

### Contributors

- Bill Hollings, The Brenwill Workshop Ltd.

### Description

The `VK_MVK_macos_surface` extension is an instance extension. It provides a mechanism to create a VkSurfaceKHR object (defined by the `VK_KHR_surface` extension) based on an `NSView`, the native surface type of macOS, which is underpinned by a CAMetalLayer, to support rendering to the surface using Apple's Metal framework.

### Deprecation by `VK_EXT_metal_surface`

The `VK_MVK_macos_surface` extension is considered deprecated and has been superseded by the `VK_EXT_metal_surface` extension.

### New Commands

- vkCreateMacOSSurfaceMVK

### New Structures

- VkMacOSSurfaceCreateInfoMVK

### New Bitmasks

- VkMacOSSurfaceCreateFlagsMVK

### New Enum Constants

- `VK_MVK_MACOS_SURFACE_EXTENSION_NAME`
- `VK_MVK_MACOS_SURFACE_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_MACOS_SURFACE_CREATE_INFO_MVK`

### Version History

- Revision 1, 2017-02-15 (Bill Hollings)
  - Initial draft.
- Revision 2, 2017-02-24 (Bill Hollings)
  - Minor syntax fix to emphasize firm requirement for `NSView` to be backed by a `CAMetalLayer`.
- Revision 3, 2020-07-31 (Bill Hollings)
  - Update documentation on requirements for `NSView`.
  - Mark as deprecated by `VK_EXT_metal_surface`.

## VK_NV_compute_shader_derivatives

### Name String

`VK_NV_compute_shader_derivatives`

### Extension Type

Device extension

### Registered Extension Number

202

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

### SPIR-V Dependencies

- SPV_NV_compute_shader_derivatives

### Deprecation State

- *Promoted* to VK_KHR_compute_shader_derivatives extension

### Contact

- Pat Brown [GitHub]nvpbrown

### Other Extension Metadata

### Last Modified Date

2018-07-19

### IP Status

No known IP claims.

### Interactions and External Dependencies

- This extension provides API support for `GL_NV_compute_shader_derivatives`

### Contributors

- Pat Brown, NVIDIA

### Description

This extension adds Vulkan support for the `SPV_NV_compute_shader_derivatives` SPIR-V extension.

The SPIR-V extension provides two new execution modes, both of which allow compute shaders to use built-ins that evaluate compute derivatives explicitly or implicitly. Derivatives will be computed via differencing over a 2x2 group of shader invocations. The `DerivativeGroupQuadsNV` execution mode assembles shader invocations into 2x2 groups, where each group has x and y coordinates of the local invocation ID of the form (2m+{0,1}, 2n+{0,1}). The `DerivativeGroupLinearNV` execution mode assembles shader invocations into 2x2 groups, where each group has local invocation index values of the form 4m+{0,1,2,3}.

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceComputeShaderDerivativesFeaturesNV

### New Enum Constants

- `VK_NV_COMPUTE_SHADER_DERIVATIVES_EXTENSION_NAME`
- `VK_NV_COMPUTE_SHADER_DERIVATIVES_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_COMPUTE_SHADER_DERIVATIVES_FEATURES_NV`

### New SPIR-V Capability

- `ComputeDerivativeGroupQuadsNV`
- `ComputeDerivativeGroupLinearNV`

### Issues

(1) Should we specify that the groups of four shader invocations used for derivatives in a compute shader are the same groups of four invocations that form a “quad” in shader subgroups?

**RESOLVED**: Yes.

### Examples

None.

### Version History

- Revision 1, 2018-07-19 (Pat Brown)
  - Initial draft

## VK_NV_dedicated_allocation

### Name String

`VK_NV_dedicated_allocation`

### Extension Type

Device extension

### Registered Extension Number

27

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

None

### Deprecation State

- *Deprecated* by VK_KHR_dedicated_allocation extension
  - Which in turn was *promoted* to Vulkan 1.1

### Contact

- Jeff Bolz [GitHub]jeffbolznv

### Other Extension Metadata

### Last Modified Date

2016-05-31

### IP Status

No known IP claims.

### Contributors

- Jeff Bolz, NVIDIA

### Description

This extension allows device memory to be allocated for a particular buffer or image resource, which on some devices can significantly improve the performance of that resource. Normal device memory allocations must support memory aliasing and sparse binding, which could interfere with

optimizations like framebuffer compression or efficient page table usage. This is important for render targets and very large resources, but need not (and probably should not) be used for smaller resources that can benefit from suballocation.

This extension adds a few small structures to resource creation and memory allocation: a new structure that flags whether am image/buffer will have a dedicated allocation, and a structure indicating the image or buffer that an allocation will be bound to.

### New Structures

- Extending VkBufferCreateInfo:
  - VkDedicatedAllocationBufferCreateInfoNV
- Extending VkImageCreateInfo:
  - VkDedicatedAllocationImageCreateInfoNV
- Extending VkMemoryAllocateInfo:
  - VkDedicatedAllocationMemoryAllocateInfoNV

### New Enum Constants

- `VK_NV_DEDICATED_ALLOCATION_EXTENSION_NAME`
- `VK_NV_DEDICATED_ALLOCATION_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_DEDICATED_ALLOCATION_BUFFER_CREATE_INFO_NV`
  - `VK_STRUCTURE_TYPE_DEDICATED_ALLOCATION_IMAGE_CREATE_INFO_NV`
  - `VK_STRUCTURE_TYPE_DEDICATED_ALLOCATION_MEMORY_ALLOCATE_INFO_NV`

### Examples

```
// Create an image with
// VkDedicatedAllocationImageCreateInfoNV::dedicatedAllocation
// set to VK_TRUE

VkDedicatedAllocationImageCreateInfoNV dedicatedImageInfo =
{
    .sType = VK_STRUCTURE_TYPE_DEDICATED_ALLOCATION_IMAGE_CREATE_INFO_NV,
    .pNext = NULL,
    .dedicatedAllocation = VK_TRUE,
};

VkImageCreateInfo imageCreateInfo =
{
    .sType = VK_STRUCTURE_TYPE_IMAGE_CREATE_INFO,
    .pNext = &dedicatedImageInfo
    // Other members set as usual
};
```

```
    VkImage image;
    VkResult result = vkCreateImage(
        device,
        &imageCreateInfo,
        NULL,               // pAllocator
        &image);

    VkMemoryRequirements memoryRequirements;
    vkGetImageMemoryRequirements(
        device,
        image,
        &memoryRequirements);

    // Allocate memory with VkDedicatedAllocationMemoryAllocateInfoNV::image
    // pointing to the image we are allocating the memory for

    VkDedicatedAllocationMemoryAllocateInfoNV dedicatedInfo =
    {
        .sType = VK_STRUCTURE_TYPE_DEDICATED_ALLOCATION_MEMORY_ALLOCATE_INFO_NV,
        .pNext = NULL,
        .image = image,
        .buffer = VK_NULL_HANDLE,
    };

    VkMemoryAllocateInfo memoryAllocateInfo =
    {
        .sType = VK_STRUCTURE_TYPE_MEMORY_ALLOCATE_INFO,
        .pNext = &dedicatedInfo,
        .allocationSize = memoryRequirements.size,
        .memoryTypeIndex = FindMemoryTypeIndex(memoryRequirements.memoryTypeBits),
    };

    VkDeviceMemory memory;
    vkAllocateMemory(
        device,
        &memoryAllocateInfo,
        NULL,               // pAllocator
        &memory);

    // Bind the image to the memory

    vkBindImageMemory(
        device,
        image,
        memory,
        0);
```

### Version History

- Revision 1, 2016-05-31 (Jeff Bolz)

- Internal revisions

## VK_NV_external_memory

### Name String

`VK_NV_external_memory`

### Extension Type

Device extension

### Registered Extension Number

57

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_NV_external_memory_capabilities

### Deprecation State

- *Deprecated* by VK_KHR_external_memory extension
  - Which in turn was *promoted* to Vulkan 1.1

### Contact

- James Jones [GitHub]cubanismo

### Other Extension Metadata

### Last Modified Date

2016-08-19

### IP Status

No known IP claims.

### Contributors

- James Jones, NVIDIA
- Carsten Rohde, NVIDIA

### Description

Applications may wish to export memory to other Vulkan instances or other APIs, or import memory from other Vulkan instances or other APIs to enable Vulkan workloads to be split up across application module, process, or API boundaries. This extension enables applications to create exportable Vulkan memory objects such that the underlying resources can be referenced outside the Vulkan instance that created them.

### New Structures

- Extending VkImageCreateInfo:
  - VkExternalMemoryImageCreateInfoNV
- Extending VkMemoryAllocateInfo:
  - VkExportMemoryAllocateInfoNV

### New Enum Constants

- `VK_NV_EXTERNAL_MEMORY_EXTENSION_NAME`
- `VK_NV_EXTERNAL_MEMORY_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_EXPORT_MEMORY_ALLOCATE_INFO_NV`
  - `VK_STRUCTURE_TYPE_EXTERNAL_MEMORY_IMAGE_CREATE_INFO_NV`

### Issues

1) If memory objects are shared between processes and APIs, is this considered aliasing according to the rules outlined in the Memory Aliasing section?

**RESOLVED**: Yes, but strict exceptions to the rules are added to allow some forms of aliasing in these cases. Further, other extensions may build upon these new aliasing rules to define specific support usage within Vulkan for imported native memory objects, or memory objects from other APIs.

2) Are new image layouts or metadata required to specify image layouts and layout transitions compatible with non-Vulkan APIs, or with other instances of the same Vulkan driver?

**RESOLVED**: No. Separate instances of the same Vulkan driver running on the same GPU should have identical internal layout semantics, so applications have the tools they need to ensure views of images are consistent between the two instances. Other APIs will fall into two categories: Those that are Vulkan compatible (a term to be defined by subsequent interopability extensions), or Vulkan incompatible. When sharing images with Vulkan incompatible APIs, the Vulkan image must be transitioned to the `VK_IMAGE_LAYOUT_GENERAL` layout before handing it off to the external API.

Note this does not attempt to address cross-device transitions, nor transitions to engines on the same device which are not visible within the Vulkan API. Both of these are beyond the scope of this extension.

### Examples

```
// TODO: Write some sample code here.
```

### Version History

- Revision 1, 2016-08-19 (James Jones)
  - Initial draft

## VK_NV_external_memory_capabilities

### Name String

`VK_NV_external_memory_capabilities`

### Extension Type

Instance extension

### Registered Extension Number

56

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

None

### Deprecation State

- *Deprecated* by VK_KHR_external_memory_capabilities extension
  - Which in turn was *promoted* to Vulkan 1.1

### Contact

- James Jones [GitHub]cubanismo

### Other Extension Metadata

### Last Modified Date

2016-08-19

### IP Status

No known IP claims.

### Interactions and External Dependencies

- Interacts with Vulkan 1.1.
- Interacts with `VK_KHR_dedicated_allocation`.
- Interacts with `VK_NV_dedicated_allocation`.

### Contributors

- James Jones, NVIDIA

### Description

Applications may wish to import memory from the Direct 3D API, or export memory to other Vulkan instances. This extension provides a set of capability queries that allow applications determine what types of win32 memory handles an implementation supports for a given set of use

cases.

### New Commands

- vkGetPhysicalDeviceExternalImageFormatPropertiesNV

### New Structures

- VkExternalImageFormatPropertiesNV

### New Enums

- VkExternalMemoryFeatureFlagBitsNV
- VkExternalMemoryHandleTypeFlagBitsNV

### New Bitmasks

- VkExternalMemoryFeatureFlagsNV
- VkExternalMemoryHandleTypeFlagsNV

### New Enum Constants

- `VK_NV_EXTERNAL_MEMORY_CAPABILITIES_EXTENSION_NAME`
- `VK_NV_EXTERNAL_MEMORY_CAPABILITIES_SPEC_VERSION`

### Issues

1) Why do so many external memory capabilities need to be queried on a per-memory-handle-type basis?

**RESOLVED**: This is because some handle types are based on OS-native objects that have far more limited capabilities than the very generic Vulkan memory objects. Not all memory handle types can name memory objects that support 3D images, for example. Some handle types cannot even support the deferred image and memory binding behavior of Vulkan and require specifying the image when allocating or importing the memory object.

2) Does the VkExternalImageFormatPropertiesNV struct need to include a list of memory type bits that support the given handle type?

**RESOLVED**: No. The memory types that do not support the handle types will simply be filtered out of the results returned by vkGetImageMemoryRequirements when a set of handle types was specified at image creation time.

3) Should the non-opaque handle types be moved to their own extension?

**RESOLVED**: Perhaps. However, defining the handle type bits does very little and does not require any platform-specific types on its own, and it is easier to maintain the bitmask values in a single extension for now. Presumably more handle types could be added by separate extensions though, and it would be midly weird to have some platform-specific ones defined in the core spec and some in extensions

### Version History

- Revision 1, 2016-08-19 (James Jones)
  - Initial version

## VK_NV_external_memory_win32

### Name String

`VK_NV_external_memory_win32`

### Extension Type

Device extension

### Registered Extension Number

58

### Revision

1

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_NV_external_memory

### Deprecation State

- *Deprecated* by VK_KHR_external_memory_win32 extension

### Contact

- James Jones [GitHub]cubanismo

### Other Extension Metadata

### Last Modified Date

2016-08-19

### IP Status

No known IP claims.

### Contributors

- James Jones, NVIDIA
- Carsten Rohde, NVIDIA

### Description

Applications may wish to export memory to other Vulkan instances or other APIs, or import memory from other Vulkan instances or other APIs to enable Vulkan workloads to be split up across application module, process, or API boundaries. This extension enables win32 applications

to export win32 handles from Vulkan memory objects such that the underlying resources can be referenced outside the Vulkan instance that created them, and import win32 handles created in the Direct3D API to Vulkan memory objects.

### New Commands

- vkGetMemoryWin32HandleNV

### New Structures

- Extending VkMemoryAllocateInfo:
  - VkExportMemoryWin32HandleInfoNV
  - VkImportMemoryWin32HandleInfoNV

### New Enum Constants

- `VK_NV_EXTERNAL_MEMORY_WIN32_EXTENSION_NAME`
- `VK_NV_EXTERNAL_MEMORY_WIN32_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_EXPORT_MEMORY_WIN32_HANDLE_INFO_NV`
  - `VK_STRUCTURE_TYPE_IMPORT_MEMORY_WIN32_HANDLE_INFO_NV`

### Issues

1) If memory objects are shared between processes and APIs, is this considered aliasing according to the rules outlined in the Memory Aliasing section?

**RESOLVED**: Yes, but strict exceptions to the rules are added to allow some forms of aliasing in these cases. Further, other extensions may build upon these new aliasing rules to define specific support usage within Vulkan for imported native memory objects, or memory objects from other APIs.

2) Are new image layouts or metadata required to specify image layouts and layout transitions compatible with non-Vulkan APIs, or with other instances of the same Vulkan driver?

**RESOLVED**: No. Separate instances of the same Vulkan driver running on the same GPU should have identical internal layout semantics, so applications have the tools they need to ensure views of images are consistent between the two instances. Other APIs will fall into two categories: Those that are Vulkan compatible (a term to be defined by subsequent interopability extensions), or Vulkan incompatible. When sharing images with Vulkan incompatible APIs, the Vulkan image must be transitioned to the `VK_IMAGE_LAYOUT_GENERAL` layout before handing it off to the external API.

Note this does not attempt to address cross-device transitions, nor transitions to engines on the same device which are not visible within the Vulkan API. Both of these are beyond the scope of this extension.

3) Do applications need to call `CloseHandle`() on the values returned from vkGetMemoryWin32HandleNV when `handleType` is `VK_EXTERNAL_MEMORY_HANDLE_TYPE_OPAQUE_WIN32_BIT_NV`?

**RESOLVED**: Yes, unless it is passed back in to another driver instance to import the object. A successful get call transfers ownership of the handle to the application, while an import transfers ownership to the associated driver. Destroying the memory object will not destroy the handle or the handle's reference to the underlying memory resource.

### Examples

```
//
// Create an exportable memory object and export an external
// handle from it.
//

// Pick an external format and handle type.
static const VkFormat format = VK_FORMAT_R8G8B8A8_UNORM;
static const VkExternalMemoryHandleTypeFlagsNV handleType =
    VK_EXTERNAL_MEMORY_HANDLE_TYPE_OPAQUE_WIN32_BIT_NV;

extern VkPhysicalDevice physicalDevice;
extern VkDevice device;

VkPhysicalDeviceMemoryProperties memoryProperties;
VkExternalImageFormatPropertiesNV properties;
VkExternalMemoryImageCreateInfoNV externalMemoryImageCreateInfo;
VkDedicatedAllocationImageCreateInfoNV dedicatedImageCreateInfo;
VkImageCreateInfo imageCreateInfo;
VkImage image;
VkMemoryRequirements imageMemoryRequirements;
uint32_t numMemoryTypes;
uint32_t memoryType;
VkExportMemoryAllocateInfoNV exportMemoryAllocateInfo;
VkDedicatedAllocationMemoryAllocateInfoNV dedicatedAllocationInfo;
VkMemoryAllocateInfo memoryAllocateInfo;
VkDeviceMemory memory;
VkResult result;
HANDLE memoryHnd;

// Figure out how many memory types the device supports
vkGetPhysicalDeviceMemoryProperties(physicalDevice,
                                    &memoryProperties);
numMemoryTypes = memoryProperties.memoryTypeCount;

// Check the external handle type capabilities for the chosen format
// Exportable 2D image support with at least 1 mip level, 1 array
// layer, and VK_SAMPLE_COUNT_1_BIT using optimal tiling and supporting
// texturing and color rendering is required.
result = vkGetPhysicalDeviceExternalImageFormatPropertiesNV(
    physicalDevice,
    format,
    VK_IMAGE_TYPE_2D,
    VK_IMAGE_TILING_OPTIMAL,
```

```
        VK_IMAGE_USAGE_SAMPLED_BIT |
        VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT,
        0,
        handleType,
        &properties);

    if ((result != VK_SUCCESS) ||
        !(properties.externalMemoryFeatures &
          VK_EXTERNAL_MEMORY_FEATURE_EXPORTABLE_BIT_NV)) {
        abort();
    }

    // Set up the external memory image creation info
    memset(&externalMemoryImageCreateInfo,
           0, sizeof(externalMemoryImageCreateInfo));
    externalMemoryImageCreateInfo.sType =
        VK_STRUCTURE_TYPE_EXTERNAL_MEMORY_IMAGE_CREATE_INFO_NV;
    externalMemoryImageCreateInfo.handleTypes = handleType;
    if (properties.externalMemoryFeatures &
        VK_EXTERNAL_MEMORY_FEATURE_DEDICATED_ONLY_BIT_NV) {
        memset(&dedicatedImageCreateInfo, 0, sizeof(dedicatedImageCreateInfo));
        dedicatedImageCreateInfo.sType =
            VK_STRUCTURE_TYPE_DEDICATED_ALLOCATION_IMAGE_CREATE_INFO_NV;
        dedicatedImageCreateInfo.dedicatedAllocation = VK_TRUE;
        externalMemoryImageCreateInfo.pNext = &dedicatedImageCreateInfo;
    }
    // Set up the  core image creation info
    memset(&imageCreateInfo, 0, sizeof(imageCreateInfo));
    imageCreateInfo.sType = VK_STRUCTURE_TYPE_IMAGE_CREATE_INFO;
    imageCreateInfo.pNext = &externalMemoryImageCreateInfo;
    imageCreateInfo.format = format;
    imageCreateInfo.extent.width = 64;
    imageCreateInfo.extent.height = 64;
    imageCreateInfo.extent.depth = 1;
    imageCreateInfo.mipLevels = 1;
    imageCreateInfo.arrayLayers = 1;
    imageCreateInfo.samples = VK_SAMPLE_COUNT_1_BIT;
    imageCreateInfo.tiling = VK_IMAGE_TILING_OPTIMAL;
    imageCreateInfo.usage = VK_IMAGE_USAGE_SAMPLED_BIT |
        VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT;
    imageCreateInfo.sharingMode = VK_SHARING_MODE_EXCLUSIVE;
    imageCreateInfo.initialLayout = VK_IMAGE_LAYOUT_UNDEFINED;

    vkCreateImage(device, &imageCreateInfo, NULL, &image);

    vkGetImageMemoryRequirements(device,
                                 image,
                                 &imageMemoryRequirements);

    // For simplicity, just pick the first compatible memory type.
    for (memoryType = 0; memoryType < numMemoryTypes; memoryType++) {
```

```
        if ((1 << memoryType) & imageMemoryRequirements.memoryTypeBits) {
            break;
        }
    }

    // At least one memory type must be supported given the prior external
    // handle capability check.
    assert(memoryType < numMemoryTypes);

    // Allocate the external memory object.
    memset(&exportMemoryAllocateInfo, 0, sizeof(exportMemoryAllocateInfo));
    exportMemoryAllocateInfo.sType =
        VK_STRUCTURE_TYPE_EXPORT_MEMORY_ALLOCATE_INFO_NV;
    exportMemoryAllocateInfo.handleTypes = handleType;
    if (properties.externalMemoryFeatures &
        VK_EXTERNAL_MEMORY_FEATURE_DEDICATED_ONLY_BIT_NV) {
        memset(&dedicatedAllocationInfo, 0, sizeof(dedicatedAllocationInfo));
        dedicatedAllocationInfo.sType =
            VK_STRUCTURE_TYPE_DEDICATED_ALLOCATION_MEMORY_ALLOCATE_INFO_NV;
        dedicatedAllocationInfo.image = image;
        exportMemoryAllocateInfo.pNext = &dedicatedAllocationInfo;
    }
    memset(&memoryAllocateInfo, 0, sizeof(memoryAllocateInfo));
    memoryAllocateInfo.sType = VK_STRUCTURE_TYPE_MEMORY_ALLOCATE_INFO;
    memoryAllocateInfo.pNext = &exportMemoryAllocateInfo;
    memoryAllocateInfo.allocationSize = imageMemoryRequirements.size;
    memoryAllocateInfo.memoryTypeIndex = memoryType;

    vkAllocateMemory(device, &memoryAllocateInfo, NULL, &memory);

    if (!(properties.externalMemoryFeatures &
          VK_EXTERNAL_MEMORY_FEATURE_DEDICATED_ONLY_BIT_NV)) {
        vkBindImageMemory(device, image, memory, 0);
    }

    // Get the external memory opaque FD handle
    vkGetMemoryWin32HandleNV(device, memory, &memoryHnd);
```

### Version History

- Revision 1, 2016-08-11 (James Jones)
  - Initial draft

## VK_NV_fragment_shader_barycentric

### Name String

`VK_NV_fragment_shader_barycentric`

**Extension Type**

Device extension

**Registered Extension Number**

204

**Revision**

1

**Ratification Status**

Not ratified

**Extension and Version Dependencies**

VK_KHR_get_physical_device_properties2
or
Vulkan Version 1.1

**SPIR-V Dependencies**

- SPV_NV_fragment_shader_barycentric

**Deprecation State**

- *Promoted* to VK_KHR_fragment_shader_barycentric extension

**Contact**

- Pat Brown [GitHub]nvpbrown

**Other Extension Metadata**

**Last Modified Date**

2018-08-03

**IP Status**

No known IP claims.

**Interactions and External Dependencies**

- This extension provides API support for `GL_NV_fragment_shader_barycentric`

**Contributors**

- Pat Brown, NVIDIA
- Daniel Koch, NVIDIA

**Description**

This extension adds support for the following SPIR-V extension in Vulkan:

- `SPV_NV_fragment_shader_barycentric`

The extension provides access to three additional fragment shader variable decorations in SPIR-V:

- `PerVertexNV`, which indicates that a fragment shader input will not have interpolated values, but

instead must be accessed with an extra array index that identifies one of the vertices of the primitive producing the fragment

- `BaryCoordNV`, which indicates that the variable is a three-component floating-point vector holding barycentric weights for the fragment produced using perspective interpolation
- `BaryCoordNoPerspNV`, which indicates that the variable is a three-component floating-point vector holding barycentric weights for the fragment produced using linear interpolation

When using GLSL source-based shader languages, the following variables from `GL_NV_fragment_shader_barycentric` maps to these SPIR-V built-in decorations:

- `in vec3 gl_BaryCoordNV;` → `BaryCoordNV`
- `in vec3 gl_BaryCoordNoPerspNV;` → `BaryCoordNoPerspNV`

GLSL variables declared using the `__pervertexNV` GLSL qualifier are expected to be decorated with `PerVertexNV` in SPIR-V.

### Promotion to `VK_KHR_fragment_shader_barycentric`

All functionality in this extension is included in `VK_KHR_fragment_shader_barycentric`, with the suffix changed to KHR.

### New Structures

- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceFragmentShaderBarycentricFeaturesNV

### New Enum Constants

- `VK_NV_FRAGMENT_SHADER_BARYCENTRIC_EXTENSION_NAME`
- `VK_NV_FRAGMENT_SHADER_BARYCENTRIC_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_FRAGMENT_SHADER_BARYCENTRIC_FEATURES_NV`

### New Built-In Variables

- `BaryCoordNV`
- `BaryCoordNoPerspNV`

### New SPIR-V Decorations

- `PerVertexNV`

### New SPIR-V Capabilities

- `FragmentBarycentricNV`

### Issues

(1) The AMD_shader_explicit_vertex_parameter extension provides similar functionality. Why write a new extension, and how is this extension different?

**RESOLVED**: For the purposes of Vulkan/SPIR-V, we chose to implement a separate extension due to several functional differences.

First, the hardware supporting this extension can provide a three-component barycentric weight vector for variables decorated with `BaryCoordNV`, while variables decorated with `BaryCoordSmoothAMD` provide only two components. In some cases, it may be more efficient to explicitly interpolate an attribute via:

```
float value = (baryCoordNV.x * v[0].attrib +
               baryCoordNV.y * v[1].attrib +
               baryCoordNV.z * v[2].attrib);
```

instead of

```
float value = (baryCoordSmoothAMD.x * (v[0].attrib - v[2].attrib) +
               baryCoordSmoothAMD.y * (v[1].attrib - v[2].attrib) +
               v[2].attrib);
```

Additionally, the semantics of the decoration `BaryCoordPullModelAMD` do not appear to map to anything supported by the initial hardware implementation of this extension.

This extension provides a smaller number of decorations than the AMD extension, as we expect that shaders could derive variables decorated with things like `BaryCoordNoPerspCentroidAMD` with explicit attribute interpolation instructions. One other relevant difference is that explicit per-vertex attribute access using this extension does not require a constant vertex number.

(2) Why do the built-in SPIR-V decorations for this extension include two separate built-ins `BaryCoordNV` and `BaryCoordNoPerspNV` when a “no perspective” variable could be decorated with `BaryCoordNV` and `NoPerspective`?

**RESOLVED**: The SPIR-V extension for this feature chose to mirror the behavior of the GLSL extension, which provides two built-in variables. Additionally, it is not clear that its a good idea (or even legal) to have two variables using the “same attribute”, but with different interpolation modifiers.

### Version History

- Revision 1, 2018-08-03 (Pat Brown)
  - Internal revisions

## VK_NV_glsl_shader

**Name String**

`VK_NV_glsl_shader`

**Extension Type**

Device extension

**Registered Extension Number**

13

**Revision**

1

**Ratification Status**

Not ratified

**Extension and Version Dependencies**

None

**Deprecation State**

- *Deprecated* without replacement

**Contact**

- Piers Daniell [GitHub]pdaniell-nv

**Other Extension Metadata**

**Last Modified Date**

2016-02-14

**IP Status**

No known IP claims.

**Contributors**

- Piers Daniell, NVIDIA

**Description**

This extension allows GLSL shaders written to the `GL_KHR_vulkan_glsl` extension specification to be used instead of SPIR-V. The implementation will automatically detect whether the shader is SPIR-V or GLSL, and compile it appropriately.

**Deprecation**

Functionality in this extension is outside of the scope of Vulkan and is better served by a compiler library such as glslang. No new implementations will support this extension, so applications **should** not use it.

### New Enum Constants

- `VK_NV_GLSL_SHADER_EXTENSION_NAME`
- `VK_NV_GLSL_SHADER_SPEC_VERSION`
- Extending VkResult:
  - `VK_ERROR_INVALID_SHADER_NV`

### Examples

#### Example 1

Passing in GLSL code

```
    char const vss[] =
        "#version 450 core\n"
        "layout(location = 0) in vec2 aVertex;\n"
        "layout(location = 1) in vec4 aColor;\n"
        "out vec4 vColor;\n"
        "void main()\n"
        "{\n"
        "    vColor = aColor;\n"
        "    gl_Position = vec4(aVertex, 0, 1);\n"
        "}\n"
    ;
    VkShaderModuleCreateInfo vertexShaderInfo = {
VK_STRUCTURE_TYPE_SHADER_MODULE_CREATE_INFO };
    vertexShaderInfo.codeSize = sizeof vss;
    vertexShaderInfo.pCode = vss;
    VkShaderModule vertexShader;
    vkCreateShaderModule(device, &vertexShaderInfo, 0, &vertexShader);
```

### Version History

- Revision 1, 2016-02-14 (Piers Daniell)
  - Initial draft

## VK_NV_ray_tracing

### Name String

`VK_NV_ray_tracing`

### Extension Type

Device extension

### Registered Extension Number

166

### Revision

3

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_KHR_get_physical_device_properties2
and
VK_KHR_get_memory_requirements2
or
Vulkan Version 1.1

### API Interactions

- Interacts with VK_VERSION_1_1
- Interacts with VK_EXT_debug_report
- Interacts with VK_KHR_get_memory_requirements2

### SPIR-V Dependencies

- SPV_NV_ray_tracing

### Deprecation State

- *Deprecated* by VK_KHR_ray_tracing_pipeline extension

### Contact

- Eric Werness [GitHub]ewerness-nv

## Other Extension Metadata

### Last Modified Date

2018-11-20

### Interactions and External Dependencies

- This extension provides API support for `GL_NV_ray_tracing`

### Contributors

- Eric Werness, NVIDIA
- Ashwin Lele, NVIDIA
- Robert Stepinski, NVIDIA
- Nuno Subtil, NVIDIA
- Christoph Kubisch, NVIDIA
- Martin Stich, NVIDIA
- Daniel Koch, NVIDIA
- Jeff Bolz, NVIDIA
- Joshua Barczak, Intel

- Tobias Hector, AMD
- Henrik Rydgard, NVIDIA
- Pascal Gautron, NVIDIA

**Description**

Rasterization has been the dominant method to produce interactive graphics, but increasing performance of graphics hardware has made ray tracing a viable option for interactive rendering. Being able to integrate ray tracing with traditional rasterization makes it easier for applications to incrementally add ray traced effects to existing applications or to do hybrid approaches with rasterization for primary visibility and ray tracing for secondary queries.

To enable ray tracing, this extension adds a few different categories of new functionality:

- Acceleration structure objects and build commands
- A new pipeline type with new shader domains
- An indirection table to link shader groups with acceleration structure items

This extension adds support for the following SPIR-V extension in Vulkan:

- `SPV_NV_ray_tracing`

**New Object Types**

- VkAccelerationStructureNV

**New Commands**

- vkBindAccelerationStructureMemoryNV
- vkCmdBuildAccelerationStructureNV
- vkCmdCopyAccelerationStructureNV
- vkCmdTraceRaysNV
- vkCmdWriteAccelerationStructuresPropertiesNV
- vkCompileDeferredNV
- vkCreateAccelerationStructureNV
- vkCreateRayTracingPipelinesNV
- vkDestroyAccelerationStructureNV
- vkGetAccelerationStructureHandleNV
- vkGetAccelerationStructureMemoryRequirementsNV
- vkGetRayTracingShaderGroupHandlesNV

**New Structures**

- VkAabbPositionsNV

- VkAccelerationStructureCreateInfoNV
- VkAccelerationStructureInfoNV
- VkAccelerationStructureInstanceNV
- VkAccelerationStructureMemoryRequirementsInfoNV
- VkBindAccelerationStructureMemoryInfoNV
- VkGeometryAABBNV
- VkGeometryDataNV
- VkGeometryNV
- VkGeometryTrianglesNV
- VkRayTracingPipelineCreateInfoNV
- VkRayTracingShaderGroupCreateInfoNV
- VkTransformMatrixNV
- Extending VkPhysicalDeviceProperties2:
  - VkPhysicalDeviceRayTracingPropertiesNV
- Extending VkWriteDescriptorSet:
  - VkWriteDescriptorSetAccelerationStructureNV

If VK_KHR_get_memory_requirements2 or Vulkan Version 1.1 is supported:

- VkMemoryRequirements2KHR

#### New Enums

- VkAccelerationStructureMemoryRequirementsTypeNV
- VkAccelerationStructureTypeNV
- VkBuildAccelerationStructureFlagBitsNV
- VkCopyAccelerationStructureModeNV
- VkGeometryFlagBitsNV
- VkGeometryInstanceFlagBitsNV
- VkGeometryTypeNV
- VkRayTracingShaderGroupTypeNV

#### New Bitmasks

- VkBuildAccelerationStructureFlagsNV
- VkGeometryFlagsNV
- VkGeometryInstanceFlagsNV

### New Enum Constants

- VK_NV_RAY_TRACING_EXTENSION_NAME
- VK_NV_RAY_TRACING_SPEC_VERSION
- VK_SHADER_UNUSED_NV
- Extending VkAccelerationStructureTypeKHR:
  - VK_ACCELERATION_STRUCTURE_TYPE_BOTTOM_LEVEL_NV
  - VK_ACCELERATION_STRUCTURE_TYPE_TOP_LEVEL_NV
- Extending VkAccessFlagBits:
  - VK_ACCESS_ACCELERATION_STRUCTURE_READ_BIT_NV
  - VK_ACCESS_ACCELERATION_STRUCTURE_WRITE_BIT_NV
- Extending VkBufferUsageFlagBits:
  - VK_BUFFER_USAGE_RAY_TRACING_BIT_NV
- Extending VkBuildAccelerationStructureFlagBitsKHR:
  - VK_BUILD_ACCELERATION_STRUCTURE_ALLOW_COMPACTION_BIT_NV
  - VK_BUILD_ACCELERATION_STRUCTURE_ALLOW_UPDATE_BIT_NV
  - VK_BUILD_ACCELERATION_STRUCTURE_LOW_MEMORY_BIT_NV
  - VK_BUILD_ACCELERATION_STRUCTURE_PREFER_FAST_BUILD_BIT_NV
  - VK_BUILD_ACCELERATION_STRUCTURE_PREFER_FAST_TRACE_BIT_NV
- Extending VkCopyAccelerationStructureModeKHR:
  - VK_COPY_ACCELERATION_STRUCTURE_MODE_CLONE_NV
  - VK_COPY_ACCELERATION_STRUCTURE_MODE_COMPACT_NV
- Extending VkDescriptorType:
  - VK_DESCRIPTOR_TYPE_ACCELERATION_STRUCTURE_NV
- Extending VkGeometryFlagBitsKHR:
  - VK_GEOMETRY_NO_DUPLICATE_ANY_HIT_INVOCATION_BIT_NV
  - VK_GEOMETRY_OPAQUE_BIT_NV
- Extending VkGeometryInstanceFlagBitsKHR:
  - VK_GEOMETRY_INSTANCE_FORCE_NO_OPAQUE_BIT_NV
  - VK_GEOMETRY_INSTANCE_FORCE_OPAQUE_BIT_NV
  - VK_GEOMETRY_INSTANCE_TRIANGLE_CULL_DISABLE_BIT_NV
  - VK_GEOMETRY_INSTANCE_TRIANGLE_FRONT_COUNTERCLOCKWISE_BIT_NV
- Extending VkGeometryTypeKHR:
  - VK_GEOMETRY_TYPE_AABBS_NV
  - VK_GEOMETRY_TYPE_TRIANGLES_NV

- Extending VkIndexType:
  - VK_INDEX_TYPE_NONE_NV
- Extending VkObjectType:
  - VK_OBJECT_TYPE_ACCELERATION_STRUCTURE_NV
- Extending VkPipelineBindPoint:
  - VK_PIPELINE_BIND_POINT_RAY_TRACING_NV
- Extending VkPipelineCreateFlagBits:
  - VK_PIPELINE_CREATE_DEFER_COMPILE_BIT_NV
- Extending VkPipelineStageFlagBits:
  - VK_PIPELINE_STAGE_ACCELERATION_STRUCTURE_BUILD_BIT_NV
  - VK_PIPELINE_STAGE_RAY_TRACING_SHADER_BIT_NV
- Extending VkQueryType:
  - VK_QUERY_TYPE_ACCELERATION_STRUCTURE_COMPACTED_SIZE_NV
- Extending VkRayTracingShaderGroupTypeKHR:
  - VK_RAY_TRACING_SHADER_GROUP_TYPE_GENERAL_NV
  - VK_RAY_TRACING_SHADER_GROUP_TYPE_PROCEDURAL_HIT_GROUP_NV
  - VK_RAY_TRACING_SHADER_GROUP_TYPE_TRIANGLES_HIT_GROUP_NV
- Extending VkShaderStageFlagBits:
  - VK_SHADER_STAGE_ANY_HIT_BIT_NV
  - VK_SHADER_STAGE_CALLABLE_BIT_NV
  - VK_SHADER_STAGE_CLOSEST_HIT_BIT_NV
  - VK_SHADER_STAGE_INTERSECTION_BIT_NV
  - VK_SHADER_STAGE_MISS_BIT_NV
  - VK_SHADER_STAGE_RAYGEN_BIT_NV
- Extending VkStructureType:
  - VK_STRUCTURE_TYPE_ACCELERATION_STRUCTURE_CREATE_INFO_NV
  - VK_STRUCTURE_TYPE_ACCELERATION_STRUCTURE_INFO_NV
  - VK_STRUCTURE_TYPE_ACCELERATION_STRUCTURE_MEMORY_REQUIREMENTS_INFO_NV
  - VK_STRUCTURE_TYPE_BIND_ACCELERATION_STRUCTURE_MEMORY_INFO_NV
  - VK_STRUCTURE_TYPE_GEOMETRY_AABB_NV
  - VK_STRUCTURE_TYPE_GEOMETRY_NV
  - VK_STRUCTURE_TYPE_GEOMETRY_TRIANGLES_NV
  - VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_RAY_TRACING_PROPERTIES_NV
  - VK_STRUCTURE_TYPE_RAY_TRACING_PIPELINE_CREATE_INFO_NV
  - VK_STRUCTURE_TYPE_RAY_TRACING_SHADER_GROUP_CREATE_INFO_NV

- `VK_STRUCTURE_TYPE_WRITE_DESCRIPTOR_SET_ACCELERATION_STRUCTURE_NV`

If VK_EXT_debug_report is supported:

- Extending VkDebugReportObjectTypeEXT:
  - `VK_DEBUG_REPORT_OBJECT_TYPE_ACCELERATION_STRUCTURE_NV_EXT`

#### New or Modified Built-In Variables

- `LaunchIdNV`
- `LaunchSizeNV`
- `WorldRayOriginNV`
- `WorldRayDirectionNV`
- `ObjectRayOriginNV`
- `ObjectRayDirectionNV`
- `RayTminNV`
- `RayTmaxNV`
- `InstanceCustomIndexNV`
- `InstanceId`
- `ObjectToWorldNV`
- `WorldToObjectNV`
- `HitTNV`
- `HitKindNV`
- `IncomingRayFlagsNV`
- (modified)`PrimitiveId`

#### New SPIR-V Capabilities

- `RayTracingNV`

#### Issues

1) Are there issues?

**RESOLVED**: Yes.

#### Sample Code

Example ray generation GLSL shader

```
#version 450 core
#extension GL_NV_ray_tracing : require
layout(set = 0, binding = 0, rgba8) uniform image2D image;
```

```
layout(set = 0, binding = 1) uniform accelerationStructureNV as;
layout(location = 0) rayPayloadNV float payload;

void main()
{
   vec4 col = vec4(0, 0, 0, 1);

   vec3 origin = vec3(float(gl_LaunchIDNV.x)/float(gl_LaunchSizeNV.x), float
(gl_LaunchIDNV.y)/float(gl_LaunchSizeNV.y), 1.0);
   vec3 dir = vec3(0.0, 0.0, -1.0);

   traceNV(as, 0, 0xff, 0, 1, 0, origin, 0.0, dir, 1000.0, 0);

   col.y = payload;

   imageStore(image, ivec2(gl_LaunchIDNV.xy), col);
}
```

### Version History

- Revision 1, 2018-09-11 (Robert Stepinski, Nuno Subtil, Eric Werness)
  - Internal revisions
- Revision 2, 2018-10-19 (Eric Werness)
  - rename to VK_NV_ray_tracing, add support for callables.
  - too many updates to list
- Revision 3, 2018-11-20 (Daniel Koch)
  - update to use InstanceId instead of InstanceIndex as implemented.

## VK_NV_win32_keyed_mutex

### Name String

`VK_NV_win32_keyed_mutex`

### Extension Type

Device extension

### Registered Extension Number

59

### Revision

2

### Ratification Status

Not ratified

### Extension and Version Dependencies

VK_NV_external_memory_win32

### Deprecation State

- *Promoted* to VK_KHR_win32_keyed_mutex extension

### Contact

- Carsten Rohde [GitHub]crohde

### Other Extension Metadata

### Last Modified Date

2016-08-19

### IP Status

No known IP claims.

### Contributors

- James Jones, NVIDIA
- Carsten Rohde, NVIDIA

### Description

Applications that wish to import Direct3D 11 memory objects into the Vulkan API may wish to use the native keyed mutex mechanism to synchronize access to the memory between Vulkan and Direct3D. This extension provides a way for an application to access the keyed mutex associated with an imported Vulkan memory object when submitting command buffers to a queue.

### New Structures

- Extending VkSubmitInfo, VkSubmitInfo2:
  - VkWin32KeyedMutexAcquireReleaseInfoNV

### New Enum Constants

- `VK_NV_WIN32_KEYED_MUTEX_EXTENSION_NAME`
- `VK_NV_WIN32_KEYED_MUTEX_SPEC_VERSION`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_WIN32_KEYED_MUTEX_ACQUIRE_RELEASE_INFO_NV`

### Examples

```
//
// Import a memory object from Direct3D 11, and synchronize
// access to it in Vulkan using keyed mutex objects.
//
```

```
    extern VkPhysicalDevice physicalDevice;
    extern VkDevice device;
    extern HANDLE sharedNtHandle;

    static const VkFormat format = VK_FORMAT_R8G8B8A8_UNORM;
    static const VkExternalMemoryHandleTypeFlagsNV handleType =
        VK_EXTERNAL_MEMORY_HANDLE_TYPE_D3D11_IMAGE_BIT_NV;

    VkPhysicalDeviceMemoryProperties memoryProperties;
    VkExternalImageFormatPropertiesNV properties;
    VkExternalMemoryImageCreateInfoNV externalMemoryImageCreateInfo;
    VkImageCreateInfo imageCreateInfo;
    VkImage image;
    VkMemoryRequirements imageMemoryRequirements;
    uint32_t numMemoryTypes;
    uint32_t memoryType;
    VkImportMemoryWin32HandleInfoNV importMemoryInfo;
    VkMemoryAllocateInfo memoryAllocateInfo;
    VkDeviceMemory mem;
    VkResult result;

    // Figure out how many memory types the device supports
    vkGetPhysicalDeviceMemoryProperties(physicalDevice,
                                        &memoryProperties);
    numMemoryTypes = memoryProperties.memoryTypeCount;

    // Check the external handle type capabilities for the chosen format
    // Importable 2D image support with at least 1 mip level, 1 array
    // layer, and VK_SAMPLE_COUNT_1_BIT using optimal tiling and supporting
    // texturing and color rendering is required.
    result = vkGetPhysicalDeviceExternalImageFormatPropertiesNV(
        physicalDevice,
        format,
        VK_IMAGE_TYPE_2D,
        VK_IMAGE_TILING_OPTIMAL,
        VK_IMAGE_USAGE_SAMPLED_BIT |
        VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT,
        0,
        handleType,
        &properties);

    if ((result != VK_SUCCESS) ||
        !(properties.externalMemoryFeatures &
          VK_EXTERNAL_MEMORY_FEATURE_IMPORTABLE_BIT_NV)) {
        abort();
    }

    // Set up the external memory image creation info
    memset(&externalMemoryImageCreateInfo,
           0, sizeof(externalMemoryImageCreateInfo));
    externalMemoryImageCreateInfo.sType =
```

```
        VK_STRUCTURE_TYPE_EXTERNAL_MEMORY_IMAGE_CREATE_INFO_NV;
    externalMemoryImageCreateInfo.handleTypes = handleType;
    // Set up the  core image creation info
    memset(&imageCreateInfo, 0, sizeof(imageCreateInfo));
    imageCreateInfo.sType = VK_STRUCTURE_TYPE_IMAGE_CREATE_INFO;
    imageCreateInfo.pNext = &externalMemoryImageCreateInfo;
    imageCreateInfo.format = format;
    imageCreateInfo.extent.width = 64;
    imageCreateInfo.extent.height = 64;
    imageCreateInfo.extent.depth = 1;
    imageCreateInfo.mipLevels = 1;
    imageCreateInfo.arrayLayers = 1;
    imageCreateInfo.samples = VK_SAMPLE_COUNT_1_BIT;
    imageCreateInfo.tiling = VK_IMAGE_TILING_OPTIMAL;
    imageCreateInfo.usage = VK_IMAGE_USAGE_SAMPLED_BIT |
        VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT;
    imageCreateInfo.sharingMode = VK_SHARING_MODE_EXCLUSIVE;
    imageCreateInfo.initialLayout = VK_IMAGE_LAYOUT_UNDEFINED;

    vkCreateImage(device, &imageCreateInfo, NULL, &image);
    vkGetImageMemoryRequirements(device,
                                 image,
                                 &imageMemoryRequirements);

    // For simplicity, just pick the first compatible memory type.
    for (memoryType = 0; memoryType < numMemoryTypes; memoryType++) {
        if ((1 << memoryType) & imageMemoryRequirements.memoryTypeBits) {
            break;
        }
    }

    // At least one memory type must be supported given the prior external
    // handle capability check.
    assert(memoryType < numMemoryTypes);

    // Allocate the external memory object.
    memset(&exportMemoryAllocateInfo, 0, sizeof(exportMemoryAllocateInfo));
    exportMemoryAllocateInfo.sType =
        VK_STRUCTURE_TYPE_EXPORT_MEMORY_ALLOCATE_INFO_NV;
    importMemoryInfo.handleTypes = handleType;
    importMemoryInfo.handle = sharedNtHandle;

    memset(&memoryAllocateInfo, 0, sizeof(memoryAllocateInfo));
    memoryAllocateInfo.sType = VK_STRUCTURE_TYPE_MEMORY_ALLOCATE_INFO;
    memoryAllocateInfo.pNext = &exportMemoryAllocateInfo;
    memoryAllocateInfo.allocationSize = imageMemoryRequirements.size;
    memoryAllocateInfo.memoryTypeIndex = memoryType;

    vkAllocateMemory(device, &memoryAllocateInfo, NULL, &mem);

    vkBindImageMemory(device, image, mem, 0);
```

```
...

const uint64_t acquireKey = 1;
const uint32_t timeout = INFINITE;
const uint64_t releaseKey = 2;

VkWin32KeyedMutexAcquireReleaseInfoNV keyedMutex =
    { VK_STRUCTURE_TYPE_WIN32_KEYED_MUTEX_ACQUIRE_RELEASE_INFO_NV };
keyedMutex.acquireCount = 1;
keyedMutex.pAcquireSyncs = &mem;
keyedMutex.pAcquireKeys = &acquireKey;
keyedMutex.pAcquireTimeoutMilliseconds = &timeout;
keyedMutex.releaseCount = 1;
keyedMutex.pReleaseSyncs = &mem;
keyedMutex.pReleaseKeys = &releaseKey;

VkSubmitInfo submit_info = { VK_STRUCTURE_TYPE_SUBMIT_INFO, &keyedMutex };
submit_info.commandBufferCount = 1;
submit_info.pCommandBuffers = &cmd_buf;
vkQueueSubmit(queue, 1, &submit_info, VK_NULL_HANDLE);
```

### Version History

- Revision 2, 2016-08-11 (James Jones)
  - Updated sample code based on the NV external memory extensions.
  - Renamed from NVX to NV extension.
  - Added Overview and Description sections.
  - Updated sample code to use the NV external memory extensions.
- Revision 1, 2016-06-14 (Carsten Rohde)
  - Initial draft.

## VK_VALVE_mutable_descriptor_type

### Name String

`VK_VALVE_mutable_descriptor_type`

### Extension Type

Device extension

### Registered Extension Number

352

### Revision

1

**Ratification Status**

Not ratified

**Extension and Version Dependencies**

VK_KHR_maintenance3

**Deprecation State**

- *Promoted* to VK_EXT_mutable_descriptor_type extension

**Special Use**

- D3D support

**Contact**

- Joshua Ashton [GitHub]Joshua-Ashton
- Hans-Kristian Arntzen [GitHub]HansKristian-Work

**Other Extension Metadata**

**Last Modified Date**

2020-12-02

**IP Status**

No known IP claims.

**Contributors**

- Joshua Ashton, Valve
- Hans-Kristian Arntzen, Valve

**Description**

This extension allows applications to reduce descriptor memory footprint by allowing a descriptor to be able to mutate to a given list of descriptor types depending on which descriptor types are written into, or copied into a descriptor set.

The main use case this extension intends to address is descriptor indexing with `VK_DESCRIPTOR_BINDING_VARIABLE_DESCRIPTOR_COUNT_BIT` where the descriptor types are completely generic, as this means applications can allocate one large descriptor set, rather than having one large descriptor set per descriptor type, which significantly bloats descriptor memory usage and causes performance issues.

This extension also adds a mechanism to declare that a descriptor pool, and therefore the descriptor sets that are allocated from it, reside only in host memory; as such these descriptors can only be updated/copied, but not bound.

These features together allow much more efficient emulation of the raw D3D12 binding model. This extension is primarily intended to be useful for API layering efforts.

### New Structures

- VkMutableDescriptorTypeListVALVE
- Extending VkDescriptorSetLayoutCreateInfo, VkDescriptorPoolCreateInfo:
  - VkMutableDescriptorTypeCreateInfoVALVE
- Extending VkPhysicalDeviceFeatures2, VkDeviceCreateInfo:
  - VkPhysicalDeviceMutableDescriptorTypeFeaturesVALVE

### New Enum Constants

- `VK_VALVE_MUTABLE_DESCRIPTOR_TYPE_EXTENSION_NAME`
- `VK_VALVE_MUTABLE_DESCRIPTOR_TYPE_SPEC_VERSION`
- Extending VkDescriptorPoolCreateFlagBits:
  - `VK_DESCRIPTOR_POOL_CREATE_HOST_ONLY_BIT_VALVE`
- Extending VkDescriptorSetLayoutCreateFlagBits:
  - `VK_DESCRIPTOR_SET_LAYOUT_CREATE_HOST_ONLY_POOL_BIT_VALVE`
- Extending VkDescriptorType:
  - `VK_DESCRIPTOR_TYPE_MUTABLE_VALVE`
- Extending VkStructureType:
  - `VK_STRUCTURE_TYPE_MUTABLE_DESCRIPTOR_TYPE_CREATE_INFO_VALVE`
  - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MUTABLE_DESCRIPTOR_TYPE_FEATURES_VALVE`

### Version History

- Revision 1, 2020-12-01 (Joshua Ashton, Hans-Kristian Arntzen)
  - Initial specification, squashed from public draft.

# Appendix F: API Boilerplate

This appendix defines Vulkan API features that are infrastructure required for a complete functional description of Vulkan, but do not logically belong elsewhere in the Specification.

## Vulkan Header Files

Vulkan is defined as an API in the C99 language. Khronos provides a corresponding set of header files for applications using the API, which may be used in either C or C++ code. The interface descriptions in the specification are the same as the interfaces defined in these header files, and both are derived from the `vk.xml` XML API Registry, which is the canonical machine-readable description of the Vulkan API. The Registry, scripts used for processing it into various forms, and documentation of the registry schema are available as described at https://registry.khronos.org/vulkan/#apiregistry .

Language bindings for other languages can be defined using the information in the Specification and the Registry. Khronos does not provide any such bindings, but third-party developers have created some additional bindings.

### Vulkan Combined API Header `vulkan.h` (Informative)

Applications normally will include the header `vulkan.h`. In turn, `vulkan.h` always includes the following headers:

- `vk_platform.h`, defining platform-specific macros and headers.
- `vulkan_core.h`, defining APIs for the Vulkan core and all registered extensions *other* than window system-specific and provisional extensions, which are included in separate header files.

In addition, specific preprocessor macros defined at the time `vulkan.h` is included cause header files for the corresponding window system-specific and provisional interfaces to be included, as described below.

### Vulkan Platform-Specific Header `vk_platform.h` (Informative)

Platform-specific macros and interfaces are defined in `vk_platform.h`. These macros are used to control platform-dependent behavior, and their exact definitions are under the control of specific platforms and Vulkan implementations.

#### Platform-Specific Calling Conventions

On many platforms the following macros are empty strings, causing platform- and compiler-specific default calling conventions to be used.

`VKAPI_ATTR` is a macro placed before the return type in Vulkan API function declarations. This macro controls calling conventions for C++11 and GCC/Clang-style compilers.

`VKAPI_CALL` is a macro placed after the return type in Vulkan API function declarations. This macro controls calling conventions for MSVC-style compilers.

VKAPI_PTR is a macro placed between the '(' and '*' in Vulkan API function pointer declarations. This macro also controls calling conventions, and typically has the same definition as VKAPI_ATTR or VKAPI_CALL, depending on the compiler.

With these macros, a Vulkan function declaration takes the form of:

```
VKAPI_ATTR <return_type> VKAPI_CALL <command_name>(<command_parameters>);
```

Additionally, a Vulkan function pointer type declaration takes the form of:

```
typedef <return_type> (VKAPI_PTR *PFN_<command_name>)(<command_parameters>);
```

#### Platform-Specific Header Control

If the VK_NO_STDINT_H macro is defined by the application at compile time, extended integer types used by the Vulkan API, such as uint8_t, **must** also be defined by the application. Otherwise, the Vulkan headers will not compile. If VK_NO_STDINT_H is not defined, the system <stdint.h> is used to define these types. There is a fallback path when Microsoft Visual Studio version 2008 and earlier versions are detected at compile time.

If the VK_NO_STDDEF_H macro is defined by the application at compile time, size_t, **must** also be defined by the application. Otherwise, the Vulkan headers will not compile. If VK_NO_STDDEF_H is not defined, the system <stddef.h> is used to define this type.

### Vulkan Core API Header vulkan_core.h

Applications that do not make use of window system-specific extensions may simply include vulkan_core.h instead of vulkan.h, although there is usually no reason to do so. In addition to the Vulkan API, vulkan_core.h also defines a small number of C preprocessor macros that are described below.

#### Vulkan Header File Version Number

VK_HEADER_VERSION is the version number of the vulkan_core.h header. This value is kept synchronized with the patch version of the released Specification.

```
// Provided by VK_VERSION_1_0
// Version of this file
#define VK_HEADER_VERSION 295
```

VK_HEADER_VERSION_COMPLETE is the complete version number of the vulkan_core.h header, comprising the major, minor, and patch versions. The major/minor values are kept synchronized with the complete version of the released Specification. This value is intended for use by automated tools to identify exactly which version of the header was used during their generation.

Applications should not use this value as their VkApplicationInfo::apiVersion. Instead applications

should explicitly select a specific fixed major/minor API version using, for example, one of the VK_API_VERSION_*_* values.

```
// Provided by VK_VERSION_1_0
// Complete version of this file
#define VK_HEADER_VERSION_COMPLETE VK_MAKE_API_VERSION(0, 1, 3, VK_HEADER_VERSION)
```

VK_API_VERSION is now commented out of vulkan_core.h and **cannot** be used.

```
// Provided by VK_VERSION_1_0
// DEPRECATED: This define has been removed. Specific version defines (e.g.
VK_API_VERSION_1_0), or the VK_MAKE_VERSION macro, should be used instead.
//#define VK_API_VERSION VK_MAKE_API_VERSION(0, 1, 0, 0) // Patch version should
always be set to 0
```

### Vulkan Handle Macros

VK_DEFINE_HANDLE defines a dispatchable handle type.

```
// Provided by VK_VERSION_1_0

#define VK_DEFINE_HANDLE(object) typedef struct object##_T* object;
```

- object is the name of the resulting C type.

The only dispatchable handle types are those related to device and instance management, such as VkDevice.

VK_DEFINE_NON_DISPATCHABLE_HANDLE defines a non-dispatchable handle type.

```
// Provided by VK_VERSION_1_0

#ifndef VK_DEFINE_NON_DISPATCHABLE_HANDLE
    #if (VK_USE_64_BIT_PTR_DEFINES==1)
        #define VK_DEFINE_NON_DISPATCHABLE_HANDLE(object) typedef struct object##_T
*object;
    #else
        #define VK_DEFINE_NON_DISPATCHABLE_HANDLE(object) typedef uint64_t object;
    #endif
#endif
```

- object is the name of the resulting C type.

Most Vulkan handle types, such as VkBuffer, are non-dispatchable.

**NOTE** The vulkan_core.h header allows the VK_DEFINE_NON_DISPATCHABLE_HANDLE

and VK_NULL_HANDLE definitions to be overridden by the application. If VK_DEFINE_NON_DISPATCHABLE_HANDLE is already defined when vulkan_core.h is compiled, the default definitions for VK_DEFINE_NON_DISPATCHABLE_HANDLE and VK_NULL_HANDLE are skipped. This allows the application to define a binary-compatible custom handle which **may** provide more type-safety or other features needed by the application. Applications **must** not define handles in a way that is not binary compatible - where binary compatibility is platform dependent.

VK_NULL_HANDLE is a reserved value representing a non-valid object handle. It may be passed to and returned from Vulkan commands only when specifically allowed.

```
// Provided by VK_VERSION_1_0

#ifndef VK_DEFINE_NON_DISPATCHABLE_HANDLE
    #if (VK_USE_64_BIT_PTR_DEFINES==1)
        #if (defined(__cplusplus) && (__cplusplus >= 201103L)) || (defined(_MSVC_LANG)
&& (_MSVC_LANG >= 201103L))
            #define VK_NULL_HANDLE nullptr
        #else
            #define VK_NULL_HANDLE ((void*)0)
        #endif
    #else
        #define VK_NULL_HANDLE 0ULL
    #endif
#endif
#ifndef VK_NULL_HANDLE
    #define VK_NULL_HANDLE 0
#endif
```

VK_USE_64_BIT_PTR_DEFINES defines whether the default non-dispatchable handles are declared using either a 64-bit pointer type or a 64-bit unsigned integer type.

VK_USE_64_BIT_PTR_DEFINES is set to '1' to use a 64-bit pointer type or any other value to use a 64-bit unsigned integer type.

```
// Provided by VK_VERSION_1_0

#ifndef VK_USE_64_BIT_PTR_DEFINES
    #if defined(__LP64__) || defined(_WIN64) || (defined(__x86_64__) &&
!defined(__ILP32__) ) || defined(_M_X64) || defined(__ia64) || defined (_M_IA64) ||
defined(__aarch64__) || defined(__powerpc64__) || (defined(__riscv) && __riscv_xlen ==
64)
        #define VK_USE_64_BIT_PTR_DEFINES 1
    #else
        #define VK_USE_64_BIT_PTR_DEFINES 0
    #endif
#endif
```

**NOTE** The `vulkan_core.h` header allows the `VK_USE_64_BIT_PTR_DEFINES` definition to be overridden by the application. This allows the application to select either a 64-bit pointer type or a 64-bit unsigned integer type for non-dispatchable handles in the case where the predefined preprocessor check does not identify the desired configuration.

**NOTE** This macro was introduced starting with the Vulkan 1.2.174 headers, and its availability can be checked at compile time by requiring `VK_HEADER_VERSION >= 174`.

It is not available if you are using older headers, such as may be shipped with an older Vulkan SDK. Developers requiring this functionality may wish to include a copy of the current Vulkan headers with their project in this case.

# Window System-Specific Header Control (Informative)

To use a Vulkan extension supporting a platform-specific window system, header files for that window system **must** be included at compile time, or platform-specific types **must** be forward-declared. The Vulkan header files are unable to determine whether or not an external header is available at compile time, so platform-specific extensions are provided in separate headers from the core API and platform-independent extensions, allowing applications to decide which ones they need to be defined and how the external headers are included.

Extensions dependent on particular sets of platform headers, or that forward-declare platform-specific types, are declared in a header named for that platform. Before including these platform-specific Vulkan headers, applications **must** include both `vulkan_core.h` and any external native headers the platform extensions depend on.

As a convenience for applications that do not need the flexibility of separate platform-specific Vulkan headers, `vulkan.h` includes `vulkan_core.h`, and then conditionally includes platform-specific Vulkan headers and the external headers they depend on. Applications control which platform-specific headers are included by #defining macros before including `vulkan.h`.

The correspondence between platform-specific extensions, external headers they require, the platform-specific header which declares them, and the preprocessor macros which enable inclusion by `vulkan.h` are shown in the following table.

*Table 108. Window System Extensions and Headers*

| Extension Name | Window System Name | Platform-specific Header | Required External Headers | Controlling `vulkan.h` Macro |
|---|---|---|---|---|
| `VK_KHR_android_surface` | Android | `vulkan_android.h` | None | `VK_USE_PLATFORM_ANDROID_KHR` |
| `VK_KHR_wayland_surface` | Wayland | `vulkan_wayland.h` | `<wayland-client.h>` | `VK_USE_PLATFORM_WAYLAND_KHR` |

| Extension Name | Window System Name | Platform-specific Header | Required External Headers | Controlling vulkan.h Macro |
|---|---|---|---|---|
| VK_KHR_win32_surface, VK_KHR_external_memory_win32, VK_KHR_win32_keyed_mutex, VK_KHR_external_semaphore_win32, VK_KHR_external_fence_win32, VK_NV_external_memory_win32, VK_NV_win32_keyed_mutex | Microsoft Windows | vulkan_win32.h | <windows.h> | VK_USE_PLATFORM_WIN32_KHR |
| VK_KHR_xcb_surface | X11 Xcb | vulkan_xcb.h | <xcb/xcb.h> | VK_USE_PLATFORM_XCB_KHR |
| VK_KHR_xlib_surface | X11 Xlib | vulkan_xlib.h | <X11/Xlib.h> | VK_USE_PLATFORM_XLIB_KHR |
| VK_EXT_directfb_surface | DirectFB | vulkan_directfb.h | <directfb/directfb.h> | VK_USE_PLATFORM_DIRECTFB_EXT |
| VK_EXT_acquire_xlib_display | X11 XRAndR | vulkan_xlib_xrandr.h | <X11/Xlib.h>, <X11/extensions/Xrandr.h> | VK_USE_PLATFORM_XLIB_XRANDR_EXT |
| VK_GGP_stream_descriptor_surface, VK_GGP_frame_token | Google Games Platform | vulkan_ggp.h | <ggp_c/vulkan_types.h> | VK_USE_PLATFORM_GGP |
| VK_MVK_ios_surface | iOS | vulkan_ios.h | None | VK_USE_PLATFORM_IOS_MVK |
| VK_MVK_macos_surface | macOS | vulkan_macos.h | None | VK_USE_PLATFORM_MACOS_MVK |
| VK_NN_vi_surface | VI | vulkan_vi.h | None | VK_USE_PLATFORM_VI_NN |
| VK_FUCHSIA_imagepipe_surface | Fuchsia | vulkan_fuchsia.h | <zircon/types.h> | VK_USE_PLATFORM_FUCHSIA |
| VK_EXT_metal_surface | Metal on CoreAnimation | vulkan_metal.h | None | VK_USE_PLATFORM_METAL_EXT |
| VK_QNX_screen_surface | QNX Screen | vulkan_screen.h | <screen/screen.h> | VK_USE_PLATFORM_SCREEN_QNX |

**NOTE** This section describes the purpose of the headers independently of the specific underlying functionality of the window system extensions themselves. Each extension name will only link to a description of that extension when viewing a specification built with that extension included.

# Provisional Extension Header Control (Informative)

*Provisional* extensions **should** not be used in production applications. The functionality defined by such extensions **may** change in ways that break backwards compatibility between revisions, and before final release of a non-provisional version of that extension.

Provisional extensions are defined in a separate *provisional header*, `vulkan_beta.h`, allowing applications to decide whether or not to include them. The mechanism is similar to window system-specific headers: before including `vulkan_beta.h`, applications **must** include `vulkan_core.h`.

**NOTE** Sometimes a provisional extension will include a subset of its interfaces in `vulkan_core.h`. This may occur if the provisional extension is promoted from an existing vendor or EXT extension and some of the existing interfaces are defined as aliases of the provisional extension interfaces. All other interfaces of that provisional extension which are not aliased will be included in `vulkan_beta.h`.

As a convenience for applications, `vulkan.h` conditionally includes `vulkan_beta.h`. Applications **can** control inclusion of `vulkan_beta.h` by #defining the macro `VK_ENABLE_BETA_EXTENSIONS` before including `vulkan.h`.

**NOTE** Starting in version 1.2.171 of the Specification, all provisional enumerants are protected by the macro `VK_ENABLE_BETA_EXTENSIONS`. Applications needing to use provisional extensions must always define this macro, even if they are explicitly including `vulkan_beta.h`. This is a minor change to behavior, affecting only provisional extensions.

**NOTE** This section describes the purpose of the provisional header independently of the specific provisional extensions which are contained in that header at any given time. The extension appendices for provisional extensions note their provisional status, and link back to this section for more information. Provisional extensions are intended to provide early access for bleeding-edge developers, with the understanding that extension interfaces may change in response to developer feedback. Provisional extensions are very likely to eventually be updated and released as non-provisional extensions, but there is no guarantee this will happen, or how long it will take if it does happen.

# Video Std Headers

Performing video coding operations usually involves the application having to provide various parameters, data structures, or other syntax elements specific to the particular video compression standard used, and the associated semantics are covered by the specification of those.

The interface descriptions of these are available in the header files derived from the `video.xml` XML file, which is the canonical machine-readable description of data structures and enumerations that are associated with the externally-provided video compression standards.

*Table 109. Video Std Headers*

| Video Std Header Name | Description | Header File | Related Extensions |
|---|---|---|---|
| vulkan_video_codecs_common | Codec-independent common definitions | <vk_video/vulkan_video_codecs_common.h> | - |
| vulkan_video_codec_h264std | ITU-T H.264 common definitions | <vk_video/vulkan_video_codec_h264std.h> | VK_KHR_video_decode_h264, VK_KHR_video_encode_h264 |
| vulkan_video_codec_h264std_decode | ITU-T H.264 decode-specific definitions | <vk_video/vulkan_video_codec_h264std_decode.h> | VK_KHR_video_decode_h264 |
| vulkan_video_codec_h264std_encode | ITU-T H.264 encode-specific definitions | <vk_video/vulkan_video_codec_h264std_encode.h> | VK_KHR_video_encode_h264 |
| vulkan_video_codec_h265std | ITU-T H.265 common definitions | <vk_video/vulkan_video_codec_h265std.h> | VK_KHR_video_decode_h265, VK_KHR_video_encode_h265 |
| vulkan_video_codec_h265std_decode | ITU-T H.265 decode-specific definitions | <vk_video/vulkan_video_codec_h265std_decode.h> | VK_KHR_video_decode_h265 |
| vulkan_video_codec_h265std_encode | ITU-T H.265 encode-specific definitions | <vk_video/vulkan_video_codec_h265std_encode.h> | VK_KHR_video_encode_h265 |
| vulkan_video_codec_av1std | AV1 common definitions | <vk_video/vulkan_video_codec_av1std.h> | VK_KHR_video_decode_av1 |
| vulkan_video_codec_av1std_decode | AV1 decode-specific definitions | <vk_video/vulkan_video_codec_av1std_decode.h> | VK_KHR_video_decode_av1 |

# Appendix G: Invariance

The Vulkan specification is not pixel exact. It therefore does not guarantee an exact match between images produced by different Vulkan implementations. However, the specification does specify exact matches, in some cases, for images produced by the same implementation. The purpose of this appendix is to identify and provide justification for those cases that require exact matches.

## Repeatability

The obvious and most fundamental case is repeated issuance of a series of Vulkan commands. For any given Vulkan and framebuffer state vector, and for any Vulkan command, the resulting Vulkan and framebuffer state **must** be identical whenever the command is executed on that initial Vulkan and framebuffer state. This repeatability requirement does not apply when using shaders containing side effects (image and buffer variable stores and atomic operations), because these memory operations are not guaranteed to be processed in a defined order.

The repeatability requirement does not apply for rendering done using a graphics pipeline that uses `VK_RASTERIZATION_ORDER_RELAXED_AMD`.

One purpose of repeatability is avoidance of visual artifacts when a double-buffered scene is redrawn. If rendering is not repeatable, swapping between two buffers rendered with the same command sequence **may** result in visible changes in the image. Such false motion is distracting to the viewer. Another reason for repeatability is testability.

Repeatability, while important, is a weak requirement. Given only repeatability as a requirement, two scenes rendered with one (small) polygon changed in position might differ at every pixel. Such a difference, while within the law of repeatability, is certainly not within its spirit. Additional invariance rules are desirable to ensure useful operation.

## Multi-Pass Algorithms

Invariance is necessary for a whole set of useful multi-pass algorithms. Such algorithms render multiple times, each time with a different Vulkan mode vector, to eventually produce a result in the framebuffer. Examples of these algorithms include:

- “Erasing” a primitive from the framebuffer by redrawing it, either in a different color or using the XOR logical operation.
- Using stencil operations to compute capping planes.

## Invariance Rules

For a given Vulkan device:

**Rule 1** *For any given Vulkan and framebuffer state vector, and for any given Vulkan command, the resulting Vulkan and framebuffer state* ***must*** *be identical each time the command is executed on that initial Vulkan and framebuffer state.*

**Rule 2** *Changes to the following state values have no side effects (the use of any other state value is not affected by the change):*

**Required:**

- *Color and depth/stencil attachment contents*
- *Scissor parameters (other than enable)*
- *Write masks (color, depth, stencil)*
- *Clear values (color, depth, stencil)*

**Strongly suggested:**

- *Stencil parameters (other than enable)*
- *Depth test parameters (other than enable)*
- *Blend parameters (other than enable)*
- *Logical operation parameters (other than enable)*

**Corollary 1** *Fragment generation is invariant with respect to the state values listed in Rule 2.*

**Rule 3** *The arithmetic of each per-fragment operation is invariant except with respect to parameters that directly control it.*

**Corollary 2** *Images rendered into different color attachments of the same framebuffer, either simultaneously or separately using the same command sequence, are pixel identical.*

**Rule 4** *Identical pipelines will produce the same result when run multiple times with the same input. The wording "Identical pipelines" means VkPipeline objects that have been created with identical SPIR-V binaries and identical state, which are then used by commands executed using the same Vulkan state vector. Invariance is relaxed for shaders with side effects, such as performing stores or atomics.*

**Rule 5** *All fragment shaders that either conditionally or unconditionally assign* `FragCoord.z` *to* `FragDepth` *are depth-invariant with respect to each other, for those fragments where the assignment to* `FragDepth` *actually is done.*

If a sequence of Vulkan commands specifies primitives to be rendered with shaders containing side effects (image and buffer variable stores and atomic operations), invariance rules are relaxed. In particular, rule 1, corollary 2, and rule 4 do not apply in the presence of shader side effects.

The following weaker versions of rules 1 and 4 apply to Vulkan commands involving shader side effects:

**Rule 6** *For any given Vulkan and framebuffer state vector, and for any given Vulkan command, the contents of any framebuffer state not directly or indirectly affected by results of shader image or buffer variable stores or atomic operations* ***must*** *be identical each time the command is executed on that initial Vulkan and framebuffer state.*

**Rule 7** *Identical pipelines will produce the same result when run multiple times with the same input*

*as long as:*

- *shader invocations do not use image atomic operations;*
- *no framebuffer memory is written to more than once by image stores, unless all such stores write the same value; and*
- *no shader invocation, or other operation performed to process the sequence of commands, reads memory written to by an image store.*

**NOTE**

The OpenGL specification has the following invariance rule: Consider a primitive p' obtained by translating a primitive p through an offset (x, y) in window coordinates, where x and y are integers. As long as neither p' nor p is clipped, it **must** be the case that each fragment f' produced from p' is identical to a corresponding fragment f from p except that the center of f' is offset by (x, y) from the center of f.

This rule does not apply to Vulkan and is an intentional difference from OpenGL.

When any sequence of Vulkan commands triggers shader invocations that perform image stores or atomic operations, and subsequent Vulkan commands read the memory written by those shader invocations, these operations **must** be explicitly synchronized.

# Tessellation Invariance

When using a pipeline containing tessellation evaluation shaders, the fixed-function tessellation primitive generator consumes the input patch specified by an application and emits a new set of primitives. The following invariance rules are intended to provide repeatability guarantees. Additionally, they are intended to allow an application with a carefully crafted tessellation evaluation shader to ensure that the sets of triangles generated for two adjacent patches have identical vertices along shared patch edges, avoiding “cracks” caused by minor differences in the positions of vertices along shared edges.

**Rule 1** *When processing two patches with identical outer and inner tessellation levels, the tessellation primitive generator will emit an identical set of point, line, or triangle primitives as long as the pipeline used to process the patch primitives has tessellation evaluation shaders specifying the same tessellation mode, spacing, vertex order, and point mode decorations. Two sets of primitives are considered identical if and only if they contain the same number and type of primitives and the generated tessellation coordinates for the vertex numbered m of the primitive numbered n are identical for all values of m and n.*

**Rule 2** *The set of vertices generated along the outer edge of the subdivided primitive in triangle and quad tessellation, and the tessellation coordinates of each, depend only on the corresponding outer tessellation level and the spacing decorations in the tessellation shaders of the pipeline.*

**Rule 3** *The set of vertices generated when subdividing any outer primitive edge is always symmetric. For triangle tessellation, if the subdivision generates a vertex with tessellation coordinates of the form (0, x, 1-x), (x, 0, 1-x), or (x, 1-x, 0), it will also generate a vertex with coordinates of exactly (0, 1-x, x), (1-x, 0, x), or (1-x, x, 0), respectively. For quad tessellation, if the subdivision generates a vertex with coordinates of (x, 0) or (0, x), it will also generate a vertex with coordinates of exactly (1-x, 0) or (0, 1-x), respectively. For isoline tessellation, if it generates vertices at (0, x) and (1, x) where x is not zero, it*

*will also generate vertices at exactly (0, 1-x) and (1, 1-x), respectively.*

**Rule 4** *The set of vertices generated when subdividing outer edges in triangular and quad tessellation* ***must*** *be independent of the specific edge subdivided, given identical outer tessellation levels and spacing. For example, if vertices at (x, 1 - x, 0) and (1-x, x, 0) are generated when subdividing the w = 0 edge in triangular tessellation, vertices* ***must*** *be generated at (x, 0, 1-x) and (1-x, 0, x) when subdividing an otherwise identical v = 0 edge. For quad tessellation, if vertices at (x, 0) and (1-x, 0) are generated when subdividing the v = 0 edge, vertices* ***must*** *be generated at (0, x) and (0, 1-x) when subdividing an otherwise identical u = 0 edge.*

**Rule 5** *When processing two patches that are identical in all respects enumerated in rule 1 except for vertex order, the set of triangles generated for triangle and quad tessellation* ***must*** *be identical except for vertex and triangle order. For each triangle n1 produced by processing the first patch, there* ***must*** *be a triangle n2 produced when processing the second patch each of whose vertices has the same tessellation coordinates as one of the vertices in n1.*

**Rule 6** *When processing two patches that are identical in all respects enumerated in rule 1 other than matching outer tessellation levels and/or vertex order, the set of interior triangles generated for triangle and quad tessellation* ***must*** *be identical in all respects except for vertex and triangle order. For each interior triangle n1 produced by processing the first patch, there* ***must*** *be a triangle n2 produced when processing the second patch each of whose vertices has the same tessellation coordinates as one of the vertices in n1. A triangle produced by the tessellator is considered an interior triangle if none of its vertices lie on an outer edge of the subdivided primitive.*

**Rule 7** *For quad and triangle tessellation, the set of triangles connecting an inner and outer edge depends only on the inner and outer tessellation levels corresponding to that edge and the spacing decorations.*

**Rule 8** *The value of all defined components of* `TessCoord` *will be in the range [0, 1]. Additionally, for any defined component x of* `TessCoord`*, the results of computing 1.0-x in a tessellation evaluation shader will be exact. If any floating-point values in the range [0, 1] fail to satisfy this property, such values* ***must*** *not be used as tessellation coordinate components.*

# Appendix H: Lexicon

This appendix defines terms, abbreviations, and API prefixes used in the Specification.

## Glossary

The terms defined in this section are used consistently throughout the Specification and may be used with or without capitalization.

**Accessible (Descriptor Binding)**

A descriptor binding is accessible to a shader stage if that stage is included in the `stageFlags` of the descriptor binding. Descriptors using that binding **can** only be used by stages in which they are accessible.

**Acquire Operation (Resource)**

An operation that acquires ownership of an image subresource or buffer range.

**Active (Descriptor Type)**

When a descriptor with *mutable* type is updated with vkUpdateDescriptorSets, the active descriptor type changes. When the descriptor is consumed by shaders, it is the active descriptor type which determines validity, i.e. `VkDescriptorSetLayoutBinding`::`descriptorType` is replaced with the active descriptor type. A mismatch in active descriptor type and consumption by shader is considered an undefined descriptor.

**Active (Transform Feedback)**

Transform feedback is made active after vkCmdBeginTransformFeedbackEXT executes and remains active until vkCmdEndTransformFeedbackEXT executes. While transform feedback is active, data written to variables in the output interface of the last pre-rasterization shader stage of the graphics pipeline are captured to the bound transform feedback buffers if those variables are decorated for transform feedback.

**Adjacent Vertex**

A vertex in an adjacency primitive topology that is not part of a given primitive, but is accessible in geometry shaders.

**Active Object (Ray Tracing)**

A primitive or instance in a ray tracing acceleration structure which has a corresponding ID, and is not *inactive* (meaning that it is visible to rays).

**Advanced Blend Operation**

Blending performed using one of the blend operation enums introduced by the `VK_EXT_blend_operation_advanced` extension. See Advanced Blending Operations.

**Alias (API type/command)**

An identical definition of another API type/command with the same behavior but a different name.

**Aliased Range (Memory)**

A range of a device memory allocation that is bound to multiple resources simultaneously.

**Allocation Scope**

An association of a host memory allocation to a parent object or command, where the allocation's lifetime ends before or at the same time as the parent object is freed or destroyed, or during the parent command.

**API command**

Any command defined in the Vulkan specification. These entry points all have a vk prefix.

**Aspect (Image)**

Some image types contain multiple kinds (called "aspects") of data for each pixel, where each aspect is used in a particular way by the pipeline and **may** be stored differently or separately from other aspects. For example, the color components of an image format make up the color aspect of the image, and **can** be used as a framebuffer color attachment. Some operations, like depth testing, operate only on specific aspects of an image.

**Attachment (Render Pass)**

A zero-based integer index name used in render pass creation to refer to a framebuffer attachment that is accessed by one or more subpasses. The index also refers to an attachment description which includes information about the properties of the image view that will later be attached.

**Availability Operation**

An operation that causes the values generated by specified memory write accesses to become available for future access.

**Available**

A state of values written to memory that allows them to be made visible.

**Axis-aligned Bounding Box**

A box bounding a region in space defined by extents along each axis and thus representing a box where each edge is aligned to one of the major axes.

**Back-Facing**

See Facingness.

**Backward Prediction**

Sample prediction performed during video decode and encode operations based on reference pictures that occur temporally (in display order) after the current picture.

**Backward Reference**

A reference picture that occurs temporally (in display order) after the current picture.

**Batch**

A single structure submitted to a queue as part of a queue submission command, describing a set of queue operations to execute.

**Backwards Compatibility**

A given version of the API is backwards compatible with an earlier version if an application, relying only on valid behavior and functionality defined by the earlier specification, is able to correctly run against each version without any modification. This assumes no active attempt by that application to not run when it detects a different version.

**Binary Semaphore**

A semaphore with a boolean payload indicating whether the semaphore is signaled or unsignaled. Represented by a VkSemaphore object created with a semaphore type of `VK_SEMAPHORE_TYPE_BINARY` .

**Binding (Memory)**

An association established between a range of a resource object and a range of a memory object. These associations determine the memory locations affected by operations performed on elements of a resource object. Memory bindings are established using the vkBindBufferMemory command for non-sparse buffer objects, using the vkBindImageMemory command for non-sparse image objects , and using the vkQueueBindSparse command for sparse resources .

**Blend Constant**

Four floating-point (RGBA) values used as an input to blending.

**Blending**

Arithmetic operations between a fragment color value and a value in a color attachment that produce a final color value to be written to the attachment.

**Buffer**

A resource that represents a linear array of data in device memory. Represented by a VkBuffer object.

**Buffer Device Address**

A 64-bit value used in a shader to access buffer memory through the `PhysicalStorageBuffer` storage class.

**Buffer View**

An object that represents a range of a specific buffer, and state controlling how the contents are interpreted. Represented by a VkBufferView object.

**Built-In Variable**

A variable decorated in a shader, where the decoration makes the variable take values provided by the execution environment or values that are generated by fixed-function pipeline stages.

**Built-In Interface Block**

A block defined in a shader containing only variables decorated with built-in decorations, and is used to match against other shader stages.

**Clip Coordinates**

The homogeneous coordinate space in which vertex positions (`Position` decoration) are written by pre-rasterization shader stages.

**Clip Distance**

A built-in output from pre-rasterization shader stages defining a clip half-space against which the primitive is clipped.

**Clip Volume**

The intersection of the view volume with all clip half-spaces.

**Coding Block (Video)**

An element of a video picture, comprising a rectangular block of texel values that are coded as a single unit in video coding. Video coding blocks have a corresponding coding block size (width and height) defining the dimensions of these elements in units of texels. These dimensions are usually codec-specific and a single video compression standard **may** support more than one coding block type and size. Video compression standards **may** use multiple coding block types to organize coding blocks of different kinds and sizes into hierarchies.

**Color Attachment**

A subpass attachment point, or image view, that is the target of fragment color outputs and blending.

**Color Fragment**

A unique color value within a pixel of a multisampled color image. The *fragment mask* will contain indices to the *color fragment*.

**Color Renderable Format**

A VkFormat where `VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BIT` is set in one of the following, depending on the image's tiling:

- VkFormatProperties::`linearTilingFeatures`
- VkFormatProperties::`optimalTilingFeatures` or a VkFormat where `VK_FORMAT_FEATURE_2_LINEAR_COLOR_ATTACHMENT_BIT_NV` is set in VkFormatProperties::`linearTilingFeatures`
- VkDrmFormatModifierPropertiesEXT::`drmFormatModifierTilingFeatures`

**Combined Image Sampler**

A descriptor type that includes both a sampled image and a sampler.

**Command Buffer**

An object that records commands to be submitted to a queue. Represented by a VkCommandBuffer object.

**Command Buffer Nesting Level**

The Command Buffer Nesting Level of a secondary command buffer is equal to the maximum nesting level of all secondary command buffers executed by that command buffer plus one, where a secondary command buffer that executes no other secondary command buffers has a nesting level of zero.

**Command Pool**

An object that command buffer memory is allocated from, and that owns that memory.

Command pools aid multithreaded performance by enabling different threads to use different allocators, without internal synchronization on each use. Represented by a VkCommandPool object.

**Compatible Allocator**

When allocators are compatible, allocations from each allocator **can** be freed by the other allocator.

**Compatible Image Formats**

When formats are compatible, images created with one of the formats **can** have image views created from it using any of the compatible formats. Also see *Size-Compatible Image Formats*.

**Compatible Queues**

Queues within a queue family. Compatible queues have identical properties.

**Complete Mipmap Chain**

The entire set of mip levels that can be provided for an image, from the largest application-specified mip level size down to the *minimum mip level size*. See Image Mip Level Sizing.

**Completed Operation**

A deferred operation whose corresponding command has been executed to completion. See Deferred Host Operations

**Component (Format)**

A distinct part of a format. Color components are represented with `R`, `G`, `B`, and `A`. Depth and stencil components are represented with `D` and `S`. Formats **can** have multiple instances of the same component. Some formats have other notations such as `E` or `X` which are not considered a component of the format.

**Compressed Texel Block**

An element of an image having a block-compressed format, comprising a rectangular block of texel values that are encoded as a single value in memory. Compressed texel blocks of a particular block-compressed format have a corresponding width, height, and depth defining the dimensions of these elements in units of texels, and a size in bytes of the encoding in memory.

**Constant Integral Expressions**

A SPIR-V constant instruction whose type is `OpTypeInt`. See *Constant Instruction* in section 2.2.1 “Instructions” of the Khronos SPIR-V Specification.

**Cooperative Matrix**

A SPIR-V type where the storage for and computations performed on the matrix are spread across a set of invocations such as a subgroup.

**Corner-Sampled Image**

A VkImage where unnormalized texel coordinates are centered on integer values instead of half-integer values. Specified by setting the `VK_IMAGE_CREATE_CORNER_SAMPLED_BIT_NV` bit on VkImageCreateInfo::`flags` at image creation.

**Coverage Index**

The index of a sample in the coverage mask.

**Coverage Mask**

A bitfield associated with a fragment representing the samples that were determined to be covered based on the result of rasterization, and then subsequently modified by fragment operations or the fragment shader.

**Cull Distance**

A built-in output from pre-rasterization shader stages defining a cull half-space where the primitive is rejected if all vertices have a negative value for the same cull distance.

**Cull Volume**

The intersection of the view volume with all cull half-spaces.

**Decode Output Picture**

A video picture resource used to store the result of a video decode operation.

**Decoded Picture Buffer**

An indexed set of reference pictures used by a video session. Abbreviated as DPB.

**Decoded Picture Buffer Slot**

An entry within a DPB that **can** be associated with a particular reference picture.

**Decoded Picture Buffer Slot Index**

The index of a DPB slot within its encompassing DPB.

**Decoration (SPIR-V)**

Auxiliary information such as built-in variables, stream numbers, invariance, interpolation type, relaxed precision, etc., added to variables or structure-type members through decorations.

**Deferrable Command**

A command which allows deferred execution of host-side work. See Deferred Host Operations.

**Deferrable Operation**

A single logical item of host-side work which can be deferred. Represented by the VkDeferredOperationKHR object. See Deferred Host Operations.

**Deprecated (feature)**

A feature is deprecated if it is no longer recommended as the correct or best way to achieve its intended purpose.

**Depth/Stencil Attachment**

A subpass attachment point, or image view, that is the target of depth and/or stencil test operations and writes.

**Depth/Stencil Format**

A VkFormat that includes depth and/or stencil components.

**Depth/Stencil Image (or ImageView)**

A VkImage (or VkImageView) with a depth/stencil format.

**Depth/Stencil Resolve Attachment**

A subpass attachment point, or image view, that is the target of a multisample resolve operation from the corresponding depth/stencil attachment at the end of the subpass.

**Derivative Group**

A set of compute, mesh, task, or fragment shader invocations that cooperate to compute derivatives, including implicit derivatives for sampled image operations.

**Descriptor**

Information about a resource or resource view written into a descriptor set that is used to access the resource or view from a shader.

**Descriptor Binding**

An entry in a descriptor set layout corresponding to zero or more descriptors of a single descriptor type in a set. Defined by a VkDescriptorSetLayoutBinding structure.

**Descriptor Pool**

An object that descriptor sets are allocated from, and that owns the storage of those descriptor sets. Descriptor pools aid multithreaded performance by enabling different threads to use different allocators, without internal synchronization on each use. Represented by a VkDescriptorPool object.

**Descriptor Set**

An object that resource descriptors are written into via the API, and that **can** be bound to a command buffer such that the descriptors contained within it **can** be accessed from shaders. Represented by a VkDescriptorSet object.

**Descriptor Set Layout**

An object defining the set of resources (types and counts) and their relative arrangement (in the binding namespace) within a descriptor set. Used when allocating descriptor sets and when creating pipeline layouts. Represented by a VkDescriptorSetLayout object.

**Device**

The processor(s) and execution environment that perform tasks requested by the application via the Vulkan API.

**Device Group**

A set of physical devices that support accessing each other's memory and recording a single command buffer that **can** be executed on all the physical devices.

**Device Index**

A zero-based integer that identifies one physical device from a logical device. A device index is valid if it is less than the number of physical devices in the logical device.

**Device Mask**

A bitmask where each bit represents one device index. A device mask value is valid if every bit that is set in the mask is at a bit position that is less than the number of physical devices in the logical device.

**Device Memory**

Memory accessible to the device. Represented by a VkDeviceMemory object.

**Device-Level Command**

Any command that is dispatched from a logical device, or from a child object of a logical device.

**Device-Level Functionality**

All device-level commands and objects, and their structures, enumerated types, and enumerants. Additionally, physical-device-level functionality defined by a device extension is also considered device-level functionality.

**Device-Level Object**

Logical device objects and their child objects. For example, VkDevice, VkQueue, and VkCommandBuffer objects are device-level objects.

**Device-Local Memory**

Memory that is connected to the device, and **may** be more performant for device access than host-local memory.

**Direct Drawing Commands**

*Drawing commands* that take all their parameters as direct arguments to the command (and not sourced via structures in buffer memory as the *indirect drawing commands*). Includes vkCmdDrawMultiIndexedEXT, vkCmdDrawMultiEXT, vkCmdDrawMeshTasksNV, vkCmdDrawMeshTasksEXT, vkCmdDraw, and vkCmdDrawIndexed.

**Disjoint**

*Disjoint planes* are *image planes* to which memory is bound independently.

A *disjoint image* consists of multiple *disjoint planes*, and is created with the `VK_IMAGE_CREATE_DISJOINT_BIT` bit set.

**Dispatchable Command**

A non-global command. The first argument to each dispatchable command is a dispatchable handle type.

**Dispatchable Handle**

A handle of a pointer handle type which **may** be used by layers as part of intercepting API commands.

**Dispatching Commands**

Commands that provoke work using a compute pipeline. Includes vkCmdDispatch and vkCmdDispatchIndirect.

**Drawing Commands**

Commands that provoke work using a graphics pipeline. Includes vkCmdDraw, vkCmdDrawIndexed, vkCmdDrawIndirectCountKHR, vkCmdDrawIndexedIndirectCountKHR, vkCmdDrawIndirectCountAMD, vkCmdDrawIndexedIndirectCountAMD, vkCmdDrawMultiIndexedEXT, vkCmdDrawMultiEXT, vkCmdDrawMeshTasksNV, vkCmdDrawMeshTasksIndirectNV, vkCmdDrawMeshTasksIndirectCountNV, vkCmdDrawMeshTasksEXT, vkCmdDrawMeshTasksIndirectEXT, vkCmdDrawMeshTasksIndirectCountEXT, vkCmdDrawIndirect, and vkCmdDrawIndexedIndirect.

**Duration (Command)**

The *duration* of a Vulkan command refers to the interval between calling the command and its return to the caller.

**Dynamic Storage Buffer**

A storage buffer whose offset is specified each time the storage buffer is bound to a command buffer via a descriptor set.

**Dynamic Uniform Buffer**

A uniform buffer whose offset is specified each time the uniform buffer is bound to a command buffer via a descriptor set.

**Dynamically Uniform**

See *Dynamically Uniform* in section 2.2 “Terms” of the Khronos SPIR-V Specification.

**Encode Input Picture**

A video picture resource used as the input of a video encode operation.

**Element**

Arrays are composed of multiple elements, where each element exists at a unique index within that array. Used primarily to describe data passed to or returned from the Vulkan API.

**Explicitly-Enabled Layer**

A layer enabled by the application by adding it to the enabled layer list in vkCreateInstance or vkCreateDevice.

**Event**

A synchronization primitive that is signaled when execution of previous commands completes through a specified set of pipeline stages. Events can be waited on by the device and polled by the host. Represented by a VkEvent object.

**Executable State (Command Buffer)**

A command buffer that has ended recording commands and **can** be executed. See also Initial State and Recording State.

**Execution Dependency**

A dependency that guarantees that certain pipeline stages’ work for a first set of commands has completed execution before certain pipeline stages’ work for a second set of commands begins

execution. This is accomplished via pipeline barriers, subpass dependencies, events, or implicit ordering operations.

**Execution Dependency Chain**

A sequence of execution dependencies that transitively act as a single execution dependency.

**Explicit chroma reconstruction**

An implementation of sampler $Y'C_BC_R$ conversion which reconstructs reduced-resolution chroma samples to luma resolution and then separately performs texture sample interpolation. This is distinct from an implicit implementation, which incorporates chroma sample reconstruction into texture sample interpolation.

**Extension Scope**

The set of objects and commands that **can** be affected by an extension. Extensions are either device scope or instance scope.

**Extending Structure**

A structure type which may appear in the `pNext` *chain* of another structure, extending the functionality of the other structure. Extending structures may be defined by either core API versions or extensions.

**External Handle**

A resource handle which has meaning outside of a specific Vulkan device or its parent instance. External handles **may** be used to share resources between multiple Vulkan devices in different instances, or between Vulkan and other APIs. Some external handle types correspond to platform-defined handles, in which case the resource **may** outlive any particular Vulkan device or instance and **may** be transferred between processes, or otherwise manipulated via functionality defined by the platform for that handle type.

**External synchronization**

A type of synchronization **required** of the application, where parameters defined to be externally synchronized **must** not be used simultaneously in multiple threads.

**Facingness (Polygon)**

A classification of a polygon as either front-facing or back-facing, depending on the orientation (winding order) of its vertices.

**Facingness (Fragment)**

A fragment is either front-facing or back-facing, depending on the primitive it was generated from. If the primitive was a polygon (regardless of polygon mode), the fragment inherits the facingness of the polygon. All other fragments are front-facing.

**Fence**

A synchronization primitive that is signaled when a set of batches or sparse binding operations complete execution on a queue. Fences **can** be waited on by the host. Represented by a VkFence object.

**Field (Video)**

Possibly discontinuous subregions of a frame. Frames **may** consist of two fields, a top field and a bottom field.

**Flat Shading**

A property of a vertex attribute that causes the value from a single vertex (the provoking vertex) to be used for all vertices in a primitive, and for interpolation of that attribute to return that single value unaltered.

**Format Features**

A set of features from VkFormatFeatureFlagBits that a VkFormat is capable of using for various commands. The list is determined by factors such as VkImageTiling.

**Forward Prediction**

Sample prediction performed during video decode and encode operations based on reference pictures that occur temporally (in display order) before the current picture.

**Forward Reference**

A reference picture that occurs temporally (in display order) before the current picture.

**Fragment**

A rectangular framebuffer region with associated data produced by rasterization and processed by fragment operations including the fragment shader.

**Fragment Area**

The width and height, in pixels, of a fragment.

**Fragment Density**

The ratio of fragments per framebuffer area in the x and y direction.

**Fragment Density Texel Size**

The (w,h) framebuffer region in pixels that each texel in a fragment density map applies to.

**Fragment Input Attachment Interface**

Variables with `UniformConstant` storage class and a decoration of `InputAttachmentIndex` that are statically used by a fragment shader's entry point, which receive values from input attachments.

**Fragment Mask**

A lookup table that associates color samples with color fragment values.

**Fragment Output Interface**

A fragment shader entry point's variables with `Output` storage class, which output to color and/or depth/stencil attachments.

**Frame (Video)**

A multi-dimensional array of luma samples and an optional multi-dimensional array of chroma samples.

**Fragment Tile Image Interface**

A fragment shader entry point's variables with `TileImageEXT` storage class and a decoration of `Location`, which are used to read values from color attachments.

**Framebuffer**

A collection of image views and a set of dimensions that, in conjunction with a render pass, define the inputs and outputs used by drawing commands. Represented by a VkFramebuffer object.

**Framebuffer Attachment**

One of the image views used in a framebuffer.

**Framebuffer Coordinates**

A coordinate system in which adjacent pixels' coordinates differ by 1 in x and/or y, with (0,0) in the upper left corner and pixel centers at half-integers.

**Framebuffer-Space**

Operating with respect to framebuffer coordinates.

**Framebuffer-Local**

A framebuffer-local dependency guarantees that only for a single framebuffer region, the first set of operations happens-before the second set of operations.

**Framebuffer-Global**

A framebuffer-global dependency guarantees that for all framebuffer regions, the first set of operations happens-before the second set of operations.

**Framebuffer Region**

A framebuffer region is a set of sample (x, y, layer, sample) coordinates that is a subset of the entire framebuffer.

**Front-Facing**

See Facingness.

**Full Compatibility**

A given version of the API is fully compatible with another version if an application, relying only on valid behavior and functionality defined by either of those specifications, is able to correctly run against each version without any modification. This assumes no active attempt by that application to not run when it detects a different version.

**Global Command**

A Vulkan command for which the first argument is not a dispatchable handle type.

**Global Workgroup**

A collection of local workgroups dispatched by a single dispatching or single mesh task drawing command.

**Handle**

An opaque integer or pointer value used to refer to a Vulkan object. Each object type has a unique handle type.

**Happen-after, happens-after**

A transitive, irreflexive and antisymmetric ordering relation between operations. An execution dependency with a source of **A** and a destination of **B** enforces that **B** happens-after **A**. The inverse relation of happens-before.

**Happen-before, happens-before**

A transitive, irreflexive and antisymmetric ordering relation between operations. An execution dependency with a source of **A** and a destination of **B** enforces that **A** happens-before **B**. The inverse relation of happens-after.

**Helper Invocation**

A fragment shader invocation that is created solely for the purposes of evaluating derivatives for use in non-helper fragment shader invocations, and which does not have side effects.

**Host**

The processor(s) and execution environment that the application runs on, and that the Vulkan API is exposed on.

**Host Mapped Device Memory**

Device memory that is mapped for host access using vkMapMemory.

**Host Mapped Foreign Memory**

Memory owned by a foreign device that is mapped for host access.

**Host Memory**

Memory not accessible to the device, used to store implementation data structures.

**Host-Accessible Subresource**

A buffer, or a linear image subresource in either the `VK_IMAGE_LAYOUT_PREINITIALIZED` or `VK_IMAGE_LAYOUT_GENERAL` layout. Host-accessible subresources have a well-defined addressing scheme which can be used by the host.

**Host-Local Memory**

Memory that is not local to the device, and **may** be less performant for device access than device-local memory.

**Host-Visible Memory**

Device memory that **can** be mapped on the host and **can** be read and written by the host.

**ICD**

Installable Client Driver. An ICD is represented as a VkPhysicalDevice.

**Identically Defined Objects**

Objects of the same type where all arguments to their creation or allocation functions, with the

exception of `pAllocator`, are

1. Vulkan handles which refer to the same object or
2. identical scalar or enumeration values or
3. Host pointers which point to an array of values or structures which also satisfy these three constraints.

**Image**

A resource that represents a multi-dimensional formatted interpretation of device memory. Represented by a VkImage object.

**Image Subresource**

A specific mipmap level, layer, and set of aspects of an image.

**Image Subresource Range**

A set of image subresources that are contiguous mipmap levels and layers.

**Image View**

An object that represents an image subresource range of a specific image, and state controlling how the contents are interpreted. Represented by a VkImageView object.

**Immutable Sampler**

A sampler descriptor provided at descriptor set layout creation time for a specific binding. This sampler is then used for that binding in all descriptor sets allocated with the layout, and it **cannot** be changed.

**Implicit chroma reconstruction**

An implementation of sampler $Y'C_BC_R$ conversion which reconstructs the reduced-resolution chroma samples directly at the sample point, as part of the normal texture sampling operation. This is distinct from an *explicit chroma reconstruction* implementation, which reconstructs the reduced-resolution chroma samples to the resolution of the luma samples, then filters the result as part of texture sample interpolation.

**Implicitly-Enabled Layer**

A layer enabled by a loader-defined mechanism outside the Vulkan API, rather than explicitly by the application during instance or device creation.

**Inactive Object (Ray Tracing)**

A primitive or instance in a ray tracing acceleration structure which has a corresponding ID, but which will never report an intersection with any ray.

**Index Buffer**

A buffer bound via vkCmdBindIndexBuffer which is the source of index values used to fetch vertex attributes for a vkCmdDrawIndexed or vkCmdDrawIndexedIndirect command.

**Indexed Drawing Commands**

*Drawing commands* which use an *index buffer* as the source of index values used to fetch vertex attributes for a drawing command. Includes vkCmdDrawIndexed,

vkCmdDrawIndexedIndirectCountKHR, vkCmdDrawIndexedIndirectCountAMD, vkCmdDrawMultiIndexedEXT, and vkCmdDrawIndexedIndirect.

**Indirect Commands**

Drawing or dispatching commands that source some of their parameters from structures in buffer memory. Includes vkCmdDrawIndirect, vkCmdDrawIndexedIndirect, vkCmdDrawIndirectCountKHR, vkCmdDrawIndexedIndirectCountKHR, vkCmdDrawIndirectCountAMD, vkCmdDrawIndexedIndirectCountAMD, vkCmdDrawMeshTasksIndirectNV, vkCmdDrawMeshTasksIndirectCountNV, vkCmdDrawMeshTasksIndirectEXT, vkCmdDrawMeshTasksIndirectCountEXT, and vkCmdDispatchIndirect.

**Indirect Commands Layout**

A definition of a sequence of commands, that are generated on the device via vkCmdPreprocessGeneratedCommandsNV and vkCmdExecuteGeneratedCommandsNV. Each sequence is comprised of multiple VkIndirectCommandsTokenTypeNV, which represent a subset of traditional command buffer commands. Represented as VkIndirectCommandsLayoutNV.

**Indirect Drawing Commands**

*Drawing commands* that source some of their parameters from structures in buffer memory. Includes vkCmdDrawIndirect, vkCmdDrawIndirectCountKHR, vkCmdDrawIndexedIndirectCountKHR, vkCmdDrawIndirectCountAMD, vkCmdDrawIndexedIndirectCountAMD, vkCmdDrawMeshTasksIndirectNV, vkCmdDrawMeshTasksIndirectCountNV, vkCmdDrawMeshTasksIndirectEXT, vkCmdDrawMeshTasksIndirectCountEXT, and vkCmdDrawIndexedIndirect.

**Initial State (Command Buffer)**

A command buffer that has not begun recording commands. See also Recording State and Executable State.

**Inline Uniform Block**

A descriptor type that represents uniform data stored directly in descriptor sets, and supports read-only access in a shader.

**Input Attachment**

A descriptor type that represents an image view, and supports unfiltered read-only access in a shader, only at the fragment's location in the view.

**Instance**

The top-level Vulkan object, which represents the application's connection to the implementation. Represented by a VkInstance object.

**Instance-Level Command**

Any command that is dispatched from an instance, or from a child object of an instance, except for physical devices and their children.

**Instance-Level Functionality**

All instance-level commands and objects, and their structures, enumerated types, and

enumerants.

**Instance-Level Object**

High-level Vulkan objects, which are not physical devices, nor children of physical devices. For example, VkInstance is an instance-level object.

**Instance (Memory)**

In a logical device representing more than one physical device, some device memory allocations have the requested amount of memory allocated multiple times, once for each physical device in a device mask. Each such replicated allocation is an instance of the device memory.

**Instance (Resource)**

In a logical device representing more than one physical device, buffer and image resources exist on all physical devices but **can** be bound to memory differently on each. Each such replicated resource is an instance of the resource.

**Inter Block (Video)**

A video coding block that uses inter prediction.

**Inter Prediction (Video)**

Sample prediction derived from reference picture samples or metadata.

**Internal Synchronization**

A type of synchronization **required** of the implementation, where parameters not defined to be externally synchronized **may** require internal mutexing to avoid multithreaded race conditions.

**Intra Block (Video)**

A video coding block that only uses intra prediction.

**Intra Prediction (Video)**

Sample prediction that is not derived from reference picture samples or metadata.

**Invocation (Shader)**

A single execution of an entry point in a SPIR-V module. For example, a single vertex's execution of a vertex shader or a single fragment's execution of a fragment shader.

**Invocation Group**

A set of shader invocations that are executed in parallel and that **must** execute the same control flow path in order for control flow to be considered dynamically uniform.

**Invocation Repack Instruction**

A ray tracing instruction where the implementation **may** change the set of invocations that are executing.

**Join (Deferred Host Operations)**

The act of instructing a thread to participate in the execution of a deferred operation. See Deferred Host Operations.

**Linear Color Attachment**

A color attachment with linear tiling

**Linear Resource**

A resource is *linear* if it is one of the following:

- a VkBuffer
- a VkImage created with `VK_IMAGE_TILING_LINEAR`
- a VkImage created with `VK_IMAGE_TILING_DRM_FORMAT_MODIFIER_EXT` and whose Linux DRM format modifier is `DRM_FORMAT_MOD_LINEAR`
- a VkAccelerationStructureNV

Because a VkAccelerationStructureKHR resource does not have memory bound to it directly, it is considered neither linear nor non-linear. However, the VkBuffer on which a VkAccelerationStructureKHR resource is placed is a linear resource.

A resource is *non-linear* if it is one of the following:

- a VkImage created with `VK_IMAGE_TILING_OPTIMAL`
- a VkImage created with `VK_IMAGE_TILING_DRM_FORMAT_MODIFIER_EXT` and whose Linux DRM format modifier is not `DRM_FORMAT_MOD_LINEAR`

**Linux DRM Format Modifier**

A 64-bit, vendor-prefixed, semi-opaque unsigned integer describing vendor-specific details of an image's memory layout. In Linux graphics APIs, *modifiers* are commonly used to specify the memory layout of externally shared images. An image has a *modifier* if and only if it is created with `tiling` equal to `VK_IMAGE_TILING_DRM_FORMAT_MODIFIER_EXT`. For more details, refer to the appendix for extension `VK_EXT_image_drm_format_modifier`.

**Local Workgroup**

A collection of compute shader invocations invoked by a single dispatching command, which share data via `WorkgroupLocal` variables and can synchronize with each other.

**Logical Device**

An object that represents the application's interface to the physical device. The logical device is the parent of most Vulkan objects. Represented by a VkDevice object.

**Logical Operation**

Bitwise operations between a fragment color value and a value in a color attachment, that produce a final color value to be written to the attachment.

**Lost Device**

A state that a logical device **may** be in as a result of unrecoverable implementation errors, or other exceptional conditions.

**Mappable**

See Host-Visible Memory.

**Memory Dependency**

A memory dependency is an execution dependency which includes availability and visibility operations such that:

- The first set of operations happens-before the availability operation
- The availability operation happens-before the visibility operation
- The visibility operation happens-before the second set of operations

**Memory Domain**

A memory domain is an abstract place to which memory writes are made available by availability operations and memory domain operations. The memory domains correspond to the set of agents that the write **can** then be made visible to. The memory domains are *host*, *device*, *shader*, *workgroup instance* (for workgroup instance there is a unique domain for each compute workgroup) and *subgroup instance* (for subgroup instance there is a unique domain for each subgroup).

**Memory Domain Operation**

An operation that makes the writes that are available to one memory domain available to another memory domain.

**Memory Heap**

A region of memory from which device memory allocations **can** be made.

**Memory Type**

An index used to select a set of memory properties (e.g. mappable, cached) for a device memory allocation.

**Mesh Shading Pipeline**

A graphics pipeline where the primitives are assembled explicitly in the shader stages. In contrast to the primitive shading pipeline where input primitives are assembled by fixed function processing.

**Mesh Tasks Drawing Commands**

*Drawing commands* which create shader invocations organized in workgroups for drawing mesh tasks. Includes vkCmdDrawMeshTasksNV, vkCmdDrawMeshTasksIndirectNV, and vkCmdDrawMeshTasksIndirectCountNV, vkCmdDrawMeshTasksEXT, vkCmdDrawMeshTasksIndirectEXT, and vkCmdDrawMeshTasksIndirectCountEXT .

**Minimum Mip Level Size**

The smallest size that is permitted for a mip level. For conventional images this is 1x1x1. For corner-sampled images, this is 2x2x2. See Image Mip Level Sizing.

**Mip Tail Region**

The set of mipmap levels of a sparse residency texture that are too small to fill a sparse block, and that **must** all be bound to memory collectively and opaquely.

**Multi-planar**

A *multi-planar format* (or “planar format”) is an image format consisting of more than one *plane*,

identifiable with a `_2PLANE` or `_3PLANE` component to the format name and listed in Formats requiring sampler $Y'C_BC_R$ conversion for `VK_IMAGE_ASPECT_COLOR_BIT` image views. A *multi-planar image* (or "planar image") is an image of a multi-planar format.

**Nested Command Buffers**

A nested command buffer is a secondary command buffer that is executed by another secondary command buffer, which may itself execute other secondary command buffers.

**Non-Dispatchable Handle**

A handle of an integer handle type. Handle values **may** not be unique, even for two objects of the same type.

**Non-Indexed Drawing Commands**

*Drawing commands* for which the vertex attributes are sourced in linear order from the vertex input attributes for a drawing command (i.e. they do not use an *index buffer*). Includes vkCmdDraw, vkCmdDrawIndirectCountKHR, vkCmdDrawIndirectCountAMD, vkCmdDrawMultiEXT, and vkCmdDrawIndirect.

**Normalized**

A value that is interpreted as being in the range [0,1] as a result of being implicitly divided by some other value.

**Normalized Device Coordinates**

A coordinate space after perspective division is applied to clip coordinates, and before the viewport transformation converts them to framebuffer coordinates.

**Obsoleted (feature)**

A feature is obsolete if it can no longer be used.

**Opaque Capture Address**

A 64-bit value representing the device address of a buffer or memory object that is expected to be used by trace capture/replay tools in combination with the `bufferDeviceAddress` feature.

**Overlapped Range (Aliased Range)**

The aliased range of a device memory allocation that intersects a given image subresource of an image or range of a buffer.

**Ownership (Resource)**

If an entity (e.g. a queue family) has ownership of a resource, access to that resource is well-defined for access by that entity.

**Packed Format**

A format whose components are stored as a single texel block in memory, with their relative locations defined within that element.

**Passthrough Geometry Shader**

A geometry shader which uses the `PassthroughNV` decoration on a variable in its input interface. Output primitives in a passthrough geometry shader always have the same topology as the input

primitive and are not produced by emitting vertices.

**Payload**

Importable or exportable reference to the internal data of an object in Vulkan.

**Per-View**

A variable that has an array of values which are output, one for each view that is being generated. A mesh shader which uses the `PerViewNV` decoration on a variable in its output interface.

**Peer Memory**

An instance of memory corresponding to a different physical device than the physical device performing the memory access, in a logical device that represents multiple physical devices.

**Physical Device**

An object that represents a single device in the system. Represented by a VkPhysicalDevice object.

**Physical-Device-Level Command**

Any command that is dispatched from a physical device.

**Physical-Device-Level Functionality**

All physical-device-level commands and objects, and their structures, enumerated types, and enumerants.

**Physical-Device-Level Object**

Physical device objects. For example, VkPhysicalDevice is a physical-device-level object.

**Picture Partition (Video)**

Collective term that refers to a subregion of a video picture resource that **may** be decoded or encoded independently according to the partitioning scheme defined in the corresponding video compression standard. Picture partitions **may** be non-rectangular.

**Pipeline**

An object controlling how graphics or compute work is executed on the device. A pipeline includes one or more shaders, as well as state controlling any non-programmable stages of the pipeline. Represented by a VkPipeline object.

**Pipeline Barrier**

An execution and/or memory dependency recorded as an explicit command in a command buffer, that forms a dependency between the previous and subsequent commands.

**Pipeline Cache**

An object that **can** be used to collect and retrieve information from pipelines as they are created, and **can** be populated with previously retrieved information in order to accelerate pipeline creation. Represented by a VkPipelineCache object.

**Pipeline Layout**

An object defining the set of resources (via a collection of descriptor set layouts) and push constants used by pipelines that are created using the layout. Used when creating a pipeline and when binding descriptor sets and setting push constant values. Represented by a VkPipelineLayout object.

**Pipeline Library**

A pipeline that cannot be directly used, instead defining a set of shaders and shader groups which will be linked into other pipelines.

**Pipeline Stage**

A logically independent execution unit that performs some of the operations defined by an action command.

**Pipeline Trace Ray Instruction**

A ray tracing instruction which traces a ray into an acceleration structure when using ray tracing pipelines. One of:

- `OpTraceNV`
- `OpTraceRayKHR`
- `OpTraceRayMotionNV`
- `OpTraceMotionNV`
- `OpHitObjectTraceRayNV`
- `OpHitObjectTraceRayMotionNV`

**`pNext` Chain**

A set of structures chained together through their `pNext` members.

**Planar**

See *multi-planar*.

**Plane**

An *image plane* is part of the representation of an image, containing a subset of the color components necessary to represent the texels in the image and with a contiguous mapping of coordinates to bound memory. Most images consist only of a single plane, but some formats spread the components across multiple image planes. The host-accessible properties of each image plane are accessible for a linear layout using vkGetImageSubresourceLayout. If a multi-planar image is created with the `VK_IMAGE_CREATE_DISJOINT_BIT` bit set, the image is described as *disjoint*, and its planes are therefore bound to memory independently.

**Point Sampling (Rasterization)**

A rule that determines whether a fragment sample location is covered by a polygon primitive by testing whether the sample location is in the interior of the polygon in framebuffer-space, or on the boundary of the polygon according to the tie-breaking rules.

**Potential Format Features**

The union of all VkFormatFeatureFlagBits that the implementation supports for a specified VkFormat, over all supported image tilings. For Android external formats the VkFormatFeatureFlagBits is provided by the implementation. For QNX Screen external formats the VkFormatFeatureFlagBits is provided by the implementation.

**Pre-rasterization**

Operations that execute before rasterization, and any state associated with those operations.

**Presentable image**

A `VkImage` object obtained from a `VkSwapchainKHR` used to present to a `VkSurfaceKHR` object.

**Preserve Attachment**

One of a list of attachments in a subpass description that is not read or written by the subpass, but that is read or written on earlier and later subpasses and whose contents **must** be preserved through this subpass.

**Primary Command Buffer**

A command buffer that **can** execute secondary command buffers, and **can** be submitted directly to a queue.

**Primitive Shading Pipeline**

A graphics pipeline where input primitives are assembled by fixed function processing. It is the counterpart to mesh shading.

**Primitive Topology**

State controlling how vertices are assembled into primitives, e.g. as lists of triangles, strips of lines, etc.

**Promoted (feature)**

A feature from an older extension is considered promoted if it is made available as part of a new core version or newer extension with wider support.

**Protected Buffer**

A buffer to which protected device memory **can** be bound.

**Protected-capable Device Queue**

A device queue to which protected command buffers **can** be submitted.

**Protected Command Buffer**

A command buffer which **can** be submitted to a protected-capable device queue.

**Protected Device Memory**

Device memory which **can** be visible to the device but **must** not be visible to the host.

**Protected Image**

An image to which protected device memory **can** be bound.

**Provisional**

A feature is released provisionally in order to get wider feedback on the functionality before it is finalized. Provisional features may change in ways that break backwards compatibility, and thus are not recommended for use in production applications.

**Provoking Vertex**

The vertex in a primitive from which flat shaded attribute values are taken. This is generally the "first" vertex in the primitive, and depends on the primitive topology.

**Push Constants**

A small bank of values writable via the API and accessible in shaders. Push constants allow the application to set values used in shaders without creating buffers or modifying and binding descriptor sets for each update.

**Push Constant Interface**

The set of variables with `PushConstant` storage class that are statically used by a shader entry point, and which receive values from push constant commands.

**Push Descriptors**

Descriptors that are written directly into a command buffer rather than into a descriptor set. Push descriptors allow the application to set descriptors used in shaders without allocating or modifying descriptor sets for each update.

**Descriptor Update Template**

An object specifying a mapping from descriptor update information in host memory to elements in a descriptor set, which helps enable more efficient descriptor set updates.

**Query Pool**

An object containing a number of query entries and their associated state and results. Represented by a VkQueryPool object.

**Queue**

An object that executes command buffers and sparse binding operations on a device. Represented by a VkQueue object.

**Queue Family**

A set of queues that have common properties and support the same functionality, as advertised in VkQueueFamilyProperties.

**Queue Operation**

A unit of work to be executed by a specific queue on a device, submitted via a queue submission command. Each queue submission command details the specific queue operations that occur as a result of calling that command. Queue operations typically include work that is specific to each command, and synchronization tasks.

**Queue Submission**

Zero or more batches and an optional fence to be signaled, passed to a command for execution on a queue. See the Devices and Queues chapter for more information.

**Ray Tracing Command**

Commands that provoke work using a ray tracing pipeline. Includes vkCmdTraceRaysNV, vkCmdTraceRaysKHR, and vkCmdTraceRaysIndirectKHR .

**Reconstructed Picture**

A video picture resource reconstructed from a compressed bitstream using video decode or encode operations that **can** be used as a reference picture by future video decode or encode operations with the same video session.

**Recording State (Command Buffer)**

A command buffer that is ready to record commands. See also Initial State and Executable State.

**Reference Picture**

A video picture resource used by video decode and encode operations to provide predictions of the values of samples in the subsequently decoded or encoded pictures.

**Reference Picture Metadata**

Opaque state associated with a DPB slot, maintained by a video session.

**Release Operation (Resource)**

An operation that releases ownership of an image subresource or buffer range.

**Render Pass**

An object that represents a set of framebuffer attachments and phases of rendering using those attachments. Represented by a VkRenderPass object.

**Render Pass Instance**

A use of a render pass in a command buffer.

**Required Extensions**

Extensions that **must** be enabled alongside extensions dependent on them (see Extension Dependencies).

**Reset (Command Buffer)**

Resetting a command buffer discards any previously recorded commands and puts a command buffer in the initial state.

**Residency Code**

An integer value returned by sparse image instructions, indicating whether any sparse unbound texels were accessed.

**Resolve Attachment**

A subpass attachment point, or image view, that is the target of a multisample resolve operation from the corresponding color attachment at the end of the subpass.

**Retired Swapchain**

A swapchain that has been used as the `oldSwapchain` parameter to vkCreateSwapchainKHR. Images cannot be acquired from a retired swapchain, however images that were acquired (but

not presented) before the swapchain was retired **can** be presented.

**Sample Index**

The index of a sample within a single set of samples.

**Sample Shading**

Invoking the fragment shader multiple times per fragment, with the covered samples partitioned among the invocations.

**Sampled Image**

A descriptor type that represents an image view, and supports filtered (sampled) and unfiltered read-only access in a shader.

**Sampler**

An object containing state controlling how sampled image data is sampled (or filtered) when accessed in a shader. Also a descriptor type describing the object. Represented by a VkSampler object.

**Secondary Command Buffer**

A command buffer that **can** be executed by a primary command buffer, and **must** not be submitted directly to a queue.

**Self-Dependency**

A subpass dependency from a subpass to itself, i.e. with `srcSubpass` equal to `dstSubpass`. A self-dependency is not automatically performed during a render pass instance, rather a subset of it **can** be performed via vkCmdPipelineBarrier during the subpass.

**Semaphore**

A synchronization primitive that supports signal and wait operations, and **can** be used to synchronize operations within a queue or across queues. Represented by a VkSemaphore object.

**Shader**

Instructions selected (via an entry point) from a shader module, which are executed in a shader stage.

**Shader Call**

An instruction which **may** cause execution to continue in a different shader stage.

**Shader Code**

A stream of instructions used to describe the operation of a shader.

**Shader Group**

A set of Shader Stages that are part of a VkPipeline containing multiple of such sets. This allows the device to make use of all the shader groups from the bound pipeline independently.

**Shader Module**

A collection of shader code, potentially including several functions and entry points, that is used to create shaders in pipelines. Represented by a VkShaderModule object.

**Shader Stage**

A stage of the graphics or compute pipeline that executes shader code.

**Shading Rate**

The ratio of the number of fragment shader invocations generated in a fully covered framebuffer region to the size (in pixels) of that region.

**Shading Rate Image**

An image used to establish the shading rate for a framebuffer region, where each pixel controls the shading rate for a corresponding framebuffer region.

**Shared presentable image**

A presentable image created from a swapchain with VkPresentModeKHR set to either `VK_PRESENT_MODE_SHARED_DEMAND_REFRESH_KHR` or `VK_PRESENT_MODE_SHARED_CONTINUOUS_REFRESH_KHR`.

**Side Effect**

A store to memory or atomic operation on memory from a shader invocation.

**Single-plane format**

A format that is not *multi-planar*.

**Size-Compatible Image Formats**

When a compressed image format and an uncompressed image format are size-compatible, it means that the texel block size of the uncompressed format **must** equal the texel block size of the compressed format.

**Sparse Block**

An element of a sparse resource that can be independently bound to memory. Sparse blocks of a particular sparse resource have a corresponding size in bytes that they use in the bound memory.

**Sparse Image Block**

A sparse block in a sparse partially-resident image. In addition to the sparse block size in bytes, sparse image blocks have a corresponding width, height, and depth defining the dimensions of these elements in units of texels or compressed texel blocks, the latter being used in case of sparse images having a block-compressed format.

**Sparse Unbound Texel**

A texel read from a region of a sparse texture that does not have memory bound to it.

**SRT**

A decomposition of a spatial transform separating out scale, rotation, and translation which has better linear interpolation properties for representing motion.

**Static Use**

An object in a shader is statically used by a shader entry point if any function in the entry point's call tree contains an instruction using the object. A reference in the entry point's interface list does not constitute a static use. Static use is used to constrain the set of descriptors used by a

shader entry point.

**Storage Buffer**

A descriptor type that represents a buffer, and supports reads, writes, and atomics in a shader.

**Storage Image**

A descriptor type that represents an image view, and supports unfiltered loads, stores, and atomics in a shader.

**Storage Texel Buffer**

A descriptor type that represents a buffer view, and supports unfiltered, formatted reads, writes, and atomics in a shader.

**Subgroup**

A set of shader invocations that **can** synchronize and share data with each other efficiently. In compute shaders, the *local workgroup* is a superset of the subgroup.

**Subgroup Mask**

A bitmask for all invocations in the current subgroup with one bit per invocation, starting with the least significant bit in the first vector component, continuing to the last bit (less than `SubgroupSize`) in the last required vector component.

**Subpass**

A phase of rendering within a render pass, that reads and writes a subset of the attachments.

**Subpass Dependency**

An execution and/or memory dependency between two subpasses described as part of render pass creation, and automatically performed between subpasses in a render pass instance. A subpass dependency limits the overlap of execution of the pair of subpasses, and **can** provide guarantees of memory coherence between accesses in the subpasses.

**Subpass Description**

Lists of attachment indices for input attachments, color attachments, depth/stencil attachment, resolve attachments, depth/stencil resolve, and preserve attachments used by the subpass in a render pass.

**Subset (Self-Dependency)**

A subset of a self-dependency is a pipeline barrier performed during the subpass of the self-dependency, and whose stage masks and access masks each contain a subset of the bits set in the identically named mask in the self-dependency.

**Texel Block**

A single addressable element of an image with an uncompressed VkFormat, or a single compressed block of an image with a compressed VkFormat.

**Texel Block Size**

The size (in bytes) used to store a texel block of a compressed or uncompressed image.

**Texel Coordinate System**

One of three coordinate systems (normalized, unnormalized, integer) defining how texel coordinates are interpreted in an image or a specific mipmap level of an image.

**Tile Image**

A per-tile view of a framebuffer attachment. If the `VK_EXT_shader_tile_image` extension is enabled, the framebuffer is considered to be divided into tiles.

**Timeline Semaphore**

A semaphore with a strictly increasing 64-bit unsigned integer payload indicating whether the semaphore is signaled with respect to a particular reference value. Represented by a VkSemaphore object created with a semaphore type of `VK_SEMAPHORE_TYPE_TIMELINE`.

**Uniform Texel Buffer**

A descriptor type that represents a buffer view, and supports unfiltered, formatted, read-only access in a shader.

**Uniform Buffer**

A descriptor type that represents a buffer, and supports read-only access in a shader.

**Units in the Last Place (ULP)**

A measure of floating-point error loosely defined as the smallest representable step in a floating-point format near a given value. For the precise definition see Precision and Operation of SPIR-V instructions or Jean-Michel Muller, “On the definition of ulp(x)”, RR-5504, INRIA. Other sources may also use the term “unit of least precision”.

**Unnormalized**

A value that is interpreted according to its conventional interpretation, and is not normalized.

**Unprotected Buffer**

A buffer to which unprotected device memory **can** be bound.

**Unprotected Command Buffer**

A command buffer which **can** be submitted to an unprotected device queue or a protected-capable device queue.

**Unprotected Device Memory**

Device memory which **can** be visible to the device and **can** be visible to the host.

**Unprotected Image**

An image to which unprotected device memory **can** be bound.

**User-Defined Variable Interface**

A shader entry point’s variables with `Input` or `Output` storage class that are not built-in variables.

**Vertex Input Attribute**

A graphics pipeline resource that produces input values for the vertex shader by reading data from a vertex input binding and converting it to the attribute’s format.

**Vertex Stream**

A vertex stream is where the last pre-rasterization shader stages outputs vertex data, which then goes to the rasterizer, is captured to a transform feedback buffer, or both. Geometry shaders **can** emit primitives to multiple independent vertex streams. Each vertex emitted by the geometry shader is directed at one of the vertex streams.

**Validation Cache**

An object that **can** be used to collect and retrieve validation results from the validation layers, and **can** be populated with previously retrieved results in order to accelerate the validation process. Represented by a VkValidationCacheEXT object.

**Variable-Sized Descriptor Binding**

A descriptor binding whose size will be specified when a descriptor set is allocated using this layout.

**Vertex Input Binding**

A graphics pipeline resource that is bound to a buffer and includes state that affects addressing calculations within that buffer.

**Vertex Input Interface**

A vertex shader entry point's variables with `Input` storage class, which receive values from vertex input attributes.

**Video Bitstream Buffer**

A resource that represents a linear array of data in device memory storing encoded video data. Represented by a VkBuffer object.

**Video Coding Scope**

A series of subsequent commands recorded into a command buffer starting with a vkCmdBeginVideoCodingKHR command and ending with a vkCmdEndVideoCodingKHR command that encompasses a set of video decode or encode operations.

**Video Coding Operations**

Any operations recorded into a command buffer within a video coding scope, including video decode and encode operations.

**Video Decode Operation**

An operation consuming data from a video bitstream buffer and zero or more reference pictures, and producing data to a decode output picture and an optional reconstructed picture.

**Video Encode Operation**

An operation consuming data from an encode input picture and zero or more reference pictures, and producing data to a video bitstream buffer and an optional reconstructed picture.

**Video Picture Resource**

A resource that represents a multi-dimensional formatted interpretation of device memory to be used with a video session as a decode output picture, encode input picture, reconstructed picture, and/or reference picture. It **may** contain metadata associated with a particular video

session it is used with. Represented by a VkImage object and referred to using VkImageView objects created from it.

**Video Session**

A resource that represents and maintains the state needed to perform video decode or encode operations. Represented by a VkVideoSessionKHR object.

**Video Session Parameters**

A resource that stores preprocessed codec-specific parameters used with a compatible video session in video codec operations. Represented by a VkVideoSessionParametersKHR object.

**Video Transcoding**

The process of using the outputs of video decoding operations as inputs in video encoding operations.

**View Mask**

When multiview is enabled, a view mask is a property of a subpass controlling which views the rendering commands are broadcast to.

**View Volume**

A subspace in homogeneous coordinates, corresponding to post-projection x and y values between -1 and +1, and z values between 0 and +1.

**Viewport Transformation**

A transformation from normalized device coordinates to framebuffer coordinates, based on a viewport rectangle and depth range.

**Visibility Operation**

An operation that causes available values to become visible to specified memory accesses.

**Visible**

A state of values written to memory that allows them to be accessed by a set of operations.

# Common Abbreviations

The abbreviations and acronyms defined in this section are sometimes used in the Specification and the API where they are considered clear and commonplace.

**Src**

Source

**Dst**

Destination

**Min**

Minimum

**Max**

Maximum

**Rect**

Rectangle

**Info**

Information

**LOD**

Level of Detail

**Log**

Logarithm

**ID**

Identifier

**UUID**

Universally Unique Identifier

**Op**

Operation

**R**

Red color component

**G**

Green color component

**B**

Blue color component

**A**

Alpha color component

**RTZ**

Round towards zero

**RTE**

Round to nearest even

# Video-Specific Abbreviations

The following abbreviations and acronyms are used in the context of video decode and encode operations to refer to commonly used video compression terms in their usual abbreviated form:

**AVC**

Advanced Video Coding

**Bipred**

Bidirectional Prediction

**CABAC**

Context-Adaptive Binary Arithmetic Coding

**CAVLC**

Context-Adaptive Variable-Length Coding

**CBR**

Constant Bit Rate

**CTB**

Coding Tree Block

**Diff**

Difference

**DPB**

Decoded Picture Buffer

**GOP**

Group Of Pictures

**HDR**

High Dynamic Range

**HEVC**

High Efficiency Video Coding

**HRD**

Hypothetical Reference Decoder

**IDC**

Indicator

**IDR**

Instantaneous Decoder Refresh

**MB**

Macroblock

**MV**

Motion Vector

**NALU**

Network Abstraction Layer Unit

**OBU**

Open Bitstream Unit

**PCM**

Pulse-Code Modulation

**Pic**

Picture

**Pred**

Prediction

**PPS**

Picture Parameter Set

**QP**

Quantization Parameter

**RC**

Rate Control

**SPS**

Sequence Parameter Set

**Std**

Standard

**VBR**

Variable Bit Rate

**VCL**

Video Coding Layer

**VPS**

Video Parameter Set

# Prefixes

Prefixes are used in the API to denote specific semantic meaning of Vulkan names, or as a label to avoid name clashes, and are explained here:

**VK/Vk/vk**

Vulkan namespace
All types, commands, enumerants and defines in this specification are prefixed with these two characters.

**PFN/pfn**

Function Pointer
Denotes that a type is a function pointer, or that a variable is of a pointer type.

**p**

Pointer
Variable is a pointer.

**vkCmd**

Commands that record commands in command buffers
These API commands do not result in immediate processing on the device. Instead, they record the requested action in a command buffer for execution when the command buffer is submitted to a queue.

**s**

Structure
Used to denote the `VK_STRUCTURE_TYPE*` member of each structure in `sType`

# Appendix I: Credits (Informative)

Vulkan 1.3 is the result of contributions from many people and companies participating in the Khronos Vulkan Working Group, as well as input from the Vulkan Advisory Panel.

Members of the Working Group, including the company that they represented at the time of their most recent contribution, are listed in the following section. Some specific contributions made by individuals are listed together with their name.

## Working Group Contributors to Vulkan

- Aaron Greig, Codeplay Software Ltd. (version 1.1)
- Aaron Hagan, AMD (version 1.1)
- Adam Jackson, Red Hat (versions 1.0, 1.1)
- Adam Śmigielski, Mobica (version 1.0)
- Aditi Verma, Qualcomm (version 1.3)
- Ahmed Abdelkhalek, AMD (version 1.3)
- Aidan Fabius, Core Avionics & Industrial Inc. (version 1.2)
- Alan Baker, Google (versions 1.1, 1.2, 1.3)
- Alan Ward, Google (versions 1.1, 1.2)
- Alejandro Piñeiro, Igalia (version 1.1)
- Alex Bourd, Qualcomm Technologies, Inc. (versions 1.0, 1.1)
- Alex Crabb, Caster Communications (versions 1.2, 1.3)
- Alex Walters, Imagination Technologies (versions 1.2, 1.3)
- Alexander Galazin, Arm (versions 1.0, 1.1, 1.2, 1.3)
- Alexey Sachkov, Intel (version 1.3)
- Allan MacKinnon, Google (version 1.3)
- Allen Hux, Intel (version 1.0)
- Alon Or-bach, Google (versions 1.0, 1.1, 1.2, 1.3) (WSI technical sub-group chair)
- Anastasia Stulova, Arm (versions 1.2, 1.3)
- Andreas Vasilakis, Think Silicon (version 1.2)
- Andres Gomez, Igalia (version 1.1)
- Andrew Cox, Samsung Electronics (version 1.0)
- Andrew Ellem, Google (version 1.3)
- Andrew Garrard, Imagination Technologies (versions 1.0, 1.1, 1.2, 1.3) (format wrangler)
- Andrew Poole, Samsung Electronics (version 1.0)
- Andrew Rafter, Samsung Electronics (version 1.0)

- Andrew Richards, Codeplay Software Ltd. (version 1.0)
- Andrew Woloszyn, Google (versions 1.0, 1.1)
- Ann Thorsnes, Khronos (versions 1.2, 1.3)
- Antoine Labour, Google (versions 1.0, 1.1)
- Aras Pranckevičius, Unity Technologies (version 1.0)
- Arseny Kapoulkine, Roblox (version 1.3)
- Ashwin Kolhe, NVIDIA (version 1.0)
- Baldur Karlsson, Valve Software (versions 1.1, 1.2, 1.3)
- Barthold Lichtenbelt, NVIDIA (version 1.1)
- Bas Nieuwenhuizen, Google (versions 1.1, 1.2)
- Ben Bowman, Imagination Technologies (version 1.0)
- Benj Lipchak, Unknown (version 1.0)
- Bill Hollings, Brenwill (versions 1.0, 1.1, 1.2, 1.3)
- Bill Licea-Kane, Qualcomm Technologies, Inc. (versions 1.0, 1.1)
- Blaine Kohl, Khronos (versions 1.2, 1.3)
- Bob Fraser, Google (version 1.3)
- Boris Zanin, Mobica (versions 1.2, 1.3)
- Brent E. Insko, Intel (version 1.0)
- Brian Ellis, Qualcomm Technologies, Inc. (version 1.0)
- Brian Paul, VMware (versions 1.2, 1.3)
- Caio Marcelo de Oliveira Filho, Intel (versions 1.2, 1.3)
- Cass Everitt, Oculus VR (versions 1.0, 1.1)
- Cemil Azizoglu, Canonical (version 1.0)
- Lina Versace, Google (versions 1.0, 1.1, 1.2)
- Chang-Hyo Yu, Samsung Electronics (version 1.0)
- Charles Giessen, LunarG (version 1.3)
- Chia-I Wu, LunarG (version 1.0)
- Chris Frascati, Qualcomm Technologies, Inc. (version 1.0)
- Chris Glover, Google (version 1.3)
- Christian Forfang, Arm (version 1.3)
- Christoph Kubisch, NVIDIA (version 1.3)
- Christophe Riccio, Unity Technologies (versions 1.0, 1.1)
- Cody Northrop, LunarG (version 1.0)
- Colin Riley, AMD (version 1.1)
- Cort Stratton, Google (versions 1.1, 1.2)

- Courtney Goeltzenleuchter, Google (versions 1.0, 1.1, 1.3)
- Craig Davies, Huawei (version 1.2)
- Dae Kim, Imagination Technologies (version 1.1)
- Damien Leone, NVIDIA (version 1.0)
- Dan Baker, Oxide Games (versions 1.0, 1.1)
- Dan Ginsburg, Valve Software (versions 1.0, 1.1, 1.2, 1.3)
- Daniel Johnston, Intel (versions 1.0, 1.1)
- Daniel Koch, NVIDIA (versions 1.0, 1.1, 1.2, 1.3)
- Daniel Rakos, AMD (versions 1.0, 1.1, 1.2, 1.3)
- Daniel Stone, Collabora (versions 1.1, 1.2)
- Daniel Vetter, Intel (version 1.2)
- David Airlie, Red Hat (versions 1.0, 1.1, 1.2, 1.3)
- David Mao, AMD (versions 1.0, 1.2)
- David Miller, Miller & Mattson (versions 1.0, 1.1) (Vulkan reference card)
- David Neto, Google (versions 1.0, 1.1, 1.2, 1.3)
- David Pankratz, Huawei (version 1.3)
- David Wilkinson, AMD (version 1.2)
- David Yu, Pixar (version 1.0)
- Dejan Mircevski, Google (version 1.1)
- Diego Novillo, Google (version 1.3)
- Dimitris Georgakakis, Think Silicon (version 1.3)
- Dominik Witczak, AMD (versions 1.0, 1.1, 1.3)
- Donald Scorgie, Imagination Technologies (version 1.2)
- Dzmitry Malyshau, Mozilla (versions 1.1, 1.2, 1.3)
- Ed Hutchins, Oculus (version 1.2)
- Emily Stearns, Khronos (versions 1.2, 1.3)
- François Duranleau, Gameloft (version 1.3)
- Frank (LingJun) Chen, Qualcomm Technologies, Inc. (version 1.0)
- Fred Liao, Mediatek (version 1.0)
- Gabe Dagani, Freescale (version 1.0)
- Gabor Sines, AMD (version 1.2)
- Graeme Leese, Broadcom (versions 1.0, 1.1, 1.2, 1.3)
- Graham Connor, Imagination Technologies (version 1.0)
- Graham Sellers, AMD (versions 1.0, 1.1)
- Graham Wihlidal, Electronic Arts (version 1.3)

- Greg Fischer, LunarG (version 1.1)
- Gregory Grebe, AMD (version 1.3)
- Hai Nguyen, Google (versions 1.2, 1.3)
- Hans-Kristian Arntzen, Valve Software (versions 1.1, 1.2, 1.3)
- Henri Verbeet, Codeweavers (version 1.2)
- Wyvern Wang, Huawei (version 1.3)
- Hwanyong Lee, Kyungpook National University (version 1.0)
- Iago Toral, Igalia (versions 1.1, 1.2)
- Ian Elliott, Google (versions 1.0, 1.1, 1.2)
- Ian Romanick, Intel (versions 1.0, 1.1, 1.3)
- Ivan Briano, Intel (version 1.3)
- James Fitzpatrick, Imagination (version 1.3)
- James Hughes, Oculus VR (version 1.0)
- James Jones, NVIDIA (versions 1.0, 1.1, 1.2, 1.3)
- James Riordon, Khronos (versions 1.2, 1.3)
- Jamie Madill, Google (version 1.3)
- Jan Hermes, Continental Corporation (versions 1.0, 1.1)
- Jan-Harald Fredriksen, Arm (versions 1.0, 1.1, 1.2, 1.3)
- Faith Ekstrand, Intel (versions 1.0, 1.1, 1.2, 1.3)
- Jean-François Roy, Google (versions 1.1, 1.2, 1.3)
- Jeff Bolz, NVIDIA (versions 1.0, 1.1, 1.2, 1.3)
- Jeff Juliano, NVIDIA (versions 1.0, 1.1, 1.2)
- Jeff Leger, Qualcomm Technologies, Inc. (versions 1.1, 1.3)
- Jeff Phillips, Khronos (version 1.3)
- Jeff Vigil, Samsung Electronics (versions 1.0, 1.1, 1.2, 1.3)
- Jens Owen, Google (versions 1.0, 1.1)
- Jeremy Hayes, LunarG (version 1.0)
- Jesse Barker, Unity Technologies (versions 1.0, 1.1, 1.2, 1.3)
- Jesse Hall, Google (versions 1.0, 1.1, 1.2, 1.3)
- Joe Davis, Samsung Electronics (version 1.1)
- Johannes van Waveren, Oculus VR (versions 1.0, 1.1)
- John Anthony, Arm (version 1.2, 1.3)
- John Kessenich, Google (versions 1.0, 1.1, 1.2, 1.3) (SPIR-V and GLSL for Vulkan spec author)
- John McDonald, Valve Software (versions 1.0, 1.1, 1.2, 1.3)
- John Zulauf, LunarG (versions 1.1, 1.2, 1.3)

- Jon Ashburn, LunarG (version 1.0)
- Jon Leech, Independent (versions 1.0, 1.1, 1.2, 1.3) (XML toolchain, normative language, release wrangler)
- Jonas Gustavsson, Samsung Electronics (versions 1.0, 1.1)
- Jonas Meyer, Epic Games (versions 1.2, 1.3)
- Jonathan Hamilton, Imagination Technologies (version 1.0)
- Jordan Justen, Intel (version 1.1)
- Joshua Ashton, Valve Software (version 1.3)
- Jungwoo Kim, Samsung Electronics (versions 1.0, 1.1)
- Jörg Wagner, Arm (version 1.1)
- Kalle Raita, Google (version 1.1)
- Karen Ghavam, LunarG (versions 1.1, 1.2, 1.3)
- Karl Schultz, LunarG (versions 1.1, 1.2)
- Kathleen Mattson, Khronos (versions 1.0, 1.1, 1.2)
- Kaye Mason, Google (version 1.2)
- Keith Packard, Valve (version 1.2)
- Kenneth Benzie, Codeplay Software Ltd. (versions 1.0, 1.1)
- Kenneth Russell, Google (version 1.1)
- Kerch Holt, NVIDIA (versions 1.0, 1.1)
- Kevin O’Neil, AMD (version 1.1)
- Kevin Petit, Arm (version 1.3)
- Kris Rose, Khronos (versions 1.2, 1.3)
- Kristian Kristensen, Intel (versions 1.0, 1.1)
- Krzysztof Iwanicki, Samsung Electronics (version 1.0)
- Larry Seiler, Intel (version 1.0)
- Laura Shubel, Caster Communications (version 1.3)
- Lauri Ilola, Nokia (version 1.1)
- Lei Zhang, Google (version 1.2)
- Lenny Komow, LunarG (versions 1.1, 1.2)
- Liam Middlebrook, NVIDIA (version 1.3)
- Lionel Landwerlin, Intel (versions 1.1, 1.2)
- Lisie Aartsen, Khronos (version 1.3)
- Liz Maitral, Khronos (version 1.2)
- Lou Kramer, AMD (version 1.3)
- Lutz Latta, Lucasfilm (version 1.0)

- Maciej Jesionowski, AMD (version 1.1)
- Mais Alnasser, AMD (version 1.1)
- Marcin Kantoch, AMD (version 1.3)
- Marcin Rogucki, Mobica (version 1.1)
- Maria Rovatsou, Codeplay Software Ltd. (version 1.0)
- Mariusz Merecki, Intel (version 1.3)
- Mark Bellamy, Arm (version 1.2, 1.3)
- Mark Callow, Independent (versions 1.0, 1.1, 1.2, 1.3)
- Mark Kilgard, NVIDIA (versions 1.1, 1.2)
- Mark Lobodzinski, LunarG (versions 1.0, 1.1, 1.2)
- Mark Young, LunarG (versions 1.1, 1.3)
- Markus Tavenrath, NVIDIA (version 1.1)
- Marty Johnson, Khronos (version 1.3)
- Mateusz Przybylski, Intel (version 1.0)
- Mathias Heyer, NVIDIA (versions 1.0, 1.1)
- Mathias Schott, NVIDIA (versions 1.0, 1.1)
- Mathieu Robart, Arm (version 1.2)
- Matt Netsch, Qualcomm Technologies, Inc. (version 1.1)
- Matthew Rusch, NVIDIA (version 1.3)
- Matthäus Chajdas, AMD (versions 1.1, 1.2, 1.3)
- Maurice Ribble, Qualcomm Technologies, Inc. (versions 1.0, 1.1)
- Maxim Lukyanov, Samsung Electronics (version 1.0)
- Michael Blumenkrantz, Self (version 1.3)
- Michael Lentine, Google (version 1.0)
- Michael O'Hara, AMD (version 1.1)
- Michael Phillip, Samsung Electronics (version 1.2)
- Michael Wong, Codeplay Software Ltd. (version 1.1)
- Michael Worcester, Imagination Technologies (versions 1.0, 1.1)
- Michal Pietrasiuk, Intel (versions 1.0, 1.3)
- Mika Isojarvi, Google (versions 1.0, 1.1)
- Mike Schuchardt, LunarG (versions 1.1, 1.2)
- Mike Stroyan, LunarG (version 1.0)
- Mike Weiblen, LunarG (versions 1.1, 1.2, 1.3)
- Minyoung Son, Samsung Electronics (version 1.0)
- Mitch Singer, AMD (versions 1.0, 1.1, 1.2, 1.3)

- Mythri Venugopal, Samsung Electronics (version 1.0)
- Naveen Leekha, Google (version 1.0)
- Neil Henning, AMD (versions 1.0, 1.1, 1.2, 1.3)
- Neil Hickey, Arm (version 1.2)
- Neil Trevett, NVIDIA (versions 1.0, 1.1, 1.2, 1.3)
- Nick Penwarden, Epic Games (version 1.0)
- Nicolai Hähnle, AMD (version 1.1)
- Niklas Smedberg, Unity Technologies (version 1.0)
- Norbert Nopper, Independent (versions 1.0, 1.1)
- Nuno Subtil, NVIDIA (versions 1.1, 1.2, 1.3)
- Pat Brown, NVIDIA (version 1.0)
- Patrick Cozzi, Independent (version 1.1)
- Patrick Doane, Blizzard Entertainment (version 1.0)
- Peter Lohrmann, AMD (versions 1.0, 1.2)
- Petros Bantolas, Imagination Technologies (version 1.1)
- Philip Rebohle, Valve Software (version 1.3)
- Pierre Boudier, NVIDIA (versions 1.0, 1.1, 1.2, 1.3)
- Pierre-Loup Griffais, Valve Software (versions 1.0, 1.1, 1.2, 1.3)
- Piers Daniell, NVIDIA (versions 1.0, 1.1, 1.2, 1.3)
- Ping Liu, Intel (version 1.3)
- Piotr Bialecki, Intel (version 1.0)
- Piotr Byszewski, Mobica (version 1.3)
- Prabindh Sundareson, Samsung Electronics (version 1.0)
- Pyry Haulos, Google (versions 1.0, 1.1) (Vulkan conformance test subcommittee chair)
- Rachel Bradshaw, Caster Communications (version 1.3)
- Rajeev Rao, Qualcomm (version 1.2)
- Ralph Potter, Samsung Electronics (versions 1.1, 1.2, 1.3)
- Raun Krisch, Samsung Electronics (version 1.3)
- Ray Smith, Arm (versions 1.0, 1.1, 1.2)
- Ricardo Garcia, Igalia (version 1.3)
- Richard Huddy, Samsung Electronics (versions 1.2, 1.3)
- Rob Barris, NVIDIA (version 1.1)
- Rob Stepinski, Transgaming (version 1.0)
- Robert Simpson, Qualcomm Technologies, Inc. (versions 1.0, 1.1, 1.3)
- Rolando Caloca Olivares, Epic Games (versions 1.0, 1.1, 1.2, 1.3)

- Ronan Keryell, Xilinx (version 1.3)
- Roy Ju, Mediatek (version 1.0)
- Rufus Hamade, Imagination Technologies (version 1.0)
- Ruihao Zhang, Qualcomm Technologies, Inc. (versions 1.1, 1.2, 1.3)
- Samuel (Sheng-Wen) Huang, Mediatek (version 1.3)
- Samuel Iglesias Gonsalvez, Igalia (version 1.3)
- Sascha Willems, Self (version 1.3)
- Sean Ellis, Arm (version 1.0)
- Sean Harmer, KDAB Group (versions 1.0, 1.1)
- Shannon Woods, Google (versions 1.0, 1.1, 1.2, 1.3)
- Slawomir Cygan, Intel (versions 1.0, 1.1, 1.3)
- Slawomir Grajewski, Intel (versions 1.0, 1.1, 1.3)
- Sorel Bosan, AMD (version 1.1)
- Spencer Fricke, Samsung Electronics (versions 1.2, 1.3)
- Stefanus Du Toit, Google (version 1.0)
- Stephen Huang, Mediatek (version 1.1)
- Steve Hill, Broadcom (versions 1.0, 1.2)
- Steve Viggers, Core Avionics & Industrial Inc. (versions 1.0, 1.2)
- Steve Winston, Holochip (version 1.3)
- Stuart Smith, AMD (versions 1.0, 1.1, 1.2, 1.3)
- Sujeevan Rajayogam, Google (version 1.3)
- Tilmann Scheller, Samsung Electronics (version 1.1)
- Tim Foley, Intel (version 1.0)
- Tim Lewis, Khronos (version 1.3)
- Timo Suoranta, AMD (version 1.0)
- Timothy Lottes, AMD (versions 1.0, 1.1)
- Tobias Hector, AMD (versions 1.0, 1.1, 1.2, 1.3) (validity language and toolchain)
- Tobin Ehlis, LunarG (version 1.0)
- Tom Olson, Arm (versions 1.0, 1.1, 1.2, 1.3) (Working Group chair)
- Tomasz Bednarz, Independent (version 1.1)
- Tomasz Kubale, Intel (version 1.0)
- Tony Barbour, LunarG (versions 1.0, 1.1, 1.2)
- Tony Zlatinski, NVIDIA (version 1.3)
- Victor Eruhimov, Unknown (version 1.1)
- Vikram Kushwaha, NVIDIA (version 1.3)

- Vincent Hindriksen, Stream HPC (versions 1.2, 1.3)
- Wasim Abbas, Arm (version 1.3)
- Wayne Lister, Imagination Technologies (version 1.0)
- Wolfgang Engel, Unknown (version 1.1)
- Yanjun Zhang, VeriSilicon (versions 1.0, 1.1, 1.2, 1.3)
- Yunxing Zhu, Huawei (version 1.3)

# Other Credits

The Vulkan Advisory Panel members provided important real-world usage information and advice that helped guide design decisions.

The wider Vulkan community have provided useful feedback, questions and specification changes that have helped improve the quality of the Specification via GitHub.

Administrative support to the Working Group for Vulkan 1.1, 1.2, and 1.3 was provided by Khronos staff including Ann Thorsnes, Blaine Kohl, Dominic Agoro-Ombaka, Emily Stearns, Jeff Phillips, Lisie Aartsen, Liz Maitral, Marty Johnson, Tim Lewis, and Xiao-Yu CHENG; and by Alex Crabb, Laura Shubel, and Rachel Bradshaw of Caster Communications.

Administrative support for Vulkan 1.0 was provided by Andrew Riegel, Elizabeth Riegel, Glenn Fredericks, Kathleen Mattson and Michelle Clark of Gold Standard Group.

Technical support was provided by James Riordon, site administration of Khronos.org and OpenGL.org.